**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные системы и технологии

(в проектировании и производстве)»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

 «Разработка моделей и контроллеров ASP.NET MVC приложения баз

данных»

Выполнил: студент гр. ИТП-31

Коробкин В. В.

         Принял: преподаватель

Асенчик О. Д.

Гомель 2020

**Цель:** ознакомиться с возможностями ASP.NET Core MVC и Entity Framework Core для разработки слоя доступа к данным, хранящимся в базе данных, и обработки запросов пользователя посредством контроллеров.

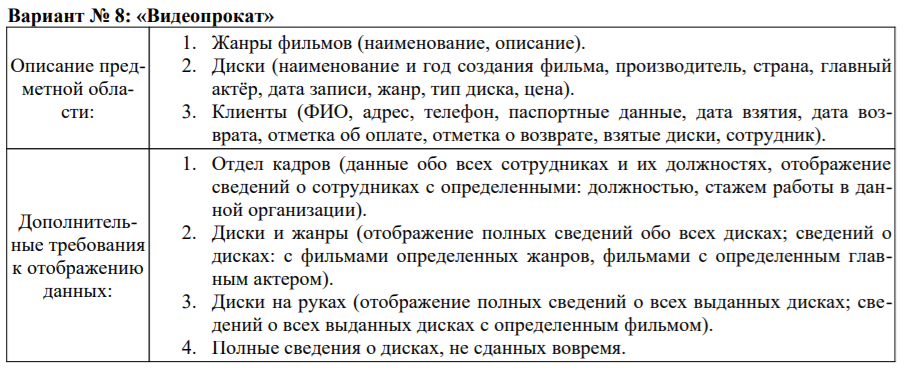
****

Рисунок 1 – вариант задания

**Задание:**

Создать с использованием *ASP.NET Core MVC Web*-приложение, содержащее набор классов, моделирующих предметную область, и осуществляющих генерацию и заполнение тестовыми наборами записей базу данных. Разработать один компонент *middleware*, контроллеры и представления для выборки и отображения информации из не менее чем 3-таблиц базы данных с использованием механизма внедрение зависимостей.

Для выполнения задания необходимо создать:

* Классы, моделирующие не менее чем три таблицы базы данных согласно вашему варианту. Перечень таблиц предварительно согласовывается с преподавателем. Одна из таблиц обязательно должна находиться на стороне отношения «многие» связи с другой таблицей в схеме базы данных.
* Класс контекста данных.
* Другие классы, например, классы *View Model* и т.п. (при необходимости).
* Компонент *middleware*, вызываемый в классе *Startup*, для инициализации базы данных путем заполнения ее таблиц тестовым набором записей.
* Классы контроллеров (по одному на каждую таблицу базы данных) для обработки обращений пользователя, выборки данных из таблиц и вызова соответствующих представлений для отображения выбранных данных.
* Разработать представления для отображения данных из таблиц, выбранных контроллерами. Представления, работающими с таблицами, стоящими на стороне отношения «многие» в схеме базы данных, должны выводить вместо кодов внешних ключей смысловые значения из связанных таблиц, стоящих на стороне отношения «один».
* Используя предварительно созданный и сконфигурированный в классе *Startup* профиль кэширования, подключить кэширование вывода для страниц с использованием атрибута *ResponseCache* для соответствующих методов контроллера. Данные в кэше хранить неизменными в течение 2\*N+240 секунд, где N- номер вашего варианта.
* С использованием средств разработчика браузера (*Chrome*, *Firefox*) продемонстрировать ускорение обработки запроса при наличии кэширования с использованием атрибута *ResponseCache*.

Листинг кодов классов *startup*, классов моделей, классов *View Model* и контекста данных:

Ссылка на GitHub: https://github.com/viktor3110/BD\_lab4

*Startup.cs*

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using WebApp.Models;

using WebApp.MiddleWares;

namespace WebApp

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)

{

// Добавление профилей кэширования для таблиц и остальных страниц

services.AddControllersWithViews(options =>

{

options.CacheProfiles.Add("TablesCaching",

new CacheProfile()

{

Duration = 256

});

options.CacheProfiles.Add("NoCaching",

new CacheProfile()

{

Location = ResponseCacheLocation.None,

NoStore = true

});

});

// Добавление контекста данных со строкой подключения, хранящейся в файле appsettings.json

services.AddDbContext<VideoRentalContext>(options => options.UseSqlServer(Configuration.GetConnectionString("SqlServerConnection")));

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.UseInitializeMiddleware();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

}

}

}

*Client.cs*

using System;

using System.Collections.Generic;

namespace WebApp.Models

{

public partial class Client

{

public Client() {}

public int Id { get; set; }

public string Fio { get; set; }

public int Number { get; set; }

public string Pasport { get; set; }

}

}

*Employee.cs*

using System;

using System.Collections.Generic;

namespace WebApp.Models

{

public partial class Employee

{

public Employee(){}

public int Id { get; set; }

public string Fio { get; set; }

public string Position { get; set; }

public DateTime DateOfWorkStart { get; set; }

}

}

*Disc.cs*

using System;

using System.Collections.Generic;

namespace WebApp.Models

{

public partial class Disc

{

public Disc() {}

public int Id { get; set; }

public string Name { get; set; }

public DateTime DateOfCreation { get; set; }

public string Creater { get; set; }

public string Country { get; set; }

public string MainActor { get; set; }

public DateTime DateOfRecord { get; set; }

public int Genre { get; set; }

public string TypeOfDisc { get; set; }

public decimal Price { get; set; }

}

}

*Genre.cs*

using System;

using System.Collections.Generic;

namespace WebApp.Models

{

public partial class Genre

{

public Genre() {}

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

}

}

*RentalRecord.cs*

using System;

using System.Collections.Generic;

namespace WebApp.Models

{

public partial class RentalRecord

{

public int Id { get; set; }

public int ClientId { get; set; }

public DateTime DateOfRent { get; set; }

public DateTime DateOfReturn { get; set; }

public int PaymentCheck { get; set; }

public int ReturnCheck { get; set; }

public int DiscId { get; set; }

public int EmployeeId { get; set; }

}

}

*DiscViewModel.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace WebApp.Models

