**9МИНИСТЕРСТВО 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»

Выполнила: студент гр. ИТП-21

Шеляхин М. С.

         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.

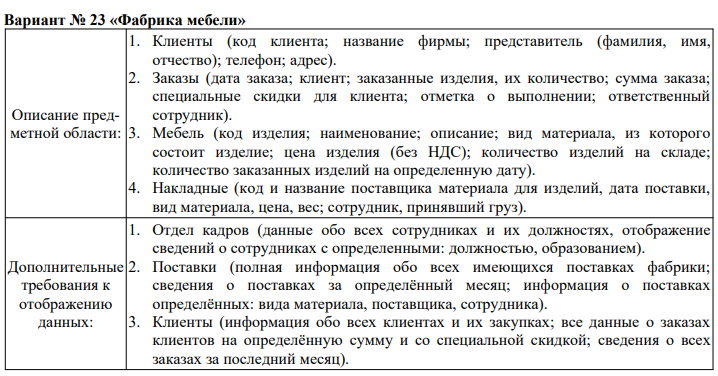
**

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:

*Startup.cs*

using CourseProject.MiddleWares;

using CourseProject.Models;

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;

namespace CourseProject

{

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 = 286

});

options.CacheProfiles.Add("NoCaching",

new CacheProfile()

{

Location = ResponseCacheLocation.None,

NoStore = true

});

});

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

services.AddDbContext<ApplicationContext>(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.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

// Модель, представляющая запись в таблице Клиенты

public class Client

{

[Key]

public int Id { get; set; }

public string Name { get; set; }

public string RepresentativeFIO { get; set; }

public int Number { get; set; }

public string Address { get; set; }

public Client() { }

}

}

*Employee.cs*

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

// Модель, представляющая запись в таблице Работники

public class Employee

{

[Key]

public int Id { get; set; }

public string FIO { get; set; }

public string Position { get; set; }

public string Education { get; set; }

public Employee() { }

}

}

*Furniture.cs*

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

// Модель, представляющая запись в таблице Мебель

public class Furniture

{

[Key]

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public string Material { get; set; }

public decimal Price { get; set; }

public int Count { get; set; }

public Furniture() { }

}

}

*Order.cs*

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

// Модель, представляющая запись в таблице Заказы

public class Order

{

[Key]

public int Id { get; set; }

public int ClientId { get; set; }

public int FurnitureId { get; set; }

public int FurnitureCount { get; set; }

public decimal Price { get; set; }

public int DiscountPercent { get; set; }

public int IsCompleted { get; set; }

public int EmployeeId { get; set; }

public Order() { }

}

}

*Waybill.cs*

using System;

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

// Модель, представляющая запись в таблице Накладные

public class Waybill

{

[Key]

public int Id { get; set; }

public int ProviderId { get; set; }

public string ProviderName { get; set; }

public DateTime DateOfSupply { get; set; }

public string Material { get; set; }

public decimal Price { get; set; }

public double Weight { get; set; }

public int FurnitureId { get; set; }

public int EmployeeId { get; set; }

public Waybill() { }

}

}

*OrderViewModel.cs*

namespace CourseProject.Models

{

// Модель отображения записей из таблицы Заказы с заменой внешних ключей смысловыми данными

public class OrderViewModel

{

public int Id { get; set; }

public string ClientName { get; set; }

public string FurnitureName { get; set; }

public int FurnitureCount { get; set; }

public decimal Price { get; set; }

public int DiscountPercent { get; set; }

public bool IsCompleted { get; set; }

public string EmployeeFIO { get; set; }

}

}

*WaybillViewModel.cs*

using System;

namespace CourseProject.Models

{

// Модель отображения записей из таблицы Накладные с заменой внешних ключей смысловыми данными

public class WaybillViewModel

{

public int Id { get; set; }

public int ProviderId { get; set; }

public string ProviderName { get; set; }

public DateTime DateOfSupply { get; set; }

public string Material { get; set; }

public decimal Price { get; set; }

public double Weight { get; set; }

public string FurnitureName { get; set; }

public string EmployeeFIO { get; set; }

}

}

*ApplicationContext.cs*

using Microsoft.EntityFrameworkCore;

namespace CourseProject.Models

{

// Класс контекста данных

public class ApplicationContext : DbContext

{

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

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

public DbSet<Furniture> Furniture { get; set; }

public DbSet<Order> Orders { get; set; }

public DbSet<Waybill> Waybills { get; set; }

public ApplicationContext(DbContextOptions<ApplicationContext> options) : base(options)

{

}

}

}

Листинг строки подключения к базе данных из конфигурационного файла:

*appsettings.json*

"ConnectionStrings": {

"SqlServerConnection": "Data Source = DESKTOP-G8EVE4H; Database=FurnitureFabric; 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 CourseProject.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(ApplicationContext 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 name = "";

string repFIO = "";

int numb;

string address = "";

// Создание 40 записей в таблице

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

{

// Получение Id

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

// Создание названия клиента-организации

int rand = randObj.Next(3, 25);

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

{

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

}

// Создание ФИО представителя

rand = randObj.Next(17, 100);

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

{

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

}

// Создание номера клиента

numb = randObj.Next(1000000, 9999999);

// Создание адреса

rand = randObj.Next(5, 40);

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

{

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

}

// Добавление записи в таблицу

db.Clients.Add(new Client { Id = clientId, Name = name, RepresentativeFIO = repFIO, Number = numb, Address = address});

}

db.SaveChanges();

}

// Проверка на наличие записей в таблице Работники

if (!db.Employees.Any())

{

int employeeId;

string fio = "";

string position = "";

string education = "";

// 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 образования

rand = randObj.Next(20, 200);

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

{

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

}

// Добавление записи в таблицу

db.Employees.Add(new Employee { Id = employeeId, FIO = fio, Position = position, Education = education });

}

db.SaveChanges();

