

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Современные платформы программирования. Часть 2

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

Программное средство считывания RSS-новостей

БГУИР КП 1-40 01 01 01

Студент: гр. 791051 Захаренко В. В.

Руководитель: Видничук В. Н.

Минск 2022

СОДЕРЖАНИЕ

Введение.....	3
1. Описание предметной области.....	4
1.1 Обзор аналогов.....	4
1.2 Основные сведения о микросервисной архитектуре.....	8
1.3 Постановка задачи.....	10
2. Разработка программного средства.....	11
2.1 Общая структура и принцип работы программы.....	11
3. Руководство пользователя.....	14
Заключение.....	20
Список используемой литературы.....	21
Приложение А. исходный код программы.....	22

ВВЕДЕНИЕ

Информацию сегодня, в основном, люди получают из телевиденья и интернета. Именно тут ежедневно возникают новые данные. Часто люди просто не успевают за ними следить, так всё очень быстро меняется. Например, то событие, которое было вчера актуальным, может сегодня уже потерять свою значимость. И в этом является основная сложность.

Мы живём в эпоху постиндустриализма. Именно сегодня знания играют самую важную роль. Но, важно помнить, что они могут устаревать. Поэтому следует постоянно следить за всем, что происходит в мире. Это позволит человеку оставаться в курсе всех новостей.

Некогда для упрощения доступа к новостям был придуман формат RSS. RSS (англ. Rich Site Summary — обогащённая сводка сайта) — семейство XML-форматов, предназначенных для описания лент новостей, анонсов статей, изменений в блогах и т. п. Информация из различных источников, представленная в формате RSS, может быть собрана, обработана и представлена пользователю в удобном для него виде специальными программами-агрегаторами или онлайн-сервисами. Обычно с помощью RSS 2.0 даётся краткое описание новой информации, появившейся на сайте, и ссылка на её полную версию. Интернет-ресурс в формате RSS называется RSS-каналом, RSS-лентой или RSS-фидом.

В ходе выполнения данного курсового проектирования будет разработано веб-приложение, которое представляет собой агрегатор новостных сводок с различных сайтов с возможностью задавать фильтрацию новостей по ключевым словам и настраивать рассылку новостей на электронный почтовый ящик. При создании программного средства планируется изучить основные принципы проектирования и реализации микросервисной архитектуры, а также закрепить знания о разработке веб-приложений на платформе .NET.

1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

Агрегаторы новостей, в том или ином виде, до сих пор являются популярными, благодаря чему в распоряжении имеется масса аналогов. Я рассмотрю только некоторые из них.

1. Readiy

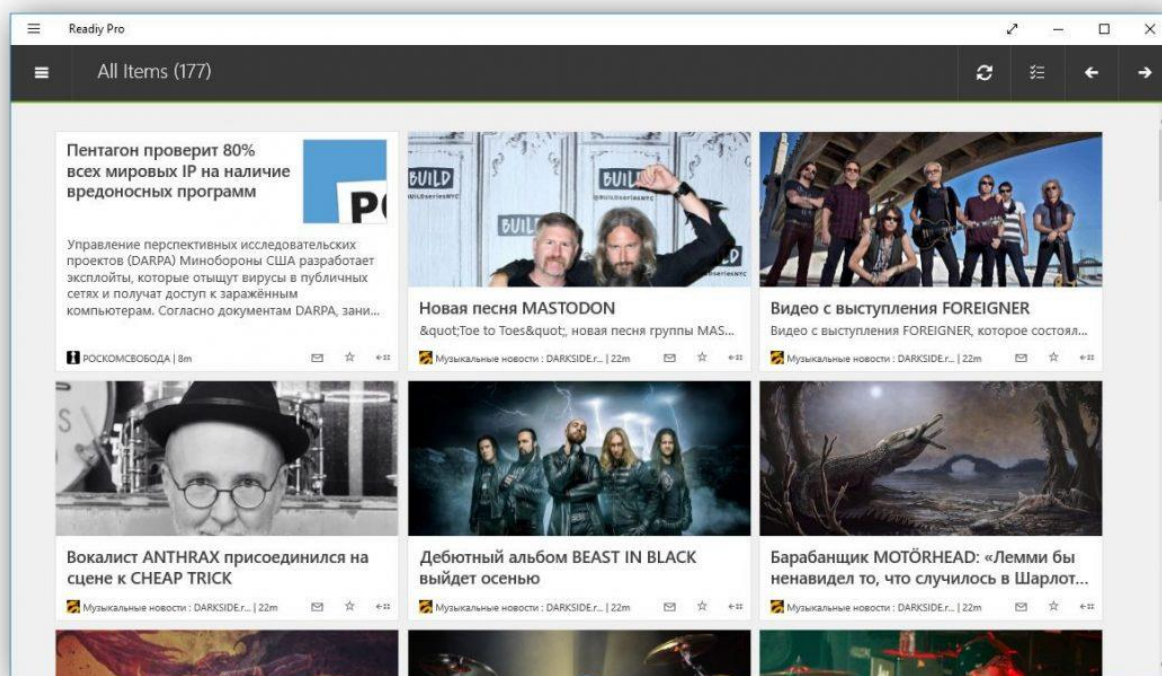


Рис 1.1 Readiy

Readiy — быстрый и простой клиент RSS. Его интерфейс оформлен в стиле Windows 10, так что его можно использовать на планшетных компьютерах. Readiy не поддерживает расширенных настроек отображения. Если вы хотите фильтровать поступающие потоки информации, это не лучший выбор. Но для удобного чтения новостей приложение подходит.

2. Veen Reader

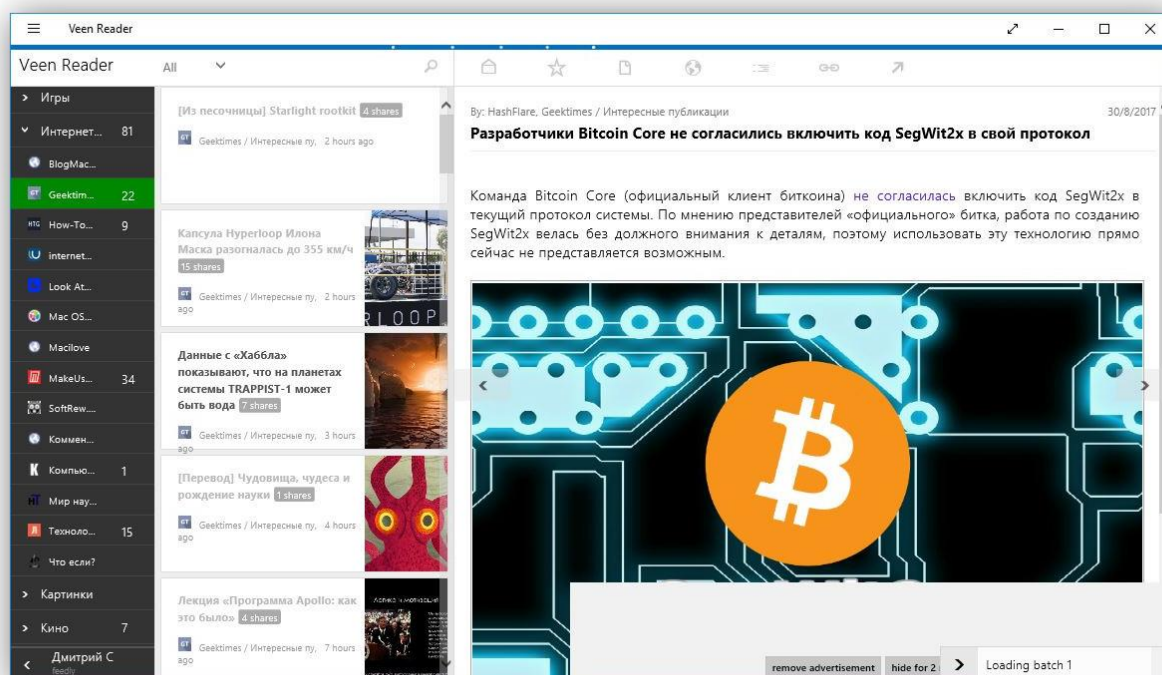


Рис 1.2 Veen Reader

Veen Reader умеет отправлять статьи в Pocket, Instapaper, Twitter, Facebook, LinkedIn, Buffer и OneNote. Можно настраивать шрифты и шаблоны отображения в статьях.

3. Nextgen Reader

Nextgen Reader пользуется популярностью в Windows Store, но не особенно отличается функциональностью в сравнении с предыдущими приложениями. Этот агрегатор может делиться статьями через системное меню и отображать их целиком, не открывая браузер.

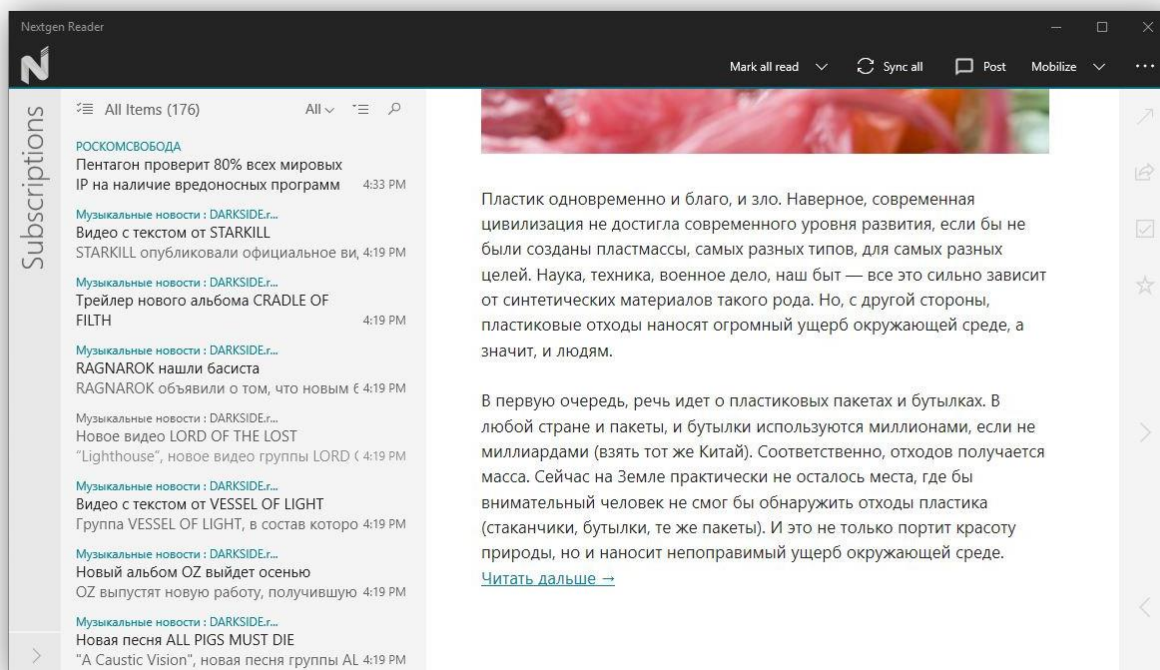


Рис 1.3 Nextgen Reader

4. Hypersonic

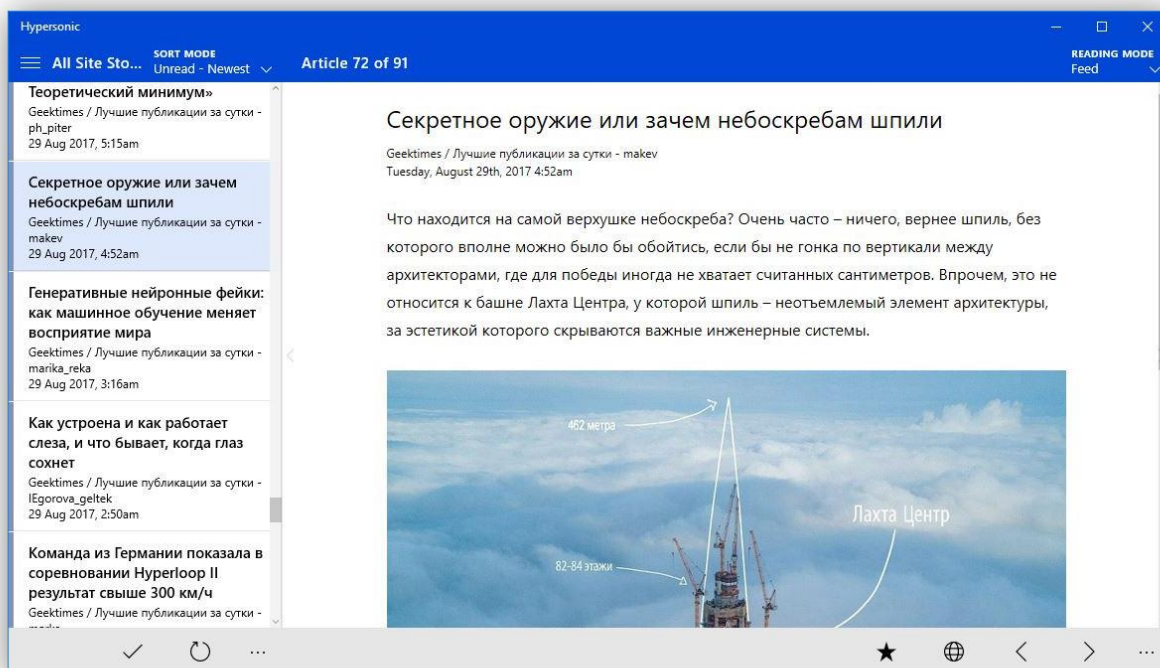


Рис 1.4 Hypersonic

Hypersonic имеет минималистичный интерфейс. Чтобы быстро переключаться между статьями, можно использовать свайп. Приложение отображает полный текст статьи по запросу.

5. Vienna

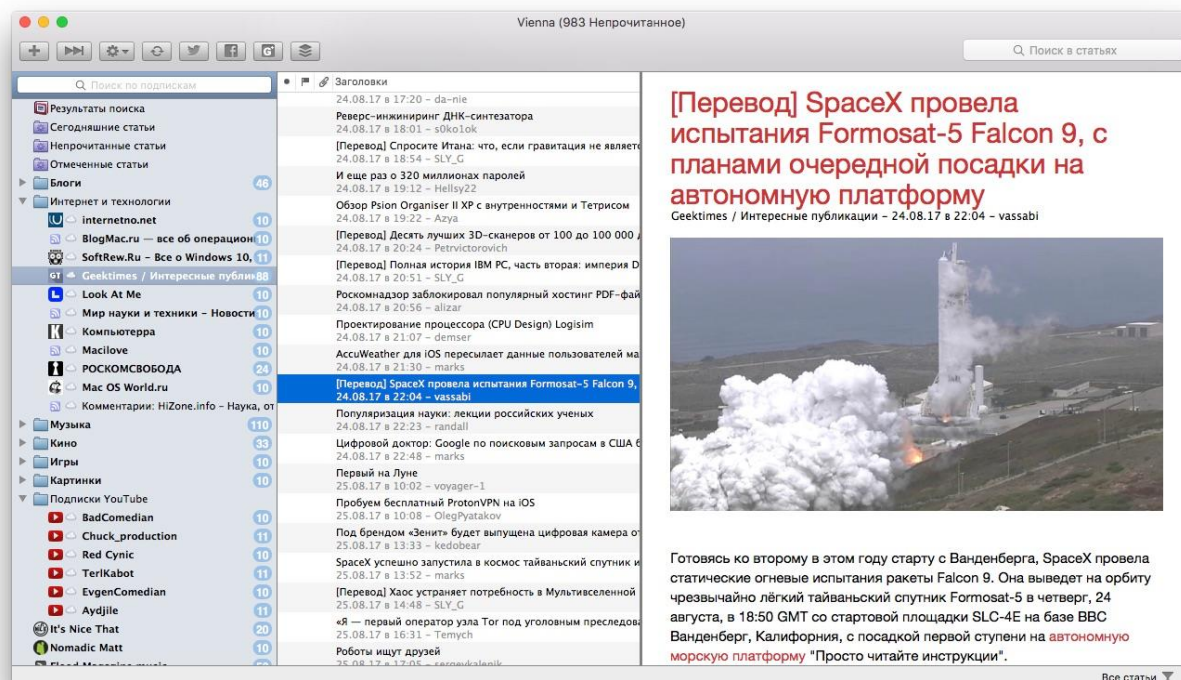


Рис 1.5 Vienna

Vienna поддерживает не столь большое количество RSS-сервисов, как Reeder или ReadKit, но оно абсолютно бесплатно. Кроме того, можно создавать смарт-папки для ваших подписок и применять умные фильтры.

6. FeedReader

FeedReader — очень простой RSS-клиент, у которого есть все необходимые функции для удобного чтения новостей. Он может искать нужные статьи по ключевым словам, отмечать их метками, перемещать в «Избранное», отправлять посты в Pocket, Instapaper или Wallabag и делиться ими по почте, Telegram или через Twitter.

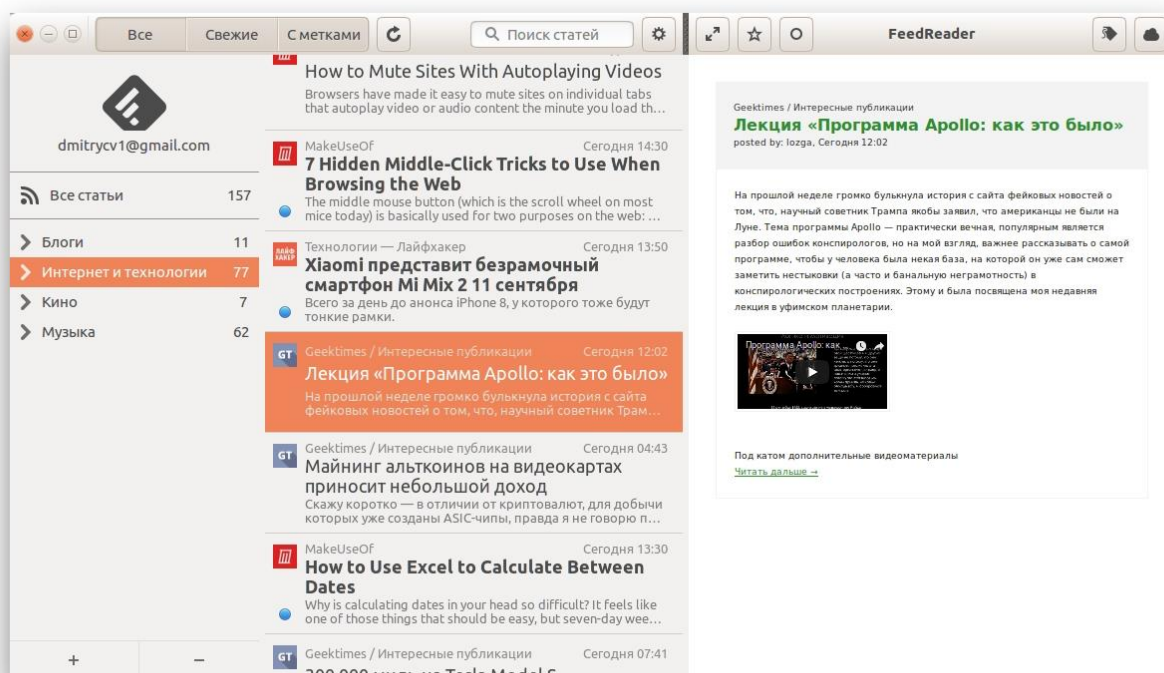


Рис 1.6 FeedReader

1.2 Основные сведения о микросервисной архитектуре

Достаточно долгое время преобладающим при разработке информационных систем был способ создания «монолитных» приложений, при котором изначально не выделялись ни пользовательский уровень, ни уровень бизнес-логики. Далее появились подходы, в которых стали выделять несколько уровней (соответствует монолитной клиент-серверной архитектуре):

- пользовательские интерфейсы;
- уровень бизнес-логики системы;
- уровень баз данных;

Первый уровень называют фронт-эндом (Front-End), второй и третий – бек-эндом (Back-End). Соответственно разработчики условно разбивались на две группы: Front-End- и Back-End-разработчики. Главным недостатком монолитных приложений является необходимость сборки всего приложения после внесения изменений, что приводит к следующим последствиям:

- существенно увеличивается время на развертывание системы и внесение изменений;

- сильная связь кода делает разработчиков зависимыми друг от друга, сбой в работе одного компонента приводит к отказу всей системы;
- большая кодовая часть не позволяет легко перейти на новые технологии, внесение изменений также затруднено, в результате приложения морально устаревают;
- затруднено масштабирование приложений;
- при внесении изменений количество ошибок значительно возрастает, что приводит к частым откатам системы;

Риски, связанные с существованием указанных недостатков, могут быть устранены путем деления приложений на более мелкие программные составляющие, что привело к возникновению способа проектирования и организации информационной архитектуры и бизнес-функциональности с использованием сервис-ориентированной архитектуры и микросервисной архитектуры.

СОА – это парадигма организации и использования распределенных возможностей. Ключевым понятием является сервис: независимая от прочих сервисов компонента информационной системы. В качестве сервиса может выступать как целое приложение, так и отдельные его функциональные модули. Сервисы могут реализовывать как бизнес-логику, так и функции более низкого уровня, в том числе некоторые системные функции.

Дальнейшим развитием СОА стала МСА, которая, по сути, является одним из способов реализации СОА. В данном случае приложение представляет собой набор небольших независимых сервисов, взаимодействующих через обмен сообщениями, например, по протоколу HTTP. При этом микросервисы могут располагаться на разных серверах, быть написанными на разных языках программирования с использованием различных технологий.

Важным элементом МСА и СОА является децентрализованное хранение данных, при котором каждый сервис может иметь свою базу данных. Все взаимодействие идет через сообщение сервисов.

Микросервисы позволяют проводить горизонтальное масштабирование приложения, при этом сервисы могут взаимодействовать между собой синхронно с использованием HTTP/REST или асинхронно с использованием брокеров сообщений типа Apache Kafka.

МСА не стоит использовать как единственный способ создания приложений, так как при использовании этой архитектуры может возникнуть целый ряд трудностей:

- Необходимо решать вопросы, связанные с обеспечением безопасности данных. Например, может потребоваться шифрование при передаче данных, должны быть продуманы и качественно реализованы решения, связанные с аутентификацией сторон.
- Необходимость шифрования, а также возникновение задержки передачи данных в сети, необходимость виртуализации приводят к замедлению работы системы в целом.
- Обратная сторона простоты в сопровождении заключается в том, что микросервисы менее надежны, чем монолитная система. А кроме того, при существенном увеличении сервисов возникают трудности конфигурирования системы в целом.
- Трудно оценить стоимость перехода от монолитного приложения к микросервисному.

1.3 Постановка задачи

В рамках данного курсового проектирования планируется разработать программное средство считывания RSS-новостей. Средство будет реализовано согласно микросервисной архитектуры с применением следующих технологий: ASP.NET Core, Docker, PostgreSQL, jQuery. В приложении будут реализованы следующие функции:

- добавление, изменение и удаление RSS-источников;
- настройка фильтрации новостей по ключевым словам;
- периодическая рассылка новостей по электронной почте с возможностью настройки периодичности рассылки и списка электронных почтовых ящиков, на которые и будет приходить рассылка;
- просмотр новостей сразу на сайте;

2. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1 Общая структура и принцип работы программы

Программа представляет собой набор микросервисов, написанных на платформе .NET, .NET веб-приложение, предоставляющее графический пользовательский интерфейс, СУБД PostgreSQL и тестового SMTP-сервера Mailhog, общение между частями происходит по протоколу HTTP, СУБД и SMTP-сервер поднимаются в Docker-контейнерах, микросервисы же запускаются при помощи инструмента Tye.