{

public class DiscViewModel

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime DateOfCreation { get; set; }

public string Creater { get; set; }

public string Country { get; set; }

public string MainActor { get; set; }

public DateTime DateOfRecord { get; set; }

public string GenreName { get; set; }

public string TypeOfDisc { get; set; }

public decimal Price { get; set; }

}

}

*RentalRecordViewModel.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace WebApp.Models

{

public class RentalRecordViewModel

{

public int Id { get; set; }

public string ClientFIO { get; set; }

public DateTime DateOfRent { get; set; }

public DateTime DateOfReturn { get; set; }

public int PaymentCheck { get; set; }

public int ReturnCheck { get; set; }

public string DiscName { get; set; }

public string EmployeeFIO { get; set; }

}

}

*VideoRentalContext.cs*

using System;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata;

namespace WebApp.Models

{

public partial class VideoRentalContext : DbContext

{

public VideoRentalContext(DbContextOptions<VideoRentalContext> options)

: base(options)

{

}

public virtual DbSet<Client> Clients { get; set; }

public virtual DbSet<Disc> Discs { get; set; }

public virtual DbSet<Employee> Employees { get; set; }

public virtual DbSet<Genre> Genres { get; set; }

public virtual DbSet<RentalRecord> RentalRecords { get; set; }

}

}

Листинг строки подключения к базе данных из конфигурационного файла:

*appsettings.json*

"ConnectionStrings": {

"SqlServerConnection": "Data Source = DESKTOP-NO2QHQ2; Database=VideoRental; Integrated Security = True; Connect Timeout = 60; Encrypt = False; TrustServerCertificate = False; ApplicationIntent = ReadWrite; MultiSubnetFailover = False"

},

Листинг исходного кода компонента *middleware* для инициализации базы данных путем заполнения ее таблиц тестовым набором записей и метода заполнения:

*DbInitializer.cs*

using System;

using System.Linq;

namespace WebApp.Models

{

// Класс, содержащий метод инициализации базы данных

public static class DbInitializer

{

// Метод инициализации базы данных путем заполнения таблиц тестовыми наборами данных

public static void Initialize(VideoRentalContext db)

{

// Метод, который проверяет существование базы данных

db.Database.EnsureCreated();

// Объекты для генерации случайных чисел и записей

Random randObj = new Random();

char[] letters = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ".ToCharArray();

// Проверка на наличие записей в таблице Клиенты

if (!db.Clients.Any())

{

int clientId;

string fio = "";

int numb;

string passport = "";

// Создание 40 записей в таблице

for (int id = 1; id <= 40; id++)

{

// Получение Id

clientId = db.Clients.Count() + 1;

// Создание ФИО

int rand = randObj.Next(17, 100);

for (int i = 1; i <= rand; i++)

{

fio += letters[randObj.Next(33)];

}

// Создание номера клиента

numb = randObj.Next(1000000, 9999999);

// Создание паспорта

for (int i = 1; i <= 9; i++)

{

passport += letters[randObj.Next(33)];

}

// Добавление записи в таблицу

db.Clients.Add(new Client { Id = clientId, Fio = fio, Pasport = passport, Number = numb });

}

db.SaveChanges();

}

// Проверка на наличие записей в таблице Работники

if (!db.Employees.Any())

{

int employeeId;

string fio = "";

string position = "";

DateTime date;

// Создание 40 записей

for (int id = 1; id <= 40; id++)

{

// Получение Id

employeeId = db.Employees.Count() + 1;

// Создание ФИО работника

int rand = randObj.Next(15, 100);

for (int i = 1; i <= rand; i++)

{

fio += letters[randObj.Next(33)];

}

// Создание должности работника

rand = randObj.Next(5, 50);

for (int i = 1; i <= rand; i++)

{

position += letters[randObj.Next(33)];

}

// Создание даты начала работы

date = DateTime.Now.AddYears(-2);

// Добавление записи в таблицу

db.Employees.Add(new Employee { Id = employeeId, Fio = fio, Position = position, DateOfWorkStart = date });

}

db.SaveChanges();

}

// Проверка на наличие записей в таблице Жанры

if (!db.Genres.Any())

{

int genreId;

string name = "";

string descr = "";

// Создание 40 записей

for (int id = 1; id <= 40; id++)

{

// Получение Id

genreId = db.Genres.Count() + 1;

// Создание названия жанра

int rand = randObj.Next(3, 21);

for (int i = 1; i <= rand; i++)

{

name += letters[randObj.Next(33)];

}

// Создание описания жанра

rand = randObj.Next(17, 200);

for (int i = 1; i <= rand; i++)

{

descr += letters[randObj.Next(33)];

}

// Добавление записи в таблицу

db.Genres.Add(new Genre { Id = genreId, Name = name, Description = descr });

}

db.SaveChanges();

}

// Проверка на наличие записей в таблице Диски

if (!db.Discs.Any())

{

int discId;

string name = "";

DateTime date;

string creat = "";

string country = "";

string mainAct = "";

DateTime recordDate;

int genreId;

string type = "";

decimal price;

// Создание 100 записей

for (int id = 1; id <= 100; id++)

{

// Получение Id

discId = db.Discs.Count() + 1;

// Создание названия диска

int rand = randObj.Next(3, 21);

for (int i = 1; i <= rand; i++)

{

name += letters[randObj.Next(33)];

}

// Получение даты создания фильма

date = DateTime.Now.AddYears(-1);

// Создание даты записи

recordDate = DateTime.Now.AddDays(-3);

// Создание создателя фильма

rand = randObj.Next(3, 21);

for (int i = 1; i <= rand; i++)