}

// Проверка на наличие записей в таблице Мебель

if (!db.Furniture.Any())

{

int furnitId;

string name = "";

string descr = "";

int count;

decimal price;

string material = "";

// Создание 40 записей

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

{

// Получение Id

furnitId = db.Furniture.Count() + 1;

// Создание названия мебели

int rand = randObj.Next(3, 51);

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

{

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

}

// Создание описания мебели

rand = randObj.Next(17, 200);

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

{

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

}

// Создание цены на мебель

price = (decimal)randObj.NextDouble() \* 10;

// Создание кол-ва мебели

count = randObj.Next(1, 21);

// Создание материала мебели

rand = randObj.Next(3, 61);

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

{

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

}

// Добавление записи в таблицу

db.Furniture.Add(new Furniture { Id = furnitId, Name = name, Description = descr, Material = material, Price = price, Count = count });

}

db.SaveChanges();

}

// Проверка на наличие записей в таблице Заказы

if (!db.Orders.Any())

{

int orderId;

int clientId;

int furnitId;

int discount;

int isComplete;

int count;

decimal price;

int emplId;

// Создание 200 записей

for (int id = 1; id <= 200; id++)

{

// Получение Id

orderId = db.Orders.Count() + 1;

// Получение Id клиента

clientId = randObj.Next(1, 41);

// Получение Id мебели

furnitId = randObj.Next(1, 41);

// Создание скидки

discount = randObj.Next(0, 50);

// Создание состояния выполнения

isComplete = randObj.Next(0, 2);

// Создание кол-ва мебели

count = randObj.Next(1, 21);

// Создание стоимости заказа

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

// Получение Id работника

emplId = randObj.Next(1, 41);

// Добавление записи в таблицу

db.Orders.Add(new Order { Id = orderId, ClientId = clientId, FurnitureId = furnitId, DiscountPercent = discount, IsCompleted = isComplete, FurnitureCount = count, Price = price, EmployeeId = emplId });

}

db.SaveChanges();

}

// Проверка на наличие записей в таблице Накладные

if (!db.Waybills.Any())

{

int waybillId;

int providerId;

string providerName = "";

DateTime date;

string material = "";

float weight;

decimal price;

int furnitId;

int emplId;

// Создание 200 записей

for (int id = 1; id <= 200; id++)

{

// Получение Id накладной

waybillId = db.Waybills.Count() + 1;

// Создание номера поставщика

providerId = randObj.Next(1, 101);

// Создание названия поставщика

int rand = randObj.Next(7, 100);

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

{

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

}

// Получение Id мебели

furnitId = randObj.Next(1, 41);

// Получение времени

date = DateTime.Now;

// Получение материала мебели

material = db.Furniture.Where(item => item.Id == furnitId).Select(item => item.Material).FirstOrDefault();

// Создание веса поставки

weight = (float)randObj.NextDouble() \* 10;

// Создание стоимости поставки

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

// Получение Id работника

emplId = randObj.Next(1, 41);

// Добавление записи в таблицу

db.Waybills.Add(new Waybill { Id = waybillId, ProviderId = providerId, ProviderName = providerName, DateOfSupply = date, Material = material, Price = price, Weight = weight, FurnitureId = furnitId, EmployeeId = emplId});

}

db.SaveChanges();

}

}

}

}

*DbInitializerMiddleWare.cs и DbInitializerExtentions.cs*

using System.Threading.Tasks;

using CourseProject.Models;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

namespace CourseProject.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, ApplicationContext 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 CourseProject.Models;

namespace CourseProject.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 CourseProject.Models;

using Microsoft.AspNetCore.Mvc;

namespace CourseProject.Controllers

{

// Контроллер представления страниц записей из таблиц

public class TablesController : Controller

{

// Объект контекста данных

private readonly ApplicationContext db;

public TablesController(ApplicationContext applicationContext)

{

db = applicationContext;

}

// Метод получения страницы клиентов.

// Данная страница кэшируется на 286 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult GetClients()

{

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

return View(clients);

}

// Метод получения страницы работников.

// Данная страница кэшируется на 286 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult GetEmployees()

{

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

return View(employees);

}

// Метод получения страницы мебели.

// Данная страница кэшируется на 286 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult GetFurniture()

{

List<Furniture> furnitures = db.Furniture.ToList();

return View(furnitures);

}

// Метод получения страницы заказов.

// Данная страница кэшируется на 286 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult GetOrders()

{

List<Order> orders = db.Orders.ToList();

// Преобразование данных для удобного представления

List<OrderViewModel> orderViewModels = new List<OrderViewModel>();

foreach (var order in orders)

{

string clientName = db.Clients.Where(item => item.Id == order.ClientId).Select(item => item.Name).First();

string furnitName = db.Furniture.Where(item => item.Id == order.FurnitureId).Select(item => item.Name).First();

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

orderViewModels.Add(new OrderViewModel()

{

Id = order.Id,

ClientName = clientName,

FurnitureName = furnitName,

IsCompleted = order.IsCompleted == 1 ? true : false,

DiscountPercent = order.DiscountPercent,

FurnitureCount = order.FurnitureCount,

Price = order.Price,

EmployeeFIO = emplFIO

});

}

return View(orderViewModels);

}

// Метод получения страницы накладных.

