МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра Вычислительные системы и технологии

Ремонт картеджей G-Принтеров

(наименование темы проекта или работы)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

(ОТЧЁТ по курсовой работе)

по дисциплине

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Сервис-ориентированные системы\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование дисциплины)

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Скорынин С.С.

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Марчус К.Р.

(подпись) (фамилия, и.,о.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2022

# 2. СОДЕРЖАНИЕ

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# 3. СПИСОК СОКРАЩЕНИЙ

SOA - сервис-ориентированная система

БД – база данных

# 4. ВВЕДЕНИЕ

## 4.1) Актуальность сервис-ориентированной архитектуры

SOА – это метод разработки программного обеспечения, который использует программные компоненты, называемые сервисами, для создания распределённых программных систем. SOА придумана в конце 1980-х. Она берёт своё начало в идеях, изложенных в CORBA, DCOM, DCE и других документах. О ней написано много, и даже есть несколько методов реализации SOA. Однако она сводится к нескольким идеям:

* Сочетаемость приложений, ориентированных на пользователей
* Многократное использование бизнес-сервисов
* Независимость от набора технологий
* Автономность

SOA очень актуальна на данный момент, так как обладает теми преимуществами, которые необходимы для создания приложений:

* Сокращение времени выхода на рынок
* Эффективное обслуживание
* Улучшенная адаптивность

## 4.2) Характеристика задачи

В ходе курсовой работы стоит задача по созданию приложения по починке картриджей G-принтеров. Данная задача может быть реализована как распределённая программная система, так как предусматривает взаимодействие клиентов, сотрудников и механиков между собой и использование учётных записей соответственно клиента, сотрудника и механика.

## 4.3) Цель курсовой работы

Реализовать как распределённую программную систему приложение по починке картриджей G-принтеров.

## 4.4) Задачи, решаемые в ходе курсовой работы

* Определение модулей проекта
* Определение структуры БД каждого из модулей
* Выбор платформы
* Реализация REST-API
* Выбор микросервисов

# 5. ПОСТАНОВКА ЗАДАЧИ

Система учёта починки картриджей G-принтеров.

Используется стек технологий: C#, ASP.NET MVC Framework, Entity Framework, Blazor, Microsoft Visual Studio, RabbitMQ

Microsoft Visual Studio - включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода, добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения.

Язык C# - язык с C-подобным синтаксисом, его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, переменные, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Blazor — бесплатная веб-платформа с открытым исходным кодом, позволяющая создавать веб-приложения с использованием C# и HTML. В рамках курсовой работы использовалась редакция Blazor WebAssembly.

Entity Framework (EF) — объектно-ориентированная технология доступа к данным, является object-relational mapping (ORM) решением для .NET Framework от Microsoft. Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL. Для облегчения построения web-решений используется как ADO.NET Data Services (Astoria), так и связка из Windows Communication Foundation и Windows Presentation Foundation, позволяющая строить многоуровневые приложения, реализуя один из шаблонов проектирования MVC, MVP или MVVM.

ASP.NET MVC Framework – базируется на взаимодействии трех компонентов: контроллера, модели и представления. Контроллер принимает запросы, обрабатывает пользовательский ввод, взаимодействует с моделью и представлением и возвращает пользователю результат обработки запроса. Модель представляет слой, описывающий логику организации данных в приложении. Представление получает данные из контроллера и генерирует элементы пользовательского интерфейса для отображения информации.

RabbitMQ — программный брокер сообщений на основе стандарта AMQP — тиражируемое связующее программное обеспечение, ориентированное на обработку сообщений. Создан на основе системы Open Telecom Platform, написан на языке Erlang, в качестве движка базы данных для хранения сообщений использует Mnesia

## 5.1) Описание проекта

Приложение позволит регистрироваться в качестве:

* Администратора
* Клиента
* Сотрудника
* Механика

Администратор может изменять состав сотрудников, механиков, подразделений, типов картриджей принтеров и т.д.

