*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   
информационной системы «Поликлиника»*

*Выполнил: студент гр. ПРИ-120*

*Мелконян Т.Г.*

*Принял: доц. кафедры ИСПИ*

*Верешинин В.В.*

*Владимир, 2022*

**Аннотация**

Тема курсового проекта: разработка программной системы для информационной системы «Поликлиника».

Выполнили: Мелконян Т.Г., Дюжик С.В.

Руководитель: Вершинин В.В.

Данный курсовой проект состоит из нескольких этапов: описание предметной области, проектирование, реализация и тестирование системы.

Для реализации программной системы использовалась платформа .NET: для создания веб-приложения ASP.NET, Entity Framework Core для работы с данными, база данных SQL Lite.

В первой части работы описывается предметная область, включающая декомпозицию и детализацию информационной системы. Определены цели проекта. В следующем этапе идет проектирование системы, разработаны диаграммы прецедентов, классов, состояний и последовательностей. Затем представлена реализация системы и тестирование системы.

Данная программная система предназначена для автоматизации работы некоторых бизнес процессов поликлиники. Такие как запись на приём к врачу, приём врачом пациента, просмотр списка записей к врачу, управление списком услуг, врачей от имени администратора. Для многопользовательского приложения была создана система авторизации и аутентификации пользователей.

Курсовой проект представлен на 41 страницах, содержит 20 рисунков, 7 таблиц, 4 приложений, 4 использованных источников.

**Annotation**

The topic of the course project: development of a software system for the information system "Polyclinic".

Performed by: Melkonyan T.G., Dyuzhik S.V.

Supervisor: Vershinin V.V.

This course project consists of several stages: description of the subject area, design, implementation and testing of the system.

A platform was used to implement the software system .NET: to create a web application ASP.NET , Entity Framework Core for working with data, SQL Lite database.

The first part of the paper describes the subject area, including the decomposition and detailing of the information system. The objectives of the project are defined. In the next stage, the system is being designed, diagrams of use cases, classes, states and sequences have been developed. Then the implementation of the system and testing of the system are presented.

This software system is designed to automate the operation of some business processes of the polyclinic. Such as making an appointment with a doctor, receiving a patient's doctor, viewing a list of doctor's appointments, managing a list of services, doctors on behalf of the administrator. A user authorization and authentication system was created for the multi-user application.

The course project is presented on 41 pages, contains 20 figures, 7 tables, 4 appendices, 4 used sources.

**Содержание**

[1. ВВЕДЕНИЕ 4](#_Toc122970599)

[2. Описание предметной области 5](#_Toc122970600)

[2.1 Описание предметной области 5](#_Toc122970601)

[2.2 Пользователи системы 5](#_Toc122970602)

[2.3 Словарь предметной области 6](#_Toc122970603)

[3. проектирование системы 8](#_Toc122970604)

[3.1 Функциональные требования 8](#_Toc122970605)

[3.2 Нефункциональные требования 8](#_Toc122970606)

[3.3 Сценарий взаимодействия пользователей с системой 8](#_Toc122970607)

[3.4 Описание прецедентов 9](#_Toc122970608)

[3.5 Диаграмма прецедентов 12](#_Toc122970609)

[3.6 Диаграмма классов 13](#_Toc122970610)

[3.7 Обоснование связей между классами 13](#_Toc122970611)

[3.8 Диаграмма состояний 14](#_Toc122970612)

[3.9 Диаграмма последовательностей 15](#_Toc122970613)

[3.10 Физическая модель базы данных 16](#_Toc122970614)

[4. Реализация компонентов системы 17](#_Toc122970615)

[4.1 Реализация авторизации и аутентификации пользователей приложения 17](#_Toc122970616)

[4.2 Реализация личного кабинета администратора 17](#_Toc122970617)

[4.3 Реализация личного кабинета врача 17](#_Toc122970618)

[4.4 Реализация личного кабинета пациента 17](#_Toc122970619)

[5. Тестирование системы 19](#_Toc122970620)

[6. заключение 26](#_Toc122970621)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27](#_Toc122970622)

[ПРИЛОЖЕНИЕ А 28](#_Toc122970623)

[ПРИЛОЖЕНИЕ Б 29](#_Toc122970624)

[ПРИЛОЖЕНИЕ В 30](#_Toc122970625)

[ПРИЛОЖЕНИЕ Г 31](#_Toc122970626)

# ВВЕДЕНИЕ

В ходе выполнения курсового проекта требовалось разработать программную систему информационной системы «Поликлиника» для обработки данных о пациентах, врачах.

Разработка курсового проекта ведется в команде, моей частью разработки является создание страницы для врача и администратора.

Цель работы: разработать программную информационную систему «Поликлиника».

Для достижения поставленной цели необходимо решить следующие задачи:

* Хранение данных о пациентах и врачах
* Создание приложения удобного для работы врачей (выписка рецептов, заключений)
* Создание приложения удобного для записи к врачу, просмотра услуг, предоставляемых поликлиникой

# Описание предметной области

## Описание предметной области

Поликлиника – это специализированное лечебно-профилактическое учреждение для оказания амбулаторных медицинских услуг.

Данное приложение подразумевает создание сервиса под многопрофильную клинику, которая оказывает медицинские услуги различных направлений (подразумеваются специалисты определенных областей).

Многопрофильность клиники – это возможность оказывать медицинские услуги по различным направлениям, например, хирургия, дерматологическое лечение, неврология и т.д.

Создаваемое web-приложение будет реализовано как многопользовательское приложение. Будут разработаны профили для пациентов, врачей. Каждый раздел подразумевает наличие собственного пользовательского интерфейса и реализацию определенного функционала.

Профиль пациента создается для взаимодействия клиентов поликлиники с сервисом. Он подразумевает разработку максимально удобного интерфейса для пользователей. Главными функциями данного раздела будут являться: предоставление информации о лечебном персонале клиники (вкладка Врачи) и дальнейшая запись на прием.

Профиль врача разрабатывается для всех врачей клиники. Он основан на таких функциях, как: просмотр всех записей (рабочее расписание приемов), возможность записи определенного пациента на повторный прием, добавление рецепта для конкретного пациента и внесение записей в амбулаторную карту и т.д.

## Пользователи системы

*Администратор* – пользователь, обладающий возможностью просматривать, изменять данные о врачах. Также администратор может добавить врача, просмотреть расписание приемов и амбулаторные карты больных. Администратор контролирует работу поликлиники.

*Пациент* – пользователь, обладающий возможностью записываться на прием к врачу и просматривать свою амбулаторную карту.

*Врач* – пользователь, обладающий возможностью добавлять записи в амбулаторные карты пациентов при приеме, назначать дальнейшее лечение пациента. Также у врача есть возможность просматривать расписание приемов на день или месяц.

*Незарегистрированный пользователь* – имеет возможность просматривать информацию о поликлинике, о врачах и предоставляемых поликлиникой услуг.

* 1. Словарь предметной области

*Поликлиника* – многопрофильное или специализированное лечебно-профилактическое учреждение для оказания амбулаторной медицинской помощи больным на приёме и на дому.

*Пациент* – человек, получающий мед. помощь, подвергающийся мед. наблюдению и лечению по поводу какого-либо заболевания.

*Врач* – специалист с высшим медицинским образованием, использующий свои навыки, знания, опыт в профилактике и лечении заболеваний. Поддержании нормальной жизнедеятельности организма человека.

*Амбулаторная карта* – это медицинский документ, в котором лечащими врачами ведется запись истории болезни пациента и назначаемого ему лечения.

*Первичный прием* – это первичное обращение к специалисту по поводу острого заболевания или обострения хронического заболевания.

*Повторный прием* – это повторное обращение к одному и тому же специалисту по одному и тому же случаю заболевания, но не дольше, чем в течение трех месяцев с момента первичного обращения.

*Запись* – это внесение необходимых данных в БД системы, свидетельствующих о том, что пациенту было назначено определенное время посещения конкретного специалиста.

*Анализы* – это метод обследования пациента на предмет выявления в его состоянии здоровья любых отклонений от нормы.

*Расписание* – это график, содержащий сведения о времени, месте и последовательности приемов врачей в клинике.

*Выписка* – это мед. документ по утвержденной государственной форме 027/У. Он отражает состояние здоровья человека: дату визита к врачу, его жалобы, результаты анализов, поставленный диагноз.

*Диагноз* – это краткое врачебное заключение о патологическом состоянии здоровья обследуемого, об имеющихся у него заболеваниях (травмах).

*Направление* – это запись, подтверждающая необходимость посещения пациентом определенного специалиста.

*Результаты анализов* – это результат, показывающий наличие или отсутствие какого-либо вещества в организме.

*Медицинская услуга* – это медицинское вмешательство или комплекс мед. вмешательств, направленных на профилактику, диагностику и лечение заболеваний, мед. реабилитацию и имеющих самостоятельное законченное значение.

*Справка* – документ, который содержит описание и подтверждение фактов заболевания и лечения.

*Учетная запись* – это хранимая в системе совокупность данных о пользователе, необходимая для его опознавания (аутентификации) и предоставления доступа к его личным данным (Личному кабинету)

*Больничный* – документ, подтверждающий, что пациент болеете, не может работать.

# проектирование системы

## Функциональные требования

* Регистрация;
* Авторизация и аутентификация;
* Выход из системы;
* Просмотр списка услуг, предоставляемых поликлиникой;
* Добавление отзывов о врачах;
* Изменить отзыв;
* Удалить отзыв;
* Просмотр расписания приемов;
* Просмотр состояния заявки на прием;
* Выписка заключения;
* Добавление записей в амбулаторную карту;
* Просмотр амбулаторной карты;
* Запись на прием;
* Отмена записи на прием;
* Поддержка справочника врачей.

## Нефункциональные требования

* Реализация на платформе .NET;
* Использование технологии ASP.NET;
* Использование MVC;
* Использование Entity Framework Core;
* Использование базы данных MSSQL.