// Данная страница кэшируется на 286 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult GetWaybills()

{

List<Waybill> waybills = db.Waybills.ToList();

// Преобразование данных для удобного представления

List<WaybillViewModel> waybillViewModels = new List<WaybillViewModel>();

foreach (var waybill in waybills)

{

string furnitName = db.Furniture.Where(item => item.Id == waybill.FurnitureId).Select(item => item.Name).First();

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

waybillViewModels.Add(new WaybillViewModel()

{

Id = waybill.Id,

ProviderId = waybill.ProviderId,

ProviderName = waybill.ProviderName,

DateOfSupply = waybill.DateOfSupply,

Material = waybill.Material,

Price = waybill.Price,

Weight = waybill.Weight,

FurnitureName = furnitName,

EmployeeFIO = emplFIO

});

}

return View(waybillViewModels);

}

}

}

*Home/Index.cshtml*

@{

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

}

<div class="text-center">

<h1 class="display-4">Welcome to the fourth task</h1>

<h2>Список таблиц в базе данных:</h2>

<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**="GetFurniture">Мебель</a>

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

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

</div>

*Tables/GetClients.cshtml*

@model List<CourseProject.Models.Client>

@{

ViewBag.Title = "Таблица клиентов";

}

@ViewBag.Title

:

<div>

<table border="1">

<thead>

<tr>

<td>

Id

</td>

<td>

Название

</td>

<td>

ФИО представителя

</td>

<td>

Номер

</td>

<td>

Адрес

</td>

</tr>

</thead>

@foreach (var elem in Model)

{

<tr>

<td>

@elem.Id

</td>

<td>

@elem.Name

</td>

<td>

@elem.RepresentativeFIO

</td>

<td>

@elem.Number

</td>

<td>

@elem.Address

</td>

</tr>

}

</table>

</div>

*Tables/GetEmployees.cshtml*

@model List<CourseProject.Models.Employee>

@{

ViewBag.Title = "Таблица работников";

}

@ViewBag.Title

:

<div>

<table border="1">

<thead>

<tr>

<td>

Id

</td>

<td>

ФИО

</td>

<td>

Должность

</td>

<td>

Образование

</td>

</tr>

</thead>

@foreach (var elem in Model)

{

<tr>

<td>

@elem.Id

</td>

<td>

@elem.FIO

</td>

<td>

@elem.Position

</td>

<td>

@elem.Education

</td>

</tr>

}

</table>

</div>

*Tables/GetFurniture.cshtml*

@model List<CourseProject.Models.Furniture>

@{

ViewBag.Title = "Таблица мебели";

}

@ViewBag.Title

:

<div>

<table border="1">

<thead>

<tr>

<td>

Id

</td>

<td>

Название

</td>

<td>

Описание

</td>

<td>

Материал

</td>

<td>

Цена

</td>

<td>

Количество

</td>

</tr>

</thead>

@foreach (var elem in Model)

{

<tr>

<td>

@elem.Id

</td>

<td>

@elem.Name

</td>

<td>

@elem.Description

</td>

<td>

@elem.Material

</td>

<td>

@elem.Price

</td>

<td>

@elem.Count

</td>

</tr>

}

</table>

</div>

*Tables/GetOrders.cshtml*

@model List<CourseProject.Models.OrderViewModel>

@{

ViewBag.Title = "Таблица заказов";

}

@ViewBag.Title

:

<div>

<table border="1">

<thead>

<tr>

<td>

Id

</td>

<td>

Название клиента

</td>

<td>

Название мебели

</td>

<td>

Кол-во мебели

</td>

<td>

Цена

</td>

<td>

Скидка

</td>

<td>

Выполнен?

</td>

<td>

ФИО сотрудника

</td>

</tr>

</thead>

@foreach (var elem in Model)

{

<tr>

<td>

@elem.Id

</td>

<td>

@elem.ClientName

</td>

<td>

@elem.FurnitureName

</td>

<td>

@elem.FurnitureCount

</td>

<td>

@elem.Price

</td>

<td>

@elem.DiscountPercent

%

</td>

@if (elem.IsCompleted)

{

<td>

Да

</td>

}

else

{

<td>

Нет

</td>

}

<td>

@elem.EmployeeFIO

</td>

</tr>

}

</table>

</div>

*Tables/GetWaybills.cshtml*

@model List<CourseProject.Models.WaybillViewModel>

@{

ViewBag.Title = "Таблица накладных";

}

@ViewBag.Title

:

<div>

<table border="1">

<thead>

<tr>

<td>

Id

</td>

<td>

Номер поставщика

</td>

<td>

Название поставщика

</td>

<td>

Дата поставок

</td>

<td>

Материал

</td>

<td>

Стоимость

</td>

<td>

Вес

</td>

<td>

Название мебели

</td>

<td>

ФИО сотрудника

</td>

</tr>

</thead>

@foreach (var elem in Model)

{

<tr>

<td>

@elem.Id

</td>

<td>

@elem.ProviderId

</td>

<td>

@elem.ProviderName

</td>

<td>

@elem.DateOfSupply

</td>

<td>

@elem.Material

</td>

<td>

@elem.Price

</td>

<td>

@elem.Weight

</td>

<td>

@elem.FurnitureName

</td>

<td>

@elem.EmployeeFIO

</td>

</tr>

}

</table>

</div>

Результаты:

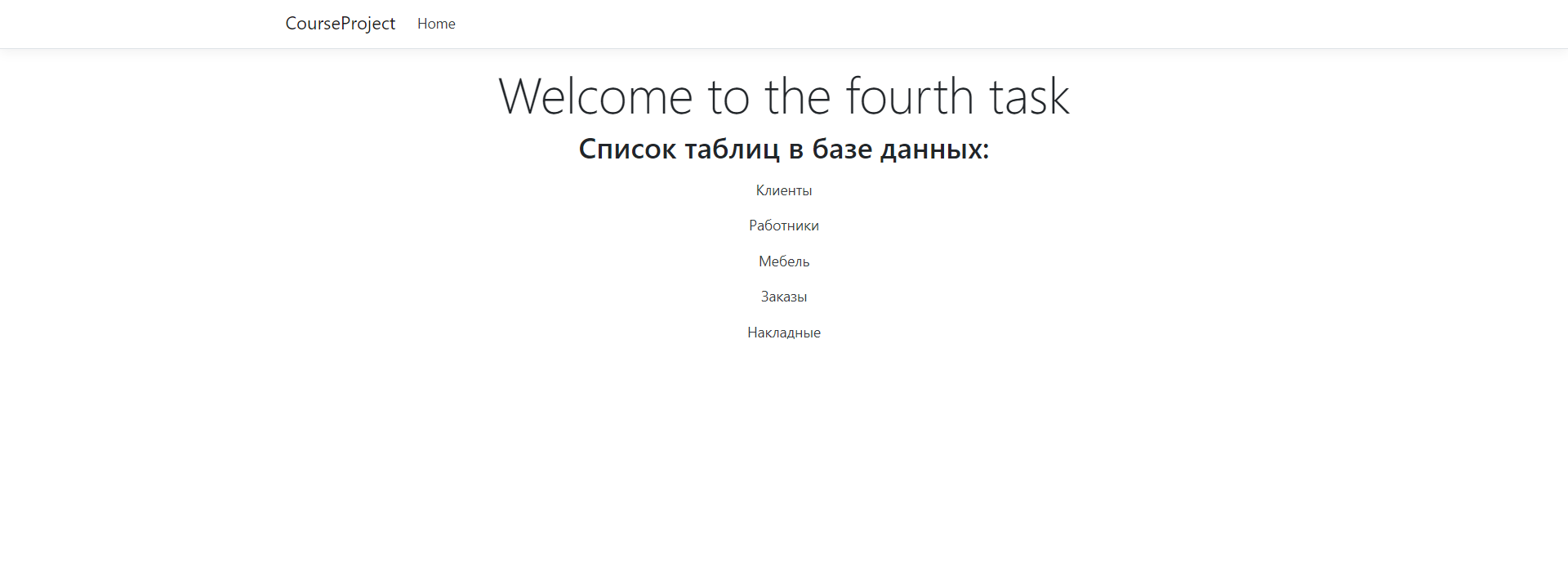


Рисунок 2 – окно меню

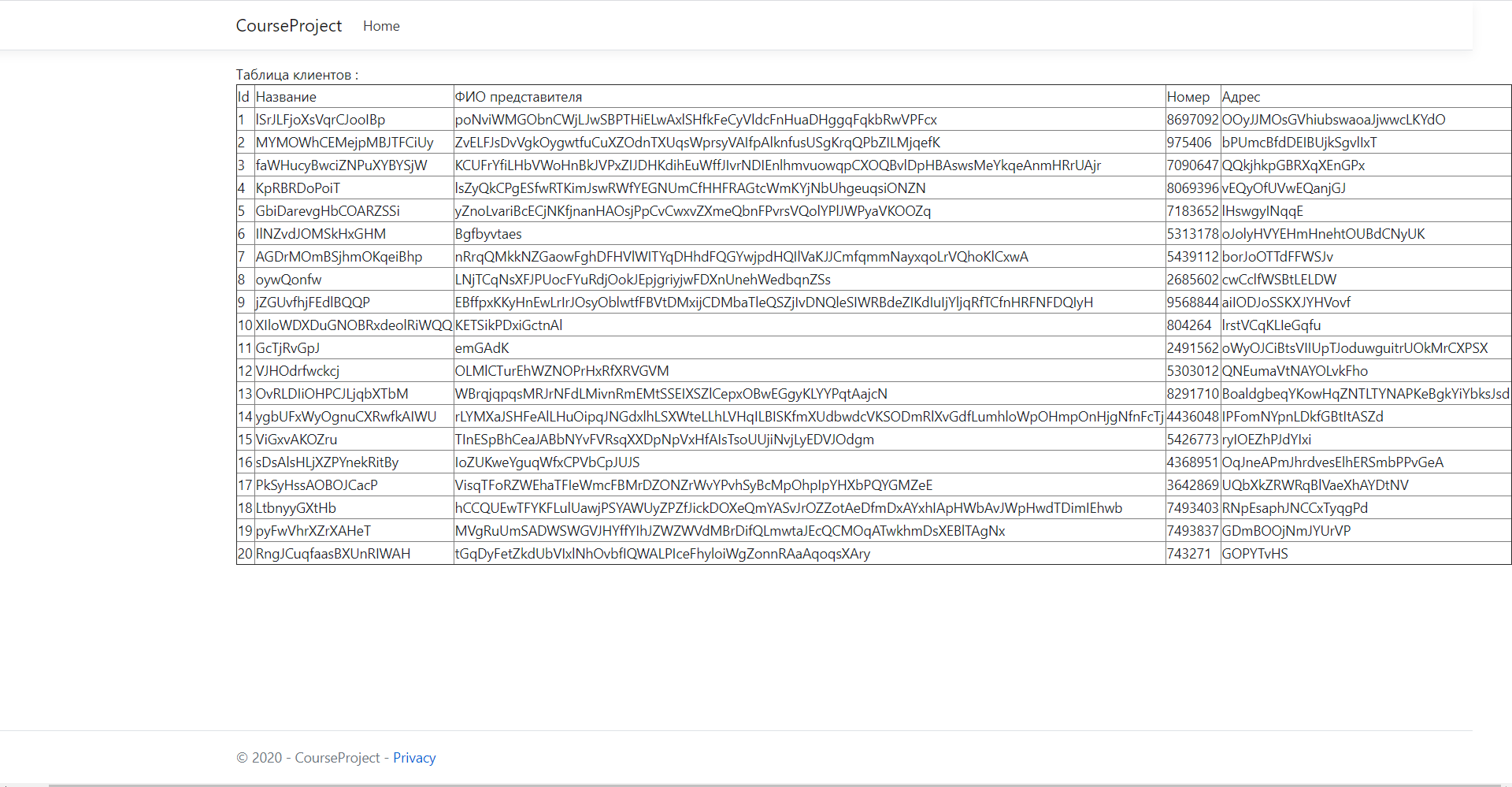


Рисунок 3 – вывод результата