Клиент и сотрудник может ввести баркод неисправного картриджа и указать его тип. Отличие клиента от сотрудника в том, что сотрудник закреплён за конкретным подразделением.

Механик выполняет задание починки картриджа и отмечает выполнение задания.

# 6. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПРОЕКТИРОВАНИЕ API

## 6.1) Объектно-ориентированная модель ресурсов

Было создано 2 модуля с классами:

* Основной модуль
  + Cartridge – перечень картриджей с баркодами
  + CartridgeType – перечень моделей картриджей
  + Department – перечень подразделений
  + Employee – перечень сотрудников
  + Event – сюда записываются любые события, связанные с изменением состояния картриджа
  + Mechanic – перечень механиков
  + Repair – перечень заданий починки
  + Status – состояние картриджа, может быть «в ремонте», «на складе», «работает или «списан»
* Модуль пользователей
  + User – перечень логинов, паролей и почт пользователей
  + Reg – дата регистрации и последнего входа в аккаунт
  + Rank – ранг пользователя: администратор, сотрудник, механик, клиент

Диаграмма классов основного модуля:



Диаграмма классов модуля пользователей:



Диаграмма взаимодействия:



## 6.2) Проектирование REST-API

Пользователь может взаимодействовать с системой при помощи HTTP-запросов. Есть точки доступа API, которые позволяют взаимодействовать с платформой извне. В данном проекте используются методы:

* GET – получение данных, аргумент Id возвращает данные одного объекта
* POST – добавление данных, используется, например, при регистрации нового пользователя
* DELETE – удаление данных
* PUT – обновление данных
* OPTIONS - используется для описания параметров соединения с целевым ресурсом.

6.2а) Описание ресурсов

Ресурсы подразделяются по классам в программе модуля. Описание ресурсов приведено в п.6.1.

6.2б) Описание групп точек доступа

Точки доступа группировались по классам в программе модуля.

6.2в) Описание отдельных точек доступа

Некоторые точки доступа используются для других целей. Например, точка доступа [GET] /api/Cartridges/statusbybarcode/{barcode} проверяет состояние картриджа по баркоду. Точка доступа [GET] /api/Cartridges/checkUniquebarcode/{barcode} возвращает true, если баркод картриджа не совпадает с баркодом другого картриджа.

6.2г) Документирование REST-API в SwaggerAPI

Для каждого модуля разработана документация в SwaggerAPI. Swagger позволяет проверять работоспособность HTTP-запросов и просматривать перечень возможных запросов. Скриншоты работы Swagger указаны в приложении.

# 7. РЕАЛИЗАЦИЯ СЕРВИС-ОРИЕНТИРОВАННОЙ СИСТЕМЫ

## 7.1) Аналитический обзор платформ и средств построения сервис-ориентированных систем

Были выбраны платформы, перечисленные и описанные в п.5.

Рассмотрим преимущества и недостатки некоторых платформ:

* Microsoft Visual Studio
  + Преимущества
    - Встроенный Web-сервер. Для обслуживания Web-приложения ASP.NET необходим Web-сервер, который будет ожидать Web-запросы и обрабатывать соответствующие страницы. Наличие в Visual Studio интегрированного Web-сервера позволяет запускать Web-сайт прямо из среды проектирования, а также повышает безопасность, исключая вероятность получения доступа к тестовому Web-сайту с какого-нибудь внешнего компьютера, поскольку тестовый сервер может принимать соединения только с локального компьютера.
    - Интуитивный стиль кодирования.
    - Более высокая скорость разработки.
  + Недостатки
    - Невозможность отладчика (Microsoft Visual Studio Debugger) отслеживать код в режиме ядра.
* Entity Framework
  + Преимущества
    - Возможность удобно устанавливать различные типы связей и тип удаления для каждого узла БД
    - Упрощение громоздкого кода взаимодействия с БД, следовательно, и ускорение процесса написания кода
  + Недостатки
    - Может генерировать плохие SQL-запросы
    - Медленный ORM и иногда генерация SQL-запросов
* Blazor
  + Преимущества
    - Позволяет в одном файле создавать секции HTTP и C#, что довольно удобно
    - Можно создать одноимённый файл формата .razor.css и задавать параметры оформления и вёрстки
  + Недостатки
    - Теряет актуальность на данный момент