## Сценарий взаимодействия пользователей с системой

* Незарегистрированный пользователь
  + - Зарегистрироваться
    - Авторизоваться и аутентифицироваться
    - Посмотреть список услуг
    - Посмотреть список врачей
    - Посмотреть информацию о поликлинике
* Пациент
  + - Записаться на прием
    - Отменить запись на прием
    - Просмотреть амбулаторную карту
    - Выйти из системы
    - Авторизоваться и аутентифицироваться
    - Посмотреть список услуг
    - Посмотреть список врачей
    - Посмотреть информацию о поликлинике
    - Зарегистрироваться
* Врач
  + - Авторизоваться и аутентифицироваться
    - Выйти из системы
    - Выписать рецепт
    - Назначить повторный прием
    - Добавить запись в амбулаторную карту
* Администратор
  + - Авторизоваться и аутентифицироваться
    - Выйти из системы
    - Добавить врача
    - Удалить врача
    - Изменять данные о врачах
    - Просмотр списка врачей
    - Просмотр списка пациентов
    - Просмотр расписания врачей
    - Посмотреть список услуг
    - Добавить услуги

## Описание прецедентов

Прецедент «Посмотреть данные о врачах»

Прецедент «Посмотреть данные о врачах». Таблица 1

|  |  |
| --- | --- |
| Название | « Посмотреть данные о врачах » |
| Предусловие | - |
| Действующее лицо | Администратор |
| Основной поток | Администратору доступна таблица со списком врачей, возможностью поиска, изменения или внесения новых данных. |

Прецедент «Посмотреть данные о пациентах»

Прецедент «Посмотреть данные о пациентах». Таблица 2

|  |  |
| --- | --- |
| Название | « Посмотреть о пациентах » |
| Предусловие | - |
| Действующее лицо | Администратор |
| Основной поток | Администратору доступна таблица со списком пациентов, возможностью поиска, изменения или внесения новых данных. |

Прецедент «Посмотреть расписание приемов»

Прецедент «Посмотреть расписание приемов». Таблица 3

|  |  |
| --- | --- |
| Название | « Посмотреть расписание приемов » |
| Предусловие | - |
| Действующее лицо | Администратор, врач |
| Основной поток | Администратору доступно расписание всех врачей.  Врачу доступно расписание его приемов на день/неделю |

Прецедент «Приём пациента»

Прецедент «Приём пациента». Таблица 4

|  |  |
| --- | --- |
| Название | « Приём пациента » |
| Предусловие | Пользователь является врачом, пациент записался к врачу |
| Действующее лицо | Врач |
| Основной поток | Врач осматривает пациента и на основе осмотра выносит заключение, которое записывается в амбулаторную карту пациента. Также врач может назначить повторный прием. |
| Постусловие | Внесение данных в амбулаторную карту пациента |

Прецедент «Запись на приём»

Прецедент «Запись на приём». Таблица 5

|  |  |
| --- | --- |
| Название | « Запись на приём » |
| Предусловие | Пользователь является пациентом |
| Действующее лицо | Пациент |
| Основной поток | Пациент выбирает врача, к которому хочет записаться, выбирает доступные дату и время. Затем записывается к врачу. |
| Постусловие | Внесение информации о записи на приём |

Прецедент «Посмотреть амбулаторную карту»

Прецедент «Посмотреть амбулаторную карту». Таблица 6

|  |  |
| --- | --- |
| Название | « Посмотреть амбулаторную карту » |
| Предусловие | Пользователь является пациентом поликлиники или врачом, к которому записался пациент |
| Действующее лицо | Пациент |
| Основной поток | Пациент может посмотреть все записи о приемах, анализах в своем личном кабинете |

## Диаграмма прецедентов

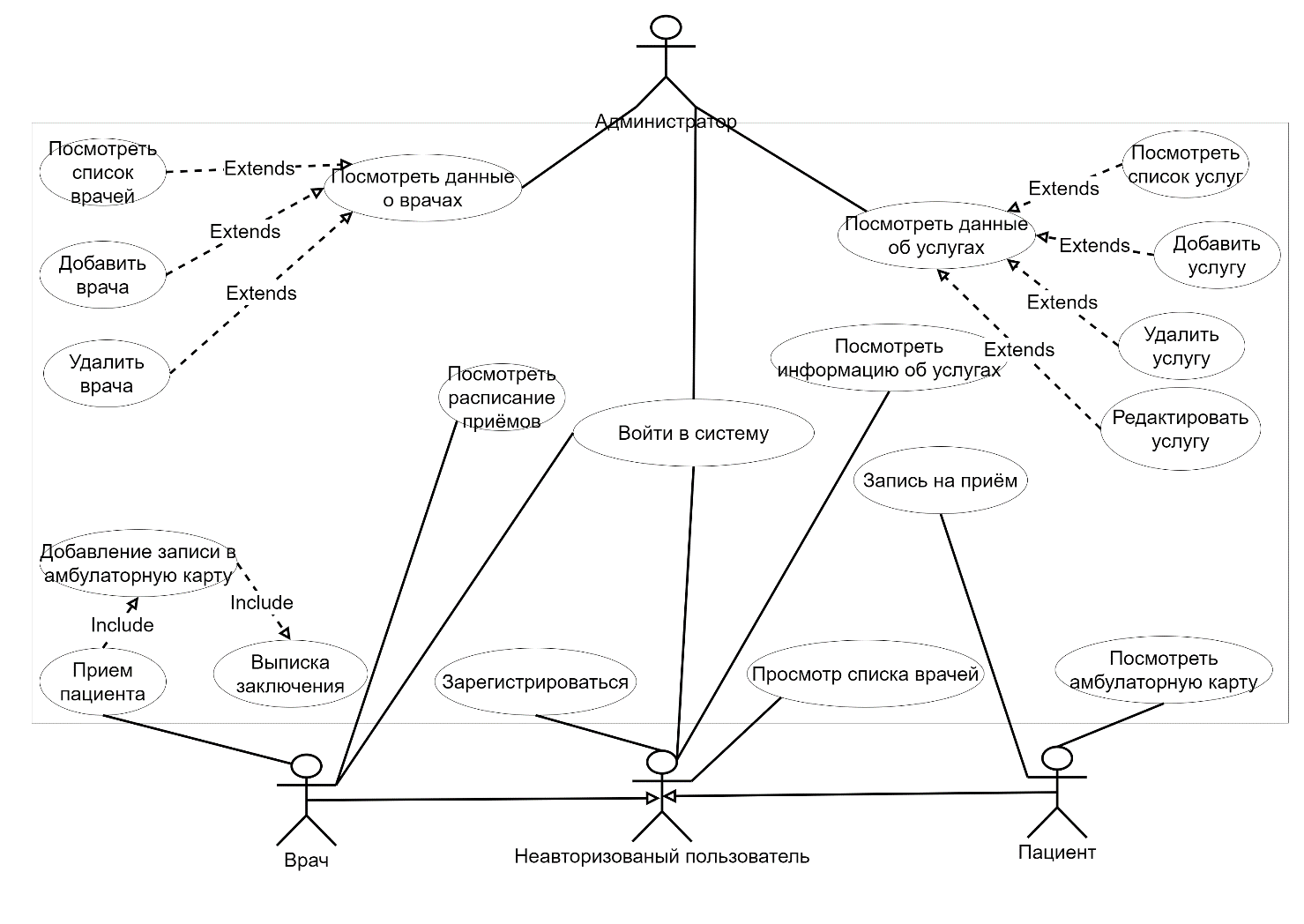


Рисунок 1. Диаграмма прецедентов

## Диаграмма классов

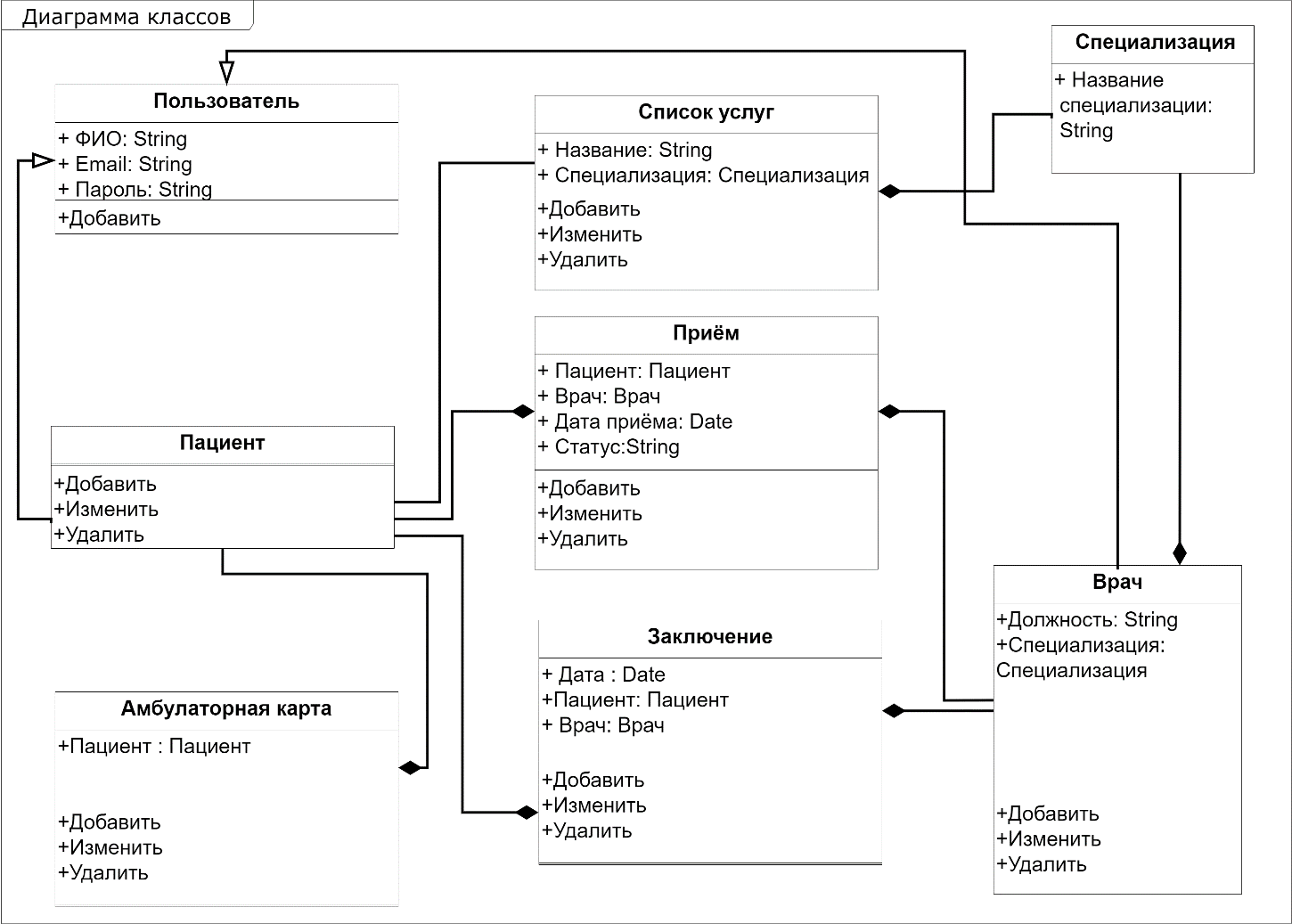


Рисунок 2. Диаграмма классов

## Обоснование связей между классами

Связи между классами. Таблица 7

|  |  |  |  |
| --- | --- | --- | --- |
| Связь | Обязательность | Кратность | Тип связи |
| Класс «Пользователь» и класс «Пациент» | Обязательна, так как класс «Пациент» строиться на основе класса «Пользователь» |  | Тип связи между классами - обобщение |
| Класс «Пользователь» и класс «Врач» | Обязательна, так как класс «Врач» строиться на основе класса «Пользователь |  | Тип связи между классами - обобщение |
| Класс «Пациент» и класс «Список услуг» | Обязательна, так как у каждый пациент должен видеть список услуг |  | Тип связи между классами - Ассоциация т.к. пациент может пользоваться списком услуг |
| Класс «Пациент» и класс «Приём» | Обязательна, так как у каждого приема должен быть пациент |  | Тип связи между классами - композиция т.к. прием не может существовать без пациента |
| Класс «Врач» и класс «Приём» | Обязательна, так как у каждого приема должен быть врач | Один врач ко многим приемам,  у одного приема может быть только один врач | Тип связи между классами - композиция т.к. прием не может существовать без врача |
| Класс «Пациент» и класс «Заключение» | Обязательна, так как у каждого заключения должен быть пациент | Один пациент ко многим заключениям,  у одного заключения может быть только один пациент | Тип связи между классами - композиция т.к. заключение не может существовать без пациента |
| Класс «Пациент» и класс «Амбулаторная карта» | Обязательна, так как у каждой амбулаторной карты должен быть пациент | Один пациент к одной амбулаторной карте,  Амбулаторная карта уникальна для каждого пациента | Тип связи между классами - композиция т.к. амбулаторная карта не может существовать без пациента |
| Класс «Врач» и класс «Заключение» | Обязательна, так как у каждого заключения должен быть врач | Один врач ко многим заключениям,  у одного заключения может быть только один врач | Тип связи между классами - композиция т.к. заключение не может существовать без врача |