запроса, содержащего /*Clients*



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

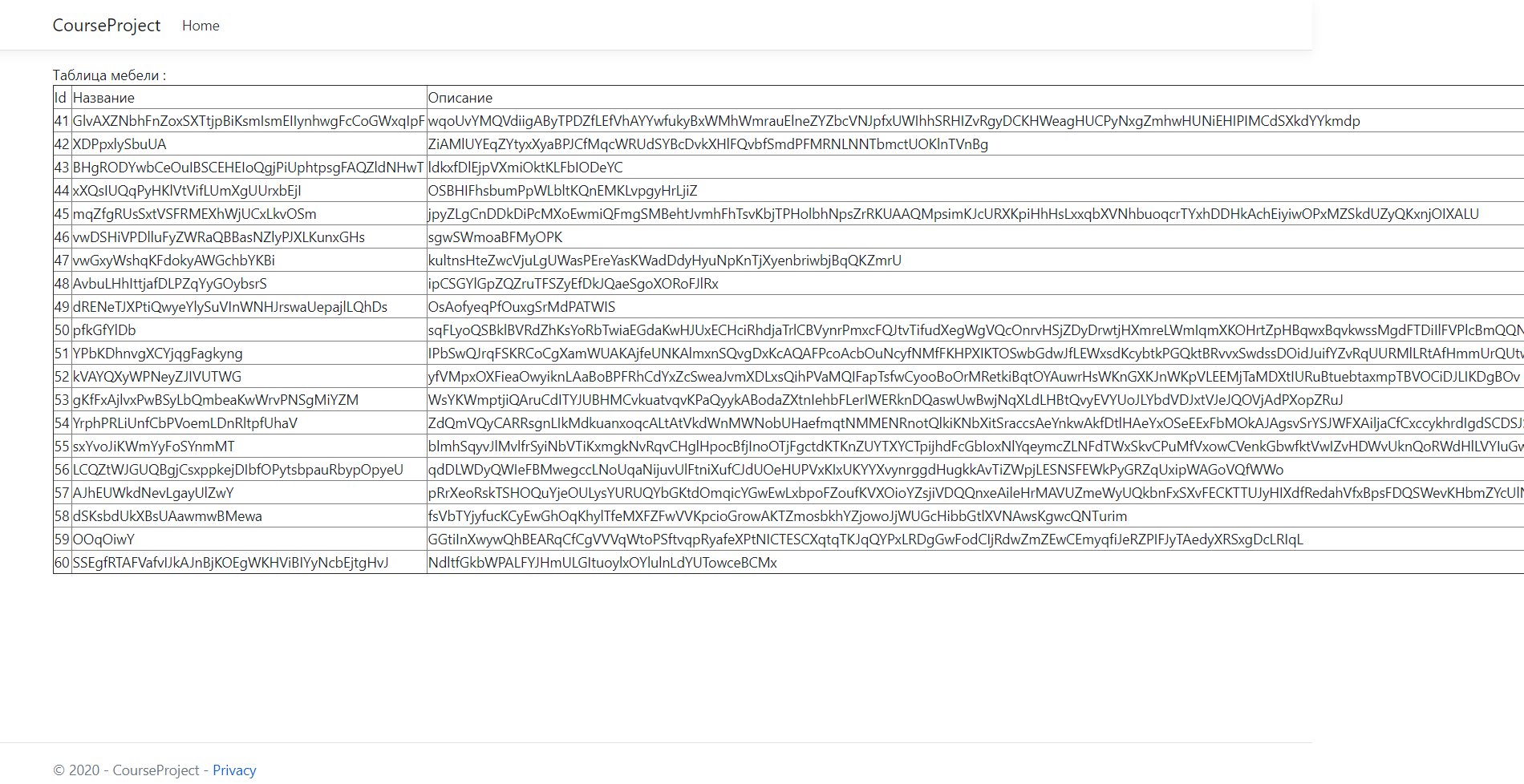