{

creat += letters[randObj.Next(33)];

}

// Создание страны

rand = randObj.Next(3, 51);

for (int i = 1; i <= rand; i++)

{

country += letters[randObj.Next(33)];

}

// Создание главного актера

rand = randObj.Next(17, 101);

for (int i = 1; i <= rand; i++)

{

mainAct += letters[randObj.Next(33)];

}

// Создание стоимости фильма

price = (decimal)randObj.NextDouble() \* 100;

// Получение Id жанра

genreId = randObj.Next(1, db.Genres.Select(elem => elem.Id).Max());

// Создание типа диска

rand = randObj.Next(3, 8);

for (int i = 1; i <= rand; i++)

{

type += letters[randObj.Next(33)];

}

// Добавление записи в таблицу

db.Discs.Add(new Disc { Id = discId, Name = name, Country = country, Creater = creat, DateOfCreation = date, DateOfRecord = recordDate, Genre = genreId, MainActor = mainAct, TypeOfDisc = type, Price = price });

}

db.SaveChanges();

}

// Проверка на наличие записей в таблице Записи проката

if (!db.RentalRecords.Any())

{

int rentId;

int clientId;

DateTime dateRent;

DateTime dateReturn;

int payCheck;

int discId;

int emplId;

// Создание 100 записей

for (int id = 1; id <= 100; id++)

{

// Получение Id

rentId = db.RentalRecords.Count() + 1;

// Создание Id клиента

clientId = randObj.Next(1, db.Clients.Select(elem => elem.Id).Max());

// Создание даты проката

dateRent = DateTime.Now.AddDays(-10);

// Создание даты возврата

dateReturn = DateTime.Now.AddDays(-3);

// Создание проверки оплаты

payCheck = randObj.Next(0,2);

// Создание Id диска

discId = randObj.Next(1, db.Discs.Select(elem => elem.Id).Max());

// Получение Id работника

emplId = randObj.Next(1, db.Employees.Select(elem => elem.Id).Max());

// Добавление записи в таблицу

db.RentalRecords.Add(new RentalRecord { Id = rentId, ClientId = clientId, DateOfRent = dateRent, DateOfReturn = dateReturn, DiscId = discId, PaymentCheck = payCheck, ReturnCheck = 1, EmployeeId = emplId});

}

db.SaveChanges();

}

}

}

}

*DbInitializerMiddleWare.cs и DbInitializerExtentions.cs*

using System.Threading.Tasks;

using WebApp.Models;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

namespace WebApp.MiddleWares

{

// Компонент middleware для инициализации базы данных

public class DbInitializMiddleWare

{

// Ссылка на следующий компонент

private readonly RequestDelegate \_next;

public DbInitializMiddleWare(RequestDelegate next)

{

\_next = next;

}

public Task Invoke(HttpContext httpContext, VideoRentalContext context)

{

DbInitializer.Initialize(context);

return \_next(httpContext);

}

}

// Метод расширения для добавления компонента middleware

public static class DbInitializeExtensions

{

public static IApplicationBuilder UseInitializeMiddleware(this IApplicationBuilder builder)

{

return builder.UseMiddleware<DbInitializMiddleWare>();

}

}

}

Листинг исходного кода контроллеров и представлений:

*HomeController.cs*

using System.Diagnostics;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using WebApp.Models;

namespace WebApp.Controllers

{

// Контроллер начальной страницы

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

// Метод получения начальной страницы.

// Начальная страница не кэшируется.

[ResponseCache(CacheProfileName = "NoCaching")]

public IActionResult Index()

{

return View();

}

// Метод получения страницы ошибки.

// Страница ошибки не кэшируется.

[ResponseCache(CacheProfileName = "NoCaching")]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

*TablesController.cs*

using System.Collections.Generic;

using System.Linq;

using WebApp.Models;

using Microsoft.AspNetCore.Mvc;

namespace WebApp.Controllers

{

// Контроллер представления страниц записей из таблиц

public class TablesController : Controller

{

// Объект контекста данных

private readonly VideoRentalContext db;

public TablesController(VideoRentalContext applicationContext)

{

db = applicationContext;

}

// Метод получения страницы клиентов.

// Данная страница кэшируется на 256 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult GetClients()

{

List<Client> clients = db.Clients.ToList();

return View(clients);

}

// Метод получения страницы работников.

// Данная страница кэшируется на 256 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult GetEmployees()

{

List<Employee> employees = db.Employees.ToList();

return View(employees);

}

// Метод получения страницы жанров.

// Данная страница кэшируется на 256 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult GetGenres()

{

List<Genre> genres = db.Genres.ToList();

return View(genres);

}

// Метод получения страницы дисков.

// Данная страница кэшируется на 256 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult GetDiscs()

{

List<Disc> discs = db.Discs.ToList();

// Преобразование данных для удобного представления

List<DiscViewModel> models = new List<DiscViewModel>();

foreach (var disc in discs)

{

var genre = db.Genres.Where(elem => elem.Id == disc.Genre).First().Name;

models.Add(new DiscViewModel()

{

Id = disc.Id,

Country = disc.Country,

DateOfCreation = disc.DateOfCreation,

DateOfRecord = disc.DateOfRecord,

TypeOfDisc = disc.TypeOfDisc,

Creater = disc.Creater,

MainActor = disc.MainActor,

Name = disc.Name,

Price = disc.Price,

GenreName = genre

});

}

return View(models);

}

// Метод получения страницы Записей проката.

