*Федеральное государственное бюджетное образовательное учреждение высшего образования*

*«Владимирский государственный университет*

*имени Александра Григорьевича и Николая Григорьевича Столетовых»*

*Кафедра информационных систем и программной инженерии*

***ПОЯСНИТЕЛЬНАЯ ЗАПИСКА***

***к курсовому проекту по дисциплине   
"Технологии программирования"***

***на тему***

*Проектирование и разработка* *программной системы*

*учета пациентов в информационной системе «Поликлиника»*

*Выполнил: студент гр. ИСТ-220*

*Крылов Е.М.*

*Принял: Доцент*

*Кафедры ИСПИ*

*Вершинин В.В.*

*Владимир, 2022*

**АННОТАЦИЯ**

В рамках курсового проекта производилась разработка программной системы учета пациентов в информационной системе «Поликлиника». Данный проект содержит в себе три этапа проектирования, состоящих из описания предметной области, описания функционала, словаря предметной области, а также схем данных и разрабатываемого кода. Для реализации информационной системы использовался язык программирования C#.

Разработанная информационная система может быть использована для хранения данных о пациентах, врачах, медицинских регистраторов, лаборантов, врачах функциональной диагностики, заявок на прием у терапевта и узкого специалиста, направлений на анализ или обследование, анализах, обследованиях. Она содержит в себе модели данных, базу данных, контроллеры, представления, благодаря которым процесс работы с программной системой автоматизирован.

Курсовой проект представлен на 46 страницах, содержит в себе 9 рисунков, 7 приложений, и 3 использованных источника.

**ANNOTATION**

As part of the course project, the development of a software system for registering patients in the information system "Polyclinic" was carried out. This project contains three stages of design, consisting of a description of the subject area, a description of the functionality, a dictionary of the subject area, as well as data schemas and developed code. The C# programming language was used to implement the information system.

The developed information system can be used to store data about patients, doctors, medical registrars, laboratory assistants, functional diagnostics doctors, applications for an appointment with a therapist and a narrow specialist, referrals for analysis or examination, analyzes, examinations. It contains data models, a database, controllers, views, thanks to which the process of working with a software system is automated.

Course project presented at 46 pages, contains 9 drawings, 7 applications, and 3 sources used.

**СОДЕРЖАНИЕ**

1. ВВЕДЕНИЕ 5

2. Описание предметной области 7

2 Проектирование системы 9

2.1 Минимальный набор функций 9

2.2 Функциональные требования 9

2.3 Диаграмма прецедентов 10

2.4 Диаграмма классов 11

2.5 Диаграммы некоторых процессов 12

2.6 Диаграмма состояний 14

2.7 Макеты типовых компонентов системы 15

4 РЕАЛИЗАЦИЯ СИСТЕМЫ 18

4.1 Технологический стек 18

4.2 Реализация модели на сервере 18

4.3 Реализация аутентификации 18

4.4 Модели системы 18

4.5 Реализация контроллеров 32

4.6 Реализация представлений 32

4.7 Реализация микросервиса 32

5 ЗАКЛЮЧЕНИЕ 33

6 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 34

ПРИЛОЖЕНИЕ А 35

ПРИЛОЖЕНИЕ Б 36

ПРИЛОЖЕНИЕ В 40

ПРИЛОЖЕНИЕ Г 42

ПРИЛОЖЕНИЕ Д 44

ПРИЛОЖЕНИЕ Е 45

ПРИЛОЖЕНИЕ Ж 46

1. ВВЕДЕНИЕ

В ходе выполнения курсового проекта требовалось разработать программную систему учета пациентов в информационной системе «Поликлиника» для автоматизации учета и хранения обращений пациентов в поликлинику.

Цель работы: разработать программную систему учета пациентов в информационной системе «Поликлиника», избавиться от большинства бумажных документов, например карточек и направлений.

Для достижения поставленной цели необходимо решить следующие задачи:

* хранение данных о пациентах, врачах, лаборантах, врачей функциональной диагностики, медицинских регистраторах;
* хранение данных о заявках на прием у врача;
* хранение данных о направлениях на обследование, анализ;
* хранение данных о результатах обследования, результатов анализов;
* хранение данных об осмотрах, проведенных врачом;
* хранение данных о выписанных рецептах;
* предоставление пациенту возможности просматривать данные о состоянии заявки на прием у терапевта, своем диагнозе, выписанном рецепте, результате осмотра;
* предоставление работнику регистратуры возможности обрабатывать подачу заявок на прием к врачам;
* предоставление врачу возможности выписывать направления на обследования, консультировать пациента по лечению, выписывать рецепт, ставить диагноз, выписывать направления на анализ;
* предоставление врачам функциональной диагностики возможности вести отчет по обследованию систем и органов человека;
* предоставление лаборантам возможности вести отчет по сбору биологических жидкостей и продуктов жизнедеятельности людей;
* уведомление пациента о состоянии заявки на прием у врача;

2. Описание предметной области

В данной работе в качестве предметной области рассматривается программная система учета пациентов в информационной системе «Поликлиника».

Информационная система «Поликлиника» позволяет:

1. пациенту просматривать данные о состоянии заявки на прием у терапевта, своем диагнозе, выписанном рецепте, проведенном осмотре;
2. любому человеку зарегистрироваться как пользователь системы;
3. зарегистрированному пользователю присвоить роль;
4. пользователю с определенной ролью внести необходимые данные о себе;
5. пациенту записываться на прием к терапевту;
6. врачу выписывать направления на обследования, консультировать пациента по лечению, выписывать рецепт, ставить диагноз, выписывать направления на анализ;
7. врачам функциональной диагностики вести отчет по обследованию систем и органов человека;
8. лаборантам вести отчет по сбору биологических жидкостей и продуктов жизнедеятельности людей;
9. работнику регистратуры обрабатывать заявки на прием у врачей;
10. автоматически высылать уведомления на почту о принятии заявки регистратурой.

**2.1 Пользователи разрабатываемой подсистемы**

*Врач (Doctor) (узкий специалист и терапевт) -* специалист с высшим медицинским образованием, использующий свои навыки, знания и опыт в профилактике и лечении заболеваний, поддержании нормальной жизнедеятельности организма человека.

*Пациент (Patient) -* больной, лечащийся у врача.

*Медицинский регистратор (Receptionist)* - сотрудник поликлиники, работающий в регистратуре. Он встречает пациентов, осуществляет распределение клиентов по специалистам и запись на прием, а также ведет картотеку организации.

*Лаборант (Assistant)* – научно-технический сотрудник лаборатории медицинского, учебного или научного учреждения. Он выполняет практическую работу в области научных исследований, разработок и анализа.

*Врач функциональной диагностики (Functional Diagnostics Doctor)* – это специалист с высшим медицинским образованием, владеющий методами исследования с применением современной медицинской аппаратуры, направленными на выявление отклонений нормальной деятельности систем органов.

**2.2 Словарь предметной области;**

*Направление –* это обращение врача к другому врачу с целью проконсультироваться относительно состояния здоровья пациента, определения или уточнение диагноза, тактики лечения (начать, продолжить, изменить, прекратить начатое лечение) или трудоспособности пациента.

*Анализ –* этоисследование биологических жидкостей человека.

*Заявка — это* заявление с указанием на потребность в чём ни будь.

*Диагноз — это медицинское заключение* о состоянии здоровья обследуемого, а также сущности болезни и состоянии пациента, выраженное в принятой медицинской терминологии и основанное на всестороннем систематическом изучении пациента.

*Предварительный диагноз –* это диагноз, формулируемый непосредственно при обращении больного за медицинской помощью на основании данных, полученных до начала систематического обследования больного. Необходим для разработки плана обследования и начальных этапов лечения.

