



Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Ордена Трудового Красного Знамени
федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

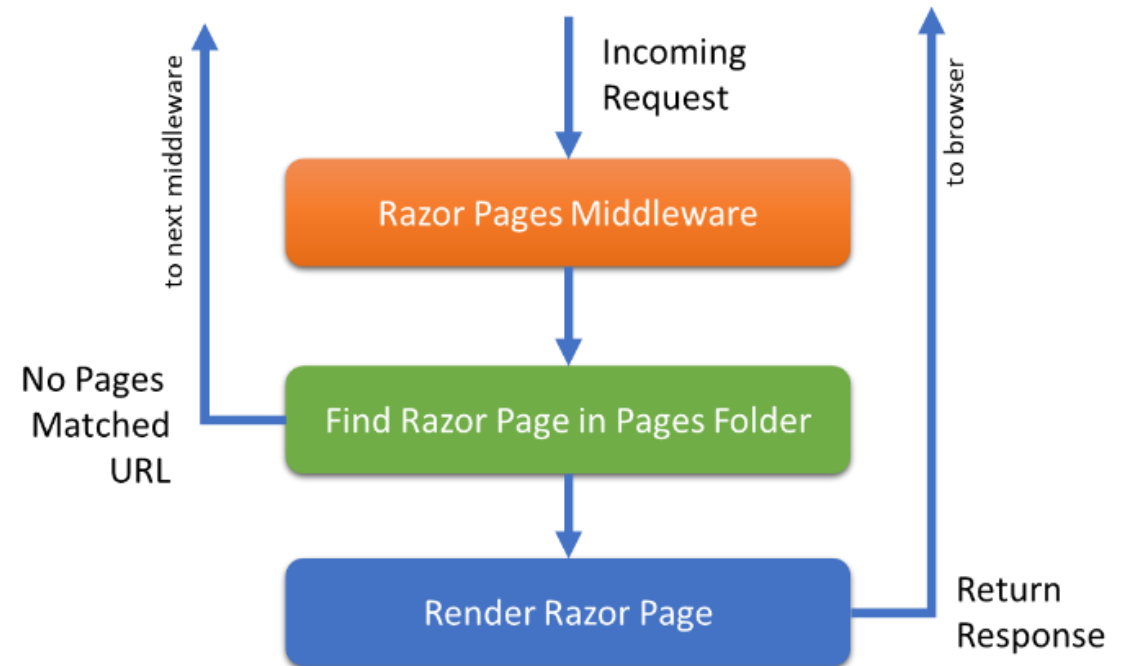
Доклад на тему

“Использование технологии Razor для создания веб-приложений и её основные
преимущества”

Горохов Кирилл Игоревич- МТУСИ, студент группы М092401(75), Москва, Россия,
studentGorokhov@gmail.com