// Данная страница кэшируется на 256 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult GetRentalRecords()

{

List<RentalRecord> rentalRecords = db.RentalRecords.ToList();

// Преобразование данных для удобного представления

List<RentalRecordViewModel> models = new List<RentalRecordViewModel>();

foreach (var rentalRecord in rentalRecords)

{

string clentFIO = db.Clients.Where(item => item.Id == rentalRecord.ClientId).Select(item => item.Fio).First();

string emplFIO = db.Employees.Where(item => item.Id == rentalRecord.EmployeeId).Select(item => item.Fio).First();

string discName = db.Discs.Where(item => item.Id == rentalRecord.DiscId).Select(item => item.Name).First();

models.Add(new RentalRecordViewModel()

{

Id = rentalRecord.Id,

DateOfRent = rentalRecord.DateOfRent,

ClientFIO = clentFIO,

DateOfReturn = rentalRecord.DateOfReturn,

DiscName = discName,

PaymentCheck = rentalRecord.PaymentCheck,

ReturnCheck = rentalRecord.ReturnCheck,

EmployeeFIO = emplFIO

});

}

return View(models);

}

}

}

*Home/Index.cshtml*

@{

ViewData["Title"] = "Home Page";

}

<div class="text-center">

<h1>Список таблиц:</h1>

<a class="nav-link text-dark" **asp-area**="" **asp-controller**="Tables" **asp-action**="GetClients">Клиенты</a>

<a class="nav-link text-dark" **asp-area**="" **asp-controller**="Tables" **asp-action**="GetEmployees">Работники</a>

<a class="nav-link text-dark" **asp-area**="" **asp-controller**="Tables" **asp-action**="GetGenres">Жанры</a>

<a class="nav-link text-dark" **asp-area**="" **asp-controller**="Tables" **asp-action**="GetDiscs">Диски</a>

<a class="nav-link text-dark" **asp-area**="" **asp-controller**="Tables" **asp-action**="GetRentalRecords">Записи проката</a>

</div>

*Tables/GetClients.cshtml*

@model List<WebApp.Models.Client>

@{

ViewBag.Title = "Таблица клиентов";

}

@ViewBag.Title

:

<div>

<table border="1">

<thead>

<tr>

<td>

Id

</td>

<td>

ФИО

</td>

<td>

Номер

</td>

<td>

Паспорт

</td>

</tr>

</thead>

@foreach (var elem in Model)

{

<tr>

<td>

@elem.Id

</td>

<td>

@elem.Fio

</td>

<td>

@elem.Number

</td>

<td>

@elem.Pasport

</td>

</tr>

}

</table>

</div>

*Tables/GetEmployees.cshtml*

@model List<WebApp.Models.Employee>

@{

ViewBag.Title = "Таблица работников";

}

@ViewBag.Title

:

<div>

<table border="1">

<thead>

<tr>

<td>

Id

</td>

<td>

ФИО

</td>

<td>

Должность

</td>

<td>

Дата начала работы

</td>

</tr>

</thead>

@foreach (var elem in Model)

{

<tr>

<td>

@elem.Id

</td>

<td>

@elem.Fio

</td>

<td>

@elem.Position

</td>

<td>

@elem.DateOfWorkStart

</td>

</tr>

}

</table>

</div>

*Tables/GetDiscs.cshtml*

@model List<WebApp.Models.DiscViewModel>

@{

ViewBag.Title = "Таблица дисков";

}

@ViewBag.Title

:

<div>

<table border="1">

<thead>

<tr>

<td>

Id

</td>

<td>

Название фильма

</td>

<td>

Дата создания

</td>

<td>

Студия

</td>

<td>

Страна

</td>

<td>

Главный актер

</td>

<td>

Дата записи

</td>

<td>

Наименование жанра

</td>

<td>

Тип диска

</td>

<td>

Цена

</td>

</tr>

</thead>

@foreach (var elem in Model)

{

<tr>

<td>

@elem.Id

</td>

<td>

@elem.Name

</td>

<td>

@elem.DateOfCreation

</td>

<td>

@elem.Creater

</td>

<td>

@elem.Country

</td>

<td>

@elem.MainActor

</td>

<td>

@elem.DateOfRecord

</td>

<td>

@elem.GenreName

</td>

<td>

@elem.TypeOfDisc

</td>

<td>

@elem.Price

</td>

</tr>

}

</table>

</div>

*Tables/GetGenres.cshtml*

@model List<WebApp.Models.Genre>

@{

ViewBag.Title = "Таблица жанров";

}

@ViewBag.Title

:

<div>

<table border="1">

<thead>

<tr>

<td>

Id

</td>

<td>

Название

</td>

<td>

Описание

</td>

</tr>

</thead>

@foreach (var elem in Model)

{

<tr>

<td>

@elem.Id

</td>

<td>

@elem.Name

</td>

<td>

@elem.Description

</td>

</tr>

}

</table>

</div>

*Tables/RentalRecords.cshtml*

@model List<WebApp.Models.RentalRecordViewModel>

@{

ViewBag.Title = "Таблица записей проката";

}

@ViewBag.Title

:

<div>

<table border="1">

<thead>

<tr>

<td>

Id

</td>

<td>

ФИО клиента

</td>

<td>

Дата взятия

</td>

<td>

Дата возврата

</td>

<td>

Оплачено?

</td>

<td>

Возвращено?

</td>

<td>

Название фильма

</td>

<td>

ФИО сотрудника

</td>

</tr>

</thead>

@foreach (var elem in Model)

{

<tr>

<td>

@elem.Id

</td>

<td>

@elem.ClientFIO

</td>

<td>

@elem.DateOfRent

</td>

<td>

@elem.DateOfReturn

</td>

@if (elem.PaymentCheck == 1)

{

<td>

Да

</td>

}

else

{

<td>

Нет

</td>

}

@if (elem.ReturnCheck == 1)

{

<td>

Да

</td>

}

else

{

<td>

Нет

</td>

}

<td>

@elem.DiscName

</td>

<td>

@elem.EmployeeFIO

</td>

</tr>

}

</table>

</div>

Результаты:

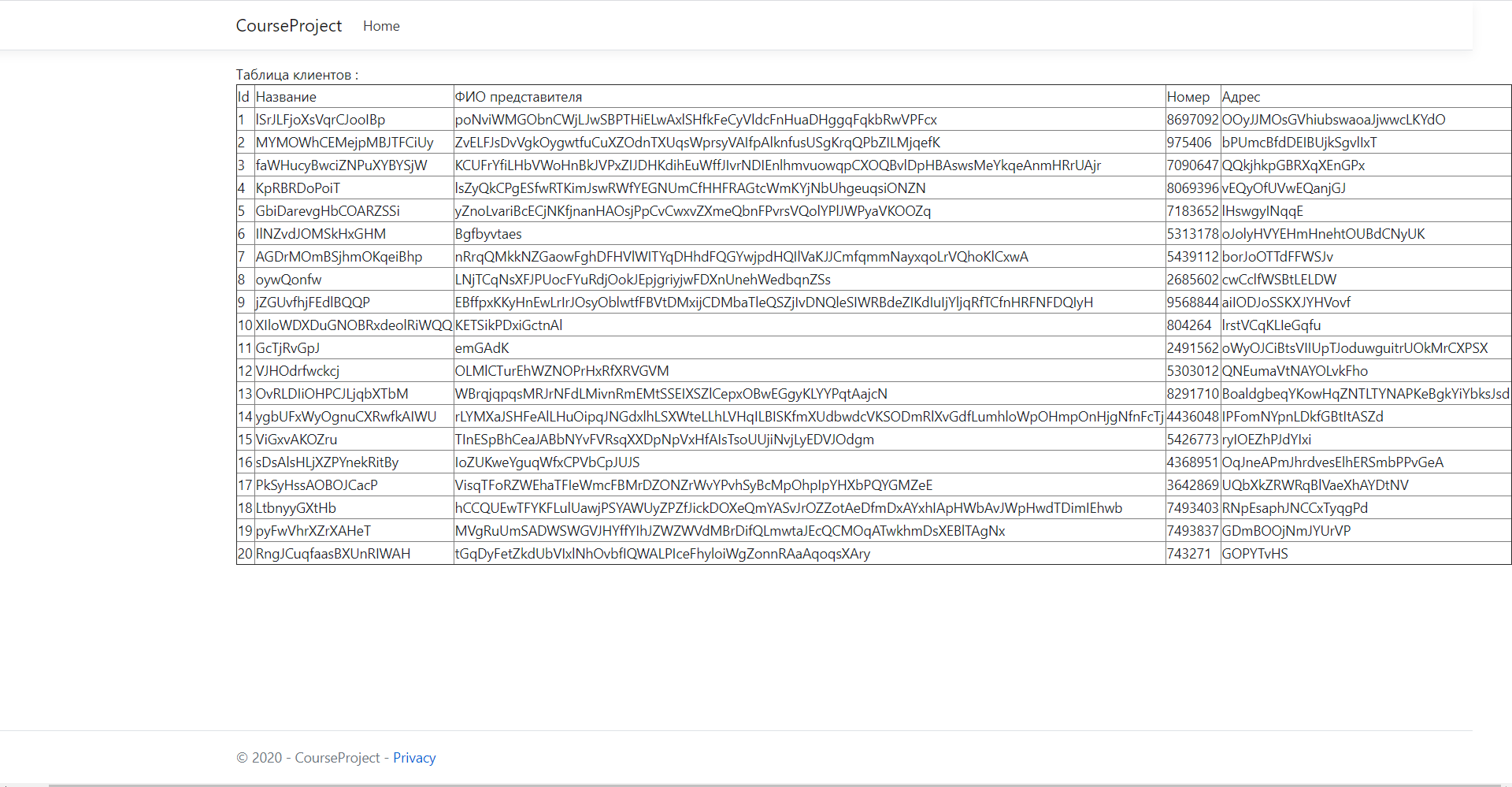


Рисунок 2 – вывод результата запроса, содержащего /*Clients*



Рисунок 3 – вывод результата запроса, содержащего /*Employees*