Некоторые из микросервисов хранят данные, относящиеся к работе этого сервиса. Эти данные хранятся в СУБД PostgreSQL. Эта СУБД была выбрана по причине своей бесплатности, в сравнении с SQL Server или Oracle Database. В рамках курсового проекта можно было выбрать любую другую бесплатную СУБД. В сторону NoSQL решений я не смотрел потому, что бонусов заметных ни одно из этих решений не дало бы, но я бы лишился весьма удобного инструмента - ORM-библиотеки Entity Framework, которая работает только с реляционными базами данных. Исключительно для удобства, я решил не создавать по отдельному экземпляру СУБД на каждый такой микросервис, но я разграничиваю доступ к данным благодаря схемам (отдельная схема на каждый микросервис). Перейти на вариант с отдельными экземплярами будет несложно.

Для общения между частями системы и между системой и пользователем используется протокол HTTP. При чем я старался реализовывать это общение согласно REST, но был не совсем в этом последователен и для удобства отходил от этой практики местами. Конкретно для общения с микросервисами поверх протокола HTTP построено что-то наподобие RPC. Для каждого микросервиса я создал дополнительно 2 библиотеки. В одной находится описание публичного интерфейса, который реализован в сервисе. А во второй - другая реализацию этого интерфейса, которая занимается только тем, что шлет запросы сервису по HTTP, получает ответ и переводит его в удобоваримое состояние. Возможно, в данном случае было бы лучше воспользоваться готовой библиотекой, реализующей RPC.

Также в целях удобства я решил не запускать полноценный SMTP-сервер, а вместо него использую Mailhog - инструмент для проверки отправки электронной почты. Письма не приходят на реальную почту, но все отправленные письма можно посмотреть на веб-панели Mailhog.

Мной были реализованы следующие микросервисы:

1. Downloader.API. Сервис отвечает за загрузку новостей из RSS-источников, хранение адресов всех используемых пользователем источников и проверку валидности введенного пользователем URL-адреса источника. Мне привычнее работает с форматом данных JSON, так что, может быть, стоило воспользоваться сервисом перевода RSS формата в JSON.
2. Filter.API. Тут происходит фильтрация новостей по ключевым словам и хранение всех указанных пользователем фильтров. Этот функционал работает достаточно просто, при дальнейшем развитии проекта можно улучшить механизм фильтрации.
3. Sender.API. Этот сервис занимается отправкой новостей на электронную почту и хранением адресов электронной почты, указанных пользователем. Пока сервис отправляет новости в тестовый инструмент Mailhog, но можно в файле настроек сервиса указать ссылку на любой реально работающий SMTP-сервер, чтобы почта стала приходить.
4. Manager.API. Сервис сконцентрирован на управлении рассылкой новостей: тут хранится периодичность рассылок, информация о том, хочет ли пользователь, чтобы ему отправлялись новости на почту, а также тут, при помощи библиотеки Hangfire происходит запуск рассылки по расписанию. Периодичность рассылки указывается в формате cron-выражений.
5. Gateway.API. Сервис-агрегатор для связи всех микросервисов воедино, принимает пользовательские запросы и перенаправляет их на указанные выше сервисы.

У каждого из вышеуказанных микросервисов есть удобная документация, реализованная с помощью инструмента Swagger, который также предоставляет удобный интерфейс для тестирования API.

Для конечного пользователя было создано веб-приложение Front, который при помощи Razor Pages, CSS и jQuery предоставляет удобный и красивый графический интерфейс (по задумке он должен быть таковым, но мне, к сожалению, не хватает дизайнерской компетенции) для доступа к функционалу программного средства. Front не имеет никакой бизнес-логики внутри, а только перенаправляет пользовательские запросы на Gateway.API и отдает ответ в виде веб-страницы. Мной был выбран именно jQuery, а не более популярные ныне библиотеки, такие как Vue.js или React потому, что размеры приложения не дали бы раскрыться преимуществам этих библиотек,

и я бы написал гораздо кода для настройки этих библиотек, чем было написано кода на jQuery. В теории, можно было бы использовать ванильный JavaScript, но я нашел библиотеку для jQuery, которая предоставляет компонент графического интерфейса для удобного управления cron-выражениями и из-за нее решил остаться на jQuery.

Я использовал инструмент Tye (Tye project), предназначенный для упрощения работы с микросервисным приложением. Полностью функционал этого инструмента раскрыть у меня не вышло, но немного приятных мелочей он мне дал.

Что касается архитектуры, мой выбор варианта микросервисов вызван моим интересом к этому способу построения приложений. Даже на таком маленьком приложении я увидел, что еще далек от хорошего умения в этой области. Самым главным вопросом, вставшим передо мной, был о том, как разбить логику приложения на максимально независимые сервисы или как разбить данные, необходимые для работы приложения, между микросервисами. Также при проектировании программного средства я старался придерживаться подходов MVC, CQRS и Layered Architecture.

3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для использования приложения, первым делом, надо подготовить среду. Необходимо установить .NET 6, Docker и PowerShell (последнее не обязательно, скрипты на PowerShell достаточно просты, и их можно выполнить в любой консоли без внесения правок). После чего надо выполнить скрипт `setup.ps1`, который поставит `ef-tools` и `tye`. Далее идет запуск. Выполните скрипт `run-env.ps1` для разворачивания Docker-контейнеров и дождитесь, когда все контейнеры полностью проинициализируются. После выполните скрипт `run-app.ps1` для запуска приложения.

При открытии сайта Вас встретит стартовая страница. На ней написано краткое описание сервиса и последовательности шагов того, как им пользоваться.

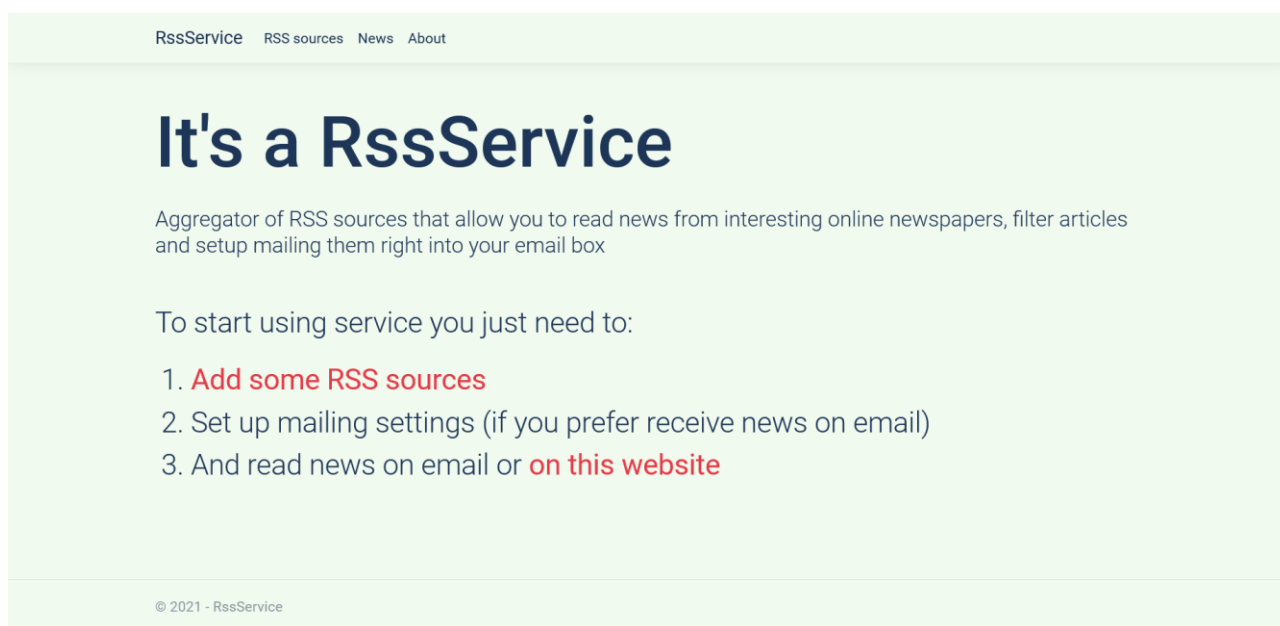


Рис 3.1 Стартовая страница

Сверху страницы можно увидеть панель навигации с основными ссылками сайта: ссылка на стартовую страницу, на страницу со списком RSS-источников, на страницу с новости и страницу с описанием сервиса. Если кликнуть на ссылку «RSS sources», то попадаешь на страницу со списком всех добавленных RSS-источников.

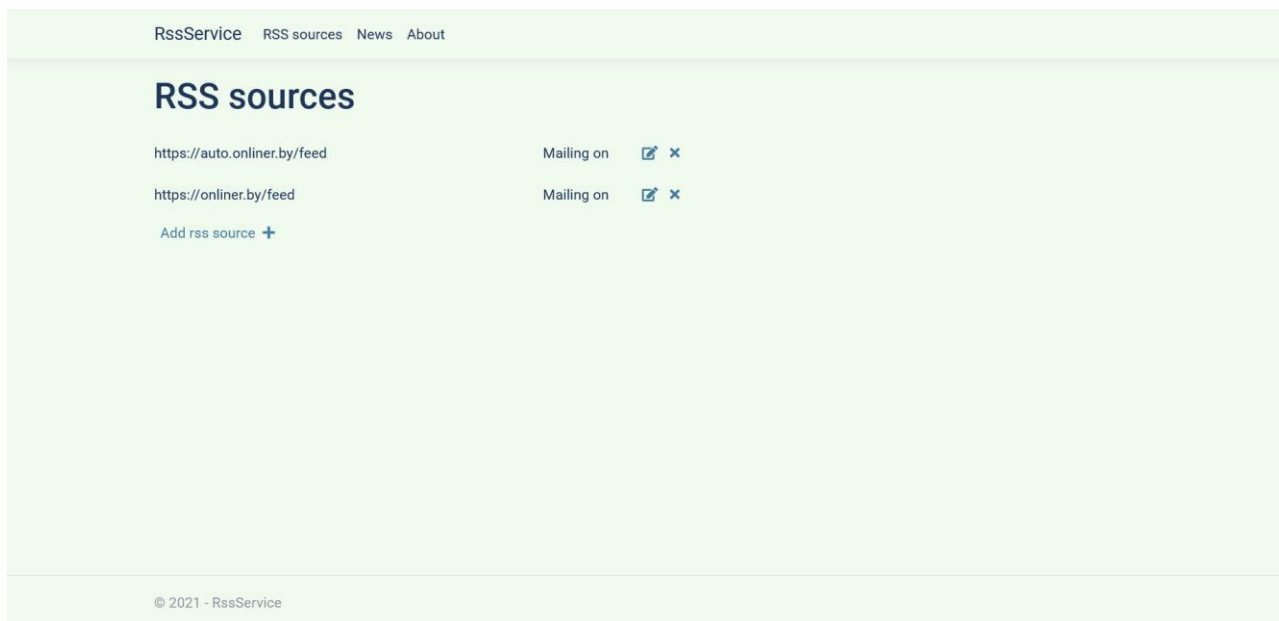


Рис 3.2 Список RSS-источников

Тут можно увидеть, что для каждого источника указан адрес, включена или нет рассылка на почту и имеются кнопки изменения и удаления источника. Также внизу представлена кнопка добавления нового источника. Если нажать кнопку удаления, то источник пропадет из списка. При нажатии кнопки изменения Вас перенаправит на страницу изменения источника.

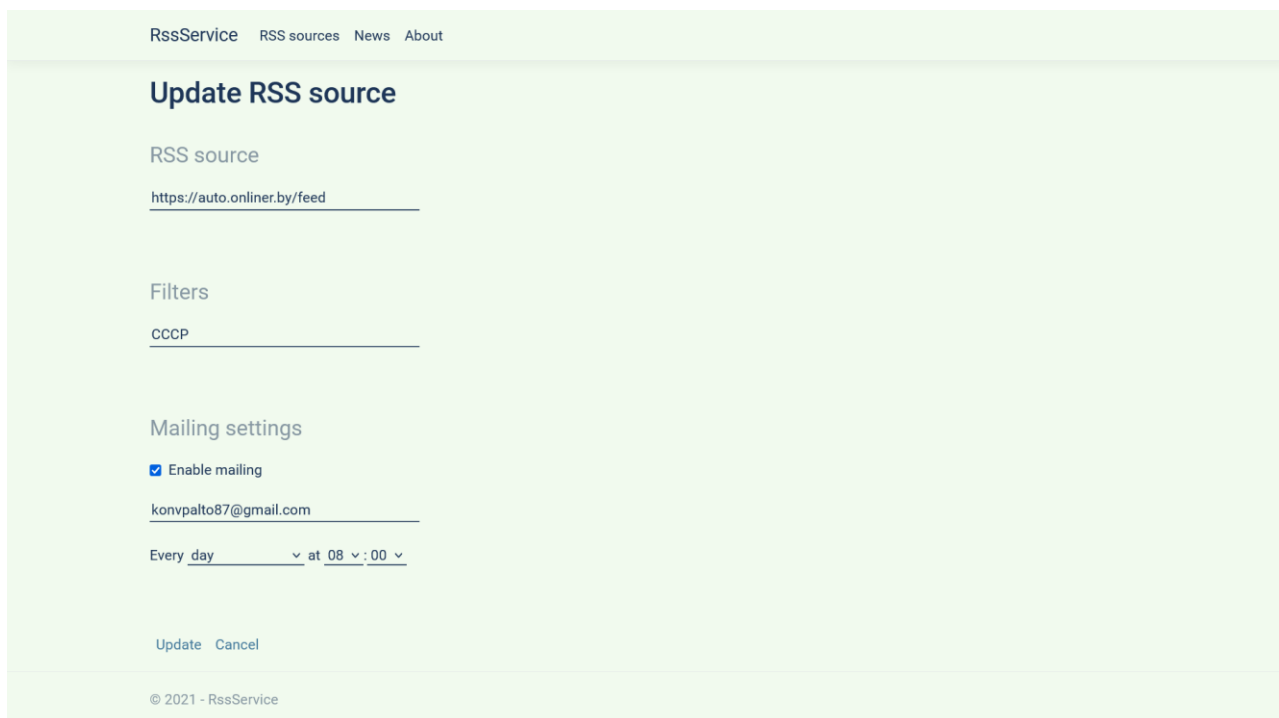


Рис 3.3 Страница изменения настроек RSS-источника

Тут можно увидеть поле для ввода адреса RSS-источника, поля для ввода ключевых слов для фильтрации (слова должны быть разделены запятой; при фильтрации приложение будет отбирать только тебе новости, в заголовке или тексте которого входит хотя бы одно из указанных слов, если слов указано не будет, то все новости будут отобраны), флажок, включающий и отключающий рассылку (установленный флажок открывает следующие за ним поля, а отключенный - прячет их), поле для ввода адреса электронного почтового ящика получателей рассылки (тоже может быть несколько значений, их надо указывать через запятую), Несколько полей для установки расписания рассылки новостей, и в самом низу кнопка подтверждения изменений вместе со ссылкой отмены изменений.

Если вернуться на страницу со списком RSS-источников, и нажать на кнопку добавления нового источника, то мы перейдем на страницу добавления источника, которая похожа на страницу изменения, за исключением некоторых слов и незаполненных полей.

RssService RSS sources News About

New RSS source

RSS source

Url

Filters

Key words (comma separated)

Mailing settings

☐ Enable mailing

Add Cancel

© 2021 - RssService

Рис 3.4 Страница добавления RSS-источника

Если нажать на ссылку «News» на панели навигации, то попадете на страницу с новостями из всех указанных источников.

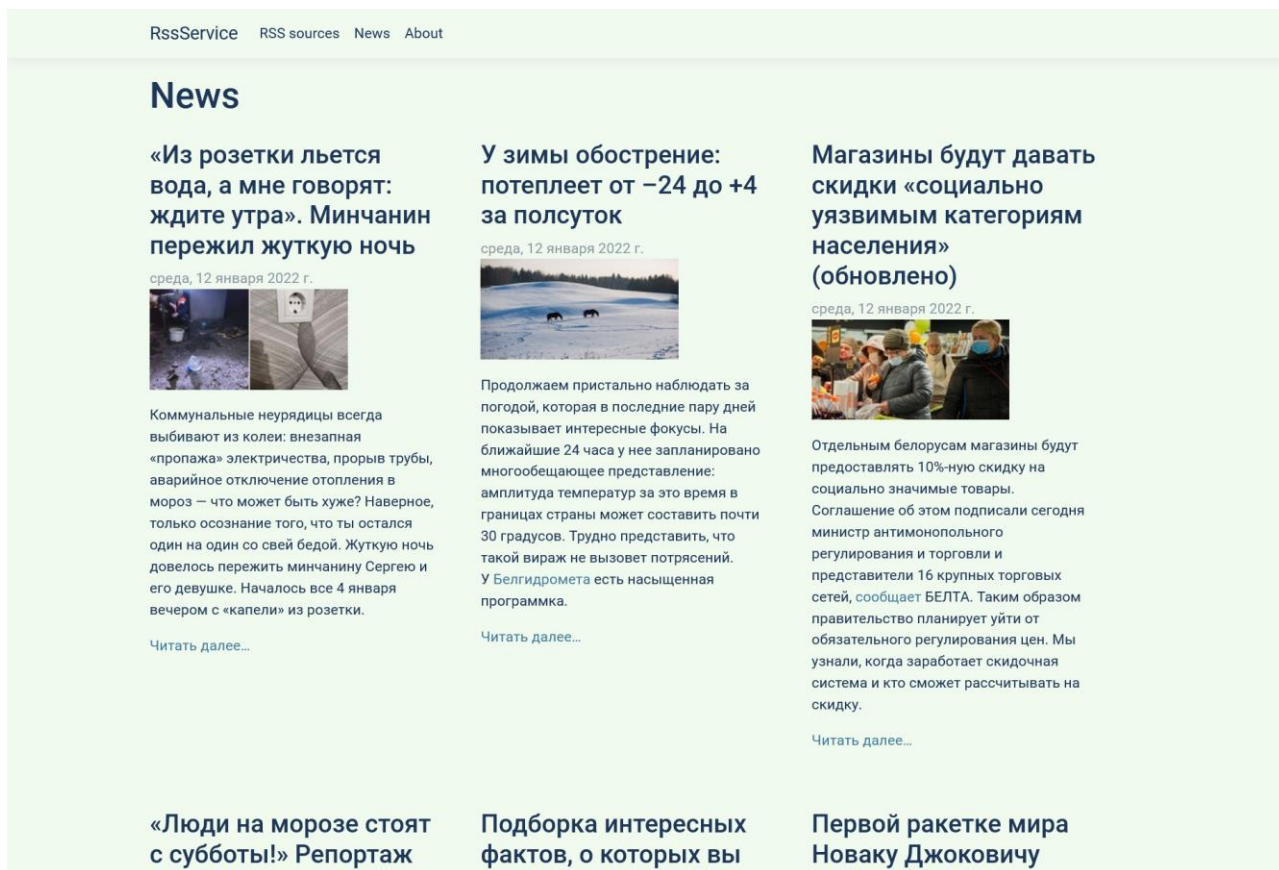


Рис 3.5 Страница новостей

Новости отображены в несколько колонок. У каждой новости есть заголовок, дата публикации и сам текст.

Ссылка «About» в навигационной панели ведет на страницу с описанием сервиса, которую я уже описывал выше.

Для проверки выполнения рассылки новостей на электронную почту надо посетить портал Mailhog-a., который по умолчанию доступен по адресу localhost:8025.

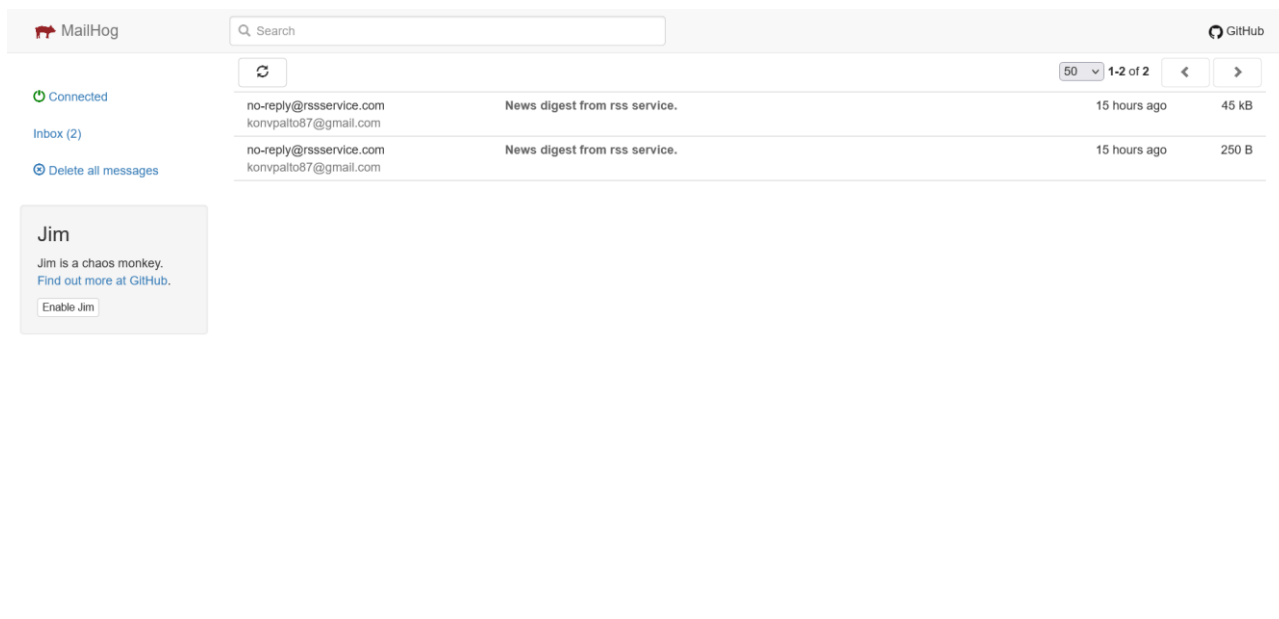


Рис 3.6 Страница Mailhog-a

В целях тестирования и администрирования может понадобиться посетить портал Hangfire-a с расписанием выполнения новостной рассылки или портал Тве, на котором можно посмотреть логи и метрики по каждому выполняющемуся сервису. Первый доступен по адресу localhost:5008/hangfire.