Рисунок 5-6 – вывод результата запроса, содержащего /*Furniture*

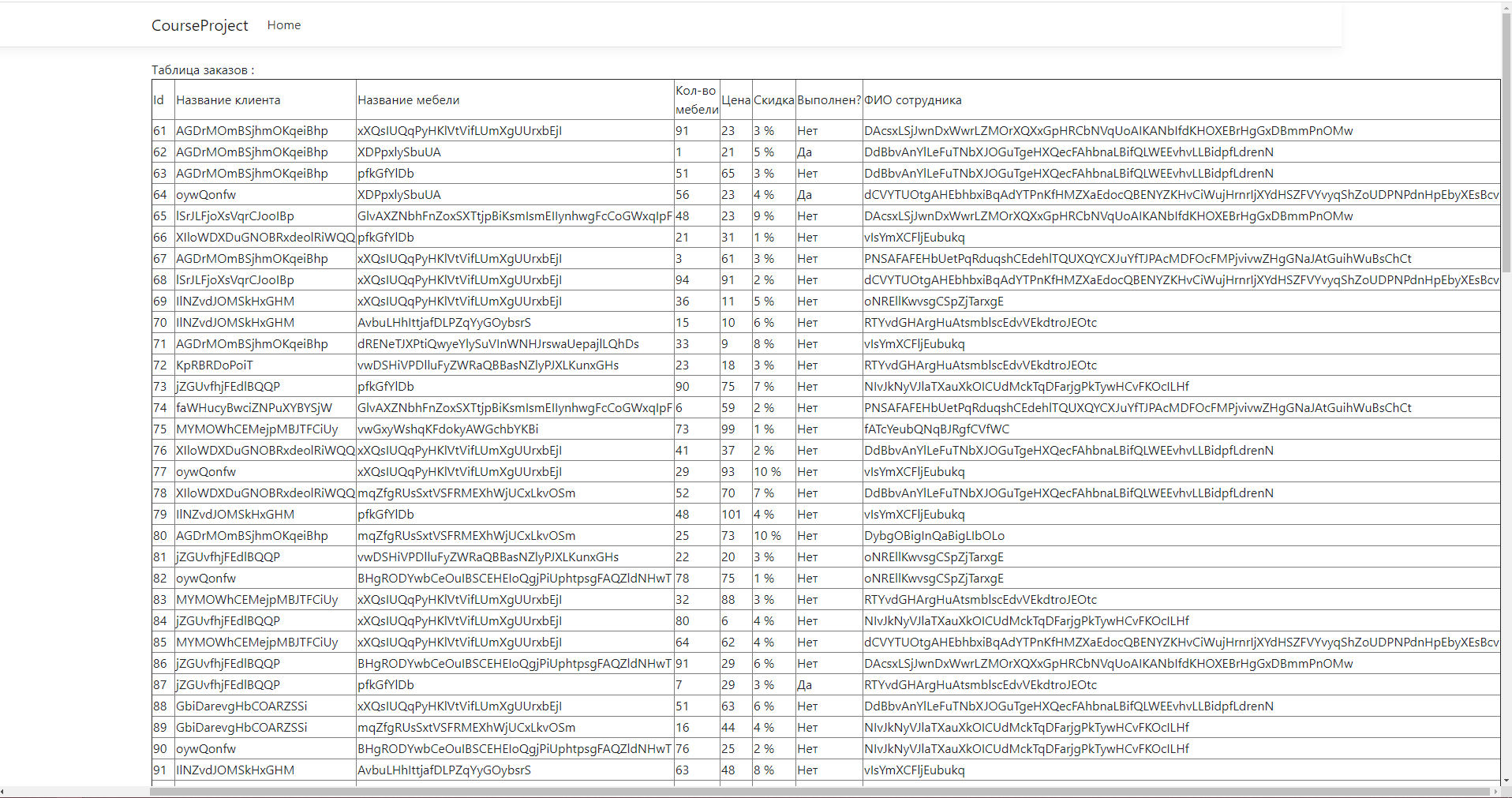
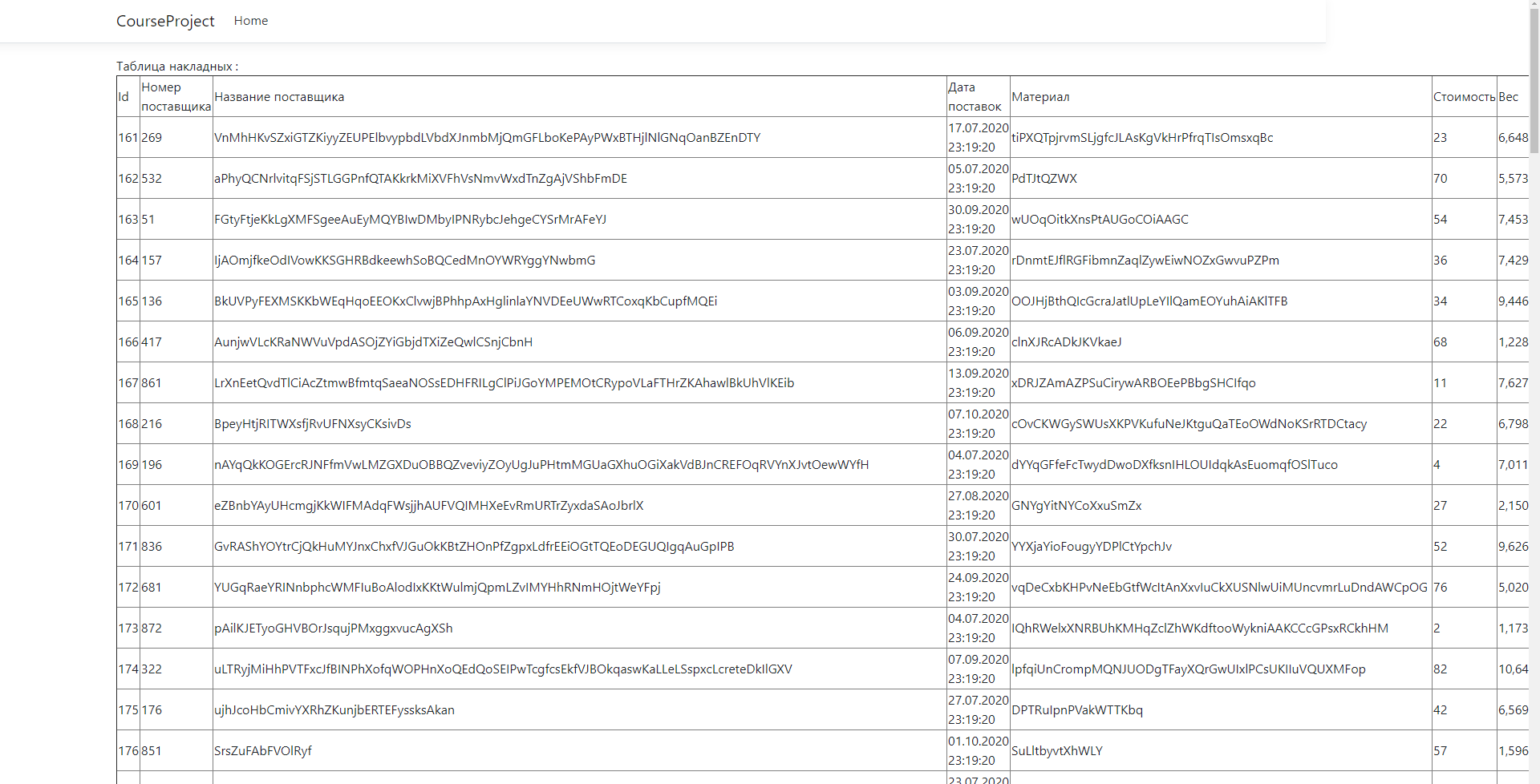


