**Руководство разработчика**

**Проект "CheckCisService"**

**Версия документа: 1.0** **Дата: 29 мая 2024 г.**

Оглавление

[Руководство разработчика 1](#_Toc198504559)

[Проект "CheckCisService" 1](#_Toc198504560)

[1. Введение 3](#_Toc198504561)

[**1.1. Обзор проекта** 3](#_Toc198504562)

[**1.2. Цель документа** 3](#_Toc198504563)

[**1.3. Целевая аудитория** 3](#_Toc198504564)

[2. Требования к окружению 4](#_Toc198504565)

[**2.1. Программное обеспечение** 4](#_Toc198504566)

[**2.2. Инструменты** 4](#_Toc198504567)

[3. Начало работы 5](#_Toc198504568)

[**3.1. Клонирование репозитория** 5](#_Toc198504569)

[**3.2. Открытие проекта в IDE** 5](#_Toc198504570)

[4. Структура проекта 6](#_Toc198504571)

[5. Сборка проекта 7](#_Toc198504572)

[6. Запуск проекта 8](#_Toc198504573)

[**6.1. Локальный запуск** 8](#_Toc198504574)

[**6.2. Параметры запуска (launchSettings.json)** 8](#_Toc198504575)

[7. База данных 9](#_Toc198504576)

[**7.1. LiteDB** 9](#_Toc198504577)

[**7.2. Файл данных** 9](#_Toc198504578)

[8. API Документация (Swagger/OpenAPI) 10](#_Toc198504579)

[9. Перечень основных классов и их назначение 11](#_Toc198504580)

[10. Ключевые технологии и библиотеки 13](#_Toc198504581)

[11. Конфигурация приложения 14](#_Toc198504582)

[**11.1. appsettings.json** 14](#_Toc198504583)

[**11.2. Properties/launchSettings.json** 15](#_Toc198504584)

[12. Архитектура приложения 17](#_Toc198504585)

[**12.1. Контроллеры (Controllers)** 17](#_Toc198504586)

[**12.2. Сервисы (Services)** 17](#_Toc198504587)

[**12.3. Репозитории (Repositories)** 17](#_Toc198504588)

[**12.4. Модели данных (Models/DTOs)** 17](#_Toc198504589)

[**12.5. Аутентификация (Basic Authentication)** 17](#_Toc198504590)

[**12.6. Вспомогательные классы (Helpers)** 17](#_Toc198504591)

[**12.7. Конфигурация (Config)** 17](#_Toc198504592)

[**12.8. Исключения (Exceptions)** 18](#_Toc198504593)

[13. Примеры вызова методов API 19](#_Toc198504594)

[**13.1. Через .http файл** 19](#_Toc198504595)

[**13.2. Через curl или Postman** 19](#_Toc198504596)

[14. Разработка и тестирование 21](#_Toc198504597)

[**14.1. Соглашения по кодированию** 21](#_Toc198504598)

[**14.2. Ветвление (Git Flow - рекомендуется)** 21](#_Toc198504599)

[**14.3. Юнит-тесты** 21](#_Toc198504600)

[15. Возможные проблемы и их решения 23](#_Toc198504601)

[16. Контакты и поддержка 24](#_Toc198504602)

# 1. Введение

**1.1. Обзор проекта**

CheckCisService – это веб-сервис, разработанный на платформе .NET 8 с использованием ASP.NET Core. Сервис предназначен для проверки кодов идентификации товаров (КИЗ) через онлайн и офлайн сервисы системы маркировки (предположительно "Честный ЗНАК" или аналогичной). Он также включает функциональность логирования запросов на проверку и базовую аутентификацию.

**1.2. Цель документа**

Данное руководство предназначено для разработчиков, которые будут участвовать в разработке, поддержке или модификации проекта CheckCisService. Документ описывает архитектуру, процесс настройки окружения, сборки, запуска и основные аспекты разработки.