2 Проектирование системы

В ходе анализа предметной области была выбрана монолитная архитектура с использованием отдельного микросервиса для отправки уведомлений. Отделение от монолита одного микросервиса обусловлено тем, что в будущем данный микросервис будет реализовывать так же запланированные, отложенные уведомления на почту, и необходимо будет разгрузить центральный сервер, на котором будет располагаться монолитная система.

Исходный код представлен в репозитории Github: <https://github.com/TopRuiner/MIS>

2.1 Минимальный набор функций

* поддержка (авторизация) различных типов пользователей (администратор, врач, пациент);
* поддержка справочников диагнозов (адм.);
* поддержка справочника врачей (адм.);
* выписка рецепта (врач);
* подача заявок и просмотр их состояния (пациент);
* назначение пациенту лечащего врача (система).

2.2 Функциональные требования

* зарегистрироваться;
* авторизироваться;
* выйти из системы;
* «регистрация» как пациент;
* «регистрация» как врач;
* «регистрация» как медицинский регистратор;
* «регистрация» как лаборант;
* «регистрация» как врач функциональной диагностики;
* подача заявления на осмотр у врача;
* подтверждение заявления;
* отклонение заявления;
* отсылка уведомлений о заявлении;
* оформление данных об осмотре;
* создание записи об анализе;
* создание записи об обследовании;
* поиск по спискам обследований, анализов, пациентов, врачей, лаборантов, врачей функциональной диагностики, направлений, заявок;
* редактирование ролей пользователей;
* изменение состояния заявок.

2.3 Диаграмма прецедентов

В ходе анализа и проектирования функционала системы была создана диаграмма прецедентов. В качестве инструмента создания диаграммы была использована программа Microsoft Visio. Диаграмма прецедентов представлена в приложении Е

**2.3.1 Описание прецедентов**

Расширенное описание прецедента «Осмотр пациента»

Название: «Осмотр пациента»

Предусловие: пациент имеет подтвержденную заявку на прием у врача, пациент зарегистрирован в системе.

Действующее лица: пациент, врач.

Основной поток: врач вносит в систему данные об осмотре, рецепте, диагнозе.

Альтернативный поток: терапевт выписывает направления на анализы, обследования.

Постусловие: -

Расширенное описание прецедента «Обработка заявок»

Название: «Обработка заявок»

Предусловие: пациент подал заявку на прием ко врачу. Пациент выбрал дату визита и врача, к которому собирается обратиться

Действующее лицо: Медицинский регистратор

Основной поток: Медицинский регистратор редактирует заявку добавляет кабинет. Состояние заявки меняется на «Подтверждена». Пациенту отправляется оповещение на почту о подтверждении заявки.

Альтернативный поток: -

Постусловие: -

2.4 Диаграмма классов

Диаграмма классов представлена в приложении А. Сущности «Assistants», «Receptionists», «Doctors», «FunctionalDiagnosticsDoctor», «Patients» — это пользователи, которые внесли личные данные в систему и им присвоена соответствующая роль. В системе так же представлена сущность «Admin» однако для нее нет отельной таблицы и поэтому она реализована только ролью зарегистрированного пользователя. Таблицы с постфиксом «-Referrals» - таблицы для различных направлений. Таблицы с префиксом «Asp-» - таблицы, которые сгенерировал Visual Studio в процессе создания миграций и подключения к проекту библиотеки Identity[1].

**2.5 Диаграммы некоторых процессов**



Рисунок 1 – прохождение осмотра



Рисунок 2 – создание направления



Рисунок 3 – создание анализа



Рисунок 4 – создание обследования

**2.6 Диаграмма состояний**

В системе существует сущность, которая может иметь несколько состояний. Эта сущность – направление на прием у врача. При создании пациентом направления ему сразу присваивается статус «ожидает подтверждения». При необходимости пациент может отменить направление до того момента пока работник регистратуры его не подтвердит или отменит его. Если пациент не отменил заявку работник регистратуры дополняет заявку номером кабинета и подтверждает ее. В это время на почту пациенту приходит письмо с данными заявки и оповещением что она подтверждена. Затем пациент приходит на осмотр, где врач просматривает заявку и указывает что она была использована и осмотр произведен.

Диаграмма состояний для данной сущности представлена в приложении Ж.

