OpenDDD.NET

Release v1.0.0-alpha.16

David Runemalm

Oct 03, 2023

USER GUIDE

Welcome to the OpenDDD.NET framework documentation.

This framework is used to implement bounded contexts in software teams with a domain-driven design approach to development.

It's heavily based on hexagonal architecture, so if you're familiar with that pattern you should get started very quickly.

Check out the *user guide* to quickly get started building your own context.

CHAPTER

ONE

EXAMPLES

Here are some examples how your code will look like:

```
/* Program.cs */

using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using OpenDDD.NET.Extensions;
using Main.Extensions;

namespace Main
{
    public class Program
    {
        public static void Main(string[] args)
            => CreateWebHostBuilder(args).Build().Run();

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseKestrel()
                .UseStartup<Startup>()
                .AddEnvFile("ENV_FILE", "CFG_")
                .AddSettings()
                .AddCustomSettings()
                .AddLogging();
    }
}
```

```
/* Startup.cs */

using System.Reflection;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using OpenDDD.Application.Settings;
using OpenDDD.Application.Settings.Persistence;
using OpenDDD.NET.Extensions;
using OpenDDD.NET.Hooks;
using Main.Extensions;
using Main.NET.Hooks;
```

(continues on next page)

(continued from previous page)

```
using Application.Actions;
using Application.Actions.Commands;
using Domain.Model.Forecast;
using Domain.Model.Summary;
using Infrastructure.Ports.Adapters.Domain;
using Infrastructure.Ports.Adapters.Http.v1;
using Infrastructure.Ports.Adapters.Interchange.Translation;
using Infrastructure.Ports.Adapters.Repositories.Memory;
using Infrastructure.Ports.Adapters.Repositories.Migration;
using Infrastructure.Ports.Adapters.Repositories.Postgres;
using HttpCommonTranslation = Infrastructure.Ports.Adapters.Http.Common.Translation;

namespace Main
{
    public class Startup
    {
        private ISettings _settings;

        public Startup(
            ISettings settings)
        {
            _settings = settings;
        }

        public void ConfigureServices(IServiceCollection services)
        {
            // OpenDDD.NET
            services.AddAccessControl(_settings);
            services.AddMonitoring(_settings);
            services.AddPersistence(_settings);
            services.AddPubSub(_settings);
            services.AddTransactional(_settings);

            // App
            AddDomainServices(services);
            AddApplicationService(services);
            AddSecondaryAdapters(services);
            AddPrimaryAdapters(services);
            AddConversion(services);
            AddHooks(services);
        }

        public void Configure(
            IApplicationBuilder app,
            IWebHostEnvironment env,
            IHostApplicationLifetime lifetime)
        {
            // OpenDDD.NET
            app.AddAccessControl(_settings);
            app.AddHttpAdapter(_settings);
            app.AddControl(lifetime);
        }
```

(continues on next page)

(continued from previous page)

```
        // App

        private void AddDomainServices(IServiceCollection services)
        {
            services.AddDomainService<IForecastDomainService, ForecastDomainService>();
        }

        private void AddApplicationService(IServiceCollection services)
        {
            AddActions(services);
        }

        private void AddSecondaryAdapters(IServiceCollection services)
        {
            services.AddEmailAdapter(_settings);
            AddRepositories(services);
        }

        private void AddPrimaryAdapters(IServiceCollection services)
        {
            AddHttpAdapters(services);
            AddInterchangeEventAdapters(services);
            AddDomainEventAdapters(services);
        }

        private void AddHooks(IServiceCollection services)
        {
            services.AddTransient<IOnBeforePrimaryAdaptersStartedHook,␣
↪OnBeforePrimaryAdaptersStartedHook>();
        }

        private void AddConversion(IServiceCollection services)
        {
            services.AddConversion(_settings);
        }

        private void AddActions(IServiceCollection services)
        {
            services.AddAction<GetAverageTemperatureAction, GetAverageTemperatureCommand>
↪();
            services.AddAction<NotifyWeatherPredictedAction,␣
↪NotifyWeatherPredictedCommand>();
            services.AddAction<PredictWeatherAction, PredictWeatherCommand>();
        }

        private void AddHttpAdapters(IServiceCollection services)
        {
            var mvcCoreBuilder = services.AddHttpAdapter(_settings);
            AddHttpAdapterCommon(services);
            AddHttpAdapterV1(services, mvcCoreBuilder);
        }
```

(continues on next page)

(continued from previous page)

```
        private void AddHttpAdapterV1(IServiceCollection services, IMvcCoreBuilder␣
↪mvcCoreBuilder)
        {
            mvcCoreBuilder.AddApplicationPart(Assembly.GetAssembly(typeof(HttpAdapter)));
            services.AddTransient<HttpCommonTranslation.Commands.
↪PredictWeatherCommandTranslator>();
            services.AddTransient<HttpCommonTranslation.ForecastIdTranslator>();
            services.AddTransient<HttpCommonTranslation.ForecastTranslator>();
            services.AddTransient<HttpCommonTranslation.SummaryIdTranslator>();
            services.AddTransient<HttpCommonTranslation.SummaryTranslator>();
        }

        private void AddHttpAdapterCommon(IServiceCollection services)
        {
            services.AddHttpCommandTranslator<HttpCommonTranslation.Commands.
↪PredictWeatherCommandTranslator>();

            services.AddHttpBuildingBlockTranslator<HttpCommonTranslation.
↪ForecastIdTranslator>();
            services.AddHttpBuildingBlockTranslator<HttpCommonTranslation.
↪ForecastTranslator>();
            services.AddHttpBuildingBlockTranslator<HttpCommonTranslation.
↪SummaryIdTranslator>();
            services.AddHttpBuildingBlockTranslator<HttpCommonTranslation.
↪SummaryTranslator>();
        }

        private void AddInterchangeEventAdapters(IServiceCollection services)
        {
            services.AddTransient<IIcForecastTranslator, IcForecastTranslator>();
        }

        private void AddDomainEventAdapters(IServiceCollection services)
        {
            services.AddListener<WeatherPredictedListener>();
        }

        private void AddRepositories(IServiceCollection services)
        {
            if (_settings.Persistence.Provider == PersistenceProvider.Memory)
            {
                services.AddRepository<IForecastRepository, MemoryForecastRepository>();
                services.AddRepository<ISummaryRepository, MemorySummaryRepository>();
            }
            else if (_settings.Persistence.Provider == PersistenceProvider.Postgres)
            {
                services.AddRepository<IForecastRepository, PostgresForecastRepository>
↪();
                services.AddRepository<ISummaryRepository, PostgresSummaryRepository>();
            }
            services.AddMigrator<ForecastMigrator>();
```

(continues on next page)

(continued from previous page)

```
            services.AddMigrator<SummaryMigrator>();
        }
    }
}
```

```
/* CreateAccountAction.cs */

using System.Threading;
using System.Threading.Tasks;
using OpenDDD.Application;
using OpenDDD.Domain.Model.Error;
using OpenDDD.Infrastructure.Ports.PubSub;
using Application.Actions.Commands;
using Domain.Model.User;

namespace Application.Actions
{
    public class CreateAccountAction : Action<CreateAccountCommand, User>
    {
        private readonly IDomainPublisher _domainPublisher;
        private readonly IUserRepository _userRepository;

        public CreateAccountAction(
            IDomainPublisher domainPublisher,
            IUserRepository userRepository,
            ITransactionalDependencies transactionalDependencies)
            : base(transactionalDependencies)
        {
            _domainPublisher = domainPublisher;
            _userRepository = userRepository;
        }

        public override async Task<User> ExecuteAsync(
            CreateAccountCommand command,
            ActionId actionId,
            CancellationToken ct)
        {
            // Validate
            var existing =
                await _userRepository.GetWithEmailAsync(
                    command.Email,
                    actionId,
                    ct);

            if (existing != null)
                throw DomainException.AlreadyExists("user", "email", command.Email);

            // Run
            var user =
                await User.CreateAccountAsync(
                    userId: UserId.Create(await _userRepository.GetNextIdentityAsync()),
                    firstName: command.FirstName,
```

(continues on next page)

(continued from previous page)

```
                    lastName: command.LastName,
                    email: command.Email,
                    password: command.Password,
                    passwordAgain: command.RepeatPassword,
                    domainPublisher: _domainPublisher,
                    actionId: actionId,
                    ct: ct);

            // Persist
            await _userRepository.SaveAsync(user, actionId, ct);

            // Return
            return user;
        }
    }
}
```

```
/* User.cs */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.AspNetCore.WebUtilities;
using OpenDDD.Application;
using OpenDDD.Domain.Model.BuildingBlocks.Aggregate;
using OpenDDD.Domain.Model.BuildingBlocks.Entity;
using OpenDDD.Domain.Model.Error;
using OpenDDD.Domain.Model.Validation;
using OpenDDD.Infrastructure.Ports.Email;
using OpenDDD.Infrastructure.Ports.PubSub;
using Domain.Model.Realm;
using ContextDomainModelVersion = Domain.Model.DomainModelVersion;
using SaltClass = Domain.Model.User.Salt;

namespace Domain.Model.User
{
    public class User : Aggregate, IAggregate, IEquatable<User>
    {
        public UserId UserId { get; set; }
        EntityId IAggregate.Id => UserId;
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public Email Email { get; set; }
        public DateTime? EmailVerifiedAt { get; set; }
        public DateTime? EmailVerificationRequestedAt { get; set; }
        public DateTime? EmailVerificationCodeCreatedAt { get; set; }
        public EmailVerificationCode? EmailVerificationCode { get; set; }
        public Password Password { get; set; }
        public Salt Salt { get; set; }
        public string ResetPasswordCode { get; set; }
```

(continues on next page)

(continued from previous page)

```
        public DateTime? ResetPasswordCodeCreatedAt { get; set; }
        public bool IsSuperUser { get; set; }
        public ICollection<RealmId> RealmIds { get; set; }

        public User() {}

        // Public

        public static async Task<User> CreateAccountAsync(
            UserId userId,
            string firstName,
            string lastName,
            Email email,
            string password,
            string passwordAgain,
            IDomainPublisher domainPublisher,
            ActionId actionId,
            CancellationToken ct)
        {
            if (password != passwordAgain)
                throw DomainException.InvariantViolation("The passwords don't match.");

            var user =
                new User
                {
                    DomainModelVersion = ContextDomainModelVersion.Latest(),
                    UserId = userId,
                    FirstName = firstName,
                    LastName = lastName,
                    Email = email,
                    EmailVerifiedAt = null,
                    EmailVerificationRequestedAt = null,
                    EmailVerificationCodeCreatedAt = null,
                    EmailVerificationCode = null,
                    IsSuperUser = false,
                    RealmIds = new List<RealmId>()
                };

            user.SetPassword(password, actionId, ct);
            user.RequestEmailValidation(actionId, ct);

            user.Validate();

            await domainPublisher.PublishAsync(new AccountCreated(user, actionId));

            return user;
        }

        public static User CreateDefaultAccountAtIdpLogin(
            UserId userId,
            string firstName,
            string lastName,
```

(continues on next page)

(continued from previous page)

```
            Email email,
            ActionId actionId,
            CancellationToken ct)
        {
            var user =
                new User
                {
                    DomainModelVersion = ContextDomainModelVersion.Latest(),
                    UserId = userId,
                    FirstName = firstName,
                    LastName = lastName,
                    Email = email,
                    EmailVerifiedAt = null,
                    EmailVerificationRequestedAt = null,
                    EmailVerificationCodeCreatedAt = null,
                    EmailVerificationCode = null,
                    IsSuperUser = false,
                    RealmIds = new List<RealmId>()
                };

            user.SetPassword(Password.Generate(), actionId, ct);

            user.Validate();

            return user;
        }

        public static User CreateRootAccountAtBoot(
            UserId userId,
            string firstName,
            string lastName,
            Email email,
            string password,
            ActionId actionId,
            CancellationToken ct)
        {
            var user =
                new User
                {
                    DomainModelVersion = ContextDomainModelVersion.Latest(),
                    UserId = userId,
                    FirstName = firstName,
                    LastName = lastName,
                    Email = email,
                    EmailVerifiedAt = null,
                    EmailVerificationRequestedAt = null,
                    EmailVerificationCodeCreatedAt = null,
                    EmailVerificationCode = null,
                    IsSuperUser = true,
                    RealmIds = new List<RealmId>()
                };
```

(continues on next page)

