МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

MEDBOOKER. ВЕБ–ПРИЛОЖЕНИЕ УПРАВЛЕНИЯ РАСПИСАНИЕМ И ЗАПИСЯМИ ПАЦИЕНТОВ В МЕДИЦИНСКИХ УЧРЕЖДЕНИЯХ

Студент 03.06.25 Савосин Владислав Алексеевич
(подпись, дата)

Руководитель 03.06.25 Зинкин С. А.
(подпись, дата)

Нормоконтролёр 03.06.25 А.С. Бычков
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 28.05.25 г. № 16)

Заведующий кафедрой _____ М.А. Митрохин
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 6 июня 2025 г. № 17)

Секретарь ГЭК _____ Е.И. Гурин
(подпись)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«06» 05 2025г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Савосин Владислав Алексеевич гр.21ВВП2 факультета вычислительной техники.

2.Тема работы: «MedBooker. Веб–приложение управления расписанием и записями пациентов в медицинских учреждениях»

Тема утверждена приказом ПГУ №601/о от "06" мая 2025г.

3. Руководитель бакалаврской работы д.т.н., профессор кафедры ВТ ПГУ Зинкин С. А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать веб–приложение система управления расписанием работы врачей и медсестёр в поликлинике, позволяющая: вести запись пациентов к врачам через онлайн–расписание, управлениерабочими слотами врачей, администрирование клиник, врачей и их расписаний, контроль статусов записей (подтвержденные, ожидающие, отмененные).

5. Перечень подлежащих разработке вопросов
Требуется произвести анализ предметных областей: изучить потребности медицинских учреждений в автоматизации записи пациентов, добавить календарь с визуализацией свободных/занятых слотов, оптимизировать маршруты Flask для минимизации нагрузки на сервер. Подготовить пользовательские инструкции для каждой роли (пациент, врач, администратор).

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	26.04.2025 – 28.05.2025	
2	Анализ требований	32	29.04.2025 – 30.05.2025	
3	Создание макета графического интерфейса	30	1.04.2025 – 3.05.2025	
4	Выбор технологий, применяемых при проектировании и разработке	40	4.05.2025 – 8.05.2025	
5	Реализация графического интерфейса	66	9.05.2025 – 15.05.2025	
6	Разработка основной части программы	70	16.05.2025 – 23.05.2025	
7	Оформление пояснительной записки	54	24.05.2025 – 30.05.2025	

Дата выдачи задания "6" мая 2025 г.

Руководитель бакалаврской работы 06.05.2025 Зинкин С. А.
(подпись, дата)

Задание к исполнению принял студент 06.05.2025 Савосин В. А.
(подпись, дата)

Работу к защите допустить
Декан факультета 06.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 104 листов, 47 рисунков, 10 таблиц, 8 источников и 3 приложения.

MEDBOOKER. ВЕБ–ПРИЛОЖЕНИЕ УПРАВЛЕНИЯ РАСПИСАНИЕМ И ЗАПИСЯМИ ПАЦИЕНТОВ В МЕДИЦИНСКИХ УЧРЕЖДЕНИЯХ

Цель работы – разработать и внедрить веб–приложение для автоматизации процессов управления медицинскими учреждениями, направленное на оптимизацию записи пациентов, организацию расписаний врачей и администрирование клиник. Проект призван повысить доступность медицинских услуг за счет удобного онлайн–бронирования слотов, снизить нагрузку на персонал через автоматизацию рутинных задач (создание расписаний, подтверждение записей) и обеспечить прозрачность взаимодействия между пациентами, врачами и администраторами. Реализация системы MedBooker позволит улучшить качество обслуживания, минимизировать ошибки ручного учета и создать единую цифровую среду для эффективного управления ресурсами медицинских организаций..

В результате проведенной работы было разработано приложение, отвечающее следующим поставленным задачам: работа в системе под учетной записью врача/медсестры, режим администратора отдельный инетрфейс для пациентов.

Содержание

Введение 6

1 Веб-приложение управления расписанием и записями пациентов в медицинских учреждениях 8

1.1 Анализ предметной области 8

1.1.1 Актуальность разработки 8

1.1.2 Анализ существующих аналогов 9

1.1.3 Обоснование актуальности разработки программы 16

1.1.4 Постановка задачи 17

1.2 Выбор технологий и средств разработки приложения 19

1.2.1 Выбор языка программирования 20

1.2.2 Выбор веб – фреймворка 21

1.2.3 Выбор СУБД 22

1.2.4 Выбор среды разработки 23

1.2.5 Выбор технологий фронтеда 23

1.2.6. Обмен данными JSON 25

1.3 Проектирование информационной системы 26

1.3.1 Паттерн MVC 26

1.3.2 Архитектура приложения 28

1.3.3 Проектирование базы данных 34

1.4 Разработка приложения 40

1.4.1 Описание программы 40

1.4.2 Руководство пользователя 47

1.4.3 Тестирование и отладка 56

2 Охрана труда 59

Заключение 63

Список использованных источников 65

Приложение А - UML - Диаграммы 66

Приложение В - Листинг программы 72

Приложение С - Презентация 99

Перечень сокращений и обозначений

В данной работе применяют следующие обозначения и сокращения

СУБД – Система управления базами данных

MVC (Model–View–Controller) – архитектурный паттерн, разделяющий приложение на три компонента: модель, представление и контроллер

JSON (JavaScript Object Notation) – текстовый формат обмена данными

HTML (HyperText Markup Language) – язык разметки для создания веб – страниц

CSS (Cascading Style Sheets) – язык стилей для оформления HTML – документов

ORM (Object – Relational Mapping) – технология преобразования данных между объектами и реляционными базами данных

API (Application Programming Interface) – интерфейс взаимодействия между компонентами программного обеспечения

ACID – набор свойств транзакций (атомарность, согласованность, изолированность, долговечность)

UML (Unified Modeling Language) – язык графического описания объектно – ориентированных систем

JIT (Just–In–Time) – технология компиляции «на лету»

REST (Representational State Transfer) – архитектурный стиль для создания веб – сервисов

Введение

В современном мире информационные технологии играют ключевую роль в оптимизации процессов даже в таких социально значимых сферах, как здравоохранение. Благодаря ним удаётся сократить время на рутинные задачи, значительно уменьшить человеческие ошибки и повысить качество услуг. Особенно необходима автоматизация и для медицинских учреждений.

Одной из основных задач в поликлиниках остается управление расписанием врачей. Ручное планирование смен, учет специализации и нагрузки часто приводит к конфликтам в графиках, переработкам персонала и ошибкам при записи пациентов. Крупные медицинские центры частично решают эти проблемы с помощью специализированных CRM систем, но в большинстве поликлиник до сих пор преобладает бумажный документооборот или использование устаревших решений, не адаптированных под современные требования.

Для небольших поликлиник внедрение коммерческих систем автоматизации экономически нецелесообразно из–за высокой стоимости лицензий, сложности настройки и отсутствия кадровых ресурсов для обслуживания. Из–за этого администраторы тратят часы на составление расписаний вручную, пациенты сталкиваются с недостоверной информацией о доступных окошек для записи, а врачи с перегруженными сменами.

В таких условиях разработка специализированного бюджетного веб–приложения для управления расписаниями необходима. Система должна не только автоматизировать планирование, но и обеспечить прозрачность для пациентов.

Целью выпускной работы является:

– анализ проблем ручного управления расписаниями в поликлиниках и существующих решений;

– разработка веб–приложения для автоматизированного планирования смен медицинского персонала;

– обеспечение безопасности данных и адаптивного интерфейса для разных категорий пользователей.

Реализация данного проекта позволит поликлиникам сократить время на составление расписаний на 70%, исключить ошибки в распределении смен и повысить доступность информации для пациентов. Это не только оптимизирует работу медицинского персонала, но и улучшит качество обслуживания, сохранив конкурентоспособность учреждения в условиях цифровизации здравоохранения.

1 Веб–приложение управления расписанием и записями пациентов в медицинских учреждениях

1.1 Анализ предметной области

1.1.1 Актуальность разработки

Успешное функционирование медицинских учреждений напрямую зависит от эффективного управления ресурсами, включая человеческие. Планирование смен врачей, учет их специализации и нагрузки, предотвращение конфликтов в графиках, обеспечение прозрачности записи пациентов – эти и другие ежедневные задачи требуют от административного персонала значительных временных затрат. При ручном подходе высока вероятность ошибок: перегруженные смены, несоответствие специализации врача и потребностей пациента, некорректная информация о доступных окнах приема.

Для решения данных проблем в крупных медицинских центрах внедряются специализированные CRM–системы, которые берут на себя часть рутинных операций. Но большинство поликлиник, особенно малого масштаба, продолжают использовать устаревшие методы: бумажные журналы, электронные таблицы или локальные программы, не адаптированные под современные требования. Это приводит к снижению качества обслуживания пациентов и увеличению нагрузки на персонал.

Сложность автоматизации управления расписаниями в небольших поликлиниках обусловлена несколькими факторами: высокая стоимость коммерческих решений, необходимость адаптации под специфические требования учреждения, отсутствие ИТ–специалистов для обслуживания сложных систем. В результате администраторы тратят часы на ручное составление графиков, пациенты сталкиваются с недостоверной информацией, а врачи–с ежедневной большой нагрузкой.

Разрабатываемая в рамках данной выпускной работы система предназначена устранить эти проблемы. Веб–приложение для

автоматизированного планирования смен будет обладать следующими преимуществами:

– низкая стоимость внедрения и обслуживания по сравнению с коммерческими аналогами;

– понятный визуальный функционал;

– адаптивный интерфейс для разных категорий пользователей (администраторы, врачи, пациенты).

Основным отличием системы станет ее ориентированность на малые поликлиники. Программа не потребует сложной настройки или дорогостоящего оборудования, а базовый функционал покроет основную задачу: понятное расписание врачей для пацентов.

Таким образом, внедрение моей системы не только оптимизирует работу медицинского персонала, но и повысит уровень доверия к учреждению, что особенно важно в условиях роста конкуренции в сфере здравоохранения. Освободившиеся ресурсы можно направить на улучшение качества услуг, внедрение инновационных методов лечения и персонализированный подход к пациентам.

1.1.2 Анализ существующих аналогов

Среди аналогов систем управления расписаниями в медицине можно выделить следующие решения:

– многофункциональная система «РТ МИС»;

– информационная система «МедСофт: Поликлиника».

Рассмотрим возможности информационной системы «РТ МИС».

Общий вид данной программы представлен (рисунок 1).

Чтобы начать работу с программой, необходимо совершить вход от имени какого–либо пользователя (рисунок 2). Количество доступных функций определяется уровнем привилегий учетной записи. В данном вход осуществлен от имени администратора.

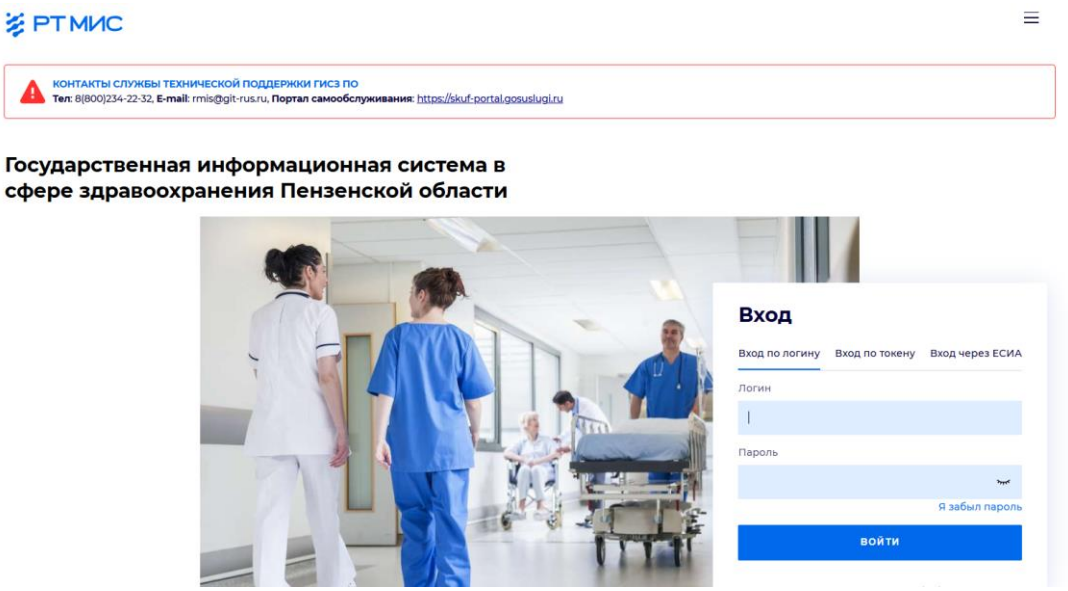


Рисунок 1 – Главный вид программы «РТ МИС»

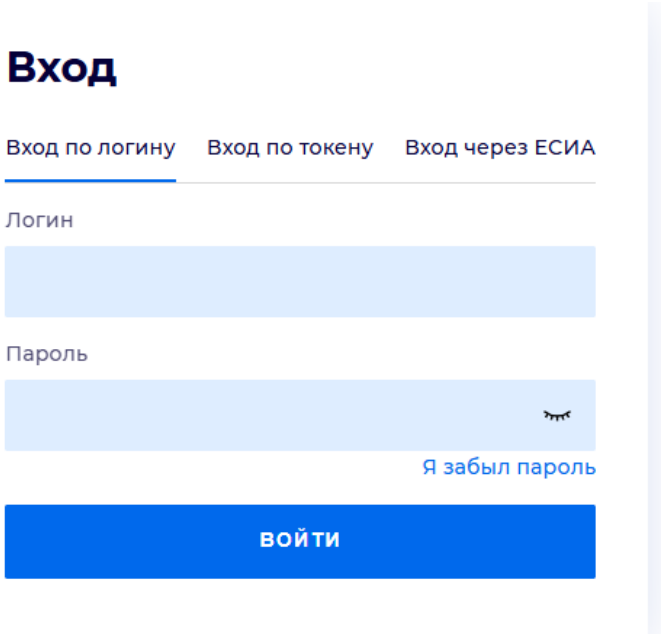


Рисунок 2 – Авторизация в системе

Весь функционал программы представлен в виде вкладок, каждая из которых выполняет определенные задачи.

После авторизации доступна вся информация о пациентах на сегодняшний день (рисунок 3).

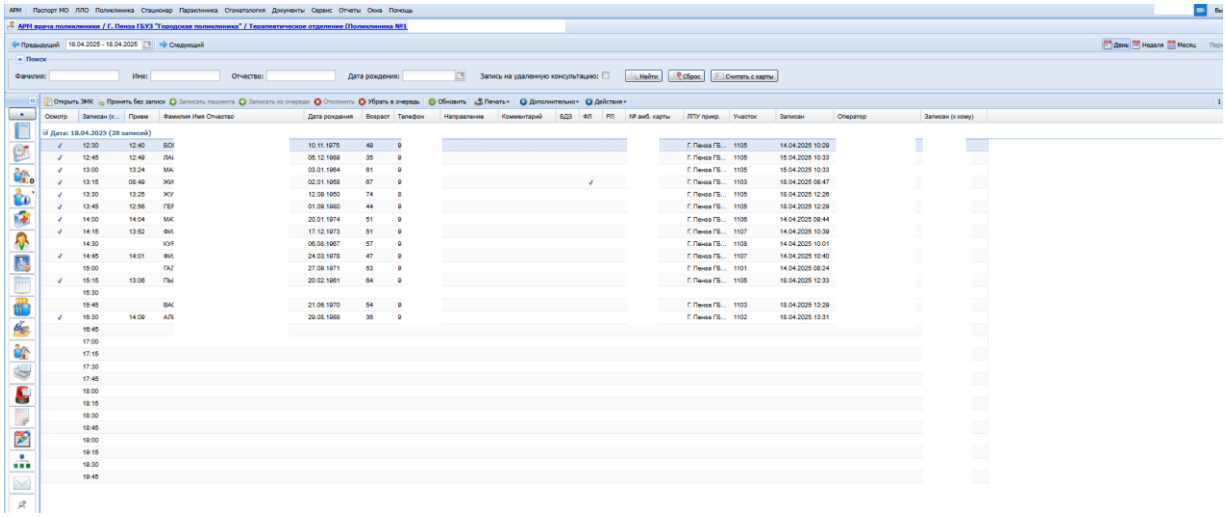


Рисунок 3 – Информация о пациентах на сегодняшний день

Для того чтобы посмотреть информацию на 2 недели вперед по записям, необходимо один раз кликнуть на соответствующую иконку (рисунок 4).

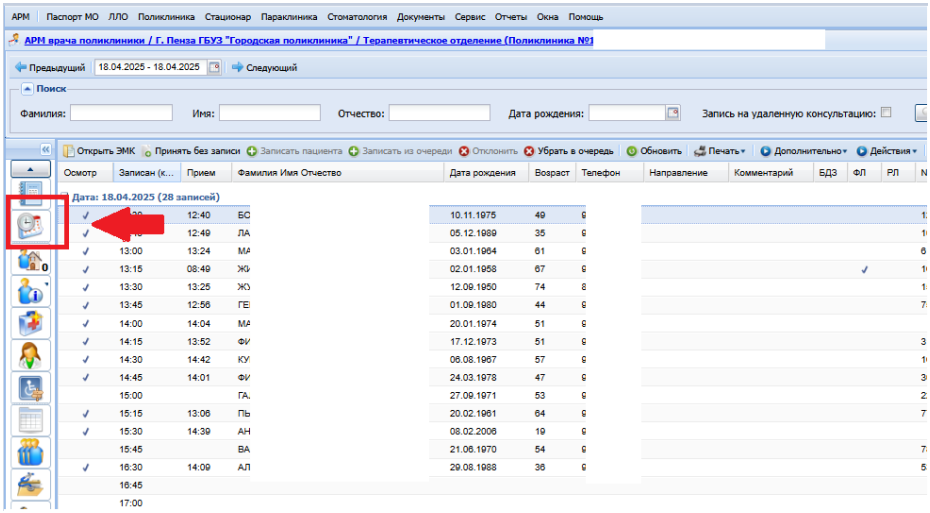


Рисунок 4 – Работа с расписанием

На вкладке «Работа с расписанием» имеется возможность создание записей или просмотра/редактирование уже имеющихся (рисунок 5).

Если врачу нужно посмотреть информацию о записанных пациентов на день или в пределах двух недель, ему достаточно навести курсор мыши на соответствующие время (рисунок 6) или нажать на соответствующий раздел под нужной врачу дате (рисунок 7).

Рисунок 5 – Расписание врача

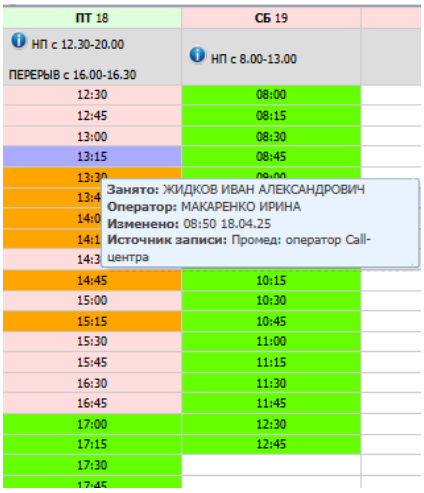


Рисунок 6 – Информация о пациента в ячейке 13:15

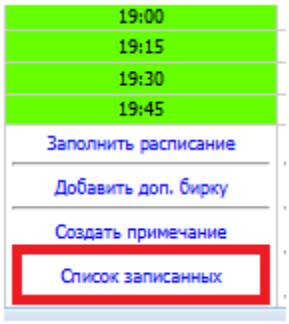


Рисунок 7 – Кнопка «Список записанных»

Нажав на кнопку «Заполнить расписание» (рисунок 8) открывается окошко с ячейками для заполнения: создать на даты, начало работы, окончание работы, длительность приёма (рисунок 9).

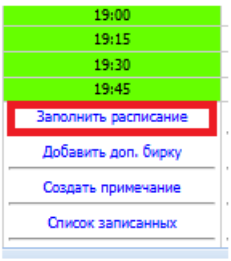


Рисунок 8 – Кнопка «Заполнить расписание»

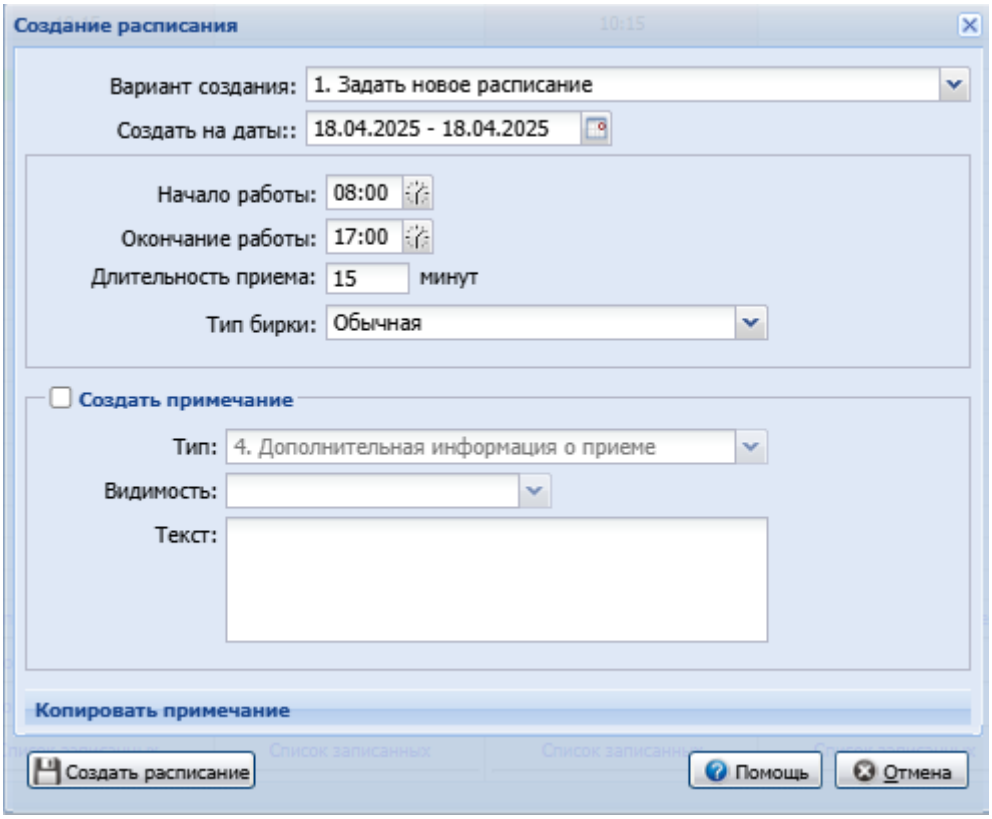


Рисунок 9 – Окошко для создания записи

Программа обладает следующими преимуществами:

– кроме управления расписанием, включает электронные медкарты, учет пациентов, интеграцию с лабораториями и другими медицинскими системами;

– круглосуточная поддержка пользователей;

– подходит для крупных медицинских центров и сетей поликлиник;

– регулярные обновления и адаптация сайта.

Из минусов данной информационной системы можно выделить следующее:

– дорогая лицензия, тем саамы экономически невогдная для малых поликлиник;

– дополнительные расходы на обучение персонала и необходимость постоянного привлечения большого количества IT–специалистов для поддержания бесперебойной работы;

– неудобный и нефункциональный дизайн, сложный для восприятия пользователям;

– нет возможности просматривать информацию самому пациенту о предстоящих визитах в больницу.

Информационная система «МедСофт: Поликлиника».

Чтобы начать работу с программой, необходимо пройти авторизацию (рисунок 10).