## 7.2) Компонентная модель сервис-ориентированной системы

Для безопасности системы приложения компоненты подразделяются на внутренние и внешние. К внутренним относится БД пользователей и его методы, этот компонент соединён с остальной системой брокером RabbitMQ. Однако администратор может иметь право удалять аккаунты и изменять привилегии.

К внешним: БД и методы основного модуля, методы и страницы фронтенда.

## 7.3) Настройка программного проекта

Настройки определяются в файлах .json

Адреса доступа к основному модулю: <https://localhost:44371>, к фронтенду: <https://localhost:44312>.

Режим работы с БД определён в appsettings.json (пример для основного модуля):

{

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=cartridgedb;Trusted\_Connection=True;MultipleActiveResultSets=true"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

Параметры серверов определены в launchSettings.json:

{

"iisSettings": {

"windowsAuthentication": false,

"anonymousAuthentication": true,

"iisExpress": {

"applicationUrl": "http://localhost:58836",

"sslPort": 44371

}

},

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

"profiles": {

"IIS Express": {

"commandName": "IISExpress",

"launchBrowser": true,

"launchUrl": "http://localhost:44371/",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"BackendCartridges": {

"commandName": "Project",

"launchBrowser": true,

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

},

"dotnetRunMessages": "true",

"applicationUrl": "https://localhost:5003;http://localhost:5002"

}

}

}

## 7.4) Описание реализации сервисов

В первую очередь создаём модули: основной и модуль пользователей при помощи IDE Microsoft Visual Studio, ASP .NET MVC и Entity Framework.

В каждом модуле создаём класс в директории Models, где будут описаны поля. Например, для класса CartridgeTypes будут поля:

* Id – первичный ключ
* Vender – наименование фирмы
* Model – модель картриджа
* PartNumber – серийный номер
* Cartridges – связь с классом Cartridges, реализация связи «один ко многим»

[Table("CartridgeTypes")]

public class CartridgeType

{

public int Id { get; set; }

public string Vender { get; set; }

public string Model { get; set; }

public string PartNumber { get; set; }

[JsonIgnore]

public virtual IEnumerable<Cartridge> Cartridges { get; set; } = new List<Cartridge>();

}

Далее создаём контроллер к каждому классу. В нём с помощью средств ASP .NET реализуем HTTP-запросы. Например, как это выглядит для класса Cartridges.

[ApiController]

[Route("api/[controller]")]

public class CartridgesController

{

CartridgeService \_cartridgeService;

public CartridgesController(CartridgeService cartridgeService)

{

\_cartridgeService = cartridgeService;

}

[HttpGet]

public IEnumerable<Cartridge> Get()

{

return \_cartridgeService.GetCartridges();

}

[HttpGet("{id}")]

public Cartridge Get(int id)

{

return \_cartridgeService.GetCartridge(id);

}

[HttpPost]

public int Add(Cartridge cartridge)

{

\_cartridgeService.AddCartridge(cartridge);

return cartridge.Id;

}

[HttpDelete("{id}")]

public void Delete(int id)

{

\_cartridgeService.DeleteCartridge(id);

}

[HttpPut]

public void Update(Cartridge cartridge)

{

\_cartridgeService.UpdateCartridge(cartridge);

}

[HttpGet("checkUniquebarcode/{barcode}")]

public bool CheckUniqueBarcode(int barcode)

{

return \_cartridgeService.CheckUniqueBarcode(barcode);

}

[HttpGet("statusbybarcode/{barcode}")]

public int StatusByBarcode(int barcode)