(continued from previous page)

```
            user.SetPassword(password, actionId, ct);

            user.Validate();

            return user;
        }

        public bool IsEmailVerified()
            => EmailVerifiedAt != null;

        public bool IsEmailVerificationRequested()
            => EmailVerificationRequestedAt != null;

        public bool IsEmailVerificationCodeExpired()
            => DateTime.UtcNow.Subtract(EmailVerificationCodeCreatedAt!.Value).
↪TotalSeconds >= (60 * 30);

        public async Task SendEmailVerificationEmailAsync(Uri verifyEmailUrl, IEmailPort␣
↪emailAdapter, ActionId actionId, CancellationToken ct)
        {
            if (Email == null)
                throw DomainException.InvariantViolation("The user has no email.");

            if (IsEmailVerified())
                throw DomainException.InvariantViolation("The email is already verified.
↪");

            if (!IsEmailVerificationRequested())
                throw DomainException.InvariantViolation("Email verification hasn't been␣
↪requested.");

            // Re-generate code
            if (EmailVerificationCode != null)
                RegenerateEmailVerificationCode();

            var link = $"{verifyEmailUrl}?code={EmailVerificationCode}&userId={UserId}";

            await emailAdapter.SendAsync(
                "no-reply@poweriam.com",
                "PowerIAM",
                Email.Value,
                $"{FirstName} {LastName}",
                $"Verify your email",
                $"Hi, please verify this email address belongs to you by clicking the␣
↪link: <a href=\"{link}\">Verify Your Email</a>",
                true,
                ct);
        }

        public async Task VerifyEmail(EmailVerificationCode code, ActionId actionId,␣
↪CancellationToken ct)
        {
```

(continues on next page)

(continued from previous page)

```
            if (Email == null)
                throw VerifyEmailException.UserHasNoEmail();

            if (IsEmailVerified())
                throw VerifyEmailException.AlreadyVerified();

            if (!IsEmailVerificationRequested())
                throw VerifyEmailException.NotRequested();

            if (!code.Equals(EmailVerificationCode))
                throw VerifyEmailException.InvalidCode();

            if (IsEmailVerificationCodeExpired())
                throw VerifyEmailException.CodeExpired();

            EmailVerifiedAt = DateTime.UtcNow;
            EmailVerificationRequestedAt = null;
            EmailVerificationCode = null;
            EmailVerificationCodeCreatedAt = null;
        }

        public void AddToRealm(RealmId realmId, ActionId actionId)
        {
            if (IsInRealm(realmId))
                throw DomainException.InvariantViolation($"User {UserId} already belongs␣
↪to realm {realmId}.");

            RealmIds.Add(realmId);
        }

        public async Task ForgetPasswordAsync(Uri resetPasswordUri, IEmailPort␣
↪emailAdapter, ActionId actionId, CancellationToken ct)
        {
            if (Email == null)
                throw DomainException.InvariantViolation("Can't send reset password␣
↪email, the user has no email.");

            ResetPasswordCode = Guid.NewGuid().ToString("n").Substring(0, 24);
            ResetPasswordCodeCreatedAt = DateTime.UtcNow;

            resetPasswordUri = new Uri(QueryHelpers.AddQueryString(resetPasswordUri.
↪ToString(), "code", ResetPasswordCode));

            var link = resetPasswordUri.ToString();

            await emailAdapter.SendAsync(
                "no-reply@poweriam.com",
                "PowerIAM",
                Email.Value,
                $"{FirstName} {LastName}",
                $"Your reset password link",
                $"Hi, someone said you forgot your password. If this wasn't you then␣
```

(continues on next page)

(continued from previous page)

```
↪ignore this email.<br>" +
                $"Follow the link to set your new password: <a href=\"{link}\">Reset␣
↪Your Password</a>",
                true,
                ct);
        }

        public bool IsInRealm(RealmId realmId)
            => RealmIds.Contains(realmId);

        public bool IsValidPassword(string password)
            => Salt != null && Password != null && (Password.CreateAndHash(password,␣
↪Salt) == Password);

        public void RemoveFromRealm(RealmId realmId, ActionId actionId)
        {
            if (!IsInRealm(realmId))
                throw DomainException.InvariantViolation($"User {UserId} doesn't belong␣
↪to realm {realmId}.");

            RealmIds.Remove(realmId);
        }

        public async Task ResetPassword(string newPassword, ActionId actionId,␣
↪CancellationToken ct)
        {
            if (ResetPasswordCode == null)
                throw DomainException.InvariantViolation(
                    "Can't reset password, there's no reset password code.");

            if (DateTime.UtcNow.Subtract(ResetPasswordCodeCreatedAt.Value).TotalMinutes >
↪ 59)
                throw DomainException.InvariantViolation(
                    "The reset password link has expired. Please generate a new one and␣
↪try again.");

            SetPassword(newPassword, actionId, ct);

            ResetPasswordCode = null;
            ResetPasswordCodeCreatedAt = null;
        }

        public void SetPassword(string password, ActionId actionId, CancellationToken ct)
        {
            Salt = SaltClass.Generate();
            Password = Password.CreateAndHash(password, Salt);
        }

        public void RequestEmailValidation(ActionId actionId, CancellationToken ct)
        {
            EmailVerifiedAt = null;
            EmailVerificationRequestedAt = DateTime.UtcNow;
```

(continues on next page)

(continued from previous page)

```
            RegenerateEmailVerificationCode();
        }

        // Private

        private void RegenerateEmailVerificationCode()
        {
            EmailVerificationCode = EmailVerificationCode.Generate();
            EmailVerificationCodeCreatedAt = DateTime.UtcNow;
        }

        protected void Validate()
        {
            var validator = new Validator<User>(this);

            var errors = validator
                .NotNull(bb => bb.UserId.Value)
                .NotNullOrEmpty(bb => bb.FirstName)
                .NotNullOrEmpty(bb => bb.LastName)
                .NotNullOrEmpty(bb => bb.Email.Value)
                .Errors()
                .ToList();

            if (errors.Any())
            {
                throw DomainException.InvariantViolation(
                    $"User is invalid with errors: " +
                    $"{string.Join(", ", errors.Select(e => $"{e.Key} {e.Details}"))}");
            }
        }

        // Equality

        public bool Equals(User? other)
        {
            if (ReferenceEquals(null, other)) return false;
            if (ReferenceEquals(this, other)) return true;
            return base.Equals(other) && UserId.Equals(other.UserId) && FirstName ==␣
↪other.FirstName && LastName == other.LastName && Email.Equals(other.Email) && Nullable.
↪Equals(EmailVerifiedAt, other.EmailVerifiedAt) && Nullable.
↪Equals(EmailVerificationRequestedAt, other.EmailVerificationRequestedAt) && Nullable.
↪Equals(EmailVerificationCodeCreatedAt, other.EmailVerificationCodeCreatedAt) &&␣
↪Equals(EmailVerificationCode, other.EmailVerificationCode) && Password.Equals(other.
↪Password) && Salt.Equals(other.Salt) && ResetPasswordCode == other.ResetPasswordCode &&
↪ Nullable.Equals(ResetPasswordCodeCreatedAt, other.ResetPasswordCodeCreatedAt) &&␣
↪IsSuperUser == other.IsSuperUser && RealmIds.Equals(other.RealmIds);
        }

        public override bool Equals(object? obj)
        {
            if (ReferenceEquals(null, obj)) return false;
            if (ReferenceEquals(this, obj)) return true;
```

(continues on next page)

(continued from previous page)

```
            if (obj.GetType() != this.GetType()) return false;
            return Equals((User)obj);
        }

        public override int GetHashCode()
        {
            var hashCode = new HashCode();
            hashCode.Add(base.GetHashCode());
            hashCode.Add(UserId);
            hashCode.Add(FirstName);
            hashCode.Add(LastName);
            hashCode.Add(Email);
            hashCode.Add(EmailVerifiedAt);
            hashCode.Add(EmailVerificationRequestedAt);
            hashCode.Add(EmailVerificationCodeCreatedAt);
            hashCode.Add(EmailVerificationCode);
            hashCode.Add(Password);
            hashCode.Add(Salt);
            hashCode.Add(ResetPasswordCode);
            hashCode.Add(ResetPasswordCodeCreatedAt);
            hashCode.Add(IsSuperUser);
            hashCode.Add(RealmIds);
            return hashCode.ToHashCode();
        }
    }
}
```

```
# Configuration file

# Logging
CFG_LOGGING_LEVEL_DOTNET=Information
CFG_LOGGING_LEVEL=Debug

# General
CFG_GENERAL_CONTEXT=Weather

# Auth
CFG_AUTH_ENABLED=false
CFG_AUTH_RBAC_PROVIDER=
CFG_AUTH_RBAC_EXTERNAL_REALM_ID=
CFG_AUTH_JWT_TOKEN_PRIVATE_KEY=
CFG_AUTH_JWT_TOKEN_NAME=
CFG_AUTH_JWT_TOKEN_LOCATION=
CFG_AUTH_JWT_TOKEN_SCHEME=

# Http Adapter
CFG_HTTP_URLS=http://localhost:5051
CFG_HTTP_CORS_ALLOWED_ORIGINS=http://localhost:5051
CFG_HTTP_DOCS_MAJOR_VERSIONS=1
CFG_HTTP_DOCS_DEFINITIONS=
CFG_HTTP_DOCS_ENABLED=true
CFG_HTTP_DOCS_HTTP_ENABLED=true
```

(continues on next page)

(continued from previous page)

```
CFG_HTTP_DOCS_HTTPS_ENABLED=false
CFG_HTTP_DOCS_HOSTNAME=localhost:5051
CFG_HTTP_DOCS_HTTP_PORT=80
CFG_HTTP_DOCS_HTTPS_PORT=443
CFG_HTTP_DOCS_AUTH_EXTRA_TOKENS=
CFG_HTTP_DOCS_TITLE=Weather API

# Persistence
CFG_PERSISTENCE_PROVIDER=Memory
CFG_PERSISTENCE_POOLING_ENABLED=true
CFG_PERSISTENCE_POOLING_MIN_SIZE=0
CFG_PERSISTENCE_POOLING_MAX_SIZE=100

# Postgres
CFG_POSTGRES_CONN_STR=

# PubSub
CFG_PUBSUB_PROVIDER=Memory
CFG_PUBSUB_MAX_DELIVERY_RETRIES=3
CFG_PUBSUB_PUBLISHER_ENABLED=true

# Monitoring
CFG_MONITORING_PROVIDER=Memory

# Rabbit
CFG_RABBIT_HOST=
CFG_RABBIT_PORT=
CFG_RABBIT_USERNAME=
CFG_RABBIT_PASSWORD=

# Email
CFG_EMAIL_ENABLED=true
CFG_EMAIL_PROVIDER=memory
CFG_EMAIL_SMTP_HOST=
CFG_EMAIL_SMTP_PORT=
CFG_EMAIL_SMTP_USERNAME=
CFG_EMAIL_SMTP_PASSWORD=
```

1.1 Installation

You could install the package as a nuget into an existing project, using the dotnet CLI:

```
$ dotnet add package OpenDDD.NET
```

1.2 Create a project

The easiest way to get started with your bounded context is using the `project template`.

By using the template, you will get the boilerplate code for free, which makes sure you create all configuration files and bootup code correctly.

Get started by installing the project templates package:

```
$ dotnet new install OpenDDD.NET-Templates
```

Then create your bounded context project:

```
$ dotnet new openddd-net -n MyBoundedContext
```

Your project will be created in a folder with the name `MyBoundedContext`.

Note: Replace *MyBoundedContext* with the actual name of your project.

1.3 Example Application

There is some example code on the *start page*.

Use the *project template* to quickly create a project with boiler plate code you can look at.

1.4 Basic Concepts

We will now give you a walkthrough of the framework's `basic concepts`:

- *Env files*
- *Domain Model Version*
- *Program.cs*
- *Startup.cs*
- *Commands*
- *Actions*
- *Entities*
- *Repositories*
- *Events*
- *Listeners*
- *Domain Services*
- *Errors*
- *Converters*
- *Migrators*
- *Unit Tests*

1.4.1 Env files

An `env file` is used to configure your bounded context for a specific environment.

It's part of the Twelve-Factor App pattern.

You will have one env file for each of your environments:

- env.prod
- env.staging
- env.local
- env.test

Load your configuration at boot time using the `ENV_FILE` environment variable. Set the value to the env file's filename. This way it will be read and loaded at boot.

Some hosting environments don't support files accessible to the deployed code package. In this case, you can put the (serialized json content of) the env file directly in the `ENV_FILE` variable.

Tip: In each of the directories that you need to create an env file there is a `*.sample` file that you can duplicate, rename and edit accordingly.

Note: The example env file below has memory adapters and authentication disabled. This helps you get started quickly. However, it also makes it not suitable for production environments.