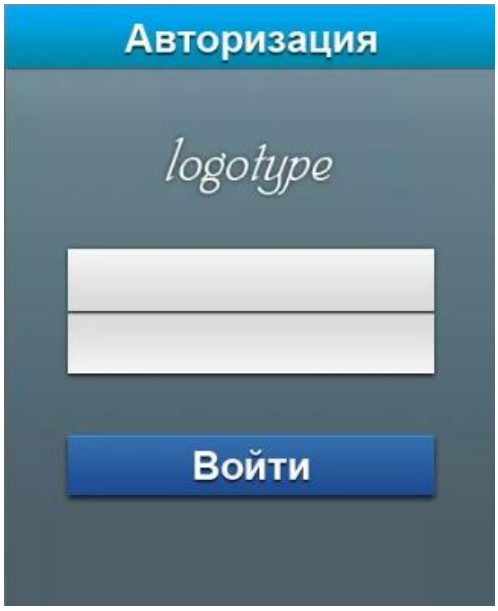


Рисунок 10 – Авторизация в «МедСофт: Поликлиника»

После авторизации откроется окно со списком записавшихся пациентов. В каждой строке содержится информация: ID, имя, адрес, город, телефон, факс, email (рисунок 11).

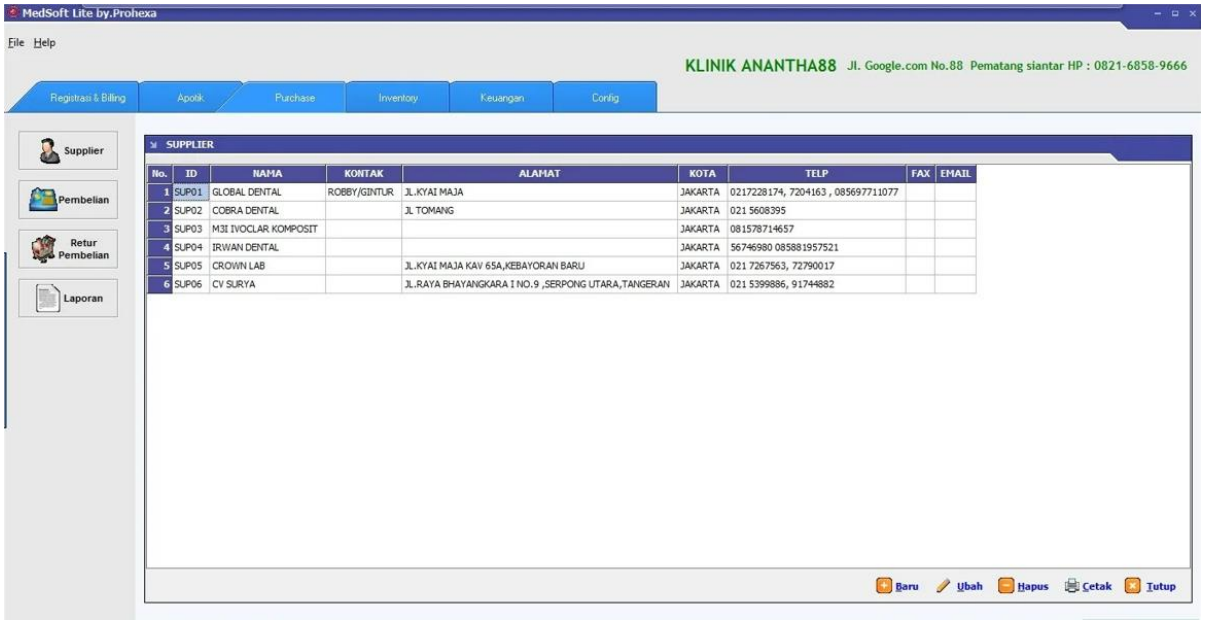


Рисунок 11 – Информация о пациентах

На вкладке «Пациенты» происходит вся работа (рисунок 12). Можно заполнить карточку пациента его диагнозов, оказания услуг, посмотреть придыдущие визиты.

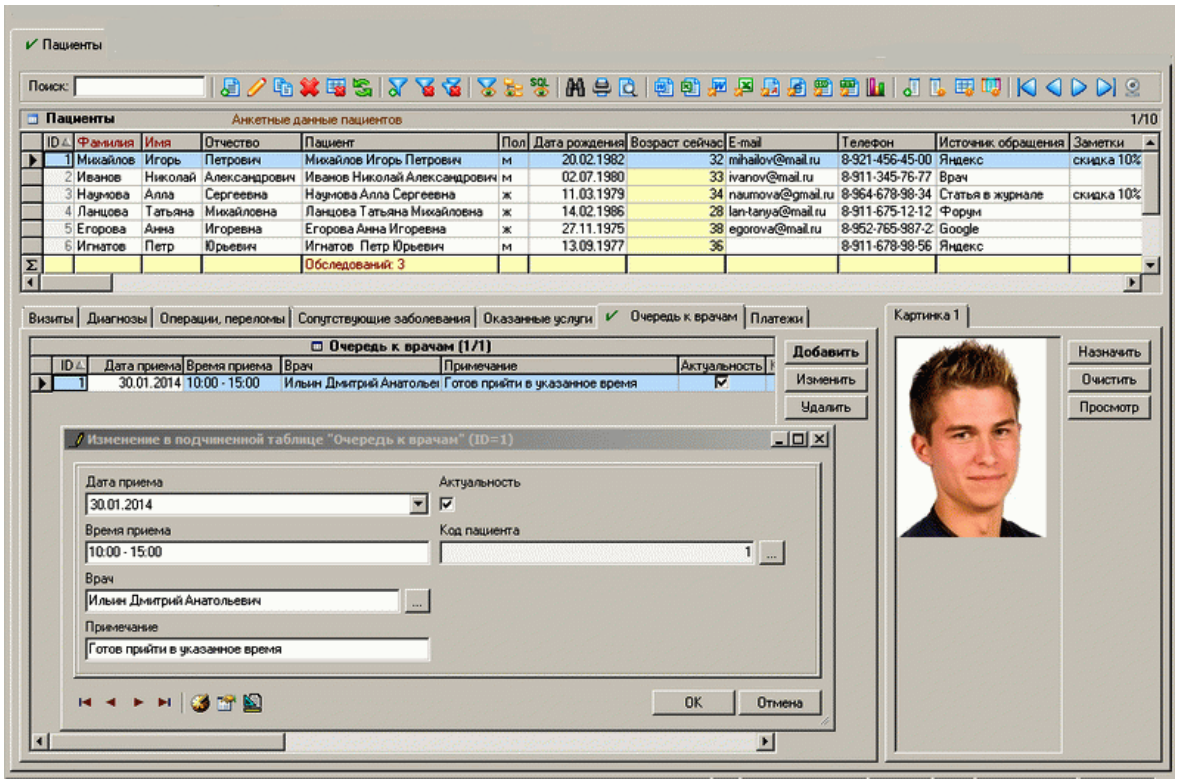


Рисунок 12 – Карточка пациента

Программа обладает следующими преимуществами:

– управление расписанием, включает модули для ведения электронных медкарт, учета пациентов;

– поддержка интеграции с федеральными системами (ЕГИСЗ, ЕМИАС);

– круглосуточная помощь пользователям, регулярные обновления.

Из минусов данной информационной системы можно выделить следующее:

– для работы необходима мощная серверная инфраструктура, что увеличивает затраты;

– не адаптироанный интерфейс на английском языке, таким образом усложняет работу врачей;

– постоянные сбои программы;

– нет возможности просматривать информацию самому пациенту о предстоящих визитах в больницу.

1.1.3 Обоснование актуальности разработки программы

В условиях роста нагрузки на медицинские учреждения и увеличения требований к эффективности работы врачей, ручное управление расписаниями персонала становится основной проблемой. Устаревшие методы планирования, на бумажных носителях или таблицах, приводят к ошибкам, конфликтам в графиках, переработкам сотрудников и задержкам при записи пациентов. Это очень негативно влияет как на качество оказания медицинских услуг.

Разрабатываемая система актуальна основываясь на следующих фактах:

– повышение доступности: Онлайн–доступ к расписаниям для пациентов упрощает возможность записи на приём, а для сотрудников – контроль изменений в режиме реального времени;

– снижение затрат: Моё решение исключает необходимость дорогостоящих коммерческих аналогов;

– цифровизация здравоохранения: Система соответствует глобальному тренду на внедрение ИТ–решений для повышения прозрачности и управляемости медицинских услуг.

1.1.4 Постановка задачи

Требуется реализовать приложение, которое будет соответствовать следующим функциональным требованиям:

– создание и редоктирование смен врача (возможность врача настравить индивидульное расписание);

– одобрение\отклонение записей (врачи подтверждают запись или отклоняют);

– конфигурирование смен (добавление новых временных окошек, изменений подтверждённых записей);

– интерфейс для пациентов (отдельный интерфейс для удобства записи на приём в нужную поликлинику и врачу);

– инетрфейс для врачей (индивидуальная учётная запись, отдельное управление настраиваемым расписанием).

На рисунках изображены диаграммы вариантов использования при входе с учетной записью врача (рисунок 13) и пациента (рисунок 14).

Разрабатываемое веб–приложение должно соответствовать следующим техническим требованиям:

– удобный и интуитивно понятный интерфейс как для пациентов, так и для врачей;

– расположение графических элементов должно предоставлять пользователю быстрое ориентирование в интерфейсе.

Минимальные требования:

Для компьютеров:

– операционная система: Windows 7 / macOS 10.12 / Linux Ubuntu 18.04 (и новее);

– оперативная память: не менее 2 ГБ ОЗУ;

– незанятное дисковое пространство: не менее 500 МБ ПЗУ;

– процессор: 2 – ядерный CPU 1.5 GHz (или аналогичный);

– разрешение экрана: 1280x720.

Для корректной работы требуется наличие браузера (Chrome, Firefox, Safari и др.) с поддержкой JavaScript и стабильного инернета (от 5 Мбит/с).

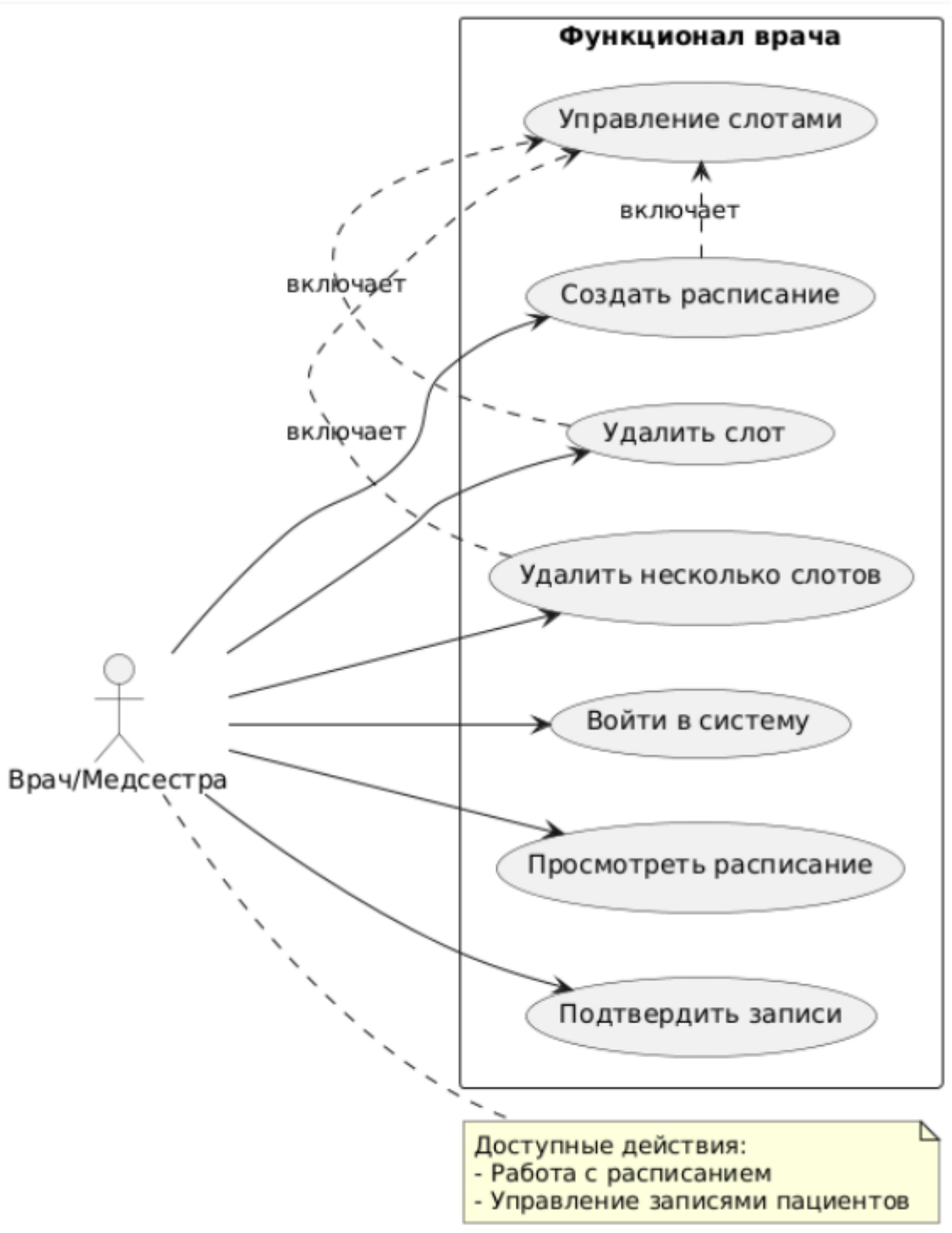


Рисунок 13 – Диаграмма вариантов использования приложения пользователя с учетной записью врача\медсестры

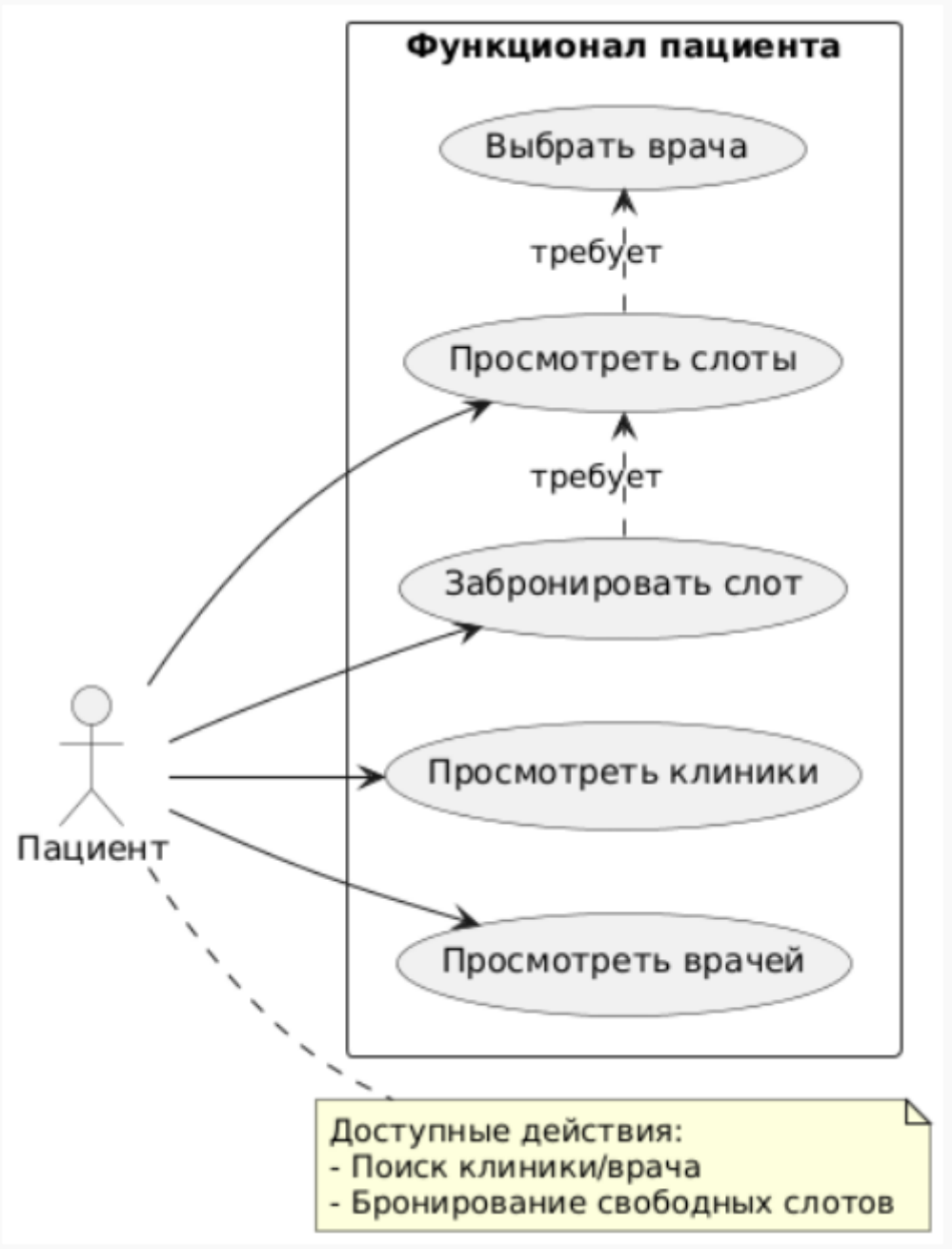


Рисунок 14 –Диаграмма вариантов использования приложения с правами обычного пользователя

1.2 Выбор технологий и средств разработки приложения

В данном разделе будут описаны технологии и средства разработки, использованные при разработке веб – приложения управление расписанием работы врачей в поликлинике .

Ниже приведен список выбранных технологий и средств разработки:

– Python;

– Flask;

– SQLite;

– PyCharm;

– CSS;

– HTML.

1.2.1 Выбор языка программирования

Python – высокоуровневый язык общего назначения, разработанный Гвидо ван Россумом и впервые представленный в 1991 году. Его ключевые принципы, сформулированные в документе «The Zen of Python», подчеркивают важность читаемости кода, простоты и явности синтаксиса. Динамическая типизация позволяет разработчикам не указывать типы переменных явно, что ускоряет написание кода, но требует внимательности к логике программы. Python поддерживает множество парадигм программирования: процедурный, объектно – ориентированный и функциональный подходы, что делает его универсальным инструментом для решения разнообразных задач.

Стандартная библиотека Python включает модули для работы с сетевыми протоколами, файловыми системами, криптографией, многопоточностью и многим другим. Например, модуль «asyncio» позволяет создавать асинхронные приложения, а «pandas» и «numpy» активно используются в data science. Для веб – разработки язык предлагает фреймворки Django и Flask, которые значительно упрощают создание серверной части приложений.

Java – язык программирования явного объявления типов и компиляции, что увеличивает время разработки. Преймущество Java заключается в том, что он обеспечивает высокую производительность за счет JIT – компиляции в байт – код.

C++ – данный язык позволяет работать на низком уровне (управление памятью, указатели), но сложен для новичков и менее гибок в веб – разработке[1].

После сравнения возможностей одних из популярных языков для реализации серверной и клиентской частей приложения Python выбран мною благодаря высокой скорости прототипирования, богатой экосистеме библиотек и

фреймворков, а также активному сообществу. Например, компании Google, Netflix и Instagram используют Python для backend – разработки и анализа данных[2]. Для моего проекта важно минимизировать время настройки инфраструктуры и сосредоточиться на логике приложения, что идеально соответствует философии Python.

1.2.2 Выбор веб – фреймворка

Flask – микрофреймворк, выпущенный в 2010 году как ответ на сложность Django. Он построен на двух основных компонентах:

– werkzeug – библиотека для обработки HTTP – запросов и маршрутизации;

– jinja2 – шаблонизатор, позволяющий генерировать HTML – страницы динамически.

Flask не включает встроенные ORM или аутентификацию, что делает его легковесным. Но разработчик может подключать только необходимые расширения, такие как:

– Flask – SQLAlchemy – интеграция с ORM;

– Flask – Login – управление сессиями пользователей;

– Flask – RESTful – создание API.

Веб – фреймворк Flask.

Django предоставляет «всё из коробки»: админ – панель, ORM, систему аутентификации. Это удобно для крупных проектов, но избыточно для небольших приложений, где важна гибкость. Например, в Django сложно изменить структуру базы данных после миграций, тогда как Flask позволяет свободно выбирать инструменты[3].

Flask выбран для проекта, так как он обеспечивает:

– минималистичную архитектуру, где каждый компонент контролируется разработчиком;

– легкую интеграцию с внешними сервисами (например, SQLite через SQLAlchemy) ;

– возможность масштабирования за счет модульности.

Пример использования:

```
from flask import Flask
app = Flask(__name__)
@app.route(«/»)
def home():
    return "Hello, World!"
if __name__ == «__main__»:
    app.run(debug=True)
```

Этот код создает базовое веб – приложение с одним маршрутом, демонстрируя простоту фреймворка Flask.

1.2.3 Выбор СУБД

SQLite – встраиваемая реляционная база данных, разработанная в 2000 году. Её главная особенность – отсутствие серверной части: все данные хранятся в одном файле на диске[4]. Это делает SQLite идеальной для использования:

– мобильных приложений (используется в Android и iOS) ;

– прототипирования и тестирования;

– небольших веб – приложений с низкой нагрузкой.

Принципы работы.

– ACID – транзакции: Гарантируют целостность данных даже при сбоях;

– динамическая типизация: Поля могут хранить значения любого типа, что упрощает схему;

– ограничения: Нет поддержки полномасштабных пользовательских прав доступа, что критично для многопользовательских систем.

Сравнение с PostgreSQL и MySQL.

– PostgreSQL: Поддерживает сложные запросы, хранимые процедуры и шардинг, но требует настройки сервера;

– MySQL: Балансирует между производительностью и простотой, но менее надежна в транзакциях.

SQLite выбрана из – за:

– нулевой конфигурации: база создается при первом подключении;

– совместимости с Python через модуль «sqlite3»;

– удобства для локальной разработки и тестирования.

Пример создания таблицы:

```
import sqlite3
conn = sqlite3.connect(«database.db»)
cursor = conn.cursor()
cursor.execute(««««
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY,
        name TEXT NOT NULL,
        email TEXT UNIQUE
    )
»»»»)
conn.commit()
conn.close()
```

1.2.4 Выбор среды разработки

PyCharm – IDE от JetBrains, выпущенная в 2010 году. Она предоставляет:

– умное автодополнение: Анализирует контекст и предлагает релевантные методы;

– интеграцию с Flask: Шаблоны для быстрого создания роутов и обработчиков;

– отладчик: Позволяет пошагово выполнять код, устанавливать точки останова;

– инструменты для работы с БД: Визуальный редактор SQLite, генерация миграций.

Сравнение с VS Code:

VS Code – легковесный редактор с плагинами (например, Python Extension Pack). Однако PyCharm предлагает более глубокую интеграцию с фреймворками и готовые конфигурации для проектов.

Мною выбран среда разработки PyCharm, так как она ускоряет разработку благодаря:

– встроенному терминалу и управлению виртуальными окружениями;

– поддержке рефакторинга (например, переименование переменных во всем проекте) ;

– интеграции с системами контроля версий (Git, Mercurial).

1.2.5 Выбор технологий фронтеда

HTML и CSS:

– HTML (HyperText Markup Language): Основа веб – страниц, определяющая структуру контента через теги («<header>«, «<form>«, « «). Современный стандарт HTML5 добавил семантические теги («<article>«, «<section>«), улучшив SEO и доступность[5];

– CSS (Cascading Style Sheets): Управляет внешним видом элементов: позиционирование, анимации, адаптивность через медиа – запросы.

Пример кода:

```
html
<!DOCTYPE html>
<html>
<head>
    <title>Проект</title>
    <style>
        .container {
            width: 80%;
            margin: 0 auto;
            font – family: Arial;
        }
        .button {
            background: #4CAF50;
            color: white;
            padding: 10px 20px;
            border: none;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Добро пожаловать</h1>
        <button class="button">Нажми меня</button>
    </div>
</body>
</html>
```

Сравнение с JavaScript – фреймворками.

React или Vue.js позволяют создавать динамические интерфейсы, но увеличивают сложность проекта. Для статических страниц или простых форм HTML/CSS достаточно.