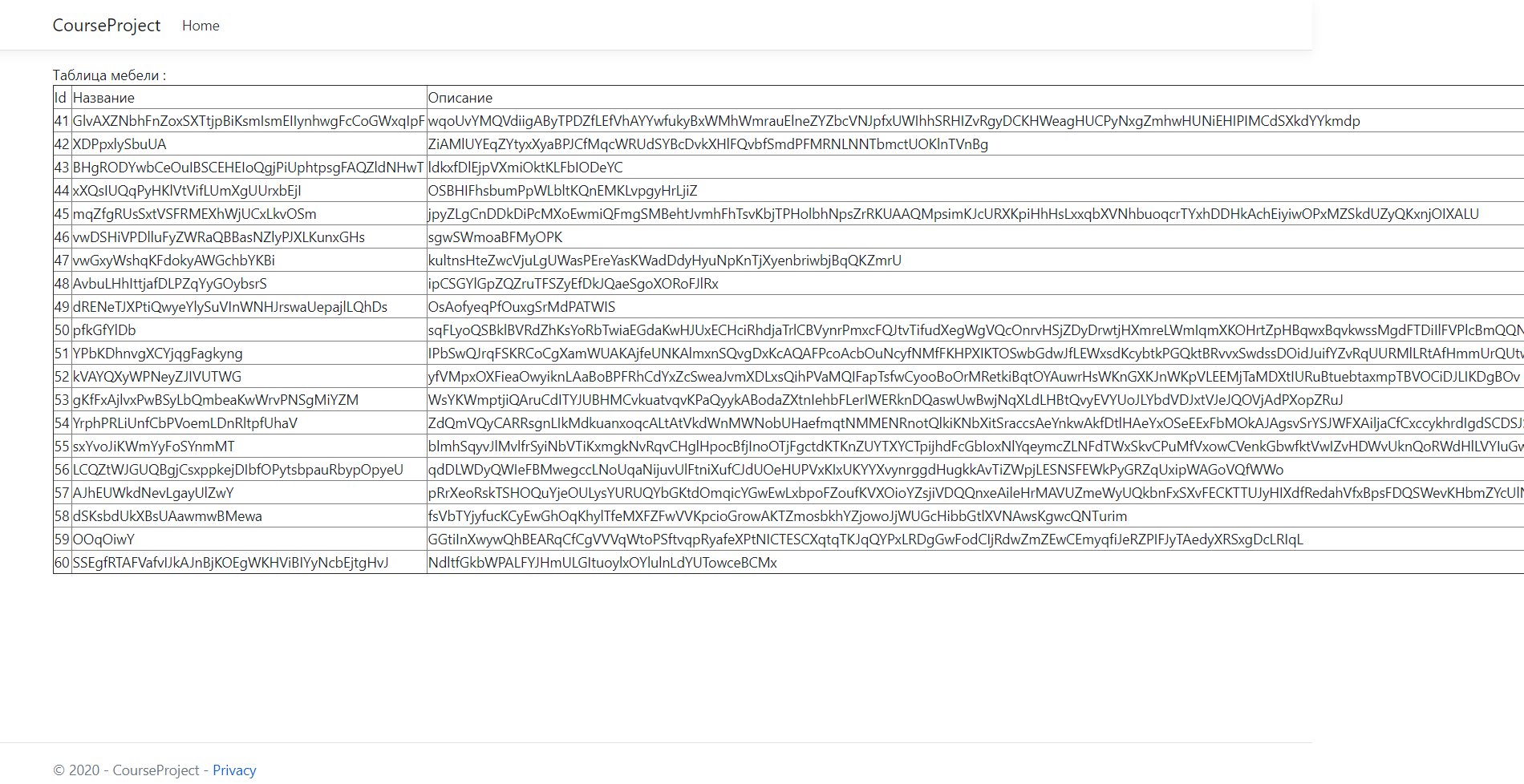


Рисунок 4 – вывод результата запроса, содержащего /*Discs*

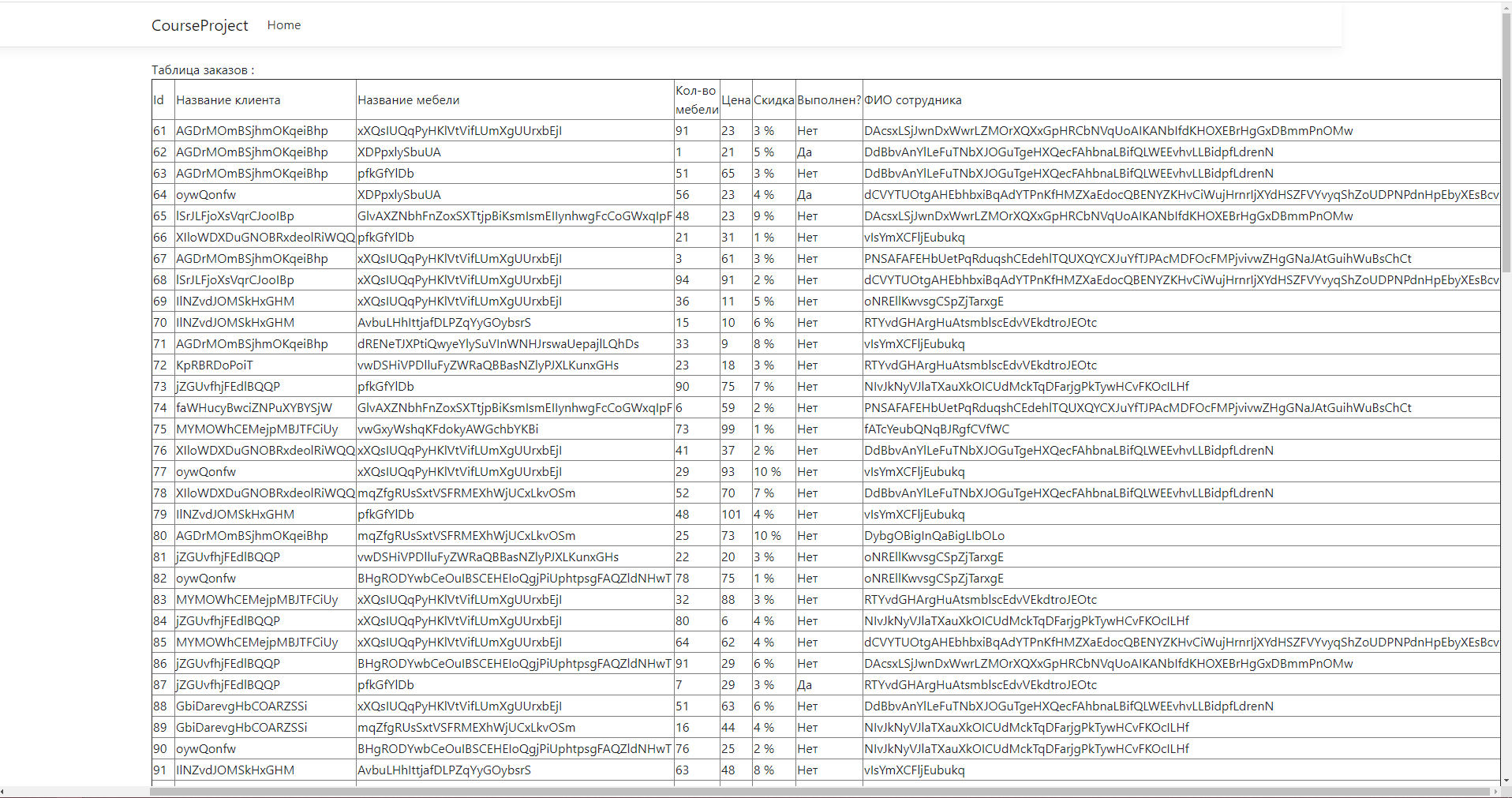