**1.3. Целевая аудитория**

C#-разработчики, знакомые с ASP.NET Core, .NET CLI, Git, REST API и принципами DI.

# 2. Требования к окружению

**2.1. Программное обеспечение**

* **.NET SDK:** Версия 8.0 или выше (согласно CheckCisService.csproj).
* **Git:** Для контроля версий.

**2.2. Инструменты**

* **IDE (Интегрированная среда разработки):**
  + Visual Studio 2022 (рекомендуется, версия с поддержкой .NET 8)
  + Visual Studio Code с C# Dev Kit и другими релевантными расширениями.
* **HTTP-клиент:** Postman, Insomnia, curl или встроенные инструменты IDE (например, .http файлы в Visual Studio/VS Code).
* **Командная строка:** PowerShell, CMD, Bash или терминал в IDE.

# 3. Начало работы

**3.1. Клонирование репозитория**

Откройте терминал или командную строку и выполните следующую команду:

git clone https://github.com/DmitriyShch/CheckCisService.git

cd CheckCisService-main

**3.2. Открытие проекта в IDE**

* **Visual Studio:**
  1. Запустите Visual Studio.
  2. Выберите Open a project or solution.
  3. Перейдите в директорию CheckCisService-main и выберите файл CheckCisService.sln.
  4. Visual Studio загрузит решение и восстановит NuGet-пакеты.
* **Visual Studio Code:**
  1. Откройте VS Code.
  2. File -> Open Folder...
  3. Выберите директорию CheckCisService-main.
  4. VS Code предложит восстановить зависимости, если это необходимо.

# 4. Структура проекта

Решение (CheckCisService.sln) включает два проекта:

* CheckCisService: Основной проект веб-сервиса ASP.NET Core.
* CheckCisServiceTests: Проект с юнит-тестами для сервиса (использует xUnit).

Основная структура проекта CheckCisService:

CheckCisService/

├── Config/ # Классы конфигурации (MdlpConfig.cs)

├── Controllers/ # ASP.NET Core API контроллеры (например, CheckCisController.cs - должен быть создан)

├── Exceptions/ # Пользовательские классы исключений

├── Helpers/ # Вспомогательные классы и расширения

├── Models/ # Модели данных, DTO, Enums

├── Properties/

│ └── launchSettings.json # Настройки запуска для различных профилей

├── Repositories/ # Классы для доступа к данным (MdlpCheckCisLogRepository.cs)

├── Services/ # Классы бизнес-логики

├── BasicAuthenticationHandler.cs # Обработчик Basic аутентификации

├── CheckCisService.csproj # Файл проекта C#

├── CheckCisService.http # Примеры HTTP-запросов (для VS/VSCode)

├── Program.cs # Точка входа, настройка DI и middleware

└── appsettings.json # Основной файл конфигурации

└── appsettings.Development.json # Конфигурация для среды разработки

# 5. Сборка проекта

Для сборки проекта используйте .NET CLI или средства IDE.

* **Через .NET CLI (в директории CheckCisService-main/CheckCisService или в корневой директории решения):**
* dotnet build

Для сборки всего решения:

dotnet build CheckCisService.sln

* **Через Visual Studio:** Меню Build -> Build Solution (или F6).

Результаты сборки (DLL) будут находиться в поддиректориях bin/Debug или bin/Release соответствующих проектов.

# 6. Запуск проекта

**6.1. Локальный запуск**

* **Через .NET CLI (в директории CheckCisService-main/CheckCisService):**
* dotnet run

Сервис запустится с использованием профиля по умолчанию из launchSettings.json (обычно http или https).

* **Через Visual Studio:**
  1. Выберите стартовый проект CheckCisService.
  2. Выберите профиль запуска (например, http, https, IIS Express) в панели инструментов.
  3. Нажмите кнопку Start (зеленый треугольник) или F5.
* **Через Visual Studio Code:**
  1. Откройте Command Palette (Ctrl+Shift+P).
  2. Введите Debug: Select and Start Debugging.
  3. Выберите подходящую конфигурацию запуска (например, .NET Core Launch (web)).

