Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина «Распределенные системы обработки информации»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  старший преподаватель кафедры экономической информатики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. П. Лыщик |
|  | \_\_\_.\_\_\_\_.2023 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«Разработка Автоматизированной системы частной клиники»**

|  |  |
| --- | --- |
|  | Выполнил студент группы 074002  Волчок Анастасия Васильевна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2023  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2023

**СОДЕРЖАНИЕ**

[Перечень условных обозначений, символов и терминов 5](#_Toc103259189)

[Введение 7](#_Toc103259190)

[1. Анализ и моделирование предметной области автоматизированной системы 9](#_Toc103259191)

[1.1 Описание деятельности частной клиники 9](#_Toc103259192)

[1.2 Разработка функциональной модели предметной области 11](#_Toc103259193)

[1.3 Анализ требований к разрабатываемой автоматизированной системе 15](#_Toc103259194)

[1.4 Модели представления автоматизированной системы и их описание 19](#_Toc103259196)

[1.5.1 Диаграмма последовательности 19](#_Toc103259197)

[1.5.2 Диаграмма состояний 21](#_Toc103259198)

[1.5.3 Диаграмма вариантов использования 21](#_Toc103259199)

[1.5.4 Диаграмма развертывания 22](#_Toc103259200)

[1.5.5 Диаграмма классов 23](#_Toc103259201)

1.5 Разработка информационной модели области управления работой частной клиники 17

[2. Проектирование и конструирование автоматизированной системы 25](#_Toc103259202)

[2.1 Постановка задачи 25](#_Toc103259203)

[2.2 Архитектурные решения 25](#_Toc103259204)

[2.3 Описание алгоритмов, реализующих бизнес-логику разрабатываемой автоматизированной системы 27](#_Toc103259205)

[2.4 Проектирование пользовательского интерфейса 30](#_Toc103259206)

[2.5 Обоснование выбора компонентов и технологий для реализации автоматизированной системы 30](#_Toc103259207)

[3. Тестирование и проверка работоспособности автоматизированной системы 33](#_Toc103259208)

[4. Руководство по развертыванию, использованию автоматизированной системы 35](#_Toc103259209)

[Заключение 42](#_Toc103259210)

[Список использованных источников 43](#_Toc103259211)

[ПРИЛОЖЕНИЕ А 44](#_Toc103259212)

[ПРИЛОЖЕНИЕ Б 45](#_Toc103259213)

[ПРИЛОЖЕНИЕ В 53](#_Toc103259214)

# **ВВЕДЕНИЕ**

Современные частные клиники стремятся обеспечить высокое качество медицинских услуг, используя передовые технологии и надежное оборудование. Сейчас информационные технологии играют важную роль в частных клиниках. Одним из основных применений информационных технологий является внедрение систем управления клиникой. Эти системы позволяют автоматизировать все процессы, начиная от приема пациентов и заканчивая выпиской рецептов и формированием отчетности. Они позволяют контролировать запасы медикаментов и оборудования, а также управлять финансами клиники.

Например, наличие автоматизированной системы для записи на прием позволяет пациентам записываться к врачу через интернет или по телефону, что упрощает процесс и сокращает время ожидания. Также такие системы позволяют контролировать загруженность врачей и распределять нагрузку между ними.

В целом, применение информационных технологий, в частности автоматизированных систем, в частных клиниках позволяет оптимизировать все процессы, связанные с приемом и лечением пациентов, повысить качество обслуживания и сократить затраты на обслуживание. Это делает информационные технологии необходимым инструментом для эффективной работы современных частных клиник.

Из выше представленного можно сделать вывод об необходимости разработки автоматизированной системы частной клиники.

*Целью курсового проекта* является оптимизация всех процессов, связанных с записью на прием пациентов, улучшение качества обслуживания и сокращение затраты на него, путем внедрения автоматизированной системы.

Поставленная цель требует решения следующих задач:

* анализ деятельности частной клиники;
* разработка функциональной модели предметной области;
* определение ключевых требований, предъявляемых к автоматизированной системе, а также способов их исполнения;
* проектирование информационной модели, хранящей данные о работе частной клиники;
* моделирование представлений автоматизированной системы;
* проектирование и структурирование автоматизированной системы;
* тестирование и проверка работоспособности каждого модуля автоматизированной системы;
* предоставление пользователю подробного руководства по взаимодействию с автоматизированной системой, включающего инструкцию по развертыванию, а также сквозные примеры функционала.

*Объектом исследования курсового проекта* является частная клиника, осуществляющая консультативные, терапевтические, диагностические и хирургические услуги.

*Предметом исследования курсового проекта* являются способы и пути оптимизации всех процессов, связанных с записью на прием пациентов, улучшение качества обслуживания и сокращение затраты на него.

1. Анализ и моделирование предметной области автоматизированной системы
   1. Описание деятельности частной клиники

Рассмотрение предметной области «Частная клиника» обосновано развитием частных клиник и повышению спроса на платные медицинские услуги. Перечень предоставляемых услуг данными клиниками очень широкий и разнообразный. Повышение спроса означает увеличение количества клиентов. Из этих факторов возникает необходимость в оптимизации и структурировании работы частных клиник. Например, ведение учета записей на прием, учет клиентов, графики работы, отделений и врачей, для всего этого необходима упорядоченная система, чтобы избежать утерю данных и большие временные и денежные затраты на выборку данных.

Перечень услуг, предоставляемых частными клиниками:

* педиатрия;
* эндокринология;
* неврология;
* гастроэнтерология;
* кардиолог;
* оториноларингология;
* дерматология;
* гематология;
* ревматология;
* офтальмология;
* травматология;
* пульмонология;
* хирургия;
* УЗИ;
* МРТ;
* другие.

Запись на данные услуги осуществляется по простому алгоритму. Клиент регистрируется в системе, выбирает необходимую ему услугу, по желанию специалиста, дату и время, затем отправляет свой запрос и при необходимости с ним связывается оператор клиники. Оплата проходит как на месте, так и онлайн. Все это помогает экономить время и силы как клиенту, так и работникам клинки. Пользователь после записи и самого приема также остается в базе данных клиники и некоторые из них даже предоставляют скидку постоянным клиентам. Также предоставляются онлайн-консультации, например, когда клиент не уверен к какому специалисту лучше обратится.

На данный момент частные клиники довольно распространены и популярны. Постоянные и новые клиенты отмечают большое количество их преимуществ:

1. Своевременность и доступность медицинской помощи;
2. Сочетание контроля за работой организации от государства и наличие постоянной конкуренции;
3. Использование передового технического оборудования и современных информационных технологий;
4. Отсутствие очередей в часы приема;
5. Специалисты непрерывно совершенствуют свою базу знаний и квалификацию;
6. Предоставление отличного сервиса и комфортного пребывания в клинике;
7. Доктора, специалисты уделяют больше внимания к состоянию здоровья своих пациентов.

Из-за данных преимуществ люди стали чаще обращаться в частные клиники, чем в государственные.

**1.2 Разработка функциональной модели предметной области**

Функциональная модель IDEF0 представляет собой набор блоков, каждый из которых представляет собой «черный ящик» со входами и выходами, управлением и механизмами, которые детализируются (декомпозируются) до необходимого уровня. Наиболее важная функция расположена в верхнем левом углу. А соединяются функции между собой при помощи стрелок и описаний функциональных блоков. При этом каждый вид стрелки или активности имеет собственное значение. Данная модель позволяет описать все основные виды процессов, как административные, так и организационные. Стрелки могут быть:

* Входящие – вводные, которые ставят определенную задачу.
* Исходящие – выводящие результат деятельности.
* Управляющие (сверху-вниз) – механизмы управления (положения, инструкции и пр).
* Механизмы (снизу-вверх) – что используется для того, чтобы произвести необходимую работу.

Входящие и исходящие стрелки точнее было бы называть вводящими и выводящими, так как по-английски они называются Input и Output соответственно. Но особенности перевода и привычные названия выглядят уже так, как сложилось. И все же для правильного понимания терминов важно помнить их значение в данном случае. Это подтверждается еще и тем, что данная нотация создана прежде всего для разработки ПО, и термины переводить правильнее в этой точки зрения.

Стрелки подписываются при помощи имен существительных (опыт, план, правила), а блоки – при помощи глаголов, т.е. в них описываются действия, которые производятся (создать товар, заключить договор, произвести отгрузку). IDEF0 – это очень простой и одновременно наглядный язык описания бизнес-процессов. С помощью этого стандарта возможна передача информации между разработчиками, консультантами и пользователями. Стандарт очень тщательно разрабатывался, он удобен для проектирования, универсален.

Для работы с ним существует множество инструментов, например, VISIO, BPWIN, ERWIN, Bussines studio и т.д. Кроме того, использование для создания бизнес-моделей IDEF0 — это не только удобно, это еще и правильно. Этот инструмент был разработан для бизнес-аналитики, он прошел длительную и тщательную отладку и шлифовку.

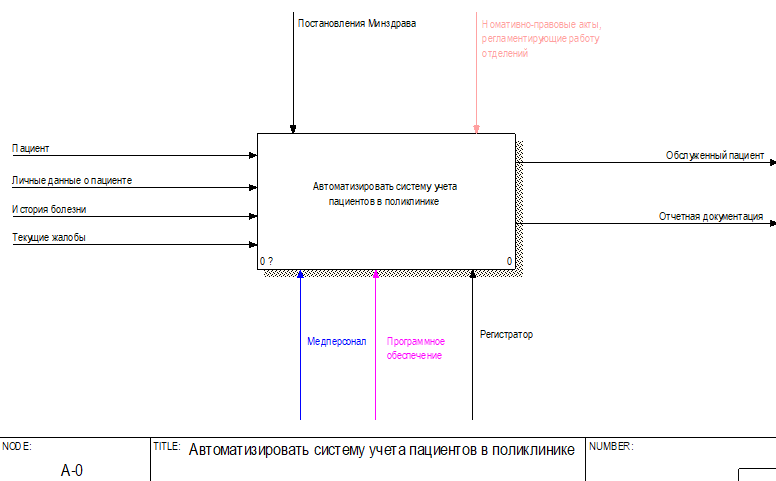


Рисунок 1.1 — Контекстная диаграмма верхнего уровня

Основной блок – это «Автоматизировать систему учёта пациентов в поликлиннике.

Входные данные – «Личные данные о пациенте», «История болезни», «Текущие жалобы», «Пациент».

В качестве управления выступают «Постановления Минздрава», «Нормативно-правовые акты, регламентирующие работу отделений».

Механизмом служат «Регистратор», «Программное обеспечение», «Медперсонал».

На выходе имеем «Обслуженный пациент», «Отчётная документация».

На рисунке 1.2 изображена декомпозиция функциональной модели.



Рисунок 1.2 – Декомпозиции контекстной диаграммы

Первый компонент «Предварительная запись пациента» означает, что пациент заранее узнает о предварительной дате записи на прием.

Второй компонент «Оформление талона» подразумевает, что пациент самостоятельно регистрирует свой талон в системе.

Третий компонент «Проведение приёма» значит, что пациенту сначала назначают обследования, проводят осмотр, а после ставят диагноз и назначают лечение.

Четвёртый компонент «Оформление отчёта и выписки» говорит о том, что пациенту уже были оказаны все необходимые услуги и по результатам лечения должен быть сформирован отчёт и запись в медицинской карте пациента.

На рисунке 1.3 представлена декомпозиция компонента «Предварительная запись пациента».

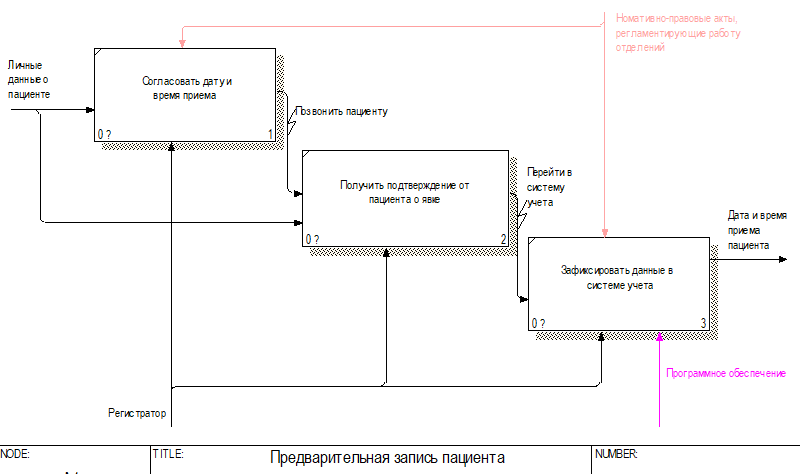


Рисунок 1.3 – Декомпозиция компонента «Предварительная запись пациента».

На рисунке 1.4 представлена декомпозиция компонента «Оформление талона».

Компонент включает в себя 3 блока: «Найти необходимого специалиста», «Подобрать дату и время приема», «Оформить талон».

В качестве входной информации используем «Дата и время приёма пациента», «Текущие жалобы». С помощью механизма «Программное обеспечение» подбирается необходимый специалист, его данные и данные, полученные раннее, заносятся вместе в систему учёта. В результате формируется талон.

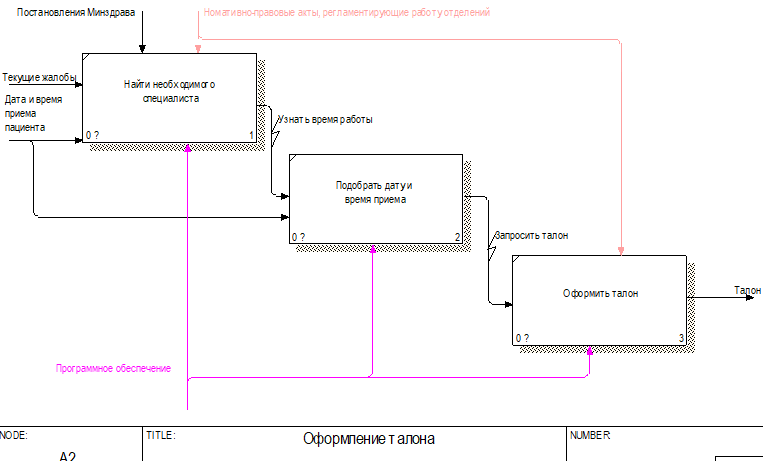


Рисунок 1.4 – Декомпозиция компонента «Оформление талона»

На рисунке 1.5 представлена декомпозиция компонента «Провести приём».

Компонент включает в себя 4 основных блока: «Провести осмотр пациента», «Собрать анализы у пациента», «Поставить диагноз», «Назначить препараты и процедуры».

Для того, чтобы провести осмотр пациента и собрать у него анализы, необходим талон, история болезни (Медицинская карта), данные о самочувствии пациента и сам пациент.

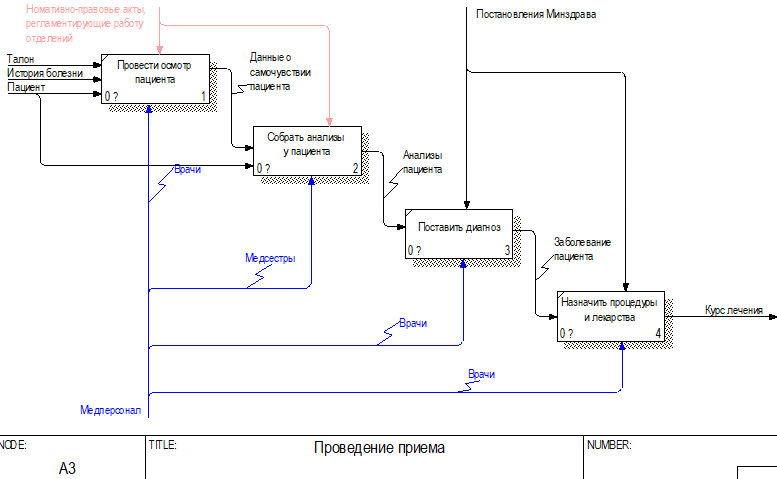


Рисунок 1.5 – Декомпозиция компонента «Провести приём».

Таким образом данная нотация IDEF0 показывает все функции системы взаимодействие и связи между ними. Также отображается структура системы, потоки информации и материальные объекты, связывающие эти функции.

**1.3 Анализ требований к разрабатываемой автоматизированной системе**

**UML** – унифицированный язык моделирования (Unified Modeling Language), система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем.

Словарь UML включает три вида строительных блоков:

* Диаграммы;
* Сущности.
* Связи.

Сущности являются основными объектно-ориентированными элементами языка. С их помощью можно создавать корректные модели. Структурные сущности – это имена существительные в моделях на языке UML. Как правило, они представляют собой статические части модели, соответствующие концептуальным или физическим элементам системы. Поведенческие сущности описывают некоторое действие или событие систем.

Диаграммы UML подразделяют на два типа — это структурные диаграммы и диаграммы поведения.

Структурные диаграммы показывают статическую структуру системы и ее частей на разных уровнях абстракции и реализации, а также их взаимосвязь. Элементы в структурной диаграмме представляют значимые понятия системы и могут включать в себя абстрактные, реальные концепции и концепции реализации. Существует семь типов структурных диаграмм:

* Диаграмма развертывания
* Диаграмма составной структуры
* Диаграмма пакетов
* Диаграмма профилей
* Диаграмма классов
* Диаграмма объектов
* Диаграмма компонентов

Диаграммы поведения показывают динамическое поведение объектов в системе, которое можно описать, как серию изменений в системе с течением времени. А к диаграммам поведения относятся:

* Диаграмма прецедентов
* Диаграмма деятельности
* Диаграмма состояний
* Диаграмма последовательности
* Диаграмма коммуникаций
* Диаграмма обзора взаимодействия
* Временная диаграмма

Диаграмма вариантов использования – это диаграмма унифицированного языка моделирования (UML) для выявления требований. Диаграммы вариантов использования предоставляют графическое представление целей (смоделированных вариантами использования), которых пользователи (представленные действующими лицами) хотят достичь с помощью системы. Варианты использования на диаграмме вариантов использования могут быть организованы и упорядочены в соответствии с их релевантностью, уровнем абстракции и влиянием на пользователя. Их можно связать, чтобы показать их зависимости, включить, расширить, создать отношения. Вариант использования описывает конкретную бизнес-цель, которой должна соответствовать создаваемая система. Графически это овал с простым названием, но это наиболее распространенный инструмент, используемый для управления бизнес-целями или задачами проекта.

Диаграмма вариантов использования программного продукта представлена на рисунке 3.1. Диаграмма и описание к следующей опроцентовке.

**1.5Модели представления автоматизированной системы и их описание**

Для описания представления системы используется язык UML.

UML – язык [графического](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B7%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) описания для [объектного моделирования](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE%D0%B5_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5&action=edit&redlink=1) в области [разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), для [моделирования бизнес-процессов 1](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D0%B2), [системного проектирования](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и отображения [организационных структур](https://ru.wikipedia.org/wiki/%D0%9E%D1%80%D0%B3%D0%B0%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0).

UML является языком широкого профиля, это — [открытый стандарт](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82), использующий графические обозначения для создания [абстрактной модели](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) [системы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, [программных систем](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0). UML не является [языком программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), но на основании UML-моделей возможна [генерация кода](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4%D0%BE%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F).

### **1.4.1 Диаграмма последовательности**

Диаграмма последовательности (Sequence Diagram) – диаграмма, на которой показаны взаимодействия объектов, упорядоченные по времени их проявления.

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники), вертикальные линии (англ. lifeline), отображающие течение времени при деятельности объекта, и стрелки, показывающие выполнение действий объектами. На данной диаграмме объекты располагаются слева направо. Ее недостатком является то, что она занимает много места.

Диаграммы последовательности, описывающие сценарии Business Use Case в виде последовательности обмена сообщениями между объектами - действующими лицами и объектами-исполнителями. Такие диаграммы помогают явно определить в модели обязанности каждого исполнителя в виде набора операций класса.

На диаграмме последовательности изображаются только те объекты, которые непосредственно участвуют во взаимодействии. Ключевым моментом для диаграмм последовательности является динамика взаимодействия объектов во времени.

В UML диаграмма последовательности имеет как бы два измерения. Первое слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия. Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый очередностью или степенью активности объектов при взаимодействии друг с другом.

Графически каждый объект изображается прямоугольником и располагается в верхней части своей линии жизни. Внутри прямоугольника записываются имя объекта и имя класса разделенные двоеточием. При этом вся запись подчеркивается, что является признаком объекта.