## Диаграмма состояний

Диаграмма состояний для объекта «Приём» (Рисунок 3).

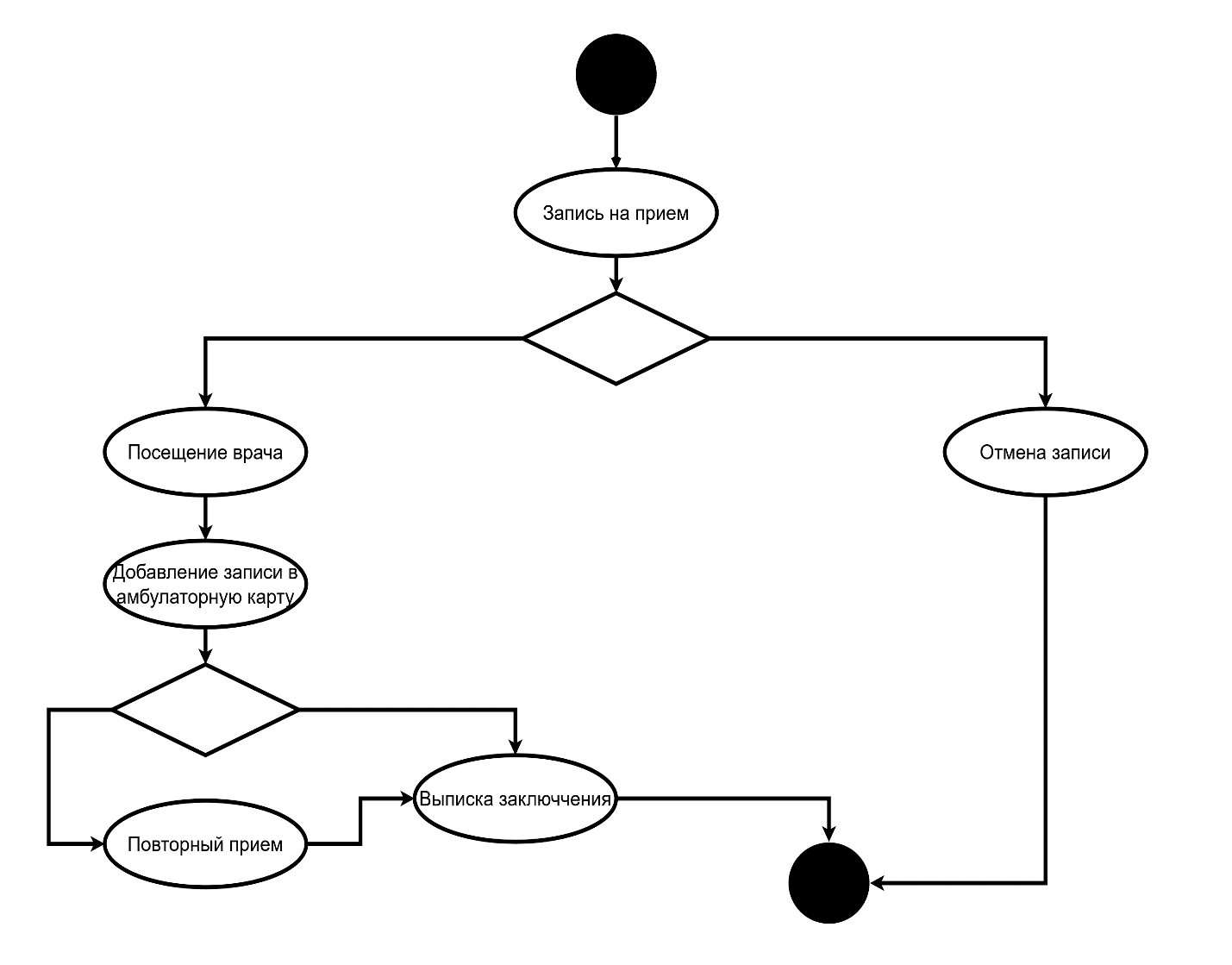


Рисунок 3. Диаграмма состояний

## Диаграмма последовательностей

На рисунке 4 представлена диаграмма последовательностей.

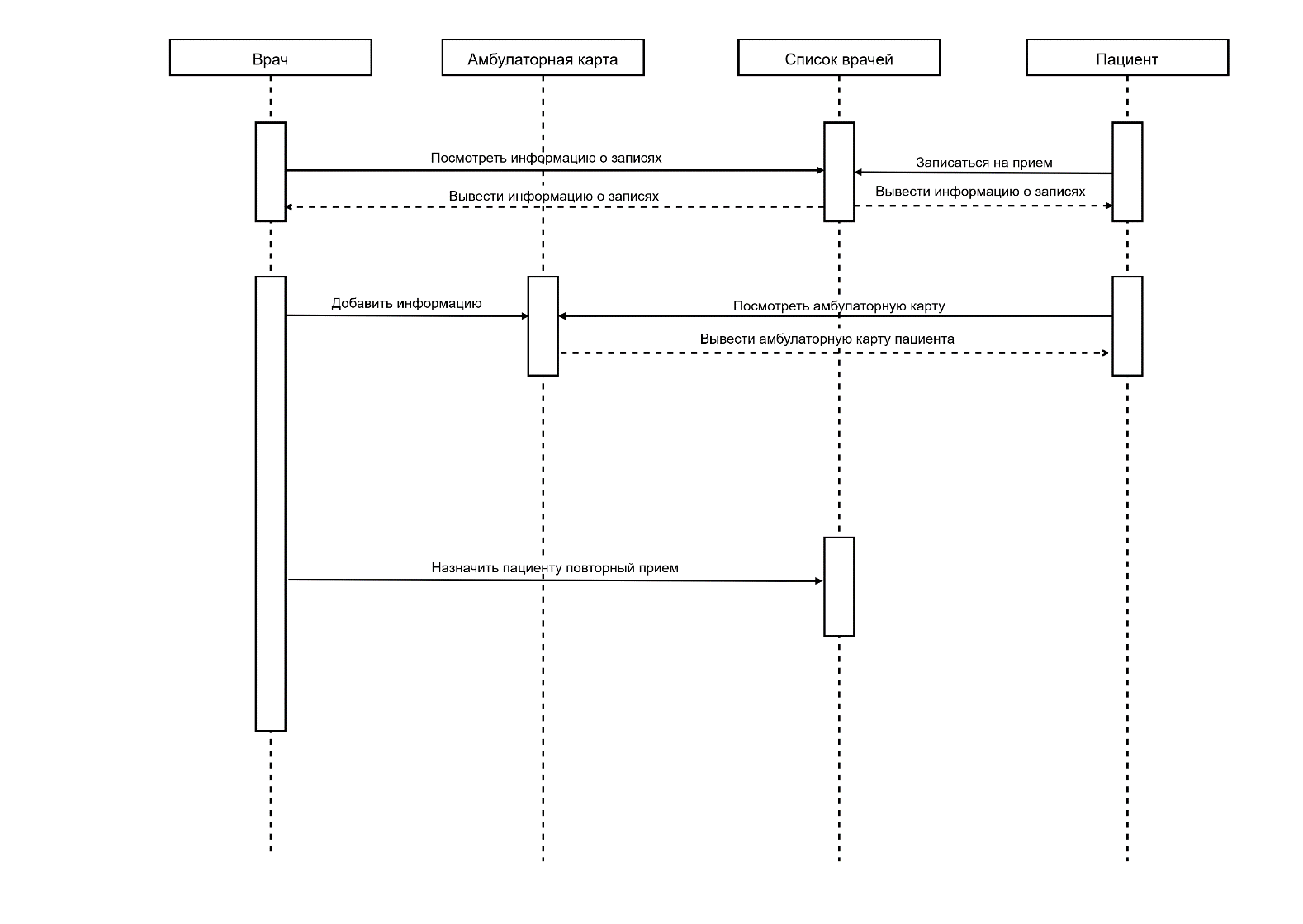


Рисунок 4. Диаграмма последовательностей

## Физическая модель базы данных

На рисунке 5 представлена физическая модель базы данных.

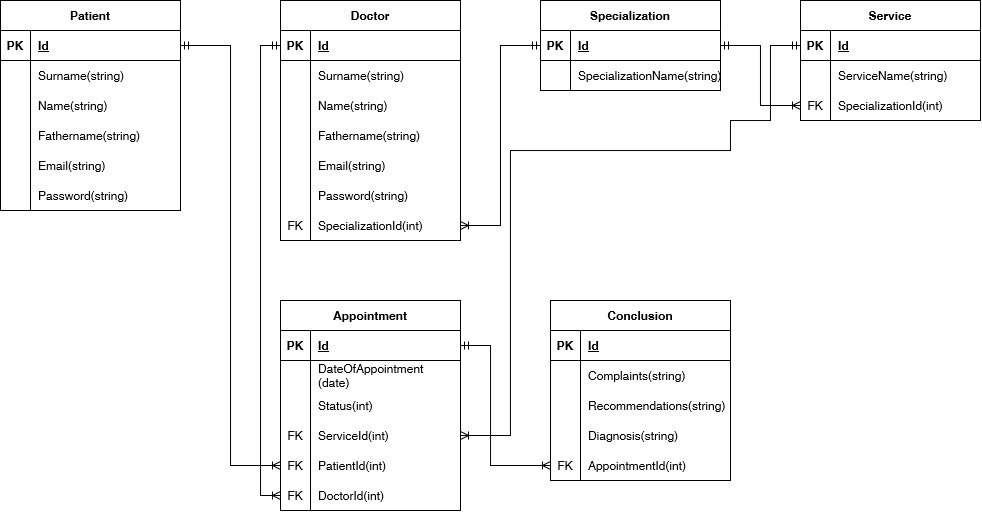


Рисунок 5. Физическая модель базы данных.