Рисунок 5 – вывод результата запроса, содержащего /*Genres*

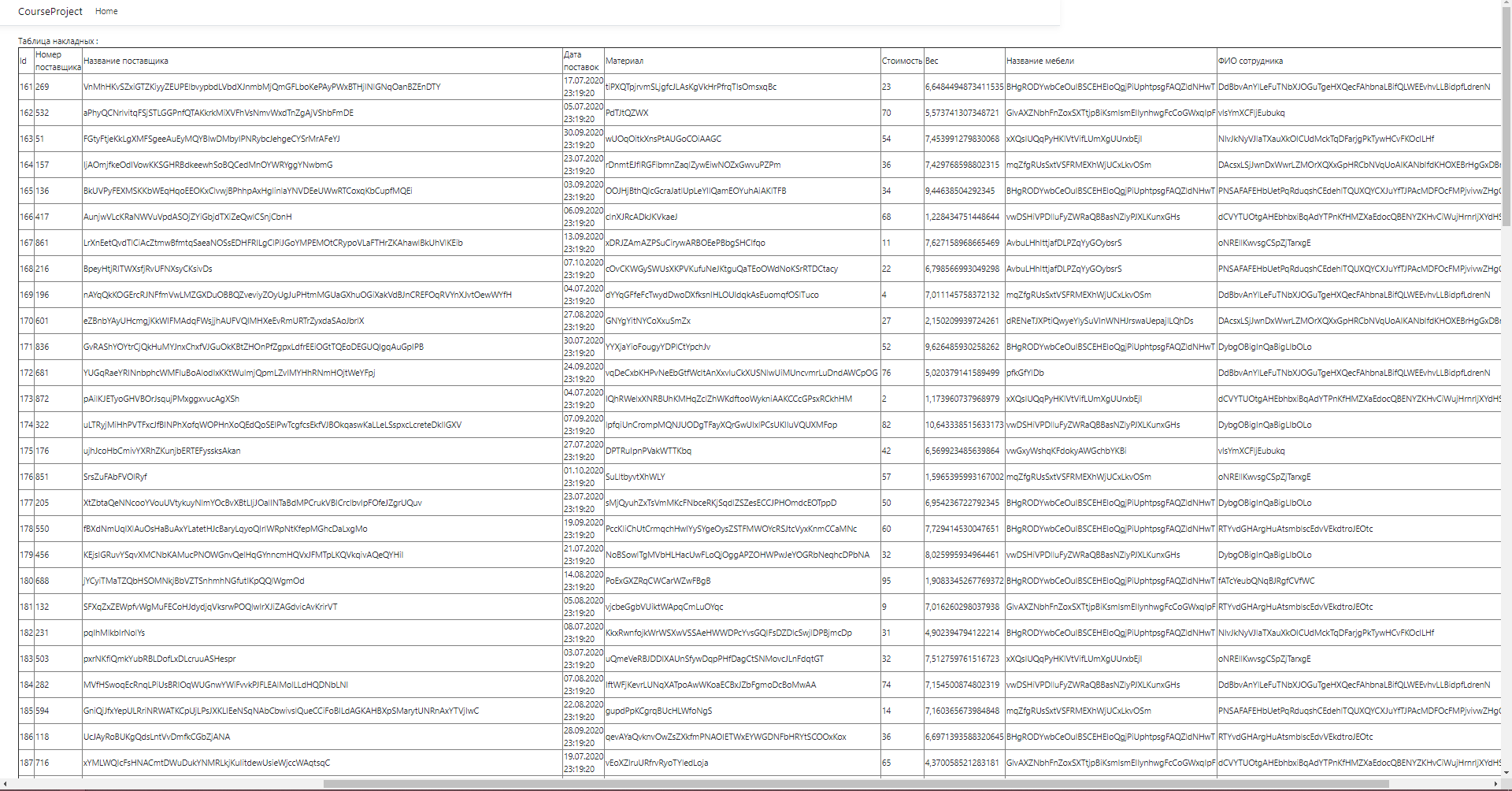
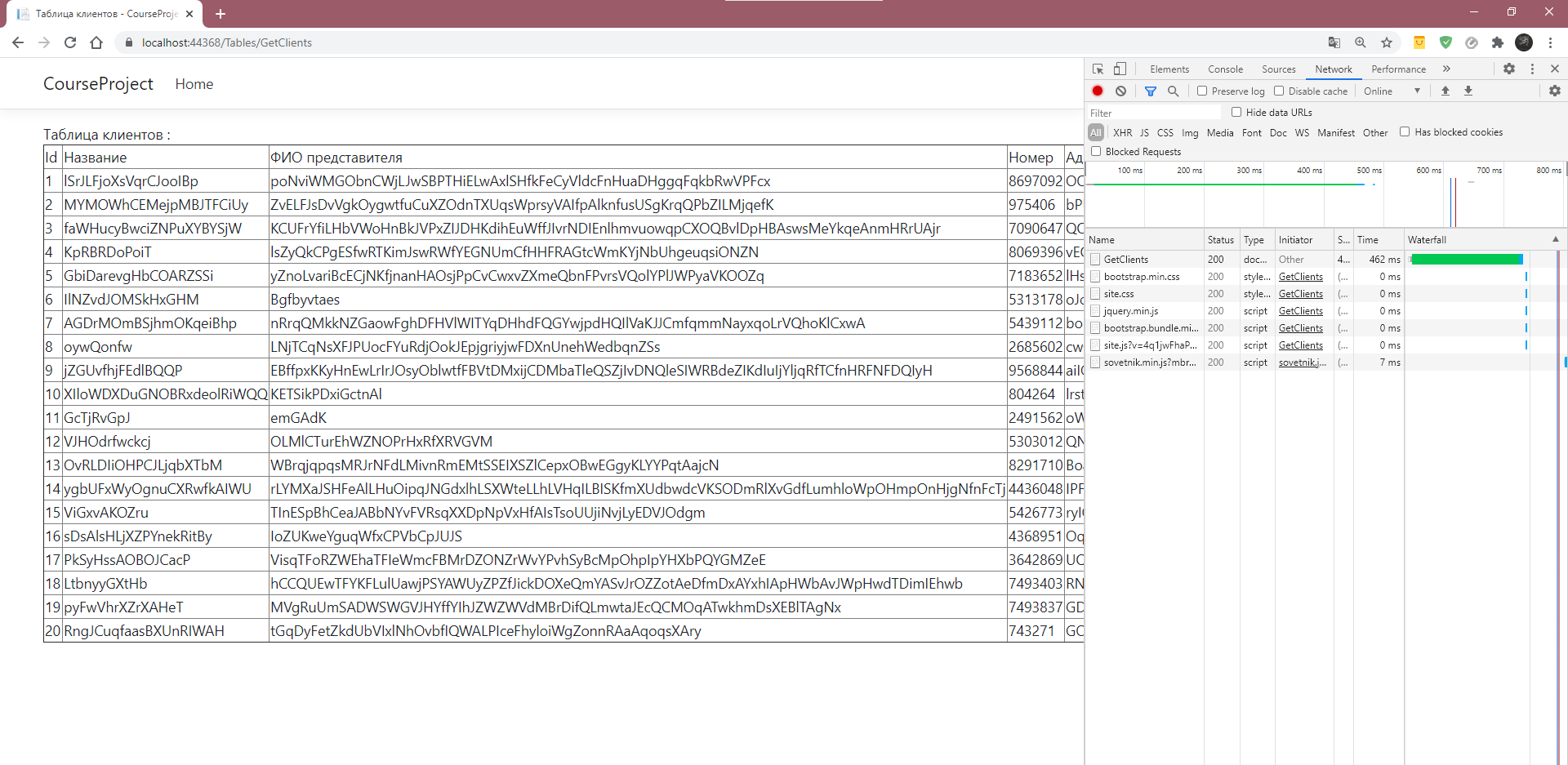