Вторым измерением диаграммы последовательности является вертикальная временная ось, направленная сверху вниз. Начальному моменту времени соответствует самая верхняя часть диаграммы. Взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения, а их порядок определяется временем возникновения. То есть, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже. Масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа «раньше-позже». Диаграмма представлена на рисунке 1.6

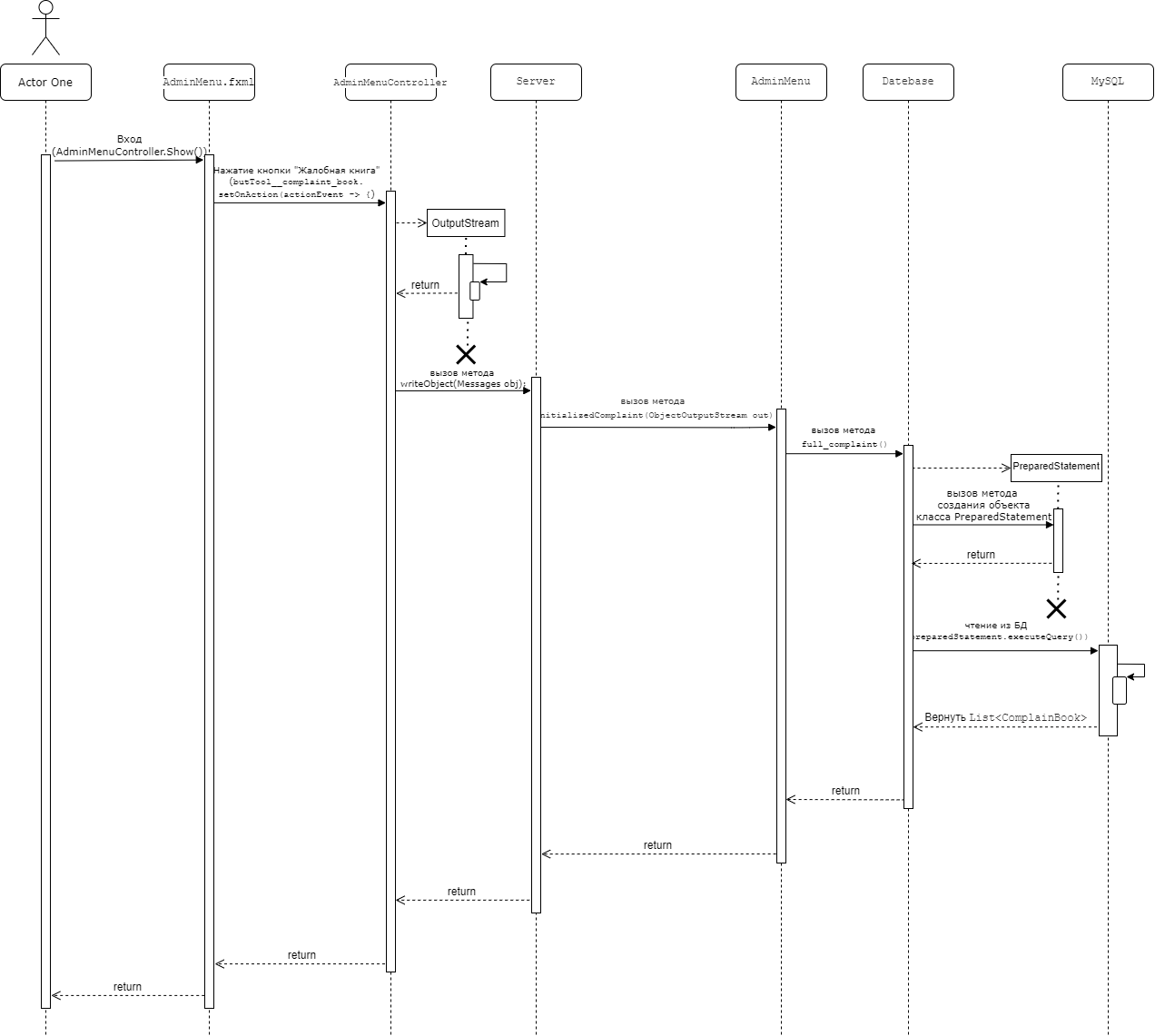


Рисунок 1.6 – Диаграмма последовательности

Сначала администратор входит в систему, где видит интерфейс со специальным окном «Жалобная книга». Далее после того, как он выбрал этот пункт, контроллер сцены обрабатывает данное действие, вызывая метод клиента для посыла информации на сервер. В выходной буфер помещается вся необходимая информация, а именно id пользователя, жалоба и ФИО пользователя и при помощи метода writeObject(Messages obj) информация отправляется на сервер TCP\IP. На сервере вызывается метод initializedComplain(ObjectOutputStream out). Далее вызывается метод full\_complaint(), создается объект класса PreparedStatement, вызывается метод для изменения таблицы в БД. Он вернет число успешно измененных строк в таблице, а затем клиенту будет возвращен List<ComplaintBook>, содержащий все жалобы.

### **1.4.2 Диаграмма состояний**

Диаграмма состояний показывает, как объект переходит из одного состояния в другое. Очевидно, что диаграммы состояний служат для моделирования динамических аспектов системы. Диаграмма состояний полезна при моделировании жизненного цикла объекта. От других диаграмм диаграмма состояний отличается тем, что описывает процесс изменения состояний только одного экземпляра определенного класса - одного объекта, причем объекта реактивного, то есть объекта, поведение которого характеризуется его реакцией на внешние события. Понятие жизненного цикла применимо как раз к реактивным объектам, настоящее состояние (и поведение) которых обусловлено их прошлым состоянием. Но диаграммы состояний важны не только для описания динамики отдельного объекта. Они могут использоваться для конструирования исполняемых систем путем прямого и обратного проектирования.

Скругленные прямоугольники представляют состояния, через которые проходит объект в течение своего жизненного цикла. Стрелками показываются переходы между состояниями, которые вызваны выполнением методов описываемого диаграммой объекта. Существует также два вида псевдосостояний: начальное, в котором находится объект сразу после его создания (обозначается сплошным кружком), и конечное, которое объект не может покинуть, если перешел в него (обозначается кружком, обведенным окружностью). При переходах указывается условие, при котором переход выполняется и, возможно, через символ / может указываться выполняемое при переходе действие. Кроме того, деятельность может указываться и в самом состоянии. Эта деятельность может позразделяться на деятельность при входе (entry/), текущая деятельность (do/) и деятельность при выходе(exit/). На рисунке 3.9 приведена диаграмма состояний для таймера. Диаграмма состояний представлена на рисунке 1.

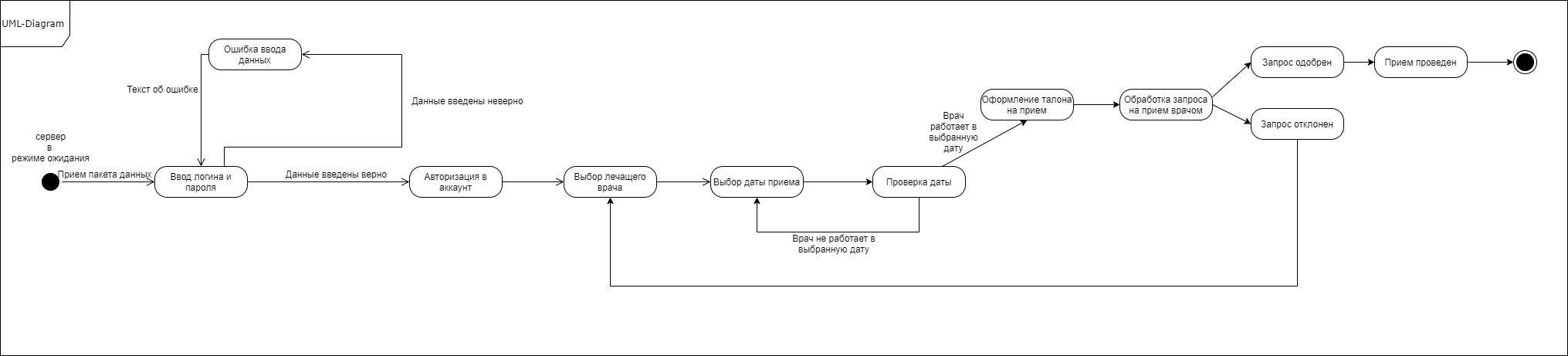


Рисунок 1.7 – Диаграмма состояний

**1.4.3 Диаграмма вариантов использования**

Диаграмма вариантов использования (англ. use-case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей. Данная диаграмма находится на рисунке 1.8



Рисунок 1.8 – Диаграмма вариантов использования

На данной диаграмме представлены 2 роли: администратор, врач и пациент. Для начала необходимо пройти регистрацию, то есть ввести личные данные, а после войти, для этого нужно всего ввести логин и пароль. У каждой роли свой функционал, но у них есть и общие функции: просмотреть информацию в своем личном профиле, изменить ее.

Администратор преимущественно работает с контролем функционирования поликлиники: просматривает профили пользователей, блокирует, просматривает статистику, составляет отчеты и регистрирует новых врачей и администраторов.

У пациента функционал основывается на бронировании талонов и написании отзывов/жалоб. Для того, чтобы заказать талон пациенту необходимо выбрать врача, а затем выбрать дату, в которую этот врач работает. Также есть возможность прочесть отзывы других пациентов о посещении специалистов представленного отделения. После приема пациент может написать свой отзыв и выставить оценку

### **1.4.4 Диаграмма развертывания**

Диаграмма развертывания (deployment diagram) - диаграмма, на которой представлены узлы выполнения программных компонентов реального времени, а также процессов и объектов.

Диаграмма развертывания применяется для представления общей конфигурации и Диаграмма вариантов использования (англ. use-case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей.по отдельным узлам системы. Кроме того, диаграмма развертывания показывает наличие физических соединений - маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих только на этапе ее исполнения (run-time). При этом представляются только те компоненты программы, которые являются исполнимыми файлами или динамическими библиотеками. Компоненты, не используемые на этапе исполнения, на диаграмме развертывания не показываются. Так, компоненты с исходными текстами программ могут присутствовать только на диаграмме компонентов. На диаграмме развертывания они не указываются.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления, диаграмма развертывания является единственной для системы в целом, поскольку должна отражать все особенности ее реализации. Диаграмма развертывания представлена на рисунке. Диаграмма развертывания и компонентов системы учета пациентов в поликлинике представлена на рисунке 5.3.

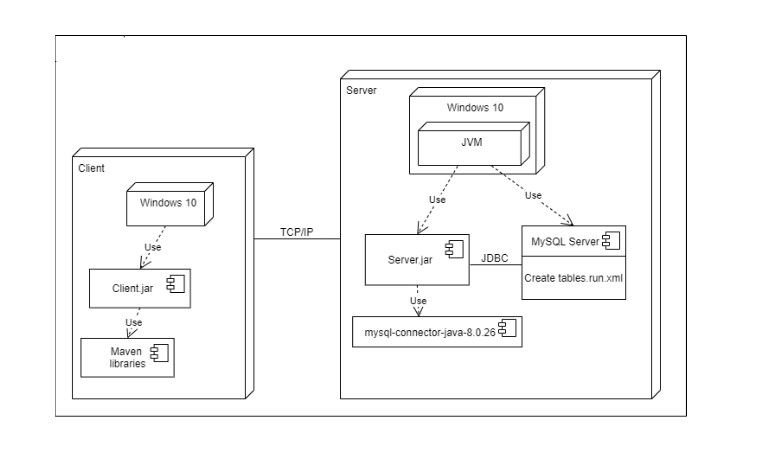


Рисунок 1.9 – Диаграмма развертывания и компонентов

На диаграмме изображены два узла: Client и Server, так же присутствуют устройства, использованные данными узлами. Client – использует Windows 10, и требует единственный компонент под названием «Сlient.jar», который в свою очередь использует библиотеки Maven. Server – использует Windows 10 вместе с установленной на ней JVM. Для работы также необходим Server.jar, который использует mysqlconnector-java:8.0.26 jar для подключения к СУБД, а также MySQL Server в качестве СУБД, включающий в себя скрипт создания таблиц в базе данных. Таким образом, была рассмотрена диаграмма развертывания с подробным описанием всех компонентов.

**1.4.5 Диаграмма классов**

Диаграмма классов – структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, из коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними.

В программе были использованы некоторые паттерны проектирования. Паттерны проектирования (шаблоны проектирования) – это готовые к использованию решения, часто возникающие в программировании задач.

Синглтон (singleton) это класс, у которого экземпляр создаётся только один раз. Для реализации синглтона нужно создать открытый статический метод, а поле закрытым. Данный паттерн был реализован в классе Client. Как видно на рисунке 1.10 в классе присутствует приватное поле singleton и публичный метод getInstance().

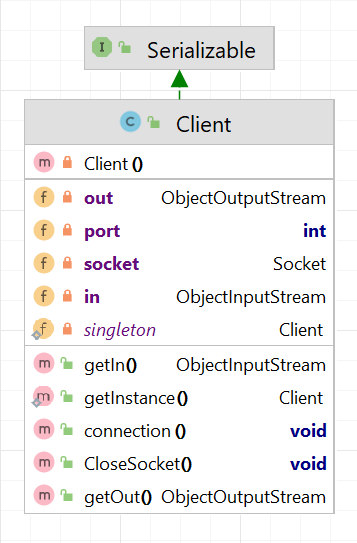


Рисунок 1.10 – Класс Client

Помимо singleton’a был также использован паттерн Builder. Строитель – это порождающий паттерн, который позволяет создавать сложные объекты пошагово. Строитель дает возможность использовать один и тот же код строительства для получения разного представления объектов. В данном курсовом проекте строитель представлен классом и интерфейсом, отображенным на рисунке 1.11:

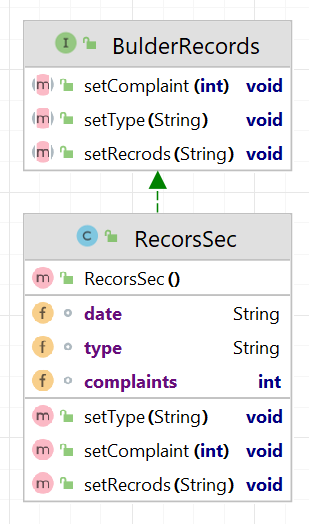


Рисунок 1.11 – Класс и интерфейс строителя

Взаимодействие между сервером и клиентом происходит благодаря классам Client, MainClient, Server и MainServer (рисунок 1.12).

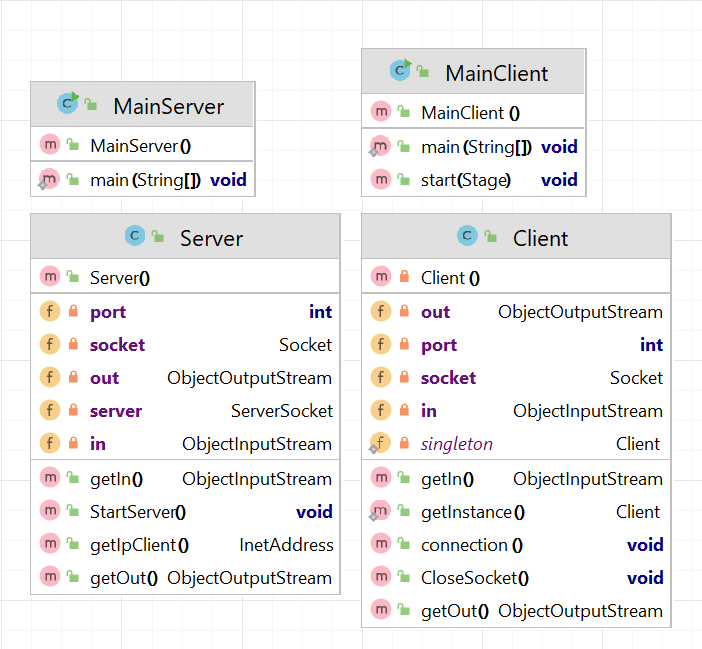


Рисунок 1.12 – Основные классы клиента и сервера

Что касается клиентской части, то для отображения окон используются классы-контроллеры. Именно они отвечают за смены сцен и отображение интерфейса. Также эти классы отвечают за прием объектов с сервера и отправку объектов на него. Диаграммы контроллеров представлены на рисунке 1.13:

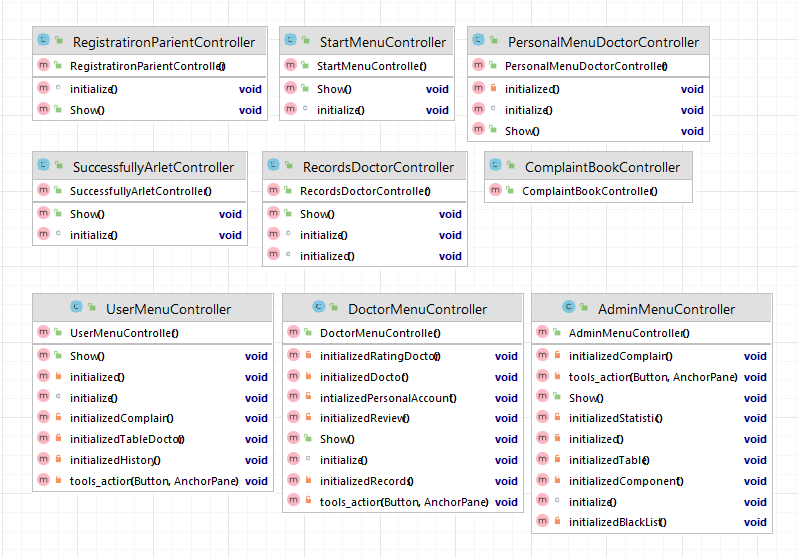


Рисунок 1.13 – Контроллеры клиента

В пакете config содержаться вспомогательные классы, которые отображены на рисунке 1.14. Они облегчают работу, а также помогают проверять корректность ввода пользователем. За проверки ввода отвечает класс FormValidator.

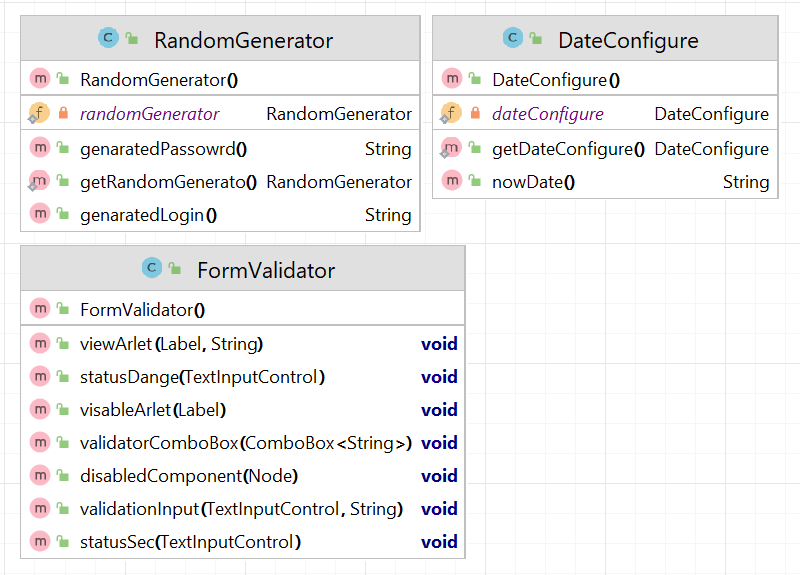


Рисунок 1.14 – Классы пакета config

В пакете message находятся перечисление Messages и класс UserStatus (рисунок 1.15). Messages содержит в себе поля, благодаря которым происходит взаимодействие клиента и сервера. Класс UserStatus задает роль пользователя, а также определяет заблокирован ли он.

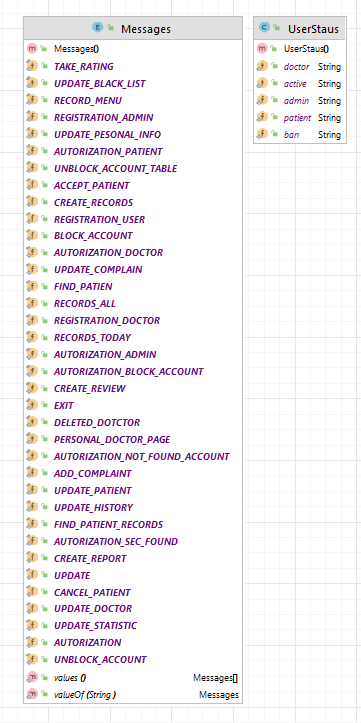


Рисунок 1.15 – Содержимое пакета message

Пакет models, классы которого представлены на рисунке 1.16, является идентичным как в клиентской части приложения, так и для сервера. В нем находятся все сущности. Информация об этих сущностях хранится и обрабатывается в базе данных.

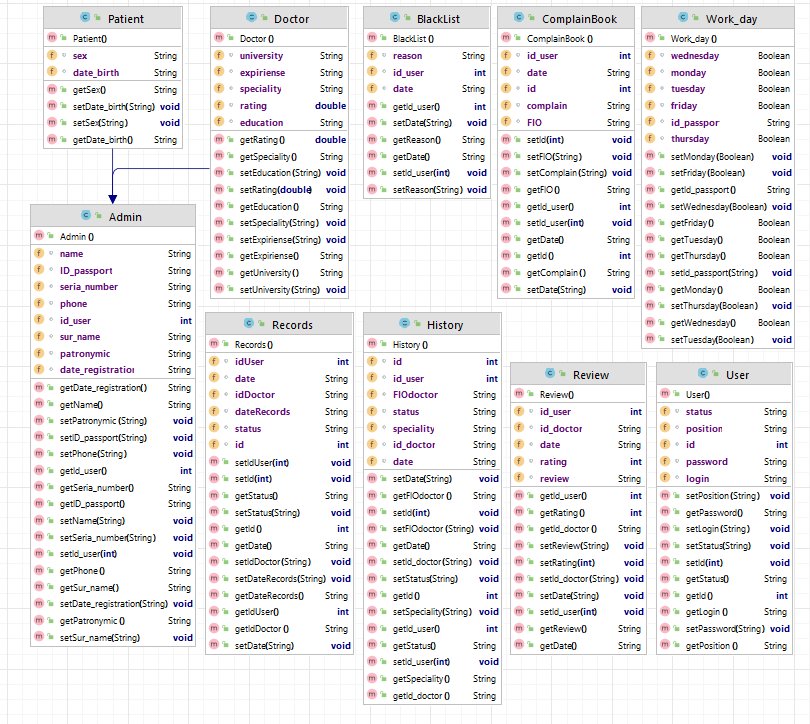


Рисунок 1.16 – Классы пакета models

Переходя к серверной части приложения можно отметить, что там также есть пакет message (рисунок 1.17), единственное отличие которого состоит в наличии еще одного класса. Класс RecordsStatus предназначен для задания статуса приема пациента. У этого статуса есть три состояния: «в ожидании приема», «обслужен», «запись отменена».