По умолчанию, если используется профиль http, сервис будет доступен по адресу http://localhost:5290. Swagger UI будет доступен по http://localhost:5290/swagger.

**6.2. Параметры запуска (launchSettings.json)**

Файл CheckCisService/Properties/launchSettings.json содержит профили запуска для локальной разработки. Он определяет переменные окружения, URL-адреса приложения и другие параметры. Подробнее см. раздел 11.2.

# 7. База данных

**7.1. LiteDB**

Проект использует LiteDB в качестве встраиваемой NoSQL базы данных для хранения логов операций проверки КИЗ (MdlpCheckCisLog). LiteDB хранит данные в одном файле.

**7.2. Файл данных**

Путь к файлу базы данных настраивается в appsettings.json в секции Mdlp:DataSource:DataFilePath. По умолчанию:

"DataSource": {

"DataFilePath": "check\_cis\_log.db"

}

Файл check\_cis\_log.db будет создан в директории, из которой запускается приложение, если он не существует. Для тестов используется :memory:, что означает базу данных в памяти.

# 8. API Документация (Swagger/OpenAPI)

Проект использует Swashbuckle.AspNetCore для автоматической генерации документации API на основе кода и аннотаций контроллеров.

* **Swagger UI:** Доступен по адресу http://localhost:5290/swagger (после запуска приложения в профиле http).
* **OpenAPI JSON:** Доступен по адресу http://localhost:5290/swagger/v1/swagger.json.

Swagger UI позволяет просматривать доступные эндпоинты, их параметры, схемы запросов/ответов и тестировать API прямо из браузера.

# 9. Перечень основных классов и их назначение

* **Program.cs**:
  + Точка входа приложения.
  + Настройка и регистрация сервисов в DI-контейнере (Dependency Injection).
  + Конфигурирование конвейера обработки HTTP-запросов (middleware), включая Swagger, аутентификацию, авторизацию, маршрутизацию.
* **CheckCisService/BasicAuthenticationHandler.cs**:
  + Реализация обработчика Basic-аутентификации для ASP.NET Core. Проверяет заголовок Authorization и учетные данные из appsettings.json.
* **CheckCisService/Controllers/CheckCisController.cs** (предполагается, что такой контроллер существует или будет создан для эндпоинта /CheckCis):
  + Обрабатывает входящие HTTP-запросы, связанные с проверкой КИЗ.
  + Взаимодействует с MarkingApiService для выполнения бизнес-логики.
  + Возвращает HTTP-ответы клиенту.
* **CheckCisService/Config/MdlpConfig.cs**:
  + Содержит классы (MdlpConfig, OnlineCheckConfig, OfflineCheckConfig, DataSourceConfig) для представления структуры конфигурации из файла appsettings.json (секция "Mdlp"). Эти классы используются для типизированного доступа к настройкам.
* **CheckCisService/Exceptions/ServiceExceptions.cs**:
  + Определяет пользовательские классы исключений (ServiceException, ObjectNotExistsException), используемые в приложении для обработки специфичных ошибок.
* **CheckCisService/Helpers/CommonExtensions.cs**:
  + Содержит статические методы-расширения для общих задач, таких как сериализация в JSON, проверка строк на пустоту, манипуляции со строками и преобразование типов.
* **CheckCisService/Helpers/MdlpCashRegHelper.cs**:
  + Вспомогательный класс для работы с кодами маркировки (КИЗ). Включает логику для разбора, валидации и исправления КИЗ, SGTIN и крипто-хвостов.
* **CheckCisService/Models/**:
  + CdnListResponse.cs: Модель ответа для списка CDN-хостов.
  + CheckCisDto.cs: DTO для результата проверки КИЗ, возвращаемого API.
  + CheckCisResult.cs: Детализированный результат проверки КИЗ.
  + CodeCheckResponse.cs: Модель ответа от внешних сервисов проверки кодов.
  + IEntity.cs: Интерфейс для сущностей, имеющих Id.
  + MarkingModuleStatusDto.cs: DTO для статуса локального модуля маркировки.
  + MdlpCheckCisLog.cs: Модель (сущность) для записи лога проверки КИЗ в LiteDB.
  + Enums/MarkType.cs: Перечисление типов маркировки.
* **CheckCisService/Repositories/MdlpCheckCisLogRepository.cs**:
  + Репозиторий для взаимодействия с базой данных LiteDB. Отвечает за сохранение и, возможно, извлечение логов MdlpCheckCisLog.
* **CheckCisService/Services/MarkingApiService.cs**:
  + Основной сервис API. Координирует процесс проверки КИЗ, выбирая между онлайн и офлайн методами.
  + Управляет кэшем CDN-хостов, их активностью и блокировкой.
  + Вызывает MdlpCheckCisLogService для записи логов.
* **CheckCisService/Services/MarkingBaseService.cs**:
  + Базовый класс, содержащий общую логику для сервисов, выполняющих HTTP-запросы (например, обработка таймаутов, базовые операции с HttpClient).
* **CheckCisService/Services/MarkingOfflineService.cs**:
  + Сервис для выполнения офлайн-проверки КИЗ. Взаимодействует с локальным (офлайн) компонентом системы маркировки.
  + Также может запрашивать статус этого локального компонента и инициировать его БД.
* **CheckCisService/Services/MarkingOnlineService.cs**:
  + Сервис для выполнения онлайн-проверки КИЗ. Взаимодействует с удаленными (онлайн) CDN-серверами системы маркировки.
  + Получает список CDN-хостов и проверяет их доступность/задержку.
* **CheckCisService/Services/MdlpCheckCisLogService.cs**:
  + Сервис, отвечающий за создание и сохранение записей логов MdlpCheckCisLog с использованием MdlpCheckCisLogRepository.