Example env file:

```
# Logging
CFG_LOGGING_LEVEL_DOTNET=Information
CFG_LOGGING_LEVEL=Debug

# General
CFG_GENERAL_CONTEXT=Weather

# Auth
CFG_AUTH_ENABLED=false
CFG_AUTH_RBAC_PROVIDER=
CFG_AUTH_RBAC_EXTERNAL_REALM_ID=
CFG_AUTH_JWT_TOKEN_PRIVATE_KEY=
CFG_AUTH_JWT_TOKEN_NAME=
CFG_AUTH_JWT_TOKEN_LOCATION=
CFG_AUTH_JWT_TOKEN_SCHEME=

# Http Adapter
CFG_HTTP_URLS=http://localhost:5051
CFG_HTTP_CORS_ALLOWED_ORIGINS=http://localhost:5051
CFG_HTTP_DOCS_MAJOR_VERSIONS=1
CFG_HTTP_DOCS_DEFINITIONS=
CFG_HTTP_DOCS_ENABLED=true
CFG_HTTP_DOCS_HTTP_ENABLED=true
CFG_HTTP_DOCS_HTTPS_ENABLED=false
CFG_HTTP_DOCS_HOSTNAME=localhost:5051
```

(continues on next page)

(continued from previous page)

```
CFG_HTTP_DOCS_HTTP_PORT=80
CFG_HTTP_DOCS_HTTPS_PORT=443
CFG_HTTP_DOCS_AUTH_EXTRA_TOKENS=
CFG_HTTP_DOCS_TITLE=Weather API

# Persistence
CFG_PERSISTENCE_PROVIDER=Memory
CFG_PERSISTENCE_POOLING_ENABLED=true
CFG_PERSISTENCE_POOLING_MIN_SIZE=0
CFG_PERSISTENCE_POOLING_MAX_SIZE=100

# Postgres
CFG_POSTGRES_CONN_STR=

# PubSub
CFG_PUBSUB_PROVIDER=Memory
CFG_PUBSUB_MAX_DELIVERY_RETRIES=3
CFG_PUBSUB_PUBLISHER_ENABLED=true

# Monitoring
CFG_MONITORING_PROVIDER=Memory

# Rabbit
CFG_RABBIT_HOST=
CFG_RABBIT_PORT=
CFG_RABBIT_USERNAME=
CFG_RABBIT_PASSWORD=

# Email
CFG_EMAIL_ENABLED=true
CFG_EMAIL_PROVIDER=memory
CFG_EMAIL_SMTP_HOST=
CFG_EMAIL_SMTP_PORT=
CFG_EMAIL_SMTP_USERNAME=
CFG_EMAIL_SMTP_PASSWORD=
```

1.4.2 Domain Model Version

Since this framework is all about focusing on an evolving and up-to-date domain model, we need to have a representation of a domain model version.

Create this class by subclassing the `DomainModelVersion` base class.

As your model evolves, you will increment the `LatestString` and add appropriate migration methods to the entity migrators. More on *migrators* in a later section.

Example domain model version:

```
namespace Domain.Model
{
    public class DomainModelVersion : DDD.Domain.Model.DomainModelVersion
    {
```

(continues on next page)

(continued from previous page)

```
        public const string LatestString = "1.0.0";

        public DomainModelVersion(string dotString) : base(dotString) { }

        public static DomainModelVersion Latest()
        {
            return new DomainModelVersion(LatestString);
        }
    }
}
```

1.4.3 Program.cs

Use the `AddXxx()` extension methods of the framework to properly configure the .NET host and application.

Tip: Use the weather forecast *project template* and you won't need to create this file.

Example Program.cs file:

```
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using OpenDDD.NET.Extensions;
using Main.Extensions;

namespace Main
{
    public class Program
    {
        public static void Main(string[] args)
            => CreateWebHostBuilder(args).Build().Run();

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseKestrel()
                .UseStartup<Startup>()
                .AddEnvFile("ENV_FILE", "CFG_")
                .AddSettings()
                .AddCustomSettings()
                .AddLogging();
    }
}
```

1.4.4 Startup.cs

Since part of the design philosophy behind this framwork is to follow the hexagonal architecture, and to make this intent clear through the structure of the code, the `Startup.cs` file is written according to a specific convention.

See the example below and create your Startup.cs file.

Tip: Use the weather forecast *project template* and you won't need to create this file.

Example Startup.cs file:

```
using System.Reflection;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using OpenDDD.Application.Settings;
using OpenDDD.Application.Settings.Persistence;
using OpenDDD.NET.Extensions;
using OpenDDD.NET.Hooks;
using Main.Extensions;
using Main.NET.Hooks;
using Application.Actions;
using Application.Actions.Commands;
using Domain.Model.Forecast;
using Domain.Model.Summary;
using Infrastructure.Ports.Adapters.Domain;
using Infrastructure.Ports.Adapters.Http.v1;
using Infrastructure.Ports.Adapters.Interchange.Translation;
using Infrastructure.Ports.Adapters.Repositories.Memory;
using Infrastructure.Ports.Adapters.Repositories.Migration;
using Infrastructure.Ports.Adapters.Repositories.Postgres;
using HttpCommonTranslation = Infrastructure.Ports.Adapters.Http.Common.Translation;

namespace Main
{
    public class Startup
    {
        private ISettings _settings;

        public Startup(
            ISettings settings)
        {
            _settings = settings;
        }

        public void ConfigureServices(IServiceCollection services)
        {
            // OpenDDD.NET
            services.AddAccessControl(_settings);
            services.AddMonitoring(_settings);
            services.AddPersistence(_settings);
            services.AddPubSub(_settings);
```

(continues on next page)

(continued from previous page)

```
            services.AddTransactional(_settings);

            // App
            AddDomainServices(services);
            AddApplicationService(services);
            AddSecondaryAdapters(services);
            AddPrimaryAdapters(services);
            AddConversion(services);
            AddHooks(services);
        }

        public void Configure(
            IApplicationBuilder app,
            IWebHostEnvironment env,
            IHostApplicationLifetime lifetime)
        {
            // OpenDDD.NET
            app.AddAccessControl(_settings);
            app.AddHttpAdapter(_settings);
            app.AddControl(lifetime);
        }

        // App

        private void AddDomainServices(IServiceCollection services)
        {
            services.AddDomainService<IForecastDomainService, ForecastDomainService>();
        }

        private void AddApplicationService(IServiceCollection services)
        {
            AddActions(services);
        }

        private void AddSecondaryAdapters(IServiceCollection services)
        {
            services.AddEmailAdapter(_settings);
            AddRepositories(services);
        }

        private void AddPrimaryAdapters(IServiceCollection services)
        {
            AddHttpAdapters(services);
            AddInterchangeEventAdapters(services);
            AddDomainEventAdapters(services);
        }

        private void AddHooks(IServiceCollection services)
        {
            services.AddTransient<IOnBeforePrimaryAdaptersStartedHook,␣
↪OnBeforePrimaryAdaptersStartedHook>();
        }
```

(continues on next page)

(continued from previous page)

```
        private void AddConversion(IServiceCollection services)
        {
            services.AddConversion(_settings);
        }

        private void AddActions(IServiceCollection services)
        {
            services.AddAction<GetAverageTemperatureAction, GetAverageTemperatureCommand>
↪();
            services.AddAction<NotifyWeatherPredictedAction,␣
↪NotifyWeatherPredictedCommand>();
            services.AddAction<PredictWeatherAction, PredictWeatherCommand>();
        }

        private void AddHttpAdapters(IServiceCollection services)
        {
            var mvcCoreBuilder = services.AddHttpAdapter(_settings);
            AddHttpAdapterCommon(services);
            AddHttpAdapterV1(services, mvcCoreBuilder);
        }

        private void AddHttpAdapterV1(IServiceCollection services, IMvcCoreBuilder␣
↪mvcCoreBuilder)
        {
            mvcCoreBuilder.AddApplicationPart(Assembly.GetAssembly(typeof(HttpAdapter)));
            services.AddTransient<HttpCommonTranslation.Commands.
↪PredictWeatherCommandTranslator>();
            services.AddTransient<HttpCommonTranslation.ForecastIdTranslator>();
            services.AddTransient<HttpCommonTranslation.ForecastTranslator>();
            services.AddTransient<HttpCommonTranslation.SummaryIdTranslator>();
            services.AddTransient<HttpCommonTranslation.SummaryTranslator>();
        }

        private void AddHttpAdapterCommon(IServiceCollection services)
        {
            services.AddHttpCommandTranslator<HttpCommonTranslation.Commands.
↪PredictWeatherCommandTranslator>();

            services.AddHttpBuildingBlockTranslator<HttpCommonTranslation.
↪ForecastIdTranslator>();
            services.AddHttpBuildingBlockTranslator<HttpCommonTranslation.
↪ForecastTranslator>();
            services.AddHttpBuildingBlockTranslator<HttpCommonTranslation.
↪SummaryIdTranslator>();
            services.AddHttpBuildingBlockTranslator<HttpCommonTranslation.
↪SummaryTranslator>();
        }

        private void AddInterchangeEventAdapters(IServiceCollection services)
        {
            services.AddTransient<IIcForecastTranslator, IcForecastTranslator>();
```

(continues on next page)

(continued from previous page)

```
        }

        private void AddDomainEventAdapters(IServiceCollection services)
        {
            services.AddListener<WeatherPredictedListener>();
        }

        private void AddRepositories(IServiceCollection services)
        {
            if (_settings.Persistence.Provider == PersistenceProvider.Memory)
            {
                services.AddRepository<IForecastRepository, MemoryForecastRepository>();
                services.AddRepository<ISummaryRepository, MemorySummaryRepository>();
            }
            else if (_settings.Persistence.Provider == PersistenceProvider.Postgres)
            {
                services.AddRepository<IForecastRepository, PostgresForecastRepository>
↪();
                services.AddRepository<ISummaryRepository, PostgresSummaryRepository>();
            }
            services.AddMigrator<ForecastMigrator>();
            services.AddMigrator<SummaryMigrator>();
        }
    }
}
```

1.4.5 Commands

All command classes need to subclass the `Command` class.

The command class is basically a data transfer object (DTO), except of course it has a very specific meaning in terms of your domain model.

The command is passed to the relevant action when an actor requests it.

Example command:

```
using System.Collections.Generic;
using System.Linq;
using DDD.Application;
using DDD.Application.Error;
using DDD.Domain.Model.Validation;
using Domain.Model.User;

namespace Application.Actions.Commands
{
    public class CreateAccountCommand : Command
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public Email Email { get; set; }
        public string Password { get; set; }
```

(continues on next page)

(continued from previous page)

```
        public string RepeatPassword { get; set; }

        public override void Validate()
        {
            var errors = GetErrors();

            if (errors.Any())
                throw new InvalidCommandException(this, errors);
        }

        public override IEnumerable<ValidationError> GetErrors()
        {
            var errors = new Validator<CreateAccountCommand>(this)
                .NotNullOrEmpty(command => command.FirstName)
                .NotNullOrEmpty(command => command.LastName)
                .Email(command => command.Email.ToString())
                .NotNullOrEmpty(command => command.Password.ToString())
                .NotNullOrEmpty(command => command.RepeatPassword.ToString())
                .Errors();

            return errors;
        }
    }
}
```

1.4.6 Actions

All action classes need to subclass the `Action<TCommand, TReturn>` class.

The `ExecuteAsync()` method is where you fetch your aggregates and delegate domain logic to them and/or domain services.

If your aggregates or domain services need to publish events or use any adapter, you inject them via the constructor and pass along in the calls that drive your domain logic through these objects.

Remember that an aggregate is only allowed to change the state of a single aggregate at a time. It must also delegate all domain logic to the aggregates and/or domain services. Domain logic doesn't belong in the application layer.

You register your action classes with the DI container like this:

```
services.AddAction<CreateAccountAction, CreateAccountCommand>();
```

> **Warning:** Delegate all domain logic to aggregates or domain services.

> **Warning:** Only act upon one aggregate per action.