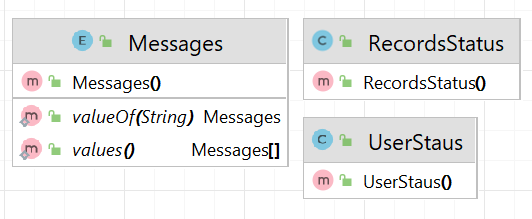


Рисунок 1.17 – Содержимое пакета message

В пакете database (рисунок 1.18) содержаться классы, осуществляющие взаимодействие с базой данных. Класс Config содержит в себе строки, необходимые для подключения к БД. Класс SqlInfo содержит поля, упрощающие и ускоряющие работу с базой данных. Database – класс, в котором происходит все взаимодействие с БД. В нем содержатся методы, которые осуществляют все необходимые для корректной работы программы действия.



Рисунок 1.18 – Классы пакета database

В серверной части приложения также есть пакет controllers (рисунок 1.19), в котором содержаться классы, осуществляющие основной функционал сервера. Контроллеры обращаются к вышеописанному классу database, чтобы отправить или получить данные из БД.

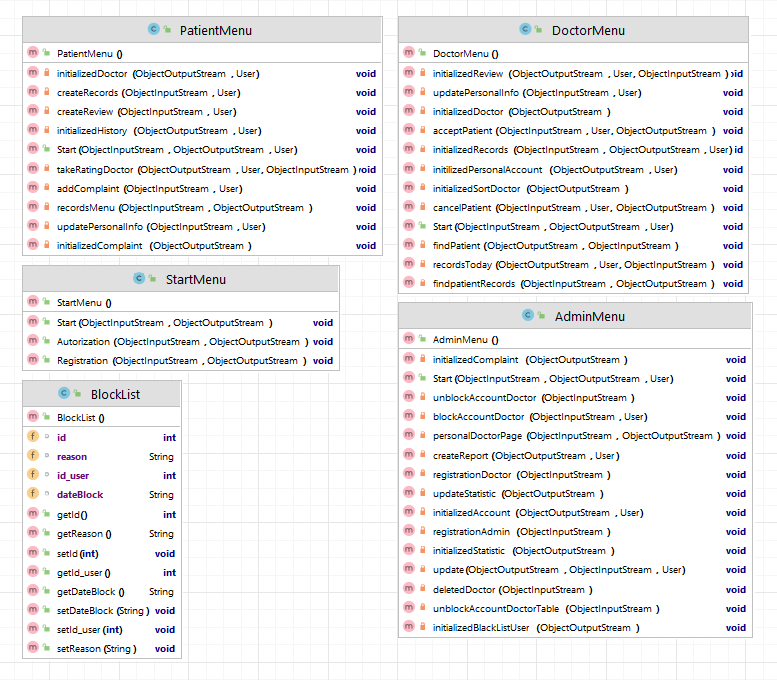


Рисунок 1.19 – Классы пакета controllers

Таким образом была рассмотрена диаграмма классов с описанием их применения, а также рассмотрены паттерны проектирования, которые применяются в данном проекте.

## **1.5 Разработка информационной модели области управления работой частной клиники.**

Информационная модель в данном курсовом проекте была построена с помощью средства MySQL HeidiSQL.

HeidiSQL – многофункциональное свободно распространяемое приложение с открытым исходным кодом для веб-разработчиков, использующих одну из популярных технологий: MySQL сервера, БД Microsoft SQL и PostgeSQL. Она позволяет просматривать и редактировать данные, создавать и изменять таблицы, представления, процедуры, триггеры и события в расписании. Она также предоставляет возможность экспортировать структуры и данные в SQL- файлы, буфер обмена или переносить на другие сервера

Сущностями данной модели являются:

* administration;
* blocklist;
* complaintbook;
* doctor;
* history;
* patient;
* review;
* ticket;
* user;
* work\_day.

Более наглядное представление сущностей базы данных представлено на рисунке 1.20:

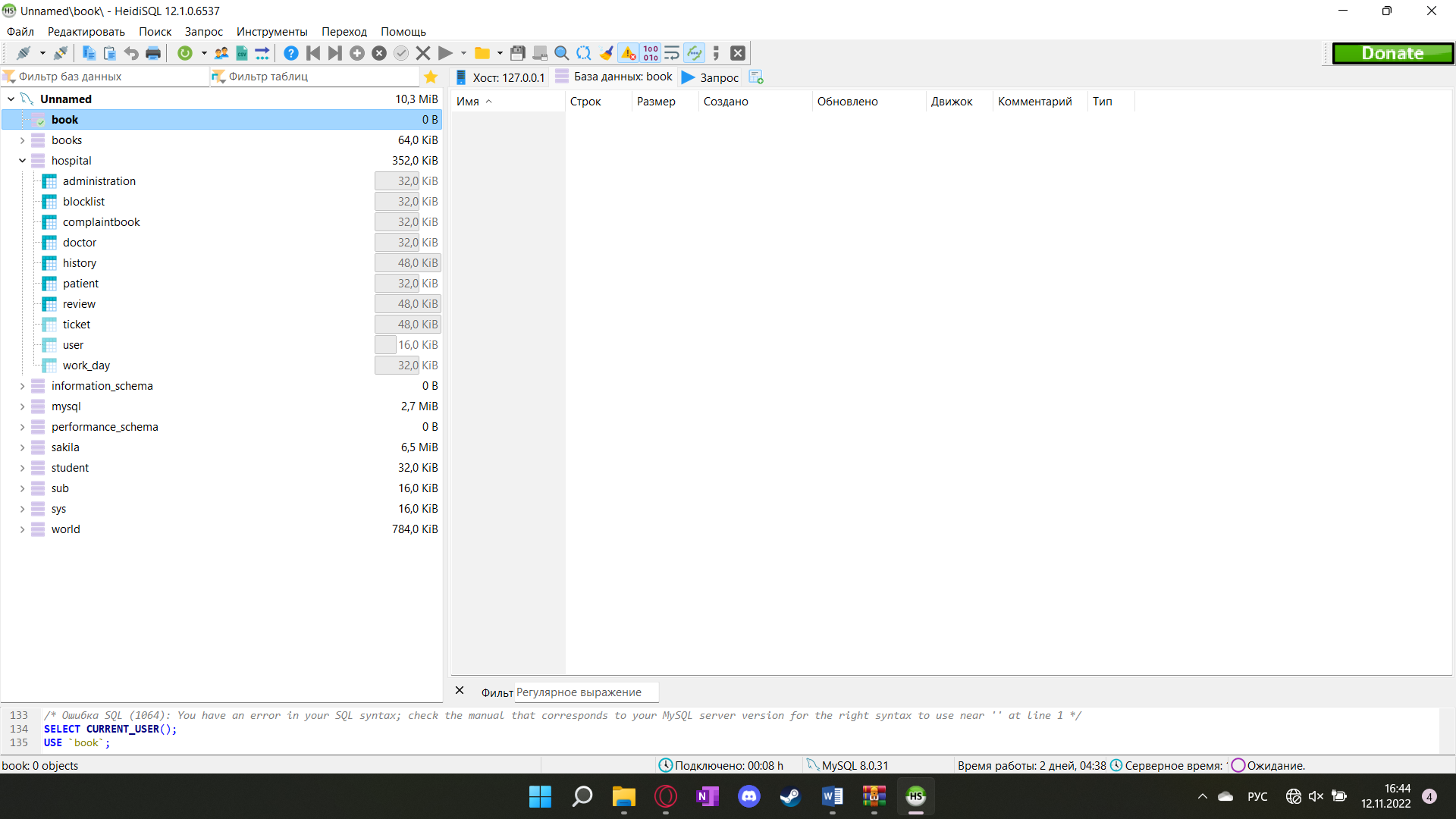


Рисунок 1.20 – Сущности БД «hospital»

Сущность «administration» хранит в себе регистрационные данные: ID\_passport (ключевое поле), seria\_number, date\_registration, id\_user, name, surname, patronymic, phone.

Сущность «blocklist» хранит информацию о заблокированных пользователях.

Сущность «complaintbook» хранит в себе всю информацию о жалобах клиентов.

Сущность «doctor» содержит в себе всю информацию о враче.

Сущность «history» содержит в себе всю информацию о действиях, производимых пользователями.

Сущность «patient» хранит всю информацию о пациенте.

Сущность «review» содержит данные с отзывами о докторах от клиентов.

Сущность «ticket» содержит информацию о талонах, взятых пациентом.

Сущность «user» хранит в себе все данные для входа в систему.

Сущность «work\_day» содержит информацию о рабочих днях врача.

Связи между сущностями и их типы представлены на рисунке 4.2:

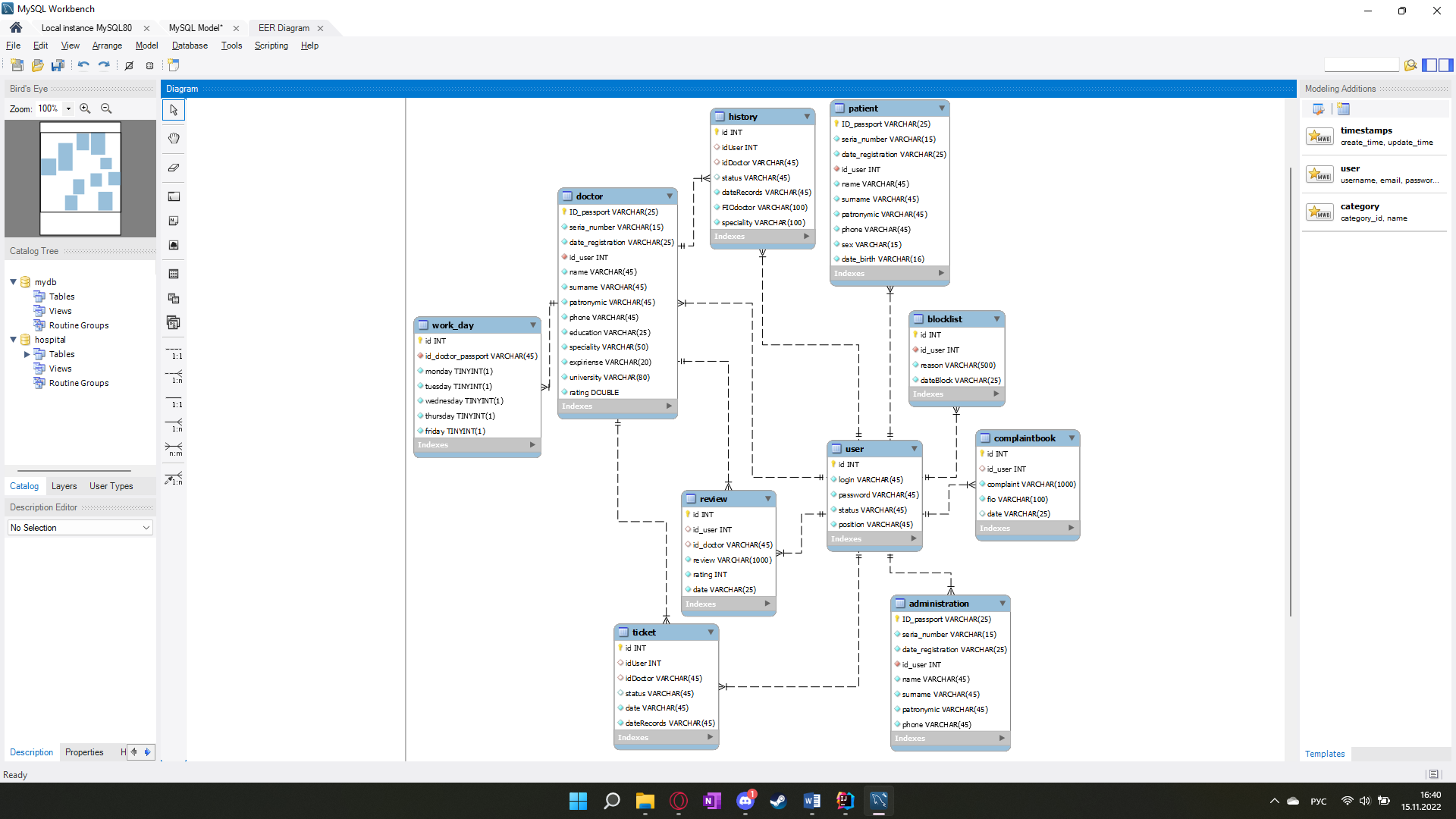


Рисунок 4.2 – Информационная модель базы данных

Таким образом, в данном разделе была рассмотрена информационная модель системы, а также представлено ее описание.

**2. ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ**

**2.1 Постановка задачи**

В данной курсовой работе основная задача заключается в создании автоматизированной системы учета пациентов в поликлинике. В анализе предметной области были рассмотрены основные функции регистратуры и поликлиники в целом, проанализированы программные аналогии будущего приложения.

Для реализации системы необходимо написать приложение в архитектуре клиент-сервер с многопоточным сервером с организацией взаимодействия с базой данных на объектно-ориентированном языке Java.

Для разработки приложения использовалась среда разработки Ultimate IntelliJ IDEA. Это наиболее интеллектуальное и комфортное средство для программирования на основе открытого кода. Продукт поддерживает все инновационные технологии и фреймворки и подходит для разработки коммерческих, мобильных и веб-приложений [7].

JavaFX – платформа на основе Java для создания приложений с насыщенным графическим интерфейсом. Graphical user interface – это графический интерфейс пользователя – разновидность пользовательского интерфейса, в котором все элементы (кнопки, меню, пикторграммы, списки), представленные пользователю на дисплее, выполнены в виде картинок, графики.

MySQL – одна из самых популярных и востребованных систем на сегодняшний день. Это реляционная база данных. Реляционная, то есть, внутри нее есть данные, которые связаны между собой, эту связь можно представить в виде таблиц. В MySQL встроены системы безопасности и разграничения доступа. Но основная причина популярности MySQL – полная поддержка SQL-языка [8].

**2.1 Архитектурные решения**

Apache Maven (обычно называют просто Maven) — это фреймворк по автоматизации и управлению сборкой Java-приложений. Apache Maven, по большей части, написан на Java. Это проект с открытым исходным кодом, который следует философии «Соглашение важнее конфигурации» (convention over configuration).

Основные преимущества Maven:

* Простая настройка проекта в соответствии с лучшими практиками
* Управление зависимостями, включая автоматическое обновление
* Возможность работать с несколькими проектами одновременно
* Большое хранилище библиотек и метаданных для использования «из коробки»
* Расширяемость с возможностью написания своих плагинов
* Создание сайта и PDF с документацией
* Проверка корректности структуры проекта
* Компиляция исходного кода, отображение ошибок/предупреждений
* Тестирование проекта на основе уже написанных тестов
* Упаковка скомпилированного кода в артефакты (например, .jar, .war, .ear, .zip-архивы и многое другое)
* Упаковка исходного кода в загружаемые архивы/артефакты
* Установка упакованных артефактов на сервер для последующего развертывания (деплоя) или в репозиторий для распространения
* Создание отчетов по тестированию
* Сообщение об удачной/неудачной сборке проекта

Но Maven имеет также и ряд недостатков:

* сложность освоения;
* неочевидность в некоторых моментах;
* огромное количество плагинов;
* трудно разобраться если возникла ошибка;
* необходим доступ в Интернет или наличие собственного репозитория артефактов.

Помимо того, что плюсов гораздо больше чем минусов, они являются также более значимыми чем минусы.

В курсовом проекте используется паттерн проектирования Singleton. Этот шаблон проектирования предназначен для ограничения ненужного создания экземпляра класса и обеспечения того, чтобы в любой момент времени для каждого экземпляра JVM существовал только один объект класса. Таким образом, с этим шаблоном любой класс, определенный как Singleton, имеет только один экземпляр с глобальной точкой доступа к нему. В отличие от обычных классов, одноэлементный класс не уничтожается до конца жизненного цикла приложения. Все реализации одиночки сводятся к тому, чтобы скрыть конструктор по умолчанию и создать публичный статический метод, который и будет контролировать жизненный цикл объекта-одиночки. Также используется паттерн проектирования Builder.

В контексте работы приложения данный паттерн применим к взаимодействию клиента с контроллерами fxml-сцен. При регистрации или входе клиента в систему вызывается в первый раз функция getInstance(), в результате которой создается единственный объект класса MyClient. В ходе дальнейшей работы приложения сцены постоянно переключаются, тем самым изменяя текущий контроллер. Каждому контроллеру необходим доступ к объекту MyClient для отправления данных на сервер, а паттерн «Одиночка» гарантирует, что в текущей сессии экземпляр клиента будет единственным, что и необходимо для корректной работы приложения [9].

**2.3 Описание алгоритмов, реализующих бизнес-логику разрабатываемой автоматизированной системы**

Блок-схема — распространённый тип [схем](https://ru.wikipedia.org/wiki/%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_(%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0)) (графических [моделей](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)), описывающих [алгоритмы](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединённых между собой линиями, указывающими направление последовательности.

Блок-схема алгоритма записи на прием представлена на рисунке 2.1.

Для начала мы получаем соединение с БД. Далее если соединение получено, то происходит формирование запроса, потом установка параметра id в запрос. После этого запрос выполняется и если он выполнен, то происходит получение и возврат результата. Если соединение не было становлено, то выбрасывается исключение.

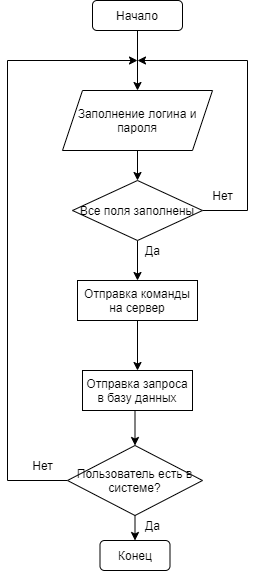


Рисунок 2.1 — Вход в систему

Блок-схема алгоритма записи на прием представлена на рисунке 2.2.

Удаление врача осуществляется с аккаунта админитстратора. После входа в систему в данной роли, выбирается врач на удаление. После выбора врач удаляется из базы данных.

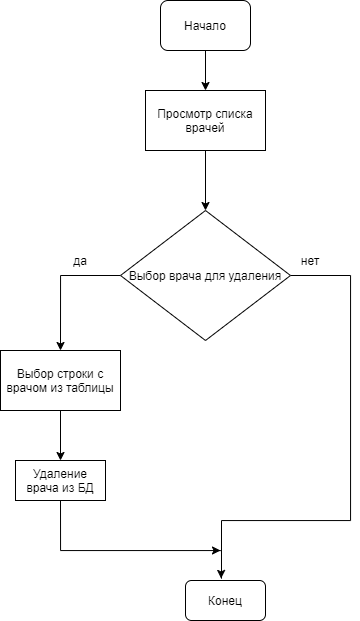


Рисунок 2.2 — Удаление врача

**2.4 Проектирование пользовательского интерфейса**

Пользовательский интерфейс (user interface или сокращенно UI) – это интерфейс, с помощью которого человек может управлять программным обеспечением или аппаратным оснащением. UI должны быть удобными в использовании, чтобы взаимодействие с ними происходило на максимально интуитивном уровне. Интерфейсы программного обеспечения также называют графическими пользовательскими интерфейсами (graphical user interface или GUI).

Graphical user interface — графический интерфейс пользователя — это разновидность пользовательского интерфейса, в котором все элементы (кнопки, меню, пиктограммы, списки) представленные пользователю на дисплее, выполнены в виде картинок, графики. В отличие от интерфейса командной строки, в GUI у пользователя есть произвольный доступ к видимым объектам с помощью устройств ввода. Зачастую элементы интерфейса реализованы в виде метафор и отображают их свойства и назначение для облегчения понимания пользователя.

JavaFX может являться инструментом GUI для Java.

Особенности JavaFX:

* JavaFX изначально поставляется с большим набором частей графического интерфейса, таких как кнопки, текстовые поля, таблицы, деревья, меню, диаграммы, что в свою очередь позволяет сэкономит время.
* JavaFX часто использует стили CSS, и мы сможем использовать специальный формат FXML для создания GUI, а не делать это в коде Java. Это облегчает быстрое размещение графического интерфейса пользователя или изменение внешнего вида или композиции без необходимости долго прописывать все в коде Java.
* JavaFX имеет готовые к использованию части диаграммы, поэтому нам не нужно писать их с нуля в любое время, когда вам нужна базовая диаграмма.

**2.5 Обоснование выбора компонентов и технологий для реализации автоматизированной системы**

Данное программное приложение реализовано на языке Java в среде разработки IntelliJ IDEA.

IntelliJ – одна из самых мощных и популярных интегрированных сред разработки (IDE) для Java. Он разработан и поддерживается JetBrains и доступен как окончательная версия для сообщества. Чтобы помочь разработчикам организовать рабочий процесс, IntelliJ IDEA предлагает им удивительный набор инструментов, который включает в себя декомпилятор, поддержку Docker, средство просмотра байт-кода, FTP и многие другие инструменты. Контроль версий – IntelliJ поддерживает большинство популярных систем контроля версий, таких как Git, Subversion, Mercurial, CVS, Perforce и TFS. Инструменты сборки – IntelliJ поддерживает Java и другие инструменты сборки, такие как Maven, Gradle, Ant, Gant, SBT, NPM, Webpack, Grunt и Gulp.Тестовый прогон и покрытие кода – IntelliJ IDEA позволяет с легкостью выполнять модульное тестирование. Среда IDE включает в себя тестовые прогоны и инструменты покрытия для основных сред тестирования, включая JUnit, TestNG, Spock, Cucumber, ScalaTest, spec2 и Karma. Декомпилятор – IntelliJ поставляется со встроенным декомпилятором для классов Java. Терминал – IntelliJ предоставляет встроенный терминал. В зависимости от вашей платформы вы можете работать с командной строкой, например PowerShell или Bash. Инструменты базы данных – IntelliJ предоставляет инструменты базы данных, которые позволяют подключаться к действующим базам данных; выполнять запросы; просматривать и обновлять данные; и даже управлять своими схемами в визуальном интерфейсе из самой IDE. Сервер приложений – IntelliJ поддерживает основные серверы приложений: Tomcat, JBoss, WebSphere, WebLogic, Glassfish и многие другие. Вы можете развернуть свои артефакты на серверах приложений и отладить развернутые приложения в самой IDE. Поддержка Docker. Через отдельный плагин IntelliJ предоставляет специальное окно инструментов, которое позволяет подключаться к локально работающим компьютерам Docker. Контроль версий – IntelliJ поддерживает большинство популярных систем контроля версий, таких как Git, Subversion, Mercurial, CVS, Perforce и TFS.

Уникальные возможности IntelliJ IDEA избавляют программиста от груза рутинной работы, помогают своевременно устранить ошибки и повысить качество кода, поднимая продуктивность разработчика на новую высоту.

Java — язык программирования общего назначения. Относится к объектно-ориентированным языкам программирования, к языкам с сильной типизацией.

Создатели реализовали принцип WORA: write once, run anywhere или «пиши один раз, запускай везде». Это значит, что написанное на Java приложение можно запустить на любой платформе, если на ней установлена среда исполнения Java (JRE, Java Runtime Environment).

Эта задача решается благодаря компиляции написанного на Java кода в байт-код. Этот формат исполняет JVM или виртуальная машина Java. JVM — часть среды исполнения Java (JRE). Виртуальная машина не зависит от платформы.

Ключевой особенностью языка Java является то, что его код сначала транслируется в специальный байт-код, независимый от платформы. А затем этот байт-код выполняется виртуальной машиной JVM (Java Virtual Machine). В этом плане Java отличается от стандартных интерпретируемых языков как PHP или Perl, код которых сразу же выполняется интерпретатором. В то же время Java не является и чисто компилируемым языком, как С или С++.

Подобная архитектура обеспечивает кроссплатформенность и аппаратную переносимость программ на Java, благодаря чему подобные программы без перекомпиляции могут выполняться на различных платформах - Windows, Linux, Mac OS и т.д. Для каждой из платформ может быть своя реализация виртуальной машины JVM, но каждая из них может выполнять один и тот же код.

Java является языком с Си-подобным синтаксисом и близок в этом отношении к C/C++ и C#. Поэтому, если вы знакомы с одним из этих языков, то овладеть Java будет легче.

Еще одной ключевой особенностью Java является то, что она поддерживает автоматическую сборку мусора. А это значит, что вам не надо освобождать вручную память от ранее использовавшихся объектов, как в С++, так как сборщик мусора это сделает автоматически за вас.

Java является объектно-ориентированным языком. Он поддерживает полиморфизм, наследование, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений.

Maven – это инструмент управления проектами и понимания, который предоставляет разработчикам полную среду жизненного цикла сборки. Команда разработчиков может автоматизировать инфраструктуру сборки проекта практически мгновенно, поскольку Maven использует стандартную структуру каталогов и жизненный цикл сборки по умолчанию.

Maven предоставляет разработчикам способы управления следующим:

* Строит
* Документация
* Составление отчетов
* зависимости
* СДМ
* релизы
* распределение
* Список рассылки

Подводя итог, Maven упрощает и стандартизирует процесс сборки проекта. Он легко обрабатывает компиляцию, распространение, документацию, совместную работу в команде и другие задачи. Maven увеличивает возможность многократного использования и выполняет большинство задач, связанных со сборкой.

MySQL – одна из самых популярных и востребованных систем на сегодняшний день. Это реляционная база данных. Реляционная, то есть, внутри нее есть данные, которые связаны между собой, эту связь можно представить в виде таблиц. В MySQL встроены системы безопасности и разграничения доступа. Но основная причина популярности MySQL – полная поддержка SQL-языка [8].

# **3. ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА**

Для создание модульных тестов использовалась инфраструктура модульного тестирования Junit 4.

Тестированию подверглась работа с пользователями, а именно методы сущности User.

Для тестирования работы данных методов было проведено функциональное тестирование и разработан тестовый класс UserTest. Результат тестирования приведён на рисунке 8.1.

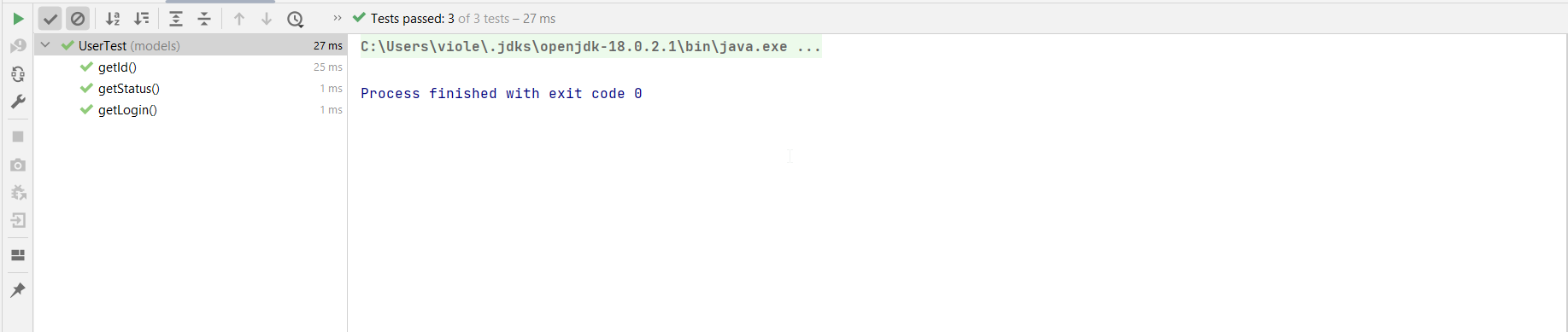


Рисунок 8.1 – Результат тестирования

Программа также подвергалась ручному тестированию, в ходе которого было выявлено, что присутствуют все проверки на полях, которые подвергались тестированию. Проверки производятся для полей с различными форматами, а именно: формат даты и времени, идентификационного номера паспорта, серии паспорта, номера телефона и др. На рисунке 8.2 приведен пример полей-подсказок, которые появляются после некорректного ввода:



Рисунок 8.2 – Пример окна после некорректного ввода информации

Таким образом, в данной главе было рассмотрены создание модульных тестов с помощью Junit4 и ручное тестирование. Протестирован класс User для добавления и обновления информации о пользователях, а также протестирован весь функционал программы на наличие проверок на правильность ввода.

# **4. РУКОВОДСТВО ПО РАЗВЕРТЫВАНИЮ И ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА**

При запуске приложения пользователя приветствует стартовое окно, изображенное на рисунке 4.1, где пользователь выбирает, зарегистрироваться или войти.

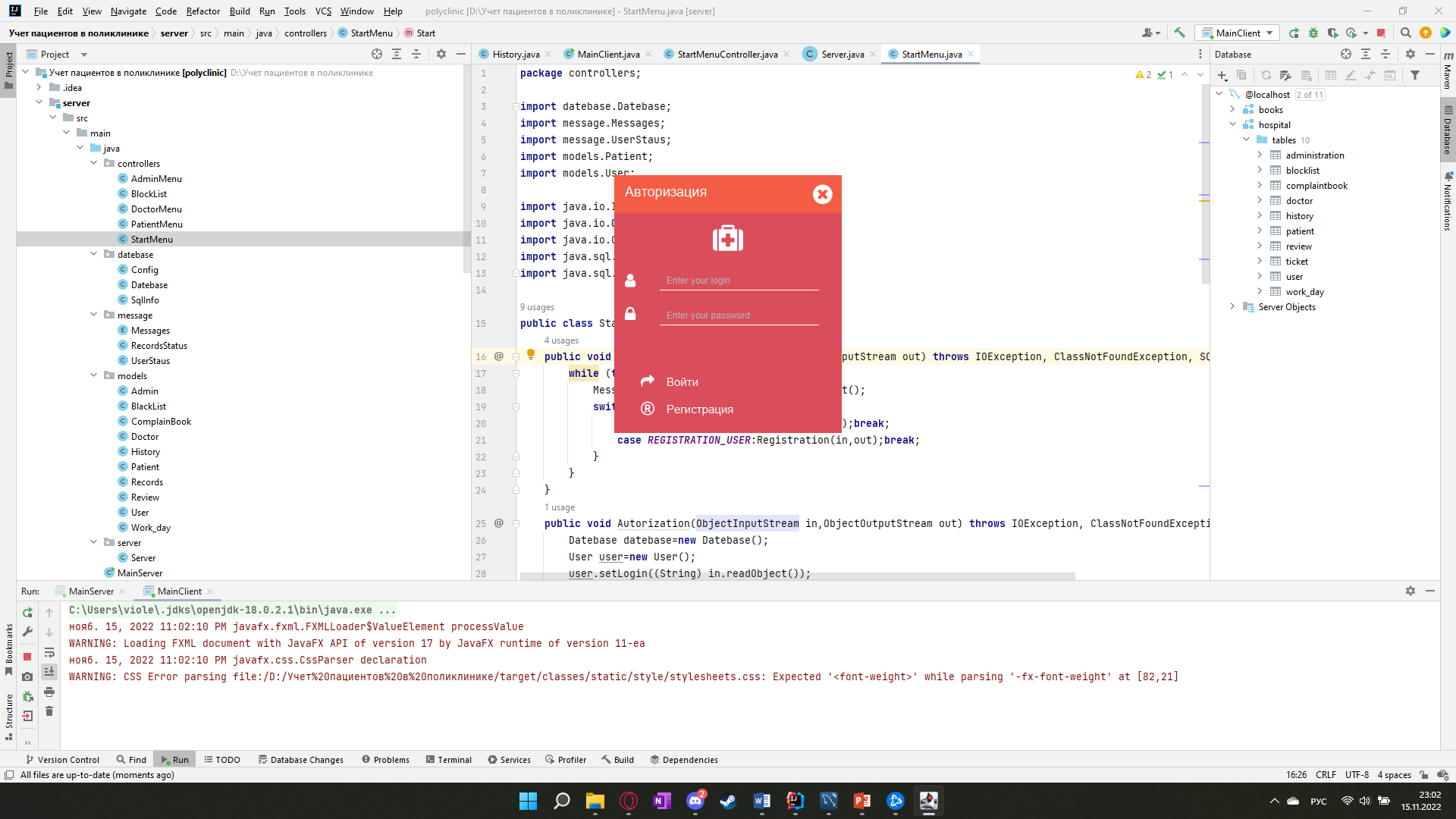


Рисунок 4.1 – Окно авторизации

Для начала рассмотрим функционал администратора системы. После ввода логина и пароля появляется следующее окно, представленное на рисунке 4.2. Администратору предоставляется меню, благодаря которому он может зарегистрировать нового админа, вносить пользователей в черный список, просматривать жалобную книгу, регистрировать и просматривать список врачей и список пациентов, составлять отчет, просматривать статистику, настраивать свой аккаунт и совершать выход из профиля.

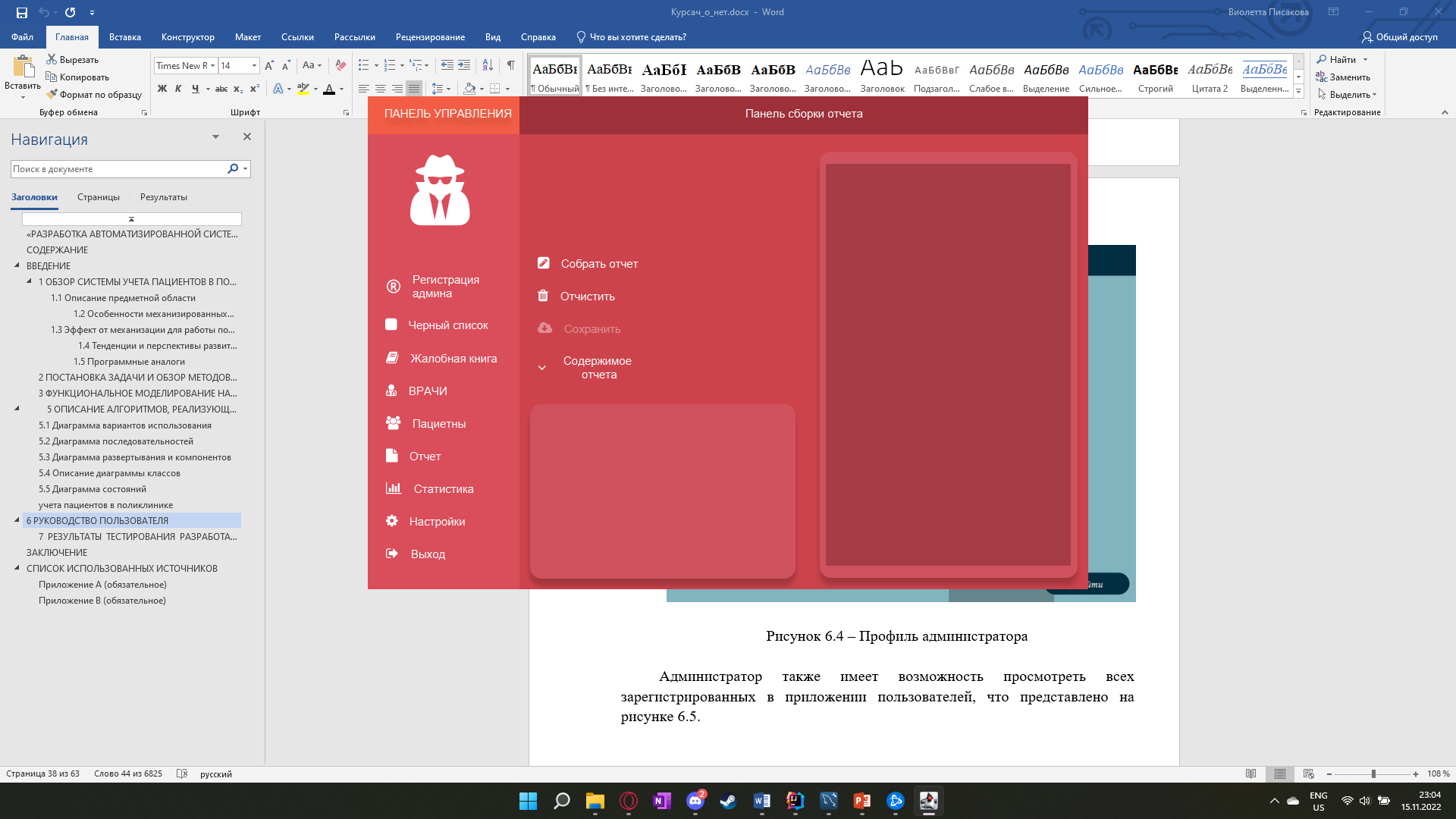


Рисунок 4.2 – Окно админа после входа

Если администратор хочет добавить нового администратора, то ему предоставляется следующая информация, которую можно вносить. Пример представлен на рисунке 4.3.

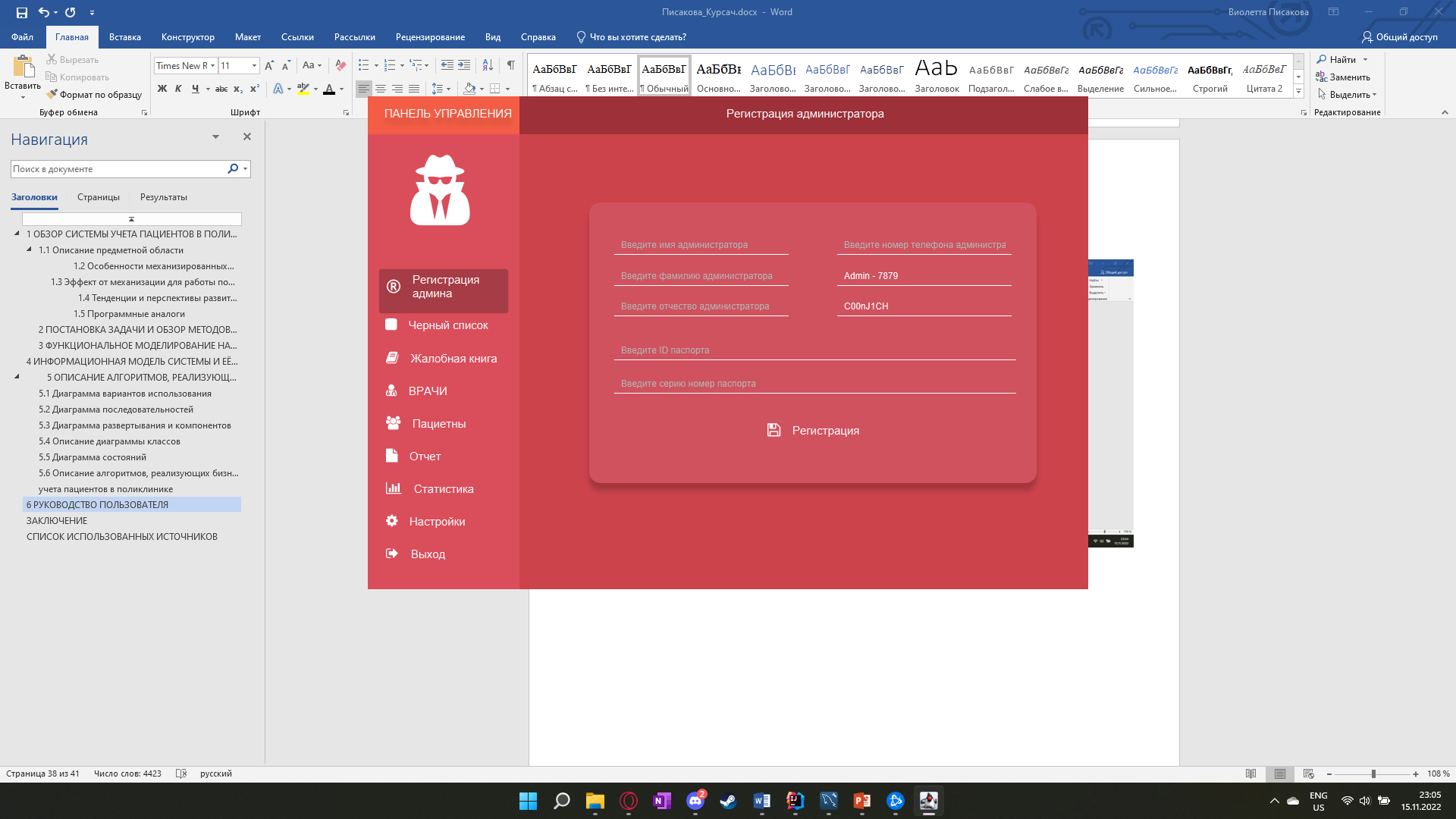


Рисунок 4.3 – Окно регистрации нового администратора

У администратора есть возможность вносить врачей в черный список. На рисунке 4.4 представлен пример окна «Черный список».

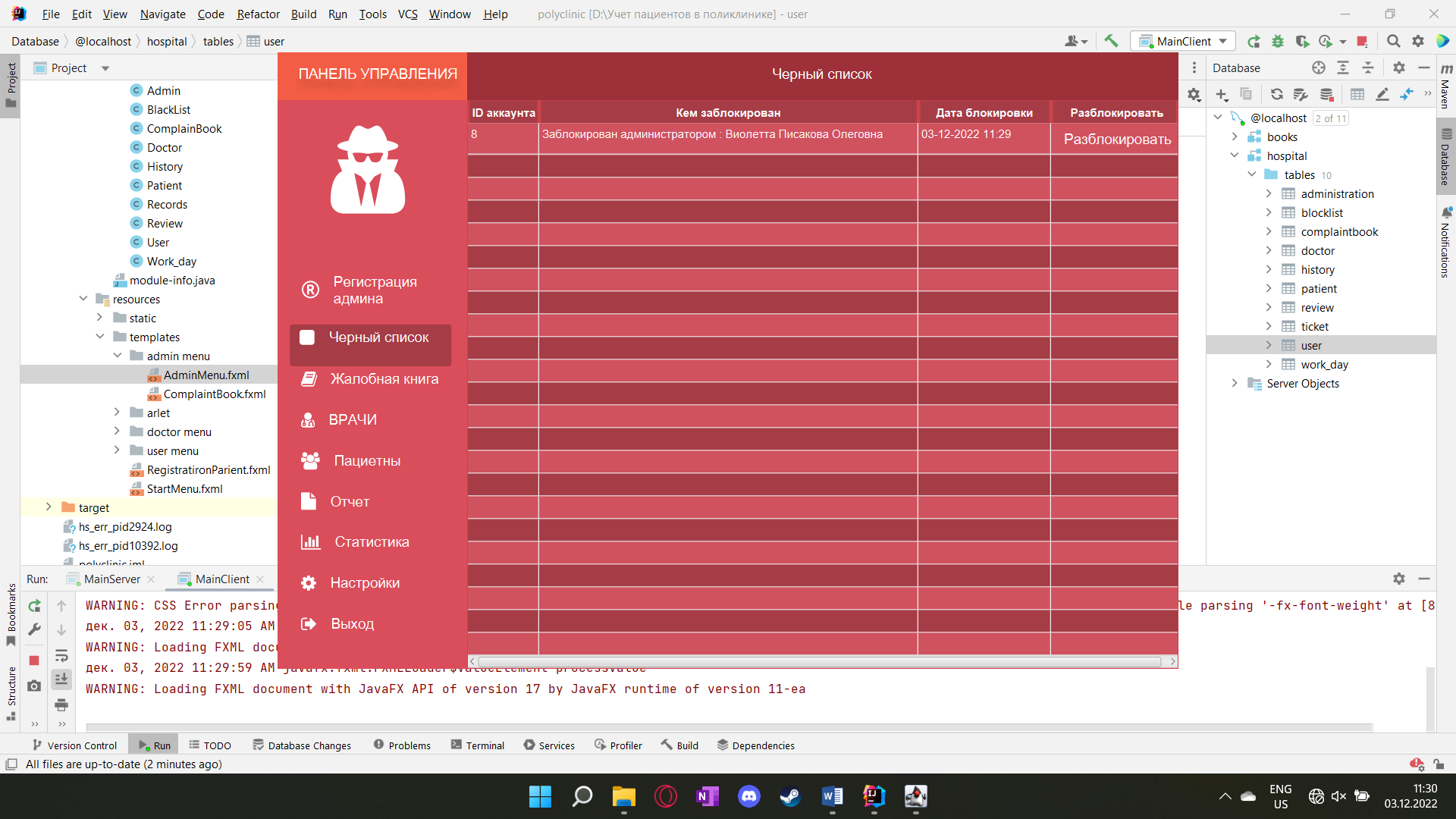


Рисунок 4.4 – Окно черного списка

Администратор также может просматривать жалобную книгу, где собраны все жалобы пациентов. Пример окна, содержащего жалобную книгу представлен на рисунке 4.5.

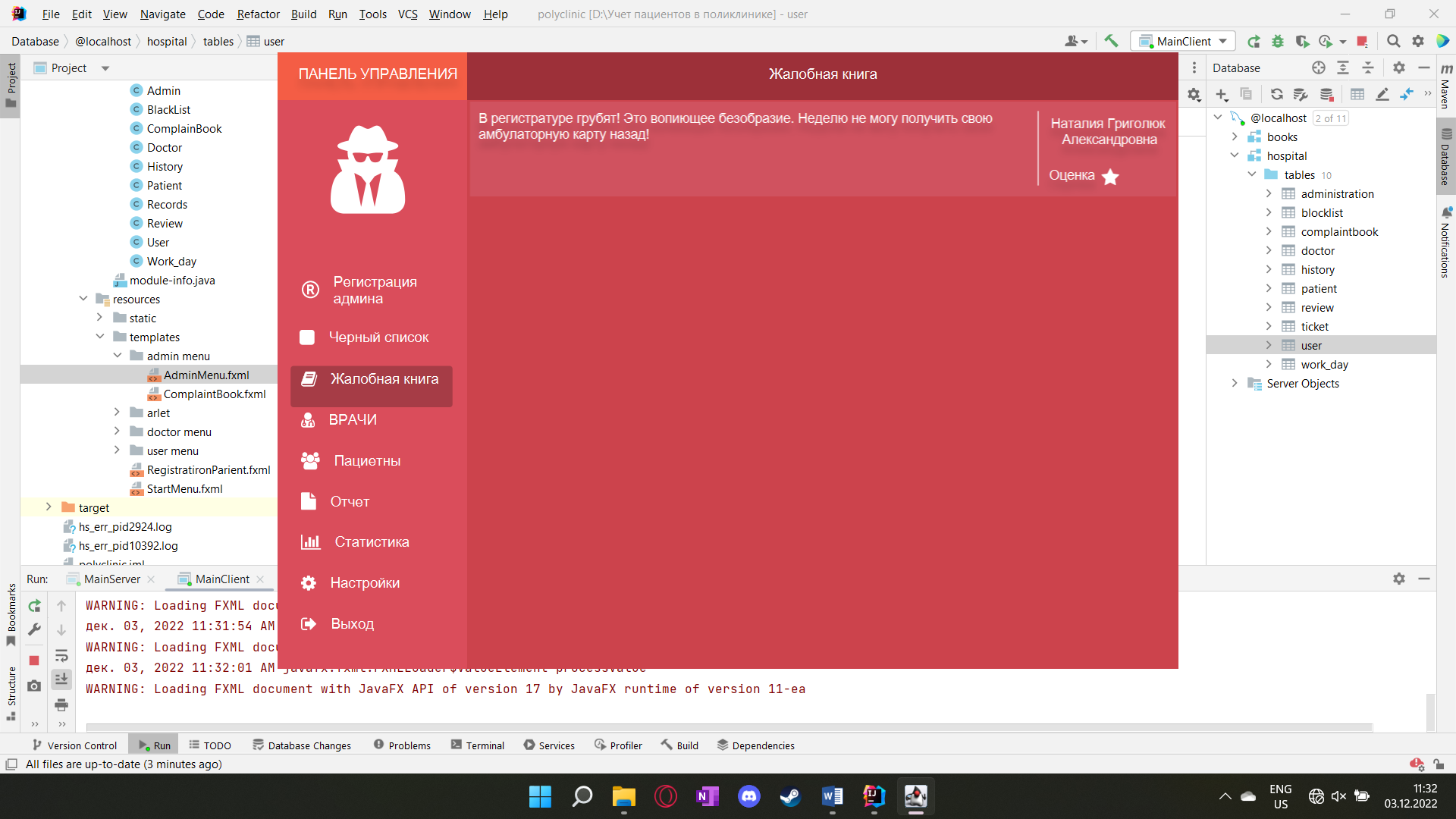


Рисунок 4.5 – Окно «Жалобная книга»

На рисунке 4,6 и 4.7 представлено окно «Врачи», в котором администратор может просматривать список и регистрировать новых врачей.

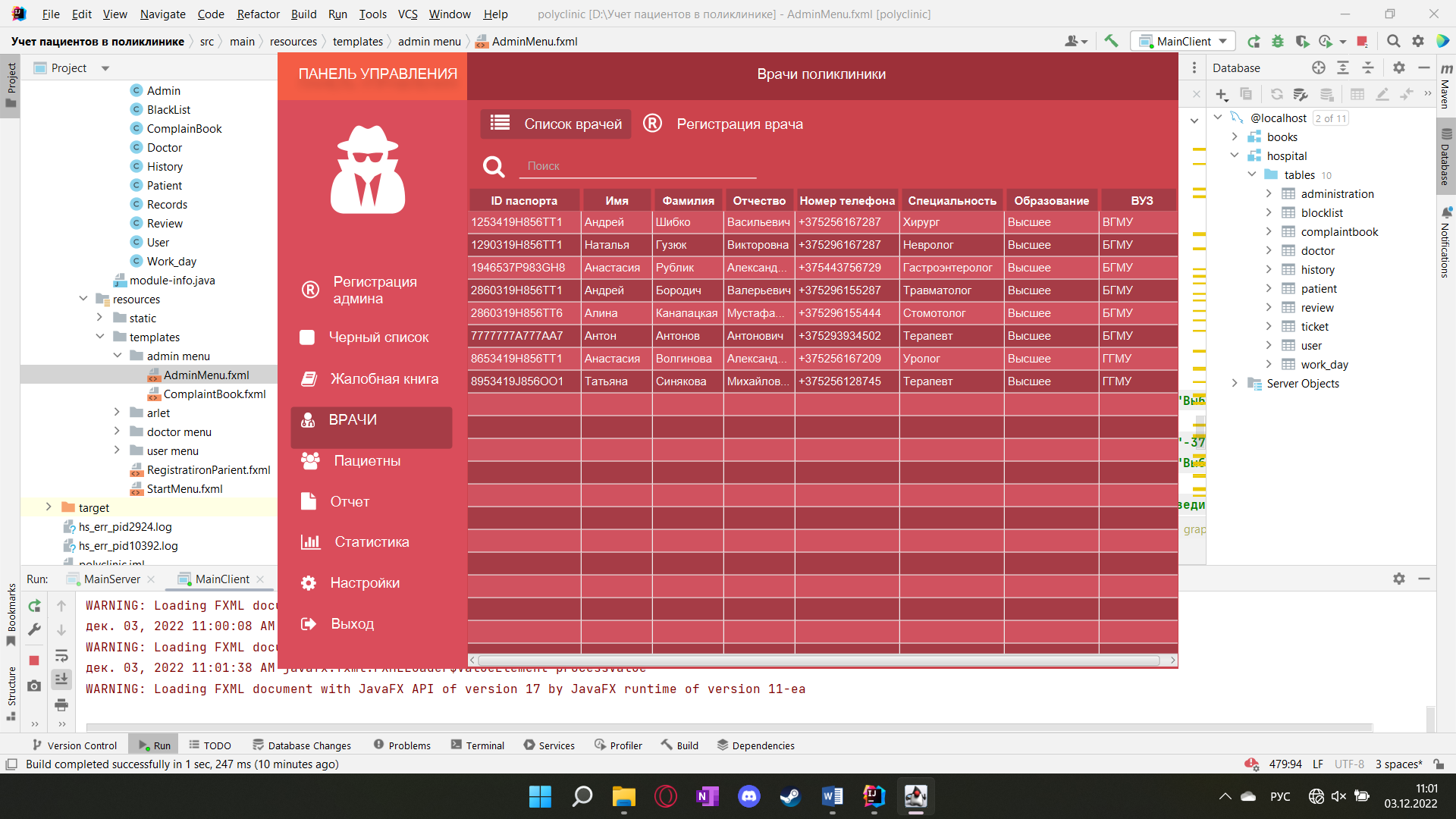


Рисунок 4.6 – Окно со списком врачей

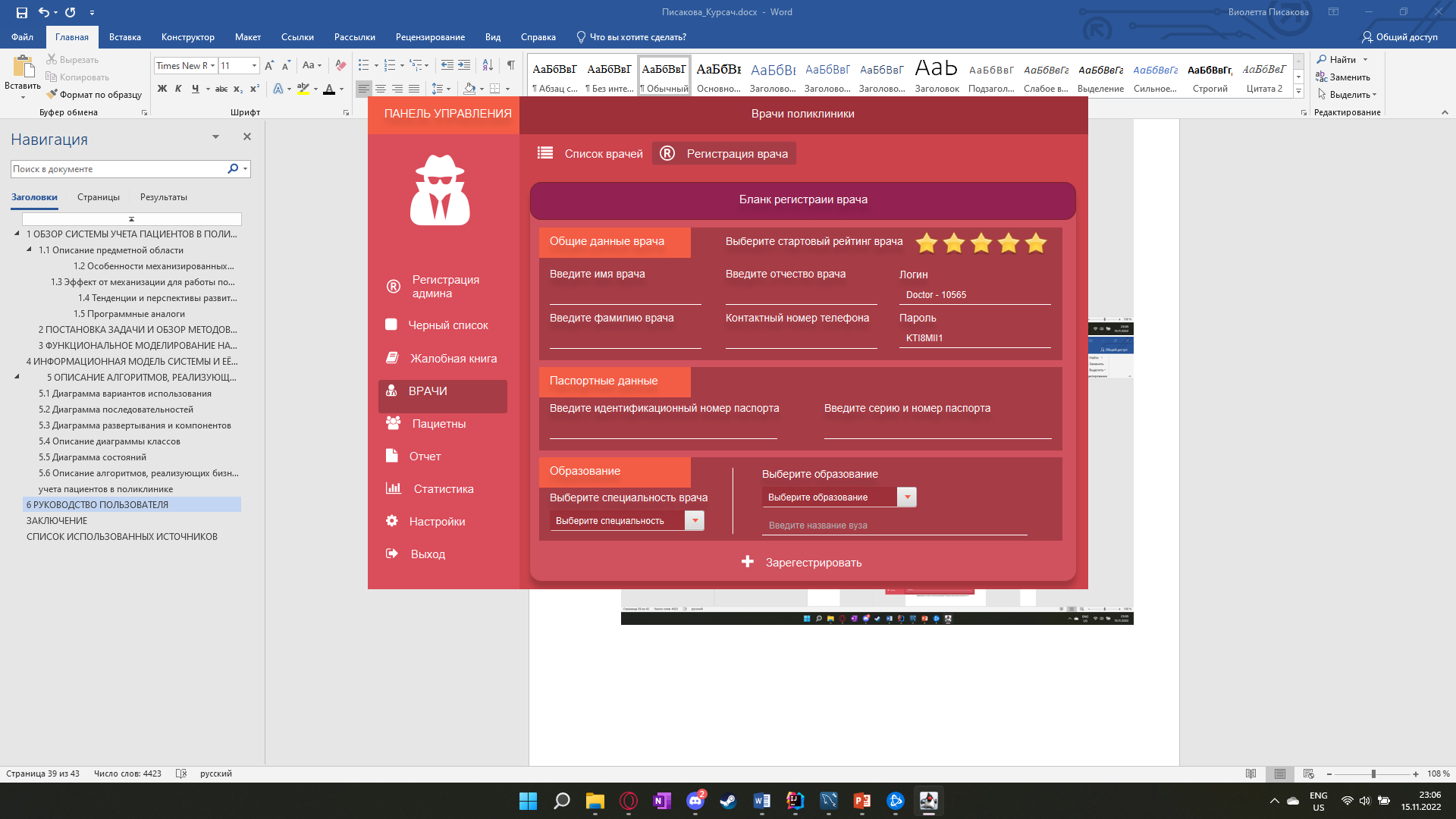


Рисунок 4.7 – Окно регистрации врача

Администратор может также просматривать список пациентов и видеть все данные о них. На рисунке 4.8 представлен пример окна «Пациенты»:

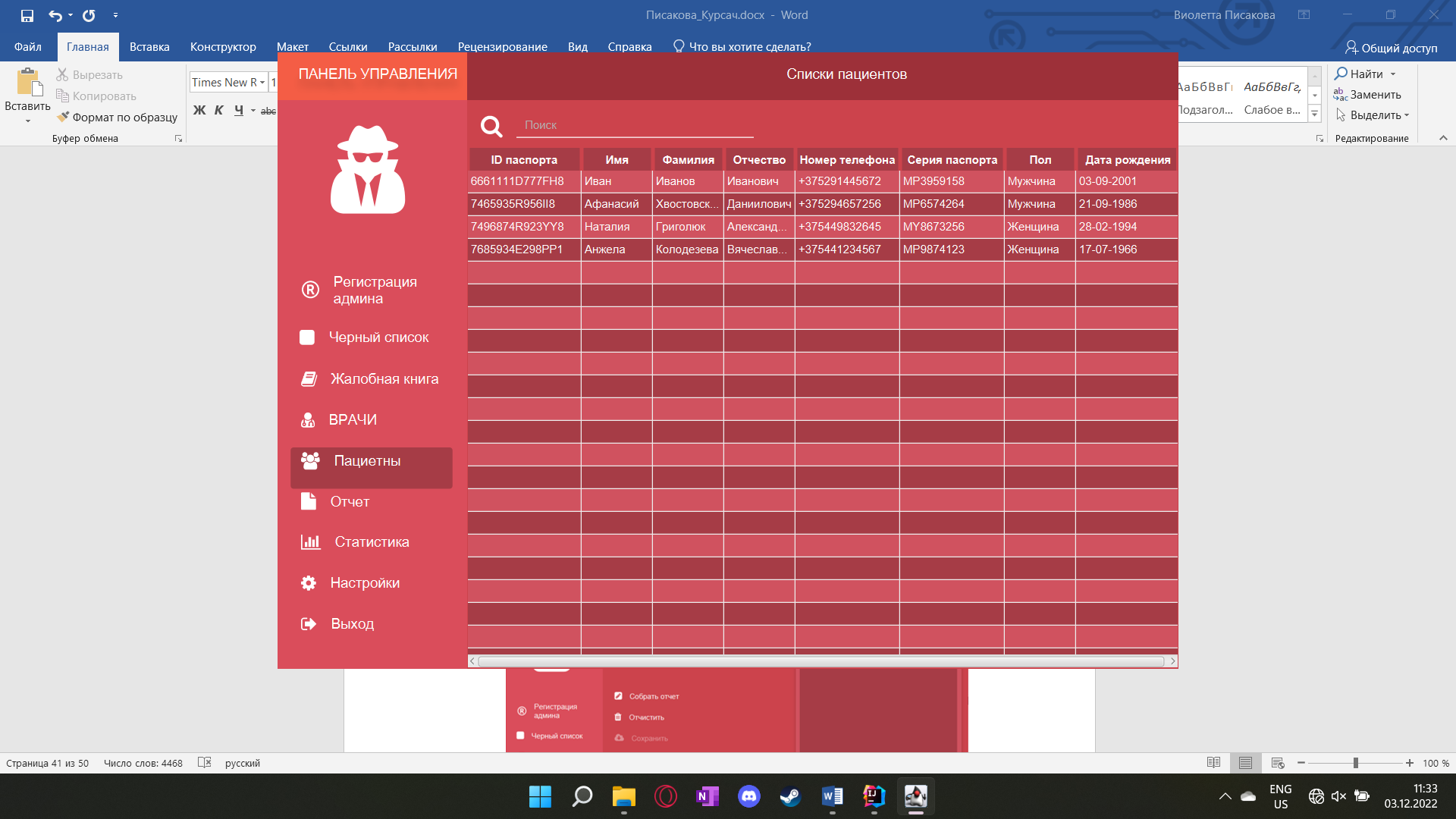


Рисунок 4.8 – Окно со списком пациентов

Во вкладке «Отчет» (рисунок 4.9) можно собрать всю статистику о работе поликлиники, а именно: количество врачей и их количество по специализациям, количество людей в черной списке, количество жалоб, средняя оценка врачей, самый загруженный день, количество талонов в день, а также дата создания отчета и ФИО админа, который его собрал. В окне присутствуют кнопки очищения окна отчета, а также кнопка для сохранения отчета в файл.

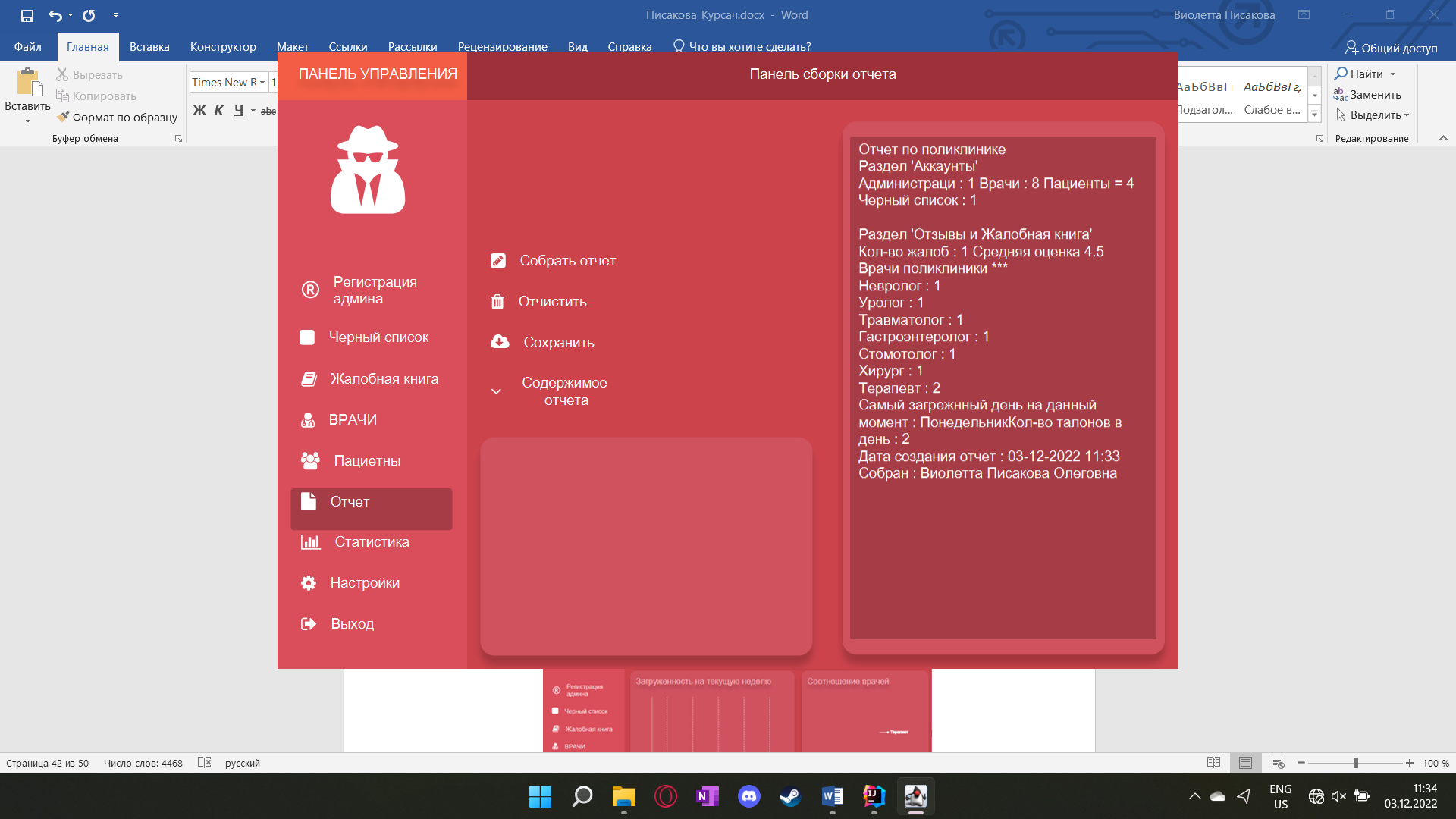


Рисунок 4.9 – Окно «Отчет»

Окно статистики (рисунок 4.10) отображает графики и диаграммы, которые визуализируют информацию из отчета.

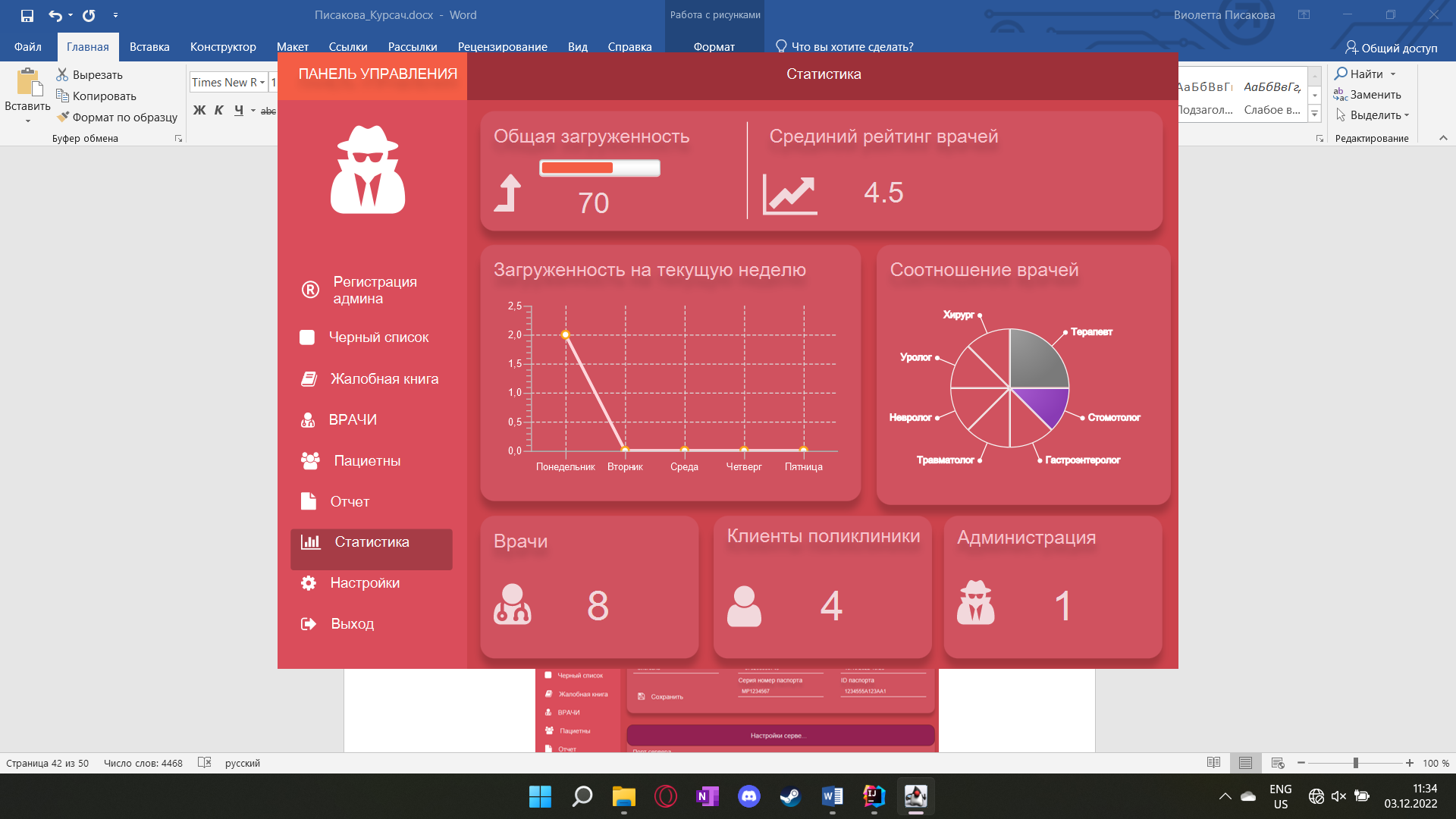


Рисунок 4.10 – Окно «Статистика»

В окне «Настройки» администратор может изменить свои личные данные, а также ввести порт сервера, что автоматически его изменит. Пример окна настроек изображен на рисунке 4.11:

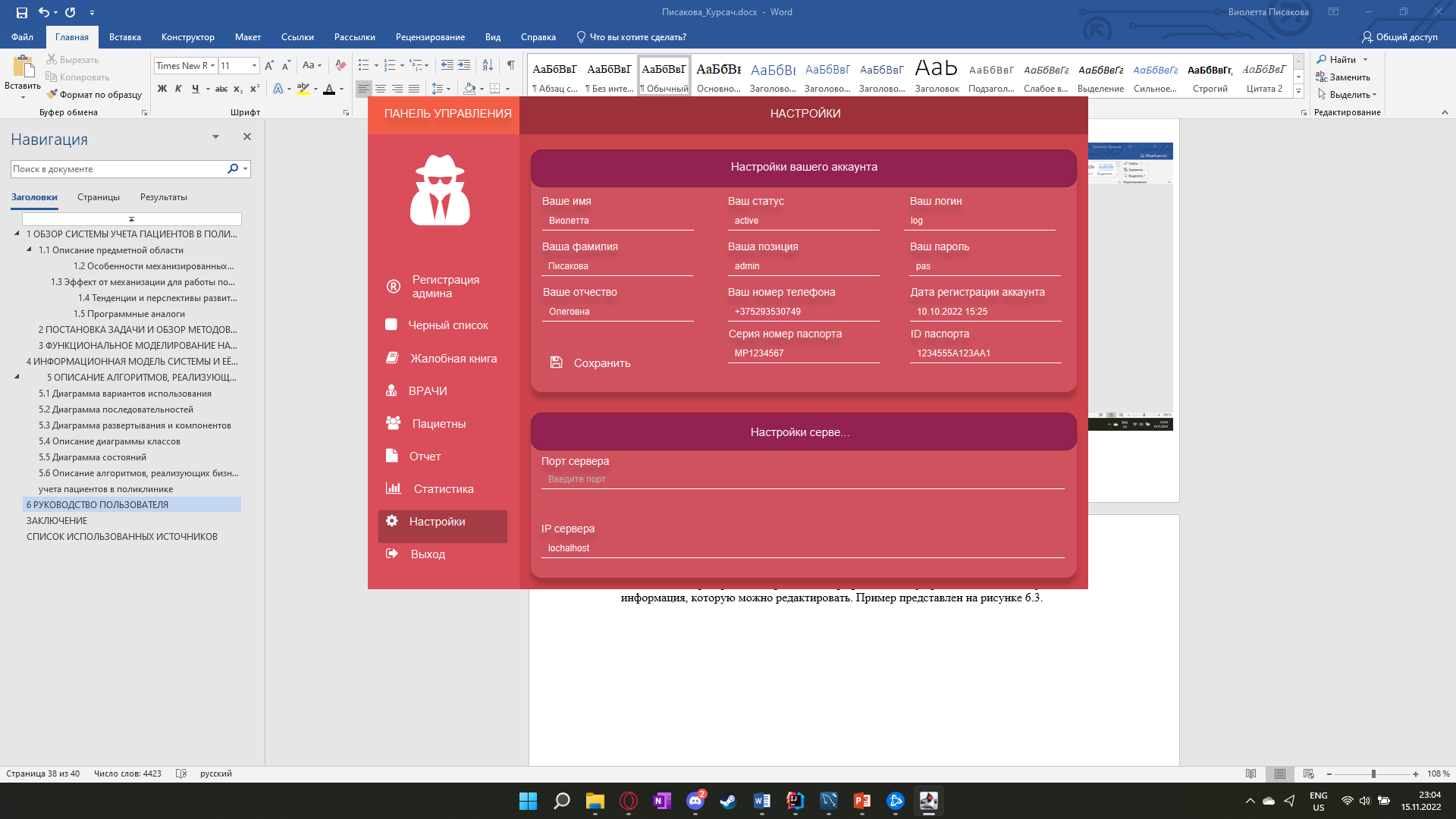


Рисунок 4.11 – Окно настроек

Из начального экрана входа можно попасть на экран регистрации (рисунок 4.12). Регистрация доступна только для пациентов, так как врачей регистрирует администратор. Во время регистрации пациенту необходимо ввести свои ФИО, номер телефона, логин, пароль, дату рождения, а также ID, серию и номер паспорта.

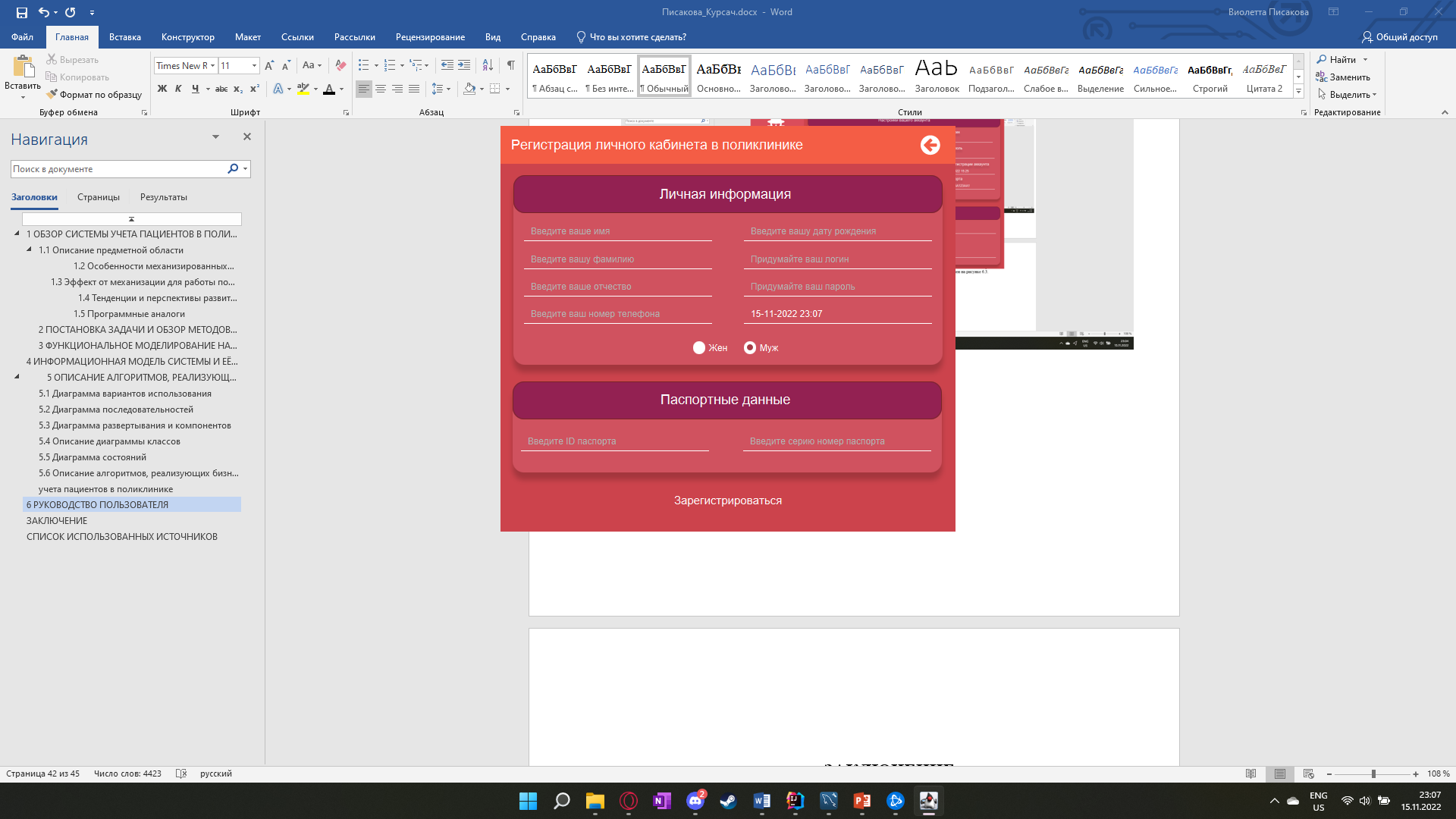


Рисунок 4.12 – Окно регистрации

После регистрации пациету необходимо будет войти в свой профиль.

Функционал окна пациента отображен на рисунке 4.13. Пациенту предоставляется возможность написать жалобу, просмотреть список врачей и взять талон на определенную дату к определенному врачу, оставить отзыв, просмотреть историю посещений и настроить свой профиль.

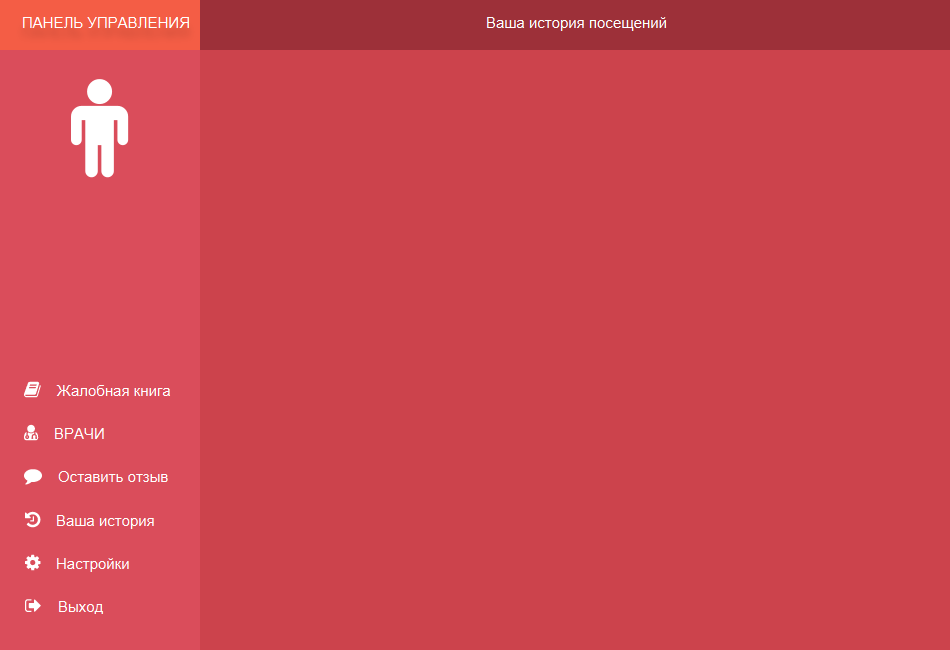


Рисунок 4.13 – Окно пациента

В окне жалоб пациент может оставить свою жалобу, а также просмотреть жалобы других пользователей. Пример окна жалоб представлен на рисунке 4.14:

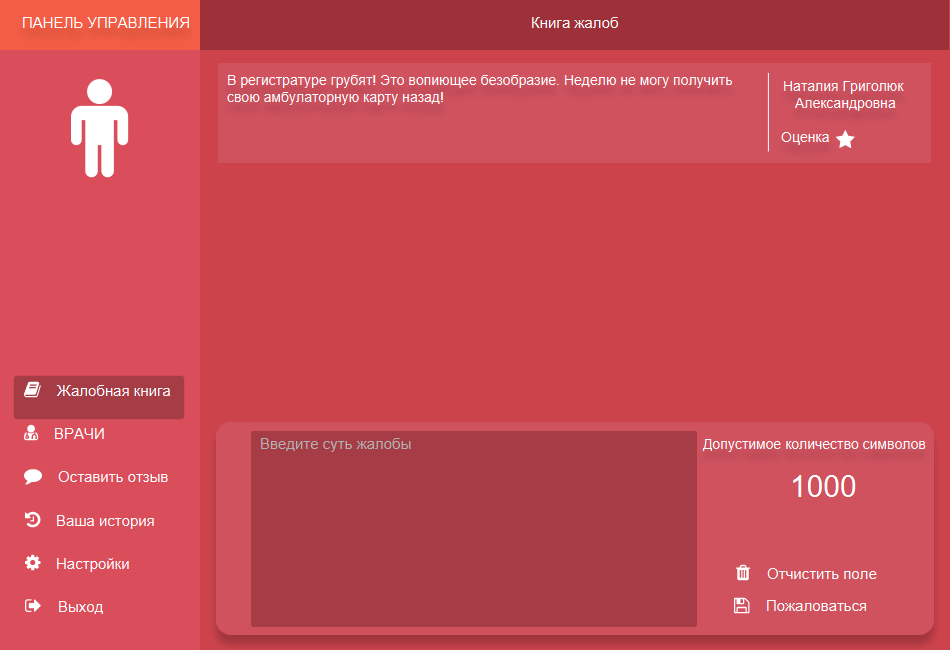


Рисунок 4.14 – Пример окна жалоб

Во вкладке «Врачи» (рисунок 4.15) пациент может просмотреть весь список врачей, а также забронировать талон к интересующему врачу на определенную дату, выбрав эту дату в календаре. Во вкладке календаря отображаются только рабочие дни доктора.

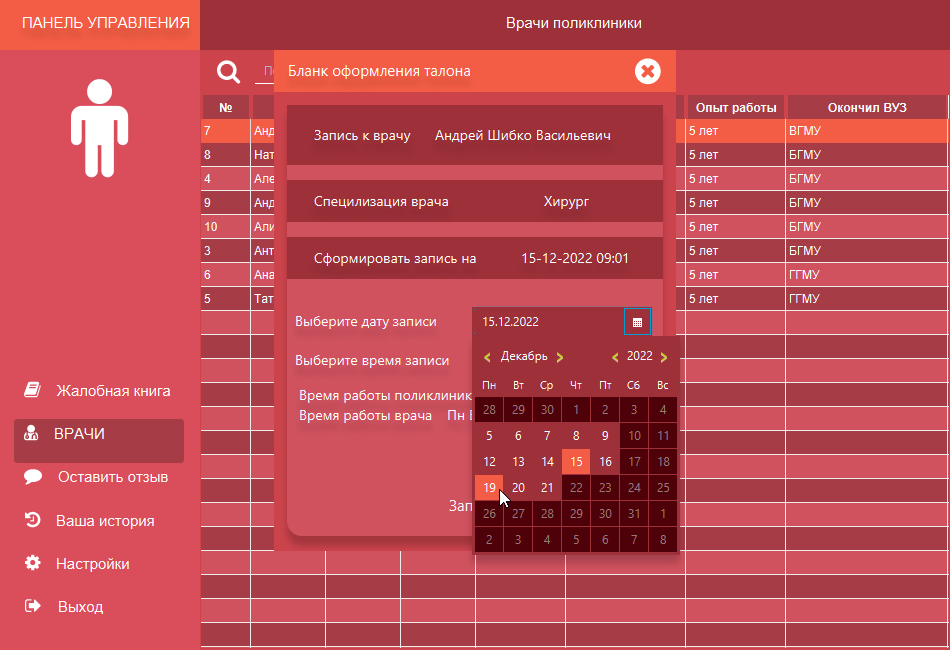


Рисунок 4.15 – Окно «Врачи»

В окне «Оставить отзыв» пользователь может написать свой отзыв на любого врача и выставить этому доктору рейтинг. Благодаря отзывам пациентов и формируется рейтинг врачей. Пример окна «Оставить отзыв» изображен на рисунке 4.16:



Рисунок 4.16 – Окно «Оставить отзыв»

Во вкладке «Ваша история» (рисунок 4.17) отображается история посещений пользователя. Показывается дата и время посещения, статус посещения, а также ФИО и специализация врача.

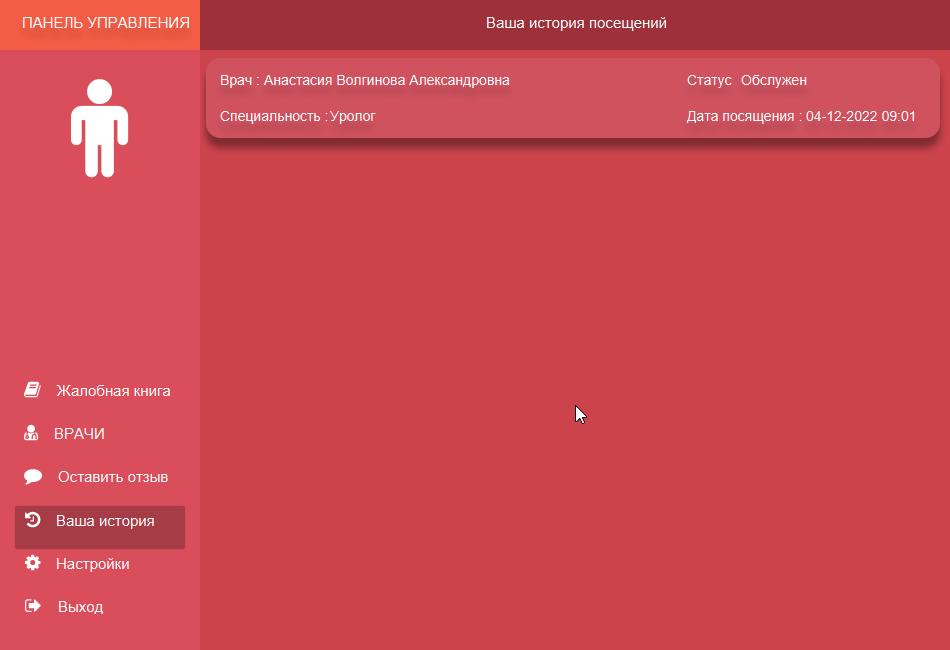


Рисунок 4.17 – Окно «Ваша история»

Последнее окно — это окно настроек (рисунок 4.18). В нем пользователь может отредактировать информацию о себе: поменять ФИО, номер телефон и остальную информацию.

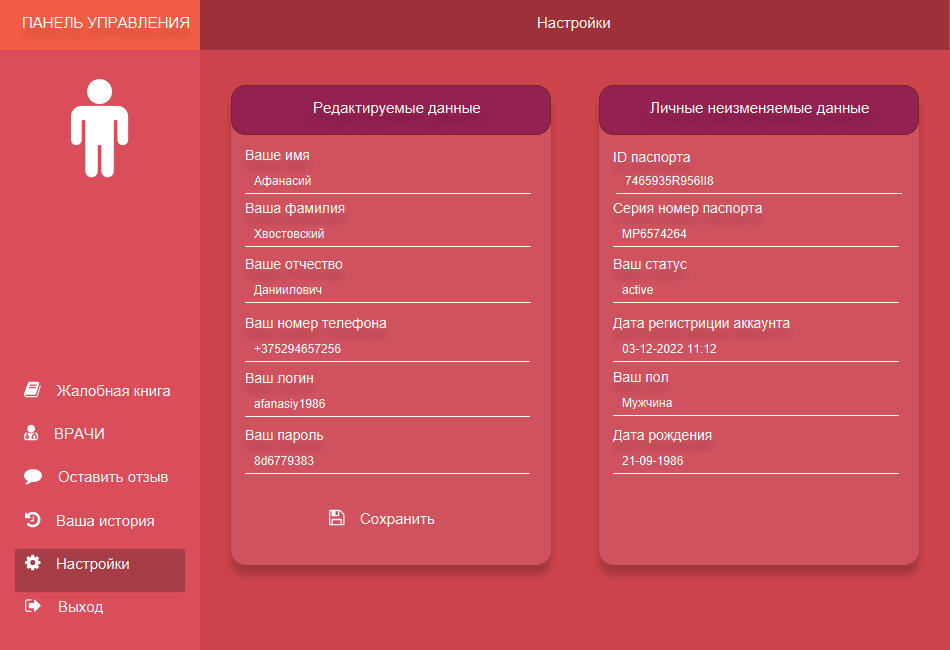


Рисунок 4.18 – Пример окна настроек

Далее рассмотрим функционал окон врача.

После входа врача встречает вкладка с рейтингом врачей. Отображаются только первые три доктора в рейтинге. На рисунке 4.19 показан пример окна рейтинга.

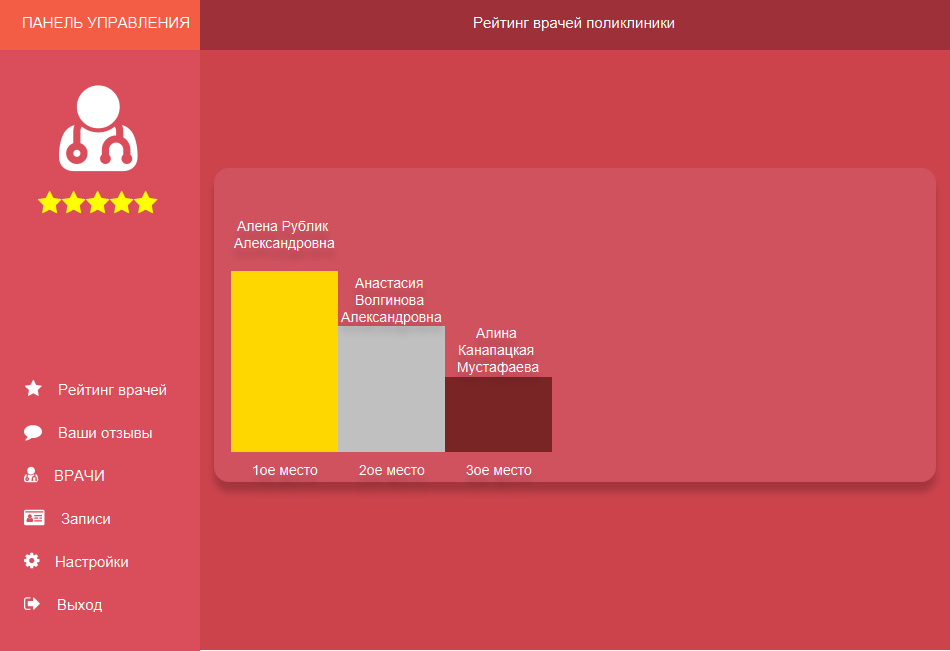


Рисунок 4.19 – Пример рейтинга врачей

На рисунке 4.20 показан пример окна отзывов. В данном окне врач может видеть отзывы, оставленные пациентами о нем, а также время выставления этого отзыва и оценку, влияющую на рейтинг этого врача.

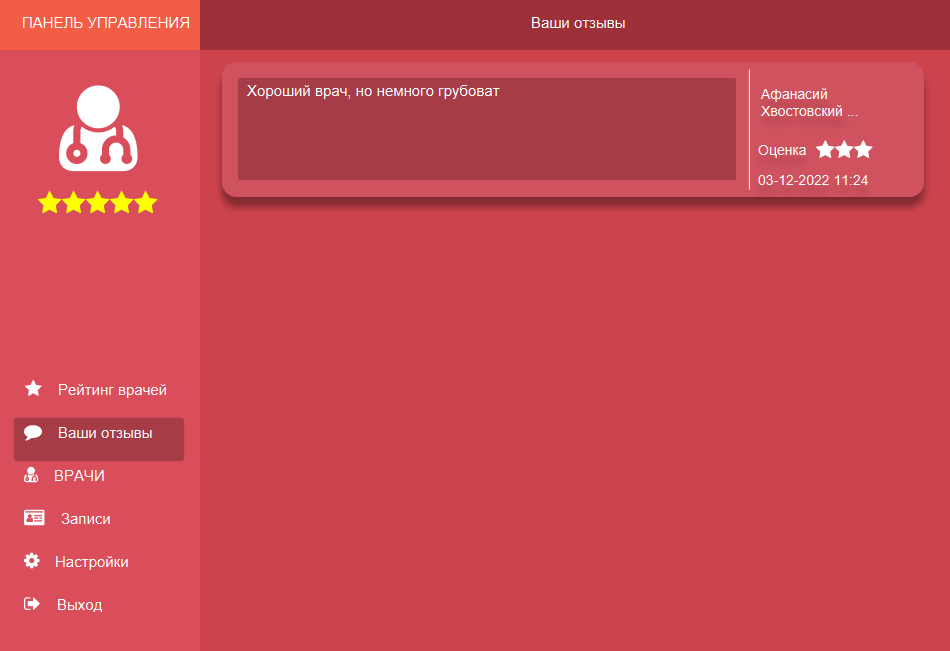


Рисунок 4.20 – Пример окна «Ваши отзывы»

В окне «Врачи» (рисунок 4.21) доктор может просмотреть информацию о коллегах, а также провести поиск по любому из параметров, чтобы быстро найти данные интересующего его врача.



Рисунок 4.21 – Окно «Врачи»

В окне записи отображаются записи врача как на все дни, так и только на текущий день. Пример окна записей изображен на рисунке 4.22:

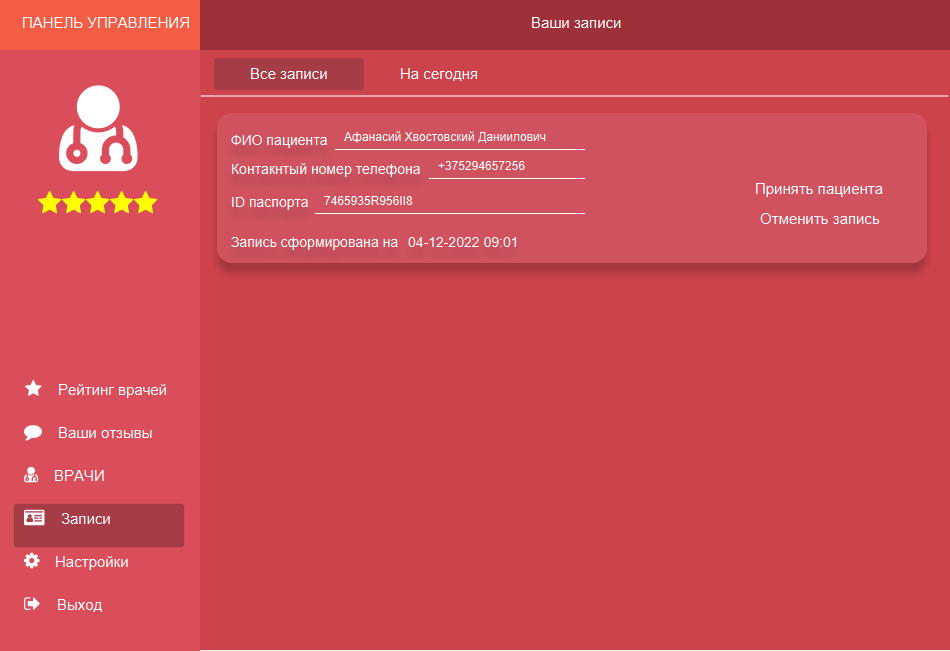


Рисунок 4.22 – Пример окна «Записи»

Последним окном является окно настроек. В нем доктор может не только изменить свои персональные данные, но и выставить рабочие дни. Окно «Настройки» представлено на рисунке 4.23:

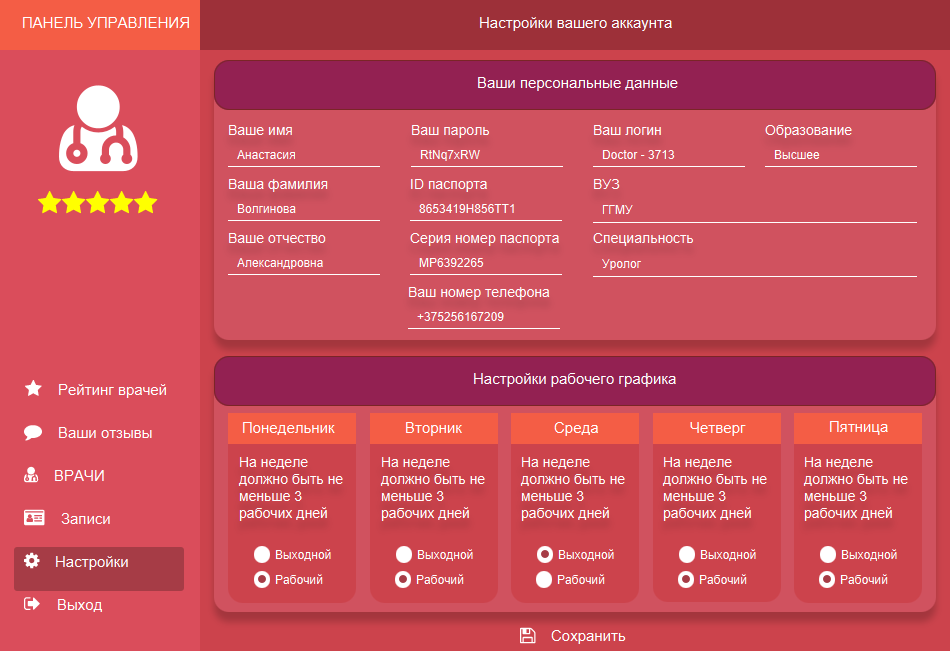


Рисунок 4.23 – Окно «Настройки»

Таким образом, в данном разделе был рассмотрен основной функционал приложения, составлено краткое руководство для пользователя.

**ЗАКЛЮЧЕНИЕ**

Современные медицинские организации производят и накапливают большие объемы информации, как о сотрудниках, так и о пациентах. Врачам нужно следить за посещением пациентов, а пациентам – иметь возможность записаться на прием, не прилагая значительных усилий. От того, насколько эффективно эта информация используется врачами, руководителями зависит качество медицинской помощи, общий уровень жизни населения. Поэтому необходимость использования больших, и при этом еще постоянно растущих, объемов информации при решении диагностических, терапевтических и управленческих задач обуславливает сегодня создание информационных систем в медицинских учреждениях.

Подводя итог, можно сказать, что в данной курсовой работе была изучена предметная область, проанализирована работа поликлиники и, в частности, регистратуры, рассмотрены программные аналоги, а также эффект от автоматизации системы для учета пациентов в поликлинике.

Также была поставлена задача, проведен обзор методов ее решения, опираясь на данные, полученные во время исследования предметной области, выделены и обоснованы положительные качества подходов. Рассмотрены паттерны проектирования, используемые в курсовом проекте.

Изучено функциональное моделирование на основе стандарта IDEF0, созданы контекстная диаграмма верхнего уровня, декомпозиция контекстной модели и каждого компонента.

Создана и описана информационная модель. Были выделены сущности и обоснован их выбор, как они участвует в реализации задуманного функционала, кратко описаны ключевые поля таблиц.

Также были описаны основные алгоритмы, реализующие бизнес-логику проектируемой системы. Создана диаграмма классов, компонентов и развертывания, последовательностей, состояний. Для каждой диаграммы было добавлено краткое описание.

Написано краткое руководство для пользователя и протестировано приложение. Подробно описаны все разделы меню для каждой из ролей.

Таким образом, все поставленные задачи были выполнены и результатом работы является многопоточное клиент-серверное приложение для оптимизации временных затрат сотрудников медучреждений и пациентов при заказе талонов на прием.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. https://studopedia.ru/ [Электронный ресурс] – Режим доступа:

<https://studopedia.ru/9_131784_organizatsiya-raboti-registraturi-polikliniki.html>– Дата доступа 18.10.2022

1. moggp6.by [Электронный ресурс] – Режим доступа: <https://moggp6.by/sovety/chto-takoe-registratura-v-poliklinike.html>Дата доступа: 25.10.2022
2. mobile.studbooks.net [Электронный ресурс] – Режим доступа:

[https://mobile.studbooks.net/1998559/informatika/proektirovanie\_avtomatizirovan nogo\_rabochego\_mesta\_registratury\_polikliniki](https://mobile.studbooks.net/1998559/informatika/proektirovanie_avtomatizirovannogo_rabochego_mesta_registratury_polikliniki) Дата доступа: 09.11.2022

1. kazedu.com [Электронный ресурс] – Режим доступа: <https://kazedu.com/referat/132773/4>Дата доступа: 09.11.2022
2. minsk.1cbit.ru [Электронный ресурс] – Режим доступа: <https://minsk.1cbit.ru/blog/avtomatizatsiya>[-meditsinskoy-kliniki/](https://minsk.1cbit.ru/blog/avtomatizatsiya-meditsinskoy-kliniki/) Дата доступа: 19.11.2022
3. talon.by [Электронный ресурс] – Режим доступа: <https://talon.by/about>Дата доступа: 19.11.2022
4. [www.jetbrains.com](http://www.jetbrains.com/) [Электронный ресурс] – Режим доступа:

https://www.jetbrains.com/ru-ru/idea/features/ Дата доступа: 21.11.2022

1. thecode.media [Электронный ресурс] – Режим доступа: <https://thecode.media/mysql/>Дата доступа: 28.11.2022
2. java-ru-blog.blogspot.com [Электронный ресурс] – Режим доступа:

<https://java-ru-blog.blogspot.com/2020/02/strategy-pattern-java.html> Дата доступа: 28.11.2022

## **Приложение А (обязательное)**

**Листинг кода**

**Client:**

**package** Client;  
*//класс подключения клиента***import** controllers.StartMenuController;  
  
**import** java.io.IOException;  
**import** java.io.ObjectInputStream;  
**import** java.io.ObjectOutputStream;  
**import** java.io.Serializable;  
**import** java.net.Socket;  
  
**public class** Client **implements** Serializable {  
 **private static** Client *singleton* = **new** Client();  
 **private int port** =8080;  
 **private** Socket **socket**;  
 **public void** CloseSocket() **throws** IOException {  
 **socket**.close();  
 }  
 **private** ObjectOutputStream **out**;  
 **private** ObjectInputStream **in**;  
 **public** ObjectOutputStream getOut(){**return out**;}  
 **public** ObjectInputStream getIn(){**return in**;}  
 **public static** Client getInstance()  
 {  
 **return** *singleton*;  
 }  
 **private** Client() {  
 }  
 **public void** connection() **throws** IOException, ClassNotFoundException {  
 **socket**=**new** Socket(**"localhost"**,**port**);  
 **out**=**new** ObjectOutputStream(**socket**.getOutputStream());  
 **in** =**new** ObjectInputStream(**socket**.getInputStream());  
 StartMenuController startMenuController=**new** StartMenuController();  
 startMenuController.Show();  
 }  
  
}

**MainClient:**

**package** controllers;  
*//включение клиента***import** Client.Client;  
**import** controllers.StartMenuController;  
**import** javafx.application.Application;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Scene;  
**import** javafx.stage.Stage;  
  
**import** java.io.IOException;  
  
**public class** MainClient **extends** Application {  
 @Override  
 **public void** start(Stage stage) **throws** IOException, ClassNotFoundException {  
 Client.*getInstance*().connection();  
 }  
  
 **public static void** main(String[] args) {  
 *launch*();  
 }  
}

**Server:**

**package** server;  
  
**import** controllers.StartMenu;  
  
**import** java.io.IOException;  
**import** java.io.ObjectInputStream;  
**import** java.io.ObjectOutputStream;  
**import** java.io.Serializable;  
**import** java.net.InetAddress;  
**import** java.net.ServerSocket;  
**import** java.net.Socket;  
**import** java.sql.SQLException;  
  
**public class** Server **implements** Serializable {  
 **private int port**=8080;  
 **private** ServerSocket **server** ;  
 **private** Socket **socket**;  
 **private** ObjectOutputStream **out**;  
 **private** ObjectInputStream **in**;  
 **public** ObjectOutputStream getOut(){**return out**;}  
 **public** ObjectInputStream getIn(){**return in**;}  
 **public** InetAddress getIpClient() { **return socket**.getInetAddress(); }  
 **public** Server() **throws** IOException {  
 **server**=**new** ServerSocket(**port**);  
 }  
 **public void** StartServer() **throws** IOException {  
 **while**(**true**) {  
 **socket** = **server**.accept();  
 **out** = **new** ObjectOutputStream(**socket**.getOutputStream());  
 **in**=**new** ObjectInputStream(**socket**.getInputStream());  
 System.***out***.println(**"Подкоючился : "**+getIpClient().toString());  
 **new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 StartMenu startMenu=**new** StartMenu();  
 **try** {  
 startMenu.Start(**in**,**out**);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **catch** (ClassNotFoundException e) {  
 e.printStackTrace();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }).start();  
  
 }  
 }  
}

**MainServer:**

**import** server.Server;  
  
**import** java.io.IOException;  
  
**public class** MainServer {  
 **public static void** main(String[] args) **throws** IOException {  
 Server server=**new** Server();  
 server.StartServer();  
 }  
}

**AdminMenu:**

**package** controllers;  
  
**import** datebase.Datebase;  
**import** message.Messages;  
**import** message.UserStaus;  
**import** models.\*;  
  
**import** java.io.IOException;  
**import** java.io.ObjectInputStream;  
**import** java.io.ObjectOutputStream;  
**import** java.sql.SQLException;  
**import** java.text.SimpleDateFormat;  
**import** java.time.DayOfWeek;  
**import** java.time.LocalDate;  
**import** java.time.format.DateTimeFormatter;  
**import** java.util.ArrayList;  
**import** java.util.Date;  
**import** java.util.List;  
  
**public class** AdminMenu {  
 **public void** Start(ObjectInputStream in, ObjectOutputStream out, User user) **throws** IOException, ClassNotFoundException, SQLException {  
 update(out,in,user);  
 **while** (**true**){  
 Messages messages=(Messages) in.readObject();  
 **switch** (messages){  
 **case *CREATE\_REPORT***:createReport(out, user);**break**;  
 **case *DELETED\_DOTCTOR***:deletedDoctor(in);**break**;  
 **case *UPDATE\_STATISTIC***:initializedStatistic(out);**break**;  
 **case *UPDATE\_BLACK\_LIST***:initializedBlackListUser(out);**break**;  
 **case *UPDATE\_COMPLAIN***:initializedComplaint(out);**break**;  
 **case *UPDATE\_DOCTOR***:initializedAccount(out,user);**break**;  
 **case *UNBLOCK\_ACCOUNT\_TABLE***:unblockAccountDoctorTable(in);**break**;  
 **case *UNBLOCK\_ACCOUNT***:unblockAccountDoctor(in);**break**;  
 **case *BLOCK\_ACCOUNT***:blockAccountDoctor(in,user);**break**;  
 **case *EXIT***:StartMenu startMenu=**new** StartMenu();startMenu.Start(in,out);**break**;  
 **case *REGISTRATION\_DOCTOR***:registrationDoctor(in);**break**;  
 **case *REGISTRATION\_ADMIN***:registrationAdmin(in);**break**;  
 **case *PERSONAL\_DOCTOR\_PAGE***:personalDoctorPage(in,out);**break**;  
 }  
 }  
 }  
 **private void** createReport(ObjectOutputStream out,User user) **throws** SQLException, ClassNotFoundException, IOException {  
 Datebase datebasea=**new** Datebase();  
 List<Doctor> doctorList=datebasea.full\_doctor();  
 **double** rating = 0;  
 **int** terapevt=0,hiryrg=0,stomotolog=0,gastro=0,travmtolog=0,nevrolog=0,yrolog=0;  
 **for** (**int** i=0;i<doctorList.size();i++){  
 **if**(doctorList.get(i).getSpeciality().equals(**"Терапевт"**))terapevt++;  
 **if**(doctorList.get(i).getSpeciality().equals(**"Хирург"**))hiryrg++;  
 **if**(doctorList.get(i).getSpeciality().equals(**"Стомотолог"**))stomotolog++;  
 **if**(doctorList.get(i).getSpeciality().equals(**"Гастроэнтеролог"**))gastro++;  
 **if**(doctorList.get(i).getSpeciality().equals(**"Травматолог"**))travmtolog++;  
 **if**(doctorList.get(i).getSpeciality().equals(**"Невролог"**))nevrolog++;  
 **if**(doctorList.get(i).getSpeciality().equals(**"Уролог"**))yrolog++;  
 rating=rating+doctorList.get(i).getRating();  
 }  
 rating=rating/doctorList.size();  
 String report=**"Отчет по поликлинике\n"**;  
 report=report+**"Раздел 'Аккаунты'\n"**;  
 report=report+**"Администраци : "**+datebasea.full\_admin().size()+**" Врачи : "**+datebasea.full\_doctor().size()+**" Пациенты = "**+datebasea.full\_patient().size()+**"\n"**;  
 report=report+**"Черный список : "**+datebasea.full\_blackList().size()+**"\n"**;  
 report=report+**"\n"**;  
 report=report+**"Раздел 'Отзывы и Жалобная книга' \n"**;  
 report=report+**"Кол-во жалоб : "**+datebasea.full\_complaint().size()+**" Средняя оценка "**+rating+**"\n"**;  
 report=report+**"Врачи поликлиники \*\*\* \n"**;  
 report=report+**"Невролог : "**+nevrolog+**"\n"**;  
 report=report+**"Уролог : "**+yrolog+**"\n"**;  
 report=report+**"Травматолог : "**+travmtolog+**"\n"**;  
 report=report+**"Гастроэнтеролог : "**+gastro+**"\n"**;  
 report=report+**"Стомотолог : "**+stomotolog+**"\n"**;  
 report=report+**"Хирург : "**+hiryrg+**"\n"**;  
 report=report+**"Терапевт : "**+terapevt+**"\n"**;  
 List<Records>List=datebasea.full\_Records();  
 **int** mon=0,tue=0,wen=0,th=0,fri=0;  
 DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*(**"dd-MM-yyyy HH:mm"**);  
 **for** (**int** i=0;i<List.size();i++){  
 LocalDate localDate=LocalDate.*parse*(List.get(i).getDateRecords(),formatter);  
 **if**(localDate.getDayOfWeek()== DayOfWeek.***MONDAY***)mon++;  
 **if**(localDate.getDayOfWeek()== DayOfWeek.***TUESDAY***)tue++;  
 **if**(localDate.getDayOfWeek()== DayOfWeek.***WEDNESDAY***)wen++;  
 **if**(localDate.getDayOfWeek()== DayOfWeek.***THURSDAY***)th++;  
 **if**(localDate.getDayOfWeek()== DayOfWeek.***FRIDAY***)fri++;  
 }  
 List<Integer>mas=**new** ArrayList<Integer>();  
 mas.add(mon);  
 mas.add(tue);  
 mas.add(wen);  
 mas.add(th);  
 mas.add(fri);  
 **int** max=0;  
 String day=**""**;  
 **for**(**int** i=0;i<mas.size();i++){  
 **if**(max<mas.get(i)){  
 max=mas.get(i);  
 **if**(i==0)day=**"Понедельник"**;  
 **if**(i==1)day=**"Вторник"**;  
 **if**(i==2)day=**"Среда"**;  
 **if**(i==3)day=**"Четверг"**;  
 **if**(i==4)day=**"Пятница"**;  
 }  
 }  
 Date date = **new** Date();  
 SimpleDateFormat simpl = **new** SimpleDateFormat(**"dd-MM-yyyy HH:mm"**);  
 String dat = simpl.format(date);  
 report=report+**"Самый загрежнный день на данный момент : "**+day+**"Кол-во талонов в день : "**+max+**"\n"**;  
 report=report+**"Дата создания отчет : "**+dat+**"\n"**;  
 Admin admin=datebasea.findAdminByIdUser(user.getId());  
 report=report+**"Собран : "**+admin.getName()+**" "**+admin.getSur\_name()+**" "**+admin.getPatronymic();  
 out.writeObject(report);  
 }  
 **private void** deletedDoctor(ObjectInputStream in) **throws** IOException, ClassNotFoundException, SQLException {  
 String id\_doctor=(String) in.readObject();  
 Datebase datebase=**new** Datebase();  
 datebase.removeDoctor(id\_doctor);  
 }  
 **private void** unblockAccountDoctorTable(ObjectInputStream in) **throws** SQLException, ClassNotFoundException, IOException {  
 Datebase datebase=**new** Datebase();  
 Doctor doctor=datebase.findDoctorByIdUser((**int**) in.readObject());  
 User user=datebase.findUserByID(doctor.getId\_user());  
 user.setStatus(UserStaus.*active*);  
 datebase.updateStatusAccount(user,UserStaus.*active*);  
 BlackList blackList=**new** BlackList();  
 blackList.setId\_user(user.getId());  
 datebase.removeAccountOnBlackList(blackList);  
 }  
 **private void** unblockAccountDoctor(ObjectInputStream in) **throws** SQLException, ClassNotFoundException, IOException {  
 Datebase datebase=**new** Datebase();  
 Doctor doctor=datebase.findDoctorByIDPassport((String) in.readObject());  
 User user=datebase.findUserByID(doctor.getId\_user());  
 user.setStatus(UserStaus.*active*);  
 datebase.updateStatusAccount(user,UserStaus.*active*);  
 BlackList blackList=**new** BlackList();  
 blackList.setId\_user(user.getId());  
 datebase.removeAccountOnBlackList(blackList);  
 }  
 **private void** blockAccountDoctor(ObjectInputStream in,User administator) **throws** IOException, ClassNotFoundException, SQLException {  
 Datebase datebase=**new** Datebase();  
 Admin admin=datebase.findAdminByIdUser(administator.getId());  
 Doctor doctor=datebase.findDoctorByIDPassport((String) in.readObject());  
 BlackList blackList=(BlackList)in.readObject();  
 User user=datebase.findUserByID(doctor.getId\_user());  
 user.setStatus(UserStaus.*ban*);  
 blackList.setId\_user(user.getId());  
 blackList.setReason(**"Заблокирован администратором : "**+admin.getName()+**" "**+admin.getSur\_name()+**" "**+admin.getPatronymic());  
 datebase.updateStatusAccount(user,UserStaus.*ban*);  
 datebase.addAccountOnBlackList(blackList);  
 }  
 **private void** update(ObjectOutputStream outputStream, ObjectInputStream inputStream ,User user) **throws** SQLException, IOException, ClassNotFoundException {  
 initializedBlackListUser(outputStream);  
 initializedAccount(outputStream,user);  
 initializedStatistic(outputStream);  
 initializedComplaint(outputStream);  
 }  
 **private void** personalDoctorPage(ObjectInputStream in ,ObjectOutputStream out) **throws** IOException, ClassNotFoundException, SQLException {  
 Datebase datebase=**new** Datebase();  
 Doctor doctor=datebase.findDoctorByIDPassport((String) in.readObject());  
 User user=datebase.findUserByID(doctor.getId\_user());  
 out.writeObject(doctor);  
 out.writeObject(user);  
 }  
 **private void** initializedComplaint(ObjectOutputStream out) **throws** SQLException, ClassNotFoundException, IOException {  
 Datebase datebase=**new** Datebase();  
 out.writeObject(datebase.full\_complaint());  
 }  
 **private void** initializedStatistic(ObjectOutputStream out) **throws** SQLException, ClassNotFoundException, IOException {  
 Datebase datebase=**new** Datebase();  
 List<Doctor> doctorList=datebase.full\_doctor();  
 List<Patient> patientList=datebase.full\_patient();  
 List<Admin>listAdmin=datebase.full\_admin();  
 out.writeObject(listAdmin.size());  
 out.writeObject(doctorList.size());  
 out.writeObject(patientList.size());  
 **double** rating = 0;  
 **int** terapevt=0,hiryrg=0,stomotolog=0,gastro=0,travmtolog=0,nevrolog=0,yrolog=0;  
 **for** (**int** i=0;i<doctorList.size();i++){  
 **if**(doctorList.get(i).getSpeciality().equals(**"Терапевт"**))terapevt++;  
 **if**(doctorList.get(i).getSpeciality().equals(**"Хирург"**))hiryrg++;  
 **if**(doctorList.get(i).getSpeciality().equals(**"Стомотолог"**))stomotolog++;  
 **if**(doctorList.get(i).getSpeciality().equals(**"Гастроэнтеролог"**))gastro++;  
 **if**(doctorList.get(i).getSpeciality().equals(**"Травматолог"**))travmtolog++;  
 **if**(doctorList.get(i).getSpeciality().equals(**"Невролог"**))nevrolog++;  
 **if**(doctorList.get(i).getSpeciality().equals(**"Уролог"**))yrolog++;  
 rating=rating+doctorList.get(i).getRating();  
 }  
 rating=rating/doctorList.size();  
 out.writeObject(Double.*toString*(rating).substring(0,Double.*toString*(rating).indexOf(**'.'**)+2));  
 out.writeObject(terapevt);  
 out.writeObject(hiryrg);  
 out.writeObject(yrolog);  
 out.writeObject(nevrolog);  
 out.writeObject(travmtolog);  
 out.writeObject(gastro);  
 out.writeObject(stomotolog);  
  
 List<Records>List=datebase.full\_Records();  
 **int** mon=0,tue=0,wen=0,th=0,fri=0;  
 DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*(**"dd-MM-yyyy HH:mm"**);  
 **for** (**int** i=0;i<List.size();i++){  
 LocalDate localDate=LocalDate.*parse*(List.get(i).getDateRecords(),formatter);  
 **if**(localDate.getDayOfWeek()== DayOfWeek.***MONDAY***)mon++;  
 **if**(localDate.getDayOfWeek()== DayOfWeek.***TUESDAY***)tue++;  
 **if**(localDate.getDayOfWeek()== DayOfWeek.***WEDNESDAY***)wen++;  
 **if**(localDate.getDayOfWeek()== DayOfWeek.***THURSDAY***)th++;  
 **if**(localDate.getDayOfWeek()== DayOfWeek.***FRIDAY***)fri++;  
 }  
 out.writeObject(mon);  
 out.writeObject(tue);  
 out.writeObject(wen);  
 out.writeObject(th);  
 out.writeObject(fri);  
 }  
 **private void** initializedBlackListUser(ObjectOutputStream outputStream) **throws** SQLException, ClassNotFoundException, IOException {  
 Datebase datebase=**new** Datebase();  
 List<BlackList> blackLists=datebase.full\_blackList();  
 outputStream.writeObject(blackLists);  
 }  
 **private void** initializedAccount(ObjectOutputStream out,User user) **throws** SQLException, ClassNotFoundException, IOException {  
 Datebase datebase=**new** Datebase();  
 Admin admin=datebase.findAdminByIdUser(user.getId());  
 List<Doctor> listDoctor=datebase.full\_doctor();  
 List<Patient> listPatient=datebase.full\_patient();  
 out.writeObject(user);  
 out.writeObject(admin);  
 out.writeObject(listDoctor);  
 out.writeObject(listPatient);  
 }  
 **private void** registrationDoctor(ObjectInputStream in) **throws** IOException, ClassNotFoundException, SQLException {  
 User user=(User) in.readObject();  
 Doctor doctor=(Doctor) in.readObject();  
 Work\_day work\_day=**new** Work\_day();  
 work\_day.setId\_passport(doctor.getID\_passport());  
 user.setPosition(UserStaus.*doctor*);  
 Datebase datebase=**new** Datebase();  
 datebase.registrationDoctorAccount(user,doctor,work\_day);  
 }  
 **private void** registrationAdmin(ObjectInputStream in) **throws** IOException, ClassNotFoundException, SQLException {  
 User user=(User) in.readObject();  
 Admin admin=(Admin) in.readObject();  
 user.setPosition(UserStaus.*admin*);  
 Datebase datebase=**new** Datebase();  
 datebase.registrationAdminAccount(user,admin);  
 }  
 **private void** updateStatistic(ObjectOutputStream outputStream){  
  
 }  
}

# **Приложение Б (обязательное)**

**Листинг скрипта генерации базы данных**

**create schema** hospital;  
**insert into** hospital.user( **login**, **password**, **status**, **position**) **VALUES** (**'log'**,**'pas'**,**'active'**,**'admin'**);  
**insert into** hospital.administration(**ID\_passport**, **seria\_number**, **date\_registration**, **id\_user**, **name**, **surname**, **patronymic**, **phone**) **VALUES** (**'1234555A123AA1'**,**'MP1234567'**,**'10.10.2022 15:25'**, (**SELECT** *MAX*(hospital.user.**id**) **FROM** hospital.user),**'Виолетта'**,**'Писакова'**,**'Олеговна'**,**'+375293530749'**);  
**create table** hospital.user(  
 **`id` int** auto\_increment **not null** ,  
 **`login` varchar**(45) **not null** ,  
 **`password` varchar**(45) **not null** ,  
 **`status` varchar**(45) **not null** ,  
 **`position` varchar**(45) **not null** ,  
 **primary key** (**id**)  
);  
**create table** hospital.administration(  
 **`ID\_passport` varchar**(25) **not null** ,  
 **`seria\_number` varchar**(15) **not null** ,  
 **`date\_registration` varchar**(25) **not null** ,  
 **`id\_user` int not null** ,  
 **`name` varchar**(45) **not null** ,  
 **`surname` varchar**(45) **not null** ,  
 **`patronymic` varchar**(45) **not null** ,  
 **`phone` varchar**(45) **not null** ,  
 **foreign key** (**`id\_user`**) **references user**(**id**) **on delete cascade**,  
 **primary key** (**ID\_passport**)  
);  
**create table** hospital.doctor(  
 **`ID\_passport` varchar**(25) **not null** ,  
 **`seria\_number` varchar**(15) **not null** ,  
 **`date\_registration` varchar**(25) **not null** ,  
 **`id\_user` int not null** ,  
 **`name` varchar**(45) **not null** ,  
 **`surname` varchar**(45) **not null** ,  
 **`patronymic` varchar**(45) **not null** ,  
 **`phone` varchar**(45) **not null** ,  
 **`education` varchar**(25) **not null** ,  
 **`speciality` varchar**(50) **not null** ,  
 **`expiriense` varchar**(20) **not null** ,  
 **`university` varchar**(80) **not null** ,  
 **`rating` double not null** ,  
 **foreign key** (**`id\_user`**) **references user**(**id**) **on delete cascade**,  
 **primary key** (**ID\_passport**)  
);  
**create table** hospital.patient(  
 **`ID\_passport` varchar**(25) **not null** ,  
 **`seria\_number` varchar**(15) **not null** ,  
 **`date\_registration` varchar**(25) **not null** ,  
 **`id\_user` int not null** ,  
 **`name` varchar**(45) **not null** ,  
 **`surname` varchar**(45) **not null** ,  
 **`patronymic` varchar**(45) **not null** ,  
 **`phone` varchar**(45) **not null** ,  
 **`sex` varchar**(15) **not null** ,  
 **`date\_birth` varchar**(16) **not null** ,  
 **foreign key** (**`id\_user`**) **references user**(**id**) **on delete cascade**,  
 **primary key** (**ID\_passport**)  
);  
**create table** hospital.blockList(  
 **`id` int** auto\_increment **not null** ,  
 **`id\_user` int not null** ,  
 **`reason` varchar**(500) **not null** ,  
 **`dateBlock` varchar**(25) **not null** ,  
 **primary key** (**id**),  
 **foreign key** (**`id\_user`**) **references user**(**id**) **on delete cascade**);  
**create table** hospital.work\_day(  
 **`id` int** auto\_increment **not null** ,  
 **`id\_doctor\_passport` varchar**(45) **not null** ,  
 **`monday`** boolean **not null** ,  
 **`tuesday`** boolean **not null** ,  
 **`wednesday`** boolean **not null** ,  
 **`thursday`** boolean **not null** ,  
 **`friday`** boolean **not null** ,  
 **primary key** (**id**),  
 **foreign key** (**`id\_doctor\_passport`**) **references** doctor(**ID\_passport**) **on delete cascade**);  
**create table** hospital.complaintBook(  
 **`id` int** auto\_increment,  
 **`id\_user` int** ,  
 **`complaint` varchar**(1000) **not null** ,  
 **`fio` varchar**(100) **not null** ,  
 **`date` varchar**(25),  
 **primary key** (**id**),  
 **foreign key** (**id\_user**) **references user**(**id**) **ON DELETE SET NULL**);  
**create table** hospital.review(  
 **`id` int** auto\_increment,  
 **`id\_user` int** ,  
 **`id\_doctor` varchar**(45) ,  
 **`review` varchar**(1000) **not null** ,  
 **`rating` int not null** ,  
 **`date` varchar**(25) **not null** ,  
 **primary key** (**id**),  
 **foreign key** (**id\_user**) **references user**(**id**) **ON DELETE SET NULL**,  
**foreign key** (**id\_doctor**) **references** doctor(**ID\_passport**) **ON DELETE SET NULL**);  
**create table** hospital.ticket(  
 **`id` int** auto\_increment,  
 **`idUser` int** ,  
 **`idDoctor` varchar**(45),  
 **`status` varchar**(45),  
 **`date` varchar**(45) **not null** ,  
 **`dateRecords` varchar**(45) **not null** ,  
 **primary key** (**`id`**),  
 **foreign key** (**idUser**) **references** hospital.user(**id**) **on delete cascade**,  
 **foreign key** (**idDoctor**) **references** hospital.doctor(**ID\_passport**) **on delete cascade**);  
**create table** hospital.history(  
 **`id` int** auto\_increment,  
 **`idUser` int** ,  
 **`idDoctor` varchar**(45),  
 **`status` varchar**(45),  
 **`dateRecords` varchar**(45) **not null** ,  
 **`FIOdoctor` varchar**(100) **not null** ,  
 **`speciality` varchar**(100) **not null** ,  
 **primary key** (**`id`**),

**foreign key** (**idUser**) **references** hospital.user(**id**) **ON DELETE SET NULL**,  
**foreign key** (**idDoctor**) **references** hospital.doctor(**ID\_passport**) **ON DELETE SET NULL**);