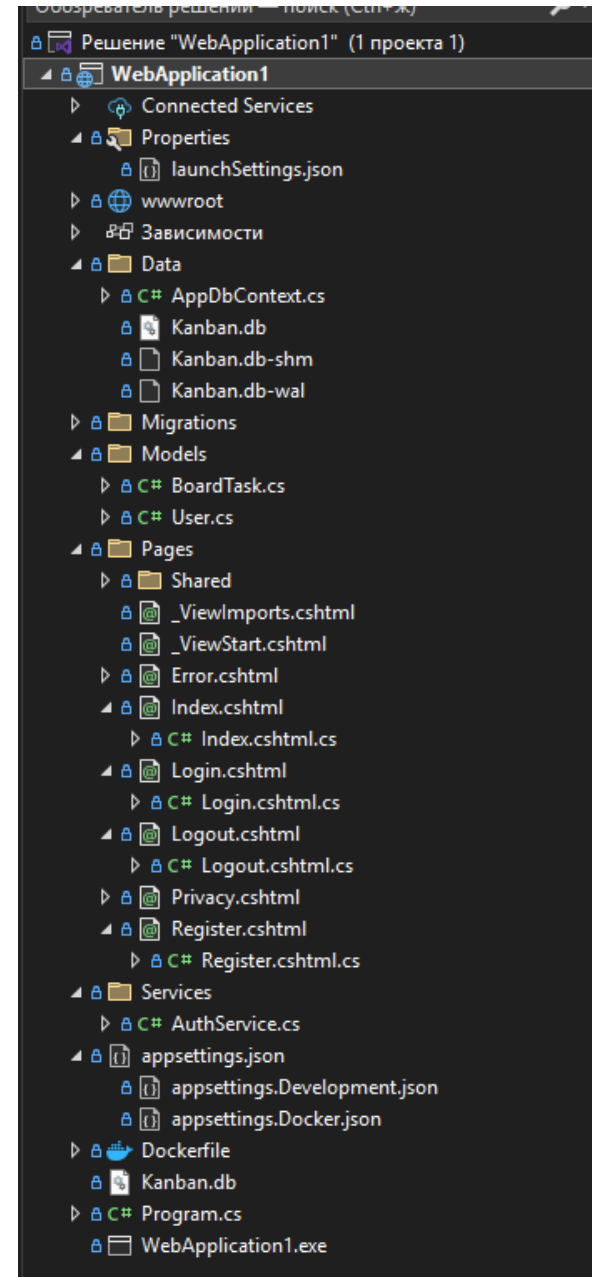
Актуальность работы

- В последние десятилетия веб-разработка стала одной из самых востребованных и динамично развивающихся областей информационных технологий. Современные пользователи предъявляют высокие требования к качеству, скорости работы и функциональности веб-приложений, что стимулирует разработчиков к поиску эффективных инструментов и технологий для создания удобных, безопасных и производительных решений. Одной из таких технологий, получивших широкое распространение в экосистеме .NET, является Razor — современный шаблонизатор и язык разметки, позволяющий создавать динамические веб-страницы с использованием синтаксиса C# непосредственно внутри HTML-кода.



Архитектура Razor Pages

В основе архитектуры Razor Pages лежит пара файлов: файл разметки с расширением `.cshtml` и связанный с ним файл модели страницы с расширением `.cshtml.cs`. Файл `.cshtml` содержит HTML-код с интеграцией C# через синтаксис Razor, а файл модели страницы (PageModel) инкапсулирует серверную логику, обработку запросов, работу с данными и сервисами. По соглашению, класс модели страницы называется так же, как и страница, с добавлением суффикса `Model`. Например, для страницы `Create.cshtml` соответствующий класс будет называться `CreateModel` и находиться в том же пространстве имён, что и страница.