HTML и CSS выбран для моего проекта, поскольку интегрируются с Flask через Jinja2:

```
from flask import render_template
@app.route(«/profile/<username>«)
```

```
def profile(username):
    return render_template(«profile.html», username=username)
```

Шаблон «profile.html» использует переменную «username», демонстрируя связь бэкенда и фронтенда.

1.2.6. Обмен данными JSON

JSON (JavaScript Object Notation).

JSON – текстовый формат, созданный в 2001 году. Его синтаксис включает:

– объекты: Пары «ключ: значение» в фигурных скобках;

– массивы: Упорядоченные списки в квадратных скобках.

Пример структуры:

```
json
{
    "user": {
        "id": 1,
        "name": "Иван",
        "roles": ["admin", "editor"]
    }
}
```

Использование в Python.

Модуль «json» сериализует данные:

```
import json
data = {"name": "Иван", "age": 30}
json_str = json.dumps(data)  # Преобразование в строку
parsed_data = json.loads(json_str)
```

Интеграция с Flask.

Расширение «Flask – JSON» автоматически преобразует ответы сервера:

```
from flask import Flask, jsonify
app = Flask(__name__)
@app.route(«/api/data»)
def get_data():
    return jsonify({"status": "success", "data": [1, 2, 3]})
```

JSON обеспечивает:

– совместимость между разными языками и платформами;

– удобство отладки благодаря читаемости;

– поддержку в REST API, что критично для взаимодействия клиента и сервера.

В данной бакалаврской работе JSON – формат будет использоваться для обмена данными между клиентской и серверной частями разрабатываемого приложения.

1.3 Проектирование информационной системы

1.3.1 Паттерн MVC

MVC (Model – View – Controller) – это фундаментальный архитектурный паттерн, который на протяжении долгого времени остается одним из наиболее востребованных подходов в разработке программного обеспечения. Его универсальность и гибкость позволяют эффективно решать задачи, связанные с созданием сложных клиент – серверных приложений, веб – платформ и даже мобильных решений. В рамках моего дипломного проекта выбор MVC обусловлен необходимостью обеспечения структурированности кода, упрощения командной работы и повышения уровня поддерживаемости системы.

Паттерн MVC был впервые представлен в 1970 – х годах как часть парадигмы разработки пользовательских интерфейсов в языке Smalltalk. Его создатели, Трюгве Реенскауг и Адель Голдберг, стремились разделить логику приложения на независимые компоненты, чтобы минимизировать 耦合 ость (взаимозависимость) между частями системы[6]. С тех пор MVC эволюционировал, адаптируясь к современным технологиям, таким как веб – фреймворки (например, Django, Ruby on Rails, Flask), что подтверждает его актуальность и универсальность.

Ключевые компоненты MVC:

– модель (Model) представляет собой ядро приложения, отвечающее за бизнес – логику, данные и правила их обработки. Она инкапсулирует взаимодействие с базами данных, валидацию входных параметров и выполнение алгоритмов. В моём дипломном проекте, связанном с системойемой записи к врачу, модель включает сущности «Пациент», «Врач», «Расписание», а также методы для создания слотов или проверки доступности времени;

– представление (View) этот компонент отвечает за визуализацию данных и взаимодействие с конечным пользователем. В веб – приложениях представление часто реализуется через HTML – шаблоны с динамической подгрузкой информации. Например, страница с расписанием врача отображает слоты, полученные от модели, используя шаблонизаторы вроде Python[7];

– контроллер (Controller) выступает посредником между моделью и представлением. Он обрабатывает пользовательские запросы (например, нажатие кнопки «Записаться»), валидирует входные данные и делегирует выполнение операций модели. После обработки контроллер обновляет представление, обеспечивая актуальность информации.

Преимущества MVC:

– четкое разделение ответственности. MVC позволяет распределить задачи между разработчиками: один может работать над моделью данных, другой – над интерфейсом, а третий – над бизнес – логикой. Это особенно важно в командных проектах, где требуется параллельная работа;

– упрощение тестирования. Независимость компонентов упрощает модульное тестирование. Например, модель можно проверить на корректность работы с базой данных, не затрагивая представление;

– масштабируемость. Добавление новых функций (например, интеграция с мессенджерами для уведомлений) не требует переписывания всей системы – достаточно расширить существующие компоненты;

– адаптивность к изменениям. Если в ходе разработки потребуется изменить интерфейс (перейти с десктопной версии на мобильную), представление можно модифицировать без влияния на модель или контроллер.

Логическая схема MVC – приложения представлена на рисунке 15.

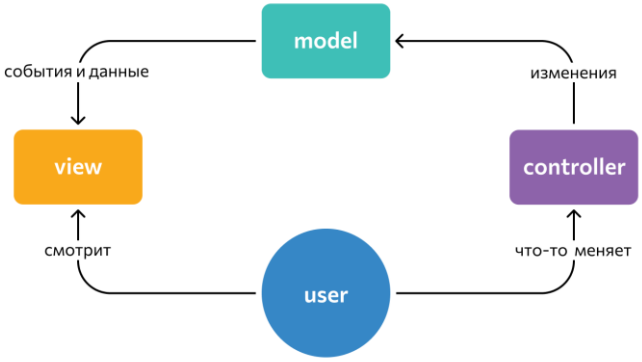


Рисунок 15 – Логическая схема MVC – приложения

Сравнение с альтернативными паттернами.

– MVVM (Model – View – ViewModel). Актуален для приложений с богатым интерфейсом (например, SPA на Angular). ViewModel здесь синхронизирует данные между View и Model, что усложняет архитектуру, но повышает гибкость.

– MVP (Model – View – Presenter). В отличие от MVC, Presenter берет на себя больше ответственности за логику отображения, что может быть полезно для проектов с активным взаимодействием с пользователем.

MVC остается «золотым стандартом» для проектов, где требуется баланс между простотой и функциональностью.

В качестве шаблона проектирования паттерн MVC выбран в мой дипломный проект не только потому что соответствует современным стандартам разработки. Он обеспечивает прозрачность кода, облегчает внедрение новых функций и позволяет создать систему, готовую к дальнейшему расширению. MVC эффективно управляет запросами к базе данных и обновлением данных, что важно для клиент – серверных приложений.

1.3.2 Архитектура приложения

На рисунке 16 представлена архитектура приложения.

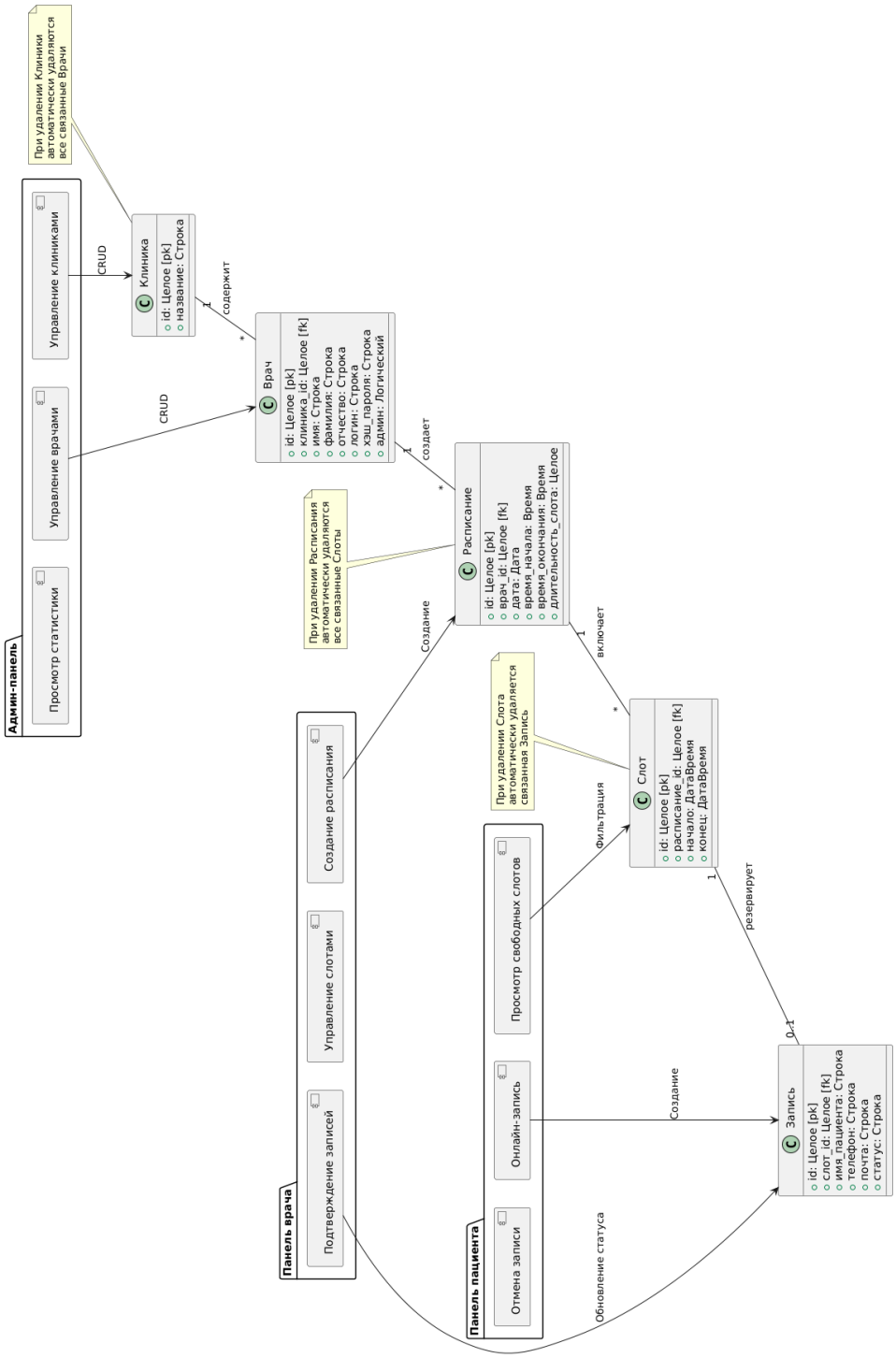


Рисунок 16 – Архитектура приложения

Приложение реализовано с использованием классической архитектуры MVC (Model – View – Controller), которая обеспечивает четкое разделение компонентов для упрощения разработки и поддержки. Модели (Model) отвечают за структуру данных и бизнес – логику: например, сущности вроде врачей, клиник и записей пациентов хранятся в базе данных SQLite через ORM SQLAlchemy, что

позволяет автоматизировать взаимодействие с таблицами и гарантирует целостность данных. Представления (View), реализованные через HTML – шаблоны, формируют пользовательский интерфейс – от форм ввода до таблиц с расписаниями, используя фильтры Jinja2 для удобного отображения времени. Контроллеры (Controller), выраженные в виде Flask – роутов, обрабатывают запросы пользователей: проверяют авторизацию, валидируют введенные данные и управляют потоком информации между моделями и представлениями, обеспечивая гибкость и безопасность процессов.

Взаимодействие пользователя с системой строится по принципу диалога: например, когда врач вводит данные для создания расписания, контроллер сначала проверяет корректность даты и времени, а затем передает информацию модели, которая генерирует временные слоты и сохраняет их в базе. Если данные не проходят валидацию (например, указано время из прошлого), система мгновенно уведомляет пользователя об ошибке через flash – сообщения, что делает работу интуитивно понятной. Важную роль играет безопасность: пароли врачей хранятся в виде хешей, а доступ к административным функциям ограничен декоратором @admin_required, который проверяет права через сессии. Это позволяет избежать несанкционированных действий и защищает конфиденциальность данных пациентов, чьи записи управляются через отдельную сущность Appointment, связанную со слотами.

Работа с данными организована через SQLAlchemy, что упрощает масштабирование приложения и интеграцию с другими СУБД, а автоматическое создание таблиц при запуске (db.create_all()) снижает риск ошибок при развертывании. Для врачей предусмотрены функции массового подтверждения или удаления записей, что экономит время при работе с расписанием, а пациенты могут легко выбирать доступные слоты через интерактивный интерфейс, где занятые ячейки подсвечиваются. Администраторы, в свою очередь, управляют клиниками и учетными записями врачей через отдельные разделы, что централизует контроль над системой. Все эти процессы объединены в единый

цикл, где каждое действие пользователя triggers цепочку проверок и обновлений, обеспечивая стабильность и отзывчивость приложения.

Сценарии использования охватывают широкий спектр задач: врач может залогиниться, создать расписание на неделю и отслеживать записи пациентов, пациент – выбрать удобное время и оставить контактные данные, а администратор – добавлять новые клиники или назначать права доступа. Такая архитектура не только упрощает повседневные операции, но и позволяет легко расширять функционал – например, добавить уведомления по email или интеграцию с календарями. Важно отметить, что разделение на MVC способствует поддержке кода: изменения в интерфейсе не затрагивают бизнес – логику, а обновления моделей можно проводить без переписывания контроллеров. В итоге приложение сочетает в себе надежность, удобство и адаптивность, оставаясь простым для понимания как разработчиками, так и конечными пользователями.

Вся работа с приложением будет осуществляться в виде диалога с пользователем. Диалоговая система состоит из трех ключевых компонентов (рисунок 17):

Для врача:

– вход через /doctor/login – проверка учетных данных – переход в личный кабинет;

– создание расписания: ввод даты, времени – генерация слотов – отображение в doctor_schedule.html;

– подтверждение/удаление записей: массовые операции через confirm – multiple и delete – multiple – slots.

Для пациента:

– выбор клиники и врача – просмотр доступных слотов (/patient/schedule/<doctor_id>);

– бронирование: заполнение формы – проверка доступности слота – создание Appointment.

Для администратора:

– управление клиниками и врачами через /admin/clinics и /admin/doctors;

– добавление/удаление сущностей с проверкой прав (@admin_required).

Система управления базами данных (СУБД) представляет собой набор программных инструментов, которые позволяют создавать, управлять и администрировать базы данных. Другими словами, СУБД предоставляет возможности для организации, контроля и администрирования данных в базах данных.

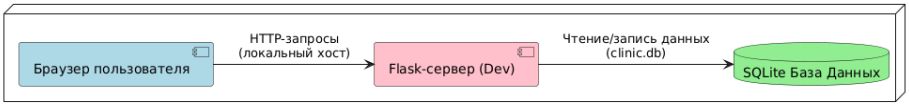


Рисунок 17 – Диаграмма развертывания приложения

На рисунке 18 представлено взаимодействие компонентов приложения в виде диаграммы последовательности.

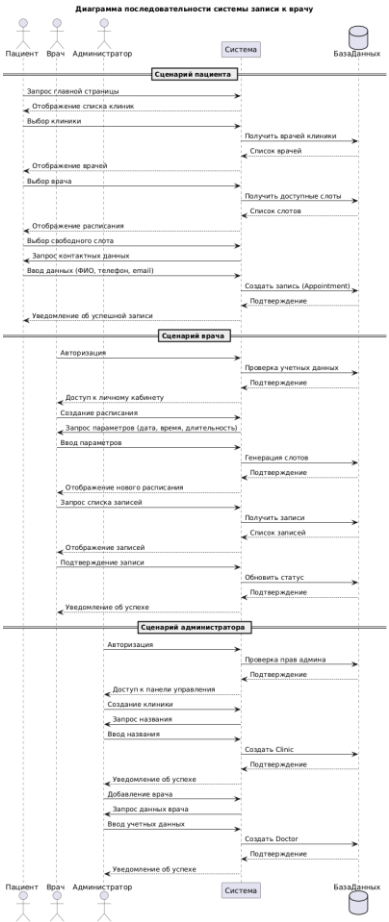


Рисунок 18 – Диаграмма последовательности

Диаграмма классов, представляющая модель данных, изображена на рисунке 19.

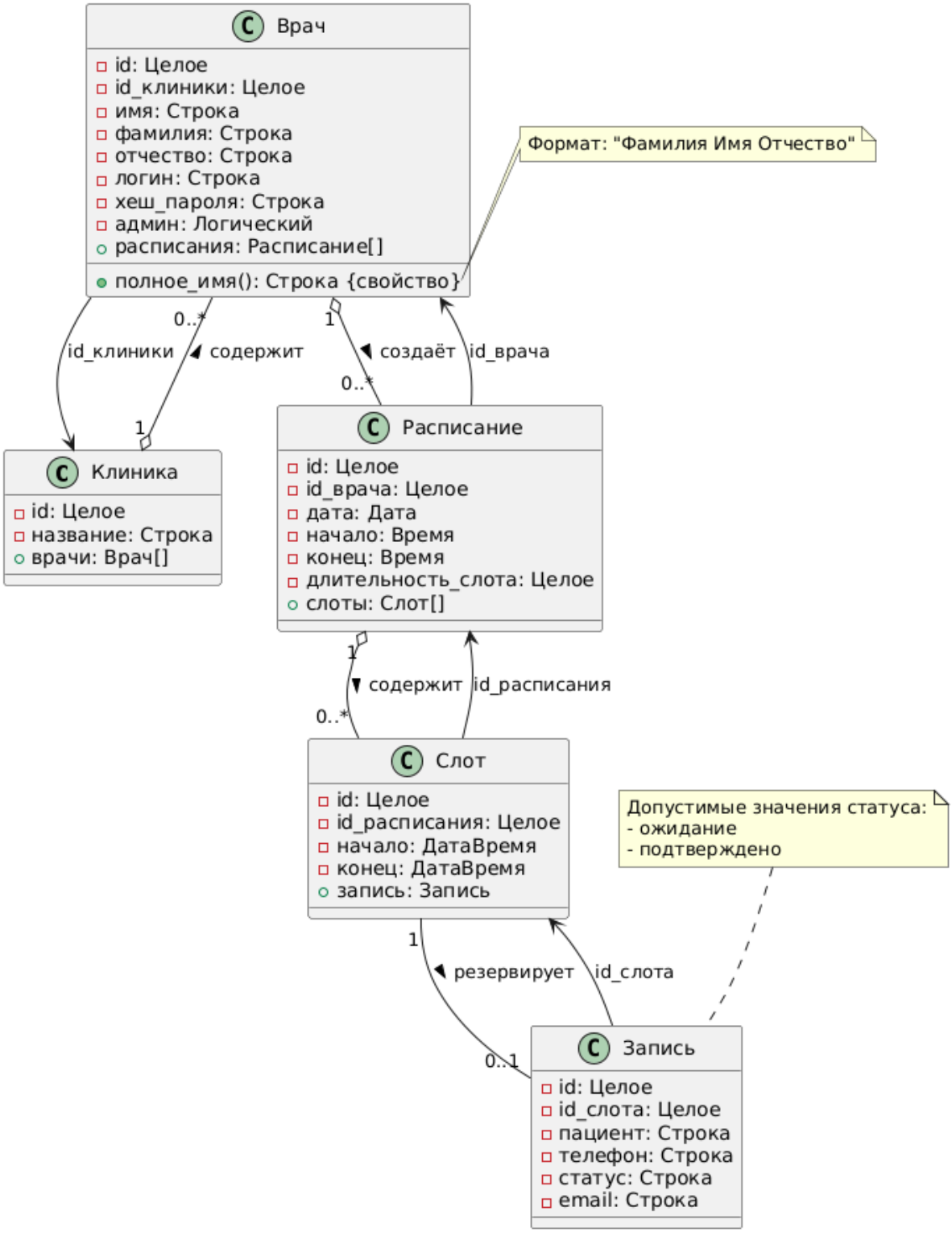


Рисунок 19 – Диаграмма классов

На рисунке 20 изображена диаграмма деятельности.

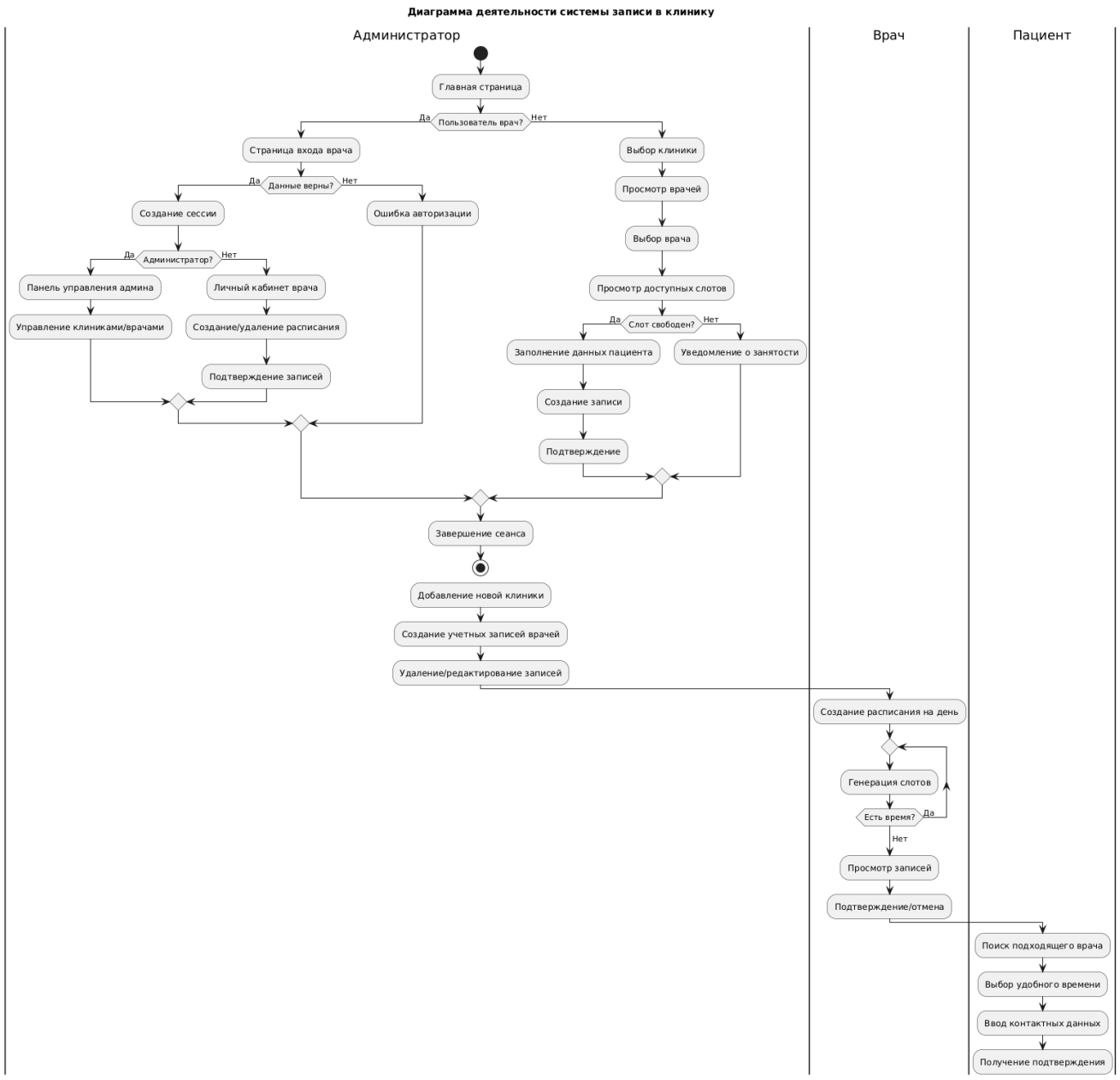


Рисунок 20 – Диаграмма деятельности

1.3.3 Проектирование базы данных

Для хранения информации необходимо реализовать базу данных со следующими таблицами:

– clinic – хранит данные медицинских учреждений: уникальный идентификатор и название (таблица 1);

– doctor – содержит информацию о врачах (таблица 2);

– schedule – фиксирует рабочие часы врачей (таблица 3);

– slot – создает временные интервалы для записи пациентов на основе расписания (таблица 4);

– appointment – управляет бронированием слотов (таблица 5);

– patient – хранит основную информацию о пациентах (таблица 6);

– medical_records – содержит историю лечения (таблица 7);

– specializations – определяет специализации врачей (таблица 8).

Таблица 1 – Атрибуты таблицы clinic

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Уникальный идентификатор	Первичный ключ (PK)	1
name	VARCHAR(100)	Название клиники	Обязательный атрибут	"Главная клиника"

Таблица 2 – Атрибуты таблицы doctor

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Уникальный идентификатор	PK	2
clinic_id	INT	ID клиники	Внешний ключ (FK)	1
first_name	VARCHAR(50)	Имя врача	Обязательный атрибут	"Иван"
last_name	VARCHAR(50)	Фамилия врача	Обязательный атрибут	"Петров"
patronymic	VARCHAR(50)	Отчество	Опциональный атрибут	"Сергеевич"
username	VARCHAR(80)	Логин	Уникальный, обязательный	"doctor_ivan"
password_hash	VARCHAR(120)	Хэш пароля	Обязательный атрибут	"sha256$..."
is_admin	BOOLEAN	Статус администратора	Флаг	TRUE

Таблица 3 – Атрибуты таблицы schedule

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Уникальный идентификатор	PK	5
doctor_id	INT	ID врача	FK	2
date	DATE	Дата расписания	Обязательный атрибут	2023 – 10 – 10
start_time	TIME	Время начала	Обязательный атрибут	09:00:00
end_time	TIME	Время окончания	Обязательный атрибут	17:00:00
slot_duration	INT	Длительность слота (мин)	Обязательный атрибут	30

Таблица 4 – Атрибуты таблицы slot

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Уникальный идентификатор	PK	10
schedule_id	INT	ID расписания	FK	5
start_time	DATETIME	Начало слота	Обязательный атрибут	2023 – 10 – 10 09:00:00
end_time	DATETIME	Окончание слота	Обязательный атрибут	2023 – 10 – 10 09:30:00

Таблица 5 – Атрибуты таблицы appointment

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Уникальный идентификатор	PK	15
slot_id	INT	ID слота	FK	10
patient_name	VARCHAR(80)	Имя пациента	Обязательный атрибут	"Анна Сидорова"

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
patient_phone	VARCHAR(20)	Телефон пациента	Обязательный атрибут	"+7 999 123 – 45 – 67"
patient_email	VARCHAR(120)	Email пациента	Обязательный атрибут	"anna@mail.ru"
status	VARCHAR(20)	Статус записи	Обязательный атрибут	"pending"

Таблица 6 – Атрибуты таблицы patient

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Уникальный идентификатор	PK	20
full_name	VARCHAR(100)	Полное имя	Обязательный атрибут	"Анна Сидорова"
phone	VARCHAR(20)	Телефон	Обязательный атрибут	"+7 999 123 – 45 – 67"
email	VARCHAR(120)	Email	Обязательный атрибут	"anna@mail.ru"
registration_date	DATE	Дата регистрации	Обязательный атрибут	2023 – 10 – 01

Таблица 7 – Атрибуты таблицы medical_records

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Уникальный идентификатор	PK	25
patient_id	INT	ID пациента	FK	20
doctor_id	INT	ID врача	FK	2
diagnosis	TEXT	Диагноз	Обязательный атрибут	"ОРВИ"
treatment	TEXT	Лечение	Обязательный атрибут	"Постельный режим"
record_date	DATE	Дата записи	Обязательный	2023–10-05

Таблица 8 – Атрибуты таблицы specializations

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Уникальный идентификатор	PK	40
name	VARCHAR(100)	Название специализации	Обязательный атрибут	"Терапевт"

Физическая модель базы данных приведена на рисунке 21.

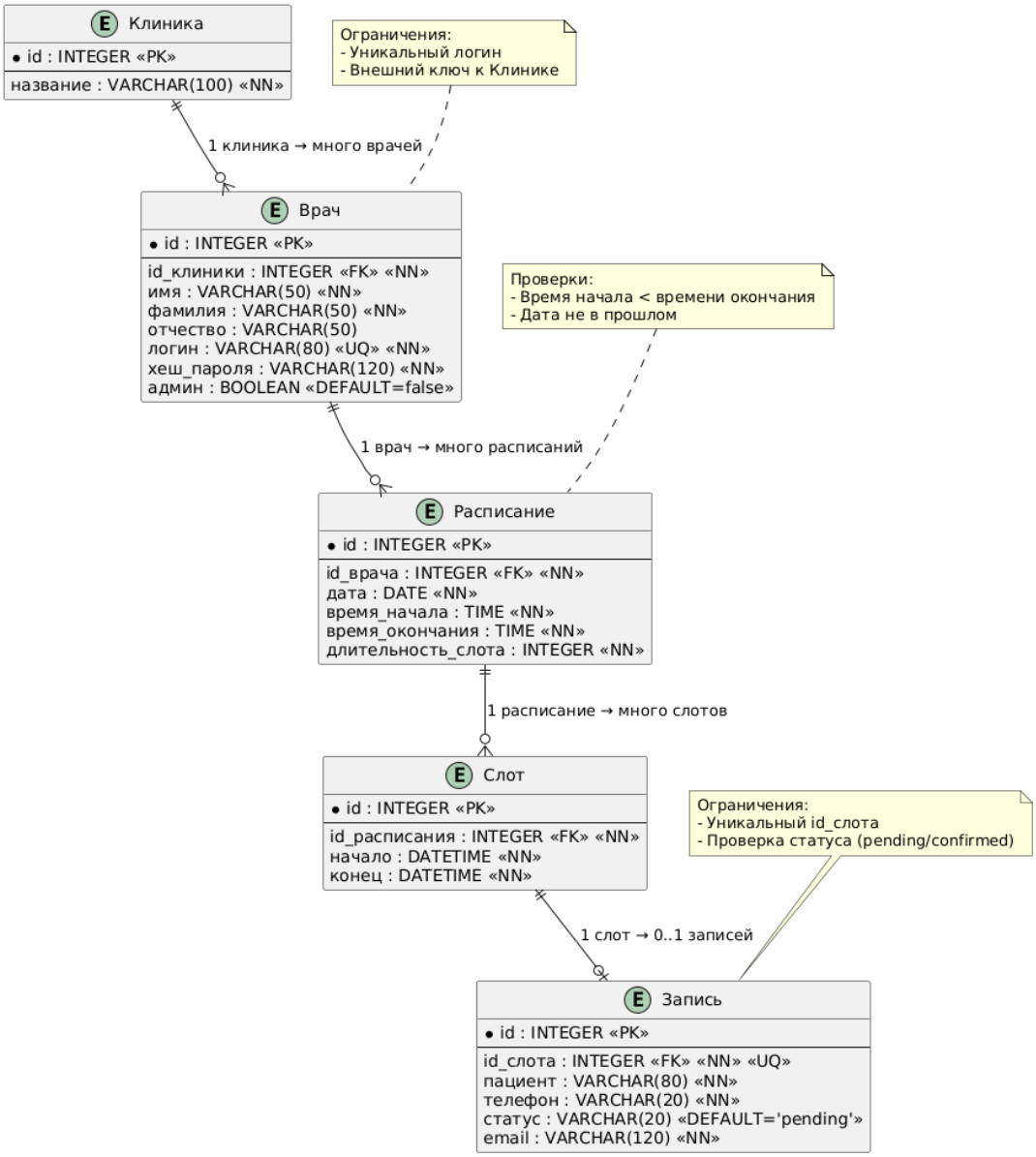


Рисунок 21 – Физическая модель БД

На рисунке 22 изображена логическая схема базы данных.

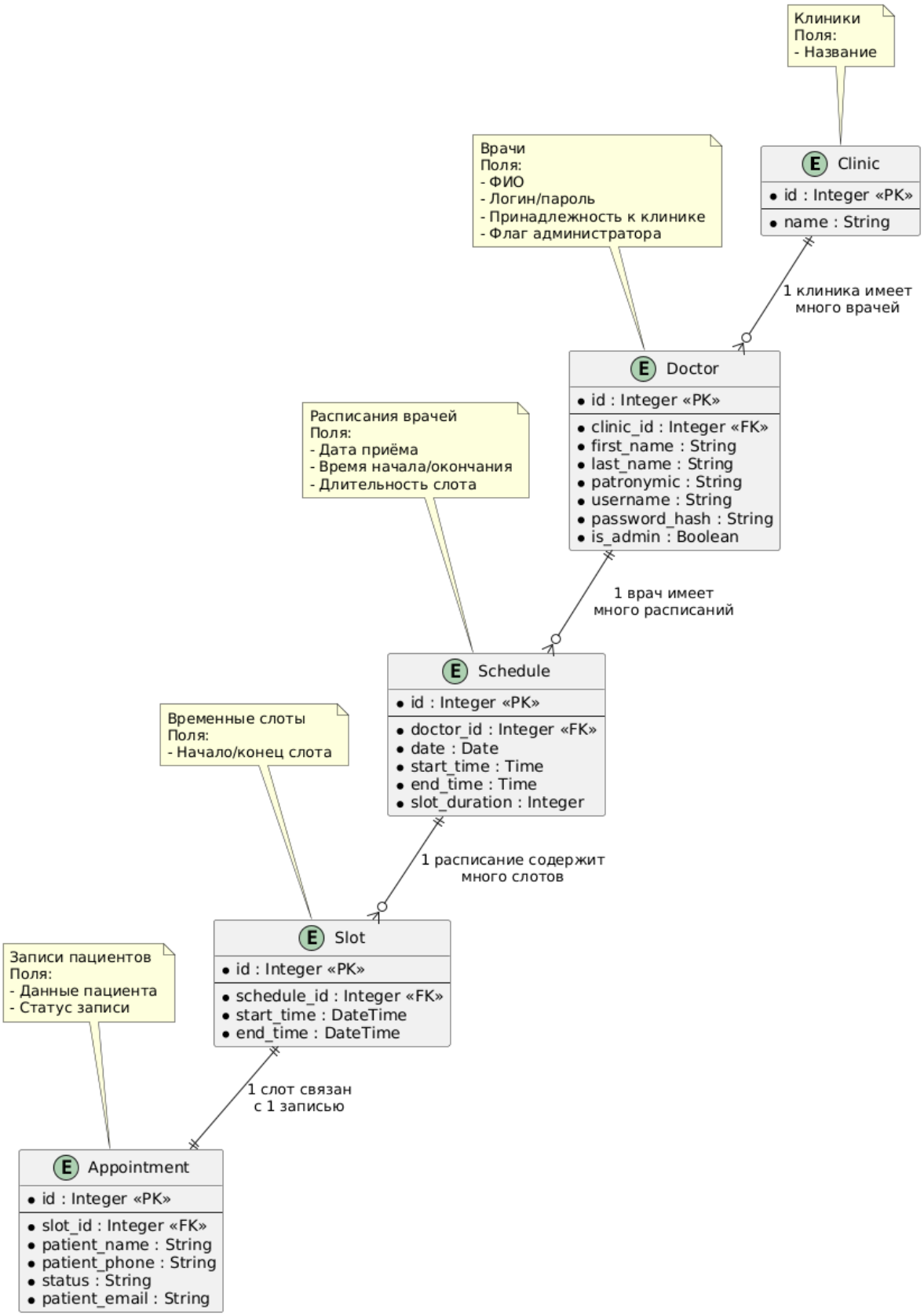


Рисунок 22 – Логическая схема БД

1.4 Разработка приложения

1.4.1 Описание программы

Основная часть программы начинается с импорта необходимых модулей Flask и связанных библиотек. Например, модуль «flask» предоставляет базовые инструменты для создания веб – приложения, а «flask_sqlalchemy» интегрирует ORM для работы с базой данных:

```
from flask import Flask, render_template, request, redirect,
url_for, session, abort
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash,
check_password_hash
```

Инициализация приложения и настройка базы данных выполняются через создание экземпляра «Flask» и конфигурацию SQLAlchemy. Секретный ключ используется для подписи сессий, а строка подключения указывает на SQLite – файл:

```
app = Flask(__name__)
app.config[«SECRET_KEY»] = «supersecretkey»
app.config[«SQLALCHEMY_DATABASE_URI»] = «sqlite:///clinic.db»
db = SQLAlchemy(app)
```

Модель «Clinic» описывает медицинские учреждения. Поле «doctors» создает отношение «один – ко – многим» с врачами, а параметр «cascade» гарантирует удаление связанных врачей при удалении клиники:

```
class Clinic(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    doctors = db.relationship(«Doctor», backref=«clinic»,
cascade=«all, delete – orphan»)
```

Модель «Doctor» хранит данные врачей, включая хеш пароля. Метод «full_name» объединяет ФИО в одну строку, а «is_admin» определяет права доступа:

```
class Doctor(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    clinic_id = db.Column(db.Integer,
db.ForeignKey(«clinic.id»), nullable=False)
    first_name = db.Column(db.String(50), nullable=False)
    last_name = db.Column(db.String(50), nullable=False)
    patronymic = db.Column(db.String(50))
    username = db.Column(db.String(80), unique=True,
nullable=False)
```

```
    password_hash = db.Column(db.String(120), nullable=False)
    is_admin = db.Column(db.Boolean, default=False)

    @property
    def full_name(self):
        return f"{self.last_name} {self.first_name}
{self.patronymic or ««}".strip()
```

Функция «create_slots» разбивает рабочий день врача на интервалы. Например, если врач принимает с 9:00 до 12:00 с интервалом в 30 минут, создаются слоты 9:00 – 9:30, 9:30 – 10:00 и т.д.:

```
def create_slots(schedule):
    start = datetime.combine(schedule.date,
schedule.start_time)
    end = datetime.combine(schedule.date, schedule.end_time)
    current = start
    slot_duration = timedelta(minutes=schedule.slot_duration)

    while current + slot_duration <= end:
        slot_end = current + slot_duration
        slot = Slot(
            schedule_id=schedule.id,
            start_time=current,
            end_time=slot_end
        )
        db.session.add(slot)
        current += slot_duration
    db.session.commit()
```

Маршрут «/doctor/login» обрабатывает авторизацию. При успешной проверке пароля в сессию записывается «doctor_id» и флаг администратора:

```
@app.route(«/doctor/login», methods=[«GET», «POST»])
def doctor_login():
    if request.method == «POST»:
        username = request.form[«username»]
        password = request.form[«password»]
        doctor =
Doctor.query.filter_by(username=username).first()
        if doctor and check_password_hash(doctor.password_hash,
password):
            session[«doctor_id»] = doctor.id
            session[«is_admin»] = doctor.is_admin
            return redirect(url_for(«admin_dashboard»)) if
doctor.is_admin else redirect(url_for(«doctor_schedule»))
        return «Неверные данные»
    return render_template(«doctor_login.html»)
```

Декоратор «admin_required» проверяет наличие прав администратора. Если пользователь не авторизован как администратор, доступ запрещается через «abort(403)»:

```
def admin_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if not session.get(«is_admin»):
            abort(403)
        return f(*args, **kwargs)
    return decorated_function
```

Маршрут «/create_admin» инициализирует тестового администратора. Хеширование пароля через «generate_password_hash» защищает чувствительные данные:

```
@app.route(«/create_admin»)
def create_admin():
    clinic = Clinic(name="Главная клиника")
    db.session.add(clinic)
    db.session.commit()

    admin = Doctor(
        first_name="Админ",
        last_name="Администов",
        patronymic="Администович",
        username="admin",
        password_hash=generate_password_hash("admin123"),
        clinic_id=clinic.id,
        is_admin=True
    )

    db.session.add(admin)
    db.session.commit()
    return "Администратор создан!"
```

Модель «Appointment» связывает слоты с данными пациентов. Поле «status» отслеживает состояние записи (например, «ожидает» или «подтверждено»):

```
class Appointment(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    slot_id = db.Column(db.Integer, db.ForeignKey(«slot.id»),
nullable=False)
    patient_name = db.Column(db.String(80), nullable=False)
    patient_phone = db.Column(db.String(20), nullable=False)
    status = db.Column(db.String(20), default=«pending»)
    patient_email = db.Column(db.String(120), nullable=False)
    slot = db.relationship(«Slot»,
back_populates=«appointment»)
```

Фильтр «datetimeformat» форматирует дату в шаблонах Jinja2. Например, дата «2023 – 10 – 05 14:30» преобразуется в «05.10.2023 14:30»:

```
@app.template_filter(«datetimeformat»)
def datetimeformat(value, format=«%d.%m.%Y %H:%M»):
    return value.strftime(format)
```

Маршрут «/doctor/schedule» обрабатывает создание расписаний. Проверка «date < datetime.now().date()» предотвращает создание слотов в прошлом:

```
@app.route(«/doctor/schedule», methods=[«GET», «POST»])
def doctor_schedule():
    if «doctor_id» not in session:
        return redirect(url_for(«doctor_login»))

    doctor_id = session[«doctor_id»]
    now = datetime.now()
    if request.method == «POST»:
        # ... валидация и сохранение расписания ...
    selected_date = request.args.get(«date»)
    # ... фильтрация слотов по дате ...
    return render_template(«doctor_schedule.html»,
schedules=full_schedule)
```

Массовое удаление слотов реализовано через обработку списка «slot_ids». Запрос «Slot.id.in_(slot_ids)» выбирает только указанные слоты, принадлежащие текущему врачу:

```
@app.route(«/delete – multiple – slots», methods=[«POST»])
def delete_multiple_slots():
    slot_ids = request.form.getlist(«slot_ids»)
    slots = Slot.query.join(Schedule).filter(
        Slot.id.in_(slot_ids),
        Schedule.doctor_id == session[«doctor_id»],
        Slot.start_time > datetime.now()
    ).all()
```

Для пациентов слоты отображаются с учетом их статуса. Например, слот с «appointment.status == «pending»« помечается как ожидающий подтверждения:

```
@app.route(«/patient/schedule/<int:doctor_id>«)
def patient_schedule(doctor_id):
    doctor = Doctor.query.get_or_404(doctor_id)
    slots = Slot.query.join(Schedule).filter(
        Schedule.doctor_id == doctor.id,
        Schedule.date >= datetime.now().date()
    ).all()
   return render_template(«patient_schedule.html»,
doctor=doctor, slots=available_slots)
```

Маршрут «/patient/book» обрабатывает запись пациента. Проверка «if slot.appointment» предотвращает двойное бронирование:

```
@app.route(«/patient/book», methods=[«POST»])
def patient_book():
    slot_id = int(request.form[«slot_id»])
    slot = Slot.query.get(slot_id)
    if slot.appointment:
        return "Этот слот уже занят!"
```

Административные маршруты, такие как «/admin/add_doctor», используют декоратор «@admin_required». Параметр «is_admin» в форме определяет, будет ли врач администратором:

```
@app.route(«/admin/add_doctor», methods=[«GET», «POST»])
@admin_required
def add_doctor():
    if request.method == «POST»:
        doctor = Doctor(
            username=request.form[«username»],

password_hash=generate_password_hash(request.form[«password»]),
            is_admin=«is_admin» in request.form,
        )
        db.session.add(doctor)
        db.session.commit()
```

Контекстный процессор «inject_datetime» делает объект «datetime» доступным во всех шаблонах, упрощая работу с датами:

```
@app.context_processor
def inject_datetime():
    return {«datetime»: datetime}
```

Код основного модуля реализует полноценную систему управления расписанием с разделением прав доступа, валидацией данных и интеграцией с базой данных, обеспечивая безопасность и удобство для всех типов пользователей.

Веб – приложение для управления медицинскими расписаниями также опирается на систему шаблонов Flask, обеспечивающих единый интерфейс для врачей, пациентов и администраторов. Основной дизайн наследуется из базового шаблона «base.html», что гарантирует консистентность навигации и стилей. Например, форма авторизации врачей в «doctor_login.html» использует адаптивную верстку Bootstrap с крупными полями ввода для улучшения юзабилити на мобильных устройствах:

```
<input type="password" name="password" class="form - control form - control - lg" required>
```

После успешного входа врач попадает в интерфейс управления расписанием («doctor_schedule.html»), где может создавать временные слоты с валидацией длительности. Форма требует указания даты, времени начала/окончания и

интервала между записями, при этом длительность слота ограничена шагом в 15 минут для стандартизации:

```
<input type="number" name="slot_duration" min="15" step="15" required>
```

Созданные слоты отображаются в виде карточек с цветовой индикацией статуса. Свободные слоты подсвечиваются синей рамкой, ожидающие подтверждения – желтой, а занятые – серой. Для массовых операций реализован механизм выбора чекбоксов и обработки через JavaScript, который динамически добавляет выбранные ID в скрытые поля форм:

```
checkboxes.forEach(checkbox => {
    const input = document.createElement(«input»);
    input.type = «hidden»;
    input.name = «slot_ids»;
    input.value = checkbox.value;
    form.appendChild(input);
});
```

Пациенты, посещающие главную страницу («home.html»), видят анимированные карточки клиник с эффектом плавного появления. Каждая карточка содержит кнопку перехода к списку врачей, где параметр «clinic_id» передается через URL – строку:

```
<a href="{{ url_for(«list_doctors») }}?clinic_id={{ clinic.id }}" class="btn btn - cta mt - 3">
    Выбрать врача
</a>
```

В разделе «doctors.html» врачи представлены в виде сетки карточек с аватарками – заглушками и кнопками просмотра расписания. Анимация при скролле («animate – on – scroll») создает эффект постепенного появления элементов при прокрутке страницы.

Когда пациент выбирает врача, система отображает доступные слоты в «patient_schedule.html», используя модальное окно Bootstrap для сбора контактных данных. JavaScript – обработчик передает ID выбранного слота в скрытое поле формы:

```
document.getElementById(«bookingModal»).addEventListener(«show.bs.modal», function(e) {
    const slotId = e.relatedTarget.getAttribute(«data - slot - id»);
    document.getElementById(«slotId»).value = slotId;
});
```

Административные функции сосредоточены в шаблонах «manage_clinics.html» и «manage_doctors.html». В первом реализовано удаление клиник через выпадающее меню с подтверждением действия. Каждая карточка клиники отображает количество привязанных врачей через фильтр «length»:

```
<p class="text - muted small mb - 0">Врачей: {{
clinic.doctors|length }}</p>
```

Во втором шаблоне администраторы видят переключатель статуса врача с disabled – состоянием, предотвращающим случайные изменения. Информация о принадлежности к клинике извлекается через отношение «doctor.clinic.name», демонстрируя мощь SQLAlchemy ORM.

Формы добавления новых сущностей («add_clinic.html», «add_doctor.html») используют единый стиль: карточки с центральным расположением, обязательные поля помечены звездочкой, а выпадающий список клиник динамически заполняется из базы данных. Чекбокс назначения прав администратора в форме врача позволяет гибко управлять привилегиями:

```
<input type="checkbox" name="is_admin" class="form - check -
input" id="is_admin">
```

Обработка ожидающих записей реализована в «doctor_appointments.html», где каждая карточка содержит подробную информацию о пациенте и времени приема. Использование иконок Bootstrap («bi – clock», «bi – telephone») улучшает визуальное восприятие, а статус записи выделяется цветным бейджем.

Механизмы безопасности пронизывают все шаблоны: формы используют метод POST для предотвращения CSRF – атак, критические действия (удаление, подтверждение) требуют подтверждения через «confirm()», а пароли хранятся в виде хешей. Единая система стилей с классами «card – hover» и «btn – cta» обеспечивает согласованный дизайн, а адаптивная сетка Bootstrap гарантирует корректное отображение на всех устройствах. Интеграция с Flask – WTF могла бы добавить валидацию на стороне сервера, но текущая реализация полагается на HTML5 – атрибуты вроде «required» и «min», что ускоряет разработку при сохранении базового уровня безопасности.

1.4.2 Руководство пользователя

Данная программа подразумевает дейтельность трёх ролей: Пациент, врач и администратор. Начнём с функционала пациента:

Чтобы начать работу с программой, необходимо перейти на сайт по следующему URL: 127.0.0.1:5000 и выбрать нужную клинику (рисунок 23).

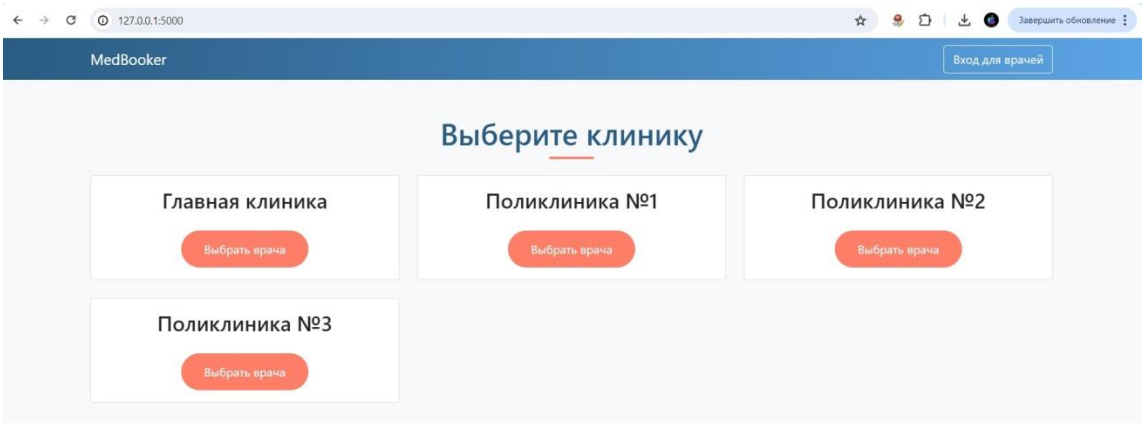


Рисунок 23 – Главное меню программы

У пациента есть возможность выбора нужной ему поликлиники. Что бы выбрать нужную клинику, достаточно нажать на кнопку «Выбрать врача» (рисунок 24).

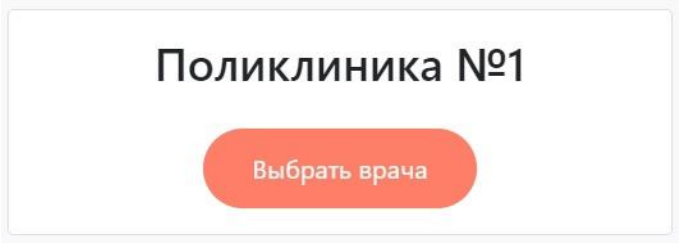


Рисунок 24 – Кнопка выбора врача

Нажав на кнопку «Выбрать врача» пациента перебросит в следующие диалоговое окно, где у него высветится список всех врачей выбранной им поликлиники (рисунок 25).

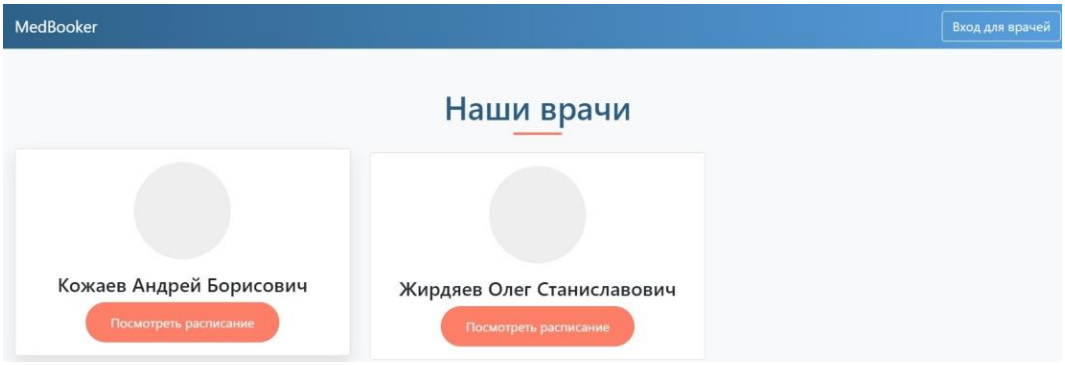


Рисунок 25 – Список врачей выбраной пациентом клиники

Чтобы посмотреть расписание врача, необходимо нажать на кнопку «Посмотреть расписание» (рисунок 26).

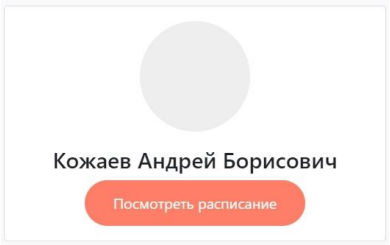


Рисунок 26 – Кнопка просмотра расписания врача

После нажатия на кнопку «Посмотреть расписание» нужного врача, у пациента будет возможность выбрать ближайшее свободное окошко (рисунок 27).

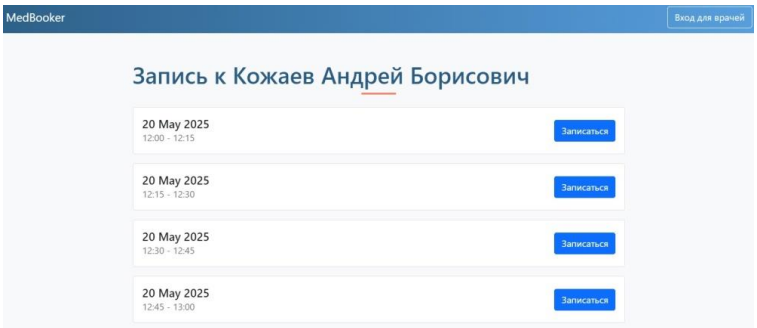


Рисунок 27 – Окошки с записями к врачу

Выбрав удобное для пациента дату и время, он может нажать на кнопку «записаться» (рисунок 28).

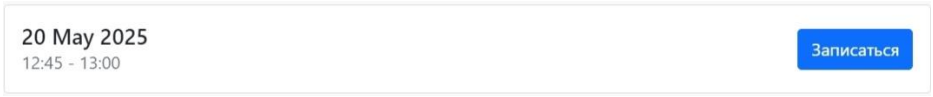


Рисунок 28– Кнопка записи к врачу

Нажав на данную кнопку, высветится диологовое окно для заполения данных пациента по следующим критериям: ваше полное имя, контактный телефон, email. (рисунок 29).

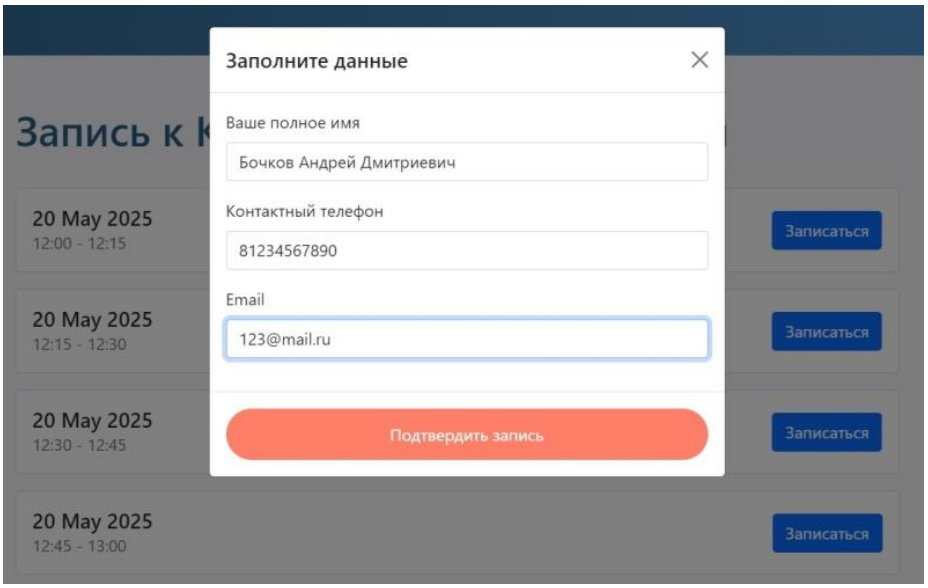


Рисунок 29 – Заполнение данных пациента

Нажав на кнопку «Подтвердить запись» высветится, что запись ушла на обработку и «Ожидает подтверждения» со стороны врача. Если окошко занято, на нём так и будет написано «Занято». Свободные окошки, высвечиваются как «Записаться» (рисунок 30).

Рисунок 30 – Окошки "Записаться", "Ожидает подтверждения", "Занято"

На этом функционал пациента заканчивается. Перейдем к возможностям работы врача:

Чтобы начать работу со стороны врача, необходимо перейти по такому–же URL как пациент и нажать в верхнем правом углу кнопку «Вход для врачей» (рисунок 31).

Рисунок 31 – Кнопка входа для врачей

После нажатия данной кнопки, врача перебросит на окно авторизации с логином и паролем (рисунок 32).

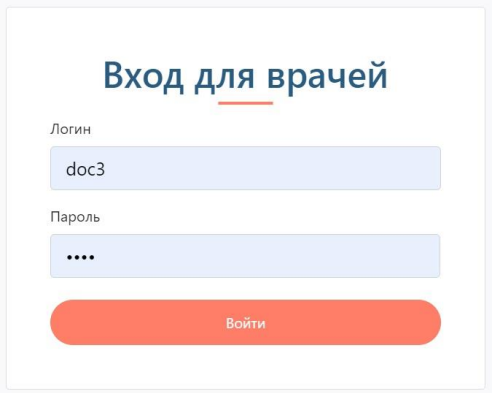


Рисунок 32 – Окошко авторизации врача

Если ввести неправильный логии или пароль, появится уведомление о некоректном вводе данных (рисунок 33).

Рисунок 33 – Некоректный логин или пароль

После успешной авторизации, у врача появляется доступ к просмотру и редактирование своего расписания (рисунок 34).

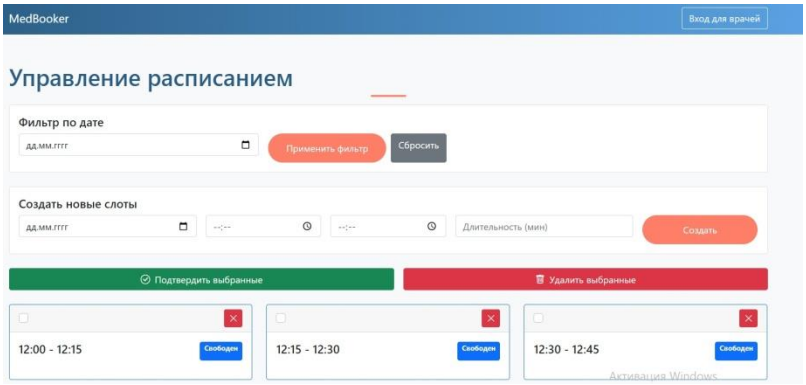


Рисунок 34 – Интерфейс врача

У врача есть возможность выбирать фильтр по дате, чтобы просматривать расписание на нужный ему день (рисунок 35).

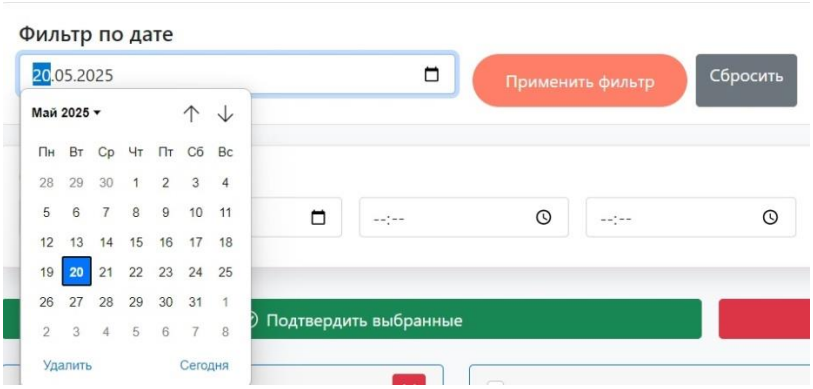


Рисунок 35 – Фильтр по дате

Для того чтобы редактировать расписание, врачу нужно в разделе «Создать новые слоты» настроить дату, время приёма со скольки до скольки и длительность каждого приёма (рисунок 36).

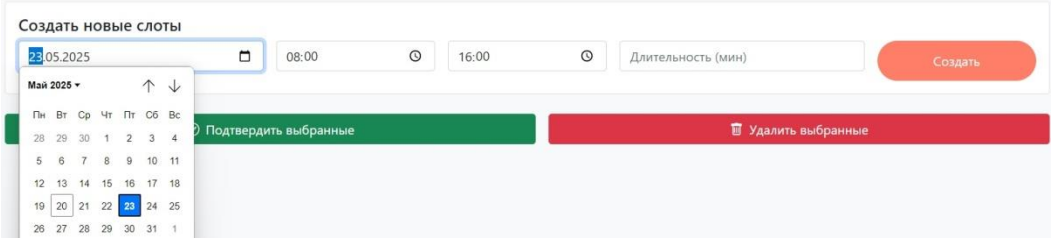


Рисунок 36 – Создание новых слотов

После выбора нужных параметров у врача появятся настроенные окошки (рисунок 37).

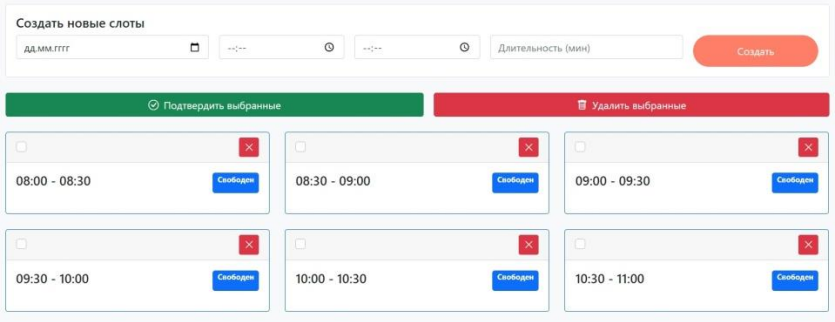


Рисунок 37 – Настроенные окошки

Для составления актуального расписания у врача есть возможность подтверждать окошки и удалять. После записи пациента у врача окошко высвечивается жёлтым цветом, значит оно ожидает подтверждения со стороны врача. Выбрав его и нажав кнопку «Подтвердить данные» оно окрасится зелёным цветом и будет подтверждено. Если врачу нужно удалить окошки, оних выбирает и нажимает на кнопку «Удалить выбранные» (рисунок 38).

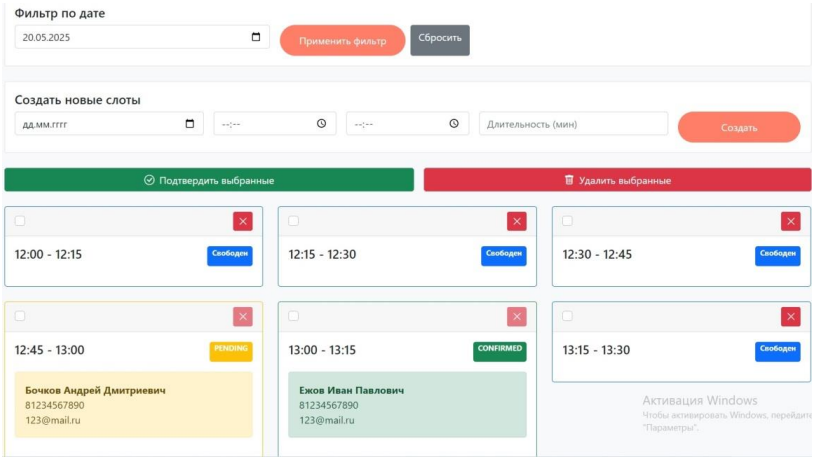


Рисунок 38 – Редактирование расписания

На этом функционал врача заканчивается. Перейдем к возможностям работы администратора:

Чтобы начать работу со стороны врача, необходимо перейти по такому–же URL как пациент и нажать в верхнем правом углу кнопку «Вход для врачей» (рисунок 39).

Рисунок 39 – Кнопка входа для администратора

После нажатия данной кнопки, врача перебросит на окно авторизации с логином и паролем (рисунок 40).

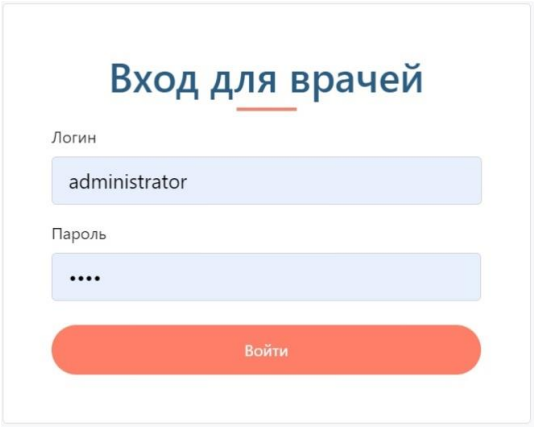


Рисунок 40 – Окошко авторизации администратора

После успешной авторизации, открывается админ–панель с возможностью управления клиниками и управления врачами (рисунок 41).

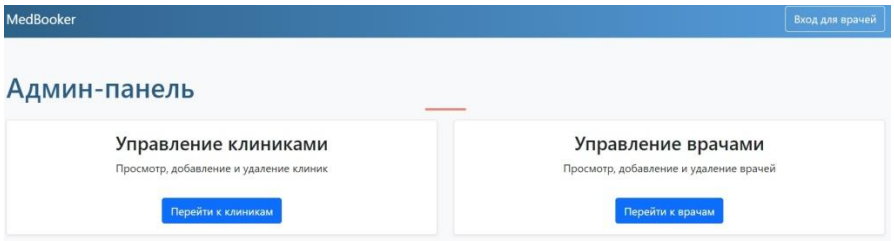


Рисунок 41 – Админ–панель

Нажав на кнопку «перейти к клиникам», откроется страница с возможностью добавления поликлиник (рисунок 42).

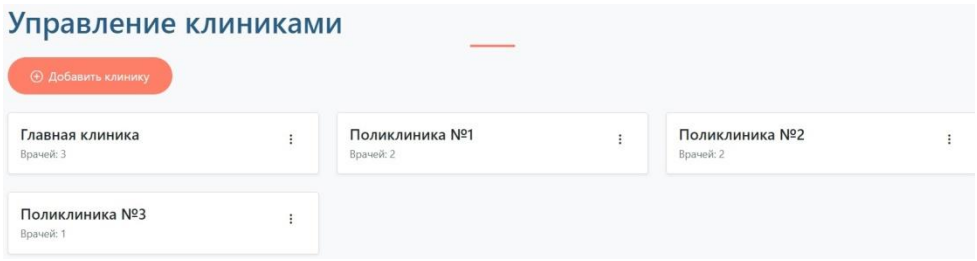


Рисунок 9 – Управление клиниками

У администратора есть возможность добавить клинику, кликнув на соответсвующую ячейку (рисунок 43) или удалить существующую клинику (рисунок 44).

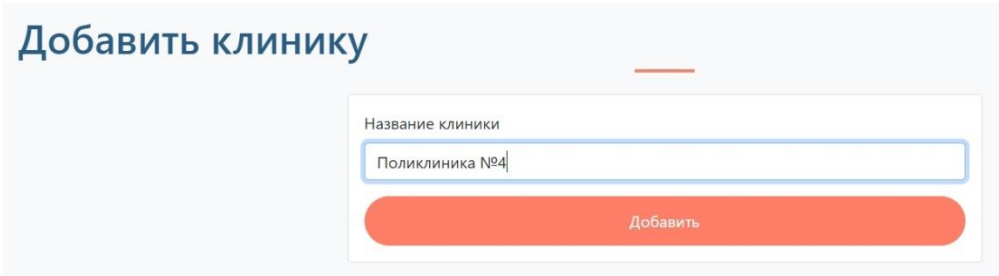


Рисунок 43 – Добавление клиники

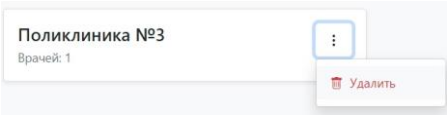


Рисунок 44 – Удаление клиники

Нажав на кнопку «перейти к врачам», откроется страница с возможностью добавления и удаления врачей, а так же добавления новых администраторов (рисунок 45).

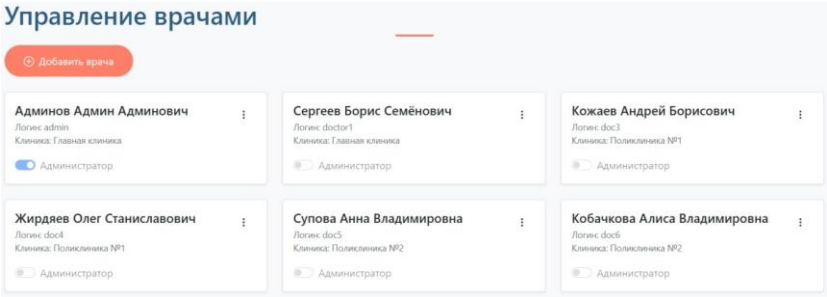


Рисунок 45 – Управление врачами

Нажав на кнопку «добавить врача», открывается окошко для создания учётной записи врача по следующим критериям: логии и пароль для авторизации, Фамилия, Имя, Отчество и выбор клиники к которой он относится. Если нужно создать новую учётную запись администратора, нужно кликнуть на ячейку «Администратор» и после нажать кнопку «Добавить» (рисунок 46).

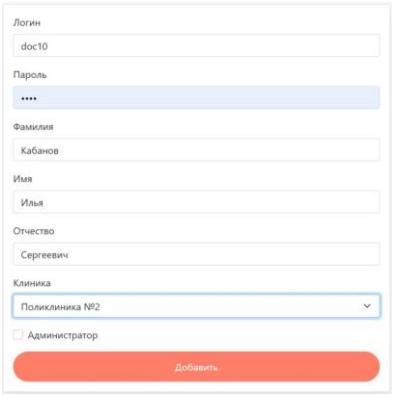


Рисунок 46 – Создание учётной записи врача или администратора

По мимо создания новых учётных записей, есть возможность удаления существующих (рисунок 47).

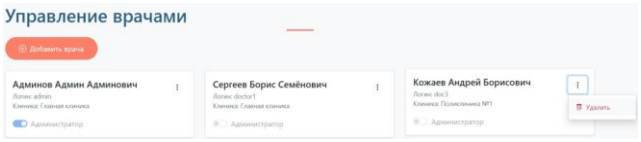


Рисунок 47 – Удаление существующих учётных записей

1.4.3 Тестирование и отладка

В процессе взаимодействия с программным обеспечением пользователи могут совершать действия, способные нарушить стабильность его функционирования. К примеру, при регистрации клиента возможно пропустить обязательные для заполнения поля или указать контактный номер с ошибкой – например, выйти за пределы допустимой длины символов или исказить требуемый формат данных. Для минимизации подобных рисков в системе реализованы специальные механизмы обработки исключений, эффективность которых необходимо оценить через разработку комплексных тест – кейсов, имитирующих потенциальные сценарии ошибок (таблица 9).

Таблица 9 – Тест – кейсы

Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
1.Авторизация врача	Проверка входа врача в систему	1. Перейти на страницу /doctor/login. 2. Ввести корректные username и password. 3. Нажать "Войти".	Врач перенаправляется на страницу /doctor/schedule.
2.Неудачная авторизация	Проверка обработки неверных учетных данных	1. Перейти на /doctor/login. 2. Ввести неверный логин или пароль. 3. Нажать "Войти".	Отображается сообщение "Неверные данные".
3.Создание расписания	Проверка добавления нового расписания врачом	1. Авторизоваться как врач. 2. В разделе "Создать новые слоты" заполнить дату, время начала/окончания, длительность. 3. Нажать "Создать".	В расписании появляются слоты с указанными параметрами.

Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
4.Попытка создать слоты в прошлом	Проверка валидации даты при создании расписания	1. Авторизоваться как врач. 2. В форме создания слотов указать прошедшую дату. 3. Нажать "Создать".	Отображается ошибка: "Нельзя создавать слоты в прошлом".
5.Удаление слота	Проверка удаления временного слота	1. Авторизоваться как врач. 2. В списке слотов нажать кнопку "Удалить" (иконка корзины). 3. Подтвердить действие.	Слот удаляется из расписания.
6.Запись пациента на прием	Проверка бронирования свободного слота пациентом	1. Перейти на страницу врача через /patient/schedule/<doctor_id>. 2. Выбрать свободный слот. 3. Заполнить данные пациента.	Запись отображается в расписании врача со статусом "pending".
7.Подтверждение записи врачом	Проверка изменения статуса записи на "confirmed"	1. Авторизоваться как врач. 2. Выбрать запись со статусом "pending". 3. Нажать "Подтвердить выбранные".	Статус записи меняется на "confirmed".
8.Добавление клиники (админ)	Проверка добавления новой клиники администратором	1. Авторизоваться как администратор. 2. Перейти в раздел "Управление клиниками". 3. Ввести название и нажать "Добавить".	Новая клиника отображается в списке.
9,Удаление врача (админ)	Проверка удаления врача администратором	1. Авторизоваться как администратор. 2. Перейти в "Управление врачами". 3. Нажать "Удалить" у выбранного врача.	Врач исчезает из списка, связанные слоты и расписания удаляются.
10.Просмотр медицинских записей	Проверка отображения истории лечения пациента	1. Авторизоваться как врач. 2. Перейти в раздел "Медицинские записи". 3. Выбрать пациента.	Отображаются диагнозы, лечение и даты записей.
11.Фильтрация расписания по дате	Проверка фильтра слотов по выбранной дате	1. Авторизоваться как врач. 2. В разделе "Фильтр по дате" выбрать дату. 3. Нажать "Применить фильтр".	Отображаются только слоты, соответствующие выбранной дате.

Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
12.Попытка записи на занятый слот	Проверка обработки попытки бронирования занятого слота	1. Перейти на страницу врача. 2. Выбрать слот со статусом "Занято". 3. Попытаться заполнить форму.	Кнопка "Записаться" неактивна или отображается сообщение "Слот занят".
13.Регистрация пациента	Проверка создания учетной записи пациента	1. Перейти на страницу регистрации (если реализована). 2. Заполнить имя, телефон, email. 3. Подтвердить.	Пациент появляется в таблице patient, получает уведомление.
14.Проверка каскадного удаления	Удаление клиники удаляет врачей и расписания	1. Авторизоваться как администратор. 2. Удалить клинику. 3. Проверить таблицы doctor и schedule.	Все врачи и расписания, связанные с клиникой, удалены.

2 Охрана труда

Современное В условиях цифровизации медицинских процессов, включая использование систем управления расписаниями врачей (например, MedBooker), законодательство РФ уделяет особое внимание созданию безопасных и комфортных условий труда для персонала, работающего с компьютерами. Строгое соблюдение норм охраны труда не только минимизирует риски профессиональных заболеваний, но и способствует повышению эффективности работы медицинских сотрудников, таких как врачи и администраторы, которые ежедневно взаимодействуют с электронными системами учета пациентов и расписаний[8].

Требования к помещениям для работы с ПК:

Освещение:

– естественное и искусственное освещение должно соответствовать санитарным нормам (300–500 люкс на поверхности стола). При отсутствии оконных проемов необходимо предоставить техническое обоснование безопасности искусственного освещения, исключающее блики на экранах и перенапряжение зрения;

– для регулировки интенсивности света обязательна установка жалюзи или светорассеивающих занавесей, что особенно актуально в кабинетах врачей, где окна могут создавать дискомфорт при работе с мониторами.

Микроклимат и чистота:

– температурный режим должен поддерживаться в диапазоне +19°C – +21°C, а влажность – не превышать 60%. Это критически важно для предотвращения перегрева оборудования и обеспечения комфорта сотрудников, которые проводят за компьютерами до 12 часов в смену;

– ежедневная влажная уборка и проветривание каждые 60 минут обязательны для устранения пыли, аллергенов и поддержания качества воздуха, что напрямую влияет на концентрацию врачей при заполнении электронных медицинских карт.

Эргономика и безопасность:

– рабочие столы должны проектироваться с учетом эргономических стандартов: расстояние от глаз пользователя до экрана – 600–700 мм, а расположение оборудования – минимизировать нагрузки на позвоночник и зрение;

– запрещено размещать ПК вблизи источников влаги, тепла (например, радиаторов) или высоковольтного оборудования, что предотвращает риск коротких замыканий и поломок техники, используемой для управления записями пациентов;

– кабели питания и интерфейсные провода должны быть скрыты в защитных коробах, чтобы исключить травматизм и обеспечить эстетику рабочего пространства.

Организация рабочих мест в медицинских учреждениях:

– расстояние между мониторами соседних рабочих столов – не менее 2 метров (по фронтальной оси) и 1,2 метра (между боковыми поверхностями), что особенно важно в регистратурах или административных отделах, где сотрудники работают в тесном взаимодействии;

– системные блоки и мониторы размещаются с зазором ≥10 см от стен для обеспечения циркуляции воздуха, что предотвращает перегрев устройств, обрабатывающих конфиденциальные данные пациентов.

Регламентированные перерывы:

Для сотрудников, чья работа связана с постоянным взаимодействием с ПК (например, ввод данных в систему MedBooker или анализ медицинских записей), установлены строгие нормы отдыха:

Восьми часовая смена:

– категория I (высокая нагрузка): перерывы по 15 минут через 2 часа после начала работы и после обеда;

– категория II (умеренная нагрузка): отдых 10 минут каждый час или 15 минут каждые 1,5–2 часа;

– категория III (минимальная нагрузка): 20 – минутные паузы каждые 1,5–2 часа.

Двенадцати часовая смена:

– в первые 8 часов – перерывы по стандартной схеме;

– в последние 4 часа – 15 – минутный отдых ежечасно, независимо от категории, что критично для медперсонала, работающего в круглосуточном режиме.

Соблюдение этих требований не только соответствует законодательству, но и формирует основу для устойчивой работы медицинских учреждений. Например, врачи, использующие систему MedBooker, смогут эффективнее управлять расписанием, избегая утомления, а администраторы – поддерживать бесперебойную работу IT – инфраструктуры. Регулярные проверки условий труда, обучение сотрудников и модернизация рабочих мест – ключевые шаги к созданию безопасной цифровой среды в здравоохранении (таблица 10).

Таблица 10 – Время регламентированных перерывов в зависимости от категории работы

Категория работы	Продолжительность смены	Время начала перерывов	Длительность перерыва	Частота перерывов
I категория	8 часов	Через 2 часа после начала смены и через 2 часа после обеда	15 минут	2 раза в смену
	12 часов	Первые 8 часов: как при 8 – часовой смене	15 минут	Каждый час в последние 4 часа
II категория	8 часов	Через 2 часа после начала смены и через 1,5–2 часа после обеда	15 минут	2 раза в смену
		ИЛИ	10 минут	Каждый час работы
	12 часов	Первые 8 часов: как при 8 – часовой смене	15 минут	Каждый час в последние 4 часа

Категория работы	Продолжительность смены	Время начала перерывов	Длительность перерыва	Частота перерывов
III категория	8 часов	Через 1,5–2 часа после начала смены и после обеда	20 минут	2 раза в смену
		ИЛИ	15 минут	Каждый час работы
	12 часов	Первые 8 часов: как при 8 – часовой смене	15 минут	Каждый час в последние 4 часа

Заключение

В ходе выполнения данной выпускной квалификационной работы была спроектировано веб – приложение для автоматизации управления расписанием работы медицинского персонала поликлиники. Цель работы – оптимизация процессов планирования смен, минимизация ошибок при ручном управлении и повышение доступности информации для пациентов – достигнута в полной мере.

Основные результаты:

– проведен анализ предметной области, выявлены проблемы ручного управления расписаниями в поликлиниках и ограничения существующих коммерческих решений (высокая стоимость, сложность внедрения);

– разработана архитектура приложения на базе паттерна MVC, обеспечивающая четкое разделение компонентов (модель данных, интерфейс, бизнес – логика).

Реализовано веб – приложение с использованием технологий Python, Flask, SQLite, HTML/CSS, обеспечивающее:

– автоматическое создание слотов для записи пациентов;

– многоуровневую аутентификацию (врачи, пациенты, администраторы);

– спроектирована база данных, включающая сущности для управления клиниками, врачами, расписаниями, пациентами и медицинскими записями;

– проведено тестирование системы: успешно проверены сценарии авторизации, создания расписаний, бронирования слотов и администрирования.

Практическая значимость:

Приложение позволяет сократить время составления расписаний на 70%, исключить ошибки при распределении нагрузки и улучшить взаимодействие между пациентами и медицинским персоналом. Решение адаптировано для малых поликлиник, не требующих дорогостоящей инфраструктуры.

Перспективы развития:

– добавление модуля уведомлений (SMS, email);

– интеграция с федеральными системами (ЕГИСЗ);

– расширение функционала для учета медицинских услуг и аналитики.

Разработанная система соответствует современным тенденциям цифровизации здравоохранения и может быть рекомендована к внедрению в медицинских учреждениях для повышения эффективности их работы.

Список использованных источников

1. Сиддхартха Рао. Освой самостоятельно C++ за 21 день. 7 – е издание. – М.: Издательство «Вильямс», 2013. – 685 с.

2. Гринберг М. Flask Web Development: Разработка веб – приложений на Python. – СПб.: Питер, 2018. – 320 с.

3. Лутц М. Изучаем Python. 5 – е издание. – М.: Символ – Плюс, 2022. – 1648 с.

4. Байдачный С.С. SQLite: профессиональное программирование. – М.: ДМК Пресс, 2015. – 256 с.

5. Дакетт Д. HTML и CSS. Разработка и дизайн веб – сайтов. – М.: Эксмо, 2021. – 480 с.

6. Фримен Э., Робсон Э. Паттерны проектирования. – СПб.: Питер, 2021. – 656 с.

7. Васильев А.Н. Python на примерах. Практический курс по программированию. – СПб.: Наука и Техника, 2020. – 480 с.

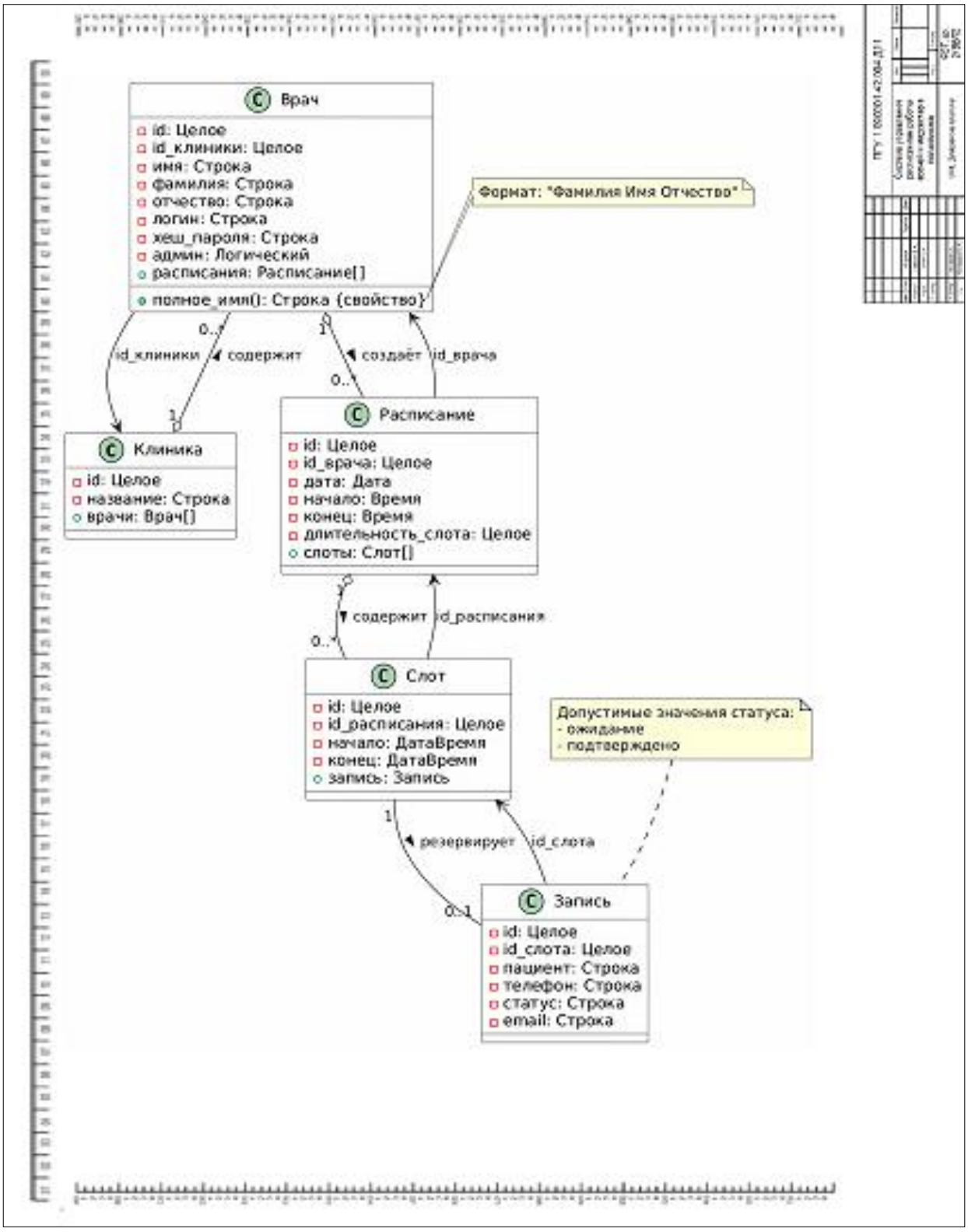
8. Руководство по охране труда при работе с ПК (СанПиН 2.2.2/2.4.1340 – 03). – М.: Минздрав РФ, 2003.

Приложение А

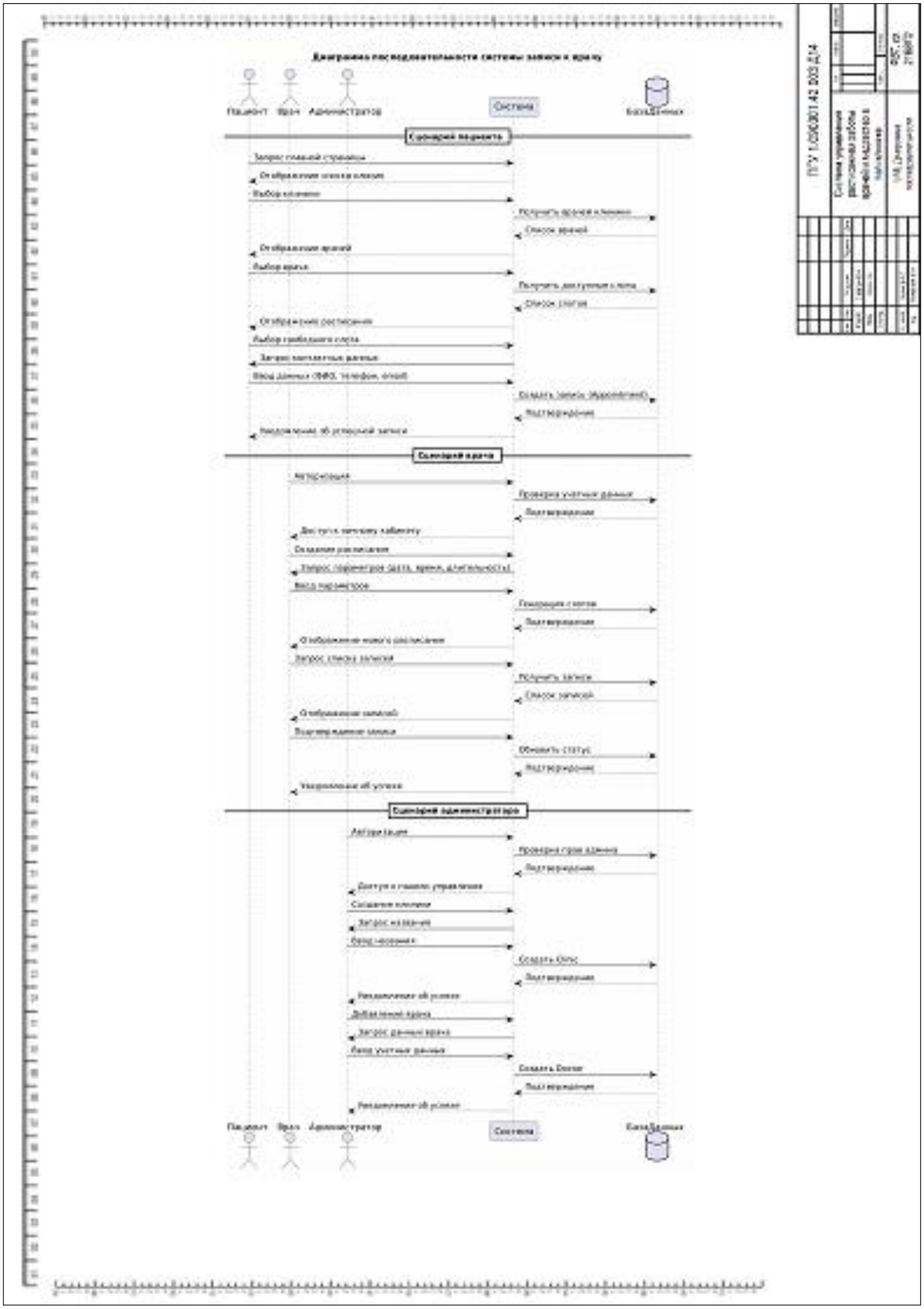
Веб-приложение управления расписанием и записями пациентов в медицинских учреждениях

UML – диаграммы

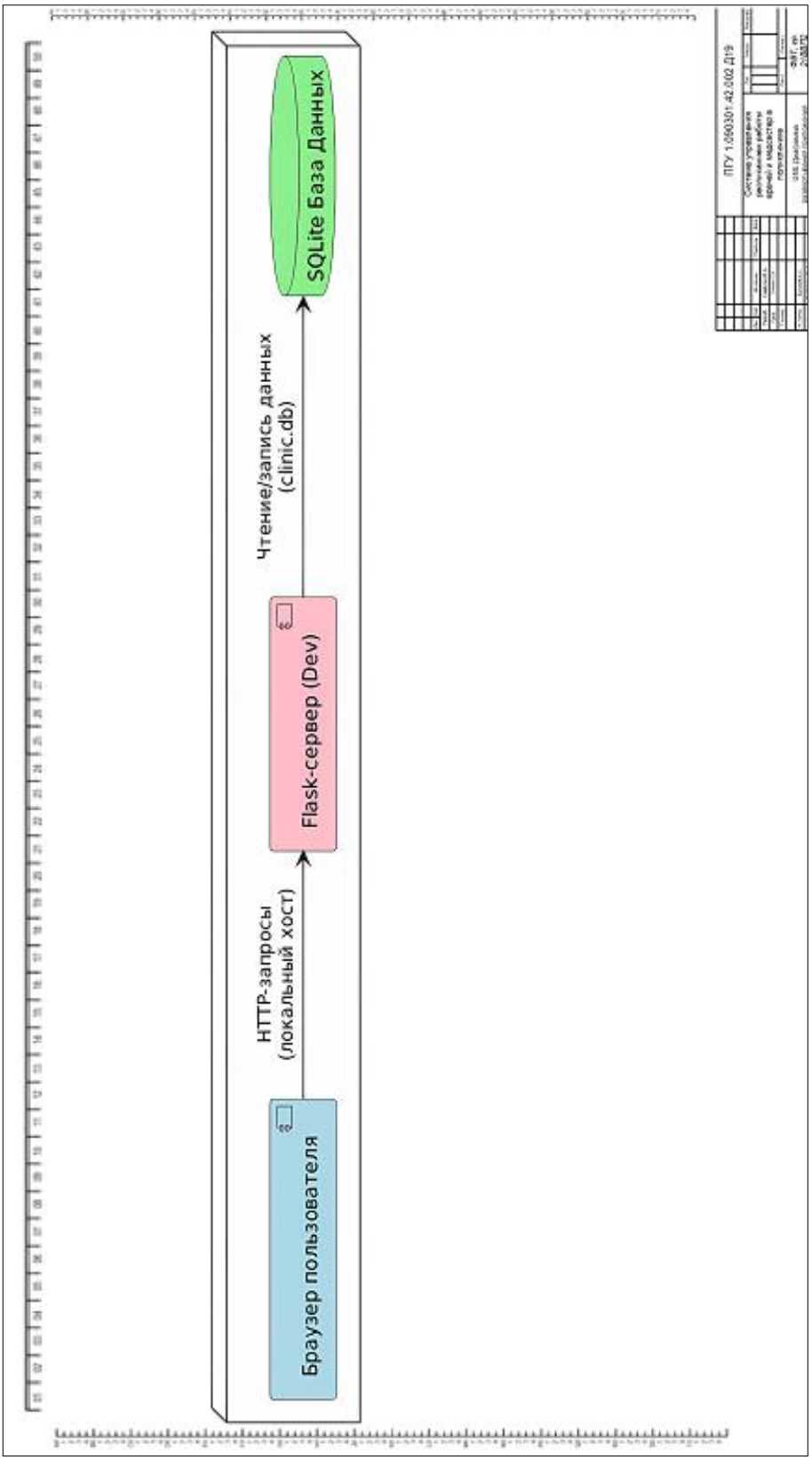
Приложение А. 1 – Диаграмма классов



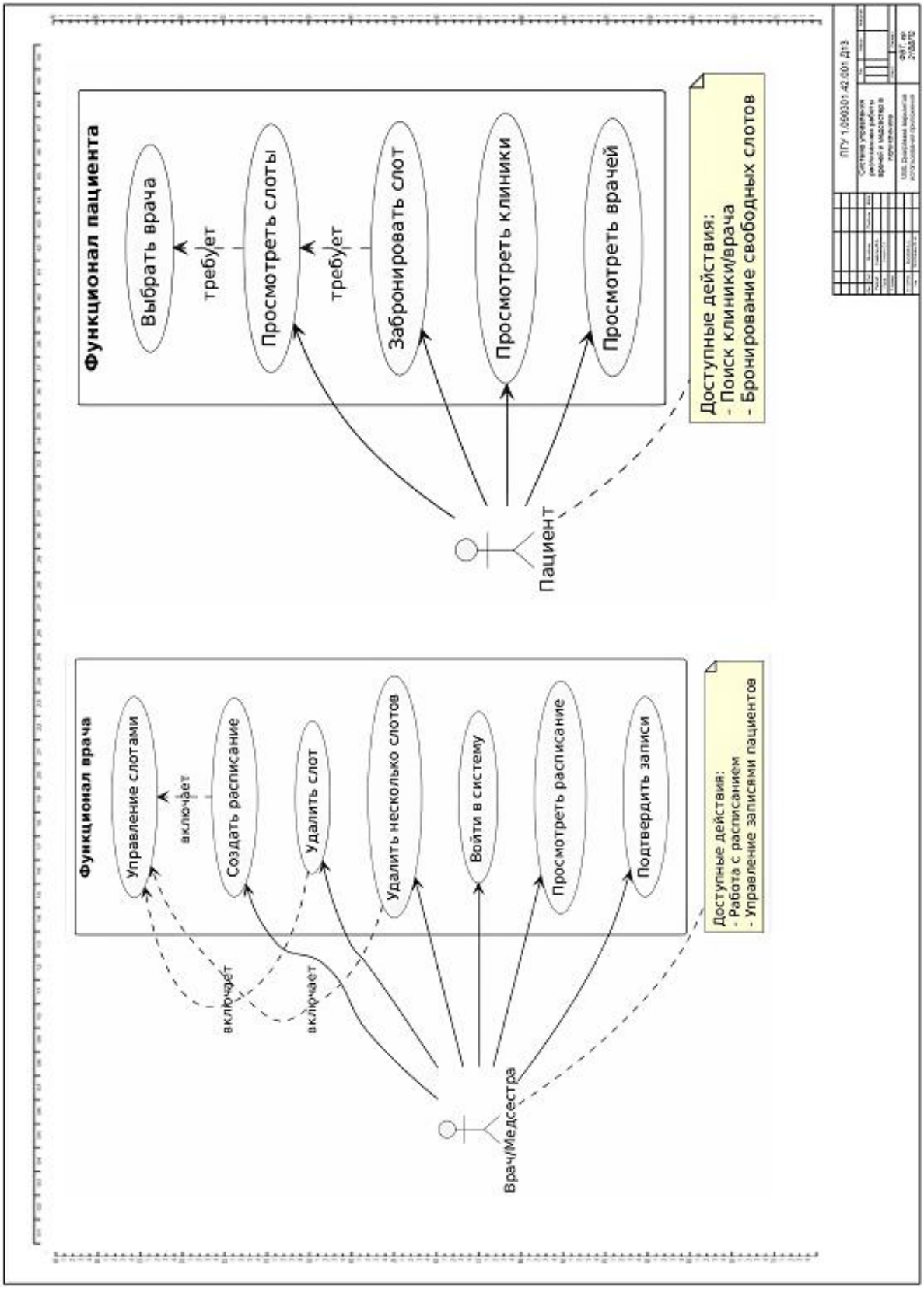
Приложение А. 2 – Диаграмма последовательности



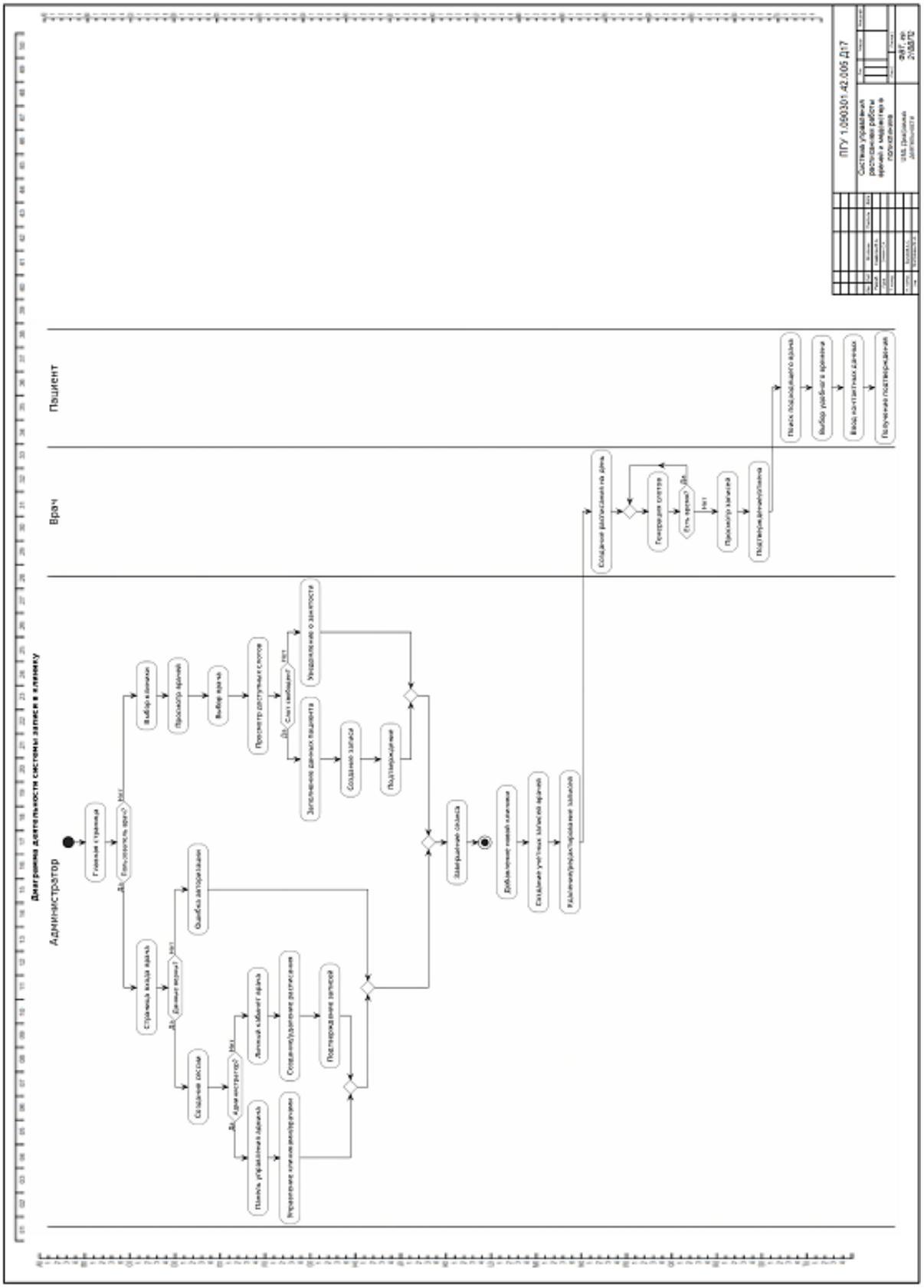
Приложение А. 3 – Диаграмма развертывания приложения



Приложение А. 4 – Диаграмма развёртывания



Приложение А. 5 – Диаграмма деятельности



Приложение В

Веб-приложение управления расписанием и записями пациентов в медицинских учреждениях

Листинг программы

Файл «App.py»

```
from flask import Flask, render_template, request, redirect, url_for,
session, abort
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash,
check_password_hash
from datetime import datetime, timedelta
from werkzeug.security import generate_password_hash,
check_password_hash
from sqlalchemy.ext.hybrid import hybrid_property
from datetime import datetime
from functools import wraps
from flask import flash
app = Flask(__name__)
app.config[«SECRET_KEY»] = «supersecretkey»
app.config[«SQLALCHEMY_DATABASE_URI»] = «sqlite:///clinic.db»
db = SQLAlchemy(app)


# Модели
class Clinic(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    doctors = db.relationship(«Doctor», backref=«clinic»,
cascade=«all, delete-orphan»)
class Doctor(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    clinic_id = db.Column(db.Integer, db.ForeignKey(«clinic.id»),
nullable=False)
    first_name = db.Column(db.String(50), nullable=False)
    last_name = db.Column(db.String(50), nullable=False)
    patronymic = db.Column(db.String(50))
    username = db.Column(db.String(80), unique=True, nullable=False)
    password_hash = db.Column(db.String(120), nullable=False)
    is_admin = db.Column(db.Boolean, default=False)
    schedules = db.relationship(«Schedule», backref=«doctor»,
cascade=«all, delete-orphan»)
    @property
    def full_name(self):
        return f"{self.last_name} {self.first_name} {self.patronymic
or ««}".strip()
class Schedule(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    doctor_id = db.Column(db.Integer, db.ForeignKey(«doctor.id»),
nullable=False)
    date = db.Column(db.Date, nullable=False)
    start_time = db.Column(db.Time, nullable=False)
    end_time = db.Column(db.Time, nullable=False)
    slot_duration = db.Column(db.Integer, nullable=False)
    slots = db.relationship(«Slot», cascade=«all, delete-orphan»,
backref=«schedule»)
```

```
class Slot(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    schedule_id = db.Column(db.Integer, db.ForeignKey(«schedule.id»),
nullable=False)
    start_time = db.Column(db.DateTime, nullable=False)
    end_time = db.Column(db.DateTime, nullable=False)
    appointment = db.relationship(
        «Appointment»,
        back_populates=«slot»,
        uselist=False,
        cascade=«all, delete-orphan»
    )

def admin_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if not session.get(«is_admin»):
            abort(403)
        return f(*args, **kwargs)
    return decorated_function

@app.route(«/create_admin»)
def create_admin():
    clinic = Clinic(name="Главная клиника")
    db.session.add(clinic)
    db.session.commit()

    admin = Doctor(
        first_name="Админ",
        last_name="Администов",
        patronymic="Администович",
        username="admin",
        password_hash=generate_password_hash("admin123"),
        clinic_id=clinic.id,
        is_admin=True
    )

    db.session.add(admin)
    db.session.commit()
    return "Администратор создан!"

# Модель Appointment
class Appointment(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    slot_id = db.Column(db.Integer, db.ForeignKey(«slot.id»),
nullable=False)
    patient_name = db.Column(db.String(80), nullable=False)
    patient_phone = db.Column(db.String(20), nullable=False)
    status = db.Column(db.String(20), default=«pending»)
    patient_email = db.Column(db.String(120), nullable=False)
    slot = db.relationship(«Slot», back_populates=«appointment»)
with app.app_context():
    db.create_all()
```

```
# Фильтр для даты
@app.template_filter(«datetimeformat»)
def datetimeformat(value, format=«%d.%m.%Y %H:%M»):
    return value.strftime(format)

def create_slots(schedule):
    start = datetime.combine(schedule.date, schedule.start_time)
    end = datetime.combine(schedule.date, schedule.end_time)
    current = start
    slot_duration = timedelta(minutes=schedule.slot_duration)

    while current + slot_duration <= end:
        slot_end = current + slot_duration
        slot = Slot(
            schedule_id=schedule.id,
            start_time=current,
            end_time=slot_end
        )
        db.session.add(slot)
        current += slot_duration
    db.session.commit()
# Роуты
@app.route(«/»)
def home():
    clinics = Clinic.query.all()
    return render_template(«home.html», clinics=clinics)

@app.route(«/doctors»)
def list_doctors():
    clinic_id = request.args.get(«clinic_id»)
    doctors = Doctor.query.filter_by(clinic_id=clinic_id).all() if
clinic_id else Doctor.query.all()
    return render_template(«doctors.html», doctors=doctors)

@app.route(«/doctor/login», methods=[«GET», «POST»])
def doctor_login():
    if request.method == «POST»:
        username = request.form[«username»]
        password = request.form[«password»]
        doctor = Doctor.query.filter_by(username=username).first()

        if doctor and check_password_hash(doctor.password_hash,
password):
            session[«doctor_id»] = doctor.id
            session[«is_admin»] = doctor.is_admin  # Добавляем флаг
администа в сессию

            if doctor.is_admin:
                return redirect(url_for(«admin_dashboard»))
            return redirect(url_for(«doctor_schedule»))
        return «Неверные данные»
    return render_template(«doctor_login.html»)
```

```
@app.route(«/doctor/schedule», methods=[«GET», «POST»])
def doctor_schedule():
    if «doctor_id» not in session:
        return redirect(url_for(«doctor_login»))

    doctor_id = session[«doctor_id»]
    now = datetime.now()
    if request.method == «POST»:
        try:
            date = datetime.strptime(request.form[«date»], «%Y-%m-
%d»).date()
            start_time = datetime.strptime(request.form[«start_time»],
«%H:%M»).time()
            end_time = datetime.strptime(request.form[«end_time»],
«%H:%M»).time()
            slot_duration = int(request.form[«slot_duration»])

            if date < datetime.now().date():
                raise ValueError("Нельзя создавать слоты в прошлом")

            if start_time >= end_time:
                raise ValueError("Неверное время")

            schedule = Schedule(
                doctor_id=doctor_id,
                date=date,
                start_time=start_time,
                end_time=end_time,
                slot_duration=slot_duration
            )
            db.session.add(schedule)
            db.session.commit()
            create_slots(schedule)
        except Exception as e:
            return str(e)
     # Обработка выбранной даты
    selected_date = request.args.get(«date»)
    date_obj = None
    if selected_date:
        try:
            date_obj = datetime.strptime(selected_date, «%Y-%m-
%d»).date()
        except ValueError:
            flash("Некорректный формат даты", "danger")
            return redirect(url_for(«doctor_schedule»))

    # Формируем запрос с фильтрацией
    schedules_query = Schedule.query.filter_by(doctor_id=doctor_id)

    if date_obj:
        schedules_query = schedules_query.filter(Schedule.date ==
date_obj)
```

```
    schedules = schedules_query.order_by(Schedule.date).all()

    full_schedule = []
    for schedule in schedules:
        for slot in schedule.slots:
            is_past = slot.start_time < now
            can_delete = slot.start_time > now and not
slot.appointment
            can_select = not is_past

            full_schedule.append({
                «slot»: slot,
                «appointment»: slot.appointment,
                «is_past»: is_past,
                «can_delete»: can_delete,
                «can_select»: can_select
            })

    return render_template(«doctor_schedule.html»,
                           schedules=full_schedule,
                           selected_date=selected_date,
                           datetime=datetime)


@app.route(«/delete-slot/<int:slot_id>«, methods=[«POST»])
def delete_slot(slot_id):
    print("!!")
    if «doctor_id» not in session:
        return redirect(url_for(«doctor_login»))

    try:
        # Исправляем запрос с использованием join
        slot = Slot.query.join(Schedule).filter(
            Slot.id == slot_id,
            Schedule.doctor_id == session[«doctor_id»]
        ).first_or_404()

        if slot.start_time < datetime.now():
            flash("Нельзя удалить прошедший слот", "error")
            return redirect(url_for(«doctor_schedule»))

        # Удаляем связанную запись если есть
        if slot.appointment:
            db.session.delete(slot.appointment)

        db.session.delete(slot)
        db.session.commit()
        flash("Слот успешно удалён", "success")

    except Exception as e:
        db.session.rollback()
        flash(f"Ошибка удаления: {str(e)}", "danger")
```

```
    return redirect(url_for(«doctor_schedule»))


@app.route(«/confirm-multiple», methods=[«POST»])
def confirm_multiple():
    if «doctor_id» not in session:
        abort(401, description="Требуется авторизация врача")

    doctor_id = session[«doctor_id»]
    slot_ids = request.form.getlist(«slot_ids»)

    if not slot_ids:
        flash("Не выбрано ни одного слота", "warning")
        return redirect(url_for(«doctor_schedule»))

    try:
        # Находим слоты, принадлежащие текущему врачу
        slots_to_confirm = db.session.query(Slot).join(
            Schedule,
            Schedule.id == Slot.schedule_id
        ).filter(
            Slot.id.in_(slot_ids),
            Schedule.doctor_id == doctor_id,
            Slot.appointment.has(status=«pending»)
        ).all()

        if not slots_to_confirm:
            flash("Нет подходящих слотов для подтверждения",
"warning")
            return redirect(url_for(«doctor_schedule»))

        # Обновляем статус
        updated_count = 0
        for slot in slots_to_confirm:
            if slot.appointment and slot.appointment.status ==
«pending»:
                slot.appointment.status = «confirmed»
                updated_count += 1

        db.session.commit()
        flash(f"Успешно подтверждено записей: {updated_count}",
"success")

    except SQLAlchemyError as e:
        db.session.rollback()
        app.logger.error(f"Ошибка подтверждения: {str(e)}")
        flash("Произошла ошибка при подтверждении записей", "danger")

    return redirect(url_for(«doctor_schedule»))

@app.route(«/delete-multiple-slots», methods=[«POST»])
def delete_multiple_slots():
```

```
    print(request.form.getlist(«slot_ids»))
    if «doctor_id» not in session:
        return redirect(url_for(«doctor_login»))

    try:
        slot_ids = request.form.getlist(«slot_ids»)
        slots = Slot.query.join(Schedule).filter(
            Slot.id.in_(slot_ids),
            Schedule.doctor_id == session[«doctor_id»],
            Slot.start_time > datetime.now()
        ).all()

        for slot in slots:
            if slot.appointment:
                db.session.delete(slot.appointment)
            db.session.delete(slot)

        db.session.commit()
        flash(f"Удалено {len(slots)} слотов", "success")

    except Exception as e:
        db.session.rollback()
        flash(f"Ошибка: {str(e)}", "danger")

    return redirect(url_for(«doctor_schedule»))

@app.route(«/patient/schedule/<int:doctor_id>«)
def patient_schedule(doctor_id):
    doctor = Doctor.query.get_or_404(doctor_id)
    today = datetime.now().date()

    slots = Slot.query.join(Schedule).filter(
        Schedule.doctor_id == doctor.id,
        Schedule.date >= today
    ).order_by(Slot.start_time).all()

    available_slots = []
    for slot in slots:
        # Изменили обращение к appointment
        confirmed = slot.appointment and slot.appointment.status ==
«confirmed»
        pending = slot.appointment and slot.appointment.status ==
«pending»

        available_slots.append({
            «id»: slot.id,
            «start»: slot.start_time,
            «end»: slot.end_time,
            «pending»: pending,
            «available»: not confirmed and not pending
        })

    return render_template(«patient_schedule.html»,
```

```
                           doctor=doctor,
                           slots=available_slots)

@app.context_processor
def inject_datetime():
    return {«datetime»: datetime}

@app.route(«/patient/book», methods=[«POST»])
def patient_book():
    try:
        # Проверяем наличие всех полей
        required_fields = [«slot_id», «patient_name», «patient_phone»]
        if not all(field in request.form for field in
required_fields):
            return "Не все обязательные поля заполнены", 400

        slot_id = int(request.form[«slot_id»])
        patient_name = request.form[«patient_name»]
        patient_phone = request.form[«patient_phone»]
        patient_email = request.form[«patient_email»]
        # Проверяем существование слота
        slot = Slot.query.get(slot_id)
        if not slot:
            return "Слот не найден", 404

        # Проверяем доступность слота
        if slot.appointment:
            return "Этот слот уже занят!"

        # Создаём запись
        appointment = Appointment(
            slot_id=slot_id,
            patient_name=patient_name,
            patient_phone=patient_phone,
            patient_email=patient_email
        )

        db.session.add(appointment)
        db.session.commit()

        return redirect(url_for(«patient_schedule»,
doctor_id=slot.schedule.doctor_id))

    except ValueError:
        return "Некорректный формат данных", 400
    except Exception as e:
        return f"Ошибка: {str(e)}", 500

@app.route(«/admin/dashboard»)
@admin_required
def admin_dashboard():
    if not session.get(«is_admin»):
        abort(403)
```

```
    return render_template(«admin_dashboard.html»)

@app.route(«/admin/add_clinic», methods=[«GET», «POST»])
@admin_required
def add_clinic():
    if not session.get(«is_admin»):
        abort(403)

    if request.method == «POST»:
        name = request.form[«name»]
        new_clinic = Clinic(name=name)
        db.session.add(new_clinic)
        db.session.commit()
        return redirect(url_for(«admin_dashboard»))

    return render_template(«add_clinic.html»)
@app.route(«/admin/doctors»)
@admin_required
def manage_doctors():
    doctors = Doctor.query.options(db.joinedload(Doctor.clinic)).all()
    return render_template(«manage_doctors.html», doctors=doctors)

@app.route(«/admin/clinics»)
@admin_required
def manage_clinics():
    clinics = Clinic.query.all()
    return render_template(«manage_clinics.html», clinics=clinics)

@app.route(«/admin/delete_doctor/<int:doctor_id>«, methods=[«POST»])
@admin_required
def delete_doctor(doctor_id):
    doctor = Doctor.query.get_or_404(doctor_id)
    db.session.delete(doctor)
    db.session.commit()
    return redirect(url_for(«manage_doctors»))

@app.route(«/admin/delete_clinic/<int:clinic_id>«, methods=[«POST»])
@admin_required
def delete_clinic(clinic_id):
    clinic = Clinic.query.get_or_404(clinic_id)
    db.session.delete(clinic)
    db.session.commit()
    return redirect(url_for(«manage_clinics»))

@app.route(«/admin/add_doctor», methods=[«GET», «POST»])
@admin_required
def add_doctor():
    if not session.get(«is_admin»):
        abort(403)

    if request.method == «POST»:
        username = request.form[«username»]
        password = request.form[«password»]
```

```
        clinic_id = request.form[«clinic_id»]
        is_admin = «is_admin» in request.form

        doctor = Doctor(
            username=username,
            password_hash=generate_password_hash(password),
            first_name=request.form[«first_name»],
            last_name=request.form[«last_name»],
            patronymic=request.form.get(«patronymic»),
            clinic_id=clinic_id,
            is_admin=is_admin
        )
        db.session.add(doctor)
        db.session.commit()
        return redirect(url_for(«admin_dashboard»))

    clinics = Clinic.query.all()
    return render_template(«add_doctor.html», clinics=clinics)

if __name__ == «__main__»:
    app.run(debug=True)
```

Файл «add_clinic.html»

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <h1 class="section-header">Добавить клинику</h1>

    <div class="row justify-content-center">
        <div class="col-md-6">
            <div class="card card-hover">
                <div class="card-body">
                    <form method="POST">
                        <div class="mb-3">
                            <label class="form-label">Название
клиники</label>
                            <input type="text" name="name"
class="form-control" required>
                        </div>
                        <button type="submit" class="btn btn-cta w-
100">Добавить</button>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

Файл «add_doctor.html»

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <h1 class="section-header">Добавить врача</h1>
```

```
    <div class="row justify-content-center">
        <div class="col-md-6">
            <div class="card card-hover">
                <div class="card-body">
                    <form method="POST">
                        <div class="mb-3">
                            <label class="form-label">Логин</label>
                            <input type="text" name="username"
class="form-control" required>
                        </div>

                        <div class="mb-3">
                            <label class="form-label">Пароль</label>
                            <input type="password" name="password"
class="form-control" required>
                        </div>
                        <div class="mb-3">
                            <label class="form-label">Фамилия</label>
                            <input type="text" name="last_name"
class="form-control" required>
                        </div>

                        <div class="mb-3">
                            <label class="form-label">Имя</label>
                            <input type="text" name="first_name"
class="form-control" required>
                        </div>

                        <div class="mb-3">
                            <label class="form-label">Отчество</label>
                            <input type="text" name="patronymic"
class="form-control">
                        </div>
                        <div class="mb-3">
                            <label class="form-label">Клиника</label>
                            <select name="clinic_id" class="form-
select" required>
                                {% for clinic in clinics %}
                                    <option value="{{ clinic.id }}">{{
clinic.name }}</option>
                                {% endfor %}
                            </select>
                        </div>

                        <div class="mb-3 form-check">
                            <input type="checkbox" name="is_admin"
class="form-check-input" id="is_admin">
                            <label class="form-check-label"
for="is_admin">Администратор</label>
                        </div>

                        <button type="submit" class="btn btn-cta w-
100">Добавить</button>
```

```
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

Файл «admin_dashboard.html»

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <h1 class="section-header mb-4">Админ-панель</h1>

    <div class="row g-4">
        <div class="col-md-6">
            <div class="card card-hover shadow-sm">
                <div class="card-body text-center">
                    <h3>Управление клиниками</h3>
                    <p>Просмотр, добавление и удаление клиник</p>
                    <a href="{{ url_for(«manage_clinics») }}"
class="btn btn-primary mt-3">
                        Перейти к клиникам
                    </a>
                </div>
            </div>
        </div>

        <div class="col-md-6">
            <div class="card card-hover shadow-sm">
                <div class="card-body text-center">
                    <h3>Управление врачами</h3>
                    <p>Просмотр, добавление и удаление врачей</p>
                    <a href="{{ url_for(«manage_doctors») }}"
class="btn btn-primary mt-3">
                        Перейти к врачам
                    </a>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