{

return \_cartridgeService.StatusByBarcode(barcode);

}

[HttpGet("bybarcode/{barcode}")]

public Cartridge GetByBarcode(int barcode)

{

return \_cartridgeService.GetByBarcode(barcode);

}

Также необходимо создать сервис для каждого класса. В нём реализуются методы, вызываемые в контроллерах, необходимые для фронтенда и создания SQL-запросов.

public class CartridgeService

{

private readonly ApplicationContext \_context;

public CartridgeService(ApplicationContext context)

{

\_context = context;

}

public IEnumerable<Cartridge> GetCartridges()

{

\_context.CartridgeTypes.Load();

\_context.Events.Load();

\_context.Departments.Load();

\_context.Statuses.Load();

return \_context.Cartridges.ToList();

}

public Cartridge GetCartridge(int id)

{

return \_context.Cartridges

.Include(x => x.Type)

.FirstOrDefault(x => x.Id == id);

}

public void AddCartridge(Cartridge cartridge)

{

\_context.Cartridges.Add(cartridge);

\_context.SaveChanges();

Event evt = new Event()

{

CartridgeId = cartridge.Id,

EventDate = DateTime.Now,

StatusId = 1,

EmployeeId = 1,

FromDepartmentId = 1,

ToDepartmentId = 1

};

\_context.Events.Add(evt);

\_context.SaveChanges();

}

public void DeleteCartridge(int id)

{

Cartridge cartridge = GetCartridge(id);

\_context.Cartridges.Remove(cartridge);

\_context.SaveChanges();

}

public void UpdateCartridge(Cartridge cartridge)

{

\_context.Cartridges.Update(cartridge);

\_context.SaveChanges();

}

public bool CheckUniqueBarcode(int barcode)

{

return !\_context.Cartridges.Any(x => x.Barcode == barcode);

}

public int StatusByBarcode(int barcode)

{

return \_context.Events.Where(x => x.Cartridge.Barcode == barcode)

.OrderByDescending(x => x.EventDate)

.Select(x => x.StatusId)

.FirstOrDefault();

}

public Cartridge GetByBarcode(int barcode)

{

return \_context.Cartridges.Include(x => x.Type)

.FirstOrDefault(x => x.Barcode == barcode);

}

}

Во фронтенд-части приложения создаём страницу Razor для каждого из реализованных методов. Так выглядит страница создания картриджа:

@page "/cartridges/add"

@using System.Net.Http

@using System.Net.Http.Json

@using System.Threading.Tasks

@using FrontendCartridges.Models

@using Microsoft.Extensions.Logging;

@inject HttpClient Http

@inject NavigationManager navigationManager

@inject ILogger<NewCartridge> Logger

<h3>New Cartridge</h3>

<**EditForm** **EditContext**="@editContext" **OnValidSubmit**="@AddAction">

<**DataAnnotationsValidator** />

<**ValidationSummary** />

<input id="Barcode" placeholder="Barcode" value="@addCartridge.Barcode" **@onchange**="@((ChangeEventArgs e) => ChangeBarcode(e?.Value?.ToString()))" />

<br />

<h6>Select Type</h6>

@if (cartridgeTypes != null)

{

<**InputSelect** id="TypeId" **@bind-Value**="addCartridge.TypeId">

@foreach (var cartridgeType in cartridgeTypes)

{

<option value="@cartridgeType.Id">@cartridgeType.FullName</option>

}

</**InputSelect**>

}

<br />

<**Button** **Type**="submit" **Name**="Add" **Width**="100" />

<**Button** **Type**="button" **Name**="Cancel" **Width**="100" **OnClick**="Back" **HasActive**="true"></**Button**>

</**EditForm**>

@code {

private EditContext editContext;

private IEnumerable<CartridgeType> cartridgeTypes;

private Cartridge addCartridge = new Cartridge();

protected override async Task OnInitializedAsync()