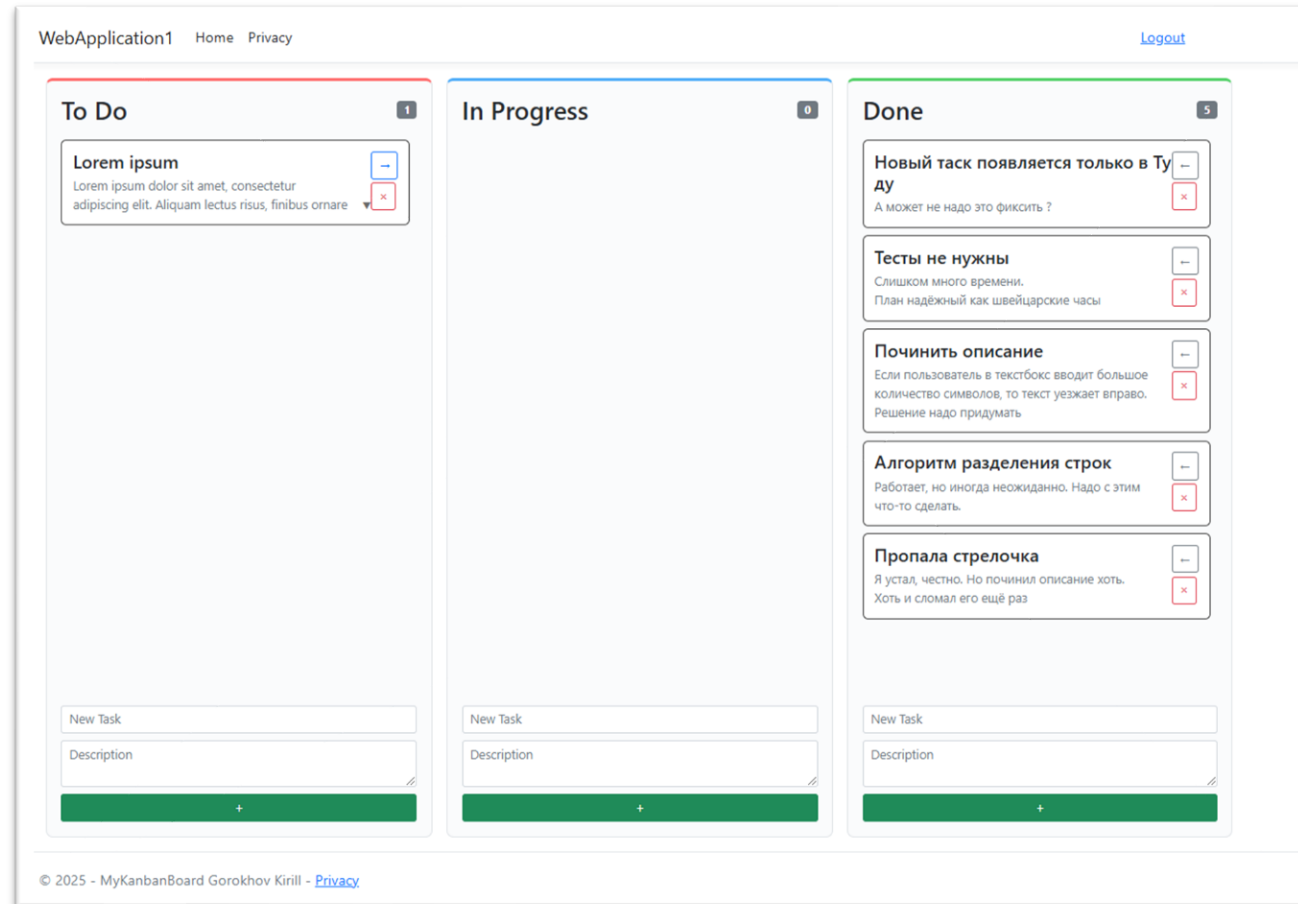
Ключевые особенности Razor Pages

В отличие от классической архитектуры MVC, где контроллеры могут становиться слишком объемными и сложными из-за необходимости обработки множества действий для разных частей приложения, Razor Pages предлагает более изолированную и модульную структуру. Каждая страница отвечает только за свой функционал, что уменьшает связанность кода и снижает вероятность появления ошибок при масштабировании или изменении приложения. Такой подход особенно эффективен для проектов с относительно простой логикой или небольшим количеством уникальных страниц, а также для корпоративных внутренних систем, где требуется быстрое внедрение и поддержка новых функций.

Еще одной архитектурной особенностью Razor Pages является возможность использования двухстороннего связывания данных (two-way data binding), что сближает подход с паттерном MVVM (Model-View-ViewModel). Это позволяет не только передавать данные от сервера к клиенту, но и легко обрабатывать обратную связь от пользователя, например, при отправке форм или изменении состояния элементов интерфейса. Такой механизм особенно полезен при работе с интерактивными элементами, где требуется мгновенное обновление данных на странице без необходимости полной перезагрузки.

Описание проекта

Проект представляет собой веб-приложение для управления задачами, реализующее базовые принципы методологии Kanban. Система сочетает функциональность трекинга задач, авторизацию пользователей и взаимодействие с реляционной базой данных. Архитектура приложения следует принципам разделения ответственности, что обеспечивает масштабируемость и поддерживаемость кода. Система предусматривает регистрацию и авторизацию пользователей через механизм cookie-сессий.