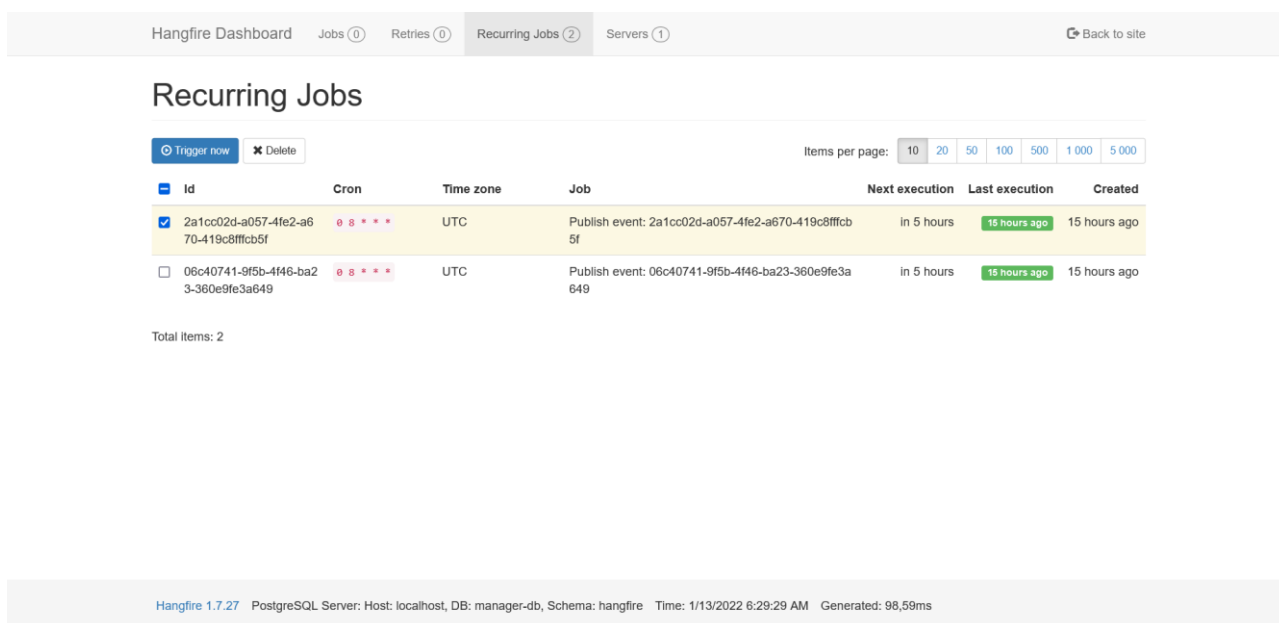
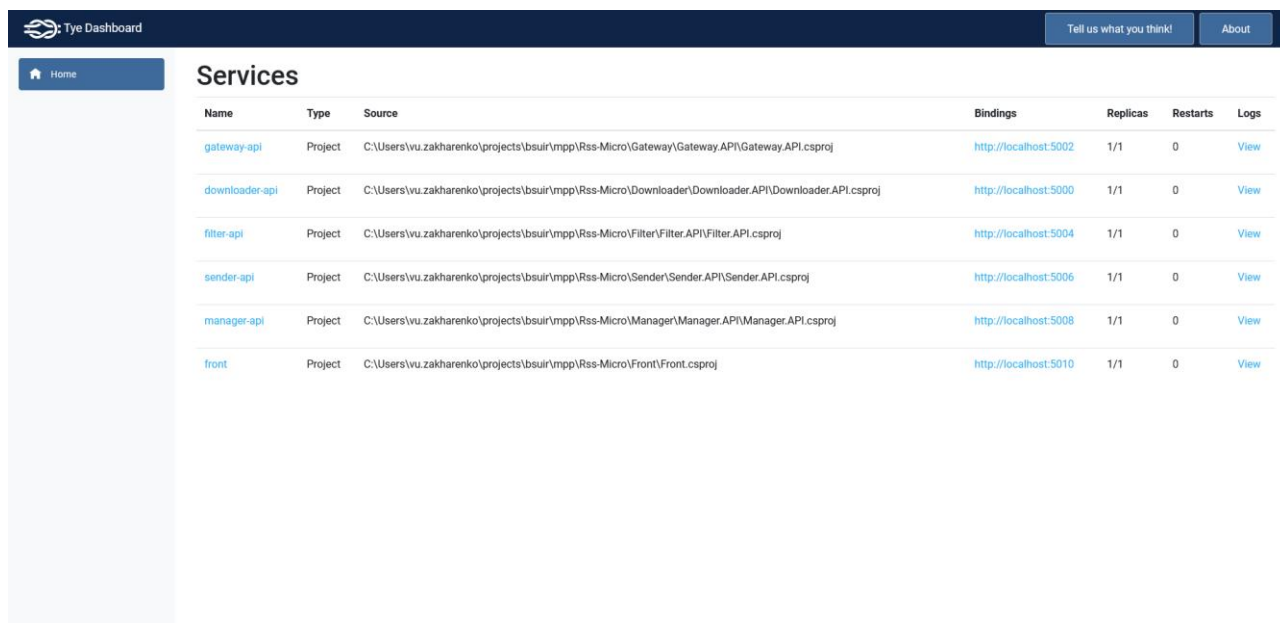


Рис 3.7 Портал Hangfire-a

Тут можно посмотреть информацию по каждой рассылке и выполнить ее вне очереди.

Тве же доступен по адресу localhost:8000.



The screenshot shows the Tye Dashboard interface. At the top, there is a dark blue header with the Tye logo and the text "Tye Dashboard". On the right side of the header, there are two buttons: "Tell us what you think!" and "About". Below the header, there is a sidebar on the left with a "Home" button. The main content area is titled "Services" and contains a table with the following columns: Name, Type, Source, Bindings, Replicas, Restarts, and Logs. The table lists six services: gateway-api, downloader-api, filter-api, sender-api, manager-api, and front. Each service has a "View" link in the Logs column.

Name	Type	Source	Bindings	Replicas	Restarts	Logs
gateway-api	Project	C:\Users\vu.zakharenko\projects\bsuir\mpp\Rss-Micro\Gateway\Gateway.API\Gateway.API.csproj	http://localhost:5002	1/1	0	View
downloader-api	Project	C:\Users\vu.zakharenko\projects\bsuir\mpp\Rss-Micro\Downloader\Downloader.API\Downloader.API.csproj	http://localhost:5000	1/1	0	View
filter-api	Project	C:\Users\vu.zakharenko\projects\bsuir\mpp\Rss-Micro\Filter\Filter.API\Filter.API.csproj	http://localhost:5004	1/1	0	View
sender-api	Project	C:\Users\vu.zakharenko\projects\bsuir\mpp\Rss-Micro\Sender\Sender.API\Sender.API.csproj	http://localhost:5006	1/1	0	View
manager-api	Project	C:\Users\vu.zakharenko\projects\bsuir\mpp\Rss-Micro\Manager\Manager.API\Manager.API.csproj	http://localhost:5008	1/1	0	View
front	Project	C:\Users\vu.zakharenko\projects\bsuir\mpp\Rss-Micro\Front\Front.csproj	http://localhost:5010	1/1	0	View

Рис 3.8 Электронная доска Тве

ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проектирования было разработано программное средство считывания RSS-новостей. Были получены знания о микросервисной архитектуре, технологиях ASP.NET Core, Docker, PostgreSQL, jQuery. В приложении были реализованы следующие функции:

- добавление, изменение и удаление RSS-источников;
- настройка фильтрации новостей по ключевым словам;
- периодическая рассылка новостей по электронной почте с возможностью настройки периодичности рассылки и списка электронных почтовых ящиков, на которые и будет приходить рассылка;
- просмотр новостей сразу на сайте;

Список используемой литературы

1. 11 самых удобных RSS-агрегаторов для Windows, Mac и Linux - Лайфхакер [электронный ресурс]. - Электронные данные. - Режим доступа: <https://lifehacker.ru/11-rss-readers/>
2. АКТУАЛЬНОСТЬ ИСПОЛЬЗОВАНИЯ МИКРОСЕРВИСОВ ПРИ РАЗРАБОТКЕ ИНФОРМАЦИОННЫХ СИСТЕМ [электронный ресурс]. - Электронные данные. - Режим доступа: <https://cyberleninka.ru/article/n/aktualnost-ispolzovaniya-mikroservisov-pri-razrabotke-informatsionnyh-sistem/viewer>
3. Project Tye [электронный ресурс]. - Электронные данные. - Режим доступа: <https://github.com/dotnet/tye>
4. .NET documentation | Microsoft Docs [электронный ресурс]. - Электронные данные. - Режим доступа: <https://docs.microsoft.com/en-us/dotnet>

Приложение А. Исходный код программы

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Threading.Tasks;
using AutoMapper;
using Common.Models;
using Downloader.Common.Contracts;
using Downloader.Common.Models;
using Downloader.Common.Models.RequestModels;
using Microsoft.AspNetCore.Mvc;

namespace Downloader.API.Controllers
{
    [ApiController]
    [Produces("application/json")]
    [Route("api/rss-sources")]
    public sealed class DownloaderController : ControllerBase
    {
        private readonly IDownloaderProvider _downloaderProvider;
        private readonly IDownloaderManager _downloaderManager;
        private readonly IMapper _mapper;

        public DownloaderController(
            IDownloaderProvider downloaderProvider,
            IDownloaderManager downloaderManager,
            IMapper mapper)
        {
            _downloaderProvider = downloaderProvider;
            _downloaderManager = downloaderManager;
            _mapper = mapper;
        }

        [HttpGet]
        [Route("{rssSourceGuid:guid}")]
        [ProducesResponseType(typeof(RssSourceReadModel), (int)HttpStatusCode.OK)]
        public async Task<IActionResult> GetRssSourceAsync([FromRoute] Guid
            rssSourceGuid)
        {
            var sources = await
                _downloaderProvider.GetAsync(rssSourceGuid).ConfigureAwait(false);
            return Ok(sources);
        }

        [HttpGet]
        [Route("")]
        [ProducesResponseType(typeof(IEnumerable<RssSourceReadModel>),
            (int)HttpStatusCode.OK)]
        public async Task<IActionResult> GetRssSourcesAsync()
        {
            var sources = await _downloaderProvider.GetAsync().ConfigureAwait(false);
            return Ok(sources);
        }

        [HttpGet]
        [Route("ensure-valid")]
        [ProducesResponseType((int)HttpStatusCode.OK)]
        [ProducesResponseType(typeof(ErrorResponse), (int)HttpStatusCode.BadRequest)]
        public async Task<IActionResult> EnsureRssSourceValidAsync([FromQuery] string
            rssSourceUrl)
        {

```

```

        await
_downloaderProvider.EnsureRssSourceIsValidAsync(rssSourceUrl).ConfigureAwait(false);
        return Ok();
    }

    [HttpPost]
    [Route("")]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    [ProducesResponseType(typeof(ErrorResponse), (int)HttpStatusCode.BadRequest)]
    public async Task<IActionResult> CreateAsync([FromBody] RssSourceManageModel
rssSource)
    {
        await _downloaderManager.CreateAsync(rssSource).ConfigureAwait(false);
        return Ok();
    }

    [HttpPut]
    [Route("{rssSourceGuid:guid}")]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    [ProducesResponseType(typeof(ErrorResponse), (int)HttpStatusCode.BadRequest)]
    public async Task<IActionResult> UpdateAsync([FromRoute] Guid rssSourceGuid,
[FromBody] RssSourceRequestModel rssSourceRequestModel)
    {
        var rssSource = _mapper.Map(rssSourceRequestModel, new RssSourceManageModel
{ Guid = rssSourceGuid });
        await _downloaderManager.UpdateAsync(rssSource).ConfigureAwait(false);
        return Ok();
    }

    [HttpDelete]
    [Route("{rssSourceGuid:guid}")]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    [ProducesResponseType(typeof(ErrorResponse), (int)HttpStatusCode.BadRequest)]
    public async Task<IActionResult> DeleteAsync([FromRoute] Guid rssSourceGuid)
    {
        await _downloaderManager.DeleteAsync(rssSourceGuid).ConfigureAwait(false);
        return Ok();
    }

    [HttpPost]
    [Route("{rssSourceGuid:guid}/news")]
    [ProducesResponseType(typeof(ErrorResponse), (int)HttpStatusCode.BadRequest)]
    [ProducesResponseType(typeof(ErrorResponse),
(int)HttpStatusCode.InternalServerError)]
    [ProducesResponseType(typeof(IEnumerable<NewsItem>), (int)HttpStatusCode.OK)]
    public async Task<IActionResult> DownloadNewsAsync([FromRoute] Guid
rssSourceGuid)
    {
        var news = await
_downloaderManager.DownloadNewsAsync(rssSourceGuid).ConfigureAwait(false);
        return Ok(news);
    }

    [HttpPost]
    [Route("news")]
    [ProducesResponseType(typeof(ErrorResponse),
(int)HttpStatusCode.InternalServerError)]
    [ProducesResponseType(typeof(IEnumerable<NewsItem>), (int)HttpStatusCode.OK)]
    public async Task<IActionResult> DownloadAllNewsAsync()
    {
        var news = await
_downloaderManager.DownloadAllNewsAsync().ConfigureAwait(false);

```

```

        return Ok(news);
    }
}

using Downloader.API.Models;
using Microsoft.EntityFrameworkCore;

namespace Downloader.API.Database
{
    public class DownloaderDbContext : DbContext
    {
        public DbSet<RssSource> RssSources { get; set; }

        public DownloaderDbContext(DbContextOptions options) : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<RssSource>(builder =>
            {
                builder.ToTable("RssSources");
                builder.HasKey(s => s.Guid).HasName("PK_RssSource");

                builder.Property(s =>
                    s.Guid).HasColumnName("Guid").IsRequired().ValueGeneratedNever();
                builder.Property(s =>
                    s.Url).HasColumnName("Url").HasColumnType("varchar(1000)").HasMaxLength(1000).IsRequired();
                builder.Property(s =>
                    s.LastSuccessDownloading).HasColumnName("LastSuccessDownloading").HasColumnType("timestamp").IsRequired();
            });
            base.OnModelCreating(modelBuilder);
        }
    }
}

using System.Collections.Generic;
using System.Threading.Tasks;
using Downloader.API.Models;

namespace Downloader.API.ExternalServices
{
    public interface IRssExternalService
    {
        Task EnsureCorrectRssSourceAsync(string rssSourceUrl);

        Task<IEnumerable<RssSourceResponseItem>> RequestRssSourceAsync(string
            rssSourceUrl);
    }
}

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
using System.Xml.Linq;
using Common.Exceptions;

```

```

using Common.Extensions;
using Downloader.API.ExternalServices;
using Downloader.API.Models;
using Downloader.API.Resources;
using Microsoft.Extensions.Logging;

namespace Downloader.API.ExternalRepositories
{
    public sealed class RssExternalService : IRssExternalService
    {
        private const string RssRootTag = "rss";
        private const string RssRootTagVersionAttribute = "version";
        private const string RssDescriptionElement = "description";
        private const string RssTitleElement = "title";
        private const string RssPublishDateElement = "pubDate";
        private const int RequiredRssVersion = 2;

        private readonly HttpClient _httpClient;
        private readonly ILogger<RssExternalService> _logger;

        public RssExternalService(HttpClient httpClient, ILogger<RssExternalService>
logger)
        {
            _httpClient = httpClient;
            _logger = logger;
        }

        public async Task EnsureCorrectRssSourceAsync(string rssSourceUrl)
        {
            var (response, stringResponse) = await
RequestAsync(rssSourceUrl).ConfigureAwait(false);

            if (!response.IsSuccessStatusCode)
            {
                _logger.LogError(Localization.RssSourceRequestFailed, rssSourceUrl,
response.StatusCode, response.ReasonPhrase, stringResponse);
                throw new BadRequestException(Localization.NotARssSource);
            }

            var xmlDocument = XDocument.Parse(stringResponse);
            if (!IsXmlIsCorrectRssDocument(xmlDocument))
            {
                throw new BadRequestException(Localization.NotARssSource);
            }
        }

        public async Task<IEnumerable<RssSourceResponseItem>>
RequestRssSourceAsync(string rssSourceUrl)
        {
            var (response, stringResponse) = await
RequestAsync(rssSourceUrl).ConfigureAwait(false);

            if (!response.IsSuccessStatusCode)
            {
                _logger.LogError(Localization.RssSourceRequestFailed, rssSourceUrl,
response.StatusCode, response.ReasonPhrase, stringResponse);
                throw new ServerInnerException(Localization.RssSourceNotAvailable);
            }

            var xmlDocument = XDocument.Parse(stringResponse);
            if (!IsXmlIsCorrectRssDocument(xmlDocument))
            {

```

```

        throw new BadRequestException(Localization.NotARssSource);
    }

    return xmlDocument.Root?.Element("channel")?.Elements("item").Select(i =>
new RssSourceResponseItem
    {
        Description = i.Element(RssDescriptionElement)?.Value ??
string.Empty,
        Title = i.Element(RssTitleElement)?.Value ?? string.Empty,
        PublishDate =
DateTime.TryParse(i.Element(RssPublishDateElement)?.Value, out var date)
            ? date
            : DateTime.MinValue
    })
    ?? Array.Empty<RssSourceResponseItem>();
}

private static bool IsXmlIsCorrectRssDocument(XDocument xmlDocument)
{
    var rootElement = xmlDocument?.Root;
    if (rootElement == null
        || !RssRootTag.Equals(rootElement.Name.LocalName)
        || !rootElement.HasAttributes)
    {
        return false;
    }

    var versionAttribute = rootElement.Attributes()
        .SingleOrDefault(a =>
RssRootTagVersionAttribute.Equals(a.Name.LocalName));
    if (versionAttribute == null
        || !double.TryParse(versionAttribute.Value, NumberStyles.Float,
CultureInfo.InvariantCulture, out var version)
        || !version.ApproximatelyEquals(RequiredRssVersion))
    {
        return false;
    }

    return true;
}

private async Task<(HttpResponseMessage response, string stringResponse)>
RequestAsync(string url)
{
    var response = await _httpClient.GetAsync(url).ConfigureAwait(false);
    var stringResponse = await
response.Content.ReadAsStringAsync().ConfigureAwait(false);

    _logger.LogTrace("{0} responded with body:\n{1}", url, stringResponse);

    return (response, stringResponse);
}
}

using System;
using Microsoft.EntityFrameworkCore.Migrations;

namespace Downloader.API.Migrations
{
    public partial class InitialCreate : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)

```



```

        {
            migrationBuilder.CreateTable(
                name: "RssSources",
                columns: table => new
                {
                    Guid = table.Column<Guid>(type: "uuid", nullable: false),
                    Url = table.Column<string>(type: "varchar(1000)", maxLength: 1000,
nullable: false),
                    LastSuccessDownloading = table.Column<DateTime>(type: "timestamp",
nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_RssSource", x => x.Guid);
                });
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "RssSources");
        }
    }
}

using System;
using Db.Common.Models;

namespace Downloader.API.Models
{
    public sealed class RssSource : IDbModel
    {
        public Guid Guid { get; set; }

        public string Url { get; set; }

        public DateTime LastSuccessDownloading { get; set; }
    }
}

using System;

namespace Downloader.API.Models
{
    public sealed class RssSourceResponseItem
    {
        public string Title { get; set; }

        public DateTime PublishDate { get; set; }

        public string Description { get; set; }
    }
}

using Db.Common.Repositories.Contracts;
using Downloader.API.Models;

namespace Downloader.API.Repositories
{
    public interface IRssSourceRepository : IBaseRepository<RssSource>
    {
    }
}

```

```

}

using Db.Common.Repositories;
using Downloader.API.Database;
using Downloader.API.Models;

namespace Downloader.API.Repositories
{
    public sealed class RssSourceRepository : BaseRepository<RssSource>,
        IRssSourceRepository
    {
        public RssSourceRepository(DownloaderDbContext downloaderDbContext)
            : base(downloaderDbContext, downloaderDbContext.RssSources)
        { }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using Common.Exceptions;
using Downloader.API.ExternalRepositories;
using Downloader.API.ExternalServices;
using Downloader.API.Models;
using Downloader.API.Repositories;
using Downloader.API.Resources;
using Downloader.Common.Contracts;
using Downloader.Common.Models;

namespace Downloader.API.Services
{
    public class DownloaderManager : IDownloaderManager
    {
        private readonly IRssSourceRepository _rssSourceRepository;
        private readonly IMapper _mapper;
        private readonly IRssExternalService _rssExternalService;

        public DownloaderManager(
            IRssSourceRepository rssSourceRepository,
            IMapper mapper,
            IRssExternalService rssExternalService)
        {
            _rssSourceRepository = rssSourceRepository;
            _mapper = mapper;
            _rssExternalService = rssExternalService;
        }

        public async Task CreateAsync(RssSourceManageModel rssSource)
        {
            if (await
                _rssSourceRepository.IsExistsAsync(rssSource.Guid).ConfigureAwait(false))
            {
                throw new AlreadyExistsException(Localization.RssSourceAlreadyExists);
            }

            await
                _rssExternalService.EnsureCorrectRssSourceAsync(rssSource.Url).ConfigureAwait(false)
            ;

            var model = _mapper.Map<RssSource>(rssSource);

```

```

        model.LastSuccessDownloading = DateTime.MinValue;

        await _rssSourceRepository.CreateAsync(model).ConfigureAwait(false);
    }

    public async Task UpdateAsync(RssSourceManageModel rssSource)
    {
        if (!await
            _rssSourceRepository.IsExistsAsync(rssSource.Guid).ConfigureAwait(false))
        {
            throw new NotFoundException(Localization.RssSourceNotFound);
        }

        await
            _rssExternalService.EnsureCorrectRssSourceAsync(rssSource.Url).ConfigureAwait(false);

        var model = await
            _rssSourceRepository.GetAsync(rssSource.Guid).ConfigureAwait(false);
        model = _mapper.Map(rssSource, model);

        await _rssSourceRepository.UpdateAsync(model).ConfigureAwait(false);
    }

    public async Task DeleteAsync(Guid rssSourceGuid)
    {
        if (!await
            _rssSourceRepository.IsExistsAsync(rssSourceGuid).ConfigureAwait(false))
        {
            throw new NotFoundException(Localization.RssSourceNotFound);
        }

        await
            _rssSourceRepository.DeleteAsync(rssSourceGuid).ConfigureAwait(false);
    }

    public async Task<IEnumerable<NewsItem>> DownloadNewsAsync(Guid rssSourceGuid)
    {
        if (!await
            _rssSourceRepository.IsExistsAsync(rssSourceGuid).ConfigureAwait(false))
        {
            throw new NotFoundException(Localization.RssSourceNotFound);
        }

        var rssSource = await
            _rssSourceRepository.GetAsync(rssSourceGuid).ConfigureAwait(false);

        var news = await
            _rssExternalService.RequestRssSourceAsync(rssSource.Url).ConfigureAwait(false);
        var result = news.Select(_mapper.Map<NewsItem>);

        rssSource.LastSuccessDownloading = DateTime.Today;
        await _rssSourceRepository.UpdateAsync(rssSource).ConfigureAwait(false);

        return result;
    }

    public async Task<IEnumerable<NewsItem>> DownloadAllNewsAsync()
    {
        var rssSources = await
            _rssSourceRepository.GetAsync().ConfigureAwait(false);
    }

```

```

        var news = new List<RssSourceResponseItem>();
        foreach (var rssSource in rssSources)
        {
            news.AddRange(await
                _rssExternalService.RequestRssSourceAsync(rssSource.Url).ConfigureAwait(false));
            rssSource.LastSuccessDownloading = DateTime.Today;
            await
                _rssSourceRepository.UpdateAsync(rssSource).ConfigureAwait(false);
        }

        return news.Select(_mapper.Map<NewsItem>);
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using Downloader.API.ExternalRepositories;
using Downloader.API.ExternalServices;
using Downloader.API.Repositories;
using Downloader.Common.Contracts;
using Downloader.Common.Models;

namespace Downloader.API.Services
{
    public sealed class DownloaderProvider : IDownloaderProvider
    {
        private readonly IRssSourceRepository _rssSourceRepository;
        private readonly IMapper _mapper;
        private readonly IRssExternalService _rssExternalService;

        public DownloaderProvider(IRssSourceRepository rssSourceRepository, IMapper mapper, IRssExternalService rssExternalService)
        {
            _rssSourceRepository = rssSourceRepository;
            _mapper = mapper;
            _rssExternalService = rssExternalService;
        }

        public async Task<RssSourceReadModel> GetAsync(Guid guid)
        {
            var rssSource = await
                _rssSourceRepository.GetAsync(guid).ConfigureAwait(false);
            return _mapper.Map<RssSourceReadModel>(rssSource);
        }

        public async Task<IEnumerable<RssSourceReadModel>> GetAsync()
        {
            var rssSources = await
                _rssSourceRepository.GetAsync().ConfigureAwait(false);
            return rssSources.Select(_mapper.Map<RssSourceReadModel>);
        }

        public async Task EnsureRssSourceIsValidAsync(string rssSourceUrl)
        {
            await
                _rssExternalService.EnsureCorrectRssSourceAsync(rssSourceUrl).ConfigureAwait(false);
        }
    }
}

```