Файл «base.html»

```
<!-- base.html -->
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>MedBooker - Онлайн запись к врачу</title>
```

```
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.
min.css" rel="stylesheet">
    <link
href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate
.min.css" rel="stylesheet">
    <style>
        :root {
            --primary: #2A5C82;
            --secondary: #5BA4E6;
            --accent: #FF7E67;
        }
        body {
            background: #f8f9fa;
            min-height: 100vh;
            display: flex;
            flex-direction: column;
        }
        .nav-gradient {
            background: linear-gradient(90deg, var(--primary) 0%,
var(--secondary) 100%);
        }
        .card-hover {
            transition: transform 0.3s, box-shadow 0.3s;
        }
        .card-hover:hover {
            transform: translateY(-5px);
            box-shadow: 0 10px 20px rgba(0,0,0,0.1);
        }
        .btn-cta {
            background: var(--accent);
            color: white;
            padding: 12px 30px;
            border-radius: 30px;
            transition: all 0.3s;
        }
        .btn-cta:hover {
            transform: scale(1.05);
            color: white;
        }
        .section-header {
            font-size: 2.5rem;
            color: var(--primary);
            position: relative;
            margin-bottom: 2rem;
        }
        .section-header:after {
            content: «»;
            position: absolute;
            bottom: -10px;
            left: 50%;
            transform: translateX(-50%);
            width: 60px;
```

```
            height: 3px;
            background: var(--accent);
        }
        .doctor-card {
            background: white;
            border-radius: 10px;
            padding: 20px;
            margin-bottom: 20px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
            transition: transform 0.2s;
        }

        .doctor-card:hover {
            transform: translateY(-3px);
        }
        .dropdown-menu {
            min-width: 150px;
            box-shadow: 0 0.5rem 1rem rgba(0, 0, 0, 0.15);
        }

        .dropdown-item {
            padding: 0.5rem 1rem;
            font-size: 0.9rem;
        }

        .dropdown-item:hover {
            background-color: #f8f9fa;
        }
    </style>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.8.1/font/bootstrap-icons.css">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark nav-gradient">
        <div class="container">
            <a class="navbar-brand animate__animated animate__fadeIn"
href="/">MedBooker</a>
            <div class="d-flex">
                <a href="{{ url_for(«doctor_login») }}" class="btn
btn-outline-light">Вход для врачей</a>
            </div>
        </div>
    </nav>

    <main class="flex-grow-1 py-5">
        {% block content %}{% endblock %}
    </main>

    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bu
ndle.min.js"></script>
    <script>
```

```
        // Простые анимации при загрузке
        document.querySelectorAll(«.animate-on-scroll»).forEach(el =>
{
            el.classList.add(«animate__animated»,
«animate__fadeInUp»);
        });
    </script>
</body>
</html>
```

Файл «confirmed_appointments.html»

```
<!-- templates/confirmed_appointments.html -->
{% extends "base.html" %}
{% block content %}
<div class="container">
    <h1 class="section-header mb-4">Подтвержденные записи</h1>

    <div class="row g-4">
        {% for appt in appointments %}
        <div class="col-md-6 animate-on-scroll">
            <div class="card card-hover border-success">
                <div class="card-body">
                    <div class="d-flex justify-content-between align-
items-center mb-3">
                        <h5 class="card-title">{{ appt.patient_name
}}</h5>
                        <span class="badge bg-success">{{
appt.status|upper }}</span>
                    </div>
                    <div class="mb-2">
                        <i class="bi bi-clock me-2"></i>
                        {{ appt.slot_start|datetimeformat }}
                    </div>
                    <div class="mb-2">
                        <i class="bi bi-telephone me-2"></i>
                        {{ appt.patient_phone }}
                    </div>
                    <div>
                        <i class="bi bi-envelope me-2"></i>
                        {{ appt.patient_email }}
                    </div>
                </div>
            </div>
        </div>
        {% else %}
        <div class="col-12">
            <div class="alert alert-info">Нет подтвержденных
записей</div>
        </div>
        {% endfor %}
    </div>
</div>
```

```
{% endblock %}
```

Файл «doctor_appointments.html»

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <h1 class="section-header mb-4">Ожидающие подтверждения</h1>

    <form method="POST">
        <div class="row g-4">
            {% for appt in appointments %}
            <div class="col-md-6 animate-on-scroll">
                <div class="card card-hover border-warning">
                    <div class="card-body">
                        <div class="form-check">
                            <input class="form-check-input"
type="checkbox"
                                   name="appointment_ids" value="{{
appt.id }}">
                        </div>
                        <div class="d-flex justify-content-between
align-items-center mb-3">
                            <h5 class="card-title">{{
appt.patient_name }}</h5>
                            <span class="badge bg-warning">{{
appt.status|upper }}</span>
                        </div>
                        <div class="mb-2">
                            <i class="bi bi-clock me-2"></i>
                            {{ appt.slot_start|datetimeformat }}
                        </div>
                        <div class="mb-2">
                            <i class="bi bi-telephone me-2"></i>
                            {{ appt.patient_phone }}
                        </div>
                        <div>
                            <i class="bi bi-envelope me-2"></i>
                            {{ appt.patient_email }}
                        </div>
                    </div>
                </div>
            </div>
            {% else %}
            <div class="col-12">
                <div class="alert alert-info">Нет записей для
подтверждения</div>
            </div>
            {% endfor %}
        </div>

        <div class="mt-4 text-center">
            <button type="submit" class="btn btn-cta">
```

```
                <i class="bi bi-check-all me-2"></i>Подтвердить
выбранные
            </button>
        </div>
    </form>
</div>
{% endblock %}
```

Файл «doctor_login.html»

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-6 col-lg-5">
            <div class="card card-hover animate-on-scroll mt-5">
                <div class="card-body p-5">
                    <h2 class="section-header text-center mb-4">Вход
для врачей</h2>

                    <form method="POST">
                        <div class="mb-3">
                            <label class="form-label">Логин</label>
                            <input type="text" name="username"
                                   class="form-control form-control-
lg" required>
                        </div>
                        <div class="mb-4">
                            <label class="form-label">Пароль</label>
                            <input type="password" name="password"
                                   class="form-control form-control-
lg" required>
                        </div>
                        <button type="submit" class="btn btn-cta w-
100">
                            Войти
                        </button>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

Файл «doctor_schedule.html»

```
<!-- templates/doctor_schedule.html -->
{% extends "base.html" %}
{% block content %}
<div class="container">
    <h1 class="section-header mb-4">Управление расписанием</h1>
```

```
    <div class="card card-hover mb-4">
        <div class="card-body">
            <h5 class="card-title">Фильтр по дате</h5>
            <form method="GET" class="row g-3">
                <div class="col-md-4">
                    <input type="date" name="date" value="{{
selected_date or «» }}" class="form-control" required>
                </div>
                <div class="col-md-8 d-flex gap-2">
                    <button type="submit" class="btn btn-
cta">Применить фильтр</button>
                    <a href="{{ url_for(«doctor_schedule») }}"
class="btn btn-secondary">Сбросить</a>
                </div>
            </form>
        </div>
    </div>

    <div class="card card-hover mb-4">
        <div class="card-body">
            <h5 class="card-title">Создать новые слоты</h5>
            <form method="POST" class="row g-3">
                <div class="col-md-3">
                    <input type="date" name="date" class="form-
control" required>
                </div>
                <div class="col-md-2">
                    <input type="time" name="start_time" class="form-
control" required>
                </div>
                <div class="col-md-2">
                    <input type="time" name="end_time" class="form-
control" required>
                </div>
                <div class="col-md-3">
                    <input type="number" name="slot_duration"
class="form-control"
                           placeholder="Длительность (мин)" min="15"
step="15" required>
                </div>
                <div class="col-md-2">
                    <button type="submit" class="btn btn-cta w-
100">Создать</button>
                </div>
            </form>
        </div>
    </div>

    <div class="d-flex gap-3 mb-4">
        <form method="POST" action="{{ url_for(«confirm_multiple») }}"
class="form-multi-action flex-grow-1">
            <button type="submit" class="btn btn-success w-100"
```

```
                    onclick="return confirm(«Подтвердить выбранные
записи?»)">
                <i class="bi bi-check-circle me-2"></i>Подтвердить
выбранные
            </button>
        </form>

        <form method="POST" action="{{
url_for(«delete_multiple_slots») }}" class="flex-grow-1">
            <button type="submit" class="btn btn-danger w-100"
                    onclick="return confirm(«Удалить выбранные
слоты?»)">
                <i class="bi bi-trash me-2"></i>Удалить выбранные
            </button>
        </form>
    </div>

    <div class="row g-4">
        {% for slot in schedules %}
        <div class="col-12 col-md-6 col-lg-4 animate-on-scroll">
            <div class="card card-hover
                {% if slot.is_past %}bg-light
                {% elif slot.appointment and slot.appointment.status
== «confirmed» %}border-success
                {% elif slot.appointment and slot.appointment.status
== «pending» %}border-warning
                {% else %}border-primary{% endif %}">

                <div class="card-header d-flex justify-content-between
align-items-center">
                    <div class="form-check">
                        <input class="form-check-input"
type="checkbox"
                               name="slot_ids" value="{{ slot.slot.id
}}">
                    </div>
                    <form method="POST" action="{{
url_for(«delete_slot», slot_id=slot.slot.id) }}">
                        <button type="submit" class="btn btn-sm btn-
danger"
                                {% if not slot.can_delete %}disabled{%
endif %}
                                onclick="return confirm(«Удалить
слот?»)">
                            <i class="bi bi-x-lg"></i>
                        </button>
                    </form>
                </div>

                <div class="card-body">
                    <div class="d-flex justify-content-between mb-3">
                        <h5 class="card-title">
```

```
                                {{
slot.slot.start_time|datetimeformat(«%H:%M») }} –
                                {{
slot.slot.end_time|datetimeformat(«%H:%M») }}
                            </h5>
                            <span class="badge
                                {% if slot.is_past %}bg-secondary
                                {% elif slot.appointment and
slot.appointment.status == «confirmed» %}bg-success
                                {% elif slot.appointment and
slot.appointment.status == «pending» %}bg-warning
                                {% else %}bg-primary{% endif %}">
                                {% if slot.is_past %}Прошедший
                                {% elif slot.appointment %}{{
slot.appointment.status|upper }}
                                {% else %}Свободен{% endif %}
                            </span>
                        </div>

                        {% if slot.appointment %}
                        <div class="alert
                            {% if slot.appointment.status == «confirmed»
%}alert-success
                            {% else %}alert-warning{% endif %}">
                            <div class="fw-bold">{{
slot.appointment.patient_name }}</div>
                            <div>{{ slot.appointment.patient_phone
}}</div>
                            <div>{{ slot.appointment.patient_email
}}</div>
                        </div>
                        {% endif %}
                    </div>
                </div>
            </div>
            {% endfor %}
        </div>
    </div>

    <script>
    document.addEventListener(«DOMContentLoaded», function() {
        const forms = document.querySelectorAll(«form.form-multi-action»);

        forms.forEach(form => {
            form.addEventListener(«submit», function(e) {
                const checkboxes = Array.from(
                    document.querySelectorAll(«.form-check-input:checked»)
                );

                if (checkboxes.length === 0) {
                    e.preventDefault();
                    alert(«Пожалуйста, выберите хотя бы один слот!»);
                    return;
```

```
            }

            checkboxes.forEach(checkbox => {
                const input = document.createElement(«input»);
                input.type = «hidden»;
                input.name = «slot_ids»;
                input.value = checkbox.value;
                form.appendChild(input);
            });
        });
    });
});

</script>
{% endblock %}
```

Файл «doctors.html»

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <h1 class="section-header text-center">Наши врачи</h1>

    <div class="row g-4">
        {% for doctor in doctors %}
        <div class="col-md-6 col-lg-4 animate-on-scroll">
            <div class="card card-hover h-100">
                <div class="card-body text-center">
                    <div class="avatar mx-auto mb-3"
                         style="width: 120px; height: 120px;
background: #eee; border-radius: 50%;"></div>
                    <h4 class="card-title">{{ doctor.full_name }}</h4>
                    <a href="{{ url_for(«patient_schedule»,
doctor_id=doctor.id) }}"
                       class="btn btn-cta">
                        Посмотреть расписание
                    </a>
                </div>
            </div>
        </div>
        {% endfor %}
    </div>
</div>
{% endblock %}
```

Файл «home.html»

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <h1 class="section-header text-center animate__animated
animate__fadeIn">Выберите клинику</h1>
```

```
    <div class="row g-4">
        {% for clinic in clinics %}
        <div class="col-md-4 animate-on-scroll">
            <div class="card card-hover h-100">
                <div class="card-body text-center">
                    <h3 class="card-title">{{ clinic.name }}</h3>
                    <a href="{{ url_for(«list_doctors») 
}}?clinic_id={{ clinic.id }}"
                       class="btn btn-cta mt-3">
                        Выбрать врача
                    </a>
                </div>
            </div>
        </div>
        {% endfor %}
    </div>
</div>
{% endblock %}
```

Файл «manage_clinics.html»

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <h1 class="section-header mb-4">Управление клиниками</h1>

    <div class="d-flex justify-content-between mb-4">
        <a href="{{ url_for(«add_clinic») }}" class="btn btn-cta">
            <i class="bi bi-plus-circle me-2"></i>Добавить клинику
        </a>
    </div>

    <div class="row g-4">
        {% for clinic in clinics %}
        <div class="col-md-6 col-lg-4">
            <div class="card card-hover shadow-sm">
                <div class="card-body">
                    <div class="d-flex justify-content-between align-
items-start">
                        <div>
                            <h5 class="card-title mb-1">{{ clinic.name
}}</h5>
                            <p class="text-muted small mb-0">
                                Врачей: {{ clinic.doctors|length }}
                            </p>
                        </div>
                        <div class="dropdown">
                            <button class="btn btn-link text-dark"
type="button"
                                    data-bs-toggle="dropdown">
                                <i class="bi bi-three-dots-
vertical"></i>
                            </button>
```

```
                            <ul class="dropdown-menu">
                                <li>
                                    <form method="POST"
                                        action="{{
url_for(«delete_clinic», clinic_id=clinic.id) }}">
                                        <button type="submit"
                                                class="dropdown-item
text-danger"
                                                onclick="return
confirm(«Удалить клинику?»)">
                                            <i class="bi bi-trash me-
2"></i>Удалить
                                        </button>
                                    </form>
                                </li>
                            </ul>
                        </div>
                    </div>
                </div>
            </div>
        </div>
        {% else %}
        <div class="col-12">
            <div class="alert alert-info">
                Нет зарегистрированных клиник
            </div>
        </div>
        {% endfor %}
    </div>
</div>
{% endblock %}
```

Файл «manage_doctors.html»

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <h1 class="section-header mb-4">Управление врачами</h1>

    <div class="d-flex justify-content-between mb-4">
        <a href="{{ url_for(«add_doctor») }}" class="btn btn-cta">
            <i class="bi bi-plus-circle me-2"></i>Добавить врача
        </a>
    </div>

    <div class="row g-4">
        {% for doctor in doctors %}
        <div class="col-md-6 col-lg-4">
            <div class="card card-hover shadow-sm">
                <div class="card-body">
                    <div class="d-flex justify-content-between align-
items-start mb-3">
                        <div>
```

```
                                <h5 class="card-title mb-1">{{
doctor.full_name }}</h5>
                                <p class="text-muted small mb-0">
                                    Логин: {{ doctor.username }}<br>
                                    Клиника: {{ doctor.clinic.name }}
                                </p>
                            </div>
                            <div class="dropdown">
                                <button class="btn btn-link text-dark"
type="button"
                                        data-bs-toggle="dropdown">
                                    <i class="bi bi-three-dots-
vertical"></i>
                                </button>
                                <ul class="dropdown-menu">
                                    <li>
                                        <form method="POST"
                                            action="{{
url_for(«delete_doctor», doctor_id=doctor.id) }}">
                                            <button type="submit"
                                                    class="dropdown-item
text-danger"
                                                    onclick="return
confirm(«Удалить врача?»)">
                                                <i class="bi bi-trash me-
2"></i>Удалить
                                            </button>
                                        </form>
                                    </li>
                                </ul>
                            </div>
                        </div>
                        <div class="form-check form-switch">
                            <input class="form-check-input"
type="checkbox"
                                id="adminSwitch{{ doctor.id }}"
                                {% if doctor.is_admin %}checked{% endif
%}
                                disabled>
                            <label class="form-check-label"
for="adminSwitch{{ doctor.id }}">
                                Администратор
                            </label>
                        </div>
                    </div>
                </div>
            </div>
            {% else %}
            <div class="col-12">
                <div class="alert alert-info">
                    Нет зарегистрированных врачей
                </div>
            </div>
```

```
        {% endfor %}
    </div>
</div>
{% endblock %}
```

Файл «patient_schedule.html»

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <div class="row justify-content-center">
        <div class="col-lg-8">
            <h1 class="section-header">Запись к {{
doctor.full_name}}</h1>

            <div class="timeline">
                {% for slot in slots %}
                <div class="card card-hover mb-3 animate-on-scroll">
                    <div class="card-body">
                        <div class="d-flex justify-content-between
align-items-center">
                            <div>
                                <h5 class="mb-0">
                                    {{ slot.start|datetimeformat(«%d
%b %Y») }}
                                </h5>
                                <span class="text-muted">
                                    {{
slot.start|datetimeformat(«%H:%M») }} – {{
slot.end|datetimeformat(«%H:%M») }}
                                </span>
                            </div>

                            {% if slot.pending %}
                                <div class="badge bg-warning">Ожидает
подтверждения</div>
                            {% elif slot.available %}
                                <button class="btn btn-primary"
                                        data-bs-toggle="modal"
                                        data-bs-target="#bookingModal"
                                        data-slot-id="{{ slot.id }}">
                                    Записаться
                                </button>
                            {% else %}
                                <div class="badge bg-
secondary">Занято</div>
                            {% endif %}
                        </div>
                    </div>
                </div>
                {% endfor %}
            </div>
        </div>
```

```
    </div>
</div>

<!-- Модальное окно записи -->
<div class="modal fade" id="bookingModal" tabindex="-1">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">Заполните данные</h5>
                <button type="button" class="btn-close" data-bs-
dismiss="modal"></button>
            </div>
            <form method="POST" action="{{ url_for(«patient_book») 
}}">
                <div class="modal-body">
                    <input type="hidden" name="slot_id" id="slotId">
                    <div class="mb-3">
                        <label class="form-label">Ваше полное
имя</label>
                        <input type="text" name="patient_name"
class="form-control" required>
                    </div>
                    <div class="mb-3">
                        <label class="form-label">Контактный
телефон</label>
                        <input type="tel" name="patient_phone"
class="form-control" required>
                    </div>
                    <div class="mb-3">
                        <label class="form-label">Email</label>
                        <input type="email" name="patient_email"
class="form-control" required>
                    </div>
                </div>
                <div class="modal-footer">
                    <button type="submit" class="btn btn-cta w-
100">Подтвердить запись</button>
                </div>
            </form>
        </div>
    </div>
</div>

<script>

document.getElementById(«bookingModal»).addEventListener(«show.bs.moda
l», function(e) {
        const button = e.relatedTarget;
        const slotId = button.getAttribute(«data-slot-id»);
        document.getElementById(«slotId»).value = slotId;
    });
</script>
{% endblock %}
```

Приложение С

Система автоматизации работы заведений общественного питания

Презентация

Управление расписанием пациентов для медицинских учреждений.

Автор работы:
Савосин В. А.
(уч. гр. 21ВВП2)

Научный руководитель:
Зинкин С. А.

Пензенский государственный университет
2025 г.

Слайд – 1

- Проблема и актуальность проекта

Медучреждения тратят до 70% времени администраторов на ручное составление расписаний, что вызывает ошибки и недовольство пациентов. Автоматизация решает эти проблемы, улучшая точность и доступность услуг в рамках цифровизации здравоохранения России.

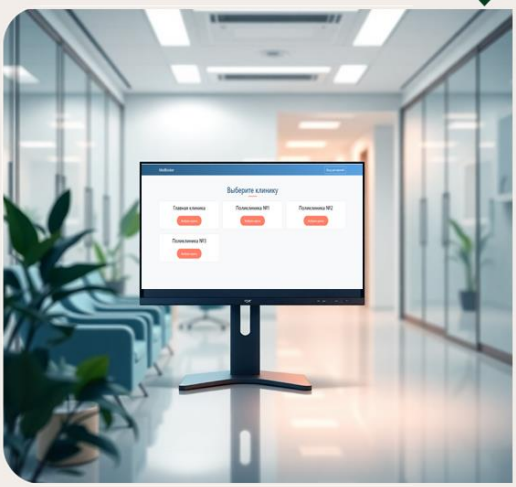
2

Слайд – 2

Цели и задачи проекта

MedBooker автоматизирует управление расписаниями, минимизируя ручную работу и повышая производительность. Для пользователей используются передовые технологии, чтобы обеспечить понятное эффективное функционирование системы.

Включает анализ требований и проектирование архитектуры с использованием MVC и REST API.



3

Слайд – 3

Анализ существующих решений

Сравнение решений для управления медицинскими расписаниями в отношении стоимости, сроков внедрения и интерфейса. MedBooker предлагает простоту внедрения и использование, что выгодно в сравнении с конкурентами.

Критерий	MedBooker	РТ МИС	МедСофт
Стоимость	Бесплатно	500к+ ₽	300к+ ₽
Время внедрения	7 дней	1 мес+	2 мес+
Интерфейс пациента	✅	❌	❌
Требования к ПО	Браузер	Сервер	Сервер

Внутренний анализ рынка 2025

4

Слайд – 4

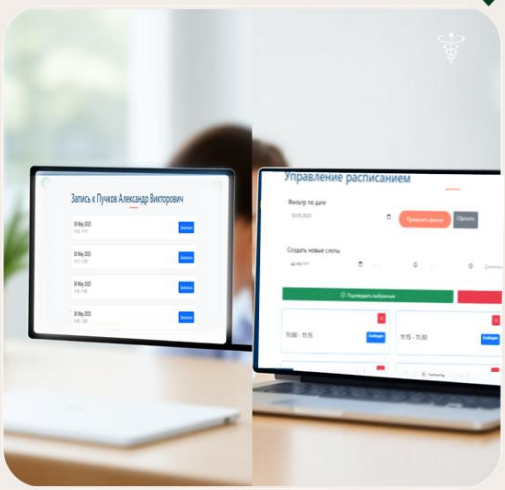
Функционал системы MedBooker

1 Функции для пользователей

Пациенты могут просматривать свободные слоты и записываться на прием онлайн. Врачи управляют своим расписанием и подтверждают заявки. Администраторы имеют возможность редактировать данные и следить за загруженностью.

2 Удобство использования

Интерфейс MedBooker интуитивно понятен, что упрощает использование для всех категорий пользователей. Цветовая индикация статусов записей позволяет быстро ориентироваться в большинстве рабочих процессов.



5

Слайд – 5

Тестирование и результаты

Итоги производительности и надежности системы по результатам пилотного внедрения.
Система стабильно работает при высокой нагрузке, поддерживая быстрое время отклика и снижая расходы.

Метрика	Значение
Время отклика	0.4 сек (среднее)
Нагрузка	1000 RPM (запросов в минуту)
Ошибки	<0.1%
Время планирования	↓ на 78%
Ошибки записи	↓ на 92%
Удовлетворенность персонала	4.8/5

6

Слайд – 6

Охрана труда и эргономика

Внедрение MedBooker соответствует всем стандартам СанПиН, создавая безопасные и удобные условия труда. Автоматизация административных задач помогает сократить бумажную работу и соблюсти баланс нагрузки сотрудников.

Посредством системы уменьшается стресс у медицинского персонала, связанный с рутинной работой. Снижение бумажного документооборота также приводит к экологическим преимуществам и оптимизации рабочего времени врачей.

7

Слайд – 7

Экономическая эффективность проекта

1 **Финансовые выгоды**

MedBooker предоставляет поликлиникам ощутимую финансовую выгоду, сокращая затраты на административные процессы и лицензионное ПО. Экономия свыше 600 000 ₽ в год обусловлена снижением трудозатрат и усовершенствованием процессов управления.

2 **Социальные преимущества**

Проект улучшает качество обслуживания за счёт уменьшения времени ожидания и повышения доступности медицинских услуг. Улучшенная прозрачность процесса записи способствует увеличению уровня доверия и удовлетворенности пациентов.

8

Слайд – 8

- Будущие направления развития

Расширение функционала мобильного приложения обеспечит более глубокую интеграцию с пользователями, предоставляя push-уведомления о новых записях, а также рейтинг врачей и поликлиник.

Внедрение искусственного интеллекта позволит более точно прогнозировать нагрузку на медучреждения, что оптимизирует распределение ресурсов. Электронные медицинские карты не только улучшат аналитику, но и повысят безопасность и доступность медицинских данных.

9

Слайд – 9

- Итоги и благодарности

MedBooker отвечает требованиям ТЗ и доказал свою эффективность. Благодарю научного руководителя и кафедру за их поддержку и помощь в реализации проекта.

Контактные данные:

4epviak@gmail.com

@V0L4APA

10

Слайд – 10