# Реализация компонентов системы

## Реализация авторизации и аутентификации пользователей приложения

Разработанное приложение предусматривает многопользовательский формат использования приложения. Поэтому была разработана система авторизации и аутентификации пользователей на основе cookie. Есть три роли: пациент, врач и администратор. Соответственно для регистрации как пациент, необходимо заполнить форму на сайте, для регистрации врачом, необходимо зайти под администратором и уже там добавить врача или изменить его данные. Ниже представлен контроллер, отвечающий за авторизацию и аутентификацию.

Код контроллера с методами для пациента представлен в приложении А.

## Реализация личного кабинета администратора

Для администратора были разработаны CRUD операции для сущности "Врач" и "Услуга". Данные методы представлены в контроллере "AdminController".

Код контроллера с методами для пациента представлен в приложении Б.

## Реализация личного кабинета врача

Для врача была разработана возможность просмотра записей на приём на определенную дату. При выборе нужного приема, врачу предлагается расписать результаты обследования пациента. При успешном вводе нужных данных, составляется заключение, которое отобразится уже в амбулаторной карте пациента.

Код контроллера с методами для пациента представлен в приложении В.

## Реализация личного кабинета пациента

Для реализации личного кабинета пациента была разработана возможность записи на прием к врачу. Запись состоит из трех этапов, сначала пациенту предлагается выбрать услугу, затем предлагается выбрать врача, который предоставляет эту услугу и далее выбирается дата и время. По завершении выводится информация об успешной записи к врачу. И у врача, к которому была произведена запись, сможет увидеть эту запись.

Также у пациента есть возможность просмотреть свои записи на приём и просмотреть свою амбулаторную карту, в которой представлены заключения по результатам приёмов.

Код контроллера с методами для пациента представлен в приложении Г.

# Тестирование системы

Демонстрация главной страницы приложения.

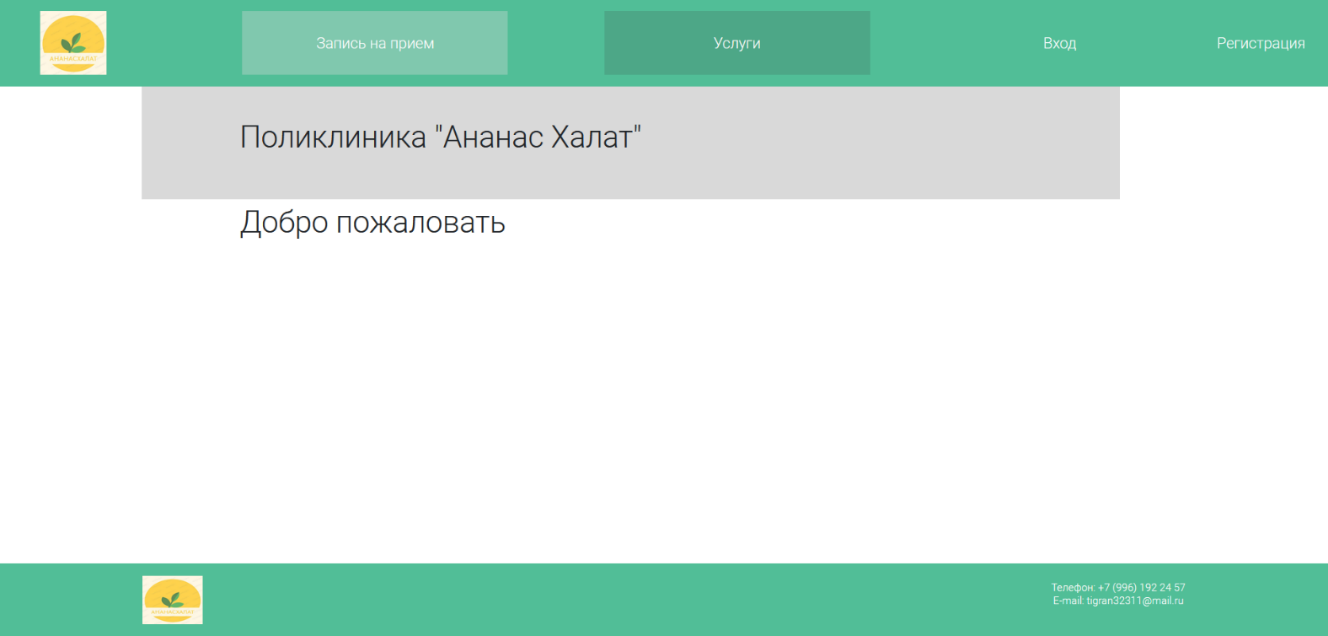


Рисунок 6. Главная страница

Демонстрация формы для регистрации и входа в систему.