Example action:

```
using System.Threading;
using System.Threading.Tasks;
using OpenDDD.Application;
using OpenDDD.Domain.Model.Error;
using OpenDDD.Infrastructure.Ports.PubSub;
using Application.Actions.Commands;
using Domain.Model.User;

namespace Application.Actions
{
    public class CreateAccountAction : Action<CreateAccountCommand, User>
    {
        private readonly IDomainPublisher _domainPublisher;
        private readonly IUserRepository _userRepository;

        public CreateAccountAction(
            IDomainPublisher domainPublisher,
            IUserRepository userRepository,
            ITransactionalDependencies transactionalDependencies)
            : base(transactionalDependencies)
        {
            _domainPublisher = domainPublisher;
            _userRepository = userRepository;
        }

        public override async Task<User> ExecuteAsync(
            CreateAccountCommand command,
            ActionId actionId,
            CancellationToken ct)
        {
            // Validate
            var existing =
                await _userRepository.GetWithEmailAsync(
                    command.Email,
                    actionId,
                    ct);

            if (existing != null)
                throw DomainException.AlreadyExists("user", "email", command.Email);

            // Run
            var user =
                await User.CreateAccountAsync(
                    userId: UserId.Create(await _userRepository.GetNextIdentityAsync()),
                    firstName: command.FirstName,
                    lastName: command.LastName,
                    email: command.Email,
                    password: command.Password,
                    passwordAgain: command.RepeatPassword,
                    domainPublisher: _domainPublisher,
                    actionId: actionId,
                    ct: ct);
```

(continues on next page)

(continued from previous page)

```
            // Persist
            await _userRepository.SaveAsync(user, actionId, ct);

            // Return
            return user;
        }
    }
}
```

1.4.7 Entities

The entities subclass either the `Aggregate` class if it's an aggregate, or the `Entity` class otherwise.

They need to implement the `IEquatable<>` interface, so that assertions in the unit tests can compare them to each other.

Actions use the methods of aggregate roots to drive the domain logic, passing adapters and publishers needed as arguments.

Example aggregate:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.AspNetCore.WebUtilities;
using OpenDDD.Application;
using OpenDDD.Domain.Model.BuildingBlocks.Aggregate;
using OpenDDD.Domain.Model.BuildingBlocks.Entity;
using OpenDDD.Domain.Model.Error;
using OpenDDD.Domain.Model.Validation;
using OpenDDD.Infrastructure.Ports.Email;
using OpenDDD.Infrastructure.Ports.PubSub;
using Domain.Model.Realm;
using ContextDomainModelVersion = Domain.Model.DomainModelVersion;
using SaltClass = Domain.Model.User.Salt;

namespace Domain.Model.User
{
    public class User : Aggregate, IAggregate, IEquatable<User>
    {
        public UserId UserId { get; set; }
        EntityId IAggregate.Id => UserId;
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public Email Email { get; set; }
        public DateTime? EmailVerifiedAt { get; set; }
        public DateTime? EmailVerificationRequestedAt { get; set; }
        public DateTime? EmailVerificationCodeCreatedAt { get; set; }
        public EmailVerificationCode? EmailVerificationCode { get; set; }
        public Password Password { get; set; }
```

(continues on next page)

(continued from previous page)

```
        public Salt Salt { get; set; }
        public string ResetPasswordCode { get; set; }
        public DateTime? ResetPasswordCodeCreatedAt { get; set; }
        public bool IsSuperUser { get; set; }
        public ICollection<RealmId> RealmIds { get; set; }

        public User() {}

        // Public

        public static async Task<User> CreateAccountAsync(
            UserId userId,
            string firstName,
            string lastName,
            Email email,
            string password,
            string passwordAgain,
            IDomainPublisher domainPublisher,
            ActionId actionId,
            CancellationToken ct)
        {
            if (password != passwordAgain)
                throw DomainException.InvariantViolation("The passwords don't match.");

            var user =
                new User
                {
                    DomainModelVersion = ContextDomainModelVersion.Latest(),
                    UserId = userId,
                    FirstName = firstName,
                    LastName = lastName,
                    Email = email,
                    EmailVerifiedAt = null,
                    EmailVerificationRequestedAt = null,
                    EmailVerificationCodeCreatedAt = null,
                    EmailVerificationCode = null,
                    IsSuperUser = false,
                    RealmIds = new List<RealmId>()
                };

            user.SetPassword(password, actionId, ct);
            user.RequestEmailValidation(actionId, ct);

            user.Validate();

            await domainPublisher.PublishAsync(new AccountCreated(user, actionId));

            return user;
        }

        public static User CreateDefaultAccountAtIdpLogin(
            UserId userId,
```

(continues on next page)

(continued from previous page)

```
            string firstName,
            string lastName,
            Email email,
            ActionId actionId,
            CancellationToken ct)
        {
            var user =
                new User
                {
                    DomainModelVersion = ContextDomainModelVersion.Latest(),
                    UserId = userId,
                    FirstName = firstName,
                    LastName = lastName,
                    Email = email,
                    EmailVerifiedAt = null,
                    EmailVerificationRequestedAt = null,
                    EmailVerificationCodeCreatedAt = null,
                    EmailVerificationCode = null,
                    IsSuperUser = false,
                    RealmIds = new List<RealmId>()
                };

            user.SetPassword(Password.Generate(), actionId, ct);

            user.Validate();

            return user;
        }

        public static User CreateRootAccountAtBoot(
            UserId userId,
            string firstName,
            string lastName,
            Email email,
            string password,
            ActionId actionId,
            CancellationToken ct)
        {
            var user =
                new User
                {
                    DomainModelVersion = ContextDomainModelVersion.Latest(),
                    UserId = userId,
                    FirstName = firstName,
                    LastName = lastName,
                    Email = email,
                    EmailVerifiedAt = null,
                    EmailVerificationRequestedAt = null,
                    EmailVerificationCodeCreatedAt = null,
                    EmailVerificationCode = null,
                    IsSuperUser = true,
                    RealmIds = new List<RealmId>()
```

(continues on next page)

(continued from previous page)

```
                };

            user.SetPassword(password, actionId, ct);

            user.Validate();

            return user;
        }

        public bool IsEmailVerified()
            => EmailVerifiedAt != null;

        public bool IsEmailVerificationRequested()
            => EmailVerificationRequestedAt != null;

        public bool IsEmailVerificationCodeExpired()
            => DateTime.UtcNow.Subtract(EmailVerificationCodeCreatedAt!.Value).
↪TotalSeconds >= (60 * 30);

        public async Task SendEmailVerificationEmailAsync(Uri verifyEmailUrl, IEmailPort␣
↪emailAdapter, ActionId actionId, CancellationToken ct)
        {
            if (Email == null)
                throw DomainException.InvariantViolation("The user has no email.");

            if (IsEmailVerified())
                throw DomainException.InvariantViolation("The email is already verified.
↪");

            if (!IsEmailVerificationRequested())
                throw DomainException.InvariantViolation("Email verification hasn't been␣
↪requested.");

            // Re-generate code
            if (EmailVerificationCode != null)
                RegenerateEmailVerificationCode();

            var link = $"{verifyEmailUrl}?code={EmailVerificationCode}&userId={UserId}";

            await emailAdapter.SendAsync(
                "no-reply@poweriam.com",
                "PowerIAM",
                Email.Value,
                $"{FirstName} {LastName}",
                $"Verify your email",
                $"Hi, please verify this email address belongs to you by clicking the␣
↪link: <a href=\"{link}\">Verify Your Email</a>",
                true,
                ct);
        }

        public async Task VerifyEmail(EmailVerificationCode code, ActionId actionId,␣
```

(continues on next page)

(continued from previous page)

```
↪CancellationToken ct)
        {
            if (Email == null)
                throw VerifyEmailException.UserHasNoEmail();

            if (IsEmailVerified())
                throw VerifyEmailException.AlreadyVerified();

            if (!IsEmailVerificationRequested())
                throw VerifyEmailException.NotRequested();

            if (!code.Equals(EmailVerificationCode))
                throw VerifyEmailException.InvalidCode();

            if (IsEmailVerificationCodeExpired())
                throw VerifyEmailException.CodeExpired();

            EmailVerifiedAt = DateTime.UtcNow;
            EmailVerificationRequestedAt = null;
            EmailVerificationCode = null;
            EmailVerificationCodeCreatedAt = null;
        }

        public void AddToRealm(RealmId realmId, ActionId actionId)
        {
            if (IsInRealm(realmId))
                throw DomainException.InvariantViolation($"User {UserId} already belongs␣
↪to realm {realmId}.");

            RealmIds.Add(realmId);
        }

        public async Task ForgetPasswordAsync(Uri resetPasswordUri, IEmailPort␣
↪emailAdapter, ActionId actionId, CancellationToken ct)
        {
            if (Email == null)
                throw DomainException.InvariantViolation("Can't send reset password␣
↪email, the user has no email.");

            ResetPasswordCode = Guid.NewGuid().ToString("n").Substring(0, 24);
            ResetPasswordCodeCreatedAt = DateTime.UtcNow;

            resetPasswordUri = new Uri(QueryHelpers.AddQueryString(resetPasswordUri.
↪ToString(), "code", ResetPasswordCode));

            var link = resetPasswordUri.ToString();

            await emailAdapter.SendAsync(
                "no-reply@poweriam.com",
                "PowerIAM",
                Email.Value,
                $"{FirstName} {LastName}",
```

(continues on next page)

(continued from previous page)

```
                $"Your reset password link",
                $"Hi, someone said you forgot your password. If this wasn't you then␣
↪ignore this email.<br>" +
                $"Follow the link to set your new password: <a href=\"{link}\">Reset␣
↪Your Password</a>",
                true,
                ct);
        }

        public bool IsInRealm(RealmId realmId)
            => RealmIds.Contains(realmId);

        public bool IsValidPassword(string password)
            => Salt != null && Password != null && (Password.CreateAndHash(password,␣
↪Salt) == Password);

        public void RemoveFromRealm(RealmId realmId, ActionId actionId)
        {
            if (!IsInRealm(realmId))
                throw DomainException.InvariantViolation($"User {UserId} doesn't belong␣
↪to realm {realmId}.");

            RealmIds.Remove(realmId);
        }

        public async Task ResetPassword(string newPassword, ActionId actionId,␣
↪CancellationToken ct)
        {
            if (ResetPasswordCode == null)
                throw DomainException.InvariantViolation(
                    "Can't reset password, there's no reset password code.");

            if (DateTime.UtcNow.Subtract(ResetPasswordCodeCreatedAt.Value).TotalMinutes >
↪ 59)
                throw DomainException.InvariantViolation(
                    "The reset password link has expired. Please generate a new one and␣
↪try again.");

            SetPassword(newPassword, actionId, ct);

            ResetPasswordCode = null;
            ResetPasswordCodeCreatedAt = null;
        }

        public void SetPassword(string password, ActionId actionId, CancellationToken ct)
        {
            Salt = SaltClass.Generate();
            Password = Password.CreateAndHash(password, Salt);
        }

        public void RequestEmailValidation(ActionId actionId, CancellationToken ct)
        {
```

(continues on next page)

(continued from previous page)

```
            EmailVerifiedAt = null;
            EmailVerificationRequestedAt = DateTime.UtcNow;
            RegenerateEmailVerificationCode();
        }

        // Private

        private void RegenerateEmailVerificationCode()
        {
            EmailVerificationCode = EmailVerificationCode.Generate();
            EmailVerificationCodeCreatedAt = DateTime.UtcNow;
        }

        protected void Validate()
        {
            var validator = new Validator<User>(this);

            var errors = validator
                .NotNull(bb => bb.UserId.Value)
                .NotNullOrEmpty(bb => bb.FirstName)
                .NotNullOrEmpty(bb => bb.LastName)
                .NotNullOrEmpty(bb => bb.Email.Value)
                .Errors()
                .ToList();

            if (errors.Any())
            {
                throw DomainException.InvariantViolation(
                    $"User is invalid with errors: " +
                    $"{string.Join(", ", errors.Select(e => $"{e.Key} {e.Details}"))}");
            }
        }

        // Equality

        public bool Equals(User? other)
        {
            if (ReferenceEquals(null, other)) return false;
            if (ReferenceEquals(this, other)) return true;
            return base.Equals(other) && UserId.Equals(other.UserId) && FirstName ==␣
↪other.FirstName && LastName == other.LastName && Email.Equals(other.Email) && Nullable.
↪Equals(EmailVerifiedAt, other.EmailVerifiedAt) && Nullable.
↪Equals(EmailVerificationRequestedAt, other.EmailVerificationRequestedAt) && Nullable.
↪Equals(EmailVerificationCodeCreatedAt, other.EmailVerificationCodeCreatedAt) &&␣
↪Equals(EmailVerificationCode, other.EmailVerificationCode) && Password.Equals(other.
↪Password) && Salt.Equals(other.Salt) && ResetPasswordCode == other.ResetPasswordCode &&
↪ Nullable.Equals(ResetPasswordCodeCreatedAt, other.ResetPasswordCodeCreatedAt) &&␣
↪IsSuperUser == other.IsSuperUser && RealmIds.Equals(other.RealmIds);
        }

        public override bool Equals(object? obj)
        {
```

(continues on next page)

(continued from previous page)