# 10. Ключевые технологии и библиотеки

* **.NET 8 / ASP.NET Core 8:** Основная платформа и фреймворк для создания веб-API.
* **C#:** Язык программирования.
* **Swashbuckle.AspNetCore:** Для генерации Swagger/OpenAPI документации.
* **LiteDB:** Встраиваемая NoSQL база данных для хранения логов.
* **Newtonsoft.Json:** Библиотека для работы с JSON (хотя ASP.NET Core 8 по умолчанию использует System.Text.Json, в проекте может быть использована и Newtonsoft.Json).
* **xUnit & Moq:** Фреймворки для написания и выполнения юнит-тестов.
* **Basic Authentication:** Механизм аутентификации, реализованный в BasicAuthenticationHandler.

# 11. Конфигурация приложения

**11.1. appsettings.json**

Файл CheckCisService/appsettings.json (и его специфичные для окружения варианты, например, appsettings.Development.json) содержит конфигурацию приложения.

{

"Logging": { *// Настройки логирования*

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"BasicAuth": { *// Учетные данные для Basic Authentication*

"Username": "admin",

"Password": "1234"

},

"AllowedHosts": "\*", *// Разрешенные хосты (для Kestrel)*

"Mdlp": { *// Основная конфигурация сервиса проверки КИЗ*

"ApiKey": "61ec1bbb-211f-4289-820c-a140cb94c99a", *// API-ключ для доступа к сервисам*

"FiscalSerialNumber": "1020", *// Фискальный серийный номер по умолчанию*

"MarkGroups": [ *// Список групп маркировки и их параметры*

{

"CheckIsOwner": false,

"CrptCode": 7,

"Name": "Лекарственные препараты"

*// ... другие свойства ...*

}

*// ... другие группы ...*

],

"OfflineService": { *// Настройки для офлайн-сервиса проверки*

"ApiKey": "p1234", *// Возможно, отдельный ключ для офлайн*

"Host": "http://192.168.1.85:5995/", *// Адрес офлайн-сервиса*

"Login": "admin", *// Логин для офлайн-сервиса*

"Pwd": "1234", *// Пароль для офлайн-сервиса*

"ReqTimeout": 1500, *// Таймаут запроса в мс*

"UrlPrefix": "api/v1/" *// Префикс URL*

},

"OnlineService": { *// Настройки для онлайн-сервиса проверки*

"BlockFailedCdnPeriod": 15, *// Период блокировки CDN после ошибок (в минутах?)*

"CheckHealthReqTimeout": 2000, *// Таймаут проверки доступности CDN в мс*

"CdnHostFailedMaxCount": 2, *// Макс. ошибок на CDN перед блокировкой*

"CdnListExpiryMaxInterval": 6, *// Макс. интервал обновления списка CDN (в минутах?)*

"CdnListExpiryMinInterval": 7, *// Мин. интервал обновления списка CDN (в минутах?)*

"Host": "https://markirovka.sandbox.crptech.ru/", *// Базовый URL онлайн-сервиса*

"ReqTimeout": 1500, *// Таймаут запроса проверки КИЗ в мс*

"UrlPrefix": "api/v4/true-api/" *// Префикс URL*

},

"DataSource": { *// Настройки источника данных (для LiteDB)*

"DataFilePath": "check\_cis\_log.db" *// Путь к файлу БД*

}

}

}

* **Logging**: Стандартные настройки логирования .NET.
* **BasicAuth**: Учетные данные для Basic-аутентификации, используемые BasicAuthenticationHandler.
* **AllowedHosts**: Список хостов, которые Kestrel будет обслуживать.
* **Mdlp**: Корневая секция для конфигурации сервиса.
  + **ApiKey**: Глобальный API-ключ.
  + **FiscalSerialNumber**: Серийный номер фискального накопителя, используемый в запросах.
  + **MarkGroups**: Массив объектов, описывающих различные товарные группы, их CRPT-коды и специфические правила проверки (например, CheckIsOwner).
  + **OfflineService**: Параметры для подключения и взаимодействия с офлайн-компонентом проверки (адрес, учетные данные, таймауты).
  + **OnlineService**: Параметры для онлайн-проверки (адрес основного сервиса, таймауты, правила работы с CDN-хостами).
  + **DataSource**: Настройки для хранилища данных (здесь LiteDB), в частности путь к файлу БД.

**11.2. Properties/launchSettings.json**

Файл CheckCisService/Properties/launchSettings.json используется Visual Studio и .NET CLI (dotnet run) для настройки локального запуска приложения. Он не используется при развертывании.

{

"$schema": "http://json.schemastore.org/launchsettings.json",

"iisSettings": { *// Настройки для запуска под IIS Express*

"windowsAuthentication": false,

"anonymousAuthentication": true,

"iisExpress": {

"applicationUrl": "http://localhost:39542", *// URL для IIS Express*

"sslPort": 44376 *// SSL порт для IIS Express*

}

},

"profiles": { *// Профили запуска*

"http": { *// Профиль для запуска с HTTP*

"commandName": "Project", *// Запуск как проекта .NET*

"dotnetRunMessages": true,

"launchBrowser": true, *// Автоматически открывать браузер*

"launchUrl": "swagger", *// Стартовая страница в браузере*

"applicationUrl": "http://localhost:5290", *// URL, на котором будет слушать приложение*

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development" *// Установка среды выполнения*

}

},

"https": { *// Профиль для запуска с HTTPS*

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": true,

"launchUrl": "swagger",

"applicationUrl": "https://localhost:7143;http://localhost:5290", *// URL (HTTPS и HTTP)*

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"IIS Express": { *// Профиль для запуска через IIS Express*

"commandName": "IISExpress",

"launchBrowser": true,

"launchUrl": "swagger",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

}

}

* **iisSettings**: Конфигурация для запуска приложения под управлением IIS Express (используется в Visual Studio).
* **profiles**: Определяет различные именованные профили запуска.
  + **commandName**: Тип запуска ("Project" для Kestrel, "IISExpress").
  + **launchBrowser**: Открывать ли браузер при запуске.
  + **launchUrl**: Относительный URL, который будет открыт в браузере.
  + **applicationUrl**: URL(ы), на которых приложение будет принимать запросы.
  + **environmentVariables**: Переменные окружения, устанавливаемые для этого профиля (например, ASPNETCORE\_ENVIRONMENT для выбора файла appsettings.{Environment}.json).