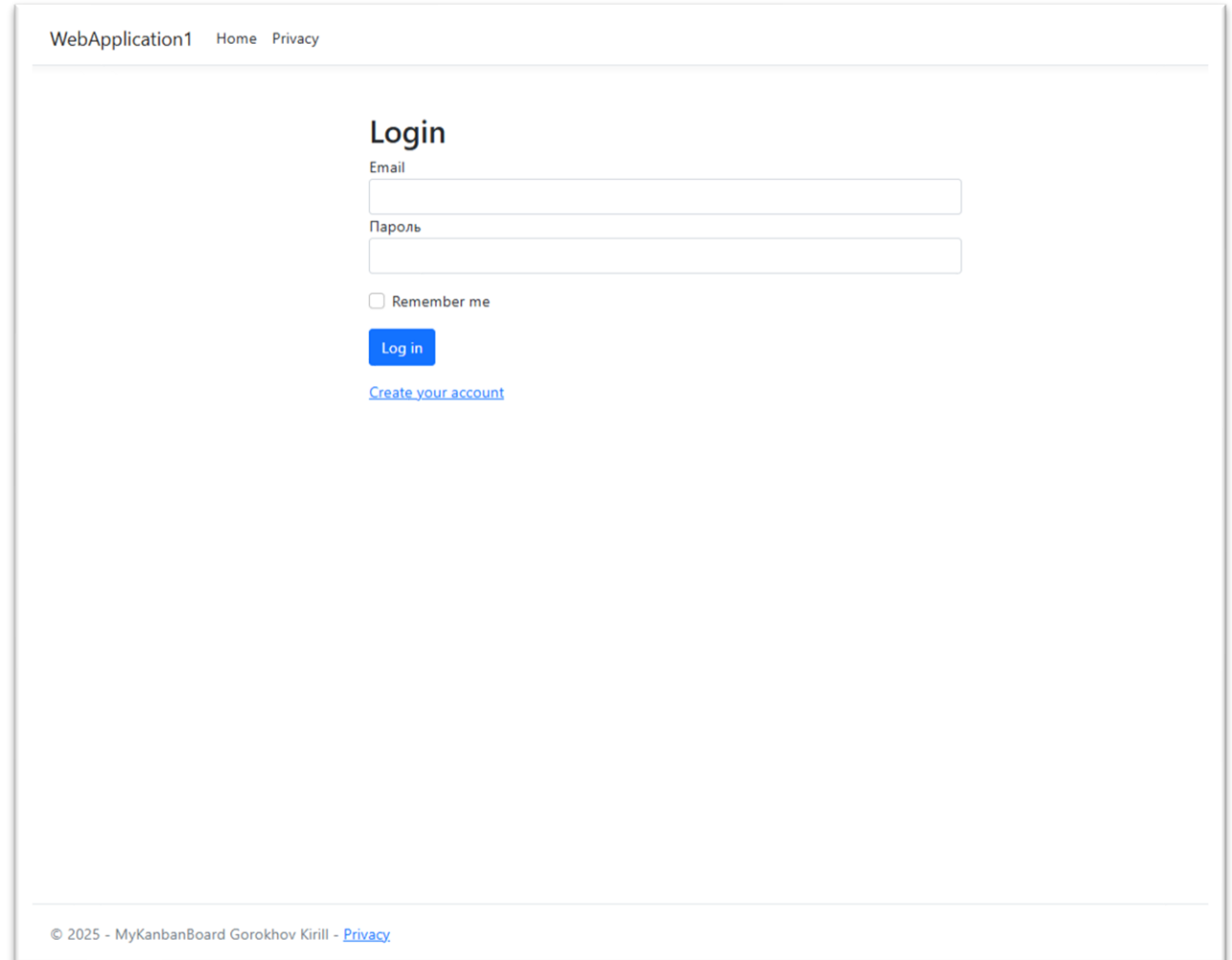
Авторизация

Модель пользователя включает следующие атрибуты:

Идентификатор (ID): Уникальный ключ, обычно реализуемый как UUID или автоинкрементное целое число.

- 1) Логин: Уникальное имя пользователя для входа в систему.
- 2) Хэш пароля: Безопасное хранение учетных данных с использованием алгоритмов хеширования.
- 3) Электронная почта: для восстановления доступа и уведомлений.
- 4) Дата регистрации: Отслеживание времени создания аккаунта.

Процесс аутентификации реализуется через промежуточное ПО (middleware), проверяющее валидность cookie при каждом запросе. После успешного входа система генерирует сессионный токен, который шифруется и сохраняется в cookie браузера