```
            if (ReferenceEquals(null, obj)) return false;
            if (ReferenceEquals(this, obj)) return true;
            if (obj.GetType() != this.GetType()) return false;
            return Equals((User)obj);
        }

        public override int GetHashCode()
        {
            var hashCode = new HashCode();
            hashCode.Add(base.GetHashCode());
            hashCode.Add(UserId);
            hashCode.Add(FirstName);
            hashCode.Add(LastName);
            hashCode.Add(Email);
            hashCode.Add(EmailVerifiedAt);
            hashCode.Add(EmailVerificationRequestedAt);
            hashCode.Add(EmailVerificationCodeCreatedAt);
            hashCode.Add(EmailVerificationCode);
            hashCode.Add(Password);
            hashCode.Add(Salt);
            hashCode.Add(ResetPasswordCode);
            hashCode.Add(ResetPasswordCodeCreatedAt);
            hashCode.Add(IsSuperUser);
            hashCode.Add(RealmIds);
            return hashCode.ToHashCode();
        }
    }
}
```

1.4.8 Repositories

A repository is the interface for getting & saving your aggregates from/to the database.

Subclass the `Repository` base class for each aggregate.

There are some base methods for e.g. getting all aggregates, getting by ID, saving an aggregate, etc. You will need to add methods for the queries that are specific to your aggregate and domain model.

You will create one interface per repository, and one adapter for each of the technology implementations you want to support.

E.g. for a user repository, you might need to create the following classes:

- IUserRepository
- MemoryUserRepository
- PostgresUserRepository

Example repository:

```
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using OpenDDD.Application;
```

(continues on next page)

(continued from previous page)

```
using OpenDDD.Application.Settings;
using OpenDDD.Infrastructure.Ports.Adapters.Common.Translation.Converters;
using OpenDDD.Infrastructure.Ports.Adapters.Repository.Postgres;
using OpenDDD.Infrastructure.Services.Persistence;
using Domain.Model.Realm;
using Domain.Model.User;
using Infrastructure.Ports.Adapters.Repository.Migration;

namespace Infrastructure.Ports.Adapters.Repository.Postgres
{
    public class PostgresUserRepository : PostgresRepository<User, UserId>,␣
↪IUserRepository
    {
        public PostgresUserRepository(ISettings settings, UserMigrator migrator,␣
↪IPersistenceService persistenceService, ConversionSettings conversionSettings)
            : base(settings, "users", migrator, persistenceService, conversionSettings)
        {

        }

        public Task<IEnumerable<User>> GetInRealmAsync(RealmId realmId, ActionId␣
↪actionId, CancellationToken ct)
            => GetWithAsync(user => user.RealmIds.Contains(realmId), actionId, ct);

        public Task<User?> GetWithEmailAsync(Email email, ActionId actionId,␣
↪CancellationToken ct)
            => GetFirstOrDefaultWithAsync(new List<(string, object)>() { ("Email",␣
↪email) }, actionId, ct);

        public Task<User?> GetWithEmailVerificationCodeAsync(EmailVerificationCode code,␣
↪ActionId actionId, CancellationToken ct)
            => GetFirstOrDefaultWithAsync(u => u.EmailVerificationCode != null && u.
↪EmailVerificationCode.Equals(code), actionId, ct);

        public Task<User?> GetWithResetPasswordCodeAsync(string code, ActionId actionId,␣
↪CancellationToken ct)
            => GetFirstOrDefaultWithAsync(u => u.ResetPasswordCode == code, actionId,␣
↪ct);
    }
}
```

1.4.9 Events

There are two classes for implementing events, `DomainEvent` and `IntegrationEvent`.

Subclass the appropriate one depending on the type of event you're implementing.

Note: Integration event names are prefixed with `Ic` to easily separate them from domain events.

Example domain event:

```
using System;
using OpenDDD.Application;
using OpenDDD.Domain.Model.BuildingBlocks.Event;

namespace Domain.Model.User
{
    public class AccountCreated : DomainEvent, IEquatable<AccountCreated>
    {
        public UserId UserId { get; set; }
        public Email Email { get; set; }

        public AccountCreated() : base("AccountCreated", DomainModelVersion.Latest(),
↪"IAM", ActionId.Create()) { }

        public AccountCreated(User user, ActionId actionId)
            : base("AccountCreated", DomainModelVersion.Latest(), "IAM", actionId)
        {
            UserId = user.UserId;
            Email = user.Email;
        }

        // Equality

        public bool Equals(AccountCreated? other)
        {
            if (ReferenceEquals(null, other)) return false;
            if (ReferenceEquals(this, other)) return true;
            return base.Equals(other) && UserId.Equals(other.UserId) && Email.
↪Equals(other.Email);
        }

        public override bool Equals(object? obj)
        {
            if (ReferenceEquals(null, obj)) return false;
            if (ReferenceEquals(this, obj)) return true;
            if (obj.GetType() != this.GetType()) return false;
            return Equals((AccountCreated)obj);
        }

        public override int GetHashCode()
        {
            return HashCode.Combine(base.GetHashCode(), UserId, Email);
        }
    }
}
```

Example integration event:

```
using System;
using OpenDDD.Application;
using OpenDDD.Domain.Model.BuildingBlocks.Event;
using ContextDomainModelVersion = Interchange.Domain.Model.DomainModelVersion;
```

(continues on next page)

(continued from previous page)

```
namespace Interchange.Domain.Model.Forecast
{
    public class IcWeatherPredicted : IntegrationEvent, IEquatable<IcWeatherPredicted>
    {
        public string ForecastId { get; set; }
        public DateTime Date { get; set; }
        public int TemperatureC { get; set; }
        public string SummaryId { get; set; }

        public IcWeatherPredicted() { }

        public IcWeatherPredicted(ActionId actionId) : base("WeatherPredicted",␣
↪ContextDomainModelVersion.Latest(), "Weather", actionId) { }

        public IcWeatherPredicted(IcForecast forecast, ActionId actionId)
            : base("WeatherPredicted", ContextDomainModelVersion.Latest(), "Interchange",
↪ actionId)
        {
            ForecastId = forecast.ForecastId;
            Date = forecast.Date;
            TemperatureC = forecast.TemperatureC;
            SummaryId = forecast.SummaryId;
        }

        // Equality

        public bool Equals(IcWeatherPredicted other)
        {
            if (ReferenceEquals(null, other)) return false;
            if (ReferenceEquals(this, other)) return true;
            return base.Equals(other) && ForecastId == other.ForecastId && Date.
↪Equals(other.Date) && TemperatureC == other.TemperatureC && SummaryId == other.
↪SummaryId;
        }

        public override bool Equals(object obj)
        {
            if (ReferenceEquals(null, obj)) return false;
            if (ReferenceEquals(this, obj)) return true;
            if (obj.GetType() != this.GetType()) return false;
            return Equals((IcWeatherPredicted)obj);
        }

        public override int GetHashCode()
        {
            return HashCode.Combine(base.GetHashCode(), ForecastId, Date, TemperatureC,␣
↪SummaryId);
        }
    }
}
```

1.4.10 Listeners

A listener is used to react to domain- and integration events.

Your listeners will basically just create a command and pass it to the action that will be run to perform the reaction necessary.

In the example below you can see how the `AccountCreated` event is reacted to by calling the `SendEmailVerification` action.

Subscribe to an event by registering the listener with the DI container:

```
services.AddListener<AccountCreatedListener>();
```

Example domain event listener:

```
using Application.Actions;
using Application.Actions.Commands;
using OpenDDD.Application;
using OpenDDD.Infrastructure.Ports.Adapters.Common.Translation.Converters;
using OpenDDD.Infrastructure.Ports.PubSub;
using OpenDDD.Logging;
using Domain.Model.User;
using ContextDomainModelVersion = Domain.Model.DomainModelVersion;

namespace Infrastructure.Ports.Adapters.Domain
{
    public class AccountCreatedListener
        : EventListener<AccountCreated, SendEmailVerificationEmailAction,␣
↪SendEmailVerificationEmailCommand>
    {
        public AccountCreatedListener(
            SendEmailVerificationEmailAction action,
            IDomainEventAdapter eventAdapter,
            IOutbox outbox,
            IDeadLetterQueue deadLetterQueue,
            ILogger logger,
            ConversionSettings conversionSettings)
            : base(
                Context.Domain,
                "AccountCreated",
                ContextDomainModelVersion.Latest(),
                action,
                eventAdapter,
                outbox,
                deadLetterQueue,
                logger,
                conversionSettings)
        {

        }

        public override SendEmailVerificationEmailCommand CreateCommand(AccountCreated␣
↪theEvent)
        {
```

(continues on next page)

(continued from previous page)

```
            var command =
                new SendEmailVerificationEmailCommand
                {
                    UserId = theEvent.UserId
                };

            return command;
        }
    }
}
```

1.4.11 Domain Services

All domain service classes need to subclass the `DomainService` class.

You register your domain services with the DI container like this:

```
services.AddDomainService<IRoleDomainService, RoleDomainService>();
```

Example domain service:

```
using System.Threading;
using System.Threading.Tasks;
using OpenDDD.Application;
using OpenDDD.Domain.Model.Error;
using OpenDDD.Domain.Services;
using Domain.Model.Assignment;
using Domain.Model.Permission;
using Domain.Model.Realm;

namespace Domain.Model.Role
{
    public class RoleDomainService : DomainService, IRoleDomainService
    {
        private readonly IAssignmentDomainService _assignmentDomainService;
        private readonly IPermissionRepository _permissionRepository;
        private readonly IRealmRepository _realmRepository;
        private readonly IRoleRepository _roleRepository;

        public RoleDomainService(
            IAssignmentDomainService assignmentDomainService,
            IPermissionRepository permissionRepository,
            IRealmRepository realmRepository,
            IRoleRepository roleRepository)
        {
            _assignmentDomainService = assignmentDomainService;
            _permissionRepository = permissionRepository;
            _realmRepository = realmRepository;
            _roleRepository = roleRepository;
        }

        public async Task<Role> AddPermissionToRoleAsync(
```

(continues on next page)

(continued from previous page)

```
            RoleId roleId, PermissionId permissionId, ActionId actionId,␣
↪CancellationToken ct)
        {
            var role = await _roleRepository.GetAsync(roleId, actionId, ct);
            var permission = await _permissionRepository.GetAsync(permissionId, actionId,
↪ ct);

            if (role == null)
                throw DomainException.NotFound("role", roleId.ToString());

            if (permission == null)
                throw DomainException.NotFound("permission", permissionId.ToString());

            // Authorize
            if (role.IsInWorld())
            {
                await _assignmentDomainService.AssurePermissionsInWorldAsync(
                    permissions: new[] { ("IAM", "ADD_PERMISSION_TO_ROLE") },
                    actionId: actionId,
                    ct: ct);
            }
            else
            {
                await _assignmentDomainService.AssurePermissionsInRealmAsync(
                    realmId: role.RealmId.ToString(),
                    externalRealmId: "",
                    permissions: new[] { ("IAM", "ADD_PERMISSION_TO_ROLE") },
                    actionId: actionId,
                    ct: ct);
            }

            if (role.IsInWorld() && !permission.IsInWorld())
                throw DomainException.InvariantViolation(
                    "Role is in world but the permission is in a realm.");

            if (role.IsInRealm() && !(permission.IsInRealm(role.RealmId) || permission.
↪IsInWorld()))
                throw DomainException.InvariantViolation(
                    "Role is in a realm but the permission is neither in that realm nor␣
↪the world.");

            role.AddPermission(permissionId, actionId);

            return role;
        }

        public async Task<Role> CreateRoleInWorldAsync(string name, string description,␣
↪ActionId actionId, CancellationToken ct)
        {
            // Authorize
            await _assignmentDomainService.AssurePermissionsInWorldAsync(
                new[] { ("IAM", "CREATE_ROLE") },
```

(continues on next page)

(continued from previous page)

```
                actionId,
                ct);

            // Run
            var existing = await _roleRepository.GetWithNameInWorldAsync(name, actionId,␣
↪ct);

            if (existing != null)
                throw DomainException.AlreadyExists("role", "name", name);

            var role = await Role.CreateInWorldAsync(
                RoleId.Create(await _roleRepository.GetNextIdentityAsync()),
                null,
                name,
                description,
                actionId);

            // Return
            return role;
        }

        public async Task<Role> CreateRoleInRealmAsync(string name, string description,␣
↪RealmId realmId, string externalRealmId, ActionId actionId, CancellationToken ct)
        {
            // Validate
            if (!(realmId != null ^ externalRealmId != null))
                throw DomainException.InvariantViolation(
                    "You must supply exactly one of realmId and externalRealmId.");

            var isExternalRealmId = realmId == null;

            // Authorize
            await _assignmentDomainService.AssurePermissionsInRealmAsync(
                realmId?.ToString(),
                externalRealmId,
                new[] { ("IAM", "CREATE_ROLE") },
                actionId,
                ct);