**2.7 Макеты типовых компонентов системы**

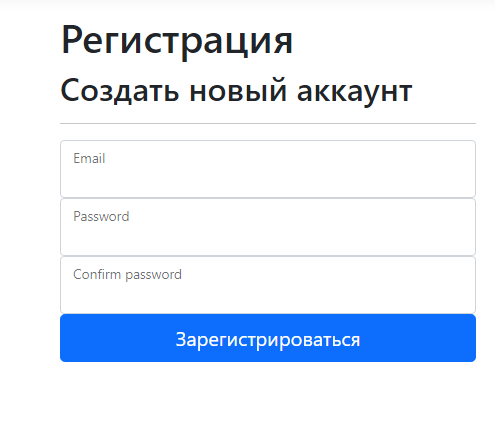
****

Рисунок 5 – типовая форма регистрации

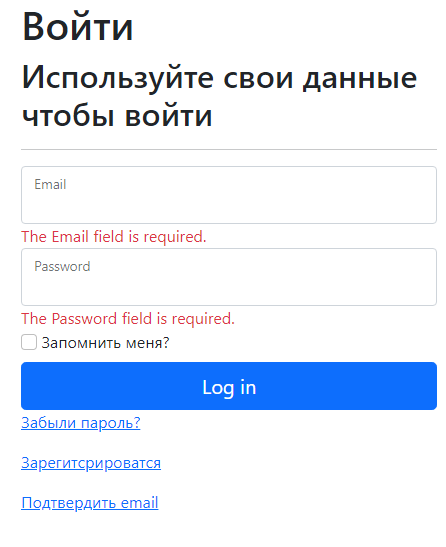


Рисунок 6 – типовая форма входа в аккаунт с валидацией

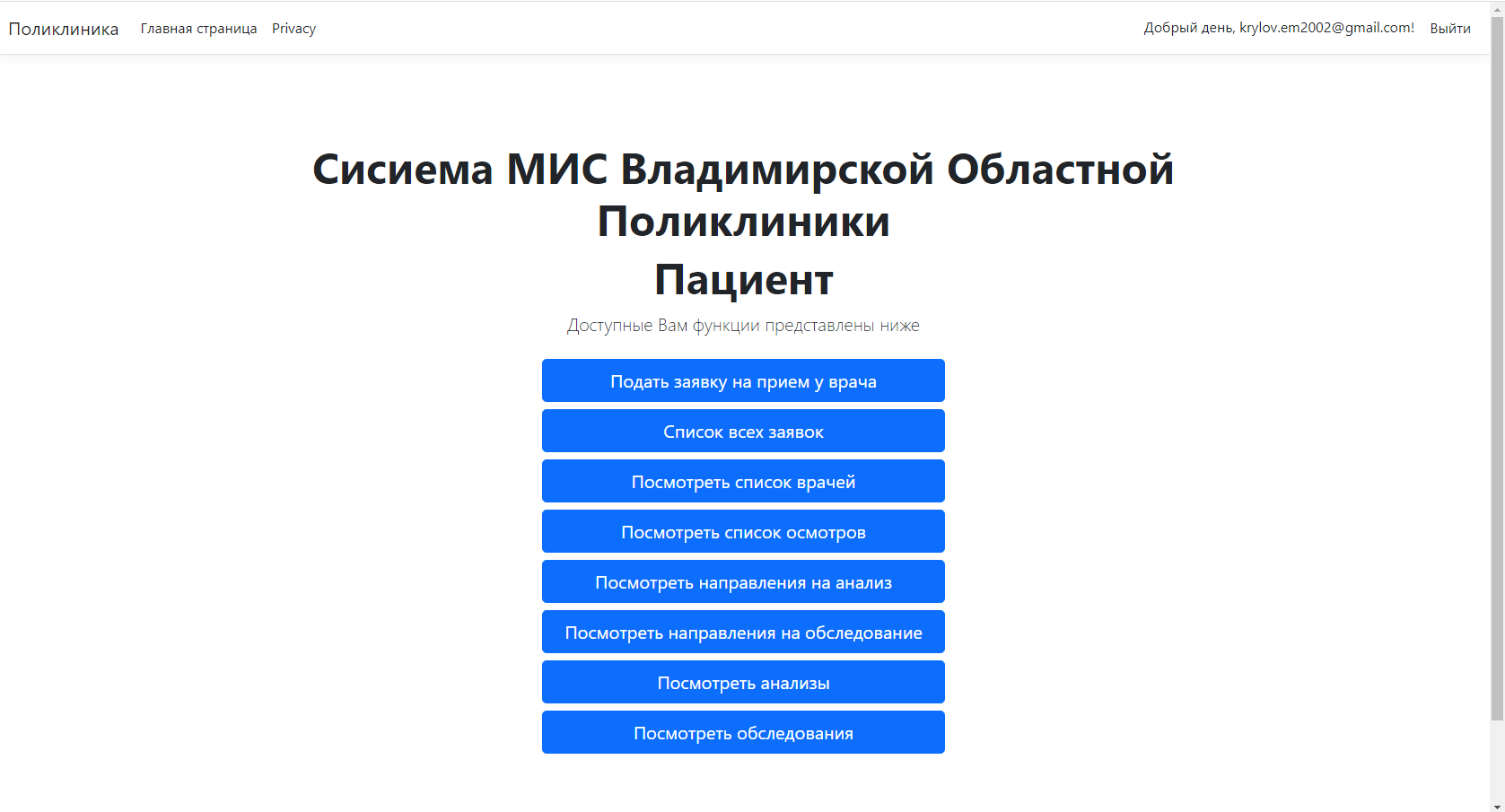


Рисунок 7 – типовая форма главной страницы навигации по сайту

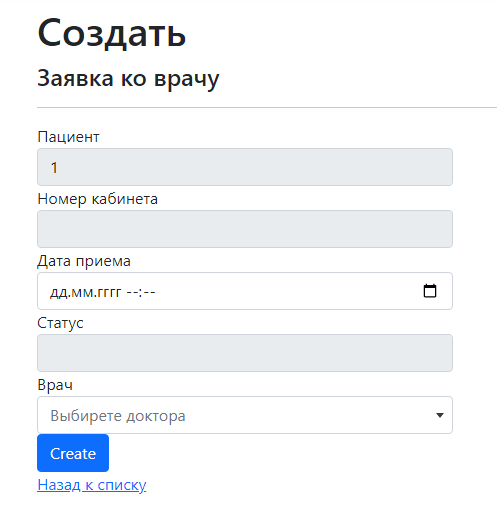


Рисунок 8 – типовая форма создания записи с поиском и автозаполняемыми полями

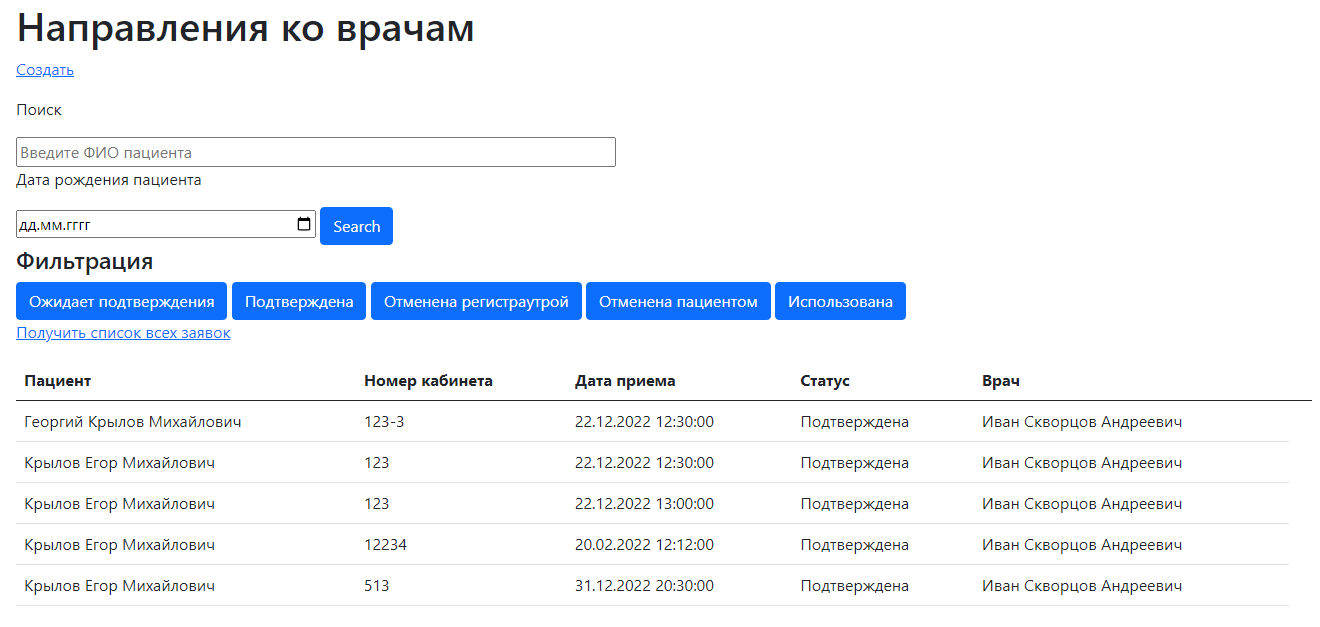


Рисунок 9 – типовая форма списка с поиском и фильтрацией

4 РЕАЛИЗАЦИЯ СИСТЕМЫ

4.1 Технологический стек

* реализация на платформе .net
* использование фреймворка asp.net core 6[2]
* использование паттерна проектирования MVC[3]
* использование entity framework
* использование linq
* использование базы данных MSSQL
* использование микросервисов на Web API

4.2 Реализация модели на сервере

Сущности базы данных сгенерированны автоматически по классам написанным на C# с использованием Entity Framework и миграций. А именно использовался подход Code First - подход, где сначала пишется код, а потом уже создается база данных.

4.3 Реализация аутентификации

Для реализации аутентификации использовался API asp.net core 6 – Identity. С помощью него реализовывался стандартный пользователь системой, а также его права доступа на базе ролей. Для поддержки разных типов пользователей помимо ролей использовались собственные таблицы со значениями, которые были связаны с таблицами, которые были сгенерированны Identity.