# 12. Архитектура приложения

Приложение следует стандартной многоуровневой архитектуре для ASP.NET Core веб-сервисов:

**12.1. Контроллеры (Controllers)**

* Расположение: CheckCisService/Controllers/ (например, CheckCisController.cs).
* Отвечают за обработку входящих HTTP-запросов, валидацию данных (если необходимо), вызов соответствующих сервисов и формирование HTTP-ответов.
* Используют атрибуты ASP.NET Core для маршрутизации ([Route], [HttpGet], [HttpPost], и т.д.) и авторизации ([Authorize]).

**12.2. Сервисы (Services)**

* Расположение: CheckCisService/Services/.
* Содержат основную бизнес-логику приложения.
* Примеры: MarkingApiService, MarkingOnlineService, MarkingOfflineService, MdlpCheckCisLogService.
* Регистрируются в DI-контейнере (в Program.cs) и внедряются в контроллеры или другие сервисы.

**12.3. Репозитории (Repositories)**

* Расположение: CheckCisService/Repositories/.
* Отвечают за доступ к данным. В данном проекте MdlpCheckCisLogRepository инкапсулирует работу с LiteDB для хранения логов.

**12.4. Модели данных (Models/DTOs)**

* Расположение: CheckCisService/Models/.
* POCO-классы (Plain Old C# Object), представляющие сущности данных, DTO (Data Transfer Objects) для обмена данными с API, и перечисления (enums).

**12.5. Аутентификация (Basic Authentication)**

* Реализована с помощью BasicAuthenticationHandler.
* Приложение настроено на использование схемы "BasicAuthentication" в Program.cs.
* Контроллеры или эндпоинты, требующие аутентификации, должны быть защищены (например, с помощью app.MapControllers().RequireAuthorization(); или атрибута [Authorize]).
* Учетные данные (имя пользователя и пароль) для Basic Auth хранятся в appsettings.json.

**12.6. Вспомогательные классы (Helpers)**

* Расположение: CheckCisService/Helpers/.
* Содержат утилитарный код, такой как методы-расширения (CommonExtensions.cs) или специфичные для домена помощники (MdlpCashRegHelper.cs).

**12.7. Конфигурация (Config)**

* Расположение: CheckCisService/Config/.
* Классы, представляющие структуру конфигурационных файлов (appsettings.json) для удобного типизированного доступа к настройкам через DI (Options pattern).

**12.8. Исключения (Exceptions)**

* Расположение: CheckCisService/Exceptions/.
* Пользовательские классы исключений для обработки специфичных для приложения ошибок.

# 13. Примеры вызова методов API

Предполагается наличие контроллера CheckCisController с методом, обрабатывающим GET /CheckCis.

**13.1. Через .http файл**

Файл CheckCisService/CheckCisService.http содержит пример запроса, который можно выполнить в Visual Studio или VS Code (с расширением REST Client):

@CheckCisService\_HostAddress = http://localhost:5290

@ApiKey = Basic YWRtaW46MTIzNA== // admin:1234 в Base64

### Проверка КИЗ

# GET /CheckCis?code=КОД\_КИЗ\_ЗДЕСЬ

# Замените КОД\_КИЗ\_ЗДЕСЬ на реальный код.

# Параметр 'code' передается в query string.

GET {{CheckCisService\_HostAddress}}/CheckCis?code=0104604060006723215ABC123%1D93defg