Рисунок 7 – вывод результата запроса, содержащего /*Orders*



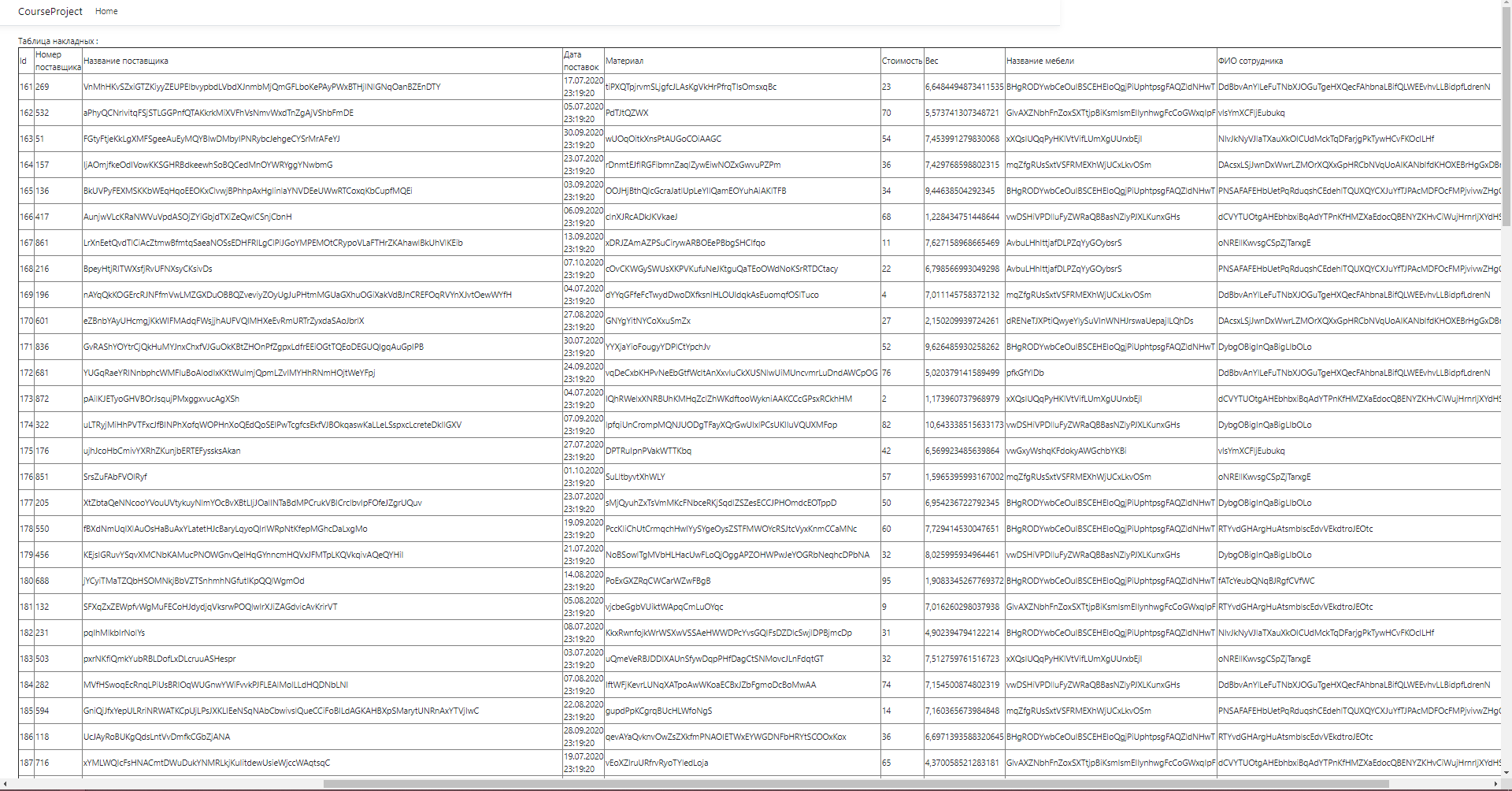
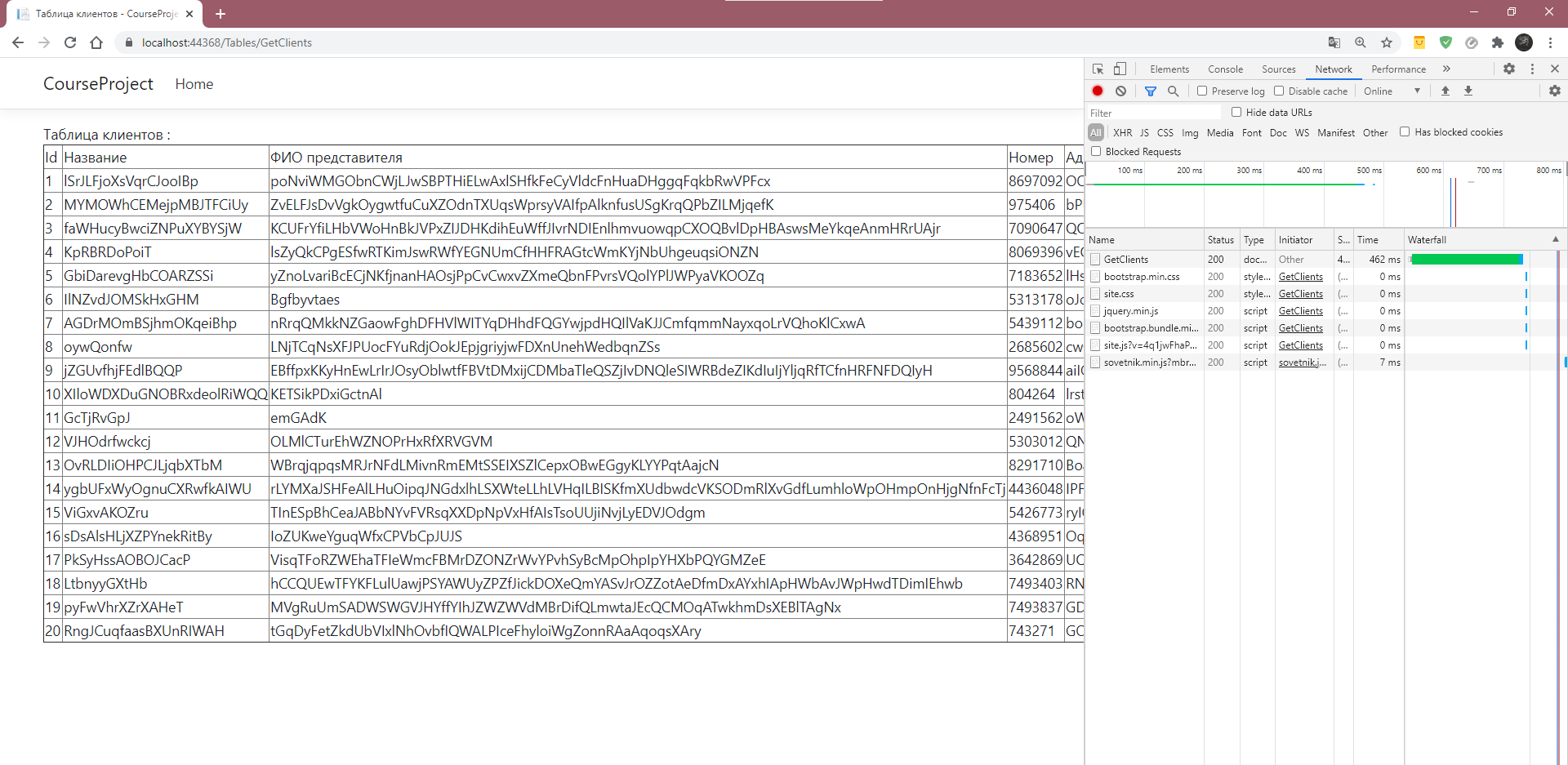