```

}

using AutoMapper;
using Common.Extensions;
using Downloader.API.Models;
using Downloader.Common.Models;
using Downloader.Common.Models.RequestModels;

namespace Downloader.API
{
    public class MapperProfile : Profile
    {
        public MapperProfile()
        {
            CreateMap<RssSourceRequestModel, RssSourceManageModel>()
                .ForMember(d => d.Url, o => o.MapFrom(s => s.Url))
                .ForAllOtherMembers(o => o.Ignore());

            CreateMap<RssSourceManageModel, RssSource>()
                .ForMember(d => d.Guid, o => o.MapFrom(s => s.Guid))
                .ForMember(d => d.Url, o => o.MapFrom(s => s.Url))
                .ForAllOtherMembers(o => o.Ignore());

            CreateMap<RssSource, RssSourceReadModel>();

            CreateMap<RssSourceResponseItem, NewsItem>();

            this.AssertConfigurationIsValid();
        }
    }
}

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;

namespace Downloader.API
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        private static IHostBuilder CreateHostBuilder(string[] args) =>
            Host
                .CreateDefaultBuilder(args)
                .ConfigureAppConfiguration((_, config) =>
                {
                    config.AddEnvironmentVariables();
                })
                .ConfigureWebHostDefaults(webBuilder => {
                    webBuilder.UseStartup<Startup>(); });
    }
}

using Api.Common;
using Downloader.API.Database;
using Downloader.API.ExternalRepositories;
using Downloader.API.ExternalServices;
using Downloader.API.Repositories;

```

```

using Downloader.API.Services;
using Downloader.Common.Contracts;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace Downloader.API
{
    public sealed class Startup : BaseStartup
    {
        public Startup(IConfiguration configuration) : base(configuration) {}

        protected override IServiceCollection RegisterServices(IServiceCollection services)
        {
            services.AddDbContext<DownloaderDbContext>(options =>
                options.UseNpgsql(Configuration["ConnectionString"]));

            services.AddScoped<IRssSourceRepository, RssSourceRepository>();
            services.AddScoped<IRssExternalService, RssExternalService>();
            services.AddScoped<IDownloaderManager, DownloaderManager>();
            services.AddScoped<IDownloaderProvider, DownloaderProvider>();

            return services;
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env,
            DownloaderDbContext dbContext)
        {
            BaseConfigure(app, env);

            dbContext.Database.Migrate();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Downloader.Common.Models;

namespace Downloader.Common.Contracts
{
    public interface IDownloaderManager
    {
        Task CreateAsync(RssSourceManageModel rssSource);

        Task UpdateAsync(RssSourceManageModel rssSource);

        Task DeleteAsync(Guid rssSourceGuid);

        Task<IEnumerable<NewsItem>> DownloadNewsAsync(Guid rssSourceGuid);

        Task<IEnumerable<NewsItem>> DownloadAllNewsAsync();
    }
}

using System;
using System.Collections.Generic;
using System.Threading.Tasks;

```



```

using Downloader.Common.Models;

namespace Downloader.Common.Contracts
{
    public interface IDownloaderProvider
    {
        Task<RssSourceReadModel> GetAsync(Guid guid);

        Task<IEnumerable<RssSourceReadModel>> GetAsync();

        Task EnsureRssSourceIsValidAsync(string rssSourceUrl);
    }
}

namespace Downloader.Common.Models.RequestModels
{
    public sealed class RssSourceRequestModel
    {
        public string Url { get; set; }
    }
}

using System;

namespace Downloader.Common.Models
{
    public class BaseRssSourceModel
    {
        public Guid Guid { get; set; }

        public string Url { get; set; }
    }
}

using System;

namespace Downloader.Common.Models
{
    public sealed class NewsItem
    {
        public string Title { get; set; }

        public DateTime PublishDate { get; set; }

        public string Description { get; set; }
    }
}

namespace Downloader.Common.Models
{
    public sealed class RssSourceManageModel : BaseRssSourceModel
    {
    }
}

using System;

namespace Downloader.Common.Models
{
    public sealed class RssSourceReadModel : BaseRssSourceModel
    {
        public DateTime LastSuccessDownloading { get; set; }
    }
}

```

```

    }
}

using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading.Tasks;
using Common.Extensions;
using Downloader.Common.Contracts;
using Downloader.Common.Models;

namespace Downloader.Facade.HttpProxy
{
    public class DownloadManagerProxy : IDownloadManager
    {
        private readonly HttpClient _httpClient;

        public DownloadManagerProxy(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        public async Task CreateAsync(RssSourceManageModel rssSource)
        {
            await _httpClient.InternalPost("/api/rss-sources",
            rssSource).ConfigureAwait(false);
        }

        public async Task UpdateAsync(RssSourceManageModel rssSource)
        {
            await _httpClient.InternalPut($" /api/rss-sources/{rssSource.Guid:D}",
            rssSource).ConfigureAwait(false);
        }

        public async Task DeleteAsync(Guid rssSourceGuid)
        {
            await _httpClient.InternalDelete($" /api/rss-
            sources/{rssSourceGuid:D}").ConfigureAwait(false);
        }

        public async Task<IEnumerable<NewsItem>> DownloadNewsAsync(Guid rssSourceGuid)
        {
            return await _httpClient.InternalPost<IEnumerable<NewsItem>>($" /api/rss-
            sources/{rssSourceGuid:D}/news").ConfigureAwait(false);
        }

        public async Task<IEnumerable<NewsItem>> DownloadAllNewsAsync()
        {
            return await _httpClient.InternalPost<IEnumerable<NewsItem>>("/api/rss-
            sources/news").ConfigureAwait(false);
        }
    }
}

using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading.Tasks;
using Common.Extensions;
using Downloader.Common.Contracts;
using Downloader.Common.Models;

```

```

namespace Downloader.Facade.HttpProxy
{
    public class DownloadProviderProxy : IDownloaderProvider
    {
        private readonly HttpClient _httpClient;

        public DownloadProviderProxy(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        public async Task<RssSourceReadModel> GetAsync(Guid guid)
        {
            return await _httpClient.InternalGet<RssSourceReadModel>($"/api/rss-sources/{guid:D}").ConfigureAwait(false);
        }

        public async Task<IEnumerable<RssSourceReadModel>> GetAsync()
        {
            return await
                _httpClient.InternalGet<IEnumerable<RssSourceReadModel>>("/api/rss-sources").ConfigureAwait(false);
        }

        public async Task EnsureRssSourceIsValidAsync(string rssSourceUrl)
        {
            //ToDo: rssSourceUrl should be encoded because also may have query params
            //which may be interpreted as main uri params
            await _httpClient.InternalGet($"/api/rss-sources/ensure-valid?rssSourceUrl={rssSourceUrl}").ConfigureAwait(false);
        }
    }
}

using System;
using System.Net.Http;
using Downloader.Common.Contracts;
using Downloader.Facade.HttpProxy;
using Microsoft.Extensions.DependencyInjection;

namespace Downloader.Facade
{
    public static class ServiceCollectionExtensions
    {
        public static IServiceCollection AddDownloaderProxies(this IServiceCollection services, string baseUrl)
        {
            services.AddScoped<IDownloaderProvider>(s =>
            {
                var httpClient =
                    s.GetRequiredService<IHttpClientFactory>().CreateClient();
                httpClient.BaseAddress = new (baseUrl, UriKind.Absolute);
                return new DownloadProviderProxy(httpClient);
            });

            services.AddScoped<IDownloaderManager>(s =>
            {
                var httpClient =
                    s.GetRequiredService<IHttpClientFactory>().CreateClient();
                httpClient.BaseAddress = new (baseUrl, UriKind.Absolute);
                return new DownloadManagerProxy(httpClient);
            });
        }
    }
}

```

```

        return services;
    }
}

using System;
using System.Collections.Generic;
using System.Net;
using System.Threading.Tasks;
using Downloader.Common.Models;
using Filter.Common.Contracts;
using Filter.Common.Models;
using Microsoft.AspNetCore.Mvc;

namespace Filter.API.Controllers
{
    [ApiController]
    [Produces("application/json")]
    [Route("api/filter")]
    public sealed class FilterController : ControllerBase
    {
        private readonly IFilterProvider _filterProvider;
        private readonly IFilterManager _filterManager;

        public FilterController(IFilterProvider filterProvider, IFilterManager filterManager)
        {
            _filterProvider = filterProvider;
            _filterManager = filterManager;
        }

        [HttpGet]
        [Route("{filterGuid:guid}")]
        [ProducesResponseType(typeof(NewsFilterModel), (int)HttpStatusCode.OK)]
        public async Task<IActionResult> GetAsync([FromRoute] Guid filterGuid)
        {
            var filter = await
            _filterProvider.GetAsync(filterGuid).ConfigureAwait(false);
            return Ok(filter);
        }

        [HttpGet]
        [Route("")]
        [ProducesResponseType(typeof(IEnumerable<NewsFilterModel>), (int)HttpStatusCode.OK)]
        public async Task<IActionResult> GetAsync()
        {
            var filters = await _filterProvider.GetAsync().ConfigureAwait(false);
            return Ok(filters);
        }

        [HttpPost]
        [Route("{filterGuid:guid}/filter-news")]
        [ProducesResponseType(typeof(IEnumerable<NewsItem>), (int)HttpStatusCode.OK)]
        public async Task<IActionResult> FilterNewsAsync([FromRoute] Guid filterGuid,
        [FromBody] IEnumerable<NewsItem> news)
        {
            var filteredNews = await _filterProvider.FilterNewsAsync(filterGuid,
            news).ConfigureAwait(false);
            return Ok(filteredNews);
        }
    }
}

```

```

        [HttpPost]
        [Route("ensure-valid")]
        [ProducesResponseType((int)HttpStatusCode.OK)]
        public async Task<IActionResult> EnsureFiltersValidAsync([FromBody]
IEnumerable<string> filters)
        {
            await
            _filterProvider.EnsureFiltersIsValidAsync(filters).ConfigureAwait(false);
            return Ok();
        }

        [HttpPost]
        [Route("")]
        [ProducesResponseType((int)HttpStatusCode.OK)]
        public async Task<IActionResult> CreateAsync([FromBody] NewsFilterModel filter)
        {
            await _filterManager.CreateAsync(filter);
            return Ok();
        }

        [HttpPut]
        [Route("")]
        [ProducesResponseType((int)HttpStatusCode.OK)]
        public async Task<IActionResult> UpdateAsync([FromBody] NewsFilterModel filter)
        {
            await _filterManager.UpdateAsync(filter);
            return Ok();
        }

        [HttpDelete]
        [Route("{filterGuid:guid}")]
        [ProducesResponseType((int)HttpStatusCode.OK)]
        public async Task<IActionResult> DeleteAsync([FromRoute] Guid filterGuid)
        {
            await _filterManager.DeleteAsync(filterGuid);
            return Ok();
        }
    }
}

using Filter.API.Models;
using Microsoft.EntityFrameworkCore;

namespace Filter.API.Database
{
    public sealed class FilterDbContext : DbContext
    {
        public DbSet<FilterModel> Filters { get; set; }

        public FilterDbContext(DbContextOptions options) : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<FilterModel>(builder =>
            {
                builder.ToTable("Filters");
                builder.HasKey(s => s.Guid).HasName("PK_Filter");
                builder.HasIndex(s =>
s.GroupGuid).IsUnique(false).HasDatabaseName("IDX_Filter_GroupGuid");
            });
        }
    }
}

```

```

        builder.Property(s =>
s.Guid).HasColumnName("Guid").IsRequired().ValueGeneratedOnAdd();
        builder.Property(s =>
s.GroupGuid).HasColumnName("GroupGuid").IsRequired().ValueGeneratedNever();
        builder.Property(s =>
s.Filter).HasColumnName("Filter").IsRequired().HasColumnType("varchar(1000)").HasMaxLength(1000);
    });
}
}

using System;
using Microsoft.EntityFrameworkCore.Migrations;

namespace Filter.API.Migrations
{
    public partial class Init : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Filters",
                columns: table => new
                {
                    Guid = table.Column<Guid>(type: "uuid", nullable: false),
                    GroupGuid = table.Column<Guid>(type: "uuid", nullable: false),
                    Filter = table.Column<string>(type: "varchar(1000)", maxLength:
1000, nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Filter", x => x.Guid);
                });

            migrationBuilder.CreateIndex(
                name: "IDX_Filter_GroupGuid",
                table: "Filters",
                column: "GroupGuid");
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Filters");
        }
    }
}

using System;
using System.Collections.Generic;
using Db.Common.Models;

namespace Filter.API.Models
{
    public sealed class FilterModel : IDbCollectionModel
    {
        public Guid Guid { get; set; }

        public Guid GroupGuid { get; set; }
    }
}

```

```

        public string Filter { get; set; }
    }
}

using Db.Common.Repositories;
using Filter.API.Database;
using Filter.API.Models;

namespace Filter.API.Repositories
{
    public sealed class FilterRepository : BaseCollectionRepository<FilterModel>,
        IFilterRepository
    {
        public FilterRepository(FilterDbContext dbContext) : base(dbContext,
            dbContext.Filters)
        {
        }
    }
}

using Db.Common.Repositories.Contracts;
using Filter.API.Models;

namespace Filter.API.Repositories
{
    public interface IFilterRepository : IBaseCollectionRepository<FilterModel> { }
}

using System;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using Common.Exceptions;
using Filter.API.Models;
using Filter.API.Repositories;
using Filter.API.Resources;
using Filter.Common.Contracts;
using Filter.Common.Models;

namespace Filter.API.Services
{
    public sealed class FilterManager : IFilterManager
    {
        private readonly IMapper _mapper;
        private readonly IFilterRepository _filterRepository;
        private readonly IFilterProvider _filterProvider;

        public FilterManager(
            IMapper mapper,
            IFilterRepository filterRepository,
            IFilterProvider filterProvider)
        {
            _mapper = mapper;
            _filterRepository = filterRepository;
            _filterProvider = filterProvider;
        }

        public async Task CreateAsync(NewsFilterModel filter)
        {
            if (await _filterRepository.IsExistsAsync(f => f.GroupGuid ==
                filter.Guid).ConfigureAwait(false))
            {

```

```

        throw new AlreadyExistsException(Localization.FilterAlreadyExists);
    }

    await
    _filterProvider.EnsureFiltersIsValidAsync(filter.Filters).ConfigureAwait(false);

    var models = filter.Filters.Select(f =>
    {
        var model = _mapper.Map<FilterModel>(filter);
        model.Filter = f;
        return model;
    });

    await _filterRepository.CreateAsync(models).ConfigureAwait(false);
}

public async Task UpdateAsync(NewsFilterModel filter)
{
    await
    _filterProvider.EnsureFiltersIsValidAsync(filter.Filters).ConfigureAwait(false);

    await
    _filterRepository.DeleteGroupAsync(filter.Guid).ConfigureAwait(false);

    await CreateAsync(filter).ConfigureAwait(false);
}

public async Task DeleteAsync(Guid filterGuid)
{
    if (!await _filterRepository.IsExistsAsync(f => f.GroupGuid ==
filterGuid).ConfigureAwait(false))
    {
        throw new NotFoundException(Localization.FilterNotFound);
    }

    await _filterRepository.DeleteGroupAsync(filterGuid).ConfigureAwait(false);
}
}

}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using AutoMapper;
using Common.Exceptions;
using Downloader.Common.Models;
using Filter.API.Repositories;
using Filter.API.Resources;
using Filter.Common.Contracts;
using Filter.Common.Models;

namespace Filter.API.Services
{
    public sealed class FilterProvider : IFilterProvider
    {
        private readonly IMapper _mapper;
        private readonly IFilterRepository _filterRepository;

        public FilterProvider(IMapper mapper, IFilterRepository filterRepository)
        {

```



```

        _mapper = mapper;
        _filterRepository = filterRepository;
    }

    public async Task<NewsFilterModel> GetAsync(Guid filterGuid)
    {
        var models = await _filterRepository.GetGroupAsync(filterGuid);
        var filters = _mapper.Map<NewsFilterModel>(models);
        filters.Guid = filterGuid;
        return filters;
    }

    public async Task<IEnumerable<NewsFilterModel>> GetAsync()
    {
        var models = await _filterRepository.GetAsync();
        return models
            .GroupBy(m => m.GroupGuid)
            .Select(m => _mapper.Map<NewsFilterModel>(m));
    }

    public async Task<IEnumerable<NewsItem>> FilterNewsAsync(Guid filterGuid,
        IEnumerable<NewsItem> news)
    {
        var models = await _filterRepository.GetGroupAsync(filterGuid);

        if (models.Any())
        {
            news = news.Where(n =>
                models.Any(m =>
                {
                    var regex = new Regex(m.Filter);
                    return regex.IsMatch(n.Description) || regex.IsMatch(n.Title);
                }
            ));
        }

        return news;
    }

    public Task EnsureFiltersIsValidAsync(IEnumerable<string> filters)
    {
        foreach (var filter in filters)
        {
            try
            {
                new Regex(filter);
            }
            catch (RegexParseException e)
            {
                return Task.FromException(new
                    BadRequestException(string.Format(Localization.NotAFilter, filter), e));
            }
        }

        return Task.CompletedTask;
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using AutoMapper;

```

```

using Common.Extensions;
using Filter.API.Models;
using Filter.Common.Models;

namespace Filter.API
{
    public class MapperProfile : Profile
    {
        public MapperProfile()
        {
            CreateMap<NewsFilterModel, FilterModel>()
                .ForMember(d => d.Guid, o => o.MapFrom(s => Guid.NewGuid()))
                .ForMember(d => d.GroupGuid, o => o.MapFrom(s => s.Guid))
                .ForMember(d => d.Filter, o => o.Ignore());

            CreateMap<IEnumerable<FilterModel>, NewsFilterModel>()
                .ForMember(d => d.Guid, o => o.MapFrom((s, _, _) =>
                    s.FirstOrDefault()?.GroupGuid ?? Guid.Empty))
                .ForMember(d => d.Filters, o => o.MapFrom(s => s.Select(f =>
                    f.Filter)));

            this.AssertConfigurationIsValid();
        }
    }
}

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;

namespace Filter.API
{
    public static class Program
    {
        {
            public static void Main(string[] args)
            {
                CreateHostBuilder(args).Build().Run();
            }

            private static IHostBuilder CreateHostBuilder(string[] args) =>
                Host.CreateDefaultBuilder(args)
                    .ConfigureAppConfiguration((_, config) =>
                        {
                            config.AddEnvironmentVariables();
                        })
                    .ConfigureWebHostDefaults(webBuilder => {
                        webBuilder.UseStartup<Startup>(); });
        }
    }

    using Api.Common;
    using Filter.API.Database;
    using Filter.API.Repositories;
    using Filter.API.Services;
    using Filter.Common.Contracts;
    using Microsoft.AspNetCore.Builder;
    using Microsoft.AspNetCore.Hosting;
    using Microsoft.EntityFrameworkCore;
    using Microsoft.Extensions.Configuration;
    using Microsoft.Extensions.DependencyInjection;

    namespace Filter.API

```

```

{
    public sealed class Startup : BaseStartup
    {
        public Startup(IConfiguration configuration) : base(configuration) {}

        protected override IServiceCollection RegisterServices(IServiceCollection
services)
        {
            services.AddDbContext<FilterDbContext>(options =>
                options.UseNpgsql(Configuration["ConnectionString"]));

            services.AddScoped<IFilterRepository, FilterRepository>();
            services.AddScoped<IFilterProvider, FilterProvider>();
            services.AddScoped<IFilterManager, FilterManager>();

            return services;
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env,
FilterDbContext dbContext)
        {
            BaseConfigure(app, env);

            dbContext.Database.Migrate();
        }
    }
}

using System;
using System.Threading.Tasks;
using Filter.Common.Models;

namespace Filter.Common.Contracts
{
    public interface IFilterManager
    {
        Task CreateAsync(NewsFilterModel filter);

        Task UpdateAsync(NewsFilterModel filter);

        Task DeleteAsync(Guid filterGuid);
    }
}

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Downloader.Common.Models;
using Filter.Common.Models;

namespace Filter.Common.Contracts
{
    public interface IFilterProvider
    {
        Task<NewsFilterModel> GetAsync(Guid filterGuid);

        Task<IEnumerable<NewsFilterModel>> GetAsync();

        Task<IEnumerable<NewsItem>> FilterNewsAsync(Guid filterGuid,
IEnumerable<NewsItem> news);

        Task EnsureFiltersIsValidAsync(IEnumerable<string> filters);
    }
}

```

```

    }
}

using System;
using System.Collections.Generic;

namespace Filter.Common.Models
{
    public sealed class NewsFilterModel
    {
        public Guid Guid { get; set; }

        public ICollection<string> Filters { get; set; }
    }
}

using System;
using System.Net.Http;
using System.Threading.Tasks;
using Common.Extensions;
using Filter.Common.Contracts;
using Filter.Common.Models;

namespace Filter.Facade.HttpProxy
{
    public sealed class FilterManagerProxy : IFilterManager
    {
        private readonly HttpClient _httpClient;

        public FilterManagerProxy(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        public async Task CreateAsync(NewsFilterModel filter)
        {
            await _httpClient.InternalPost("/api/filter",
            filter).ConfigureAwait(false);
        }

        public async Task UpdateAsync(NewsFilterModel filter)
        {
            await _httpClient.InternalPut("/api/filter", filter).ConfigureAwait(false);
        }

        public async Task DeleteAsync(Guid filterGuid)
        {
            await
            _httpClient.InternalDelete($"{"/api/filter/{filterGuid:D}"}").ConfigureAwait(false);
        }
    }
}

using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading.Tasks;
using Common.Extensions;
using Downloader.Common.Models;
using Filter.Common.Contracts;
using Filter.Common.Models;

```

```

namespace Filter.Facade.HttpProxy
{
    public sealed class FilterProviderProxy : IFilterProvider
    {
        private readonly HttpClient _httpClient;

        public FilterProviderProxy(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        public async Task<NewsFilterModel> GetAsync(Guid filterGuid)
        {
            return await
            _httpClient.InternalGet<NewsFilterModel>($"/api/filter/{filterGuid:D}").ConfigureAwait(
            false);
        }

        public async Task<IEnumerable<NewsFilterModel>> GetAsync()
        {
            return await
            _httpClient.InternalGet<IEnumerable<NewsFilterModel>>("/api/filter").ConfigureAwait(
            false);
        }

        public async Task<IEnumerable<NewsItem>> FilterNewsAsync(Guid filterGuid,
        IEnumerable<NewsItem> news)
        {
            return await
            _httpClient.InternalPost<IEnumerable<NewsItem>>($"/api/filter/{filterGuid:D}/filter-
            news", news).ConfigureAwait(false);
        }

        public async Task EnsureFiltersIsValidAsync(IEnumerable<string> filters)
        {
            await _httpClient.InternalPost("/api/filter/ensure-valid",
            filters).ConfigureAwait(false);
        }
    }
}

using System;
using System.Net.Http;
using Filter.Common.Contracts;
using Filter.Facade.HttpProxy;
using Microsoft.Extensions.DependencyInjection;

namespace Filter.Facade
{
    public static class ServiceCollectionExtensions
    {
        {
            public static IServiceCollection AddFilterProxies(this IServiceCollection
            services, string baseUrl)
            {
                services.AddScoped<IFilterProvider>(s =>
                {
                    var httpClient =
                    s.GetRequiredService<IHttpClientFactory>().CreateClient();
                    httpClient.BaseAddress = new (baseUrl, UriKind.Absolute);
                    return new FilterProviderProxy(httpClient);
                });
            }
        }
    }
}

```

```

        services.AddScoped<IFilterManager>(s =>
        {
            var httpClient =
s.GetRequiredService<IHttpClientFactory>().CreateClient();
            httpClient.BaseAddress = new (baseUrl, UriKind.Absolute);
            return new FilterManagerProxy(httpClient);
        });

        return services;
    }
}

using System.Collections.Generic;
using System.Threading.Tasks;
using Downloader.Common.Models;
using Gateway.Common.Contracts;
using Microsoft.AspNetCore.Mvc;

namespace Gateway.Controllers
{
    [ApiController]
    [Produces("application/json")]
    [Route("api/news")]
    public class NewsController : ControllerBase
    {
        private readonly IRssServiceManager _rssServiceManager;

        public NewsController(IRssServiceManager rssServiceManager)
        {
            _rssServiceManager = rssServiceManager;
        }

        /// <summary>Get all news</summary>
        [HttpGet("")]
        public async Task<ICollection<NewsItem>> DownloadNewsAsync()
        {
            return await _rssServiceManager.DownloadNewsAsync().ConfigureAwait(false);
        }
    }
}

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Gateway.Common.Contracts;
using Gateway.Common.Models;
using Microsoft.AspNetCore.Mvc;

namespace Gateway.Controllers
{
    [ApiController]
    [Produces("application/json")]
    [Route("api/rss-subscription")]
    public class RssSubscriptionController : ControllerBase
    {
        private readonly IRssServiceProvider _rssServiceProvider;
        private readonly IRssServiceManager _rssServiceManager;

        public RssSubscriptionController(IRssServiceProvider rssServiceProvider,
            IRssServiceManager rssServiceManager)
        {

```

```

        _rssServiceProvider = rssServiceProvider;
        _rssServiceManager = rssServiceManager;
    }

    /// <summary>Get information about specific subscription</summary>
    /// <param name="guid">subscription GUID</param>
    [HttpGet("{guid:guid}")]
    public async Task<RssSubscription> GetRssSubscriptionsAsync([FromRoute] Guid
guid)
    {
        return await
        _rssServiceProvider.GetSubscriptionAsync(guid).ConfigureAwait(false);
    }

    /// <summary>Get information about all stored subscriptions</summary>
    [HttpGet("")]
    public async Task<ICollection<RssSubscription>> GetAllRssSubscriptionsAsync()
    {
        return await
        _rssServiceProvider.GetSubscriptionsAsync().ConfigureAwait(false);
    }

    /// <summary>Add new or update existing subscription on a rss source</summary>
    [HttpPost("")]
    public async Task<Guid> CreateOrUpdateSubscriptionAsync(RssSubscription
rssSubscription)
    {
        return await
        _rssServiceManager.CreateOrUpdateSubscriptionAsync(rssSubscription).ConfigureAwait(f
alse);
    }

    /// <summary>Delete existing subscription on a rss source</summary>
    /// <param name="rssSubscriptionGuid">subscription GUID</param>
    [HttpDelete("{rssSubscriptionGuid:guid}")]
    public async Task DeleteRssSubscriptionAsync([FromRoute] Guid
rssSubscriptionGuid)
    {
        await
        _rssServiceManager.DeleteSubscriptionAsync(rssSubscriptionGuid).ConfigureAwait(false
);
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using Downloader.Common.Contracts;
using Downloader.Common.Models;
using Filter.Common.Contracts;
using Filter.Common.Models;
using Gateway.Common.Contracts;
using Gateway.Common.Models;
using Manager.Common.Contracts;
using Manager.Common.Models;
using Sender.Common.Contracts;
using Sender.Common.Models;

namespace Gateway.Services

```

```

{
    public sealed class RssServiceManager : IRssServiceManager
    {
        private readonly IMapper _mapper;
        private readonly IDownloaderProvider _downloaderProvider;
        private readonly IDownloaderManager _downloaderManager;
        private readonly IFilterProvider _filterProvider;
        private readonly IFilterManager _filterManager;
        private readonly ISenderProvider _senderProvider;
        private readonly ISenderManager _senderManager;
        private readonly IManagerProvider _managerProvider;
        private readonly IManagerManager _managerManager;

        public RssServiceManager(
            IMapper mapper,
            IDownloaderProvider downloaderProvider,
            IDownloaderManager downloaderManager,
            IFilterProvider filterProvider,
            IFilterManager filterManager,
            ISenderProvider senderProvider,
            ISenderManager senderManager,
            IManagerProvider managerProvider,
            IManagerManager managerManager)
        {
            _mapper = mapper;
            _downloaderProvider = downloaderProvider;
            _downloaderManager = downloaderManager;
            _filterProvider = filterProvider;
            _filterManager = filterManager;
            _senderProvider = senderProvider;
            _senderManager = senderManager;
            _managerProvider = managerProvider;
            _managerManager = managerManager;
        }

        public async Task<Guid> CreateOrUpdateSubscriptionAsync(RssSubscription
            rssSubscription)
        {
            await
            EnsureSubscriptionInfoValidAsync(rssSubscription).ConfigureAwait(false);

            return rssSubscription.Guid == Guid.Empty
                ? await CreateSubscriptionAsync(rssSubscription).ConfigureAwait(false)
                : await UpdateSubscriptionAsync(rssSubscription).ConfigureAwait(false);
        }

        public async Task DeleteSubscriptionAsync(Guid guid)
        {
            await Task.WhenAll(
                _downloaderManager.DeleteAsync(guid),
                _filterManager.DeleteAsync(guid),
                _senderManager.DeleteAsync(guid),
                _managerManager.DeleteAsync(guid));
        }

        public async Task<ICollection<NewsItem>> DownloadNewsAsync()
        {
            var rssSources = await
            _downloaderProvider.GetAsync().ConfigureAwait(false);
            var guids = rssSources.Select(s => s.Guid);

            var allNews = new List<NewsItem>();

```



```

        foreach (var guid in guides)
        {
            var news = await
            _downloaderManager.DownloadNewsAsync(guid).ConfigureAwait(false);
            allNews.AddRange(await _filterProvider.FilterNewsAsync(guid,
            news).ConfigureAwait(false));
        }

        return allNews
            .OrderBy(n => n.PublishDate)
            .ThenBy(n => n.Title)
            .ToArray();
    }

    private async Task EnsureSubscriptionInfoValidAsync(RssSubscription
    rssSubscription)
    {
        var validationTasks = new List<Task>
        {

            _downloaderProvider.EnsureRssSourceIsValidAsync(rssSubscription.RssSource),
            _filterProvider.EnsureFiltersIsValidAsync(rssSubscription.Filters)
        };

        if (rssSubscription.NeedToSendEmails)
        {
            validationTasks.AddRange(new[]
            {

                _senderProvider.EnsureReceiversIsValidAsync(rssSubscription.Receivers),
                _managerProvider.EnsureJobPeriodicityIsValidAsync(rssSubscription.Periodicity)
            });
        }

        await Task.WhenAll(validationTasks).ConfigureAwait(false);
    }

    private async Task<Guid> CreateSubscriptionAsync(RssSubscription
    rssSubscription)
    {
        rssSubscription.Guid = Guid.NewGuid();

        var rssSource = _mapper.Map<RssSourceManageModel>(rssSubscription);
        await _downloaderManager.CreateAsync(rssSource).ConfigureAwait(false);

        var filter = _mapper.Map<NewsFilterModel>(rssSubscription);
        await _filterManager.CreateAsync(filter).ConfigureAwait(false);

        var receivers = _mapper.Map<ReceiversModel>(rssSubscription);
        await _senderManager.CreateAsync(receivers).ConfigureAwait(false);

        var job = _mapper.Map<JobModel>(rssSubscription);
        await _managerManager.CreateAsync(job).ConfigureAwait(false);

        return rssSource.Guid;
    }

    private async Task<Guid> UpdateSubscriptionAsync(RssSubscription
    rssSubscription)
    {
        var rssSource = _mapper.Map<RssSourceManageModel>(rssSubscription);

```

```

        await _downloaderManager.UpdateAsync(rssSource).ConfigureAwait(false);

        var filter = _mapper.Map<NewsFilterModel>(rssSubscription);
        await _filterManager.UpdateAsync(filter).ConfigureAwait(false);

        var receivers = _mapper.Map<ReceiversModel>(rssSubscription);
        await _senderManager.UpdateAsync(receivers).ConfigureAwait(false);

        var job = _mapper.Map<JobModel>(rssSubscription);
        await _managerManager.UpdateAsync(job).ConfigureAwait(false);

        return rssSource.Guid;
    }
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using Downloader.Common.Contracts;
using Filter.Common.Contracts;
using Gateway.Common.Contracts;
using Gateway.Common.Models;
using Manager.Common.Contracts;
using Microsoft.Extensions.Logging;
using Sender.Common.Contracts;

```

```
namespace Gateway.Services
```

```

{
    public sealed class RssServiceProvider : IRssServiceProvider
    {
        private readonly ILogger<RssServiceProvider> _logger;
        private readonly IMapper _mapper;
        private readonly IDownloaderProvider _downloaderProvider;
        private readonly IFilterProvider _filterProvider;
        private readonly ISenderProvider _senderProvider;
        private readonly IManagerProvider _managerProvider;

        public RssServiceProvider(
            ILogger<RssServiceProvider> logger,
            IMapper mapper,
            IDownloaderProvider downloaderProvider,
            IFilterProvider filterProvider,
            ISenderProvider senderProvider,
            IManagerProvider managerProvider)
        {
            _logger = logger;
            _mapper = mapper;
            _downloaderProvider = downloaderProvider;
            _filterProvider = filterProvider;
            _senderProvider = senderProvider;
            _managerProvider = managerProvider;
        }

        public async Task<RssSubscription> GetSubscriptionAsync(Guid guid)
        {
            // Better to have separate service with full data

            var rssSource = await
                _downloaderProvider.GetAsync(guid).ConfigureAwait(false);

```

```

        var model = _mapper.Map<RssSubscription>(rssSource);

        var filter = await _filterProvider.GetAsync(guid).ConfigureAwait(false);
        model = _mapper.Map(filter, model);

        var receivers = await _senderProvider.GetAsync(guid).ConfigureAwait(false);
        model = _mapper.Map(receivers, model);

        var job = await _managerProvider.GetAsync(guid).ConfigureAwait(false);
        model = _mapper.Map(job, model);

        return model;
    }

    public async Task<ICollection<RssSubscription>> GetSubscriptionsAsync()
    {
        var rssSources = (await
            _downloaderProvider.GetAsync().ConfigureAwait(false)).ToArray();
        var rssSourcesMap = rssSources.ToDictionary(s => s.Guid, s => s);

        var filters = (await
            _filterProvider.GetAsync().ConfigureAwait(false)).ToArray();
        var filterMap = filters.ToDictionary(s => s.Guid, s => s);

        var receivers = (await
            _senderProvider.GetAsync().ConfigureAwait(false)).ToArray();
        var receiverMap = receivers.ToDictionary(s => s.Guid, s => s);

        var jobs = (await
            _managerProvider.GetAsync().ConfigureAwait(false)).ToArray();
        var jobMap = jobs.ToDictionary(s => s.Guid, s => s);

        var guids = rssSources.Select(s => s.Guid);
        var rssSubscriptions = guids
            .Select(g =>
            {
                if (!rssSourcesMap.ContainsKey(g))
                {
                    _logger.LogError("Can't find rss source with guid {Guid:D}",
g);

                    return null;
                }
                if (!jobMap.ContainsKey(g))
                {
                    _logger.LogError("Can't find job with guid {Guid:D}", g);
                    return null;
                }

                var result = _mapper.Map<RssSubscription>(rssSourcesMap[g]);

                if (filterMap.ContainsKey(g))
                {
                    result = _mapper.Map(filterMap[g], result);
                }

                if (receiverMap.ContainsKey(g))
                {
                    result = _mapper.Map(receiverMap[g], result);
                }

                result = _mapper.Map(jobMap[g], result);
            });
    }

```

```

        return result;
    })
    .Where(s => s != null)
    .ToArray();

    return rssSubscriptions;
}
}

using AutoMapper;
using Common.Extensions;
using Downloader.Common.Models;
using Filter.Common.Models;
using Gateway.Common.Models;
using Manager.Common.Models;
using Sender.Common.Models;

namespace Gateway
{
    public class MapperProfile : Profile
    {
        public MapperProfile()
        {
            CreateMap<RssSubscription, RssSourceManageModel>()
                .ForMember(d => d.Url, o => o.MapFrom(s => s.RssSource))
                .ForMember(d => d.Guid, o => o.MapFrom(d => d.Guid));

            CreateMap<RssSourceReadModel, RssSubscription>()
                .ForMember(d => d.Guid, o => o.MapFrom(d => d.Guid))
                .ForMember(d => d.RssSource, o => o.MapFrom(d => d.Url))
                .ForAllOtherMembers(o => o.Ignore());

            CreateMap<RssSubscription, NewsFilterModel>()
                .ForMember(d => d.Filters, o => o.MapFrom(s => s.Filters))
                .ForMember(d => d.Guid, o => o.MapFrom(d => d.Guid))
                .ReverseMap()
                .ForMember(d => d.Guid, o => o.MapFrom(d => d.Guid))
                .ForMember(d => d.Filters, o => o.MapFrom(d => d.Filters))
                .ForAllOtherMembers(o => o.Ignore());

            CreateMap<RssSubscription, ReceiversModel>()
                .ForMember(d => d.Receivers, o => o.MapFrom(s => s.Receivers))
                .ForMember(d => d.Guid, o => o.MapFrom(d => d.Guid))
                .ReverseMap()
                .ForMember(d => d.Guid, o => o.MapFrom(d => d.Guid))
                .ForMember(d => d.Receivers, o => o.MapFrom(d => d.Receivers))
                .ForAllOtherMembers(o => o.Ignore());

            CreateMap<RssSubscription, JobModel>()
                .ForMember(d => d.Periodicity, o => o.MapFrom(s => s.Periodicity))
                .ForMember(d => d.Guid, o => o.MapFrom(d => d.Guid))
                .ForMember(d => d.IsJobEnabled, o => o.MapFrom(d =>
d.NeedToSendEmails))
                .ReverseMap()
                .ForMember(d => d.Guid, o => o.MapFrom(d => d.Guid))
                .ForMember(d => d.Periodicity, o => o.MapFrom(d => d.Periodicity))
                .ForMember(d => d.NeedToSendEmails, o => o.MapFrom(d =>
d.IsJobEnabled))
                .ForAllOtherMembers(o => o.Ignore());

            this.AssertConfigurationIsValid();
        }
    }
}

```

```

    }
}

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;

namespace Gateway
{
    public static class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        private static IHostBuilder CreateHostBuilder(string[] args) =>
            Host
                .CreateDefaultBuilder(args)
                .ConfigureAppConfiguration((_, config) =>
                {
                    config.AddEnvironmentVariables();
                })
                .ConfigureWebHostDefaults(webBuilder => {
                    webBuilder.UseStartup<Startup>(); });
    }
}

using Api.Common;
using Downloader.Facade;
using Filter.Facade;
using Gateway.Common.Contracts;
using Gateway.Common.Models;
using Gateway.Services;
using Manager.Facade;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.OpenApi.Any;
using Sender.Facade;
using Swashbuckle.AspNetCore.SwaggerGen;

namespace Gateway
{
    public sealed class Startup : BaseStartup
    {
        private const string DownloaderServiceName = "downloader-api";
        private const string FilterServiceName = "filter-api";
        private const string SenderServiceName = "sender-api";
        private const string ManagerServiceName = "manager-api";

        public Startup(IConfiguration configuration) : base(configuration)
        {
        }

        protected override IServiceCollection RegisterServices(IServiceCollection services)
        {
            services.AddScoped<IRssServiceProvider, RssServiceProvider>();
            services.AddScoped<IRssServiceManager, RssServiceManager>();
        }
    }
}

```

```

services.AddDownloaderProxies(Configuration.GetServiceUri(DownloaderServiceName).ToString());

services.AddFilterProxies(Configuration.GetServiceUri(FilterServiceName).ToString());
;

services.AddSenderProxies(Configuration.GetServiceUri(SenderServiceName).ToString());
;

services.AddManagerProxies(Configuration.GetServiceUri(ManagerServiceName).ToString());

        return services;
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        BaseConfigure(app, env);
    }

    protected override void AddSwaggerDocs(SwaggerGenOptions swaggerGenOptions)
    {
        swaggerGenOptions.MapType<RssSubscription>(() => new()
        {
            Example =
                new OpenApiObject
                {
                    { "guid", new OpenApiString("3fa85f64-5717-4562-b3fc-2c963f66afa6") },
                    { "periodicity", new OpenApiString("*/5 * * * *") },
                    { "rssSource", new OpenApiString("https://onliner.by/feed") },
                    {
                        "filters", new OpenApiArray
                        {
                            new OpenApiString("covid")
                        }
                    },
                    {
                        "receivers", new OpenApiArray
                        {
                            new OpenApiString("example@gmail.com")
                        }
                    }
                }
        });
        base.AddSwaggerDocs(swaggerGenOptions);
    }
}

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Downloader.Common.Models;
using Gateway.Common.Models;

namespace Gateway.Common.Contracts
{
    public interface IRssServiceManager
    {

```

```

        Task<Guid> CreateOrUpdateSubscriptionAsync(RssSubscription rssSubscription);

        Task DeleteSubscriptionAsync(Guid guid);

        Task<ICollection<NewsItem>> DownloadNewsAsync();
    }
}

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Gateway.Common.Models;

namespace Gateway.Common.Contracts
{
    public interface IRssServiceProvider
    {
        Task<RssSubscription> GetSubscriptionAsync(Guid guid);

        Task<ICollection<RssSubscription>> GetSubscriptionsAsync();
    }
}

using System;
using System.Collections.Generic;

namespace Gateway.Common.Models
{
    /// <summary>
    /// Rss subscription
    /// </summary>
    /// <example>
    /// {
    ///     "guid": "17a42ccd-bdac-421d-896e-35f208a0eefc",
    ///     "periodicity": "* /5 * * * *",
    ///     "rssSource": "https://onliner.by/feed",
    ///     "filters": [
    ///         "covid"
    ///     ],
    ///     "receivers": [
    ///         "example@gmail.com"
    ///     ]
    /// }
    /// </example>
    public sealed class RssSubscription
    {
        public Guid Guid { get; set; }

        /// <summary>
        /// Cron expression for periodicity of news mailing
        /// </summary>
        public string Periodicity { get; set; }

        public bool NeedToSendEmails { get; set; }

        public string RssSource { get; set; }

        /// <summary>
        /// List of regexes to filter header and description of news items
        /// </summary>
        public ICollection<string> Filters { get; set; } = Array.Empty<string>();
    }
}

```

```

        /// <summary>
        /// List of emails where to send news
        /// </summary>
        public ICollection<string> Receivers { get; set; } = Array.Empty<string>();
    }
}

using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading.Tasks;
using Common.Extensions;
using Downloader.Common.Models;
using Gateway.Common.Contracts;
using Gateway.Common.Models;

namespace Gateway.Facade.HttpProxy
{
    public sealed class RssServiceManagerProxy : IRssServiceManager
    {
        private readonly HttpClient _httpClient;

        public RssServiceManagerProxy(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        public async Task<Guid> CreateOrUpdateSubscriptionAsync(RssSubscription
            rssSubscription)
        {
            return await _httpClient.InternalPost<Guid>("/api/rss-subscription",
                rssSubscription).ConfigureAwait(false);
        }

        public async Task DeleteSubscriptionAsync(Guid guid)
        {
            await _httpClient.InternalDelete($"/api/rss-
                subscription/{guid:D}").ConfigureAwait(false);
        }

        public async Task<ICollection<NewsItem>> DownloadNewsAsync()
        {
            return await
                _httpClient.InternalGet<ICollection<NewsItem>>("/api/news").ConfigureAwait(false);
        }
    }
}

using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading.Tasks;
using Common.Extensions;
using Gateway.Common.Contracts;
using Gateway.Common.Models;

namespace Gateway.Facade.HttpProxy
{
    public sealed class RssServiceProviderProxy : IRssServiceProvider
    {
        private readonly HttpClient _httpClient;
    }
}

```



```

        public RssServiceProviderProxy(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        public async Task<RssSubscription> GetSubscriptionAsync(Guid guid)
        {
            return await _httpClient.InternalGet<RssSubscription>($"/api/rss-
subscription/{guid:D}").ConfigureAwait(false);
        }

        public async Task<ICollection<RssSubscription>> GetSubscriptionsAsync()
        {
            return await
            _httpClient.InternalGet<ICollection<RssSubscription>>("/api/rss-
subscription").ConfigureAwait(false);
        }
    }
}

using System;
using System.Net.Http;
using Gateway.Common.Contracts;
using Gateway.Facade.HttpProxy;
using Microsoft.Extensions.DependencyInjection;

namespace Gateway.Facade
{
    public static class ServiceCollectionExtensions
    {
        public static IServiceCollection AddGatewayProxies(this IServiceCollection
services, string baseUrl)
        {
            services.AddScoped<IRssServiceProvider>(s =>
            {
                var httpClient =
s.GetRequiredService<IHttpClientFactory>().CreateClient();
                httpClient.BaseAddress = new (baseUrl, UriKind.Absolute);
                return new RssServiceProviderProxy(httpClient);
            });

            services.AddScoped<IRssServiceManager>(s =>
            {
                var httpClient =
s.GetRequiredService<IHttpClientFactory>().CreateClient();
                httpClient.BaseAddress = new (baseUrl, UriKind.Absolute);
                return new RssServiceManagerProxy(httpClient);
            });

            return services;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Net;
using System.Threading.Tasks;
using Common.Models;
using Manager.Common.Contracts;
using Manager.Common.Models;
using Microsoft.AspNetCore.Mvc;

```

```

namespace Manager.API.Controllers
{
    [ApiController]
    [Produces("application/json")]
    [Route("api/manager")]
    public sealed class ManagerController : ControllerBase
    {
        private readonly IManagerProvider _managerProvider;
        private readonly IManagerManager _managerManager;

        public ManagerController(
            IManagerProvider managerProvider,
            IManagerManager managerManager)
        {
            _managerProvider = managerProvider;
            _managerManager = managerManager;
        }

        [HttpGet]
        [Route("{jobGuid:guid}")]
        [ProducesResponseType(typeof(JobModel), (int)HttpStatusCode.OK)]
        public async Task<IActionResult> GetAsync([FromRoute] Guid jobGuid)
        {
            var model = await _managerProvider.GetAsync(jobGuid).ConfigureAwait(false);
            return Ok(model);
        }

        [HttpGet]
        [Route("")]
        [ProducesResponseType(typeof(IEnumerable<JobModel>), (int)HttpStatusCode.OK)]
        public async Task<IActionResult> GetAsync()
        {
            var models = await _managerProvider.GetAsync().ConfigureAwait(false);
            return Ok(models);
        }

        [HttpGet]
        [Route("ensure-valid")]
        [ProducesResponseType((int)HttpStatusCode.OK)]
        [ProducesResponseType(typeof(ErrorResponse), (int)HttpStatusCode.BadRequest)]
        public async Task<IActionResult> EnsureRssSourceValidAsync([FromQuery] string
periodicity)
        {
            await
_managerProvider.EnsureJobPeriodicityIsValidAsync(periodicity).ConfigureAwait(false)
;
            return Ok();
        }

        [HttpPost]
        [Route("")]
        [ProducesResponseType((int)HttpStatusCode.OK)]
        [ProducesResponseType(typeof(ErrorResponse), (int)HttpStatusCode.BadRequest)]
        public async Task<IActionResult> CreateAsync([FromBody] JobModel model)
        {
            await _managerManager.CreateAsync(model).ConfigureAwait(false);
            return Ok();
        }

        [HttpPut]
        [Route("")]

```

```

        [ProducesResponseType((int)HttpStatusCode.OK)]
        [ProducesResponseType(typeof(ErrorResponse), (int)HttpStatusCode.BadRequest)]
        public async Task<IActionResult> UpdateAsync([FromBody] JobModel model)
        {
            await _managerManager.UpdateAsync(model).ConfigureAwait(false);
            return Ok();
        }

        [HttpDelete]
        [Route("{jobGuid:guid}")]
        [ProducesResponseType((int)HttpStatusCode.OK)]
        [ProducesResponseType(typeof(ErrorResponse), (int)HttpStatusCode.BadRequest)]
        public async Task<IActionResult> DeleteAsync([FromRoute] Guid jobGuid)
        {
            await _managerManager.DeleteAsync(jobGuid).ConfigureAwait(false);
            return Ok();
        }
    }
}

using Manager.API.Models;
using Microsoft.EntityFrameworkCore;

namespace Manager.API.Database
{
    public sealed class ManagerDbContext : DbContext
    {
        public ManagerDbContext(DbContextOptions options) : base(options) { }

        public DbSet<JobPeriodicityModel> JobPeriodicities { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<JobPeriodicityModel>(builder =>
            {
                builder.ToTable("JobPeriodicities");
                builder.HasKey(s => s.Guid).HasName("PK_JobPeriodicity");

                builder.Property(s =>
                    s.Guid).HasColumnName("Guid").IsRequired().ValueGeneratedNever();
                builder.Property(s =>
                    s.Periodicity).HasColumnName("Periodicity").HasColumnType("varchar(40)").HasMaxLength(40).IsRequired();
                builder.Property(s =>
                    s.IsJobEnabled).HasColumnName("IsJobEnabled").HasColumnType("boolean").IsRequired();
            });
            base.OnModelCreating(modelBuilder);
        }
    }
}

using System;
using Microsoft.EntityFrameworkCore.Migrations;

namespace Manager.API.Migrations
{
    public partial class Init : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "JobPeriodicities",

```

```

        columns: table => new
        {
            Guid = table.Column<Guid>(type: "uuid", nullable: false),
            Periodicity = table.Column<string>(type: "varchar(40)", maxLength:
40, nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_JobPeriodicity", x => x.Guid);
        });
    }

    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "JobPeriodicities");
    }
}

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace Manager.API.Migrations
{
    public partial class AddEnableFlag : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AddColumn<bool>(
                name: "IsJobEnabled",
                table: "JobPeriodicities",
                type: "boolean",
                nullable: false,
                defaultValue: true);
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropColumn(
                name: "IsJobEnabled",
                table: "JobPeriodicities");
        }
    }
}

using System;
using Db.Common.Models;

namespace Manager.API.Models
{
    public sealed class JobPeriodicityModel : IDbModel
    {
        public Guid Guid { get; set; }

        public bool IsJobEnabled { get; set; }

        public string Periodicity { get; set; }
    }
}

```

```

using Db.Common.Repositories.Contracts;
using Manager.API.Models;

namespace Manager.API.Repositories
{
    public interface IJobPeriodicityRepository : IBaseRepository<JobPeriodicityModel> {
    }
}

using Db.Common.Repositories;
using Manager.API.Database;
using Manager.API.Models;

namespace Manager.API.Repositories
{
    public sealed class JobPeriodicityRepository : BaseRepository<JobPeriodicityModel>,
        IJobPeriodicityRepository
    {
        public JobPeriodicityRepository(ManagerDbContext dbContext) : base(dbContext,
            dbContext.JobPeriodicities) { }
    }
}

using System;
using System.Threading.Tasks;

namespace Manager.API.Services
{
    public interface IMailingService
    {
        Task SendNewsAsync(Guid jobId);
    }
}

using System;
using System.Threading.Tasks;
using Downloader.Common.Contracts;
using Filter.Common.Contracts;
using Sender.Common.Contracts;

namespace Manager.API.Services
{
    public sealed class MailingService : IMailingService
    {
        private readonly IDownloaderManager _downloaderManager;
        private readonly IFilterProvider _filterProvider;
        private readonly ISenderManager _senderManager;

        public MailingService(
            IDownloaderManager downloaderManager,
            IFilterProvider filterProvider,
            ISenderManager senderManager)
        {
            _downloaderManager = downloaderManager;
            _filterProvider = filterProvider;
            _senderManager = senderManager;
        }

        public async Task SendNewsAsync(Guid jobId)
        {
            var news = await
                _downloaderManager.DownloadNewsAsync(jobId).ConfigureAwait(false);

```

```

        news = await _filterProvider.FilterNewsAsync(jobId,
news).ConfigureAwait(false);
        await _senderManager.SendNewsAsync(jobId, news).ConfigureAwait(false);
    }
}

using System;
using System.Threading.Tasks;
using AutoMapper;
using Common.Exceptions;
using Hangfire;
using Manager.API.Models;
using Manager.API.Repositories;
using Manager.API.Resources;
using Manager.Common.Contracts;
using Manager.Common.Models;

namespace Manager.API.Services
{
    public sealed class ManagerManager : IManagerManager
    {
        private readonly IMapper _mapper;
        private readonly IJobPeriodicityRepository _jobPeriodicityRepository;
        private readonly IManagerProvider _managerProvider;
        private readonly IRecurringJobManager _recurringJobManager;

        public ManagerManager(
            IMapper mapper,
            IJobPeriodicityRepository jobPeriodicityRepository,
            IManagerProvider managerProvider,
            IRecurringJobManager recurringJobManager)
        {
            _mapper = mapper;
            _jobPeriodicityRepository = jobPeriodicityRepository;
            _managerProvider = managerProvider;
            _recurringJobManager = recurringJobManager;
        }

        public async Task CreateAsync(JobModel job)
        {
            if (await
_jobPeriodicityRepository.IsExistsAsync(job.Guid).ConfigureAwait(false))
            {
                throw new AlreadyExistsException(Localization.JobAlreadyExists);
            }

            await
_managerProvider.EnsureJobPeriodicityIsValidAsync(job.Periodicity).ConfigureAwait(fa
lse);

            var model = _mapper.Map<JobPeriodicityModel>(job);

            await _jobPeriodicityRepository.CreateAsync(model).ConfigureAwait(false);

            if (job.IsJobEnabled) {
                ScheduleJob(job);
            }
        }

        public async Task UpdateAsync(JobModel job)
        {

```

```

        if (!await
_jobPeriodicityRepository.IsExistsAsync(job.Guid).ConfigureAwait(false))
        {
            throw new NotFoundException(Localization.JobNotFound);
        }

        await
_managerProvider.EnsureJobPeriodicityIsValidAsync(job.Periodicity).ConfigureAwait(fa
lse);

        var model = _mapper.Map<JobPeriodicityModel>(job);

        await _jobPeriodicityRepository.UpdateAsync(model).ConfigureAwait(false);

        if (job.IsJobEnabled)
        {
            ScheduleJob(job);
        }
        else
        {
            RemoveJob(job);
        }
    }

    public async Task DeleteAsync(Guid jobGuid)
    {
        if (!await
_jobPeriodicityRepository.IsExistsAsync(jobGuid).ConfigureAwait(false))
        {
            throw new NotFoundException(Localization.JobNotFound);
        }

        await _jobPeriodicityRepository.DeleteAsync(jobGuid).ConfigureAwait(false);

        _recurringJobManager.RemoveIfExists(jobGuid.ToString("D"));
    }

    private void ScheduleJob(JobModel job)
    {
        _recurringJobManager.AddOrUpdate<IMailingService>(
            job.Guid.ToString("D"),
            x => x.SendNewsAsync(job.Guid),
            job.Periodicity);
    }

    private void RemoveJob(JobModel job)
    {
        _recurringJobManager.RemoveIfExists(
            job.Guid.ToString("D"));
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using AutoMapper;
using Common.Exceptions;
using Manager.API.Repositories;
using Manager.API.Resources;

```

```

using Manager.Common.Contracts;
using Manager.Common.Models;

namespace Manager.API.Services
{
    public sealed class ManagerProvider : IManagerProvider
    {
        private static readonly Regex CronRegex =
            new(@"(@(annually|yearly|monthly|weekly|daily|hourly|reboot))|(@every
            (\d+(ns|us|µs|ms|s|m|h))+)|((((\d+,)+\d+|(\d+(-|/)\d+)|\d+|\*) ?){5,7})");

        private readonly IJobPeriodicityRepository _jobPeriodicityRepository;
        private readonly IMapper _mapper;

        public ManagerProvider(IJobPeriodicityRepository jobPeriodicityRepository,
            IMapper mapper)
        {
            _jobPeriodicityRepository = jobPeriodicityRepository;
            _mapper = mapper;
        }

        public async Task<JobModel> GetAsync(Guid guid)
        {
            var jobPeriodicity = await
            _jobPeriodicityRepository.GetAsync(guid).ConfigureAwait(false);
            return _mapper.Map<JobModel>(jobPeriodicity);
        }

        public async Task<IEnumerable<JobModel>> GetAsync()
        {
            var jobPeriodicities = await
            _jobPeriodicityRepository.GetAsync().ConfigureAwait(false);
            return jobPeriodicities.Select(_mapper.Map<JobModel>);
        }

        public Task EnsureJobPeriodicityIsValidAsync(string jobPeriodicity)
        {
            return CronRegex.IsMatch(jobPeriodicity)
                ? Task.CompletedTask
                : Task.FromException(new
                BadRequestException(Localization.NotAJobPeriodicity));
        }
    }
}

using Hangfire.Dashboard;

namespace Manager.API
{
    public class DashboardAuthorizationFilter : IDashboardAuthorizationFilter
    {
        public bool Authorize(DashboardContext context)
        {
            return true;
        }
    }
}

using AutoMapper;
using Common.Extensions;
using Manager.API.Models;
using Manager.Common.Models;

```



```

namespace Manager.API
{
    public sealed class MapperProfile : Profile
    {
        public MapperProfile()
        {
            CreateMap<JobModel, JobPeriodicityModel>()
                .ReverseMap();

            this.AssertConfigurationIsValid();
        }
    }
}

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;

namespace Manager.API
{
    public static class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        private static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureAppConfiguration((_, config) =>
                {
                    config.AddEnvironmentVariables();
                })
                .ConfigureWebHostDefaults(webBuilder => {
                    webBuilder.UseStartup<Startup>(); });
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using Api.Common;
using Downloader.Facade;
using Filter.Facade;
using Hangfire;
using Hangfire.Dashboard;
using Hangfire.PostgreSql;
using Manager.API.Database;
using Manager.API.Repositories;
using Manager.API.Services;
using Manager.Common.Contracts;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Sender.Facade;

namespace Manager.API
{
    public sealed class Startup : BaseStartup

```

```

{
    private const string ConnectionStringSetting = "ConnectionString";
    private const string DownloaderServiceName = "downloader-api";
    private const string FilterServiceName = "filter-api";
    private const string SenderServiceName = "sender-api";

    public Startup(IConfiguration configuration) : base(configuration) {}

    protected override IServiceCollection RegisterServices(IServiceCollection
services)
    {
        services.AddDbContext<ManagerDbContext>(options =>
            options.UseNpgsql(Configuration[ConnectionStringSetting]));

        services.AddScoped<IJobPeriodicityRepository, JobPeriodicityRepository>();

        services.AddScoped<IManagerManager, ManagerManager>();
        services.AddScoped<IManagerProvider, ManagerProvider>();
        services.AddScoped<IMailingService, MailingService>();

        services.AddDownloaderProxies(Configuration.GetServiceUri(DownloaderServiceName).ToString());

        services.AddFilterProxies(Configuration.GetServiceUri(FilterServiceName).ToString())
        ;

        services.AddSenderProxies(Configuration.GetServiceUri(SenderServiceName).ToString())
        ;

        services.AddHangfire(configuration => configuration
            .SetDataCompatibilityLevel(CompatibilityLevel.Version_170)
            .UseSimpleAssemblyNameTypeSerializer()
            .UseRecommendedSerializerSettings()
            .UsePostgreSqlStorage(Configuration[ConnectionStringSetting]));

        services.AddHangfireServer(options =>
        {
            options.WorkerCount = 2;
        });

        return services;
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env,
ManagerDbContext dbContext)
    {
        BaseConfigure(app, env);

        app.UseHangfireDashboard("/hangfire", new ()
        {
            Authorization = new List<IDashboardAuthorizationFilter> { new
DashboardAuthorizationFilter() },
            IgnoreAntiforgeryToken = true,
            DisplayNameFunc = (context, job) =>
            {
                if (job.Args?.FirstOrDefault() is Guid jobGuid)
                {
                    return $"Publish event: {jobGuid}";
                }
            }

            return job.ToString();
        });
    }
}

```

```

        });
    }

    dbContext.Database.Migrate();
}

using System;
using System.Threading.Tasks;
using Manager.Common.Models;

namespace Manager.Common.Contracts
{
    public interface IManagerManager
    {
        Task CreateAsync(JobModel job);

        Task UpdateAsync(JobModel job);

        Task DeleteAsync(Guid jobGuid);
    }
}

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Manager.Common.Models;

namespace Manager.Common.Contracts
{
    public interface IManagerProvider
    {
        Task<JobModel> GetAsync(Guid jobGuid);

        Task<IEnumerable<JobModel>> GetAsync();

        Task EnsureJobPeriodicityIsValidAsync(string jobPeriodicity);
    }
}

using System;

namespace Manager.Common.Models
{
    public sealed class JobModel
    {
        public Guid Guid { get; set; }

        public bool IsJobEnabled { get; set; }

        public string Periodicity { get; set; }
    }
}

using System;
using System.Net.Http;
using System.Threading.Tasks;
using Common.Extensions;
using Manager.Common.Contracts;
using Manager.Common.Models;

```

```

namespace Manager.Facade.HttpProxy
{
    public sealed class ManagerManagerProxy : IManagerManager
    {
        private readonly HttpClient _httpClient;

        public ManagerManagerProxy(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        public async Task CreateAsync(JobModel job)
        {
            await _httpClient.InternalPost("/api/manager", job).ConfigureAwait(false);
        }

        public async Task UpdateAsync(JobModel job)
        {
            await _httpClient.InternalPut("/api/manager", job).ConfigureAwait(false);
        }

        public async Task DeleteAsync(Guid jobGuid)
        {
            await
            _httpClient.InternalDelete($"/api/manager/{jobGuid:D}").ConfigureAwait(false);
        }
    }
}

using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading.Tasks;
using Common.Extensions;
using Manager.Common.Contracts;
using Manager.Common.Models;

namespace Manager.Facade.HttpProxy
{
    public sealed class ManagerProviderProxy : IManagerProvider
    {
        private readonly HttpClient _httpClient;

        public ManagerProviderProxy(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        public async Task<JobModel> GetAsync(Guid receiversGuid)
        {
            return await
            _httpClient.InternalGet<JobModel>($"/api/manager/{receiversGuid:D}").ConfigureAwait(
            false);
        }

        public async Task<IEnumerable<JobModel>> GetAsync()
        {
            return await
            _httpClient.InternalGet<IEnumerable<JobModel>>("/api/manager").ConfigureAwait(false)
            ;
        }
    }
}

```

```

        public async Task EnsureJobPeriodicityIsValidAsync(string jobPeriodicity)
        {
            await _httpClient.InternalGet($"/api/manager/ensure-
valid?periodicity={jobPeriodicity}").ConfigureAwait(false);
        }
    }
}

using System;
using System.Net.Http;
using Manager.Common.Contracts;
using Microsoft.Extensions.DependencyInjection;
using Manager.Facade.HttpProxy;

namespace Manager.Facade
{
    public static class ServiceCollectionExtensions
    {
        public static IServiceCollection AddManagerProxies(this IServiceCollection
services, string baseUrl)
        {
            services.AddScoped<IManagerProvider>(s =>
            {
                var httpClient =
s.GetRequiredService<IHttpClientFactory>().CreateClient();
                httpClient.BaseAddress = new (baseUrl, UriKind.Absolute);
                return new ManagerProviderProxy(httpClient);
            });

            services.AddScoped<IManagerManager>(s =>
            {
                var httpClient =
s.GetRequiredService<IHttpClientFactory>().CreateClient();
                httpClient.BaseAddress = new (baseUrl, UriKind.Absolute);
                return new ManagerManagerProxy(httpClient);
            });

            return services;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Net;
using System.Threading.Tasks;
using Downloader.Common.Models;
using Microsoft.AspNetCore.Mvc;
using Sender.Common.Contracts;
using Sender.Common.Models;

namespace Sender.API.Controllers
{
    [ApiController]
    [Produces("application/json")]
    [Route("api/sender")]
    public sealed class ReceiversController : ControllerBase
    {
        private readonly ISenderProvider _senderProvider;
        private readonly ISenderManager _senderManager;
    }
}

```

```

    public ReceiversController(ISenderProvider senderProvider, ISenderManager
senderManager)
    {
        _senderProvider = senderProvider;
        _senderManager = senderManager;
    }

    [HttpGet]
    [Route("{receiverGuid:guid}")]
    [ProducesResponseType(typeof(ReceiversModel), (int)HttpStatusCode.OK)]
    public async Task<IActionResult> GetAsync([FromRoute] Guid receiverGuid)
    {
        var receivers = await
_senderProvider.GetAsync(receiverGuid).ConfigureAwait(false);
        return Ok(receivers);
    }

    [HttpGet]
    [Route("")]
    [ProducesResponseType(typeof(IEnumerable<ReceiversModel>),
(int)HttpStatusCode.OK)]
    public async Task<IActionResult> GetAsync()
    {
        var receivers = await _senderProvider.GetAsync().ConfigureAwait(false);
        return Ok(receivers);
    }

    [HttpPost]
    [Route("ensure-valid")]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    public async Task<IActionResult> EnsureReceiversValidAsync([FromBody]
IEnumerable<string> receivers)
    {
        await
_senderProvider.EnsureReceiversIsValidAsync(receivers).ConfigureAwait(false);
        return Ok();
    }

    [HttpPost]
    [Route("{receiverGuid:guid}/send-news")]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    public async Task<IActionResult> SendNewsAsync([FromRoute] Guid receiverGuid,
[FromBody] IEnumerable<NewsItem> news)
    {
        await _senderManager.SendNewsAsync(receiverGuid,
news).ConfigureAwait(false);
        return Ok();
    }

    [HttpPost]
    [Route("")]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    public async Task<IActionResult> CreateAsync([FromBody] ReceiversModel
receivers)
    {
        await _senderManager.CreateAsync(receivers);
        return Ok();
    }

    [HttpPut]
    [Route("")]
    [ProducesResponseType((int)HttpStatusCode.OK)]

```

```

        public async Task<IActionResult> UpdateAsync([FromBody] ReceiversModel
receivers)
        {
            await _senderManager.UpdateAsync(receivers);
            return Ok();
        }

        [HttpDelete]
        [Route("{receiverGuid:guid}")]
        [ProducesResponseType((int)HttpStatusCode.OK)]
        public async Task<IActionResult> DeleteAsync([FromRoute] Guid receiverGuid)
        {
            await _senderManager.DeleteAsync(receiverGuid);
            return Ok();
        }
    }
}

using Microsoft.EntityFrameworkCore;
using Sender.API.Models;

namespace Sender.API.Database
{
    public sealed class SenderDbContext : DbContext
    {
        public DbSet<ReceiverEmailModel> Emails { get; set; }

        public SenderDbContext(DbContextOptions options) : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<ReceiverEmailModel>(builder =>
            {
                builder.ToTable("Emails");
                builder.HasKey(s => s.Guid).HasName("PK_Email");
                builder.HasIndex(s =>
s.GroupGuid).IsUnique(false).HasDatabaseName("IDX_Email_GroupGuid");

                builder.Property(s =>
s.Guid).HasColumnName("Guid").IsRequired().ValueGeneratedOnAdd();
                builder.Property(s =>
s.GroupGuid).HasColumnName("GroupGuid").IsRequired().ValueGeneratedNever();
                builder.Property(s =>
s.Email).HasColumnName("Email").IsRequired().HasColumnType("varchar(1000)").HasMaxLength(1000);
            });
        }
    }
}

using System.Collections.Generic;
using System.Threading.Tasks;

namespace Sender.API.ExternalServices
{
    public interface ISmtpService
    {
        Task SendMailAsync(IEnumerable<string> receivers, string subject, string body);
    }
}

```

```

using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Common.Extensions;
using MailKit.Net.Smtp;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using MimeKit;
using Sender.API.Sections;

namespace Sender.API.ExternalServices
{
    public sealed class SmtpService : ISmtpService
    {
        private readonly SmtpClient _smtpClient;
        private readonly ILogger<SmtpService> _logger;
        private readonly IOptions<SmtpSection> _smtpOptions;

        public SmtpService(SmtpClient smtpClient, ILogger<SmtpService> logger,
            IOptions<SmtpSection> smtpOptions)
        {
            _smtpClient = smtpClient;
            _logger = logger;
            _smtpOptions = smtpOptions;
        }

        public async Task SendMailAsync(IEnumerable<string> receivers, string subject,
            string body)
        {
            receivers = receivers.Where(r => !string.IsNullOrEmpty(r)).ToArray();

            var message = new MimeMessage(
                from: new [] { MailboxAddress.Parse(_smtpOptions.Value.From) },
                to: receivers.Select(MailboxAddress.Parse),
                subject: subject,
                body: BuildBody(body));

            if (!_smtpClient.IsConnected)
            {
                await _smtpClient.ConnectAsync(_smtpOptions.Value.Host,
                    _smtpOptions.Value.Port, false);
            }

            if (!_smtpClient.IsAuthenticated)
            {
                await _smtpClient.AuthenticateAsync(string.Empty, string.Empty);
            }

            _logger.LogInformation("Start sending email to {To}",
                receivers.JoinString());

            await _smtpClient.SendAsync(message);

            _logger.LogInformation("End sending email to {To}",
                receivers.JoinString());
        }

        private static MimeEntity BuildBody(string body)
        {
            var bodyBuilder = new BodyBuilder
            {

```



```

        HtmlBody = body
    };
    return bodyBuilder.ToMessageBody();
}

}

}

using System;
using Microsoft.EntityFrameworkCore.Migrations;

namespace Sender.API.Migrations
{
    public partial class Init : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Emails",
                columns: table => new
                {
                    Guid = table.Column<Guid>(type: "uuid", nullable: false),
                    GroupGuid = table.Column<Guid>(type: "uuid", nullable: false),
                    Email = table.Column<string>(type: "varchar(1000)", maxLength:
1000, nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Email", x => x.Guid);
                });

            migrationBuilder.CreateIndex(
                name: "IDX_Email_GroupGuid",
                table: "Emails",
                column: "GroupGuid");
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Emails");
        }
    }
}

using System;
using Db.Common.Models;

namespace Sender.API.Models
{
    public sealed class ReceiverEmailModel : IDbCollectionModel
    {
        public Guid Guid { get; set; }

        public Guid GroupGuid { get; set; }

        public string Email { get; set; }
    }
}

using Db.Common.Repositories;

```

```

using Sender.API.Database;
using Sender.API.Models;

namespace Sender.API.Repository
{
    public sealed class EmailRepository : BaseCollectionRepository<ReceiverEmailModel>,
    IEmailRepository
    {
        public EmailRepository(SenderDbContext dbContext) : base(dbContext,
        dbContext.Emails)
        {
        }
    }
}

using Db.Common.Repositories.Contracts;
using Sender.API.Models;

namespace Sender.API.Repository
{
    public interface IEmailRepository : IBaseCollectionRepository<ReceiverEmailModel> {
    }
}

namespace Sender.API.Sections
{
    public class SmtplibSection
    {
        public string Host { get; set; }

        public int Port { get; set; }

        public string From { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using Common.Exceptions;
using Common.Extensions;
using Downloader.Common.Models;
using Sender.API.ExternalServices;
using Sender.API.Models;
using Sender.API.Repository;
using Sender.API.Resources;
using Sender.Common.Contracts;
using Sender.Common.Models;

namespace Sender.API.Services
{
    public sealed class SenderManager : ISenderManager
    {
        private readonly IMapper _mapper;
        private readonly IEmailRepository _emailRepository;
        private readonly ISenderProvider _senderProvider;
        private readonly ISmtplibService _smtpService;

        public SenderManager(
            IMapper mapper,

```

```

        IEmailRepository emailRepository,
        ISenderProvider senderProvider,
        ISmtpService smtpService)
    {
        _mapper = mapper;
        _emailRepository = emailRepository;
        _senderProvider = senderProvider;
        _smtpService = smtpService;
    }

    public async Task CreateAsync(ReceiversModel receivers)
    {
        if (await _emailRepository.IsExistsAsync(f => f.GroupGuid ==
receivers.Guid).ConfigureAwait(false))
        {
            throw new AlreadyExistsException(Localization.ReceiverAlreadyExists);
        }

        await
_senderProvider.EnsureReceiversIsValidAsync(receivers.Receivers).ConfigureAwait(fals
e);

        var models = receivers.Receivers.Select(f =>
        {
            var model = _mapper.Map<ReceiverEmailModel>(receivers);
            model.Email = f;
            return model;
        });

        await _emailRepository.CreateAsync(models).ConfigureAwait(false);
    }

    public async Task UpdateAsync(ReceiversModel receivers)
    {
        await
_senderProvider.EnsureReceiversIsValidAsync(receivers.Receivers).ConfigureAwait(fals
e);

        await
_emailRepository.DeleteGroupAsync(receivers.Guid).ConfigureAwait(false);

        await CreateAsync(receivers).ConfigureAwait(false);
    }

    public async Task DeleteAsync(Guid receiversGuid)
    {
        if (!await _emailRepository.IsExistsAsync(f => f.GroupGuid ==
receiversGuid).ConfigureAwait(false))
        {
            throw new NotFoundException(Localization.ReceiverNotFound);
        }

        await
_emailRepository.DeleteGroupAsync(receiversGuid).ConfigureAwait(false);
    }

    public async Task SendNewsAsync(Guid receiversGuid, IEnumerable<NewsItem> news)
    {
        var receivers = await
_emailRepository.GetGroupAsync(receiversGuid).ConfigureAwait(false);
        var emails = receivers.Select(r => r.Email);
        var body = news.Select(n =>

```

```

        $"{n.Title}\n</br></br>\n{n.Description}").JoinString("\n</br>\n");
        await _smtpService.SendMailAsync(emails, Localization.EmailSubject,
        body).ConfigureAwait(false);
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Mail;
using System.Threading.Tasks;
using AutoMapper;
using Common.Exceptions;
using Sender.API.Repository;
using Sender.API.Resources;
using Sender.Common.Contracts;
using Sender.Common.Models;

namespace Sender.API.Services
{
    public sealed class SenderProvider : ISenderProvider
    {
        private readonly IMapper _mapper;
        private readonly IEmailRepository _emailRepository;

        public SenderProvider(IMapper mapper, IEmailRepository emailRepository)
        {
            _mapper = mapper;
            _emailRepository = emailRepository;
        }

        public async Task<ReceiversModel> GetAsync(Guid receiverGuid)
        {
            var models = await _emailRepository.GetGroupAsync(receiverGuid);
            var receivers = _mapper.Map<ReceiversModel>(models);
            receivers.Guid = receiverGuid;
            return receivers;
        }

        public async Task<IEnumerable<ReceiversModel>> GetAsync()
        {
            var models = await _emailRepository.GetAsync();
            return models
                .GroupBy(m => m.GroupGuid)
                .Select(m => _mapper.Map<ReceiversModel>(m));
        }

        public Task EnsureReceiversIsValidAsync(IEnumerable<string> receiverEmails)
        {
            foreach (var email in receiverEmails)
            {
                try {
                    var address = new MailAddress(email).Address;
                } catch (FormatException e) {
                    return Task.FromException(new
                    BadRequestException(string.Format(Localization.NotAnEmail, email), e));
                }
            }

            return Task.CompletedTask;
        }
    }
}

```

```

    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using AutoMapper;
using Common.Extensions;
using Sender.API.Models;
using Sender.Common.Models;

namespace Sender.API
{
    public sealed class MapperProfile : Profile
    {
        public MapperProfile()
        {
            CreateMap<ReceiversModel, ReceiverEmailModel>()
                .ForMember(d => d.Guid, o => o.MapFrom(s => Guid.NewGuid()))
                .ForMember(d => d.GroupGuid, o => o.MapFrom(s => s.Guid))
                .ForMember(d => d.Email, o => o.Ignore());

            CreateMap<IEnumerable<ReceiverEmailModel>, ReceiversModel>()
                .ForMember(d => d.Guid, o => o.MapFrom((s, _, _) =>
                    s.FirstOrDefault()?.GroupGuid ?? Guid.Empty))
                .ForMember(d => d.Receivers, o => o.MapFrom(s => s.Select(f =>
                    f.Email)));

            this.AssertConfigurationIsValid();
        }
    }
}

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;

namespace Sender.API
{
    public static class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        private static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureAppConfiguration((_, config) =>
                {
                    config.AddEnvironmentVariables();
                })
                .ConfigureWebHostDefaults(webBuilder => {
                    webBuilder.UseStartup<Startup>(); });
    }
}

using Api.Common;
using MailKit.Net.Smtp;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;

```

```

using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Sender.API.Database;
using Sender.API.ExternalServices;
using Sender.API.Repository;
using Sender.API.Sections;
using Sender.API.Services;
using Sender.Common.Contracts;

namespace Sender.API
{
    public sealed class Startup : BaseStartup
    {
        public Startup(IConfiguration configuration) : base(configuration) {}

        protected override IServiceCollection RegisterServices(IServiceCollection services)
        {
            services.AddDbContext<SenderDbContext>(options =>
                options.UseNpgsql(Configuration["ConnectionString"]));

            services.AddScoped<IEmailRepository, EmailRepository>();

            services.Configure<SmtpSection>(Configuration.GetSection("SmtpSettings"));
            services.AddScoped<ISmtpService, SmtpService>();
            services.AddScoped<SmtpClient>(_ => new ());

            services.AddScoped<ISenderProvider, SenderProvider>();
            services.AddScoped<ISenderManager, SenderManager>();

            return services;
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env,
            SenderDbContext dbContext)
        {
            BaseConfigure(app, env);

            dbContext.Database.Migrate();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Downloader.Common.Models;
using Sender.Common.Models;

namespace Sender.Common.Contracts
{
    public interface ISenderManager
    {
        Task CreateAsync(ReceiversModel model);

        Task UpdateAsync(ReceiversModel model);

        Task DeleteAsync(Guid receiversGuid);

        Task SendNewsAsync(Guid receiversGuid, IEnumerable<NewsItem> news);
    }
}

```

```

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Sender.Common.Models;

namespace Sender.Common.Contracts
{
    public interface ISenderProvider
    {
        Task<ReceiversModel> GetAsync(Guid receiverGuid);

        Task<IEnumerable<ReceiversModel>> GetAsync();

        Task EnsureReceiversIsValidAsync(IEnumerable<string> receiverEmails);
    }
}

using System;
using System.Collections.Generic;

namespace Sender.Common.Models
{
    public sealed class ReceiversModel
    {
        public Guid Guid { get; set; }

        public IEnumerable<string> Receivers { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading.Tasks;
using Common.Extensions;
using Downloader.Common.Models;
using Sender.Common.Contracts;
using Sender.Common.Models;

namespace Sender.Facade.HttpProxy
{
    public sealed class SenderManagerProxy : ISenderManager
    {
        private readonly HttpClient _httpClient;

        public SenderManagerProxy(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        public async Task CreateAsync(ReceiversModel receivers)
        {
            await _httpClient.InternalPost("/api/sender",
            receivers).ConfigureAwait(false);
        }

        public async Task UpdateAsync(ReceiversModel receivers)
        {
            await _httpClient.InternalPut("/api/sender",
            receivers).ConfigureAwait(false);
        }
    }
}

```

```

        public async Task DeleteAsync(Guid receiversGuid)
        {
            await
            _httpClient.InternalDelete($"/api/sender/{receiversGuid:D}").ConfigureAwait(false);
        }

        public async Task SendNewsAsync(Guid receiversGuid, IEnumerable<NewsItem> news)
        {
            await _httpClient.InternalPost($"/api/sender/{receiversGuid:D}/send-news",
            news).ConfigureAwait(false);
        }
    }
}

using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading.Tasks;
using Common.Extensions;
using Sender.Common.Contracts;
using Sender.Common.Models;

namespace Sender.Facade.HttpProxy
{
    public sealed class SenderProviderProxy : ISenderProvider
    {
        private readonly HttpClient _httpClient;

        public SenderProviderProxy(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        public async Task<ReceiversModel> GetAsync(Guid receiverGuid)
        {
            return await
            _httpClient.InternalGet<ReceiversModel>($"/api/sender/{receiverGuid:D}").ConfigureAwait(
            await(false);
        }

        public async Task<IEnumerable<ReceiversModel>> GetAsync()
        {
            return await
            _httpClient.InternalGet<IEnumerable<ReceiversModel>>("/api/sender").ConfigureAwait(f
            alse);
        }

        public async Task EnsureReceiversIsValidAsync(IEnumerable<string>
        receiverEmails)
        {
            await _httpClient.InternalPost("/api/sender/ensure-valid",
            receiverEmails).ConfigureAwait(false);
        }
    }
}

using System;
using System.Net.Http;
using Microsoft.Extensions.DependencyInjection;
using Sender.Common.Contracts;
using Sender.Facade.HttpProxy;

```



```

namespace Sender.Facade
{
    public static class ServiceCollectionExtensions
    {
        public static IServiceCollection AddSenderProxies(this IServiceCollection
services, string baseUrl)
        {
            services.AddScoped<ISenderProvider>(s =>
            {
                var httpClient =
s.GetRequiredService<IHttpClientFactory>().CreateClient();
                httpClient.BaseAddress = new (baseUrl, UriKind.Absolute);
                return new SenderProviderProxy(httpClient);
            });

            services.AddScoped<ISenderManager>(s =>
            {
                var httpClient =
s.GetRequiredService<IHttpClientFactory>().CreateClient();
                httpClient.BaseAddress = new (baseUrl, UriKind.Absolute);
                return new SenderManagerProxy(httpClient);
            });

            return services;
        }
    }
}

using System;
using System.Net;
using System.Threading.Tasks;
using Common.Exceptions;
using Common.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;

namespace Api.Common.Middlewares
{
    // https://jasonwatmore.com/post/2020/10/02/aspnet-core-31-global-error-handler-
tutorial
    public sealed class ErrorHandlerMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger<ErrorHandlerMiddleware> _logger;

        public ErrorHandlerMiddleware(RequestDelegate next,
ILogger<ErrorHandlerMiddleware> logger)
        {
            _next = next;
            _logger = logger;
        }

        public async Task Invoke(HttpContext context)
        {
            try
            {
                await _next(context);
                _logger.LogTrace("Incoming request {Path} completed with {Code} status
code", context.Request.Path.Value, context.Response.StatusCode);
            }
        }
    }
}

```

```

        catch (Exception error)
        {
            var response = context.Response;
            response.ContentType = "application/json";

            switch(error)
            {
                case BaseHttpException e:
                    response.StatusCode = (int) e.ErrorCode;
                    break;
                default:
                    // unhandled error
                    response.StatusCode = (int)HttpStatusCode.InternalServerError;
                    break;
            }

            _logger.LogError("Incoming request {0} failed with {1} status code,
error:{2}\n", context.Request.Path.Value, context.Response.StatusCode, error);

            var result = JsonConvert.SerializeObject(new ErrorResponse{
ErrorMessage = error.Message });
            await response.WriteAsync(result);
        }
    }
}

using System;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Reflection;
using Api.Common.Middlewares;
using Common.Exceptions;
using Common.Json;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Newtonsoft.Json;
using Swashbuckle.AspNetCore.SwaggerGen;

namespace Api.Common
{
    public abstract class BaseStartup
    {
        private const string CorsPolicy = "AllowAnyOrigins";

        protected BaseStartup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        protected IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to
        the container.
        public void ConfigureServices(IServiceCollection services)
        {
            var entryAssembly = GetEntryAssembly();
            var assemblyName = GetEntryAssemblyName();

```

```

        services.AddControllers()
            .ConfigureApiBehaviorOptions(o =>
            {
                o.InvalidModelStateResponseFactory =
                    context => throw new
BadRequestException(context.ModelState.Values.First().Errors.First().ErrorMessage);
            })
            .AddNewtonsoftJson(options =>
            {
                options.SerializerSettings.Converters.Add(new
TimeSpanJsonConverter());
                options.SerializerSettings.DateTimeZoneHandling =
DateTimeZoneHandling.Utc;
            });

        services.AddSwaggerGen(c =>
        {
            var commentsFileName = entryAssembly?.GetName().Name + ".xml";
            var commentsFile = Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
commentsFileName);

            c.SwaggerDoc("v1", new () {Title = assemblyName, Version = "v1"});
            c.IncludeXmlComments(commentsFile);

            AddSwaggerDocs(c);
        });
        services.AddSwaggerGenNewtonsoftSupport();

        services.AddAutoMapper(entryAssembly);

        services.AddHttpClient();
        // https://docs.microsoft.com/en-
us/dotnet/architecture/microservices/implement-resilient-applications/use-
httpclientfactory-to-implement-resilient-http-requests
        services.AddSingleton(provider =>
provider.GetService<IHttpClientFactory>().CreateClient());

        services.AddCors(o => o.AddPolicy(CorsPolicy, builder =>
            builder
                .AllowAnyOrigin()
                .AllowAnyMethod()
                .AllowAnyHeader()));

        RegisterServices(services);
    }

    // This method gets called by the runtime. Use this method to configure the
    HTTP request pipeline.
    protected void BaseConfigure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseSwagger();
        app.UseSwaggerUI(c =>
        {
            c.SwaggerEndpoint("/swagger/v1/swagger.json",
$"GetEntryAssemblyName() v1");
        });
    }

```

```

        });

        app.UseHttpsRedirection();

        app.UseRouting();

        app.UseCors(CorsPolicy);

        app.UseMiddleware<ErrorHandlerMiddleware>();

        app.UseAuthorization();

        app.UseEndpoints(endpoints => { endpoints.MapControllers(); });
    }

    protected abstract IServiceCollection RegisterServices(IServiceCollection
services);

    protected virtual void AddSwaggerDocs(SwaggerGenOptions swaggerGenOptions) {}

    private string GetEntryAssemblyName() => GetEntryAssembly().GetName().Name ??
string.Empty;

    private Assembly GetEntryAssembly() => Assembly.GetEntryAssembly() ??
GetType().Assembly;
}

using System;
using System.Net;

namespace Common.Exceptions
{
    public sealed class AlreadyExistsException : BaseHttpException
    {
        public AlreadyExistsException(string message)
            : base(message, HttpStatusCode.Conflict)
        {
        }

        public AlreadyExistsException(string message, Exception innerException)
            : base(message, innerException, HttpStatusCode.Conflict)
        {
        }
    }
}

using System;
using System.Net;

namespace Common.Exceptions
{
    public sealed class BadRequestException : BaseHttpException
    {
        public BadRequestException(string message)
            : base(message, HttpStatusCode.BadRequest)
        {
        }

        public BadRequestException(string message, Exception innerException)
            : base(message, innerException, HttpStatusCode.BadRequest)
        {
        }
    }
}

```

```

    }
}

using System;
using System.Diagnostics;
using System.Net;

namespace Common.Exceptions
{
    public class BaseHttpException : Exception
    {
        public static BaseHttpException Create(string message, HttpStatusCode
        errorCode)
        {
            return errorCode switch
            {
                HttpStatusCode.Conflict => new AlreadyExistsException(message),
                HttpStatusCode.BadRequest => new BadRequestException(message),
                HttpStatusCode.NotFound => new NotFoundException(message),
                HttpStatusCode.InternalServerError => new
                ServerInnerException(message),
                _ => new ("{errorCode:G}: {message}", errorCode)
            };
        }

        protected BaseHttpException(string message, HttpStatusCode errorCode)
            : base(message)
        {
            ErrorCode = errorCode;
        }

        protected BaseHttpException(string message, Exception innerException,
        HttpStatusCode errorCode)
            : base(message, innerException)
        {
            ErrorCode = errorCode;
        }

        public HttpStatusCode ErrorCode { get; }
    }
}

using System;
using System.Net;

namespace Common.Exceptions
{
    public sealed class NotFoundException : BaseHttpException
    {
        public NotFoundException(string message)
            : base(message, HttpStatusCode.NotFound)
        {
        }

        public NotFoundException(string message, Exception innerException)
            : base(message, innerException, HttpStatusCode.NotFound)
        {
        }
    }
}

```

```

using System;
using System.Net;

namespace Common.Exceptions
{
    public sealed class ServerInnerException : BaseHttpException
    {
        public ServerInnerException(string message)
            : base(message, HttpStatusCode.InternalServerError)
        {
        }

        public ServerInnerException(string message, Exception innerException)
            : base(message, innerException, HttpStatusCode.InternalServerError)
        {
        }
    }
}

using AutoMapper;

namespace Common.Extensions
{
    public static class AutomapperExtensions
    {
        public static void AssertConfigurationIsValid(this Profile profile)
        {
            var profileCopy = profile;
            var cfg = new MapperConfiguration(x => x.AddProfile(profileCopy));

            cfg.AssertConfigurationIsValid();
        }
    }
}

namespace Common.Extensions
{
    public static class DoubleExtensions
    {
        private const double DoubleComparisonPrecision = 0.000001;

        public static bool ApproximatelyEquals(this double source, int other)
        {
            return other - DoubleComparisonPrecision <= source
                && source <= other + DoubleComparisonPrecision;
        }
    }
}

using System.Collections.Generic;

namespace Common.Extensions
{
    public static class EnumerableExtensions
    {
        public static string JoinString(this IEnumerable<string> strings, string
            separator = ", ")
            => string.Join(separator, strings);
    }
}

using System;

```

```

using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using Common.Exceptions;
using Common.Models;
using Common.Resources;
using Newtonsoft.Json;

namespace Common.Extensions
{
    public static class HttpClientExtensions
    {
        private const string JsonMediaType = "application/json";

        public static async Task InternalPost(this HttpClient httpClient, string url,
            object? body = null)
        {
            var message = new HttpRequestMessage
            {
                Content = body is null
                    ? null
                    : new StringContent(JsonConvert.SerializeObject(body),
Encoding.UTF8, JsonMediaType),
                Method = HttpMethod.Post,
                RequestUri = new (url, UriKind.Relative)
            };

            await httpClient.SendInternalRequest(message).ConfigureAwait(false);
        }

        public static async Task<TResponse> InternalPost<TResponse>(this HttpClient
            httpClient, string url, object? body = null)
        {
            var message = new HttpRequestMessage
            {
                Content = body is null
                    ? null
                    : new StringContent(JsonConvert.SerializeObject(body),
Encoding.UTF8, JsonMediaType),
                Method = HttpMethod.Post,
                RequestUri = new (url, UriKind.Relative)
            };

            return await
            httpClient.SendInternalRequest<TResponse>(message).ConfigureAwait(false);
        }

        public static async Task InternalPut<TBody>(this HttpClient httpClient, string
            url, TBody body)
        {
            var message = new HttpRequestMessage
            {
                Content = new StringContent(JsonConvert.SerializeObject(body),
Encoding.UTF8, JsonMediaType),
                Method = HttpMethod.Put,
                RequestUri = new (url, UriKind.Relative)
            };

            await httpClient.SendInternalRequest(message).ConfigureAwait(false);
        }

        public static async Task InternalDelete(this HttpClient httpClient, string url)
    }
}

```

```

    {
        var message = new HttpRequestMessage
        {
            Method = HttpMethod.Delete,
            RequestUri = new (url, UriKind.Relative)
        };

        await httpClient.SendInternalRequest(message).ConfigureAwait(false);
    }

    public static async Task<TResponse> InternalGet<TResponse>(this HttpClient
httpClient, string url)
    {
        var message = new HttpRequestMessage
        {
            Method = HttpMethod.Get,
            RequestUri = new (url, UriKind.Relative)
        };

        return await
httpClient.SendInternalRequest<TResponse>(message).ConfigureAwait(false);
    }

    public static async Task InternalGet(this HttpClient httpClient, string url)
    {
        var message = new HttpRequestMessage
        {
            Method = HttpMethod.Get,
            RequestUri = new (url, UriKind.Relative)
        };

        await httpClient.SendInternalRequest(message).ConfigureAwait(false);
    }

    public static async Task<T?> ParseNullableBodyAsync<T>(this HttpResponseMessage
response)
    {
        return await ParseNullableJsonBodyAsync<T>(response).ConfigureAwait(false);
    }

    public static async Task<T> ParseBodyAsync<T>(this HttpResponseMessage
response)
    {
        return await ParseJsonBodyAsync<T>(response).ConfigureAwait(false);
    }

    private static async Task<T> SendInternalRequest<T>(this HttpClient httpClient,
HttpRequestMessage message)
    {
        var response = await SendInternalRequest(httpClient,
message).ConfigureAwait(false);
        return await ParseJsonBodyAsync<T>(response).ConfigureAwait(false);
    }

    private static async Task<HttpResponseBody> SendInternalRequest(this
HttpClient httpClient, HttpRequestMessage message)
    {
        var response = await httpClient.SendAsync(message).ConfigureAwait(false);

        if (!response.IsSuccessStatusCode)
        {
            var error = await

```



```

ParseNullableJsonBodyAsync<ErrorResponse>(response).ConfigureAwait(false);
        throw BaseHttpException.Create(
            error?.ErrorMessage ?? string.Empty,
            response.StatusCode);
    }

    return response;
}

private static async Task<T> ParseJsonBodyAsync<T>(HttpResponseBodyMessage
response)
{
    var result = await
ParseNullableJsonBodyAsync<T>(response).ConfigureAwait(false);
    if (result is null)
    {
        var detailedMessage = string.Format(
            Localization.CanNotParseBody,
            response.RequestMessage?.RequestUri,
            response.Content.Headers.ContentType,
            typeof(T));

        throw new ServerInnerException(Localization.ServerInternalDefaultError,
            new JsonException(detailedMessage));
    }

    return result;
}

private static async Task<T?> ParseNullableJsonBodyAsync<T>(HttpResponseBodyMessage
response)
{
    var stringBody = await
response.Content.ReadAsStringAsync().ConfigureAwait(false);

    if ((!response.Content.Headers.ContentType?.MediaType?.Contains("json")) ??
false)
    {
        throw new ServerInnerException(
            $"Auth0 response not in json format:
{response.Content.Headers.ContentType?.MediaType}, body: {stringBody}");
    }

    try
    {
        return JsonConvert.DeserializeObject<T>(stringBody);
    }
    catch (JsonReaderException e)
    {
        throw new JsonException($"Can't parse {stringBody}", e);
    }
}
}

using System;
using System.Globalization;
using Common.Exceptions;
using Common.Resources;
using Newtonsoft.Json;

namespace Common.Json

```

```

{
    public sealed class TimeSpanJsonConverter : JsonConverter<TimeSpan>
    {
        public override void WriteJson(JsonWriter writer, TimeSpan value,
            JsonSerializer serializer)
        {
            writer.WriteValue(value.ToString(DateTimeFormats.TimeSpanJsonFormat));
        }

        public override TimeSpan ReadJson(JsonReader reader, Type objectType, TimeSpan
            existingValue, bool hasExistingValue,
            JsonSerializer serializer)
        {
            var stringValue = (string)reader.Value;

            if (string.IsNullOrEmpty(stringValue)
                || !TimeSpan.TryParseExact(stringValue,
                    DateTimeFormats.TimeSpanJsonFormat,
                    CultureInfo.InvariantCulture, out var time))
            {
                throw new
                BadRequestException(string.Format(Localization.IncorrectTimeFormat, stringValue,
                    DateTimeFormats.TimeSpanJsonFormat));
            }

            return time;
        }
    }
}

namespace Common.Models
{
    public sealed class ErrorResponse
    {
        public string ErrorMessage { get; set; }
    }
}

namespace Common
{
    public static class DateTimeFormats
    {
        public const string TimeSpanJsonFormat = @"d'.'hh':'mm':'ss";
    }
}

using System;

namespace Db.Common.Models
{
    public interface IDbCollectionModel : IDbModel
    {
        Guid GroupGuid { get; }
    }
}

using System;

namespace Db.Common.Models
{
    public interface IDbModel
    {

```

```

        Guid Guid { get; }
    }
}

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Db.Common.Models;

namespace Db.Common.Repositories.Contracts
{
    public interface IBaseCollectionRepository<TModel> : IBaseRepository<TModel>
        where TModel : IDbCollectionModel
    {
        Task<IEnumerable<TModel>> GetGroupAsync(Guid groupGuid);

        Task<bool> IsGroupExistsAsync(Guid groupGuid);

        Task UpdateGroupAsync(IEnumerable<TModel> models);

        Task DeleteGroupAsync(Guid groupGuid);
    }
}

using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using System.Threading.Tasks;
using Db.Common.Models;

namespace Db.Common.Repositories.Contracts
{
    public interface IBaseRepository<TModel>
        where TModel : IDbModel
    {
        Task CreateAsync(TModel model);

        Task CreateAsync(IEnumerable<TModel> models);

        Task<TModel> GetAsync(Guid guid);

        Task<IEnumerable<TModel>> GetAsync();

        Task<IEnumerable<TModel>> GetAsync(Expression<Func<TModel, bool>> query);

        Task<bool> IsExistsAsync(Guid guid);

        Task<bool> IsExistsAsync(Expression<Func<TModel, bool>> query);

        Task UpdateAsync(TModel model);

        Task DeleteAsync(Guid guid);

        Task DeleteAsync(Expression<Func<TModel, bool>> query);
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Db.Common.Models;

```

```

using Db.Common.Repositories.Contracts;
using Microsoft.EntityFrameworkCore;

namespace Db.Common.Repositories
{
    public abstract class BaseCollectionRepository<TModel> : BaseRepository<TModel>,
        ICollectionRepository<TModel>
        where TModel : class, IDbCollectionModel
    {
        protected BaseCollectionRepository(DbContext dbContext, DbSet<TModel> dbSet) :
            base(dbContext, dbSet) { }

        public async Task<IEnumerable<TModel>> GetGroupAsync(Guid groupGuid)
        {
            return await GetAsync(s => s.GroupGuid == groupGuid);
        }

        public async Task<bool> IsGroupExistsAsync(Guid groupGuid)
        {
            return await IsExistsAsync(s => s.GroupGuid == groupGuid);
        }

        public async Task UpdateGroupAsync(IEnumerable<TModel> models)
        {
            models = models.ToList();

            var groupGuid = models.First().GroupGuid;
            var oldModels = await DbSet.Where(s => s.GroupGuid ==
groupGuid).ToListAsync();
            DbContext.RemoveRange(oldModels);

            DbSet.AddRange(models);

            await DbContext.SaveChangesAsync().ConfigureAwait(false);
        }

        public async Task DeleteGroupAsync(Guid groupGuid)
        {
            await DeleteAsync(s => s.GroupGuid == groupGuid);
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Threading.Tasks;
using Db.Common.Models;
using Db.Common.Repositories.Contracts;
using Microsoft.EntityFrameworkCore;

namespace Db.Common.Repositories
{
    public abstract class BaseRepository<TModel> : IBaseRepository<TModel> where TModel
        : class, IDbModel
    {
        protected readonly DbContext DbContext;
        protected readonly DbSet<TModel> DbSet;

        protected BaseRepository(DbContext dbContext, DbSet<TModel> dbSet)
        {

```

```

        DbContext = dbContext;
        DbSet = dbContext;
    }

    public async Task CreateAsync(TModel model)
    {
        DbSet.Add(model);
        await DbContext.SaveChangesAsync().ConfigureAwait(false);
    }

    public async Task CreateAsync(IEnumerable<TModel> models)
    {
        DbSet.AddRange(models);
        await DbContext.SaveChangesAsync().ConfigureAwait(false);
    }

    public async Task<TModel> GetAsync(Guid guid)
    {
        return await DbSet.SingleOrDefaultAsync(s => s.Guid ==
guid).ConfigureAwait(false);
    }

    public async Task<IEnumerable<TModel>> GetAsync()
    {
        return await DbSet.ToListAsync().ConfigureAwait(false);
    }

    public async Task<IEnumerable<TModel>> GetAsync(Expression<Func<TModel, bool>>
query)
    {
        return await DbSet.Where(query).ToListAsync().ConfigureAwait(false);
    }

    public async Task<bool> IsExistsAsync(Guid guid)
    {
        return await DbSet.AnyAsync(s => s.Guid == guid).ConfigureAwait(false);
    }

    public async Task<bool> IsExistsAsync(Expression<Func<TModel, bool>> query)
    {
        return await DbSet.AnyAsync(query).ConfigureAwait(false);
    }

    public async Task UpdateAsync(TModel model)
    {
        DbSet.Update(model);
        await DbContext.SaveChangesAsync().ConfigureAwait(false);
    }

    public async Task DeleteAsync(Guid guid)
    {
        var model = await DbSet.SingleOrDefaultAsync(s => s.Guid == guid);
        DbContext.Remove(model);
        await DbContext.SaveChangesAsync().ConfigureAwait(false);
    }

    public async Task DeleteAsync(Expression<Func<TModel, bool>> query)
    {
        var models = await DbSet.Where(query).ToListAsync();
        DbContext.RemoveRange(models);
        await DbContext.SaveChangesAsync().ConfigureAwait(false);
    }

```

```

    }
}

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Threading.Tasks;
using AutoMapper;
using Common.Exceptions;
using Microsoft.AspNetCore.Mvc;
using Front.Models;
using Front.ViewModels;
using Gateway.Common.Contracts;
using Gateway.Common.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Configuration;

namespace Front.Controllers
{
    [Route("")]
    public class HomeController : Controller
    {
        private const string WelcomePageViewed = "WelcomePageViewed";
        private const string FrontServiceName = "front";

        private readonly IRssServiceProvider _rssServiceProvider;
        private readonly IRssServiceManager _rssServiceManager;
        private readonly IMapper _mapper;
        private readonly IConfiguration _configuration;

        public HomeController(
            IRssServiceProvider rssServiceProvider,
            IRssServiceManager rssServiceManager,
            IMapper mapper,
            IConfiguration configuration)
        {
            _rssServiceProvider = rssServiceProvider;
            _rssServiceManager = rssServiceManager;
            _mapper = mapper;
            _configuration = configuration;
        }

        [HttpGet]
        [Route("")]
        public IActionResult Index()
        {
            return IsWelcomePageViewed()
                ? RedirectToAction("ShowNews")
                : RedirectToAction("ShowAboutPage");
        }

        [HttpGet]
        [Route("rss-source")]
        public async Task<IActionResult> ShowRssSourcesAsync()
        {
            var rssSources = await
                _rssServiceProvider.GetSubscriptionsAsync().ConfigureAwait(false);
            var viewModels =
                _mapper.Map<ICollection<RssSubscriptionViewModel>>(rssSources);
            return View("RssSources", viewModels);
        }
    }
}

```

```

[HttpGet]
[Route("about")]
public IActionResult ShowAboutPageAsync()
{
    if (!IsWelcomePageViewed())
    {
        SetWelcomePageViewed();
    }

    return View("About");
}

[HttpGet]
[Route("rss-source/create")]
public IActionResult ShowRssSourceCreatePageAsync()
{
    var rssSource = new RssSubscriptionViewModel { Guid = Guid.Empty };
    return View("RssSourceForm", rssSource);
}

[HttpGet]
[Route("rss-source/{guid:guid}/update")]
public async Task<IActionResult> ShowRssSourceUpdatePageAsync([FromRoute] Guid
guid)
{
    var rssSource = await _rssServiceProvider.GetSubscriptionAsync(guid);
    var viewModel = _mapper.Map<RssSubscriptionViewModel>(rssSource);
    return View("RssSourceForm", viewModel);
}

[HttpPost]
[Route("rss-source")]
public async Task<IActionResult> CreateOrUpdateRssSourceAsync([FromForm]
RssSubscriptionViewModel rssSourceViewModel)
{
    var model = _mapper.Map<RssSubscription>(rssSourceViewModel);
    try
    {
        await
        _rssServiceManager.CreateOrUpdateSubscriptionAsync(model).ConfigureAwait(false);
    }
    catch (BaseHttpException e)
    {
        ModelState.AddModelError(nameof(rssSourceViewModel.RssSource), "Not a
valid rss source");
        return View("RssSourceForm", rssSourceViewModel);
    }

    return RedirectToAction("ShowRssSources");
}

[HttpPost]
[Route("rss-source/{rssSourceGuid:guid}/delete")]
public async Task<IActionResult> DeleteRssSourceAsync([FromRoute] Guid
rssSourceGuid)
{
    await
    _rssServiceManager.DeleteSubscriptionAsync(rssSourceGuid).ConfigureAwait(false);
    return RedirectToAction("ShowRssSources");
}

[HttpGet]

```

```

        [Route("news")]
        public async Task<IActionResult> ShowNewsAsync()
        {
            var news = await
            _rssServiceManager.DownloadNewsAsync().ConfigureAwait(false);
            return View("News", news);
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore =
true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
        }

        private bool IsWelcomePageViewed()
            => HttpContext.Request.Cookies.ContainsKey(WelcomePageViewed);

        private void SetWelcomePageViewed()
            => HttpContext.Response.Cookies.Append(WelcomePageViewed, string.Empty, new
CookieOptions
        {
            HttpOnly = true,
            Domain = _configuration.GetServiceUri(FrontServiceName)?.ToString(),
            MaxAge = TimeSpan.MaxValue,
            SameSite = SameSiteMode.Strict
        });
    }
}

namespace Front.Models
{
    public class ErrorViewModel
    {
        public string RequestId { get; set; }

        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}

using System;
using System.ComponentModel.DataAnnotations;

namespace Front.ViewModels;

public sealed class RssSubscriptionViewModel
{
    [Required]
    public Guid Guid { get; set; }

    public string Periodicity { get; set; } = "0 8 * * *";

    [Required]
    [Display(Name = "Enable mailing", Description = "Enable sending news on email")]
    public bool NeedToSendEmails { get; set; }

    [Required]
    [Url]
    public string RssSource { get; set; }

    [RegularExpression(@"[\w\-\#]+")]

```



```

        public string Filters { get; set; } = string.Empty;

        public string Receivers { get; set; } = string.Empty;
    }

    @model Front.ViewModels.RssSubscriptionViewModel

    <form asp-action="DeleteRssSource" asp-route-rssSourceGuid="@Model.Guid" class="rss-
    source">
        <span class="rss-source_info">@Model.RssSource</span>
        <span class="rss-source_info">Mailing @(Model.NeedToSendEmails ? "on" :
        "off")</span>

        <a asp-action="ShowRssSourceUpdatePage" asp-route-guid="@Model.Guid"
        class="link__button"><i class="fas fa-edit"></i></a>
        <button type="submit" class="button__icon" ><i class="fas fa-times"></i></button>
    </form>

    @{
        ViewData["Title"] = "About";
    }

    <article class="about">
        <h1 class="about_header">It's a RssService</h1>
        <h3 class="about_description">Aggregator of RSS sources that allow you to read news
        from interesting online newspapers, filter articles and setup mailing them right
        into your email box</h3>
        <p class="about_use-case">To start using service you just need to:</p>
        <ol>
            <li class="about_use-case_action">
                <a class="link__call-to-action" asp-action="ShowRssSources">Add some RSS
                sources</a>
            </li>
            <li class="about_use-case_action">
                Set up mailing settings (if you prefer receive news on email)
            </li>
            <li class="about_use-case_action">
                And read news on email or
                <a class="link__call-to-action" asp-action="ShowNews">on this website</a>
            </li>
        </ol>
    </article>

    @model ICollection<Downloader.Common.Models.NewsItem>
    @{
        ViewData["Title"] = "News";
    }

    @if (!Model.Any())
    {
        <h1>There is no news</h1>
    }
    else
    {
        <h1 class="news-block_header">@ViewData["Title"]</h1>

        <div class="news-block">
            @foreach (var newsItem in Model)
            {
                <article>
                    <h3>@newsItem.Title</h3>
                    <time class="text__calm"

```

```

        datetime="@newsItem.PublishDate.ToString("s")">@newsItem.PublishDate.ToString("D")</
time>
        @Html.Raw(newsItem.Description)
    </article>
    }
</div>
}
}

@model Front.ViewModels.RssSubscriptionViewModel

@{
    var isSourceNew = Model.Guid == Guid.Empty;
    ViewBag.Title = isSourceNew ? "New RSS source" : "Update RSS source";
}

<h2>@ViewBag.Title</h2>

<form asp-action="CreateOrUpdateRssSource" class="rss-source-form">
    @Html.HiddenFor(m => m.Guid)

    <fieldset class="rss-source-form_field-group">
        <legend class="text__calm rss-source-form_legend">RSS source</legend>
        @Html.TextBoxFor(m => m.RssSource, new { placeholder = "Url", type = "url",
@class = "input__stretching rss-source-form_field" })
    </fieldset>

    <fieldset class="rss-source-form_field-group">
        <legend class="text__calm rss-source-form_legend">Filters</legend>
        @Html.TextBoxFor(m => m.Filters, new { placeholder = "Key words (comma
separated)", multiple = true, @class = "input__stretching rss-source-form_field" })
    </fieldset>

    <fieldset class="rss-source-form_field-group">
        <legend class="text__calm rss-source-form_legend">Mailing settings</legend>

        @Html.CheckBoxFor(m => m.NeedToSendEmails, new { @class = "rss-source-
form_field" })
        @Html.LabelFor(m => m.NeedToSendEmails, new { @class = "rss-source-form_label"
})

        <div @(Model.NeedToSendEmails ? "" : "hidden") id="MailingOptions">
            @Html.TextBoxFor(m => m.Receivers, new { placeholder = "Emails (comma
separated)", multiple = true, type = "email", @class = "input__stretching rss-
source-form_field" })

            <div class="cron-input rss-source-form_field"></div>
            @Html.HiddenFor(m => m.Periodicity, new { @class = "cron-input_value" })
        </div>
    </fieldset>

    <div class="rss-source-form_actions">
        <button type="submit">@(isSourceNew ? "Add" : "Update")</button>
        <a asp-action="ShowRssSources">Cancel</a>
    </div>
</form>

@model ICollection<Front.ViewModels.RssSubscriptionViewModel>
@{
    ViewData["Title"] = "RSS sources";
}
<h1>@ViewData["Title"]</h1>

```

```

@foreach (var source in Model)
{
    @await Html.PartialAsync("_RssSourceItem", source)
}

<a asp-action="ShowRssSourceCreatePage" class="link__button">Add rss source <i
    class="fas fa-plus"></i></a>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>@ViewData["Title"] - RssService</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css"/>
    <link rel="stylesheet" href="~/lib/jquery-cron/cron/jquery-cron.css"/>
    <link rel="stylesheet"
href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css" integrity="sha384-
AYmEC3Yw5cVb3ZcuHtOA93w35dYTsvhLPVnYs9eStHfGJvOvKxVfELGroGkvsg+p"
crossorigin="anonymous"/>
    <link rel="stylesheet" href="~/css/site.css"/>
    <link rel="stylesheet" href="~/css/custom.css"/>
</head>
<body>
<header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm border-bottom box-shadow
mb-3">
        <div class="container">
            <a class="navbar-brand navigation-bar_link" asp-area="" asp-
controller="Home" asp-action="Index">RssService</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target=".navbar-collapse" aria-controls="navbarSupportedContent"
                aria-expanded="false" aria-label="Toggle navigation">
                <i class="fas fa-bars"></i>
            </button>
            <div class="navbar-collapse collapse d-sm-inline-flex justify-content-
between">
                <ul class="navbar-nav flex-grow-1">
                    <li class="nav-item">
                        <a class="nav-link navigation-bar_link" asp-area="" asp-
controller="Home" asp-action="ShowRssSources">RSS sources</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link navigation-bar_link" asp-area="" asp-
controller="Home" asp-action="ShowNews">News</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link navigation-bar_link" asp-area="" asp-
controller="Home" asp-action="ShowAboutPage">About</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
</header>
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>

<footer class="border-top footer text__calm">

```

```

    <div class="container">
        &copy; @DateTime.Today.Year - RssService
    </div>
</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/lib/jquery-cron/cron/jquery-cron.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
</body>
</html>

$(document).ready(function() {
    if (document.querySelector('.cron-input')) {
        const valueHolder = document.querySelector('.cron-input + .cron-input_value');
        $('.cron-input').cron({
            initial: valueHolder.value,
            onChange: function() {
                valueHolder.value = $(this).cron("value");
            },
            customValues: {
                "every hour": "0 * * * *",
                "every 2 hours": "0 */2 * * *",
                "every 3 hours": "0 */3 * * *",
                "every 12 hours": "0 */12 * * *",
            }
        });
    }
});

document.querySelector('.input__stretching').addEventListener('input',
    onStretchingInput);

function onStretchingInput(){
    this.style.width = '0px';
    this.style.width = `${this.scrollWidth}px`;
}

document.querySelector('#NeedToSendEmails').addEventListener('input',
    onMailingCheckboxSwitch);

function onMailingCheckboxSwitch() {
    document.querySelector('#MailingOptions').toggleAttribute("hidden");
}

@import
    url('https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;1,100;1,300;1,400;1,500;1,700;1,900&display=swap');

:root {
    --color-background: #f1faee;
    --color-main: #1d3557;
    --color-main-faded: #1d355780;
    --color-accent: #e63946;
    --color-accent-transparent: #e6394600;
    --color-accent-calm-1: #a8dadc;
    --color-accent-calm-2: #457b9d;
    --color-accent-calm-2-transparent: #457b9d00;

    --animation-normal: 300ms;
}

```

```

body, nav {
    background: var(--color-background);
    color: var(--color-main);
    font-family: 'Roboto', sans-serif;
    box-sizing: border-box;
}

h1 {
    margin-bottom: 25px;
}

input, select {
    color: var(--color-main);
}

input:not([type="checkbox"]) {
    border: none;
    background: transparent;
    border-bottom: solid 0.1rem;
    min-width: 300px;
}

select {
    border: none;
    background: transparent;
    border-bottom: solid 0.1rem;
}

::placeholder {
    color: var(--color-accent-calm-1);
}

.text__calm {
    color: var(--color-main-faded);
}

.navigation-bar_link {
    color: var(--color-main);
    transition: text-decoration-color 0s;
}

/* links */
a, a:hover, a i {
    color: var(--color-accent-calm-2);
}

a {
    text-decoration: underline 0.15em var(--color-accent-calm-2-transparent);
    transition: text-decoration-color var(--animation-normal);
}

a:hover {
    text-decoration-color: var(--color-accent-calm-2);
}

/* icons */

i {
    color: var(--color-accent-calm-2);
    width: 1em;
    height: 1em;
    margin: 0;
}

```

```

    padding: 0;
}

/* buttons */

a.link__button {
    text-decoration: none;
    transition: text-decoration-color 0s;
}

.link__button i {
    margin-left: 0.25rem;
}

button, .button__icon, .link__button {
    background: transparent;
    color: var(--color-accent-calm-2);
    border: 0.15em solid var(--color-accent-calm-2-transparent);
    transition: border-color var(--animation-normal);
    padding: 0.3rem;
    border-radius: 0.25rem;
    line-height: 1rem;
}

button:hover, .button__icon:hover, .link__button:hover {
    border-color: var(--color-accent-calm-2);
}

/* call-to-action link */
.link__call-to-action {
    color: var(--color-accent);
    text-decoration: underline 0.15em var(--color-accent-transparent);
    transition: text-decoration-color var(--animation-normal);
    font-weight: 400;
}

.link__call-to-action:hover {
    color: var(--color-accent);
    text-decoration: underline 0.15em var(--color-accent);
}

/* about page */
.about {
    display: flex;
    flex-direction: column;
}

.about_header {
    font-size: 5rem;
    margin: 1.5rem 0;
}

.about_description {
    font-size: 1.5rem;
    font-weight: 300;
    margin-bottom: 3rem;
}

.about_use-case, .about_use-case_action {
    font-size: 2rem;
    font-weight: 300;
}

```

```

/*Rss source*/
.rss-source {
    margin-bottom: 1rem;
    display: grid;
    grid-template-columns: 1fr 100px 2rem 2rem;
    max-width: 600px;
    align-items: baseline;
}

@media (max-width: 600px) {
    .rss-source {
        grid-template-columns: auto 80px 2rem 2rem;
    }
}

.rss-source_info {
    text-overflow: ellipsis;
    white-space: nowrap;
    overflow: hidden;
}

/*Rss source form*/
.rss-source-form {
    margin-top: 2rem;
}

.rss-source-form_field-group {
    margin-bottom: 3rem;
}

.rss-source-form_actions {
    display: grid;
    grid-template-columns: 1fr 1fr;
    width: fit-content;
    column-gap: 0.5rem;
    align-items: center;
}

.rss-source-form_label {
    margin-left: 0.25rem;
}

.rss-source-form_field {
    margin-bottom: 1.5rem;
}

.rss-source-form_legend {
    margin-bottom: 1rem;
}

/* news block */

.news-block {
    display: grid;
    grid-template-columns: repeat(auto-fit, 320px);
    grid-gap: 3rem;
}

using System;
using System.Linq;
using AutoMapper;

```

```

using Common.Extensions;
using Front.ViewModels;
using Gateway.Common.Models;

namespace Front;

public sealed class MapperProfile : Profile
{
    public MapperProfile()
    {
        CreateMap<RssSubscription, RssSubscriptionViewModel>()
            .ForMember(d => d.Filters, o => o.MapFrom(s => s.Filters.JoinString(", ")))
            .ForMember(d => d.Receivers, o => o.MapFrom(s => s.Receivers.JoinString(", ")))
            .ReverseMap()
            .ForMember(d => d.Filters, o => o.MapFrom(
                s => s.Filters
                    .Split(",", StringSplitOptions.TrimEntries |
StringSplitOptions.RemoveEmptyEntries)
                    .ToArray()))
            .ForMember(d => d.Receivers, o => o.MapFrom(
                s => s.Receivers
                    .Split(",", StringSplitOptions.TrimEntries |
StringSplitOptions.RemoveEmptyEntries)
                    .ToArray()));
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace Front
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder => {
                    webBuilder.UseStartup<Startup>();
                })
    }
}

using System.Linq;
using System.Net.Http;
using System.Reflection;
using Common.Exceptions;
using Gateway.Facade;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

```



```

using Microsoft.Extensions.Hosting;

namespace Front
{
    public class Startup
    {
        private const string GatewayServiceName = "gateway-api";

        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to
        the container.
        public void ConfigureServices(IServiceCollection services)
        {
            var entryAssembly = Assembly.GetAssembly(GetType());

            services.AddControllersWithViews()
                .ConfigureApiBehaviorOptions(o =>
                {
                    o.InvalidModelStateResponseFactory =
                        context => throw new
BadRequestException(context.ModelState.Values.First().Errors.First().ErrorMessage);
                });

            services.AddGatewayProxies(Configuration.GetServiceUri(GatewayServiceName).ToString(
));

            services.AddHttpClient();
            // https://docs.microsoft.com/en-
us/dotnet/architecture/microservices/implement-resilient-applications/use-
httpclientfactory-to-implement-resilient-http-requests
            services.AddSingleton(provider =>
provider.GetService<IHttpClientFactory>().CreateClient());

            services.AddAutoMapper(entryAssembly);
        }

        // This method gets called by the runtime. Use this method to configure the
        HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
                // The default HSTS value is 30 days. You may want to change this for
                production scenarios, see https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();

```

```

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
}
}

```