4.4 Модели системы

Листинг 1. Модель контекста базы данных

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using Polyclinic.Areas.Identity.Data;

using Polyclinic.Models;

namespace Polyclinic.Data;

public class PolyclinicContext : IdentityDbContext<PolyclinicUser>

{

public PolyclinicContext(DbContextOptions<PolyclinicContext> options)

: base(options)

{

}

public DbSet<Patient> Patients { get; set; }

public DbSet<Analysis> Analyses { get; set; }

public DbSet<Assistant> Assistants { get; set; }

public DbSet<Diagnosis> Diagnoses { get; set; }

public DbSet<Examination> Examinations { get; set; }

public DbSet<FunctionalDiagnosticsDoctor> FunctionalDiagnosticsDoctors { get; set; }

public DbSet<Inspection> Inspections { get; set; }

public DbSet<AnalysisReferral> AnalysisReferrals { get; set; }

public DbSet<Doctor> Doctors { get; set; }

public DbSet<DoctorAppointment> DoctorAppointments { get; set; }

public DbSet<DoctorReferral> DoctorReferrals { get; set; }

public DbSet<ExaminationReferral> ExaminationReferrals { get; set; }

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

// Customize the ASP.NET Identity model and override the defaults if needed.

// For example, you can rename the ASP.NET Identity table names and more.

// Add your customizations after calling base.OnModelCreating(builder);

builder.ApplyConfiguration(new ApplicationUserEntityConfiguration());

}

public DbSet<Polyclinic.Models.Receptionist> Receptionist { get; set; }

}

internal class ApplicationUserEntityConfiguration : IEntityTypeConfiguration<PolyclinicUser>

{

public void Configure(EntityTypeBuilder<PolyclinicUser> builder)

{

}

}

Листинг 2. Модель «Анализ»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Polyclinic.Models

{

public class Analysis

{

[Key]

public int Id { get; set; }

[ForeignKey("Patient")]

[Display(Name = "Пациент")]

public int? PatientId { get; set; }

[Display(Name = "Пациент")]

public Patient? Patient { get; set; }

[Display(Name = "Тип")]

public string Type { get; set; }

[Display(Name = "Описание")]

public string Description { get; set; }

[ForeignKey("Assistant")]

[Display(Name = "Лаборант")]

public int? AssistantId { get; set; }

[Display(Name = "Лаборант")]

public virtual Assistant? Assistant { get; set; }

}

}

Листинг 3. Модель «Направление на анализ»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Polyclinic.Models

{

public class AnalysisReferral

{

[Key]

public int Id { get; set; }

[ForeignKey("Diagnosis")]

[Display(Name = "Предварительный диагноз")]

public string? DiagnosisId { get; set; }

[Display(Name = "Предварительный диагноз")]

public Diagnosis? Diagnosis { get; set; }

[ForeignKey("Doctor")]

[Display(Name = "Врач")]

public int? DoctorId { get; set; }

[Display(Name = "Врач")]

public virtual Doctor? Doctor { get; set; }

[ForeignKey("Patient")]

[Display(Name = "Пациент")]

public int? PatientId { get; set; }

[Display(Name = "Пациент")]

public virtual Patient? Patient { get; set; }

[Display(Name = "Тип")]

public string? Type { get; set; }

[ForeignKey("Assistant")]

[Display(Name = "Лаборант")]

public int? AssistantId { get; set; }

[Display(Name = "Лаборант")]

public virtual Assistant? Assistant { get; set; }

[Display(Name = "Номер кабинета")]

public string? СabinetNum { get; set; }

[Display(Name = "Дата")]

public DateTime? DateTime { get; set; }

}

}

Листинг 4. Модель «Лаборант»

using Polyclinic.Areas.Identity.Data;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Polyclinic.Models

{

public class Assistant

{

[Key]

public int Id { get; set; }

[Display(Name = "Имя")]

public string FirstName { get; set; }

[Display(Name = "Фамилия")]

public string LastName { get; set; }

[Display(Name = "Отчество")]

public string MiddleName { get; set; }

[Column(TypeName = "date")]

[Display(Name = "Дата рождения")]

public DateTime? BirthDate { get; set; }

[ForeignKey("PolyclinicUser")]

public string? PolyclinicUserID { get; set; }

public virtual PolyclinicUser? PolyclinicUser { get; set; }

public IEnumerable<Analysis>? Analyses { get; set; }

[NotMapped]

public string ReturnDateForDisplay

{

get

{

return this.BirthDate?.ToShortDateString();

}

}

[NotMapped]

public string ReturnFIOAndBirthDate

{

get

{

return LastName + " " + FirstName + " " + MiddleName + " " + this.BirthDate?.ToShortDateString();

}

}

}

}

Листинг 5. Модель «Диагноз»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Polyclinic.Models

{

public class Diagnosis

{

[Key]

[Display(Name = "Код МКБ")]

public string Id { get; set; }

[Display(Name = "Описание")]

public string? Description { get; set; }

[NotMapped]

public string ReturnIdAndDescription

{

get

{

return Id + " " + Description;

}

}

}

}

Листинг 6. Модель «Доктор»

using Polyclinic.Areas.Identity.Data;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Polyclinic.Models

{

public class Doctor

{

[Key]

public int Id { get; set; }

[Display(Name = "Имя")]

public string FirstName { get; set; }

[Display(Name = "Фамилия")]

public string LastName { get; set; }

[Display(Name = "Отчетсво")]

public string MiddleName { get; set; }

[Column(TypeName = "date")]

[Display(Name = "Дата рождения")]

public DateTime? BirthDate { get; set; }

[ForeignKey("PolyclinicUser")]

[Display(Name = "Id пользователя")]

public string? PolyclinicUserID { get; set; }

[Display(Name = "Id пользователя")]

public virtual PolyclinicUser? PolyclinicUser { get; set; }

[Display(Name = "Специальнсть")]

public string Speciality { get; set; }

[Display(Name = "Категория")]

public string? Category { get; set; }

[Display(Name = "Степень")]

public string? Degree { get; set; }

[NotMapped]

public string ReturnDateForDisplay

{

get

{

return this.BirthDate?.ToShortDateString();

}

}

[NotMapped]

public string ReturnFIO

{

get

{

return LastName + " " + FirstName + " " + MiddleName;

}

}

[NotMapped]

public string ReturnFIOAndSpeciality

{

get

{

return LastName + " " + FirstName + " " + MiddleName + " (" + Speciality + ")";

}

}

}

}

Листинг 7. Модель «Направление»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Polyclinic.Models

{

public class DoctorAppointment

{

[Key]

public int Id { get; set; }

[ForeignKey("Patient")]

[Display(Name = "Пациент")]

public int? PatientId { get; set; }

[Display(Name = "Пациент")]

public virtual Patient? Patient { get; set; }

[Display(Name = "Номер кабинета")]

public string? CabinetId { get; set; }

[Display(Name = "Дата приема")]

public DateTime? DateTime { get; set; }

[Display(Name = "Статус")]

public string? Status { get; set; }

[ForeignKey("Doctor")]

[Display(Name = "Врач")]

public int? DoctorId { get; set; }

[Display(Name = "Врач")]

public virtual Doctor? Doctor { get; set; }

}

}

Листинг 8. Модель «Обследование»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Polyclinic.Models

{

public class Examination

{

[Key]

public int Id { get; set; }

[Display(Name = "Тип")]

public string Type { get; set; }

[Display(Name = "Описание")]

public string Description { get; set; }

[ForeignKey("FunctionalDiagnosticsDoctor")]

[Display(Name = "Врач функциональной диагностики")]

public int? FunctionalDiagnosticsDoctorId { get; set; }

[Display(Name = "Врач функциональной диагностики")]

public virtual FunctionalDiagnosticsDoctor? FunctionalDiagnosticsDoctor { get; set; }

[ForeignKey("Patient")]

[Display(Name = "Пациент")]

public int? PatientId { get; set; }

[Display(Name = "Пациент")]

public virtual Patient? Patient { get; set; }

}

}

Листинг 9. Модель «Направление на обследование»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Polyclinic.Models

{

public class ExaminationReferral

{

[Key]

public int Id { get; set; }

[ForeignKey("Diagnosis")]