            // Run
            Realm.Realm realm;

            if (isExternalRealmId)
                realm = await _realmRepository.GetWithExternalIdAsync(externalRealmId,␣
↪actionId, ct);
            else
                realm = await _realmRepository.GetAsync(realmId, actionId, ct);

            if (realm == null)
                throw DomainException.NotFound("realm", (isExternalRealmId ? null :␣
↪realmId).ToString());
```

(continues on next page)

(continued from previous page)

```
            // Exists?
            var existing = await _roleRepository.GetWithNameInRealmAsync(name, realm.
↪RealmId, actionId, ct);

            if (existing != null)
                throw DomainException.AlreadyExists("role", "name", name);

            var role = await Role.CreateInRealmAsync(
                RoleId.Create(await _roleRepository.GetNextIdentityAsync()),
                realmId,
                null,
                name,
                description,
                actionId);

            // Return
            return role;
        }
    }
}
```

1.4.12 Errors

When an error occurs in your domain model, you manifest it by *throwing an exception* containing the `DomainError`.

The `DomainError` is of the following model:

- Code
- Message
- User Message

The `Code` is simply an identifier for the error.

The `Message` should contain a message with a description useful and aimed towards understanding the error by an integrating developer.

The `User Message` should contain a message with a description useful and aimed towards understanding the error in a frontend by an end user.

Tip: It's recommeded that the frontend development team utilizes the `Code` to craft the most helpful and precise user message, instead of simply relying on the more generic `User Message`.

Note: The generic domain errors are to be found in the `DomainError` base class of the framework.

Example domain error:

```
using OpenDDD.Domain.Model.Error;

namespace Domain.Model.Error
{
```

(continues on next page)

(continued from previous page)

```
    public class DomainError : OpenDDD.Domain.Model.Error.DomainError
    {
        // Codes

        private const int VerifyEmail_NotRequested_Code = 1001;
        private const string VerifyEmail_NotRequested_Msg = "Email verification hasn't␣
↪been requested.";
        private const string VerifyEmail_NotRequested_UsrMsg = "No verification of your␣
↪email has been requested.";

        private const int VerifyEmail_AlreadyVerified_Code = 1002;
        private const string VerifyEmail_AlreadyVerified_Msg = "The email has already␣
↪been verified.";
        private const string VerifyEmail_AlreadyVerified_UsrMsg = "You email address has␣
↪already been verified.";

        private const int VerifyEmail_NoCode_Code = 1003;
        private const string VerifyEmail_NoCode_Msg = "The user has no email␣
↪verification code.";
        private const string VerifyEmail_NoCode_UsrMsg = "An unknown error has occured.␣
↪You can't verify your email because there's no email verification code.";

        private const int VerifyEmail_InvalidCode_Code = 1004;
        private const string VerifyEmail_InvalidCode_Msg = "The code is invalid.";
        private const string VerifyEmail_InvalidCode_UsrMsg = "The email verification␣
↪code you provided is invalid. Please request a new verification code and try again.";

        private const int VerifyEmail_CodeExpired_Code = 1005;
        private const string VerifyEmail_CodeExpired_Msg = "The code has expired.";
        private const string VerifyEmail_CodeExpired_UsrMsg = "The verification code you␣
↪provided has expired. Please request a new verification code.";

        private const int VerifyEmail_NoUserWithCode_Code = 1006;
        private const string VerifyEmail_NoUserWithCode_Msg = "There's no user with that␣
↪code.";
        private const string VerifyEmail_NoUserWithCode_UsrMsg = "We couldn't find a␣
↪user with that email verification code. Please make sure you entered the correct code␣
↪and try again. Alternatively request a new verification code.";

        private const int VerifyEmail_UserHasNoEmail_Code = 1007;
        private const string VerifyEmail_UserHasNoEmail_Msg = "The user has no email.";
        private const string VerifyEmail_UserHasNoEmail_UsrMsg = "We couldn't verify␣
↪your email because you haven't provided one. Please provide one and try verification␣
↪again.";

        public static IDomainError VerifyEmail_NotRequested() => Create(VerifyEmail_
↪NotRequested_Code, VerifyEmail_NotRequested_Msg, VerifyEmail_NotRequested_UsrMsg);
        public static IDomainError VerifyEmail_AlreadyVerified() => Create(VerifyEmail_
↪AlreadyVerified_Code, VerifyEmail_AlreadyVerified_Msg, VerifyEmail_AlreadyVerified_
↪UsrMsg);
        public static IDomainError VerifyEmail_NoCode() => Create(VerifyEmail_NoCode_
↪Code, VerifyEmail_NoCode_Msg, VerifyEmail_NoCode_UsrMsg);
```

(continues on next page)

(continued from previous page)

```
        public static IDomainError VerifyEmail_InvalidCode() => Create(VerifyEmail_
↪InvalidCode_Code, VerifyEmail_InvalidCode_Msg, VerifyEmail_InvalidCode_UsrMsg);
        public static IDomainError VerifyEmail_CodeExpired() => Create(VerifyEmail_
↪CodeExpired_Code, VerifyEmail_CodeExpired_Msg, VerifyEmail_CodeExpired_UsrMsg);
        public static IDomainError VerifyEmail_NoUserWithCode() => Create(VerifyEmail_
↪NoUserWithCode_Code, VerifyEmail_NoUserWithCode_Msg, VerifyEmail_NoUserWithCode_
↪UsrMsg);
        public static IDomainError VerifyEmail_UserHasNoEmail() => Create(VerifyEmail_
↪UserHasNoEmail_Code, VerifyEmail_UserHasNoEmail_Msg, VerifyEmail_UserHasNoEmail_
↪UsrMsg);
    }
}
```

1.4.13 Exceptions

The error(s) are manifested by throwing an `DomainException`, containing the error(s).

There are two types of exceptions:

- Highly precise `Custom exceptions` that are specific to your domain model and
- `Generic exceptions` that are part of the framework and can be used by any bounded context.

It's up to you to decided which would be best to use in each of your cases.

In the example below, the `VerifyEmailException.AlreadyVerified()` exception is used, but it could also have been implemented using the generic `DomainException.InvariantViolation("Email is already verified.")` exception, (with a custom message sent as argument).

Example exception:

```
using OpenDDD.Domain.Model.Error;
using DomainError = Domain.Model.Error.DomainError;

namespace Domain.Model.User
{
    public class VerifyEmailException : DomainException
    {
        public static VerifyEmailException NotRequested()
            => new VerifyEmailException(DomainError.VerifyEmail_NotRequested());

        public static VerifyEmailException AlreadyVerified()
            => new VerifyEmailException(DomainError.VerifyEmail_AlreadyVerified());

        public static VerifyEmailException NoCode()
            => new VerifyEmailException(DomainError.VerifyEmail_NoCode());

        public static VerifyEmailException InvalidCode()
            => new VerifyEmailException(DomainError.VerifyEmail_InvalidCode());

        public static VerifyEmailException CodeExpired()
            => new VerifyEmailException(DomainError.VerifyEmail_CodeExpired());

        public static VerifyEmailException UserHasNoEmail()
```

(continues on next page)

(continued from previous page)

```
            => new VerifyEmailException(DomainError.VerifyEmail_UserHasNoEmail());

        public static VerifyEmailException NoUserWithCode()
            => new VerifyEmailException(DomainError.VerifyEmail_NoUserWithCode());

        public VerifyEmailException(IDomainError error) : base(error)
        {

        }
    }
}
```

Example of throwing exceptions:

```
public async Task VerifyEmail(EmailVerificationCode code, ActionId actionId,␣
↪CancellationToken ct)
{
    if (Email == null)
        throw VerifyEmailException.UserHasNoEmail();

    if (IsEmailVerified())
        throw VerifyEmailException.AlreadyVerified();

    if (!IsEmailVerificationRequested())
        throw VerifyEmailException.NotRequested();

    if (!code.Equals(EmailVerificationCode))
        throw VerifyEmailException.InvalidCode();

    if (IsEmailVerificationCodeExpired())
        throw VerifyEmailException.CodeExpired();

    EmailVerifiedAt = DateTime.UtcNow;
    EmailVerificationRequestedAt = null;
    EmailVerificationCode = null;
    EmailVerificationCodeCreatedAt = null;
}
```

1.4.14 Converters

Converters are used to serialize and deserialize your aggregates and events into strings and back, so that they can be persisted and/or sent on a message bus.

The OpenDDD.NET framework bases conversion on the Json.NET framework by Newtonsoft.

Json.NET comes with converters for many non-primitive generic types, such as e.g. DateTime and classes themselves. OpenDDD.NET provides missing converters for DDD-generic types such as EntityId and DomainModelVersion.

However, for all the entities and value objects that are `unique` to your domain model, you need to create a `corresponding converter`.

You create a converter by subclassing the `Converter<T>` base class.

Tip: Utilize the `ReadJsonUsingMethod()` method of the OpenDDD framework base class to conveniently deserialize strings using your entity- and value object classes static factory methods.

Note: Don't mistake the Converter<T> class for the class with the same name in the Json.NET framework.

Example converter:

```
using System;
using Newtonsoft.Json;
using OpenDDD.Infrastructure.Ports.Adapters.Common.Translation.Converters;
using Domain.Model.User;

namespace Infrastructure.Ports.Adapters.Common.Translation.Converters
{
    public class EmailConverter : Converter<Email>
    {
        public override void WriteJson(
            JsonWriter writer,
            object? value,
            JsonSerializer serializer)
        {
            writer.WriteValue(value.ToString());
        }

        public override object ReadJson(
            JsonReader reader,
            Type objectType,
            object? existingValue,
            JsonSerializer serializer)
        {
            if (reader.Value == null)
                return null;
            return ReadJsonUsingMethod(reader, "Create", objectType);
        }
    }
}
```

Registering your converter dependencies is a three-step process:

1. Create the ConversionSettings class, (if you haven't already).
2. Add the converter to the `Converters` collection in the constructor.
3. Register your ConversionSettings class with the DI container.

Example conversion settings:

```
using DddConversionSettings = OpenDDD.Infrastructure.Ports.Adapters.Common.Translation.
↪Converters.ConversionSettings;

namespace Infrastructure.Ports.Adapters.Common.Translation.Converters
{
    public class ConversionSettings : DddConversionSettings
```

(continues on next page)

(continued from previous page)

```
    {
        public ConversionSettings()
        {
            Converters.Add(new EmailConverter());
            Converters.Add(new EmailVerificationCodeConverter());
            Converters.Add(new PasswordConverter());
            Converters.Add(new SaltConverter());
        }
    }
}
```

You register your serializer settings with the DI container like this:

```
services.AddTransient<OpenDddConversionSettings, ConversionSettings>();
```

Note: The `AddConversion()` call in Startup.cs of the *project template* does almost all of this work for you. You just need to create your converters and add them to the collection in the constructor.

1.4.15 Migrators

Whenver you bump your domain model version, you need to create a migration for all the entities that have changed.

Subclass the `Migrator` base class and implement the `FromVX_X_X()` method for all your entities affected by the change.

Domain model versioning is a first-class citizen in this DDD framework. Thus, migration should be as easy as possible so that the domain model can be evolved continuously with minimal effort.

You register your migrator classes with the DI container like this:

```
services.AddMigrator<UserMigrator>();
```

Note: Entities will migrate on-the-fly next time they are fetched and saved by the repositories.

Note: If an entity has not changed it's model from one version to another, simply don't add a method for that version to the migrator class.

Example migrator:

```
using System.Collections.Generic;
using OpenDDD.Infrastructure.Ports.Adapters.Repository;
using Domain.Model.Realm;
using Domain.Model.User;
using ContextDomainModelVersion = Domain.Model.DomainModelVersion;

namespace Infrastructure.Ports.Adapters.Repository.Migration
{
    public class UserMigrator : Migrator<User>
```

(continues on next page)

(continued from previous page)

```
    {
        public UserMigrator() : base(ContextDomainModelVersion.Latest())
        {

        }

        public User FromV1_0_2(User userV1_0_2)
        {
            var salt = Salt.Generate();
            var password = Password.GenerateAndHash(salt);

            userV1_0_2.Salt = salt;
            userV1_0_2.Password = password;
            userV1_0_2.ResetPasswordCode = null;
            userV1_0_2.ResetPasswordCodeCreatedAt = null;
            userV1_0_2.DomainModelVersion = new ContextDomainModelVersion("1.0.3");
            return userV1_0_2;
        }