Рисунок 6 – вывод результата запроса, содержащего /*RentalRecords*



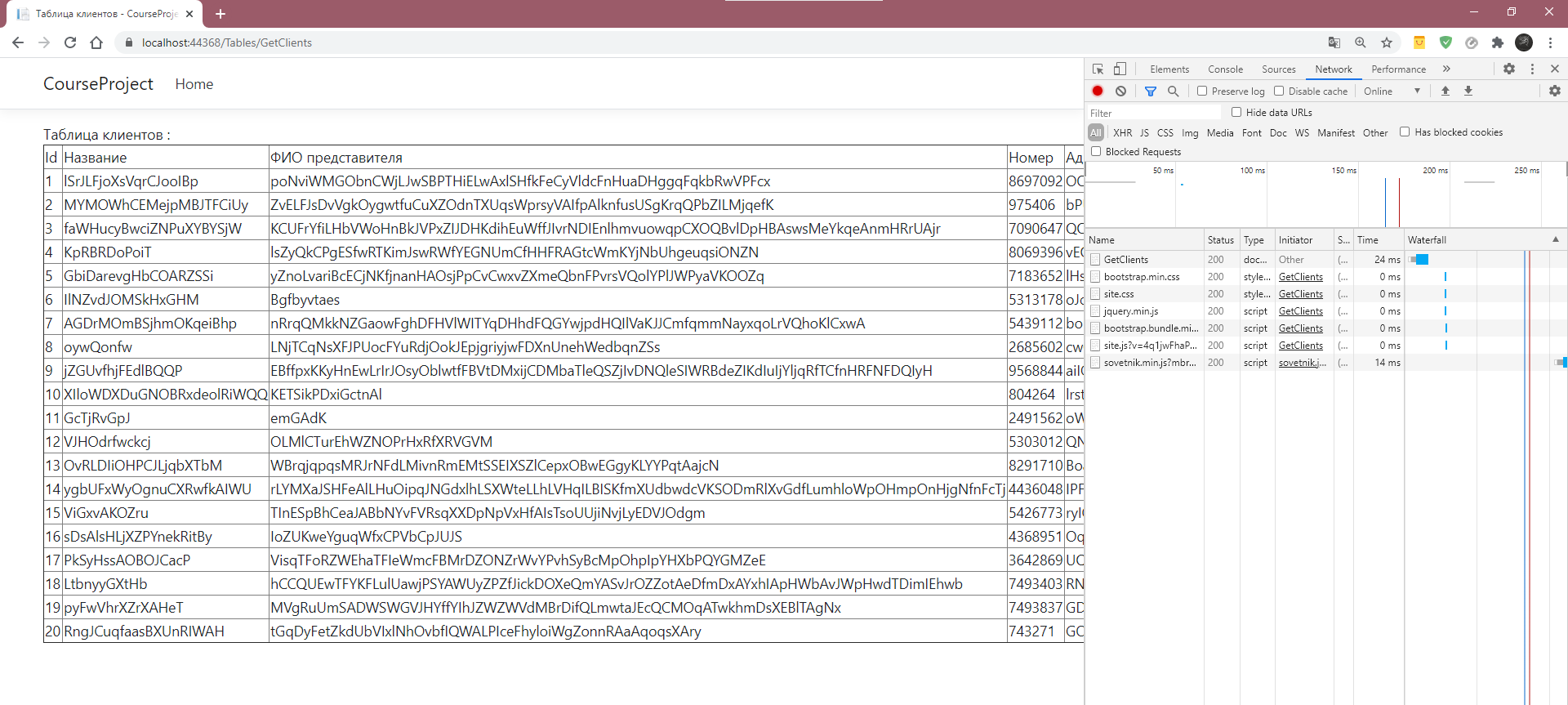


Рисунок 7-8 – результаты сравнения скорости создания страницы через кэш и последовательно

**Вывод:** в результате выполнения работы были изучена работа классов представлений, контроллеров и *ResponseCache*.