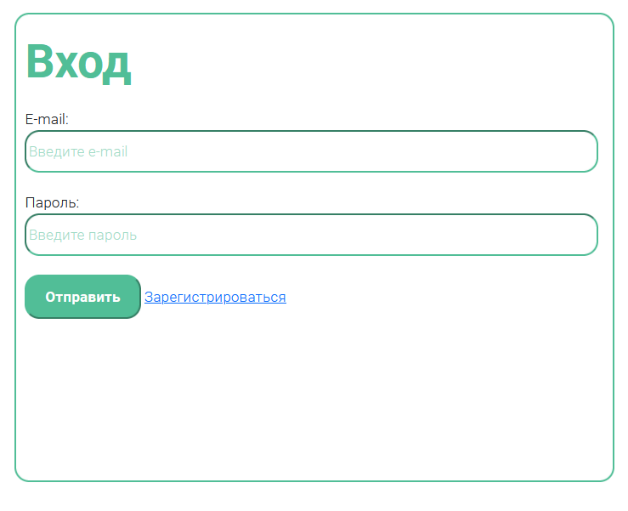


Рисунок 7. Вход в систему

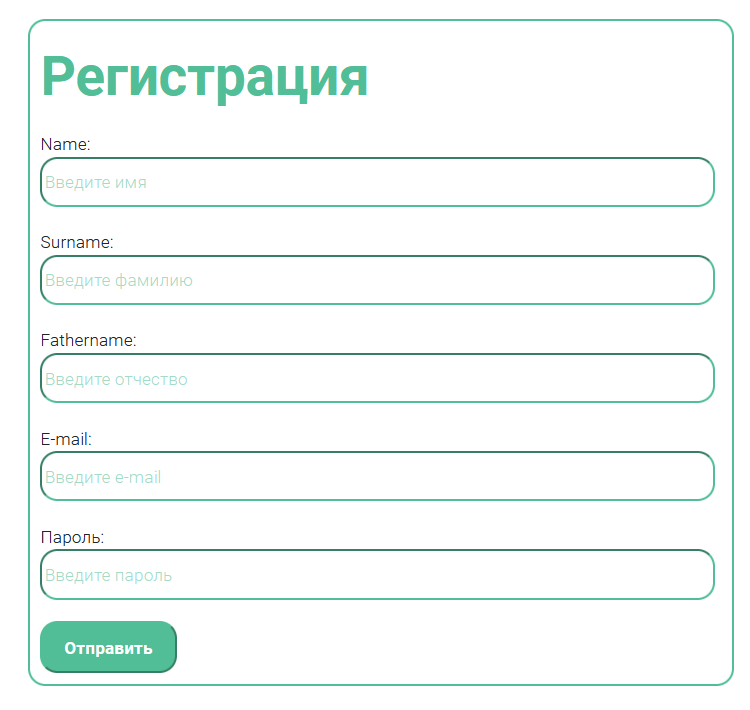


Рисунок 8. Регистрация

Демонстрация личного кабинета администратора

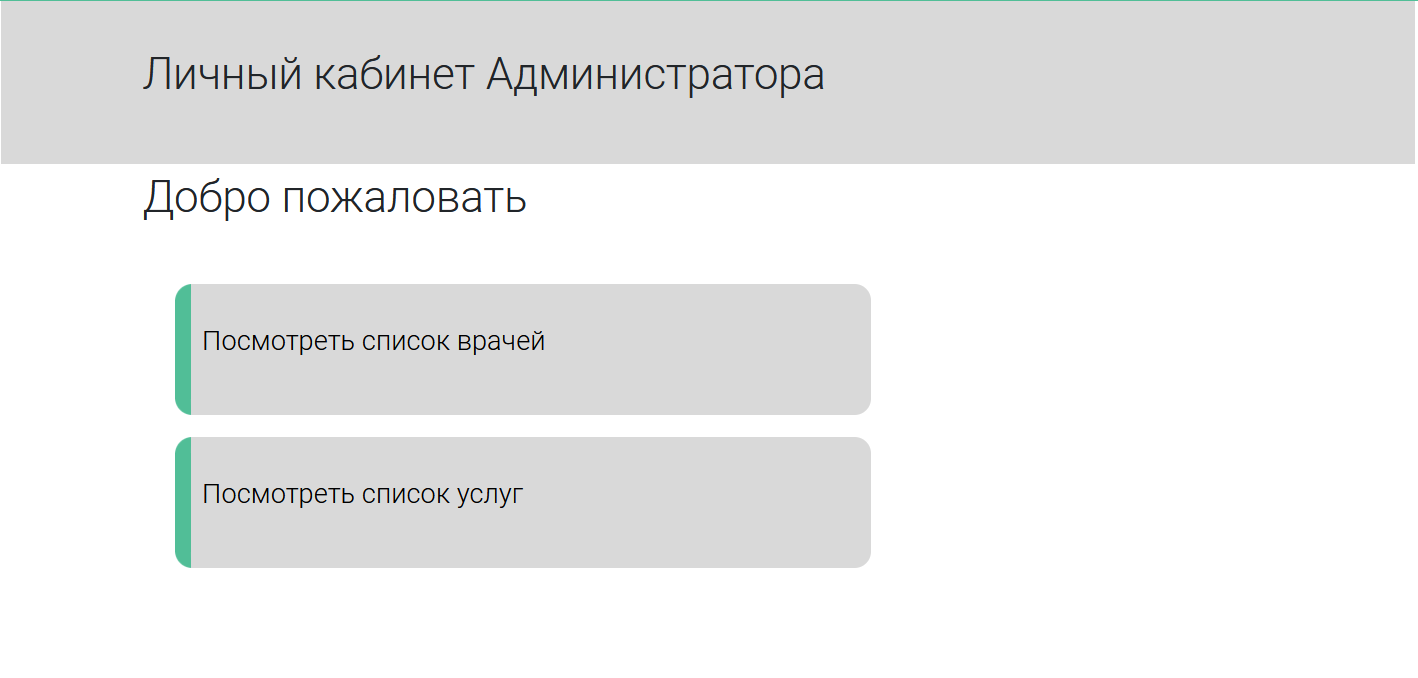


Рисунок 9. Личный кабинет администратора

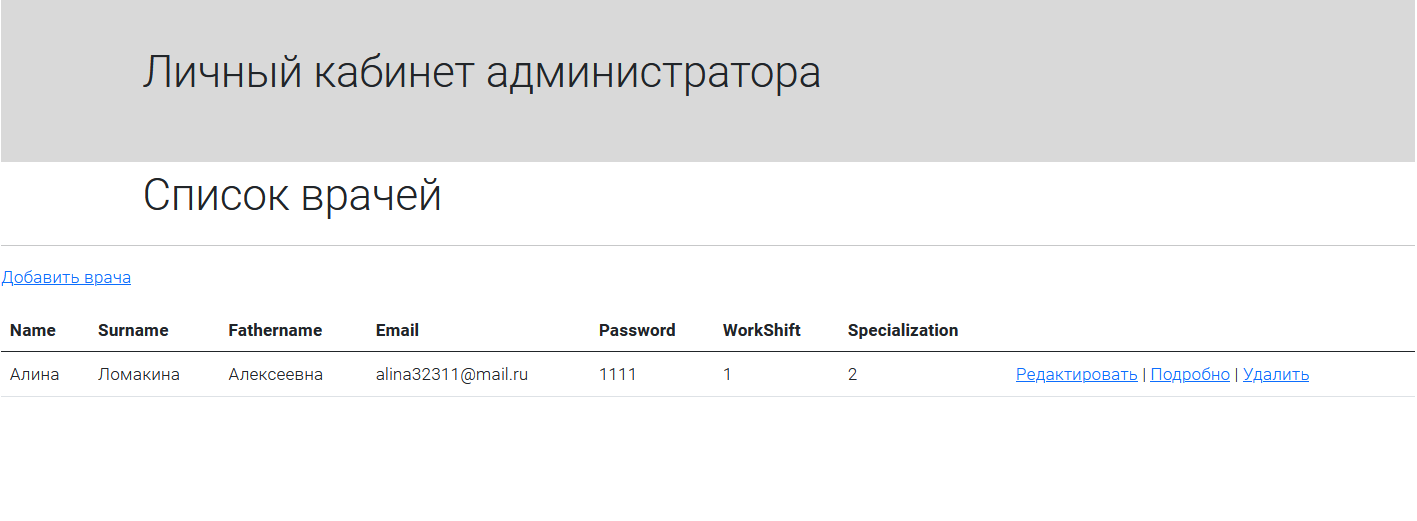


Рисунок 10. Список врачей

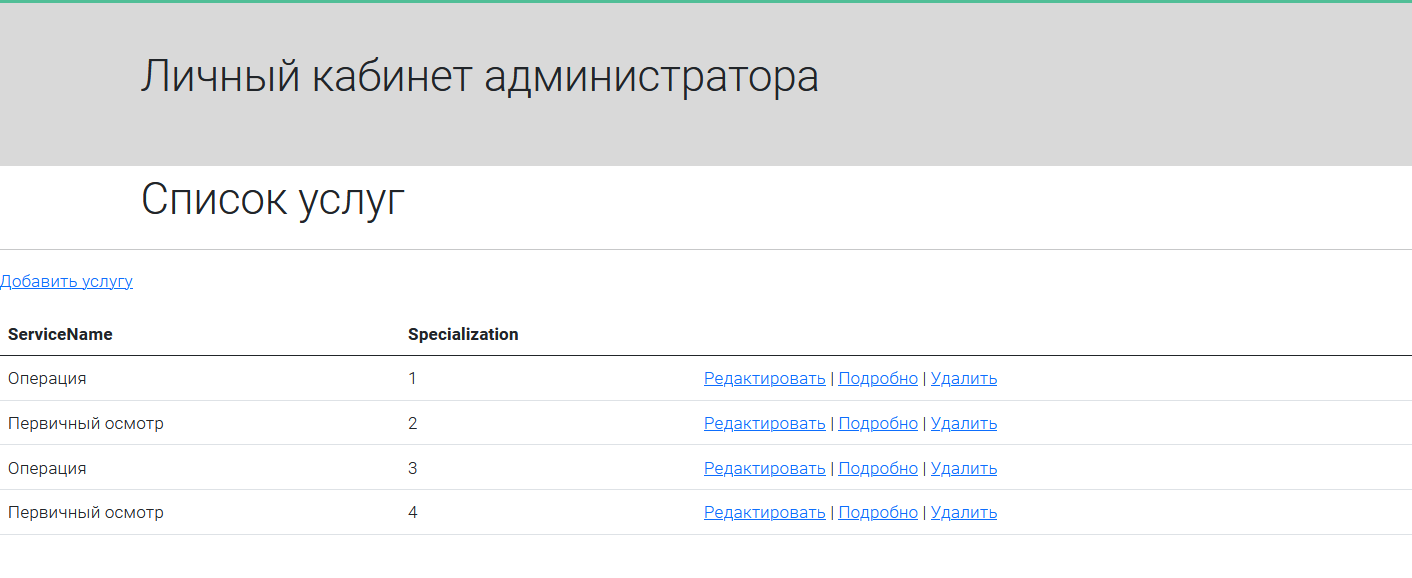


Рисунок 11.Список услуг

Демонстрация личного кабинета врача

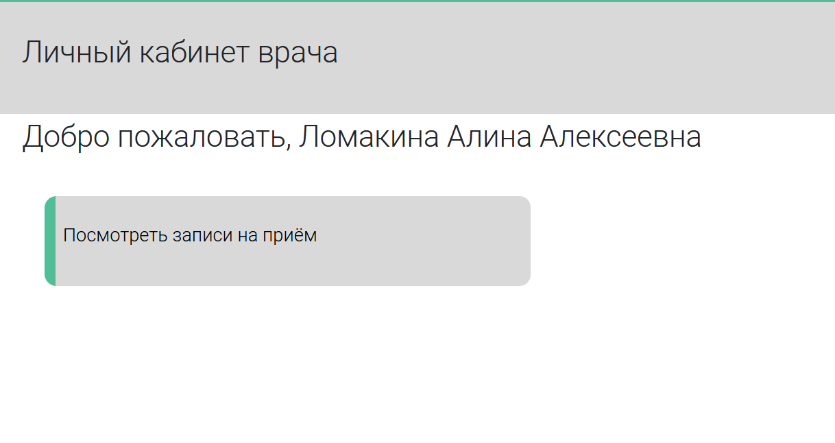


Рисунок 12. Личный кабинет врача

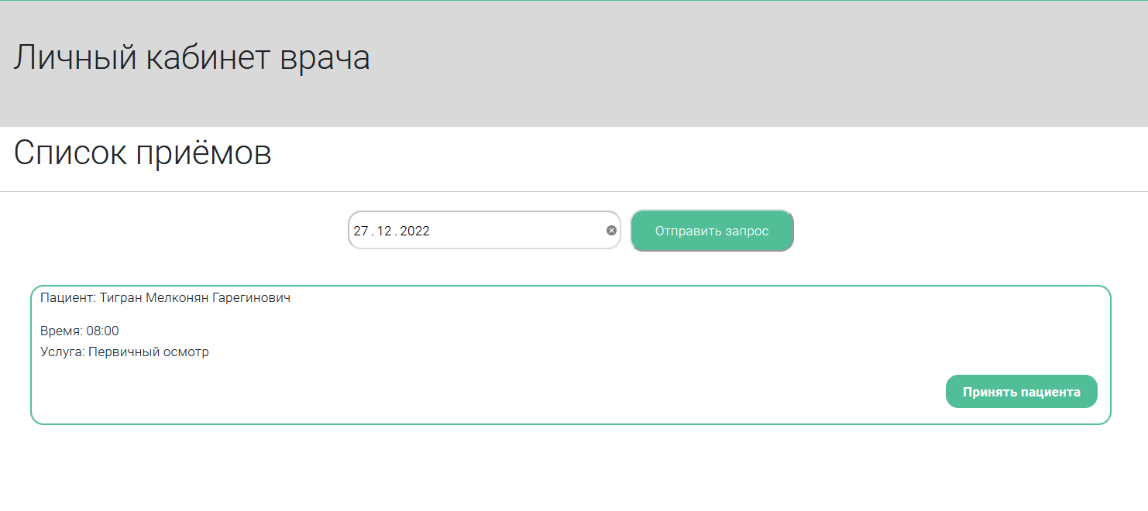


Рисунок 13. Поиск приёмов по дате

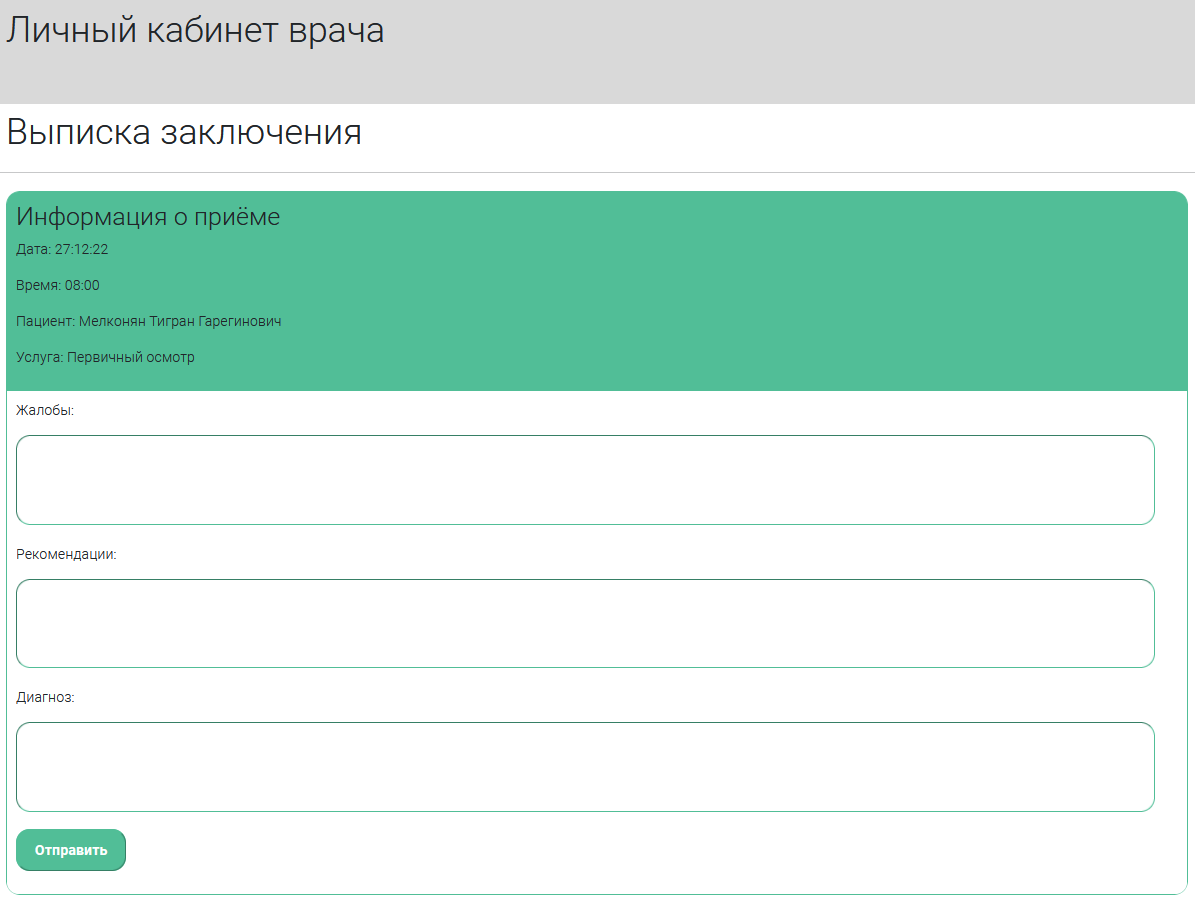


Рисунок 14. Выписка заключения.