        /* There's no changes in model for v1.0.2. */

        public User FromV1_0_0(User userV1_0_0)
        {
            userV1_0_0.RealmIds = new List<RealmId>();
            userV1_0_0.IsSuperUser = false;
            userV1_0_0.DomainModelVersion = new ContextDomainModelVersion("1.0.1");
            return userV1_0_0;
        }
    }
}
```

1.4.16 Unit Tests

To achieve full test coverage of your bounded context, you need to implement a full suite of unit tests for each of your domain model actions.

Subclass `ActionUnitTests` for each of your action unit test suites. Then add your test methods to cover all paths.

The test methods are based on the standard `xUnit` testing model, so you will be familiar with the `Arrange`, `Act` and `Assert` sections.

Note: You need to create your own action unit tests base class. See the *section below* on how to do this.

> **Warning:** Remember that the unit tests need to reflect the domain model and ubiquitous language.

Example action unit tests:

```
using Xunit;
using Application.Actions.Commands;
```

(continues on next page)

(continued from previous page)

```
using Domain.Model.User;

namespace Tests.Actions;

public class VerifyEmailTests : ActionUnitTests
{
    public VerifyEmailTests()
    {
        Configure();
        EmptyDb();
    }

    [Fact]
    public async Task TestSuccess_EmailVerified()
    {
        // Arrange
        await EnsureRootUserAsync();
        await EnsureIamDomainAsync();
        await EnsureIamPermissionsAsync();

        await CreateAccount(email: "test.testsson@poweriam.com");

        // Act
        var command = new VerifyEmailCommand { Code = User.EmailVerificationCode };
        await VerifyEmailAction.ExecuteAsync(command, ActionId, CancellationToken.None);

        await Refresh(User);

        // Assert
        AssertTrue(User.IsEmailVerified());
        AssertNow(User.EmailVerifiedAt);
    }

    [Fact]
    public async Task TestFail_UserHasNoEmail()
    {
        // Arrange
        await EnsureRootUserAsync();
        await EnsureIamDomainAsync();
        await EnsureIamPermissionsAsync();

        await CreateAccount(email: "test.testsson@poweriam.com");

        // ..hack
        await Refresh(User);
        User.Email = null;
        await UserRepository.SaveAsync(User, ActionId, CancellationToken.None);

        // Act & Assert
        var command = new VerifyEmailCommand()
        {
            Code = User.EmailVerificationCode
```

(continues on next page)

(continued from previous page)

```
        };

        await AssertFailure(VerifyEmailException.UserHasNoEmail(), VerifyEmailAction.
↪ExecuteAsync(command, ActionId, CancellationToken.None));
    }

    [Fact]
    public async Task TestFail_AlreadyVerified()
    {
        // Arrange
        await EnsureRootUserAsync();
        await EnsureIamDomainAsync();
        await EnsureIamPermissionsAsync();

        await CreateAccount(email: "test.testsson@poweriam.com");

        var command = new VerifyEmailCommand()
        {
            Code = User.EmailVerificationCode
        };

        await VerifyEmailAction.ExecuteAsync(command, ActionId, CancellationToken.None);

        // ..hack
        await Refresh(User);
        User.EmailVerificationCode = command.Code;
        await UserRepository.SaveAsync(User, ActionId, CancellationToken.None);

        // Act & Assert
        await AssertFailure(VerifyEmailException.AlreadyVerified(), VerifyEmailAction.
↪ExecuteAsync(command, ActionId, CancellationToken.None));
    }

    [Fact]
    public async Task TestFail_NotRequested()
    {
        // Arrange
        await EnsureRootUserAsync();
        await EnsureIamDomainAsync();
        await EnsureIamPermissionsAsync();

        await CreateAccount(email: "test.testsson@poweriam.com");

        // ..hack
        await Refresh(User);
        User.EmailVerificationRequestedAt = null;
        await UserRepository.SaveAsync(User, ActionId, CancellationToken.None);

        // Act & Assert
        var command = new VerifyEmailCommand()
        {
            Code = User.EmailVerificationCode
```

(continues on next page)

(continued from previous page)

```
        };

        await AssertFailure(VerifyEmailException.NotRequested(), VerifyEmailAction.
↪ExecuteAsync(command, ActionId, CancellationToken.None));
    }

    [Theory]
    [InlineData(null)]
    [InlineData("some-invalid-code")]
    public async Task TestFail_InvalidCode(string? code)
    {
        // Arrange
        await EnsureRootUserAsync();
        await EnsureIamDomainAsync();
        await EnsureIamPermissionsAsync();

        await CreateAccount(email: "test.testsson@poweriam.com");

        // Act & Assert
        var command = new VerifyEmailCommand()
        {
            Code = EmailVerificationCode.Create(code)
        };

        await AssertFailure(VerifyEmailException.InvalidCode(), VerifyEmailAction.
↪ExecuteAsync(command, ActionId, CancellationToken.None));
    }

    [Fact]
    public async Task TestFail_ExpiredCode()
    {
        // Arrange
        await EnsureRootUserAsync();
        await EnsureIamDomainAsync();
        await EnsureIamPermissionsAsync();

        await CreateAccount(email: "test.testsson@poweriam.com");

        User.EmailVerificationCodeCreatedAt = DateTime.MinValue;
        await UserRepository.SaveAsync(User, ActionId, CancellationToken.None);

        // Act & Assert
        var command = new VerifyEmailCommand()
        {
            Code = User.EmailVerificationCode
        };

        await AssertFailure(VerifyEmailException.CodeExpired(), VerifyEmailAction.
↪ExecuteAsync(command, ActionId, CancellationToken.None));
    }
}
```

1.4.17 The ActionUnitTests class

The purpose of your `ActionUnitTests` class is to provide a set of convenience methods and properties for your action unit tests to use.

The design philosophy of this framework states that the unit tests should be easy to read, understand and maintain. Furthermore they need to reflect and express the domain model in a clear manner.

To achive all of the above, your subclass will contain the following:

- Action excecution methods.
- State properties.
- `CreateWebHostBuilder()` (used to setup the TestServer).
- `EmptyAggregateRepositories()` (used to empty your repositories before each test)
- Dependency properties.
- Assertion methods.

Subclass `ActionUnitTests` to create your own base class for the unit tests.

Note: This is a very concise description of the relatively big `ActionUnitTests` concept. Later we'll add more documentation and guides on the topic of testing but for now you should be able to look at the example code and get started with your action testing.

Example action unit tests class:

```
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Xunit;
using OpenDDD.NET.Extensions;
using OpenDDD.Domain.Model.Auth;
using OpenDDD.Domain.Services.Auth;
using OpenDDD.NET.Hooks;
using Main;
using Main.Extensions;
using Main.NET.Hooks;
using Application.Actions;
using Application.Actions.Commands;
using Application.Settings;
using Domain.Model.Assignment;
using Domain.Model.Domain;
using Domain.Model.Permission;
using Domain.Model.Realm;
using Domain.Model.Role;
using Domain.Model.User;
using DddActionUnitTests = OpenDDD.Tests.ActionUnitTests;

namespace Tests
{
    public class ActionUnitTests : DddActionUnitTests
    {
        protected global::Domain.Model.Domain.Domain Domain => Domains.First();
```

(continues on next page)

(continued from previous page)

```
        protected List<global::Domain.Model.Domain.Domain> Domains = new();
        protected Permission Permission => Permissions.First();
        protected List<Permission> Permissions = new();
        protected Realm Realm => Realms.First();
        protected List<Realm> Realms = new();
        protected Role Role => Roles.First();
        protected List<Role> Roles = new();
        protected AccessToken Token;
        protected User User => Users.First();
        protected List<User> Users = new();

        // Setup

        protected override IWebHostBuilder CreateWebHostBuilder()
        {
            var builder = WebHost.CreateDefaultBuilder()
                .UseKestrel()
                .UseStartup<Startup>()
                .AddEnvFile($"ENV_FILE_{ActionName}", $"CFG_{ActionName}_", "", false)
                .AddSettings()
                .AddCustomSettings()
                .AddLogging();
            return builder;
        }

        protected override void EmptyAggregateRepositories(CancellationToken ct)
        {
            AssignmentRepository.DeleteAll(ActionId, CancellationToken.None);
            DomainRepository.DeleteAll(ActionId, CancellationToken.None);
            PermissionRepository.DeleteAll(ActionId, CancellationToken.None);
            RealmRepository.DeleteAll(ActionId, CancellationToken.None);
            RoleRepository.DeleteAll(ActionId, CancellationToken.None);
            UserRepository.DeleteAll(ActionId, CancellationToken.None);
        }

        protected override async Task EmptyAggregateRepositoriesAsync(CancellationToken␣
↪ct)
        {
            await AssignmentRepository.DeleteAllAsync(ActionId, CancellationToken.None);
            await DomainRepository.DeleteAllAsync(ActionId, CancellationToken.None);
            await PermissionRepository.DeleteAllAsync(ActionId, CancellationToken.None);
            await RealmRepository.DeleteAllAsync(ActionId, CancellationToken.None);
            await RoleRepository.DeleteAllAsync(ActionId, CancellationToken.None);
            await UserRepository.DeleteAllAsync(ActionId, CancellationToken.None);
        }

        protected Task EnsureRootUserAsync()
            => new EnsureRootUser(CustomSettings, UserRepository).ExecuteAsync();

        protected Task EnsureIamDomainAsync()
            => new EnsureIamDomain(DomainRepository).ExecuteAsync();
```

(continues on next page)

(continued from previous page)

```
        protected Task EnsureIamPermissionsAsync()
            => new EnsureIamPermissions(CustomSettings, UserRepository, DomainRepository,
↪ PermissionRepository).ExecuteAsync();

        // Do as actor

        protected async Task DoAsRoot(Func<Task> actionsAsync)
        {
            await AuthenticateRootUser();
            await actionsAsync();
            Credentials.JwtToken = null;
        }

        protected async Task DoAsUser(Func<Task> actionsAsync)
        {
            await AuthenticateUser();
            await actionsAsync();
            Credentials.JwtToken = null;
        }

        // Actions

        protected AddPermissionToRoleAction AddPermissionToRoleAction => TestServer.Host.
↪Services.GetRequiredService<AddPermissionToRoleAction>();
        protected AddUserToRealmAction AddUserToRealmAction => TestServer.Host.Services.
↪GetRequiredService<AddUserToRealmAction>();
        protected AssignRoleAction AssignRoleAction => TestServer.Host.Services.
↪GetRequiredService<AssignRoleAction>();
        protected AuthenticateAction AuthenticateAction => TestServer.Host.Services.
↪GetRequiredService<AuthenticateAction>();
        protected CreateAccountAction CreateAccountAction => TestServer.Host.Services.
↪GetRequiredService<CreateAccountAction>();
        protected CreateDomainAction CreateDomainAction => TestServer.Host.Services.
↪GetRequiredService<CreateDomainAction>();
        protected CreatePermissionAction CreatePermissionAction => TestServer.Host.
↪Services.GetRequiredService<CreatePermissionAction>();
        protected CreateRealmAction CreateRealmAction => TestServer.Host.Services.
↪GetRequiredService<CreateRealmAction>();
        protected CreateRoleAction CreateRoleAction => TestServer.Host.Services.
↪GetRequiredService<CreateRoleAction>();
        protected DeleteDomainAction DeleteDomainAction => TestServer.Host.Services.
↪GetRequiredService<DeleteDomainAction>();
        protected ForgetPasswordAction ForgetPasswordAction => TestServer.Host.Services.
↪GetRequiredService<ForgetPasswordAction>();
        protected GetDomainsAction GetDomainsAction => TestServer.Host.Services.
↪GetRequiredService<GetDomainsAction>();
        protected GetPermissionsGrantedAction GetPermissionsGrantedAction => TestServer.
↪Host.Services.GetRequiredService<GetPermissionsGrantedAction>();
        protected GetRoleAssignmentsAction GetRoleAssignmentsAction => TestServer.Host.
↪Services.GetRequiredService<GetRoleAssignmentsAction>();
        protected SendEmailVerificationEmailAction SendEmailVerificationEmailAction =>␣
↪TestServer.Host.Services.GetRequiredService<SendEmailVerificationEmailAction>();
```

(continues on next page)

(continued from previous page)

```
        protected VerifyEmailAction VerifyEmailAction => TestServer.Host.Services.
↪GetRequiredService<VerifyEmailAction>();

        // Auth

        protected IAuthDomainService AuthDomainService => TestServer.Host.Services.
↪GetRequiredService<IAuthDomainService>();

        // Credentials

        protected ICredentials Credentials => TestServer.Host.Services.GetRequiredService
↪<ICredentials>();

        // Settings

        protected ICustomSettings CustomSettings => TestServer.Host.Services.
↪GetRequiredService<ICustomSettings>();

        // Domains

        protected Task<global::Domain.Model.Domain.Domain> GetIamDomainAsync()
            => DomainRepository.GetWithNameInWorldAsync("IAM", ActionId,␣
↪CancellationToken.None);