Authorization: {{ApiKey}}

Accept: application/json

### Пример из файла CheckCisService.http

GET {{CheckCisService\_HostAddress}}/CheckCis HTTP/1.1

# Auth: Basic user:passwd # Этот формат неверен для заголовка Authorization

Authorization: {{ApiKey}}

Accept: application/json

# Примечание: GET-запрос без параметров обычно не используется для передачи данных КИЗ.

# Скорее всего, КИЗ должен передаваться как query parameter, например:

# GET {{CheckCisService\_HostAddress}}/CheckCis?code=YOUR\_CIS\_CODE\_HERE HTTP/1.1

**Важно:** В предоставленном CheckCisService.http GET-запрос к /CheckCis не содержит параметра с кодом КИЗ. Скорее всего, он должен передаваться как query-параметр, например, ?code=XXXXX. Пользователю нужно будет адаптировать эндпоинт контроллера для приема этого параметра. Заголовок Auth: Basic user:passwd также является нестандартным, правильный заголовок Authorization: Basic <base64\_credentials>.

**13.2. Через curl или Postman**

* **Предполагаемый эндпоинт:** GET http://localhost:5290/CheckCis?code={your\_cis\_code}
* **Аутентификация:** Basic Auth. Логин: admin, Пароль: 1234 (из appsettings.json). В Base64 это YWRtaW46MTIzNA==.

**Пример с curl (замените {your\_cis\_code}):**

curl -X GET "http://localhost:5290/CheckCis?code=0104604060006723215ABC123%1D93defg" \

-H "Accept: application/json" \

-H "Authorization: Basic YWRtaW46MTIzNA=="

**В Postman:**

1. Метод: GET
2. URL: http://localhost:5290/CheckCis
3. Вкладка Params:
   * KEY: code
   * VALUE: {your\_cis\_code}
4. Вкладка Authorization:
   * Type: Basic Auth
   * Username: admin
   * Password: 1234
5. Вкладка Headers:
   * Accept: application/json (обычно Postman добавляет это автоматически)
6. Нажмите Send.

# 14. Разработка и тестирование

**14.1. Соглашения по кодированию**

* Придерживайтесь официальных C# Coding Conventions от Microsoft.
* Используйте осмысленные имена для классов, методов и переменных.
* Форматируйте код единообразно (Visual Studio и VS Code имеют встроенные средства форматирования).
* Пишите XML-комментарии для публичных API.

**14.2. Ветвление (Git Flow - рекомендуется)**

Рекомендуется использовать модель ветвления Git Flow или подобную:

* main (или master): Стабильная версия, готовая к релизу.
* develop: Основная ветка разработки.
* feature/имя-фичи: Ветки для разработки новых функций. Создаются от develop, вливаются обратно в develop.
* release/версия: Ветки для подготовки релиза.
* hotfix/имя-исправления: Ветки для срочных исправлений в main.

**14.3. Юнит-тесты**

**14.3.1. Обзор тестовых проектов**

В решении присутствует проект CheckCisServiceTests, предназначенный для юнит-тестирования основного сервиса CheckCisService. Он использует:

* **xUnit:** Популярный фреймворк для тестирования в .NET.
* **Moq:** Библиотека для создания мок-объектов (заглушек) зависимостей.

**14.3.2. Перечень тестовых классов и их назначение**

* **CheckCisServiceTests/MarkingOfflineServiceUnitTests.cs**:
  + Содержит юнит-тесты для класса MarkingOfflineService.
  + CheckCisUriResolver\_GetStatusUri(): Проверяет корректность формирования URI для различных методов офлайн-сервиса в классе MarkingOfflineService.CheckCisUriResolver.
  + GetStatus\_SuccessfulResponse\_ReturnsDeserializedDto(): Тестирует успешное получение и десериализацию статуса офлайн-модуля.
  + GetStatus\_HttpError\_ThrowsIssException(): Проверяет, что при HTTP-ошибке от офлайн-сервиса выбрасывается ServiceException.
  + CheckCis\_Timeout\_ReturnsFailedDto(): Тестирует обработку таймаута при запросе к офлайн-сервису (ожидается TimeoutException).
  + CheckCis\_NullResponse\_ReturnsFailedDto(): Проверяет, что при получении null ответа от HttpClient выбрасывается HttpRequestException.
* **CheckCisServiceTests/MarkingOnlineServiceUnitTests.cs**:
  + Содержит юнит-тесты для класса MarkingOnlineService.
  + GetStatus\_SuccessfulResponse\_ReturnsDeserializedDto() (вероятно, имелось в виду GetCdnList\_SuccessfulResponse\_ReturnsDeserializedDto): Тестирует успешное получение списка CDN-хостов.
  + GetHostResponseTime\_Successful(): Проверяет успешное получение времени отклика от CDN-хоста.
  + CheckCode\_Successful(): Тестирует успешную проверку кода через онлайн-сервис.
* **CheckCisServiceTests/MdlpCheckCisLogRepositoryTests.cs**:
  + Содержит юнит-тесты для класса MdlpCheckCisLogRepository.
  + Add\_ValidItem\_ReturnsItem(): Проверяет успешное добавление валидной записи лога в LiteDB.
  + Add\_NullItem\_ThrowsServiceException(): Проверяет, что попытка добавить null в качестве лога приводит к ServiceException.

**14.3.3. Запуск тестов**

* **Через .NET CLI (в корневой директории решения CheckCisService-main или в директории CheckCisServiceTests):**
* dotnet test

Эта команда найдет все тестовые проекты в решении (или указанный проект) и выполнит тесты.

* **Через Visual Studio:**
  1. Откройте Test Explorer (меню Test -> Test Explorer).
  2. В Test Explorer вы увидите список всех найденных тестов.
  3. Можно запустить все тесты (Run All Tests In View) или выбрать и запустить конкретные тесты/классы.
* **Через Visual Studio Code:**
  1. Убедитесь, что установлено расширение C# Dev Kit, которое включает поддержку тестов.
  2. Откройте панель "Testing" (иконка с колбой на боковой панели).
  3. VS Code обнаружит тесты, и вы сможете их запустить.

# 15. Возможные проблемы и их решения

* **Порт занят (Address already in use):**
  + Убедитесь, что порт, указанный в launchSettings.json (например, 5290), не используется другим приложением.
  + Измените applicationUrl в launchSettings.json на другой порт.
* **Ошибка конфигурации (например, Mdlp секция не найдена):**
  + Проверьте правильность структуры и имен ключей в appsettings.json.
  + Убедитесь, что файл appsettings.json копируется в выходную директорию при сборке (свойство Copy to Output Directory должно быть Copy if newer или Copy always).
* **Проблемы с LiteDB (например, файл заблокирован):**
  + Убедитесь, что у приложения есть права на запись в директорию, где должен создаваться файл check\_cis\_log.db.
  + Если файл используется другим процессом, закройте его.
* **Ошибки аутентификации (401 Unauthorized):**
  + Проверьте, что вы правильно передаете заголовок Authorization: Basic <credentials\_in\_base64>.
  + Убедитесь, что имя пользователя и пароль в BasicAuth секции appsettings.json совпадают с теми, что вы используете.
* **Не удается подключиться к внешним сервисам (офлайн/онлайн):**
  + Проверьте сетевое соединение.
  + Убедитесь, что адреса хостов в appsettings.json (в OfflineService:Host и OnlineService:Host) корректны и доступны.
  + Проверьте ApiKey и другие учетные данные для внешних сервисов.
  + Проверьте настройки файрвола/прокси, если они есть.

# 16. Контакты и поддержка

* **Владелец репозитория:** DmitriyShch (<https://github.com/DmitriyShch>)
* **Issues:** Для сообщения об ошибках или предложениях используйте раздел "Issues" в GitHub репозитории: <https://github.com/DmitriyShch/CheckCisService/issues>