Демонстрация личного кабинета пациента

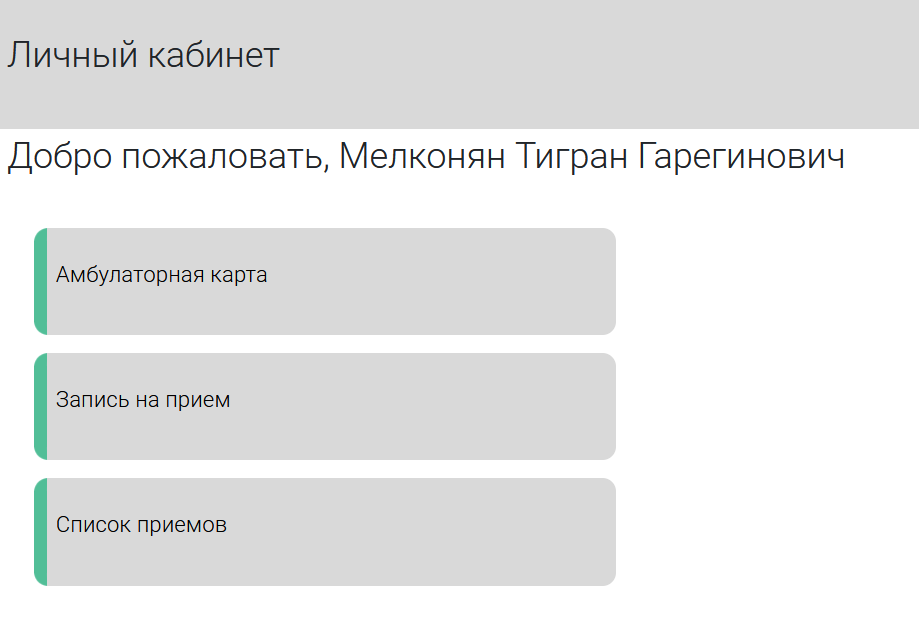


Рисунок 15. Личный кабинет пациента

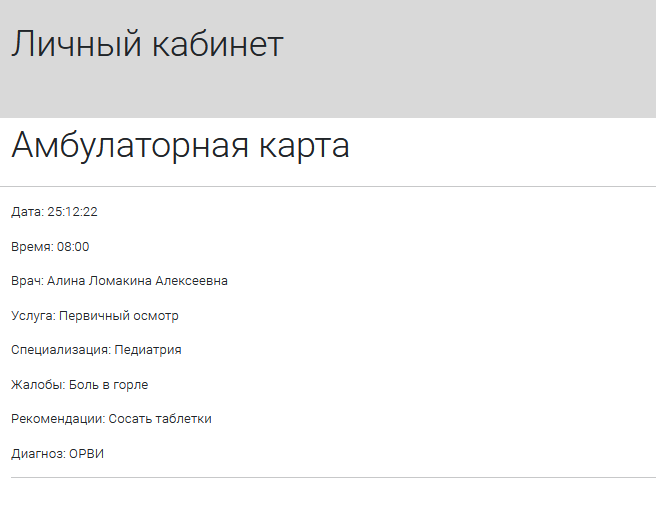


Рисунок 16. Амбулаторная карта пациента.

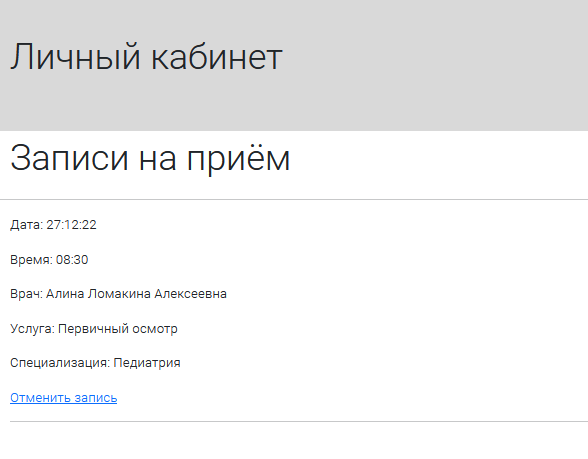


Рисунок 17. Просмотр записей на приём.

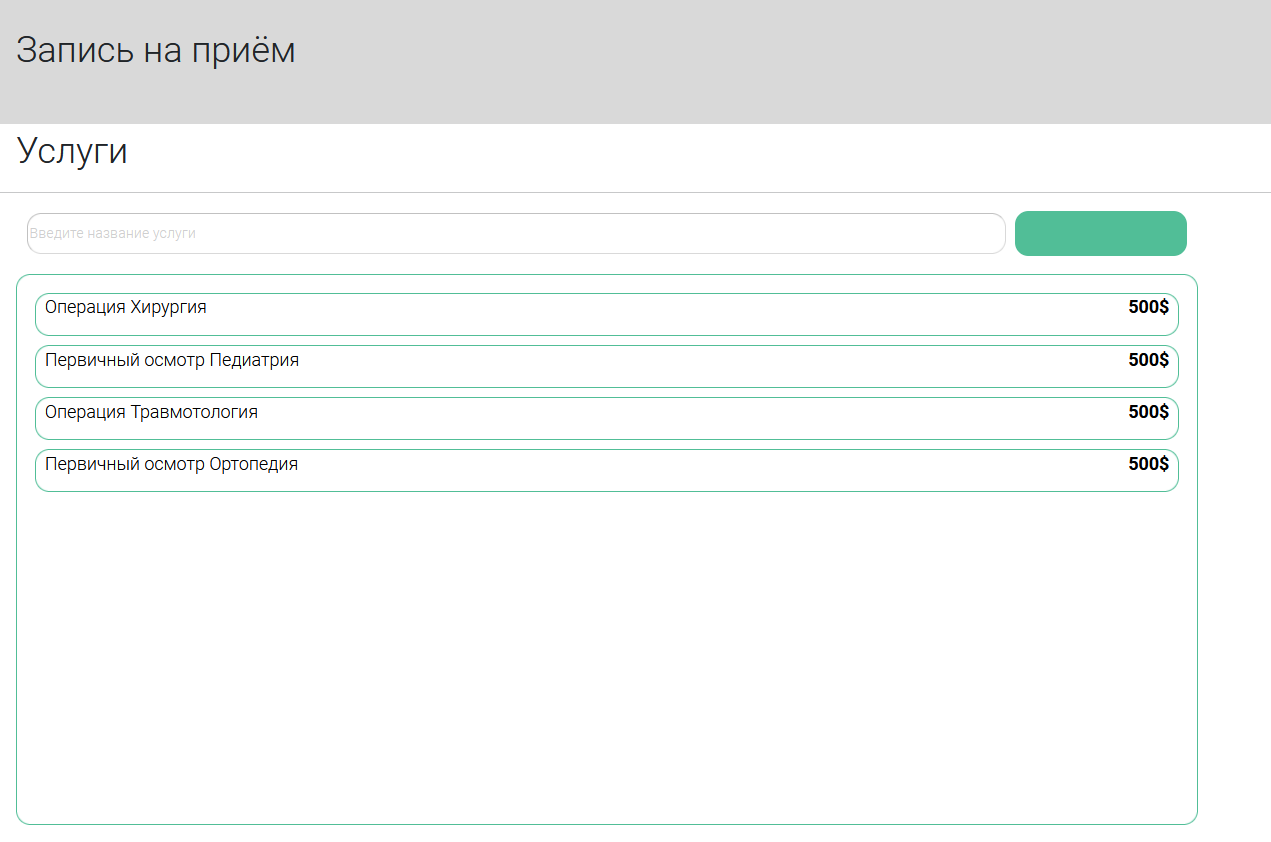


Рисунок 18. Выбор услуги.

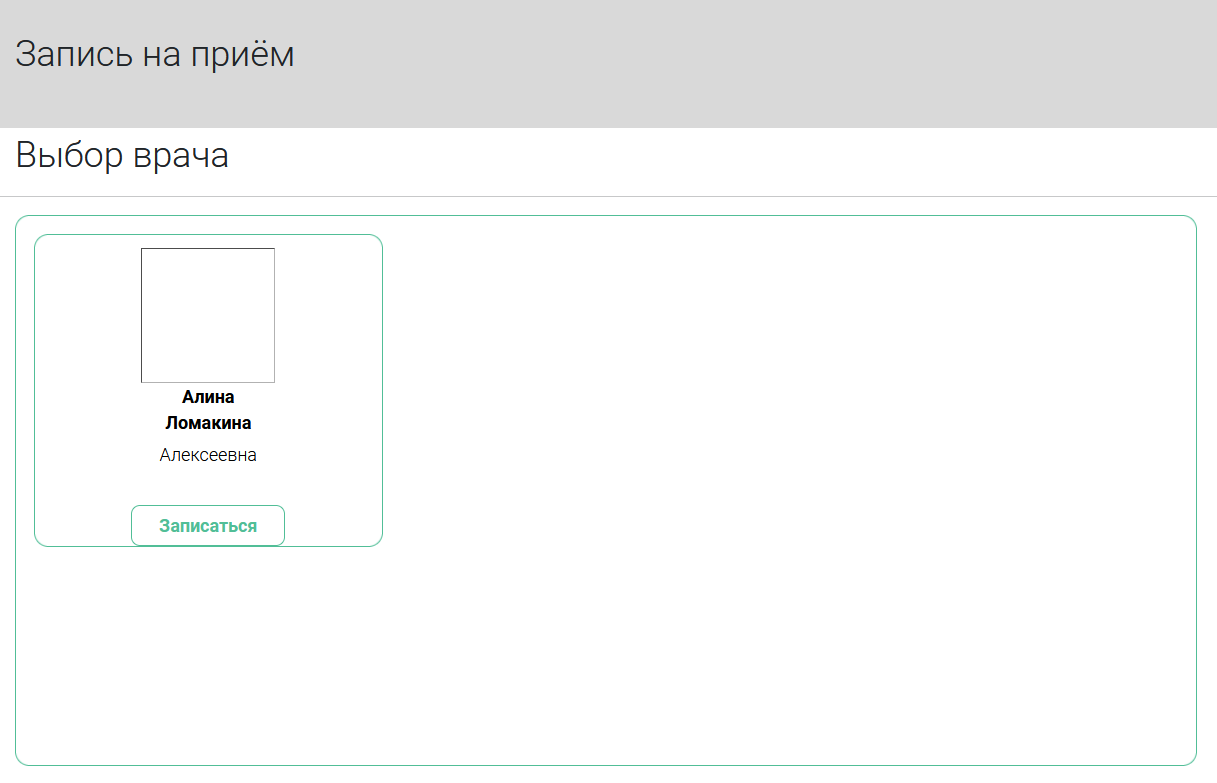


Рисунок 19. Выбор врача.