Рисунок 8-9 – вывод результата запроса, содержащего /*Waybills*



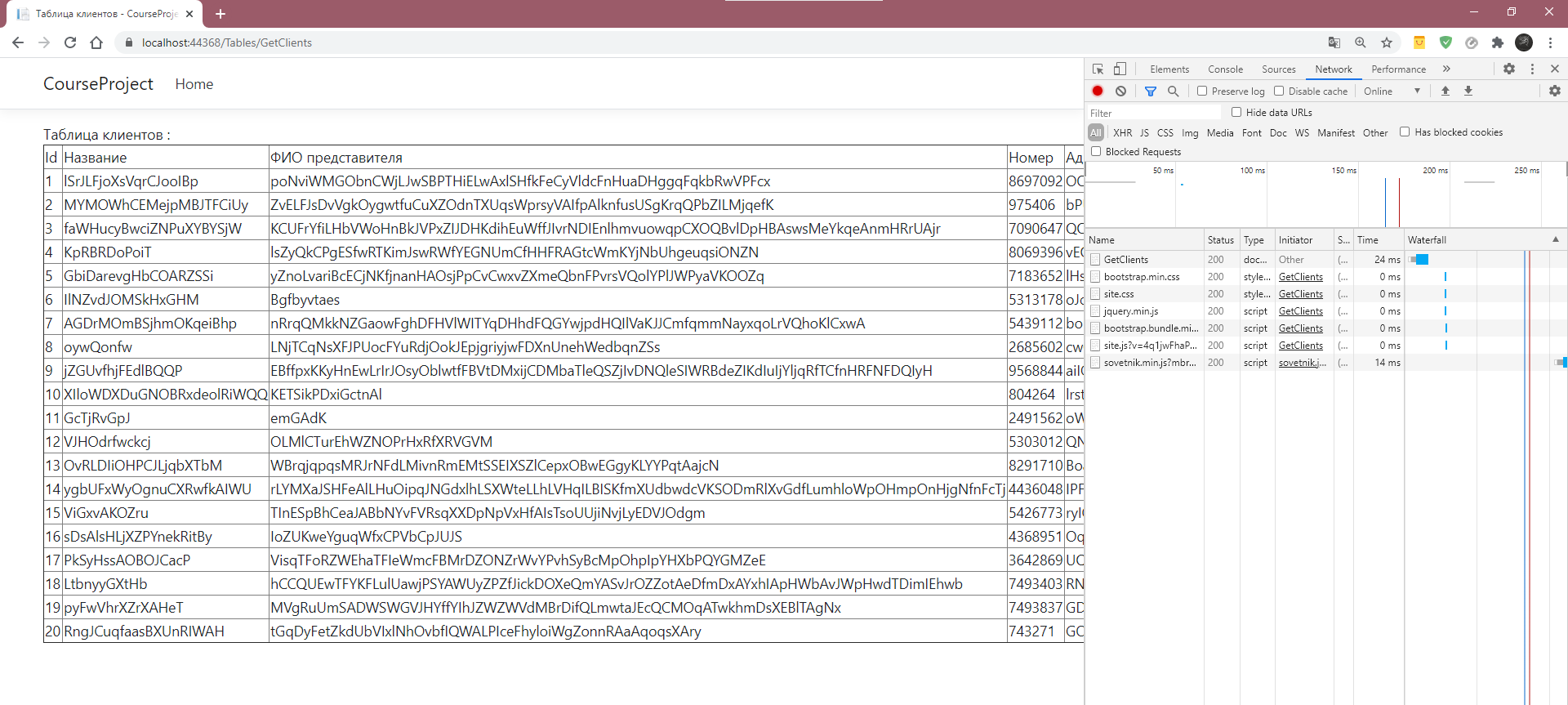


Рисунок 17-20 – результаты сравнения скорости создания страницы через кэш и последовательно

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