[Display(Name = "Предварительный диагноз")]

public string? DiagnosisId { get; set; }

[Display(Name = "Предварительный диагноз")]

public virtual Diagnosis? Diagnosis { get; set; }

[ForeignKey("Doctor")]

[Display(Name = "Доктор")]

public int? DoctorId { get; set; }

[Display(Name = "Доктор")]

public virtual Doctor? Doctor { get; set; }

[ForeignKey("Patient")]

[Display(Name = "Пациент")]

public int? PatientId { get; set; }

[Display(Name = "Пациент")]

public virtual Patient? Patient { get; set; }

[Display(Name = "Тип")]

public string? Type { get; set; }

[ForeignKey("FunctionalDiagnosticsDoctor")]

[Display(Name = "Врач функциональной диагностики")]

public int? FunctionalDiagnosticsDoctorId { get; set; }

[Display(Name = "Врач функциональной диагностики")]

public virtual FunctionalDiagnosticsDoctor? FunctionalDiagnosticsDoctor { get; set; }

[Display(Name = "Номер кабинета")]

public string? СabinetNum { get; set; }

[Display(Name = "Дата")]

public DateTime? DateTime { get; set; }

}

}

Листинг 10. Модель «Врач функциональной диагностики»

using Polyclinic.Areas.Identity.Data;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Polyclinic.Models

{

public class FunctionalDiagnosticsDoctor

{

[Key]

public int Id { get; set; }

[Display(Name = "Имя")]

public string FirstName { get; set; }

[Display(Name = "Фамилия")]

public string LastName { get; set; }

[Display(Name = "Отчество")]

public string? MiddleName { get; set; }

[Column(TypeName = "date")]

[Display(Name = "Дата рождения")]

public DateTime? BirthDate { get; set; }

[ForeignKey("PolyclinicUser")]

[Display(Name = "Id пользователя")]

public string? PolyclinicUserID { get; set; }

[Display(Name = "Id пользователя")]

public virtual PolyclinicUser? PolyclinicUser { get; set; }

public IEnumerable<Examination>? Examinations { get; set; }

[NotMapped]

public string ReturnDateForDisplay

{

get

{

return this.BirthDate?.ToShortDateString();

}

}

[NotMapped]

public string ReturnFIOAndBirthDate

{

get

{

return LastName + " " + FirstName + " " + MiddleName + " " + this.BirthDate?.ToShortDateString();

}

}

}

}

Листинг 11. Модель «Осмотр»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Polyclinic.Models

{

public class Inspection

{

[Key]

public int Id { get; set; }

[ForeignKey("Patient")]

[Display(Name = "Пациент")]

public int? PatientID { get; set; }

[Display(Name = "Пациент")]

public virtual Patient? Patient { get; set; }

[Display(Name = "Жалобы")]

public string? Complaint { get; set; }

[Display(Name = "Рецепт")]

public string? Recipe { get; set; }

[ForeignKey("Diagnosis")]

[Display(Name = "Диагноз")]

public string? DiagnosisId { get; set; }

[Display(Name = "Диагноз")]

public virtual Diagnosis? Diagnosis { get; set; }

[Display(Name = "Дата проведения")]

public DateTime? Date { get; set; }

[Display(Name = "Тип осмотра")]

public string Type { get; set; }

[ForeignKey("Doctor")]

[Display(Name = "Доктор")]

public int DoctorId { get; set; }

[Display(Name = "Доктор")]

public virtual Doctor? Doctor { get; set; }

}

}

Листинг 12. Модель «Пациент»

using Polyclinic.Areas.Identity.Data;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Polyclinic.Models

{

public class Patient

{

[Key]

public int Id { get; set; }

[Display(Name = "Имя")]

public string FirstName { get; set; }

[Display(Name = "Фамилия")]

public string LastName { get; set; }

[Display(Name = "Отчество")]

public string? MiddleName { get; set; }

[Column(TypeName = "date")]

[Display(Name = "Дата рождения")]

public DateTime? BirthDate { get; set; }

[ForeignKey("PolyclinicUser")]

[Display(Name = "Id пользователя")]

public string? PolyclinicUserID { get; set; }

[Display(Name = "Id пользователя")]

public virtual PolyclinicUser? PolyclinicUser { get; set; }

[Display(Name = "Номер полиса")]

public int PolisID { get; set; }

[Display(Name = "Компания полиса")]

public string PoilsCompany { get; set; }

[Column(TypeName = "date")]

[Display(Name = "Дата окончания полиса")]

public DateTime? PolisEndDate { get; set; }

[Display(Name = "Номер снилса")]

public int SnilsNumber { get; set; }

[Display(Name = "Место работы")]

public string? WorkPlace { get; set; }

public IEnumerable<Analysis>? Analyses { get; set; }

public IEnumerable<Examination>? Examinations { get; set; }

public IEnumerable<Inspection>? Inspections { get; set; }

[NotMapped]

public string? ReturnBirthDateForDisplay

{

get

{

return this.BirthDate?.ToShortDateString();

}

}

[NotMapped]

public string? ReturnPolisEndDateForDisplay

{

get

{

return this.PolisEndDate?.ToShortDateString();

}

}

[NotMapped]

public string? ReturnFIO

{

get

{

return LastName + " " + FirstName + " " + MiddleName;

}

}

[NotMapped]

public string? ReturnFIOAndBirthDate

{

get

{

return LastName + " " + FirstName + " " + MiddleName + " " + this.BirthDate?.ToShortDateString();

}

}

}

}

Листинг 13. Модель «Мед регистратор»

using Microsoft.EntityFrameworkCore.Metadata.Internal;

using Polyclinic.Areas.Identity.Data;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Polyclinic.Models

{

public class Receptionist

{

[Key]

public int Id { get; set; }

[Display(Name = "Имя")]

public string FirstName { get; set; }

[Display(Name = "Фамилия")]

public string LastName { get; set; }

[Display(Name = "Отчество")]

public string? MiddleName { get; set; }

[Column(TypeName = "date")]

[Display(Name = "Дата рождения")]

public DateTime? BirthDate { get; set; }

[ForeignKey("PolyclinicUser")]

[Display(Name = "Id пользователя")]

public string? PolyclinicUserID { get; set; }

[Display(Name = "Id пользователя")]

public virtual PolyclinicUser? PolyclinicUser { get; set; }

[NotMapped]

public string ReturnDateForDisplay

{

get

{

return this.BirthDate?.ToShortDateString();

}

}

[NotMapped]

public string ReturnFIO

{

get

{

return LastName + " " + FirstName + " " + MiddleName;

}

}

[NotMapped]

public string ReturnFIOAndBirthDate

{

get

{

return LastName + " " + FirstName + " " + MiddleName + " " + this.BirthDate?.ToShortDateString();

}

}

}

}

4.5 Реализация контроллеров

Листинг типового контроллера представлен в приложении Б

4.6 Реализация представлений

Листинг типового представления со списком представлен в приложении В.

Листинг типового представления создания представлен в приложении Г.

4.7 Реализация микросервиса

Листинг микросервиса представлен в приложении Д.

5 ЗАКЛЮЧЕНИЕ

В результате выполненной работы был произведен анализ предметной области «Поликлиника», определены основные процессы, происходящие в системе, спроектирована система хранения данных, описана реализация бизнес-логики.