        // Permissions

        protected async Task<Permission> GetIamPermissionAsync(string name)
            => (await PermissionRepository.GetWithNameInWorldAsync(name, (await␣
↪GetIamDomainAsync()).DomainId, ActionId, CancellationToken.None))!;

        // Hooks

        protected IOnBeforePrimaryAdaptersStartedHook OnBeforePrimaryAdaptersStartedHook␣
↪=> TestServer.Host.Services.GetRequiredService<IOnBeforePrimaryAdaptersStartedHook>();

        // Repositories

        protected IAssignmentRepository AssignmentRepository => TestServer.Host.Services.
↪GetRequiredService<IAssignmentRepository>();
        protected IDomainRepository DomainRepository => TestServer.Host.Services.
↪GetRequiredService<IDomainRepository>();
        protected IPermissionRepository PermissionRepository => TestServer.Host.Services.
↪GetRequiredService<IPermissionRepository>();
        protected IRealmRepository RealmRepository => TestServer.Host.Services.
↪GetRequiredService<IRealmRepository>();
        protected IRoleRepository RoleRepository => TestServer.Host.Services.
↪GetRequiredService<IRoleRepository>();
        protected IUserRepository UserRepository => TestServer.Host.Services.
↪GetRequiredService<IUserRepository>();

        // Assertions
```

(continues on next page)

(continued from previous page)

```
        protected void AssertEmailSent(Email toEmail)
            => AssertEmailSent(toEmail: toEmail, msgContains: null);

        protected void AssertEmailSent(Email toEmail, string? msgContains)
        {
            var subString = "";

            if (msgContains != null)
                subString = $" containing '{msgContains}'";

            Assert.True(
                EmailAdapter.HasSent(
                    toEmail: toEmail.ToString(),
                    msgContains: msgContains),
                $"Expected an email{subString} to be sent to {toEmail}.");
        }

        // Execute

        protected async Task AddPermissionToRole(PermissionId permissionId, RoleId␣
↪roleId)
        {
            var command = new AddPermissionToRoleCommand
            {
                PermissionId = permissionId,
                RoleId = roleId
            };

            await AddPermissionToRoleAction.ExecuteAsync(command, ActionId,␣
↪CancellationToken.None);
        }

        protected async Task AddUserToRealm(UserId userId, RealmId realmId)
        {
            var command = new AddUserToRealmCommand
            {
                UserId = userId,
                RealmId = realmId
            };

            await AddUserToRealmAction.ExecuteAsync(command, ActionId, CancellationToken.
↪None);
        }

        protected async Task AssignRole(RoleId roleId, UserId? toUserId, RealmId?␣
↪inRealmId = null)
        {
            var command = new AssignRoleCommand
            {
                RoleId = roleId,
                ToUserId = toUserId,
                InRealmId = inRealmId
```

(continues on next page)

(continued from previous page)

```
            };

            await AssignRoleAction.ExecuteAsync(command, ActionId, CancellationToken.
↪None);
        }

        protected async Task Authenticate(Email email, string password)
        {
            var command = new AuthenticateCommand
            {
                Email = email,
                Password = password
            };

            var accessToken = await AuthenticateAction.ExecuteAsync(command, ActionId,␣
↪CancellationToken.None);

            Credentials.JwtToken = JwtToken.Read(accessToken.ToString());
        }

        protected async Task AuthenticateRootUser()
        {
            var command = new AuthenticateCommand
            {
                Email = CustomSettings.RootUser.Email,
                Password = CustomSettings.RootUser.Password
            };

            var accessToken = await AuthenticateAction.ExecuteAsync(command, ActionId,␣
↪CancellationToken.None);

            Credentials.JwtToken = JwtToken.Read(accessToken.ToString());
        }

        protected async Task AuthenticateUser(string password = "test-password")
        {
            var command = new AuthenticateCommand
            {
                Email = User.Email,
                Password = password
            };

            var accessToken = await AuthenticateAction.ExecuteAsync(command, ActionId,␣
↪CancellationToken.None);

            Credentials.JwtToken = JwtToken.Read(accessToken.ToString());
        }

        protected async Task CreateAccount(string email = "test.testsson@poweriam.com",␣
↪string password = "test-password")
        {
            var command = new CreateAccountCommand
```

(continues on next page)

(continued from previous page)

```
            {
                FirstName = "Test",
                LastName = "Testsson",
                Email = Email.Create(email),
                Password = password,
                RepeatPassword = password
            };

            var user = await CreateAccountAction.ExecuteAsync(command, ActionId,␣
↪CancellationToken.None);

            Users.Add(user);
        }

        protected async Task CreateDomain(RealmId inRealmId, string name = "Test Domain",
↪ string description = "Test description")
        {
            var command = new CreateDomainCommand
            {
                Name = name,
                Description = description,
                InRealmId = inRealmId
            };

            var domain = await CreateDomainAction.ExecuteAsync(command, ActionId,␣
↪CancellationToken.None);

            Domains.Add(domain);
        }

        protected async Task CreatePermission(string name = "Test Permission", RealmId?␣
↪inRealmId = null, DomainId? inDomainId = null)
        {
            var command = new CreatePermissionCommand
            {
                Name = name,
                Description = "Test Permission",
                ExternalId = "some-external-id",
                InRealmId = inRealmId,
                InDomainId = inDomainId
            };

            var permission = await CreatePermissionAction.ExecuteAsync(command, ActionId,
↪ CancellationToken.None);

            Permissions.Add(permission);
        }

        protected async Task CreateRealm(string name = "Test Realm")
        {
            var command = new CreateRealmCommand
            {
```

(continues on next page)

(continued from previous page)

```
                Name = name,
                Description = "Test Realm",
                ExternalId = "some-external-id"
            };

            var realm = await CreateRealmAction.ExecuteAsync(command, ActionId,␣
↪CancellationToken.None);

            Realms.Add(realm);
        }

        protected async Task CreateRole(string name = "Test Permission", RealmId?␣
↪inRealmId = null, string? inExternalRealmId = null)
        {
            var command = new CreateRoleCommand
            {
                Name = name,
                Description = "Test Role",
                InRealmId = inRealmId,
                InExternalRealmId = inExternalRealmId
            };

            var role = await CreateRoleAction.ExecuteAsync(command, ActionId,␣
↪CancellationToken.None);

            Roles.Add(role);
        }

        protected async Task<IEnumerable<Assignment>> GetRoleAssignments(UserId toUserId,
↪ RealmId? inRealmId = null)
        {
            var command = new GetRoleAssignmentsCommand
            {
                ToUserId = toUserId,
                InRealmId = inRealmId
            };

            var assignments = await GetRoleAssignmentsAction.ExecuteAsync(command,␣
↪ActionId, CancellationToken.None);

            return assignments;
        }

        // Data

        protected async Task Refresh(User user)
        {
            var users = new List<User>();
            foreach (var u in Users)
                if (u.UserId == user.UserId)
                    users.Add(await UserRepository.GetAsync(u.UserId, ActionId,␣
↪CancellationToken.None));
```

(continues on next page)

(continued from previous page)

```
                else
                    users.Add(u);
            Users = users;
        }
    }
}
```

1.5 Troubleshooting

If you suspect something in the nuget isn't working as expected, it will be helpful to increase the logging level of the framework to the `DEBUG` level in the `env file` like this:

```
CFG_LOGGING_LEVEL=Debug
```

This should provide useful information about what's going on inside the OpenDDD.NET core.

1.6 Semantic Versioning

The SemVer2.0 policy is used for versioning of the domain model, as well as the HTTP API of the primary http adapter, and this nuget.

In SemVer2.0, *backwards compatible* changes increments the patch- and minor versions, whereas *backwards incompatible* changes increments the major version.

See the table for examples how this works.

Table 1: Examples: When to increment which version.

Code Status	Stage	Rule	Example version
First release	New product	Start with 1.0.0	1.0.0
Backward-compatible bug fixes	Patch release	Increment the third digit	1.0.1
Backward-compatible new features	Minor release	Increment the middle digit and reset last digit to zero	1.1.0
Changes that break backward compatibility	Major release	Increment the first digit and reset middle and last digits to zero	2.0.0

1.7 Design Patterns

We believe in `standing on the shoulders of giants`, and not to reinvent the wheel.

The software industry has summarized best design patterns and practices for software development and this framework is based on the following design patterns:

- Domain-Driven Design
- Hexagonal Architecture
- Event-Carried State Transfer
- Near-infinite Scalability
- xUnit
- Expand and Contract
- Env files

You are encouraged to `read up on each of the patterns` above. They are foundational for the framework and the better you know them the easier time you will have to get going using this framework.

1.8 Version history

Note: We are still in alpha stage.

Tip: Subscribe to the github repository to get notifications when we move into beta and rc phases!

1.0.0-alpha.17 - 2023-06-27

- Add DateTimeProvider and support time traveling in tests.
- Add some test convenience- and assertion methods.
- Refactor error code numbers.

1.0.0-alpha.16 - 2023-05-07

- Add support for multiple listeners per event.
- Setting *MaxDeliveryRetries* of '0' now means '0 retries' (not infinite retries).
- Add test method to simulate receiving a domain event.

1.0.0-alpha.15 - 2023-05-01

- Re-enable previously disabled publisher service.
- Change message bus topic name format for events.

1.0.0-alpha.14 - 2023-04-30

- Change listeners to wildcard both minor and patch versions.

1.0.0-alpha.13 - 2023-04-28

- Rename 'Serialization' to 'Conversion'.
- Add 'PositiveIamAdapter' that permits everything.

1.0.0-alpha.12 - 2023-04-28

- Rename framework to 'OpenDDD.NET'.
- Add project template for .NET Core 3.1.
- Add project template for .NET 5.
- Introduce Transactional and use in Action. (**breaking**)
- Add extension method 'AddDomainService()'.

1.0.0-alpha.11 - 2023-04-25

- Add support to disable emails in tests.
- Fix code generation templates.
- Replace IApplicationLifetime with IHostApplicationLifetime. (**breaking**)

1.0.0-alpha.10 - 2023-04-24

- Add more synchronous versions of methods used by tests.
- Break out application error classes.
- Fix minor issue in code generation tool.

1.0.0-alpha.9 - 2023-04-19

- Add synchronous versions of methods. (**breaking**)

1.0.0-alpha.8 - 2023-04-11

- Add support for context hooks.
- Add error codes support. (**breaking**)
- Fix database connections leak.
- Add support for enabling/disabling publishers in tests.
- Add assertion methods.
- Fix issues with running tests in parallell.
- Use newtonsoft json everywhere. (**breaking**)
- Add base email adapter. (**breaking**)
- Properly start & stop outbox. (**breaking**)
- Properly start & stop repositories. (**breaking**)

1.0.0-alpha.7 - 2023-01-01

- Add credentials support to smtp adapter.
- Use api version 2.0.0 in poweriam adapter.

1.0.0-alpha.6 - 2023-01-01

- Add base class for domain services.
- Use new permissions string format: "<domain>:<permission>". (**breaking**)

1.0.0-alpha.5 - 2022-12-26

- Refactor to follow semver2.0 strictly in http adapter. (**breaking**)
- Add support for configuring persistence pooling.

- Add html support to email port. (**breaking**)
- Fix memory leak where db connections weren't closed.

1.0.0-alpha.4 - 2022-12-10

- Add configuration setting for which server urls to listen to. (**breaking**)
- Fix concurrency issues with memory repositories.
- Add support for IAM ports.
- Add 'PowerIAM' adapter.
- Add RBAC auth settings. (**breaking**)
- Add a base 'Migrator' class. (**breaking**)

1.0.0-alpha.3 - 2022-11-20

- Refactor JwtToken and add IdToken. (**breaking**)
- Add more tasks to code generation tool.
- Add support for http put methods to code generation tool.
- Add some missing repository method implementations.
- Add GetAsync(IEnumerable<...> ...) to repositories.
- Add convenience methods to ApplicationExtensions.
- Return 400 http status code on domain- and invariant exceptions in primary http adapter.

1.0.0-alpha.2 - 2022-10-09

- Make the hexagonal architecture more represented in the namespaces.

1.0.0-alpha.1 - 2022-10-02

This is the first (alpha) release of the framework. Please try it out and submit tickets or otherwise reach out if you find any issues or have any questions.

0.9.0-alpha7 - 2022-07-31

First alpha release on nuget.org.

1.9 Contribution

You are welcome to suggest changes and to submit bug reports at the github repository.

Note: We rely on the community to come up with more in-depth guides on how to develop with the framework, e.g. how to setup Rider, Visual Studio or other IDEs and editors.

Tip: If you have a guide you think should be included in this documentation, please submit it to us.