{

editContext = new(addCartridge);

cartridgeTypes = await Http.GetFromJsonAsync<IEnumerable<CartridgeType>>("https://localhost:44371/api/cartridgetypes");

}

private async Task AddCartridge()

{

await Http.PostAsJsonAsync("https://localhost:44371/api/cartridges", addCartridge);

}

private void Back()

{

navigationManager.NavigateTo("https://localhost:44312/cartridges");

}

private async Task AddAction()

{

if (editContext != null && editContext.Validate())

{

await AddCartridge();

Back();

}

}

private async Task ChangeBarcode(string value)

{

addCartridge.Barcode = int.Parse(value);

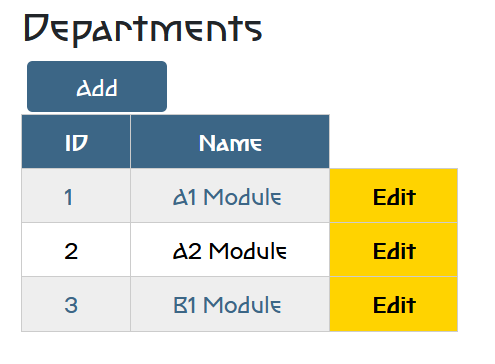
addCartridge.IsUniqueBarcode = await Http.GetFromJsonAsync<bool>($"https://localhost:44371/api/cartridges/checkUniquebarcode/{addCartridge.Barcode}");

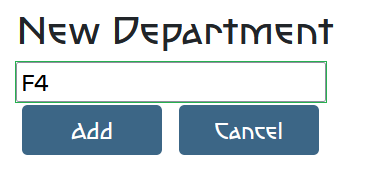
}

}

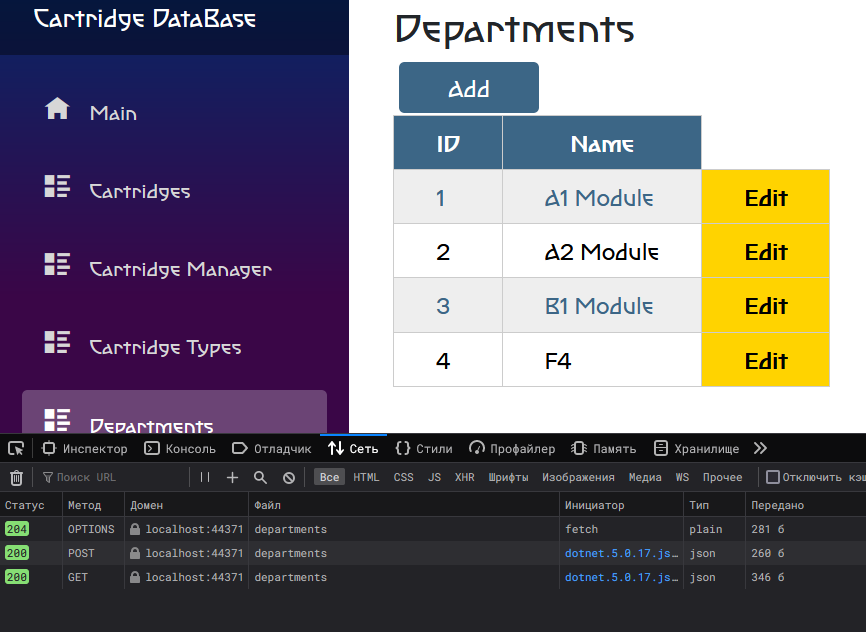
## 7.5) Описание тестового примера

Запустим приложение и включим отладчик для отслеживания выполнения HTTP-запросов. Попытаемся создать четвёртое подразделение.