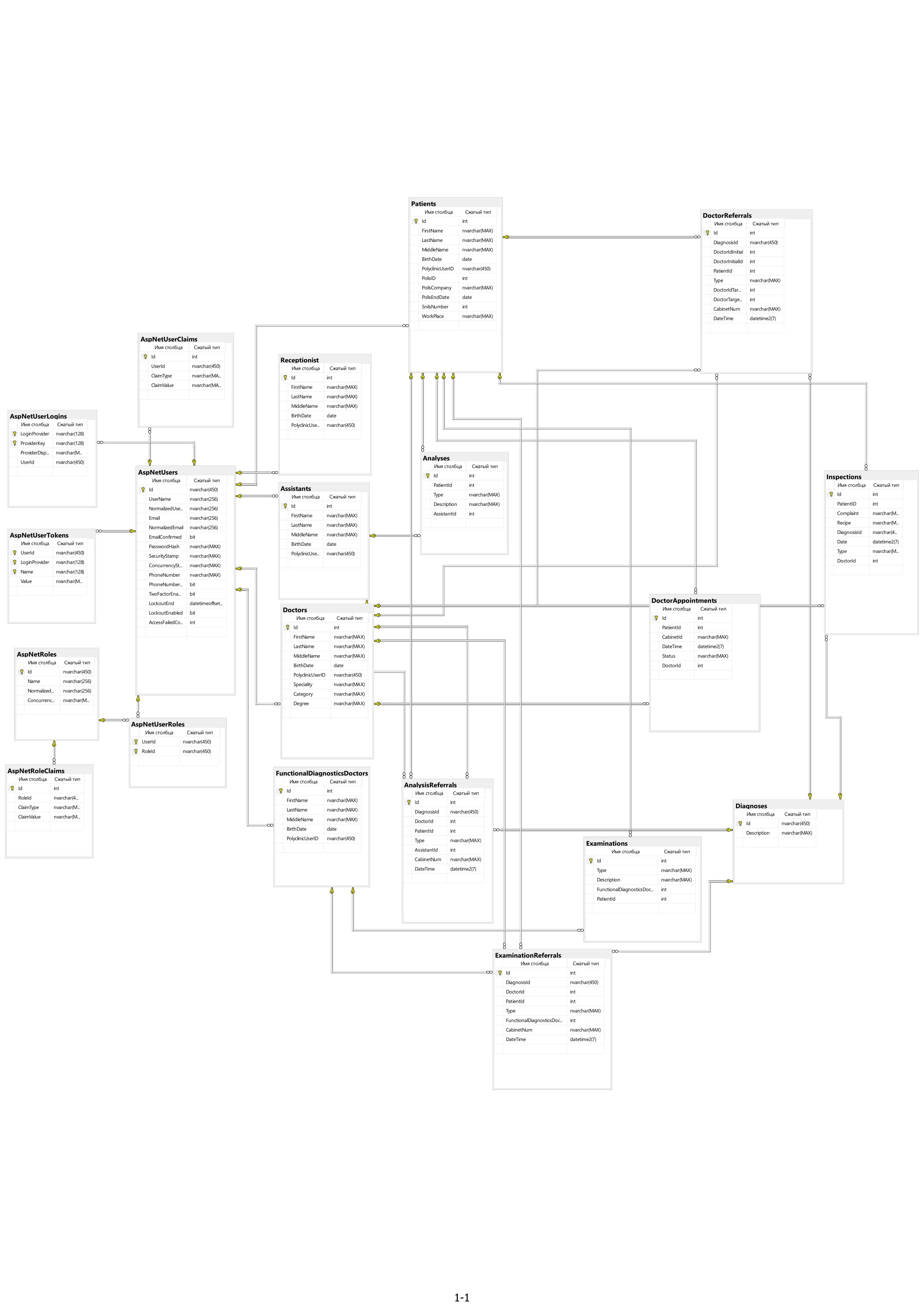
Были разработаны страницы регистрации, авторизации, меню, подтверждения данных, присвоения ролей пользователям, списка заявлений, осмотров, пациентов, врачей, лаборантов, врачей функциональной диагностики, мед регистраторов, анализов, обследований.

Был спроектирован микросервис позволяющий автоматически отправлять уведомления на почту.

Текущее состояние системы позволяет ее использование в реальной поликлинике.

6 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Введение в Identity в ASP.NET Core // https://learn.microsoft.com/ URL: https://learn.microsoft.com/ru-ru/aspnet/core/security/authentication/identity?view=aspnetcore-6.0&tabs=visual-studio (дата обращения: 9.12.22).
2. Новые возможности в ASP.NET Core 6.0 // https://learn.microsoft.com/ URL: https://learn.microsoft.com/ru-ru/aspnet/core/release-notes/aspnetcore-6.0?view=aspnetcore-6.0 (дата обращения: 10.12.22).
3. https://www.tutorialspoint.com/design\_pattern/mvc\_pattern.htm // tutorialspoint.com URL: Design Patterns - MVC Pattern (дата обращения: 10.12.22).

ПРИЛОЖЕНИЕ А

ПРИЛОЖЕНИЕ Б

Контроллер для сущности Врач

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Polyclinic.Areas.Identity.Data;

using Polyclinic.Data;

using Polyclinic.Models;

namespace Polyclinic.Controllers

{

public class DoctorsController : Controller

{

private readonly PolyclinicContext \_context;

private readonly SignInManager<PolyclinicUser> \_signInManager;

public DoctorsController(SignInManager<PolyclinicUser> signInManager, PolyclinicContext context)

{

\_context = context;

\_signInManager = signInManager;

}

// GET: Doctors

[Authorize(Roles = "Admin,Patient")]

public async Task<IActionResult> Index()

{

var polyclinicContext = \_context.Doctors.Include(d => d.PolyclinicUser);

return View(await polyclinicContext.ToListAsync());

}

[HttpGet]

[Authorize(Roles = "Admin,Patient,Doctor,Receptionist")]

public async Task<IActionResult> Index(string? doctorLastName, string? doctorSpeciality)

{

ViewData["DoctorLastName"] = doctorLastName;

ViewData["DoctorSpeciality"] = doctorSpeciality;

var doctorsQuery = from x in \_context.Doctors.Include(d => d.PolyclinicUser) select x;

if (!String.IsNullOrEmpty(doctorLastName))

{

doctorsQuery = doctorsQuery.Where(x => x.LastName.Contains(doctorLastName));

}

if (!String.IsNullOrEmpty(doctorSpeciality))

{

doctorsQuery = doctorsQuery.Where(x => x.Speciality.Contains(doctorSpeciality));

}

return View(await doctorsQuery.AsNoTracking().ToListAsync());

}

// GET: Doctors/Details/5

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.Doctors == null)

{

return NotFound();

}

var doctor = await \_context.Doctors

.Include(d => d.PolyclinicUser)

.FirstOrDefaultAsync(m => m.Id == id);

if (doctor == null)

{

return NotFound();

}

return View(doctor);

}

// GET: Doctors/Create

[Authorize(Roles = "CanRegisterAsDoctor")]

public IActionResult Create()

{

ViewData["PolyclinicUserID"] = new SelectList(\_context.Users, "Id", "Id");

return View();

}

// POST: Doctors/Create

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "CanRegisterAsDoctor")]

public async Task<IActionResult> Create([Bind("Id,FirstName,LastName,MiddleName,BirthDate,PolyclinicUserID,Speciality,Category,Degree")] Doctor doctor)

{

if (ModelState.IsValid)

{

\_context.Add(doctor);

var userRoleBefore = new IdentityUserRole<string> { RoleId = "2", UserId = doctor.PolyclinicUserID };

var userRoleAfter = new IdentityUserRole<string> { RoleId = "1", UserId = doctor.PolyclinicUserID };

\_context.UserRoles.Remove(userRoleBefore);

\_context.UserRoles.Add(userRoleAfter);

await \_context.SaveChangesAsync();

PolyclinicUser user = \_context.Users.Find(doctor.PolyclinicUserID);

await \_signInManager.RefreshSignInAsync(user);

return RedirectToAction(nameof(Index));

}

ViewData["PolyclinicUserID"] = new SelectList(\_context.Users, "Id", "Id", doctor.PolyclinicUserID);

return View(doctor);

}

// GET: Doctors/Edit/5

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.Doctors == null)

{

return NotFound();

}

var doctor = await \_context.Doctors.FindAsync(id);

if (doctor == null)

{

return NotFound();

}

ViewData["PolyclinicUserID"] = new SelectList(\_context.Users, "Id", "Id", doctor.PolyclinicUserID);

return View(doctor);

}

// POST: Doctors/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Edit(int id, [Bind("Id,FirstName,LastName,MiddleName,BirthDate,PolyclinicUserID,Speciality,Category,Degree")] Doctor doctor)

{

if (id != doctor.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(doctor);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!DoctorExists(doctor.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["PolyclinicUserID"] = new SelectList(\_context.Users, "Id", "Id", doctor.PolyclinicUserID);

return View(doctor);

}

// GET: Doctors/Delete/5

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.Doctors == null)

{

return NotFound();

}

var doctor = await \_context.Doctors

.Include(d => d.PolyclinicUser)

.FirstOrDefaultAsync(m => m.Id == id);

if (doctor == null)

{

return NotFound();

}

return View(doctor);

}

// POST: Doctors/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.Doctors == null)

{

return Problem("Entity set 'PolyclinicContext.Doctors' is null.");

}

var doctor = await \_context.Doctors.FindAsync(id);

if (doctor != null)

{

\_context.Doctors.Remove(doctor);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool DoctorExists(int id)

{

return \_context.Doctors.Any(e => e.Id == id);

}

}

}

ПРИЛОЖЕНИЕ В

@model IEnumerable<Polyclinic.Models.Doctor>

@{

ViewData["Title"] = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1>Список врачей</h1>

@if (this.User.IsInRole("Admin"))

{

<p>

<**a** **asp-action**="Create">Создать</**a**>

</p>

}

<**form** method="get" **asp-action**="Index">

<p>

<h2>Поиск</h2>

<p style="color:red;">@Html.ValidationMessage("error\_msg")</p>

<p>Фамилия врача</p>

<input type="search" placeholder="Введите фамилию врача (Необязательно)" value="@ViewData["DoctorLastName"]" name="DoctorLastName" style="width:600px;" />

<p>Специальность врача</p>

<input type="search" placeholder="Введите специальность врача (Необязательно)" value="@ViewData["DoctorSpeciality"]" name="DoctorSpeciality" style="width:600px;" />

<input type="submit" value="Search" class="btn btn-primary" />

<p><**a** **asp-action**="Index">Получить список всех врачей</**a**></p>

</p>

</**form**>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.FirstName)

</th>

<th>

@Html.DisplayNameFor(model => model.LastName)

</th>

<th>

@Html.DisplayNameFor(model => model.MiddleName)

</th>

<th>

@Html.DisplayNameFor(model => model.BirthDate)

</th>

@if (this.User.IsInRole("Admin"))

{

<th>

@Html.DisplayNameFor(model => model.PolyclinicUser)

</th>

}

<th>

@Html.DisplayNameFor(model => model.Speciality)

</th>

<th>

@Html.DisplayNameFor(model => model.Category)

</th>

<th>

@Html.DisplayNameFor(model => model.Degree)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.FirstName)

</td>

<td>

@Html.DisplayFor(modelItem => item.LastName)

</td>

<td>

@Html.DisplayFor(modelItem => item.MiddleName)

</td>

<td>

@Html.DisplayFor(modelItem => item.ReturnDateForDisplay)

</td>

@if (this.User.IsInRole("Admin"))

{

<td>

@Html.DisplayFor(modelItem => item.PolyclinicUser.Id)

</td>

}

<td>

@Html.DisplayFor(modelItem => item.Speciality)

</td>

<td>

@Html.DisplayFor(modelItem => item.Category)

</td>

<td>

@Html.DisplayFor(modelItem => item.Degree)

</td>

@if (this.User.IsInRole("Admin"))

{

<td>

<**a** **asp-action**="Edit" **asp-route-id**="@item.Id">Edit</**a**> |

<**a** **asp-action**="Details" **asp-route-id**="@item.Id">Details</**a**> |

<**a** **asp-action**="Delete" **asp-route-id**="@item.Id">Delete</**a**>

</td>

}

</tr>

}

</tbody>

</table>

ПРИЛОЖЕНИЕ Г

@using Polyclinic.Areas.Identity.Data;

@using Polyclinic.Models;

@model Polyclinic.Models.Doctor

@{

ViewData["Title"] = "Create";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

@inject Microsoft.AspNetCore.Identity.UserManager<PolyclinicUser> \_userManager

@{

string userId = \_userManager.GetUserId(User);

}

<h1>Регистрация как доктор</h1>

<h4>Доктор</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** **asp-action**="Create">

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<div class="form-group">

<**label** **asp-for**="PolyclinicUserID" class="control-label"></**label**>

<**input** **asp-for**="PolyclinicUserID" class="form-control" **value**=@userId readonly />

<**span** **asp-validation-for**="PolyclinicUserID" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="LastName" class="control-label"></**label**>

<**input** **asp-for**="LastName" class="form-control" />

<**span** **asp-validation-for**="LastName" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="FirstName" class="control-label"></**label**>

<**input** **asp-for**="FirstName" class="form-control" />

<**span** **asp-validation-for**="FirstName" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="MiddleName" class="control-label"></**label**>

<**input** **asp-for**="MiddleName" class="form-control" />

<**span** **asp-validation-for**="MiddleName" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="BirthDate" class="control-label"></**label**>

<**input** **type**="date" **asp-for**="BirthDate" class="form-control" />

<**span** **asp-validation-for**="BirthDate" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Speciality" class="control-label"></**label**>

<**input** **asp-for**="Speciality" class="form-control" />

<**span** **asp-validation-for**="Speciality" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Category" class="control-label"></**label**>

<**input** **asp-for**="Category" class="form-control" />

<**span** **asp-validation-for**="Category" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Degree" class="control-label"></**label**>

<**input** **asp-for**="Degree" class="form-control" />

<**span** **asp-validation-for**="Degree" class="text-danger"></**span**>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</**form**>

</div>

</div>

<!--

<div>

<a asp-action="Index">Back to List</a>

</div>

-->

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

**ПРИЛОЖЕНИЕ Д**

using EmailNotifier.PrivateSettings;

using Microsoft.AspNetCore.Mvc;

namespace EmailNotifier.Controllers

{

[Route("Approved")]

[ApiController]

public class AppointmentApproved : ControllerBase

{

[HttpGet]

[Route("SendEmailNotification")]

public string Send([FromQuery] string email, [FromQuery] string subject, [FromQuery] string message)

{

try

{

var smtpClient = new System.Net.Mail.SmtpClient("smtp.mail.ru", 587);

smtpClient.Credentials = new System.Net.NetworkCredential(MailSettings.Email, MailSettings.Password);

smtpClient.EnableSsl = true;

smtpClient.Send(new System.Net.Mail.MailMessage(MailSettings.Email, email, subject, message));

smtpClient.Dispose();

return "Оповещение отправлено";

}

catch (Exception ex)

{

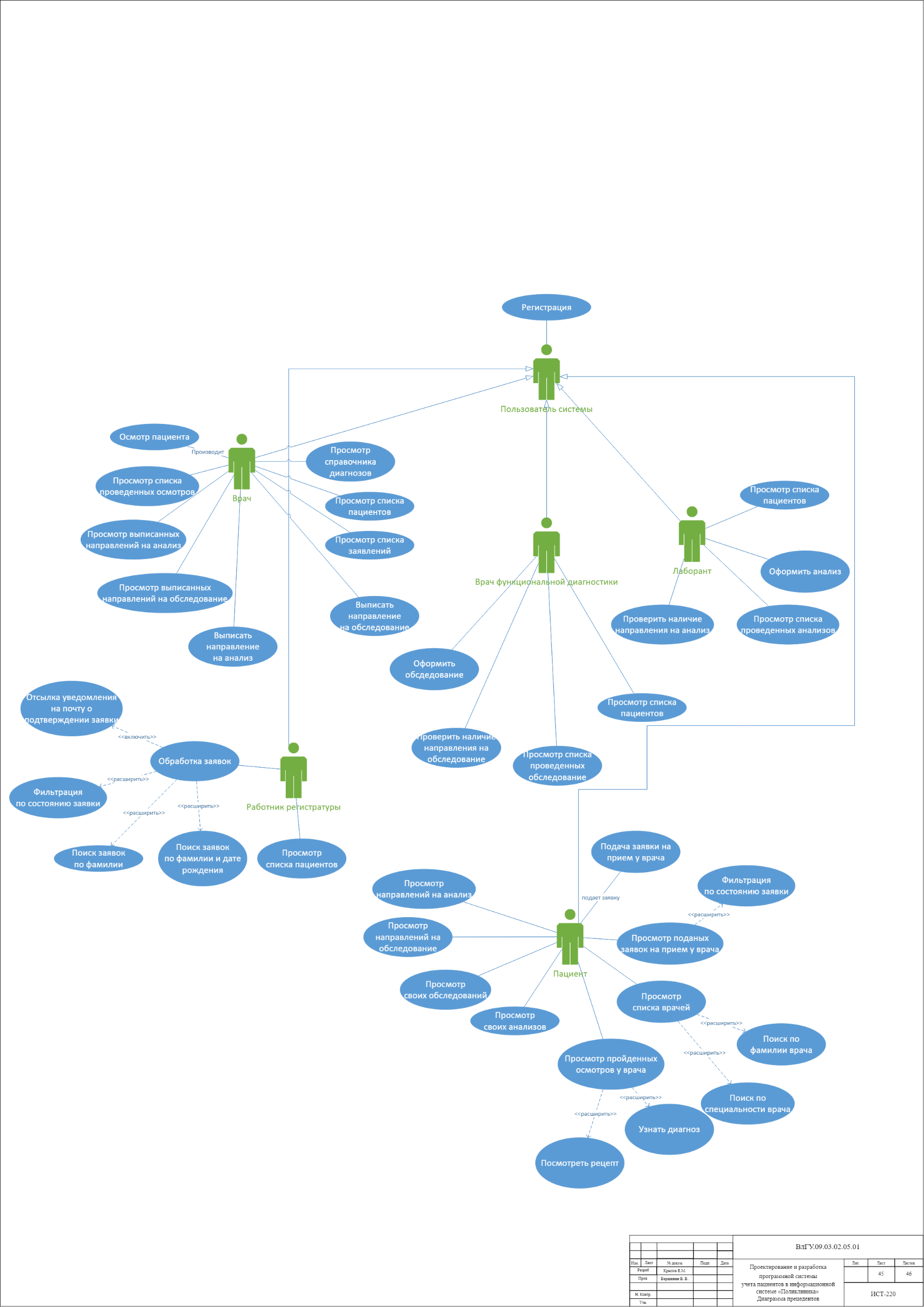
return ex.Message;

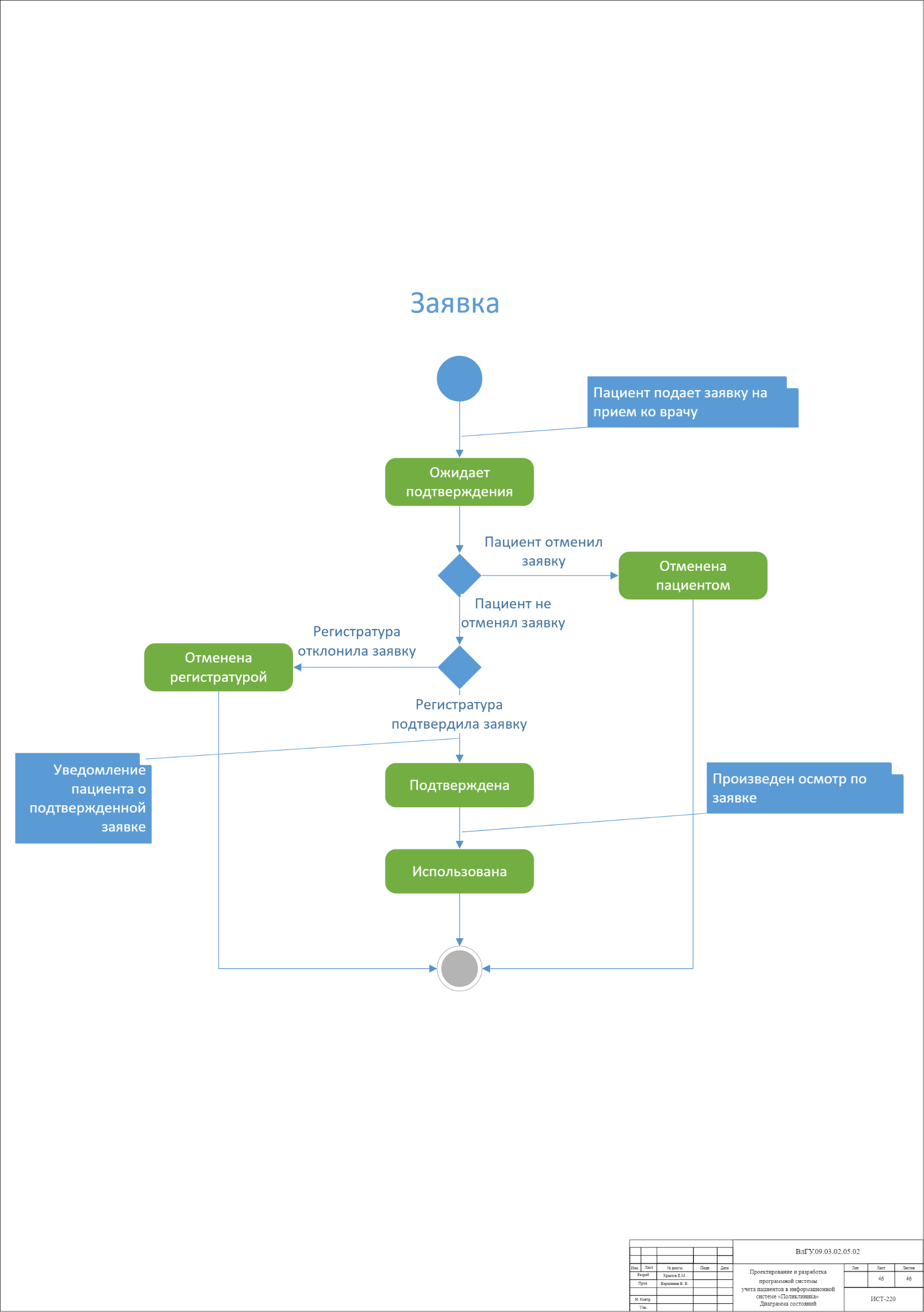
}

}

}

}

**ПРИЛОЖЕНИЕ Е**

****

**ПРИЛОЖЕНИЕ Ж**