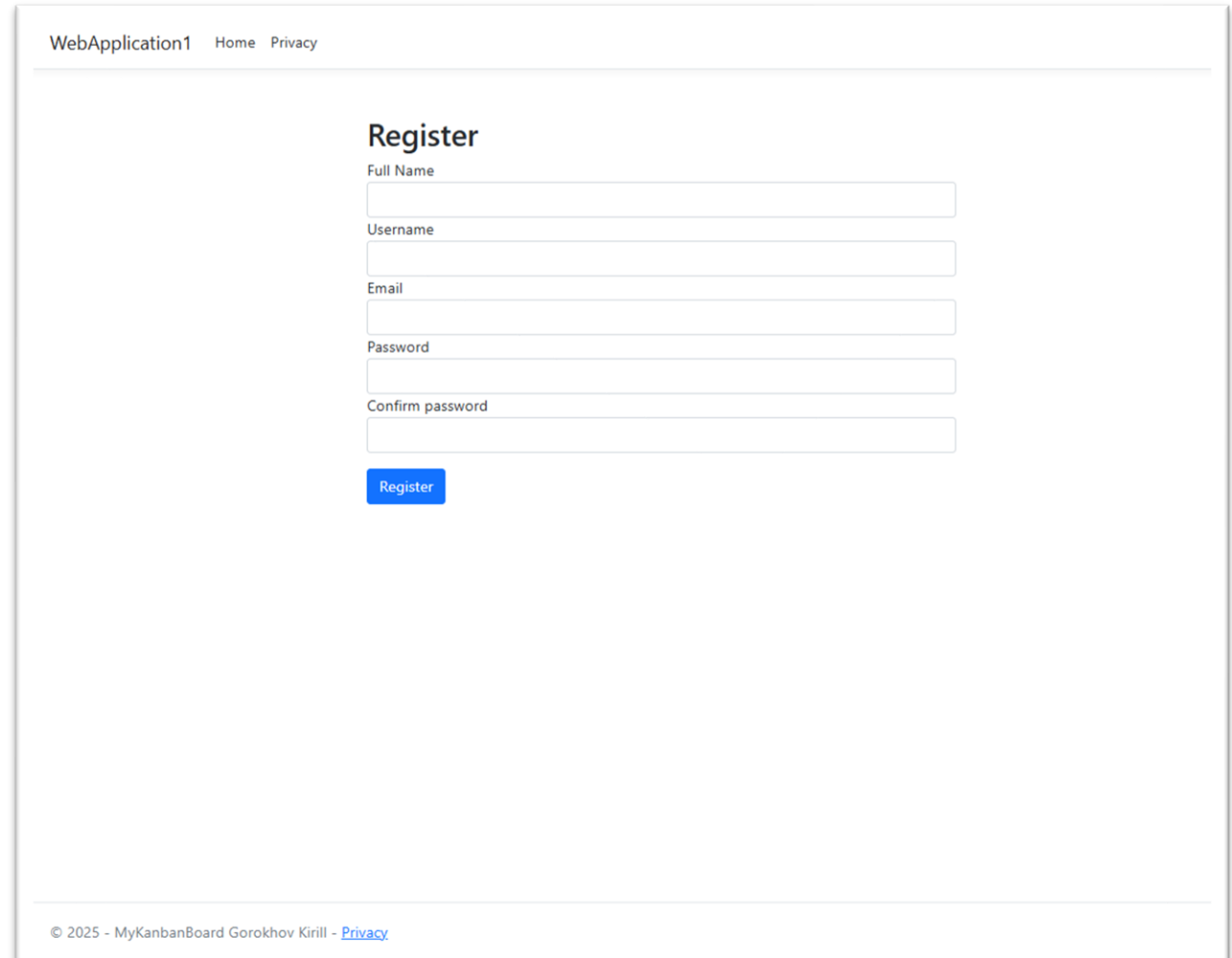
The screenshot shows a web application interface for a login page. At the top, there is a navigation bar with the text "WebApplication1" and links for "Home" and "Privacy". The main content area is titled "Login" and contains two input fields: "Email" and "Пароль" (Password). Below the password field is a checkbox labeled "Remember me". A blue "Log in" button is positioned below the checkbox. At the bottom of the login section, there is a link that says "Create your account". The footer of the page contains the copyright notice "© 2025 - MyKanbanBoard Gorokhov Kirill - Privacy" with a link to the privacy policy.

Регистрация

Метод `Register` выполняет регистрацию нового пользователя. Он проверяет уникальность email в базе данных через `_context.Users.AnyAsync(u => u.Email == user.Email)`. Если email занят, генерируется исключение с сообщением "Email is already taken".

Для безопасного хранения пароля вызывается метод `CreatePasswordHash`, который создает хэш и соль. Эти значения присваиваются свойствам `user.PasswordHash` и `user.PasswordSalt`. Пользователь добавляется в базу через `_context.Users.Add(user)`, а изменения сохраняются асинхронно с помощью `SaveChangesAsync()`.

В результате метод возвращает зарегистрированного пользователя.



The screenshot shows a web application interface for a registration form. At the top, there is a navigation bar with the text "WebApplication1" and links for "Home" and "Privacy". The main content area is titled "Register" and contains five input fields: "Full Name", "Username", "Email", "Password", and "Confirm password". Below these fields is a blue button labeled "Register". At the bottom of the page, there is a footer with the text "© 2025 - MyKanbanBoard Gorokhov Kirill - [Privacy](#)".

Запуск через .NET SDK

Первый способ – использование .NET SDK. Так для того, чтобы запустить сервер необходимо пройти по пути `WebApplication1/WebApplication1` и прописать команду

- `dotnet run --environment Development`

После использования данной команды начнётся сборка проекта и применятся настройки, которые заложены под .NET SDK и откроется браузер по адресу `localhost` и выбранном порту.

```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (10ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT COUNT(*) FROM "sqlite_master" WHERE "type" = 'table' AND "rootpage" IS NOT NULL;
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (0ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT EXISTS (
        SELECT 1
        FROM "BoardTasks" AS "b")
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5232
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\frost\source\repos\WebProject\WebApplication1\WebApplication1
```


Запуск через Docker

Второй способ — это запуск сервера через Docker Compose. Для этого был разработан файл Dockerfile, а также docker-compose. В нём были прописаны параметры сборки и подключения сервисов, а также монтирование тома базы данных. Для запуска из докера, требуется перейти по пути WebApplication1/WebApplication1 и прописать команду:

- `docker compose up --build`

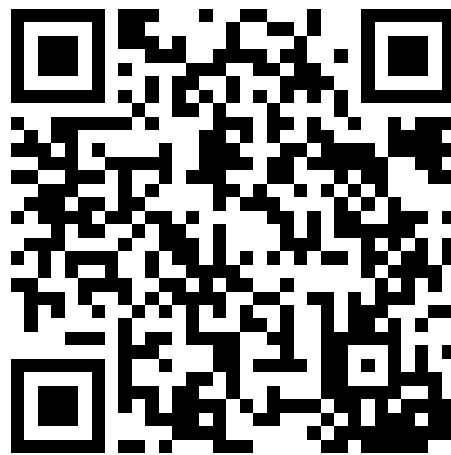
Или

- `docker-compose up --build`

```
[+] Running 2/2
✓ csharp-webapplication1 Built
✓ Container csharp-webapplication1 Recreated
Attaching to csharp-webapplication1
csharp-webapplication1 | info: Microsoft.EntityFrameworkCore.Database.Command[20101]
csharp-webapplication1 | Executed DbCommand (18ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
csharp-webapplication1 | SELECT COUNT(*) FROM "sqlite_master" WHERE "type" = 'table' AND "rootpage" IS NOT NULL;
csharp-webapplication1 | info: Microsoft.EntityFrameworkCore.Database.Command[20101]
csharp-webapplication1 | Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
csharp-webapplication1 | SELECT EXISTS (
csharp-webapplication1 |     SELECT 1
csharp-webapplication1 |     FROM "BoardTasks" AS "b")
csharp-webapplication1 | warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]
csharp-webapplication1 | Storing keys in a directory '/home/app/.aspnet/DataProtection-Keys' that may not be persisted outside of the container. Pr
csharp-webapplication1 | warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
csharp-webapplication1 | No XML encryptor configured. Key {7e09f270-236b-4b47-90ca-00e7d07d84bc} may be persisted to storage in unencrypted form.
csharp-webapplication1 | info: Microsoft.Hosting.Lifetime[14]
csharp-webapplication1 | Now listening on: http://[::]:8080
csharp-webapplication1 | info: Microsoft.Hosting.Lifetime[0]
csharp-webapplication1 | Application started. Press Ctrl+C to shut down.
csharp-webapplication1 | info: Microsoft.Hosting.Lifetime[0]
csharp-webapplication1 | Hosting environment: Docker
csharp-webapplication1 | info: Microsoft.Hosting.Lifetime[0]
csharp-webapplication1 | Content root path: /app
```

```
PS C:\Users\frost> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
b2ac696406cb   webapplication1-csharp-webapplicati "dotnet WebApplicati..." 53 seconds ago Up 53 seconds  0.0.0.0:8080->8080/tcp, 8081/tcp    csharp-webapplication1
PS C:\Users\frost>
```

Спасибо за внимание!



Репозиторий проекта