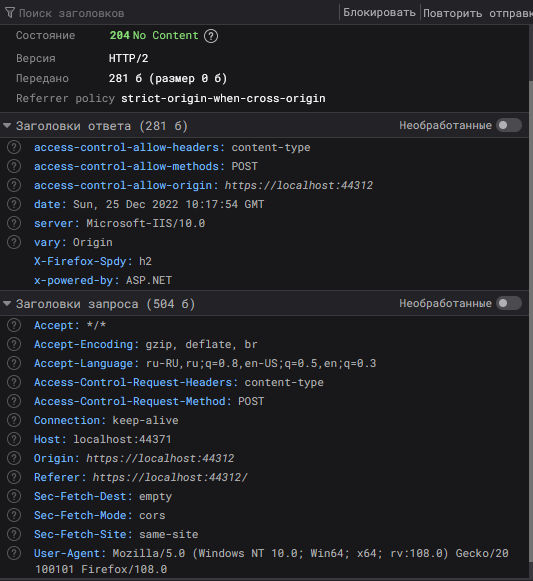


Подразаделение добавлено успешно



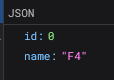
Рассмотрим подробнее запросы, появившиеся в отладчике.

Первый запрос имеет метод OPTIONS. Он описывает параметры соединения с целевым ресурсом. Его структура:



В поле access-control-allow-methods указан метод, который должен быть применён к ресурсу.

Далее следует запрос с методом POST, добавляющий новую запись. На вход идёт имя подразделения.



Метод возвращает первый свободный идентификатор (4) 

При добавлении записи во фронтенде реализуются эти программные методы

private async Task AddDepartment()

{

await Http.PostAsJsonAsync("https://localhost:44371/api/departments", addDepartment);

}

private void Back()

{

navigationManager.NavigateTo("https://localhost:44312/departments");

}

private async Task AddAction()

{

if (editContext != null && editContext.Validate())

{

await AddDepartment();

Back();

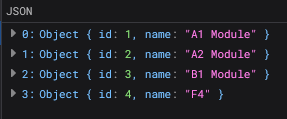
}

}

Вызов функции Back() отправляет ещё один запрос уже метода GET, так как страница https://localhost:44312/departments имеет этот программный метод:

protected override async Task OnInitializedAsync() =>

departments = await Http.GetFromJsonAsync<Department[]>("https://localhost:44371/api/departments");



# 8. ЗАКЛЮЧЕНИЕ

В результате выполнения этой курсовой работы было создано приложение по починке картриджей G-принтеров. Была продумана структура проекта. Во время выполнения работы были сложности с брокером, а, следовательно, и с регистрацией пользователей, поскольку брокер RabbitMQ не может полноценно работать без Docker.

Проект имеет дальнейшие перспективы развития. Например, используя уже имеющееся разграничение пользователей на просто клиентов и сотрудников, которые на данный момент различаются только принадлежностью к подразделениям, можно в дальнейшем развивать их роли. Например, сотрудники могут быть заместителями администратора или модераторами приложения.

Также возможно расширить сферу проекта, к примеру, починка не только картриджей принтеров, но и всей IT-техники в целом.