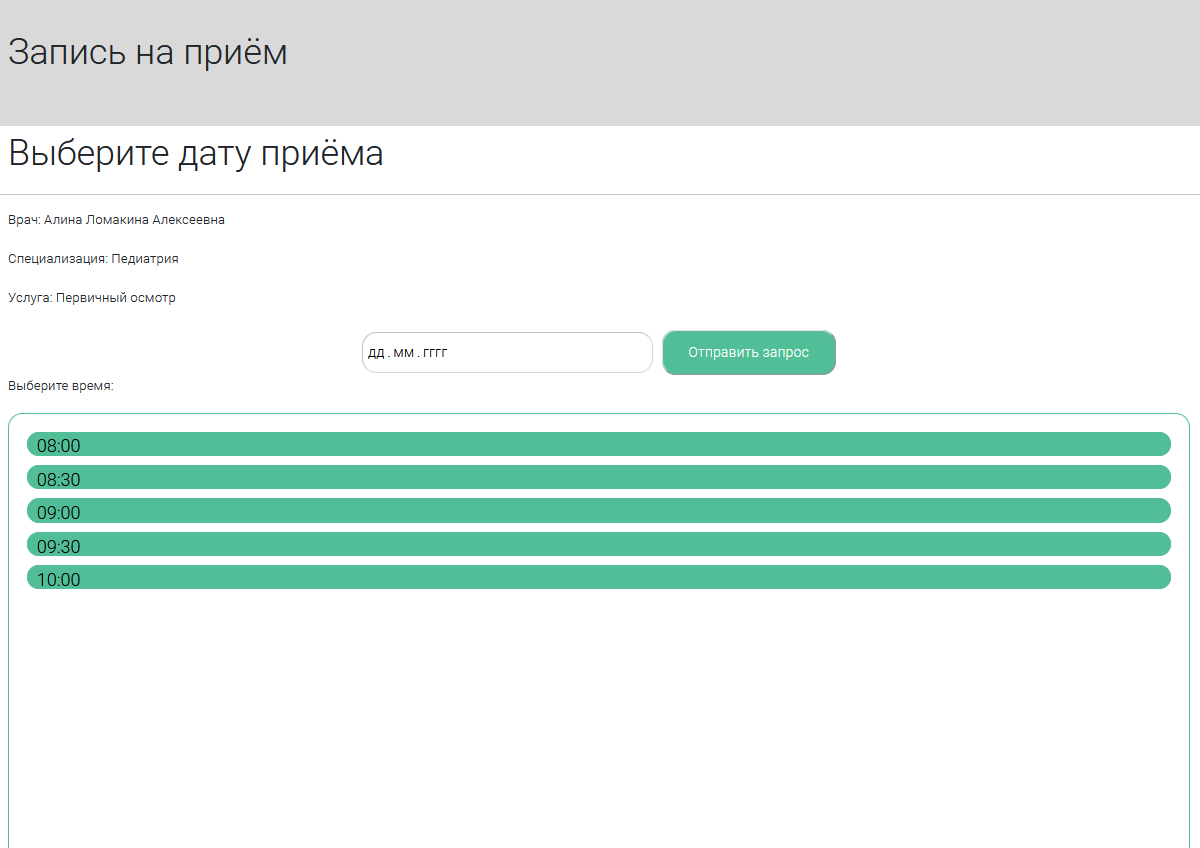


Рисунок 20. Выбор даты и времени для записи.

# заключение

Было разработано многопользовательское приложение для информационной системы "Поликлиника". Был реализован основной бизнес-процесс поликлиники – это запись на приём, и проведение приёма. Также была создана страница администратора, который может редактировать список врачей, услуг и информацию о них.

Приложение было разработано на основе ASP.NET Core 6.0, была использована база данных MSSQL.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный сайт Microsoft, с обучающими материалами для разработки приложений на ASP.NET Core 6.0 - https://learn.microsoft.com/ru-ru/aspnet/core/getting-started/?view=aspnetcore-6.0&tabs=windows

2. Лок, Э. ASP.NET Core в действии : руководство / Э. Лок ; перевод с английского Д. А. Беликова. — Москва : ДМК Пресс, 2021. — 906 с. — ISBN 978-5-97060-550-9. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/241079.

3. Снетков, В. М. Программирование на ASP.NET : учебное пособие / В. М. Снетков. — 2-е изд. — Москва : ИНТУИТ, 2016. — 901 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/100402

4. Конфигурирование и настройка Microsoft ASP.NET : учебное пособие. — 2-е изд. — Москва : ИНТУИТ, 2016. — 248 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/100625

ПРИЛОЖЕНИЕ А

using Clinic.Data;

using Clinic.Models;

using Microsoft.AspNetCore.Mvc;

namespace Clinic.Controllers

{

public class AccountController : Controller

{

private readonly ClinicDbContext \_context;

public AccountController(ClinicDbContext clinicDbContext)

{

\_context = clinicDbContext;

}

//проверка Email при регистрации

public async Task<IActionResult> CheckEmail([Bind("Email")] Patient patient)

{

if (\_context.patients.Any(x => x.Email == patient.Email))

{

return Json(false);

}

return Json(true);

}

public IActionResult Login([Bind ("Email,Password")] ForLogin forLogin)

{

if (\_context.patients.Any(a=>a.Password == forLogin.Password && a.Email==forLogin.Email))

{

int id = \_context.patients.Where(a => a.Password == forLogin.Password && a.Email == forLogin.Email).Select(a=>a.Id).FirstOrDefault();

Response.Cookies.Append("userId", id.ToString());

return RedirectToAction("MainPagePatient","Patient", new {id = id});

} else if (\_context.doctors.Any(a => a.Password == forLogin.Password && a.Email == forLogin.Email))

{

int id = \_context.doctors.Where(a => a.Password == forLogin.Password && a.Email == forLogin.Email).Select(a => a.Id).FirstOrDefault();

Response.Cookies.Append("userId", id.ToString());

return RedirectToAction("MainPageDoctor", "Doctor", new { id = id });

}

else if (\_context.admins.Any(a => a.Password == forLogin.Password && a.Email == forLogin.Email))

{

int id = \_context.admins.Where(a => a.Password == forLogin.Password && a.Email == forLogin.Email).Select(a => a.Id).FirstOrDefault();

Response.Cookies.Append("userId", id.ToString());

return RedirectToAction("MainPageAdmin", "Admin", new { id = id });

}

return View();

}

public IActionResult Logout()

{

Response.Cookies.Delete("userId");

return RedirectToAction("MainPage", "Home"); //добавить контроллер и метод для главной страницы

}

public IActionResult Register([Bind ("Name,Surname,Fathername,Email,Password")] Patient patient)

{

if (patient.Name!=null)

{

\_context.patients.Add(patient);

\_context.SaveChanges();

int id = \_context.patients.Where(a => a.Password == patient.Password && a.Email == patient.Email).Select(a => a.Id).FirstOrDefault();

Response.Cookies.Append("userId", id.ToString());

return RedirectToAction("MainPagePatient", "Patient", new { id = id });

} else

{

return View();

}

}

}

}

ПРИЛОЖЕНИЕ Б

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Clinic.Data;

using Clinic.Models;

namespace Clinic.Controllers

{

public class AdminController : Controller

{

private readonly ClinicDbContext \_context;

public AdminController(ClinicDbContext context)

{

\_context = context;

}

public IActionResult MainPageAdmin()

{

return View();

}

// GET: Admin

public async Task<IActionResult> IndexDoctors()

{

var clinicDbContext = \_context.doctors.Include(d => d.Specialization);

return View(await clinicDbContext.ToListAsync());

}

public async Task<IActionResult> IndexServices()

{

var clinicDbContext = \_context.services.Include(d => d.Specialization);

return View(await clinicDbContext.ToListAsync());

}

// GET: Admin/Details/5

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

{

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

{

return NotFound();

}

var doctor = await \_context.doctors

.Include(d => d.Specialization)

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

if (doctor == null)

{

return NotFound();

}

return View(doctor);

}

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

{

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

{

return NotFound();

}

var doctor = await \_context.services

.Include(d => d.Specialization)

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

if (doctor == null)

{

return NotFound();

}

return View(doctor);

}

// GET: Admin/Create

public IActionResult CreateDoctors()

{

ViewData["SpecializationId"] = new SelectList(\_context.specializations, "Id", "Id");

return View();

}

public IActionResult CreateServices()

{

ViewData["SpecializationId"] = new SelectList(\_context.specializations, "Id", "Id");

return View();

}

// POST: Admin/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]

public async Task<IActionResult> CreateDoctors([Bind("Id,Name,Surname,Fathername,Email,Password,WorkShift,SpecializationId")] Doctor doctor)

{

if (ModelState.IsValid)

{

\_context.Add(doctor);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

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

return View(doctor);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> CreateServices([Bind("Id,ServiceName,SpecializationId")] Service service)

{

if (ModelState.IsValid)

{

\_context.Add(service);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(IndexServices));

}

ViewData["SpecializationId"] = new SelectList(\_context.specializations, "Id", "Id", service.SpecializationId);

return View(service);

}

// GET: Admin/Edit/5

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

{

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

{

return NotFound();

}

var service = await \_context.services.FindAsync(id);

if (service == null)

{

return NotFound();

}

ViewData["SpecializationId"] = new SelectList(\_context.specializations, "Id", "Id", service.SpecializationId);

return View(service);

}

// POST: Admin/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]

public async Task<IActionResult> EditDoctors(int id, [Bind("Id,Name,Surname,Fathername,Email,Password,WorkShift,SpecializationId")] 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["SpecializationId"] = new SelectList(\_context.specializations, "Id", "Id", doctor.SpecializationId);

return View(doctor);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> EditServices(int id, [Bind("Id,ServiceName,SpecializationId")] Service service)

{

if (id != service.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(service);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!DoctorExists(service.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["SpecializationId"] = new SelectList(\_context.specializations, "Id", "Id", service.SpecializationId);

return View(service);

}

// GET: Admin/Delete/5

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

{

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

{

return NotFound();

}

var doctor = await \_context.doctors

.Include(d => d.Specialization)

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

if (doctor == null)

{

return NotFound();

}

return View(doctor);

}

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

{

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

{

return NotFound();

}

var service = await \_context.services

.Include(d => d.Specialization)

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

if (service == null)

{

return NotFound();

}

return View(service);

}

// POST: Admin/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmedDoctors(int id)

{

if (\_context.doctors == null)

{

return Problem("Entity set 'ClinicDbContext.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);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmedServices(int id)

{

if (\_context.doctors == null)

{

return Problem("Entity set 'ClinicDbContext.doctors' is null.");

}

var service = await \_context.services.FindAsync(id);

if (service != null)

{

\_context.services.Remove(service);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ServiceExists(int id)

{

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

}

}

}

ПРИЛОЖЕНИЕ В

using Microsoft.AspNetCore.Mvc;

using Clinic.Data;

using Clinic.Models;

using Microsoft.EntityFrameworkCore;

namespace Clinic.Controllers

{

public class DoctorController : Controller

{

private readonly ClinicDbContext \_context;

public DoctorController(ClinicDbContext context)

{

\_context = context;

}

public IActionResult MainPageDoctor()

{

ViewData["Doctor"] = \_context.doctors.Where(a => a.Id == Convert.ToInt32(Request.Cookies["userId"])).ToList();

return View();

}

public IActionResult SuccessCreate()

{

return View();

}

public IActionResult ListAppointments()

{

ViewData["Appointment"] = \_context.appointments.Where(a => a.DoctorId == Convert.ToInt32(Request.Cookies["userId"]) && a.Status == 1 && a.DateOfAppointment==DateTime.Today).ToList();

ViewData["Services"] = \_context.services.ToList();

ViewData["Doctor"] = \_context.doctors.Where(a => a.Id == Convert.ToInt32(Request.Cookies["userId"])).ToList();

ViewData["Patient"] = \_context.patients.ToList();

return View();

}

[HttpPost]

public async Task<IActionResult> ListAppointments([Bind("DateOfAppointment")] Appointment appointment)

{

if (appointment.DateOfAppointment!=null && \_context.appointments.Any(a => a.DoctorId == Convert.ToInt32(Request.Cookies["userId"]) && a.Status == 1 && a.DateOfAppointment == appointment.DateOfAppointment))

{

ViewData["Appointment"] = await \_context.appointments.Where(a => a.DoctorId == Convert.ToInt32(Request.Cookies["userId"]) && a.Status == 1 && a.DateOfAppointment == appointment.DateOfAppointment).ToListAsync();

ViewData["Services"] = \_context.services.ToList();

ViewData["Doctor"] = \_context.doctors.Where(a => a.Id == Convert.ToInt32(Request.Cookies["userId"])).ToList();

ViewData["Patient"] = \_context.patients.ToList();

ViewData["times"] = \_context.times.ToList();

return View();

}

else

{

return View();

}

}

public IActionResult DoAppointment(int id)

{

Response.Cookies.Append("appointmentId", id.ToString());

ViewData["Appointment"] = \_context.appointments.Where(a => a.Id == Convert.ToInt32(Request.Cookies["appointmentId"])).ToList();

ViewData["Services"] = \_context.services.ToList();

ViewData["Doctor"] = \_context.doctors.Where(a => a.Id == Convert.ToInt32(Request.Cookies["userId"])).ToList();

ViewData["Patient"] = \_context.patients.ToList();

ViewData["times"] = \_context.times.ToList();

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> CreateConclusion([Bind("Complaints,Recommendations,Diagnosis")] Сonclusion сonclusion)

{

Appointment appointment = new Appointment();

appointment.Id = Convert.ToInt32(Request.Cookies["appointmentId"]);

appointment.Status = 3;

appointment.DateOfAppointment = \_context.appointments.Where(a => a.Id == appointment.Id).Select(a => a.DateOfAppointment).FirstOrDefault();

appointment.PatientId = \_context.appointments.Where(a => a.Id == appointment.Id).Select(a => a.PatientId).FirstOrDefault();

appointment.DoctorId = \_context.appointments.Where(a => a.Id == appointment.Id).Select(a => a.DoctorId).FirstOrDefault();

appointment.ServiceId = \_context.appointments.Where(a => a.Id == appointment.Id).Select(a => a.ServiceId).FirstOrDefault();

appointment.timeId = \_context.appointments.Where(a => a.Id == appointment.Id).Select(a => a.timeId).FirstOrDefault();

\_context.appointments.Update(appointment);

await \_context.SaveChangesAsync();

сonclusion.AppointmentId = appointment.Id;

Response.Cookies.Delete("appointmentId");

\_context.сonclusions.Add(сonclusion);

await \_context.SaveChangesAsync();

return Redirect("~/Doctor/SuccessCreate");

}

}

}

ПРИЛОЖЕНИЕ Г

using Clinic.Data;

using Clinic.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Collections;

namespace Clinic.Controllers

{

public class PatientController : Controller

{

private readonly ClinicDbContext \_context;

public PatientController(ClinicDbContext context)

{

\_context = context;

}

public IActionResult MainPagePatient() //добавить данные клиента

{

ViewData["Patient"] = \_context.patients.Where(a => a.Id == Convert.ToInt32(Request.Cookies["userId"])).ToList();

return View();

}

public IActionResult SuccessCreate()

{

return View();

}

public IActionResult MakeAppointmentChooseService()

{

ViewData["Specialization"] = \_context.specializations.ToList();

ViewData["Services"] = \_context.services.ToList();

return View();

}

public IActionResult MakeAppointmentChooseDoctor(int id)

{

//получение специализации услуги выбранной пользователем

int specId = \_context.services.Where(a=> a.Id==id).Select(a=> a.SpecializationId).FirstOrDefault();

Response.Cookies.Append("servicesId", id.ToString());

Response.Cookies.Append("specId", specId.ToString());

//отправка данных на представление

ViewData["Specialization"] = \_context.specializations.ToList();

ViewData["Doctors"] = \_context.doctors.Where(a=> a.SpecializationId==specId).ToList();

return View();

}

[HttpGet]

public IActionResult MakeAppointmentChooseData(int id)

{

Response.Cookies.Append("docId", id.ToString());

// логика расписания int docId

ViewData["Services"] = \_context.services.Where(a => a.Id == Convert.ToInt32(Request.Cookies["servicesId"])).ToList();

ViewData["Specialization"] = \_context.specializations.Where(a => a.Id == Convert.ToInt32(Request.Cookies["specId"])).ToList();

ViewData["Doctors"] = \_context.doctors.Where(a => a.Id == id).ToList();

ViewData["Appointment"] = \_context.appointments.ToList();

return View();

}

[HttpPost,ActionName("MakeAppointmentChooseData")]

public async Task<IActionResult> MakeAppointmentChooseData(DateTime DateOfAppointment)

{

Appointment appointment = new Appointment();

Response.Cookies.Append("dateOfApp", DateOfAppointment.ToString());

ViewData["times"] = await \_context.times.ToListAsync();

ViewData["Services"] = \_context.services.Where(a => a.Id == Convert.ToInt32(Request.Cookies["servicesId"])).ToList();

ViewData["Specialization"] = \_context.specializations.Where(a => a.Id == Convert.ToInt32(Request.Cookies["specId"])).ToList();

ViewData["Doctors"] = \_context.doctors.Where(a => a.Id == Convert.ToInt32(Request.Cookies["docId"])).ToList();

if (DateOfAppointment > DateTime.Now)

{

ViewData["Appointment"] = \_context.appointments.Where(a => a.DateOfAppointment == DateOfAppointment).ToList();

}

return View(Convert.ToInt32(Request.Cookies["docId"]));

}

public async Task<IActionResult> CreateApp(int id)

{

Appointment appointment = new Appointment();

appointment.DateOfAppointment = Convert.ToDateTime(Request.Cookies["dateOfApp"]);

appointment.Status = 1;

appointment.PatientId = Convert.ToInt32(Request.Cookies["userId"]);

appointment.DoctorId = Convert.ToInt32(Request.Cookies["docId"]);

appointment.ServiceId = Convert.ToInt32(Request.Cookies["servicesId"]);

appointment.timeId = id;

Response.Cookies.Delete("docId");

Response.Cookies.Delete("servicesId");

Response.Cookies.Delete("dateOfApp");

\_context.appointments.Add(appointment);

await \_context.SaveChangesAsync();

return Redirect("~/Patient/SuccessCreate");

}

public IActionResult LookAppointments()

{

ViewData["Appointments"] = \_context.appointments

.Where(a => a.PatientId == Convert.ToInt32(Request.Cookies["userId"]) && a.DateOfAppointment >= DateTime.Now && a.Status<3)

.ToList();

ViewData["Services"] = \_context.services.ToList();

ViewData["Specialization"] = \_context.specializations.ToList();

ViewData["Doctors"] = \_context.doctors.ToList();

ViewData["times"] = \_context.times.ToList();

return View();

}

public IActionResult LookAmbulancyCard()

{

ViewData["Appointments"] = \_context.appointments.ToList();

ViewData["Services"] = \_context.services.ToList();

ViewData["Specialization"] = \_context.specializations.ToList();

ViewData["Doctors"] = \_context.doctors.ToList();

ViewData["times"] = \_context.times.ToList();

ViewData["Conclusions"] = \_context.сonclusions

.Where(a => a.AppointmentId == \_context.appointments.Where(a => a.PatientId == Convert.ToInt32(Request.Cookies["userId"]))

.Select(a=>a.Id).FirstOrDefault())

.ToList();

return View();

}

public async Task<IActionResult> DeleteAppointments(int id)

{

var appointment = await \_context.appointments.FindAsync(id);

if (appointment != null)

{

\_context.appointments.Remove(appointment);

}

await \_context.SaveChangesAsync();

ViewData["Appointments"] = \_context.appointments

.Where(a => a.PatientId == Convert.ToInt32(Request.Cookies["userId"]) && a.DateOfAppointment >= DateTime.Now && a.Status < 3)

.ToList();

ViewData["Services"] = \_context.services.ToList();

ViewData["Specialization"] = \_context.specializations.ToList();

ViewData["Doctors"] = \_context.doctors.ToList();

ViewData["times"] = \_context.times.ToList();

return Redirect("~/Patient/LookAppointments");

}

public IActionResult ListServices()

{

ViewData["Services"] = \_context.services.Include(s => s.Specialization).ToList();

return View();

}

}

}