Учебные задачи по C#

Данные задачи можно выполнять как в ходе обучения, так и после окончания Академии. Задачи направлены на приобретение опыта работы с популярными библиотеками.

1. Сервис по обмену файлами (easy)

Необходимо разработать сервер для сервиса по обмену файлами. На сервер можно загружать файлы, которые при этом должны получать уникальный идентификатор (и уникальную ссылку). Пользователи могут скачивать файлы по уникальным ссылкам. Файлы должны автоматически удаляться через N дней (по умолчанию - 1 день). Что должно быть реализовано:

- Загрузка файла через REST API на сервер (использовать FormData для загрузки)
- Возможность задать пароль для ручного удаления файла
- Сохранение файла на диск
- Создание уникального идентификатора для файла, запись в базу данных пути к файлу и его уникального идентификатора
- Скачивание файла по уникальному идентификатору (по ссылке, содержащей уникальный идентификатор в параметрах)
- Возможность удаления файла при предоставлении заданного при загрузке файла пароля

Технологии:

- Сервер должен быть сделан на ASP.NET
- База данных - PostgreSQL с использованием ORM
- Для создания hash использовать Argon или Scrypt
- Тестировать работу API можно через Postman
- Для создания уникальных идентификаторов использовать CUID
- Для реализации автоматического удаления файлов можно использовать CRON (какую-либо из его реализаций для C#)

2. Сервис по созданию short-ссылок (easy)

Необходимо разработать сервер для сервиса по созданию коротких ссылок (аналог https://bit.ly и подобных сервисов). Нужно генерировать короткую ссылку с использованием CUID и сохранять ее в базу данных. При переходе по ссылке пользователь должен попадать на страницу, указанную при генерации короткой ссылки. Что необходимо реализовать:

- API для генерации коротких ссылок на основе предоставленной пользователем ссылки
- Возможность задания пароля для удаления сгенерированной ссылки (пароль задается при создании, хранится в виде hash-суммы в базе данных)
- API для переадресации пользователя с короткой ссылки на оригинальную
- API для удаления ссылки (требует наличие пароля, заданного при создании ссылки)
- Также требуется написать unit-тесты для контроллеров

Технологии:

- MediatR (в контроллере должен использоваться CQRS)
- ASP.NET
- MongoDB + ORM
- CUID
- Redis для кеширования данных из MongoDB (кешированные данные должны храниться в оперативной памяти ограниченное время)

3. Подсчет размера файлов на диске (easy)
Необходимо разработать консольное приложение на C#, которое считает общий размер файлов указанного формата, хранящихся на диске. При запуске приложения пользователю необходимо указать начальный путь, откуда будет начинаться поиск файлов, а также указать формат файла (например jpeg). Приложение должно создать текстовой файл с отчетом, в котором будет указан общий размер всех найденных файлов.
На что необходимо обратить внимание:
- должна использоваться многопоточность
- не должно быть состояния гонок при сложении размера найденного файла и переменной-накопителя
- результат должен быть записан в файл один раз в конце работы приложения
- проход по файловой системе должен быть рекурсивным
- необходимо, чтобы одновременно могли работать только N потоков, где N - переменная окружения, которую можно указать при запуске приложения (т. е. необходимо использовать pool потоков)
- рекомендуется выполнить данную задачу без использования сторонних модулей

4. Регистрация, аутентификация, авторизация (medium)
Необходимо написать сервер, предоставляющий API для регистрации и аутентификации пользователей. Авторизация пользователей должна производиться при помощи авторизационного Middleware. Авторизованный запрос должен позволять изменять данные пользователя. Запрос без авторизации должен позволять получать список пользователей. Что нужно сделать:
- Разработать API для регистрации пользователей (first name, last name, email, password)
- Пароль хранить в отдельной таблице в виде hash-суммы
- Разработать API для аутентификации (login)
- При аутентификации выдавать Access-токен и Refresh-токен
- Хранить Refresh-токены в базе (один пользователь может иметь много RT)
- Разработать API для обновления истекшего Access-токена (и всю сопутствующую логику)
- Разработать API для получения списка пользователей (не требует авторизации при помощи Access-токена)
- Разработать Middleware для авторизации входящих запросов

(Access-токен должен доставаться из Authorization-заголовка HTTP-запроса)
- Разработать API для изменения данных пользователя, требующий авторизацию (пользователь должен иметь возможность изменять только свои собственные данные)
- Использовать Redis для кэширования данных, реализовать логику по инвалидации кеша (кешировать список пользователей, очищать кэш при добавлении пользователя либо при изменении пользователем своих данных)
- Разработать API для удаления пользователя (с авторизацией - пользователь может удалить только свой собственный аккаунт)
- Переменные окружения должны браться из .env-файла, лежащего на диске
- Написать тесты для контроллеров
- Написать документацию к серверу (как разворачивать, как собирать, как запускать, какие Environment Variables необходимы)
- Обернуть в Docker-контейнер

Технологии:
- ASP.NET
- PostgreSQL + ORM, либо RAW запросы
- JWT (использовать какую-либо JWT-библиотеку для C# / .NET)
- Redis
- Argon / Scrypt / Bcrypt
- Dotenv.net (https://github.com/bolorundurowb/dotenv.net) или аналоги
- Docker

5. Real-time events (medium)

Необходимо разработать 2 приложения: клиент и сервер. Сервер представляет собой Connection Hub, в который подключаются клиентские приложения. Клиентские приложения должны иметь возможность обмениваться сообщениями в реальном времени, при этом сообщения должны проходить через Hub и сохраняться в базу данных. Сервер должен предоставлять возможность регистрации клиентов, а также должен производить аутентификацию и авторизацию сообщений. Что необходимо сделать:
- приложение-сервер, работающее с подключенными клиентами
- приложение-клиент, которое может подключаться к серверу и отправлять сообщения, а также получать сообщения в реальном времени
- базу данных, в которой хранятся сообщения

Технологии:
- ASP.NET
- PostgreSQL
- EF Core
- SignalR
- Docker (опционально)

6. Разбиение PDF на страницы (medium)

Необходимо разработать микросервис, разбивающий загруженный файл на отдельные страницы (форматы страниц - JPEG / PNG). При разбиении страницы должны сохраняться на диск, после чего должны быть запакованы в один архивный файл (gzip / zip / rar). Пользователю должна предоставляться прямая ссылка на скачивание созданного архива с изображениями. Что необходимо сделать:
- приложение-сервер, имеющее API для загрузки PDF файла и разбиения его на отдельные страницы
- добавить возможность передачи дополнительных параметров: формат изображения, размер изображения (опционально), DPI (опционально)
- загрузка файла на сервер должна осуществляться через FormData
- размеры PDF должны быть ограничены: 10 MB
- при разбиении PDF необходимо использовать все доступные ядра / потоки для ускорения процесса разбиения
- имена файлов должны идти по порядку (page1, page2, page3, …)
- после создания архива сохраненные на диске изображения должны быть удалены
Технологии:
- ASP.NET Core
- System.IO.Compression (либо сторонний модуль, если будут сложности при работе со стандартным)
- Модуль для разбиения PDF - выбрать любой, позволяющий использовать многопоточность

7. Биржа (medium)
Необходимо разработать 2 микросервиса:
- первый микросервис должен получать каждые 5 секунд от стороннего API данные о стоимости акций 10-ти крупных компаний и писать эти данные их в RabbitMQ
- второй сервис должен читать сообщения из RabbitMQ по мере их поступления и писать их в базу MongoDB
Также необходимо настроить GraphQL на втором микросервисе, для гибкой отдачи данных из базы данных в клиентское приложение. Что требуется сделать:
- Микросервис для чтения данных из выбранного источника (источник данных - какое-либо API, возвращающее данные об акциях, которое можно выбрать тут: https://github.com/public-apis/public-apis)
- Микросервис для сохранения данных в MongoDB
- Подключить GraphQL для удобства клиентской части
- Обернуть оба микросервиса в Docker-контейнеры (RabbitMQ - по возможности тоже)
Технологии:
- ASP.NET
- MongoDB + ORM
- RabbitMQ (https://habr.com/ru/post/649915/)
- GraphQL (https://github.com/graphql-dotnet/graphql-dotnet) либо

аналоги
- Docker / Docker-Compose

8. Планировщик (medium)
Необходимо разработать 2 микросервиса, которые будут взаимодействовать между собой при помощи RabbitMQ:
- первый микросервис должен каждые 10 секунд генерировать сообщение (нужно брать данные из какого-нибудь публичного апи, например, weather reports, можно выбрать любой из https://github.com/public-apis/public-apis) и отправлять его в очередь с помощью rabbitmq
- второй микросервис должен представлять из себя планировщик, который будет каждый день в 10:00 UTC генерировать небольшой ежедневный отчет (в формате CSV) из данных, которые он получил из rabbitmq и сохранил в свою БД. Также у него должно быть API, которое позволяет скачать отчет, который можно будет открыть в Excel, за выбранный день.
- Для валидации данных во втором микросервисе нужно использовать FluentValidation.
- Оба микросервиса должны быть покрыты юнит тестами минимум на 50%
Технологии:
- ASP.NET Core
- RabbitMQ (MassTransit)
- Refit
- Quartz
- XUnit
-Moq
- Fixture
- PostgreSQL + ORM

9. Электростанции (hard)
Необходимо разработать 3 микросервиса, которые будут взаимодействовать между собой при помощи Kafka:
- первый микросервис должен каждые 5 секунд генерировать сообщение с случайно сгенерированной информацией, в сообщении должно быть:
- номер электростанции (от 1 до 9)
- количество отдаваемой электроэнергии (от 0 до 1000, может быть числом с плавающей точкой)
- единица измерения 'вт' у всех электростанций
- дата создания сообщения (created_at)
После этого сформированное сообщение отправлять в Kafka
- второй микросервис должен каждые 5 секунд генерировать сообщение со случайно сгенерированной информацией, в сообщении должно быть:
- номер электростанции (от 1 до 9)
- мощность, с которой работает электростанция в данный момент (от 1% до 100%)
- единица измерения - 'процент' для всех станций
- дата создания сообщения (created_at)
После этого сформированное сообщение отправлять в Kafka

- третий микросервис должен читать сообщения из Kafka и сохранять их в базу данных

Кроме того необходимо реализовать API для отдачи данных на клиент, должна быть возможность получить:

- данные о мощности всех или только выбранных электростанций, должна быть возможность выбрать промежуток: день / месяц / год для получаемых данных
- данные о отдаваемой электроэнергии всех или только выбранных электростанций, должна быть возможность выбрать промежуток: день / месяц / год для получаемых данных

Технологии:

- ASP.NET
- Kafka (https://www.instaclustr.com/support/documentation/kafka/accessing-and-using-kafka/use-kafka-with-c-sharp)
- PostgreSQL + ORM