# 9. СПИСОК ЛИТЕРАТУРЫ

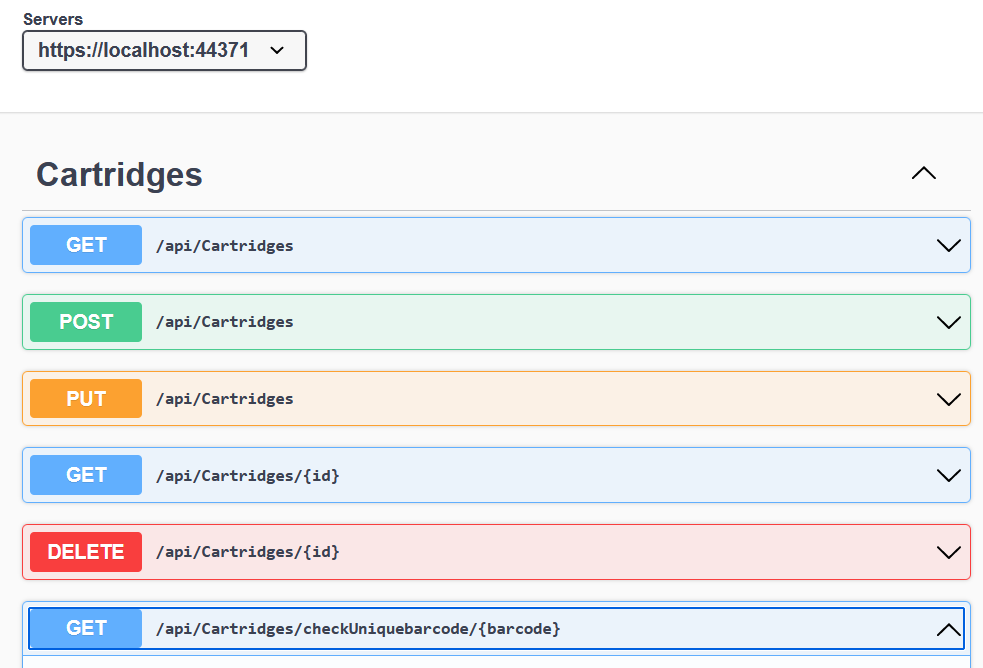
1 <https://habr.com/ru/company/vk/blog/342526/>

2 <https://aws.amazon.com/ru/what-is/service-oriented-architecture/>

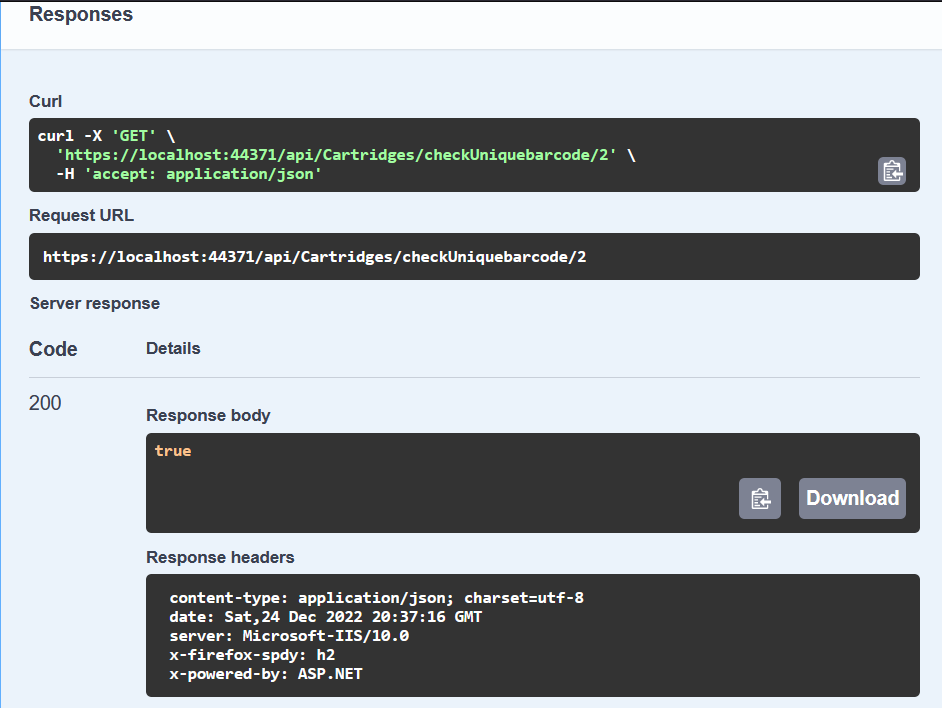
3 https://studbooks.net/2258619/informatika/opisanie\_sredy\_razrabotki\_microsoft\_visual\_studio

# 10. ПРИЛОЖЕНИЯ

Перечень HTTP-запросов (на скриншоте – запросы группы Catridges) и доступных хостов:



Ответ на запрос в Swagger:



UML-диаграмма компонентов (ввиду большого кол-во методов отображены только наименования классов):



UML-диаграмма размещения:

