

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ**  
**по дисциплине «Введение в нереляционные базы данных»**  
**Тема: База данных стоматологических карточек**

Студенты гр. 2381

Богатов И.А.  
Газукина Д.Д.  
Дудкин М.В.

Студенты гр. 2382

Вакуленко И.Ю.  
Ваньков Я.С.

Преподаватель

Заславский М.М.

Санкт-Петербург  
2025

## **ЗАДАНИЕ**

### **Студенты**

Богатов И.А. 2381

Газукина Д.Д. 2381

Дудкин М.В. 2381

Вакуленко И.Ю. 2382

Ваньков Я.С. 2382

**Тема проекта:** Разработка сервиса для ведения истории приемов в стоматологии.

### **Исходные данные:**

Приложение необходимо реализовать с использованием нереляционной базы данных MongoDB и синтетических данных.

### **Содержание пояснительной записки:**

«Содержание»

«Введение»

«Сценарий использования»

«Модель данных»

«Разработанное приложение»

«Выводы»

«Приложение»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц

Дата выдачи задания: 11.02.2025

Дата сдачи реферата: 01.06.2025

Студенты гр. 2381

Богатов И.А.  
Газукина Д.Д.  
Дудкин М.В.

Студенты гр. 2382

Вакуленко И.Ю.  
Ваньков Я.С.

Преподаватель

Заславский М.М.

## **АННОТАЦИЯ**

В рамках данного курса было разработано веб-приложение для ведения стоматологических карточек пациентов с использованием Django и нереляционной базы данных MongoDB. Данная тема была выбрана ввиду её прикладной значимости и потенциала для использования в сфере частной медицины. Основное внимание уделялось созданию удобного и интуитивно понятного интерфейса, а также построению простой и расширяемой архитектуры.

Исходный код и дополнительную информацию можно найти по ссылке: <https://github.com/moevm/nosql1h25-dentistry>

## **SUMMARY**

As part of this course, a web application for managing dental patient records was developed using Django and the non-relational database MongoDB. This topic was chosen due to its practical relevance and potential application in the field of private healthcare. The main focus was on creating a user-friendly and intuitive interface, as well as building a simple and scalable architecture.

The source code and additional information are available at: <https://github.com/moevm/nosql1h25-dentistry>

## Оглавление

1. Введение.....	6
2. Сценарии использования.....	8
2.1. Макет UI.....	8
2.2. Сценарии использования для задачи импорта данных.....	8
2.3. Сценарии использования для задачи представления данных.....	9
2.4. Сценарии использования для задачи анализа данных.....	10
2.5. Сценарии использования для задачи экспорта данных.....	10
2.6. Вывод о преобладающих операциях.....	11
3. Модель данных.....	12
3.1. Нереляционная модель данных MongoDB.....	12
3.1.1. Описание назначений коллекций, типов данных и сущностей.....	12
3.1.2. Оценка объема информации, хранимой в модели.....	13
3.1.3. Избыточность данных.....	14
3.1.4. Направление роста модели при увеличении количества объектов каждой сущности.....	15
3.1.5. Примеры данных.....	15
3.1.6. Примеры запросов.....	16
3.1.7. Количество запросов.....	20
3.2 Реляционная модель данных.....	21
3.2.1. Описание назначений коллекций, типов данных и сущностей.....	21
3.2.2. Оценка объема информации, хранимой в модели.....	23
3.2.3. Избыточность данных.....	23
3.2.4. Направление роста модели при увеличении количества объектов каждой сущности.....	24
3.2.5. Примеры данных.....	24
3.2.6. Примеры запросов.....	25
3.2.7. Количество запросов.....	28
3.3. Сравнение моделей.....	29
3.3.1. Удельный объем информации.....	29
3.3.2. Запросы и их сложность.....	29
3.3.3. Итоговое сравнение по объёму и запросам.....	29
3.3.4. Вывод.....	30
4. Разработанное приложение.....	31
5. Выводы.....	38
6. Приложения.....	39
7. Литература.....	40

## **1. Введение**

### **Актуальность решаемой проблемы**

Современная медицинская практика требует эффективного и централизованного способа хранения и обработки информации о пациентах. В стоматологических клиниках до сих пор часто используются бумажные карточки, что замедляет процесс обслуживания, увеличивает вероятность ошибок, а также усложняет доступ к истории лечения. Электронные медицинские карты решают эту проблему, позволяя врачу быстро получать информацию о пациенте, добавлять новые записи, а пациенту — отслеживать собственную историю лечения.

Однако большинство готовых решений являются либо закрытыми, либо платными, и часто не учитывают специфики стоматологической отрасли, особенно в условиях небольших или частных клиник. Это создаёт необходимость разработки специализированного, удобного и доступного инструмента, адаптированного под нужды конкретных пользователей.

### **Постановка задачи**

Целью проекта является разработка веб-приложения для хранения и управления стоматологическими карточками, которое позволит:

- Пациенту просматривать свою медицинскую карту и записываться на приём;
- Врачу работать с информацией о пациентах и вносить записи в карту;
- Администратору управлять списками врачей и пациентов, а также просматривать статистику загруженности;
- Всем ролям — получать доступ к необходимой информации в понятной и визуально доступной форме.

Приложение должно обеспечивать авторизацию, фильтрацию и поиск записей, экспорт и импорт данных, а также отображение статистики по приёмам.

### **Предлагаемое решение**

В качестве серверного фреймворка для разработки выбран Django —

зрелый и функционально богатый инструмент, обеспечивающий удобную архитектуру, быстрый запуск проекта и поддержку ролевой модели доступа.

СУБД MongoDB была выбрана как обязательное условие в рамках учебного проекта. Несмотря на это, её особенности хорошо подходят для задач, связанных с медицинскими данными. MongoDB представляет собой документоориентированную базу данных, что позволяет хранить информацию в гибком и расширяемом формате (JSON-подобные документы). Это особенно актуально при работе с медицинскими картами, поскольку структура записей может отличаться в зависимости от пациента, врача или конкретного приёма.

Кроме того, MongoDB упрощает реализацию функций импорта и экспорта, а также облегчает обработку вложенных данных (например, истории приёмов, динамики состояний, процедур и назначений). Всё это делает её подходящей технологией в рамках поставленной задачи.

Пользовательский интерфейс реализован с учётом трёх ключевых ролей — пациента, стоматолога и администратора. Каждая из ролей имеет доступ только к релевантной информации и функционалу, что обеспечивает как безопасность, так и удобство взаимодействия с системой.

### **Качественные требования к решению**

- Надёжность — корректная работа всех функций, включая записи, экспорт, фильтрацию и отображение истории лечения;
- Юзабилити — интуитивно понятный интерфейс для всех типов пользователей (в том числе людей без технических навыков);
- Безопасность — разграничение прав доступа и защита персональных данных пациентов;
- Масштабируемость — возможность расширения функциональности, например, добавления новых ролей, интеграции с календарями, напоминаний и других сервисов;
- Поддержка импорта и экспорта — возможность загрузки и выгрузки

данных в машиночитаемых форматах (JSON, CSV и т.д.) для удобства обмена данными между системами.

## 2. Сценарии использования

### 2.1. Макет UI

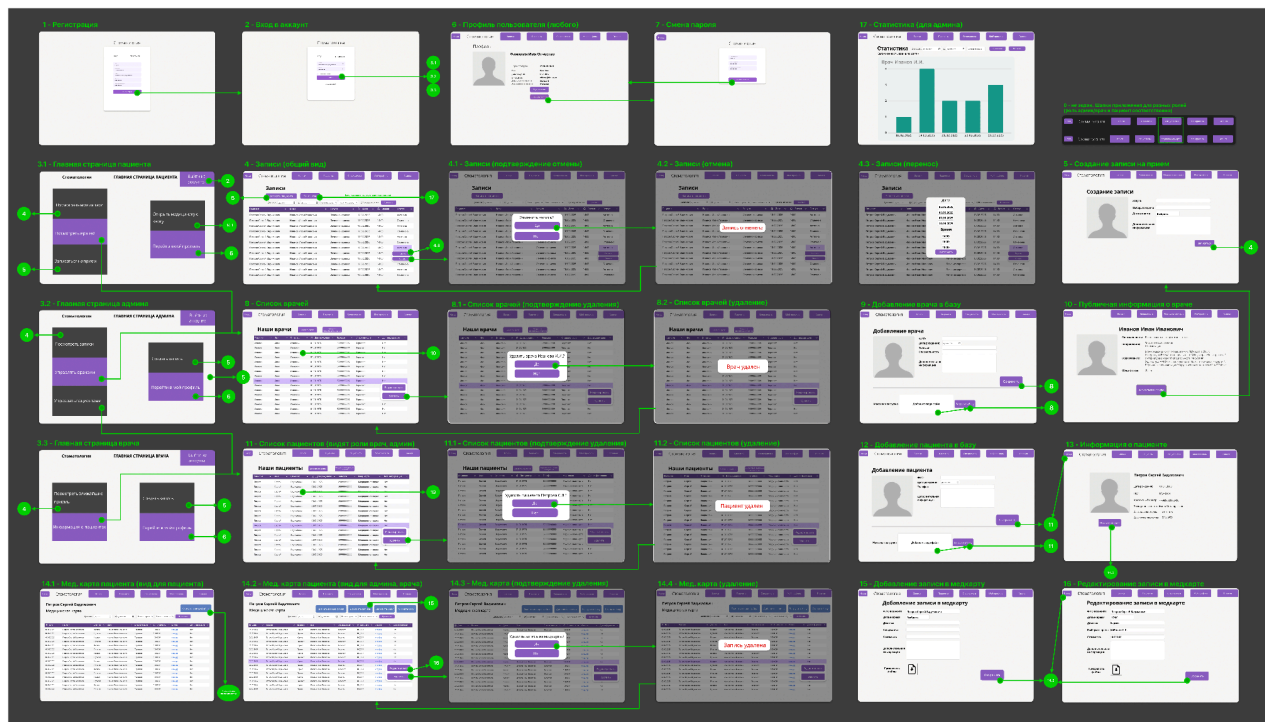


Рисунок 1 — UI-макет приложения

### 2.2. Сценарии использования для задачи импорта данных

#### 1. Массовый импорт базы пациентов (врачей)

Основной сценарий:

- 1.1. Переход по кнопке “Управлять пациентами (врачами)”
- 1.2. Администратор нажимает “Добавить пациента (врача)”
- 1.3. Прикрепляет файл с базой, которую необходимо импортировать
- 1.4. Подтверждает действие, происходит переход на страницу со всеми пациентами (врачами)

Альтернативный сценарий:

Ошибка: админ прикрепил некорректный формат файла

#### 2. Ручное добавление нового врача (пациента)

Основной сценарий:



2.1. Переход по кнопке “Управлять пациентами (врачами)” или “Пациенты (Специалисты)”

2.2. Администратор нажимает “Добавить пациента (врача)”

2.3. Заполняет данные и нажимает “Сохранить”, происходит переход на страницу со всеми пациентами (врачами)

Альтернативный сценарий:

Ошибка: пациент уже создан, выведется соответствующее сообщение

### **2.3. Сценарии использования для задачи представления данных**

*1. Просмотр записей к врачам (для admin доступны все, для patient и dentist — свои)*

Основной сценарий:

1.1. Авторизация при первом посещении

1.2. Переход с Главной страницы по кнопке “Посмотреть записи” или с любой другой страницы по кнопке “Записи”

1.3. Выставление фильтров, если это необходимо (диапазон дат, поиск по врачу и т.д.)

1.4. Просмотр необходимой информации

Альтернативный сценарий:

Не найдены записи в выбранном диапазоне дат

*2. Получение информации о конкретном враче (роль patient)*

Основной сценарий:

2.1. Переход по кнопке “Посмотреть врачей” или “Специалисты” на страницу со всеми врачами

2.2. Поиск врача по ФИО, выставление фильтра

2.3. Нажатие на конкретного врача, открывается его профиль с данными о специализации, опыте и образовании

Альтернативный сценарий:

Не найдено ФИО врача

*3. Просмотр ближайших приемов (роль dentist)*

Основной сценарий:

3.1. Переход с Главной страницы по кнопке “Посмотреть ближайшие приемы” или с любой другой страницы по кнопке “Записи”

3.2. Выставление фильтров, если это необходимо (диапазон дат, поиск по пациенту и т.д.)

3.3. Просмотр необходимой информации

Альтернативный сценарий:

Ошибка: в выбранном диапазоне дат нет приемов

## **2.4. Сценарии использования для задачи анализа данных**

### *1. Просмотр загруженности врача по дням (роль admin)*

Основной сценарий:

1.1. Переход по кнопке “Записи”

1.2. По нажатию на кнопку “Статистика” переход на экран со статистикой загруженности

1.3. Выбор врача и диапазона дат

1.4. Нажатие на кнопку “Применить”

1.5. Просмотр столбчатой диаграммы статистики

1.6. Если необходимо, администратор может скачать статистику по кнопке “Скачать”

Альтернативный сценарий:

Админ выбрал даты без записей, статистика пуста (0 записей на каждую дату)

## **2.5. Сценарии использования для задачи экспорта данных**

### *1. Просмотр и скачивание своей медицинской карты (роль patient)*

Основной сценарий:

1.1. Переход по кнопке “Открыть медицинскую карту”

1.2. Выставление фильтров, если это необходимо (диапазон дат, поиск по врачу и т.д.)

1.3. Пациент нажимает “Скачать медкарту”, происходит скачивание файла

Альтернативный сценарий:

Ошибка при скачивании файла — таблица не может быть пустой

## *2. Массовый экспорт базы пациентов (врачей) (роль admin)*

Основной сценарий:

- 2.1. Переход по кнопке “Управлять пациентами (врачами)”
- 2.2. Выставление необходимых фильтров для получения статистики
- 2.3. Администратор нажимает на кнопку “Выгрузить информацию о пациентах (врачах)”
- 2.4. Происходит скачивание файла

Альтернативный сценарий:

Ошибка: таблица не может быть пустой, нужно изменить фильтры

### **2.6. Вывод о преобладающих операциях**

В приложении будут преобладать операции чтения, так как основная часть пользовательского взаимодействия связана с просмотром информации — записей, медицинских карт, данных о врачах и пациентах, статистики по приёмам. Операции записи используются реже и чаще всего выполняются администраторами и врачами.

### 3. Модель данных

#### 3.1. Нереляционная модель данных MongoDB

##### JSON-схема CustomUser

```
{
  "_id": "ObjectId",
  "first_name": "String",
  "second_name": "String",
  "third_name": "String (nullable)",
  "email": "String", // уникальный
  "password_hash": "String",
  "avatar": "String (nullable)",
  "additional_info": "Object", // произвольный вложенный JSON-объект
  "role_id": "Integer", // 1 = admin, 2 = patient, 3 = dentist
  "phone": "String", // формат: +7 900 000 00 00
  "birth_date": "Date",
  "dt_add": "Date",
  "dt_upd": "Date"
}
```

##### JSON-схема Record

```
{
  "_id": "ObjectId",
  "dentist_id": "ObjectId", // ссылка на CustomUser._id
  "patient_id": "ObjectId", // ссылка на CustomUser._id
  "status": "Integer", // 1 = scheduled, 2 = completed, 3 = canceled
  "diagnosis": "String",
  "price": "Integer",
  "notes": "String (nullable)",
  "files": "[String]",
  "dt_add": "Date",
  "dt_upd": "Date",
  "dt_rec": "Date" // фактическая дата приема
}
```

Рисунок 2 — JSON-схема нереляционной модели данных

##### 3.1.1. Описание назначений коллекций, типов данных и сущностей

###### CustomUser

- `_id` (ObjectId, уникальный id) — 12 байт
- `first_name` (строка, имя) — в среднем 30 байт
- `second_name` (строка, фамилия) — в среднем 30 байт
- `third_name` (строка, отчество) — в среднем 30 байт

- email (строка, адрес электронной почты) — в среднем 30 байт
- password\_hash (строка, хэш пароля) — в среднем 60 байт
- avatar (строка, ссылка на файл) — в среднем 100 байт
- additional\_info (произвольный JSON, доп информация) — в среднем 200 байт
- role\_id (число, код роли пользователя) — 4 байта
- phone (строка, номер телефона) — 11 байт
- birth\_date (дата, дата рождения) — 10 байт
- dt\_add (дата, дата регистрации пользователя) — 8 байт
- dt\_upd (дата, дата последнего обновления информации) — 8 байт

### Record

- \_id (ObjectId) — 12 байт
- dentist\_id (ObjectId) — 12 байт
- patient\_id (ObjectId) — 12 байт
- status (число, статус записи) — 4 байта
- diagnosis (строка, диагноз) — в среднем 100 байт
- price (число, стоимость) — 4 байта
- notes (строка или null, дополнительная информация) — в среднем 300 байт
- files (список строк, ссылки на прикрепленные файлы) — в среднем 100 байт
- dt\_add (дата, дата создания записи) — 8 байт
- dt\_upd (дата, дата последнего обновления информации) — 8 байт
- dt\_rec (дата, дата приема у врача) — 8 байт

### 3.1.2. Оценка объема информации, хранимой в модели

- N — количество пользователей
- M — количество записей
- U — средний размер документа CustomUser 523 байта

- R — средний размер документа Record 568 байт Пусть доля врачей среди всех пользователей — 5%, пациентов — 94%, админов — 1%. Допустим средний врач работает 20 дней в месяц и принимает в день 6 пациентов. Записи хранятся 5 лет, затем архивируются. Тогда зависимость количества записей от количества пользователей  $M(N) = N * 0.05 * 6 * 20 * 12 * 5 = 360 * N$ . Общий объем базы данных:  $V(N) = N * U + M(N) * R = 205\,003 * N$  байт

### 3.1.3. Избыточность данных

В модели данных NoSQL, коллекция CustomUser имеет средний размер документа 523 байта. При исключении из расчёта таких избыточных элементов, как идентификатор `_id` (12 байт), поле `role_id` (4 байта) и поле `additional_info` (200 байт), остаётся 307 байт полезной нагрузки, несущей уникальную и значимую информацию о пользователе.

Аналогично, для коллекции Record, где средний размер записи составляет 568 байт, исключаются поля `_id` (12 байт), `dentist_id` (12 байт), `patient_id` (12 байт), `status` (4 байта) и `diagnosis` (100 байт). В результате этого остаётся 428 байт полезных данных на каждую запись.

Если принять, что на одного пользователя приходится в среднем 360 записей (что вытекает из предположения о 5-летней истории приёма врачей), то итоговый объём информации без учёта избыточных данных можно выразить как  $V^*(N) = 307 \times N + 428 \times 360 \times N$ . Это упрощается до  $V^*(N) = 154387 \times N$  байт, где N — количество пользователей.

Для сравнения, полный объём без оптимизации составляет  $205003 \times N$  байт. Таким образом, коэффициент избыточности модели составляет примерно 1.33. Это означает, что около 33% данных в исходной NoSQL-модели являются избыточными, и могут быть исключены при оптимизации без потери семантически важной информации.

### 3.1.4. Направление роста модели при увеличении количества объектов каждой сущности

Когда речь идет о масштабировании модели, необходимо учитывать, как увеличение количества объектов каждой сущности влияет на структуру и поведение базы данных. В данной модели при увеличении числа пользователей или записей наблюдается линейный рост количества документов в базе данных. Это означает, что по мере добавления новых пользователей или записей система сохраняет свою структуру и принципы работы, добавляя новые единицы данных без значительного изменения общего порядка обработки. При этом средний размер каждого документа остается неизменным.

### 3.1.5. Примеры данных

CustomUser — Администратор

```
{
  "_id": ObjectId("6611f3c34f1e8b5f6b1a0001"),
  "first_name": "Анна",
  "second_name": "Иванова",
  "third_name": "Петровна",
  "email": "admin@clinic.ru",
  "password_hash": "$2b$12$abc123abc123abc123abc1",
  "avatar": null,
  "additional_info": {
    "department": "Главный офис"
  },
  "role_id": 1,
  "phone": "+7 900 111 22 33",
  "birth_date": ISODate("1985-03-15T00:00:00Z"),
  "dt_add": ISODate("2025-04-06T08:00:00Z"),
  "dt_upd": ISODate("2025-04-06T08:00:00Z")
}
```

CustomUser — Пациент

```
{
  "_id": ObjectId("6611f3c34f1e8b5f6b1a0003"),
  "first_name": "Елена",
  "second_name": "Кузнецова",
  "third_name": null,
  "email": "elena.kuznetsova@gmail.com",
  "password_hash": "$2b$12$ghi789ghi789ghi789ghi7",
  "avatar": null,
  "additional_info": {
    "insurance_number": "1234-5678-9012"
  },
  "role_id": 2,
  "phone": "+7 902 333 44 55",
}
```

```

"birth_date": ISODate("1995-12-05T00:00:00Z")
"dt_add": ISODate("2025-04-06T08:00:00Z"),
"dt_upd": ISODate("2025-04-06T08:00:00Z")
}

```

### CustomUser — Врач

```

{
  "_id": ObjectId("6611f3c34f1e8b5f6b1a0002"),
  "first_name": "Сергей",
  "second_name": "Смирнов",
  "third_name": "Алексеевич",
  "email": "dr.smirnov@clinic.ru",
  "password_hash": "$2b$12$def456def456def456def4",
  "avatar": "https://clinic.ru/images/avatars/sergey.jpg",
  "additional_info": {
    "specialization": "Терапевт",
    "experience_years": 8
  },
  "role_id": 3,
  "phone": "+7 901 222 33 44",
  "birth_date": ISODate("1995-08-10T00:00:00Z")
  "dt_add": ISODate("2025-04-06T08:00:00Z"),
  "dt_upd": ISODate("2025-04-06T08:00:00Z")
}

```

### Record — приём пациента у врача

```

{
  "_id": ObjectId("6611f4834f1e8b5f6b1a1001"),
  "dentist_id": ObjectId("6611f3c34f1e8b5f6b1a0002"),
  "patient_id": ObjectId("6611f3c34f1e8b5f6b1a0003"),
  "status": 1,
  "diagnosis": "Кариес второй степени",
  "price": 3500,
  "notes": "Пациент жалуется на боль при жевании. Рекомендуются
пломбировка.",
  "files": [
    "before_xray.png",
    "after_xray.png",
  ],
  "dt_add": ISODate("2025-04-06T09:30:00Z"),
  "dt_upd": ISODate("2025-04-06T09:30:00Z"),
  "dt_rec": ISODate("2025-04-08T14:00:00Z")
}

```

## 3.1.6. Примеры запросов

### 1. Просмотр записей к врачам

Для admin доступны все записи:

```
db.Record.find({});
```



Для patient:

```
db.Record.find({ patient_id:
ObjectId("605c72ef1532071d2c1f78e5") });
```

Для dentist:

```
db.Record.find({ dentist_id:
ObjectId("605c72ef1532071d2c1f78e6") });
```

## 2. Просмотр пациентом своей медицинской карты

```
db.Record.find({ patient_id:
ObjectId("605c72ef1532071d2c1f78e5") });
```

## 3. Информация о конкретном враче

```
db.CustomUser.find({ _id: ObjectId("605c72ef1532071d2c1f78e6")
}, { first_name: 1, second_name: 1, additional_info: 1 });
```

4. Запись на прием (пациент выбирает специалиста, услугу (диагноз), дату, время)

Запись на прием для пациента с ID 3, выбирающего врача с ID 5:

```
db.Record.insertOne({
  dentist_id: ObjectId("605c72ef1532071d2c1f78e6"),
  patient_id: ObjectId("605c72ef1532071d2c1f78e5"),
  status: 1,
  diagnosis: "Кариес",
  price: 5000,
  notes: "Процедура по пломбированию",
  files: ["xray_before.png"],
  dt_rec: ISODate("2025-04-10T10:00:00Z"),
  dt_add: new Date(),
  dt_upd: new Date()
});
```

## 5. Перенос записи

Перенос записи на новую дату:

```
db.Record.updateOne(
  { _id: ObjectId("605c72ef1532071d2c1f78e7") },
  { $set: { dt_rec: ISODate("2025-04-12T10:00:00Z"), dt_upd:
new Date() } }
);
```

## 6. Отмена записи

```
db.Record.updateOne (
```

```

    { _id: ObjectId("605c72ef1532071d2c1f78e7") },
    {
      $set: {
        status: 3,
        dt_upd: new Date()
      }
    }
  );

```

## 7. Просмотр медицинской карты пациента и редактирование в ней записи

Просмотр медицинской карты пациента с ID 3:

```

db.Record.find({ patient_id:
ObjectId("605c72ef1532071d2c1f78e5") });

```

Редактирование записи с ID 2 для врача с ID 5:

```

db.Record.updateOne(
  { _id: ObjectId("605c72ef1532071d2c1f78e7"), dentist_id:
ObjectId("605c72ef1532071d2c1f78e6") },
  { $set: { diagnosis: "Новый диагноз", price: 6000, notes:
"Дополнительные заметки", dt_upd: new Date() } }
);

```

## 8. Просмотр ближайших приемов с пациентами

Для врача с ID 5:

```

db.Record.find({ dentist_id:
ObjectId("605c72ef1532071d2c1f78e6"), dt_rec: { $gte: new Date() }
}).sort({ dt_rec: 1 });

```

## 9. Добавление нового врача (пациента) в базу

```

db.CustomUser.insertOne({
  first_name: "Иван",
  second_name: "Иванов",
  third_name: "Иванович",
  email: "ivanov@clinic.com",
  password_hash: "password_hash",
  role_id: 3, // 3 - для врача
  phone: "+7 900 123 45 67",
  birth_date: "1980-05-15",
  dt_add: new Date(),
  dt_upd: new Date()
});

```

## 10. Массовый импорт базы пациентов (врачей)

```

db.CustomUser.insertMany([
  {

```

```

    first_name: "Мария",
    second_name: "Петрова",
    third_name: "Ивановна",
    email: "petrova@clinic.com",
    password_hash: "password_hash",
    role_id: 2, // 2 - для пациента
    phone: "+7 900 234 56 78",
    birth_date: "1990-02-25",
    dt_add: new Date(),
    dt_upd: new Date()
  },
  {
    first_name: "Алексей",
    second_name: "Смирнов",
    third_name: "Владимирович",
    email: "smirnov@clinic.com",
    password_hash: "password_hash",
    role_id: 3, // 3 - для врача
    phone: "+7 900 345 67 89",
    birth_date: "1975-06-10",
    dt_add: new Date(),
    dt_upd: new Date()
  }
]
);

```

## 11. Массовый экспорт базы пациентов (врачей)

Для экспорта в CSV:

```

const cursor = db.CustomUser.find({}, { first_name: 1,
second_name: 1, third_name: 1, email: 1, phone: 1 });
cursor.forEach(function(doc) {
  printjson(doc); // Результаты можно экспортировать в CSV
или другие форматы
});

```

## 12. Удаление из базы пациента

Удаление пациента с ID 3:

```

db.CustomUser.deleteOne({ _id:
ObjectId("605c72ef1532071d2c1f78e5") });

```

## 13. Просмотр загруженности врача по дням

Для врача с ID 5, загруженность на апрель 2025 года:

```

db.Record.aggregate([
  { $match: { dentist_id:
ObjectId("605c72ef1532071d2c1f78e6"), dt_rec: { $gte:
ISODate("2025-04-01T00:00:00Z"), $lte:
ISODate("2025-04-30T23:59:59Z") } } },

```

```

    { $group: { _id: { $dateToString: { format: "%Y-%m-%d",
date: "$dt_rec" } } }, appointments_count: { $sum: 1 } } },
    { $sort: { "_id": 1 } }
  ]);

```

Для всех врачей в апреле 2025 года:

```

db.Record.aggregate([
  { $match: { dt_rec: { $gte: ISODate("2025-04-01T00:00:00Z"),
$lte: ISODate("2025-04-30T23:59:59Z") } } },
  { $group: { _id: "$dentist_id", appointments_count: { $sum:
1 } } },
  { $sort: { appointments_count: -1 } }
]);

```

### 3.1.7. Количество запросов

Как правило, для каждого сценария нужен один запрос для показа страницы по умолчанию (без фильтров), далее запрос с выставленными фильтрами, добавление/редактирование записи в базе или агрегатный запрос для статистики. Итого в среднем по 2 запроса на каждый сценарий. Количество запросов зависит от конкретного сценария, но обычно их не больше 3.

#### Задействованные таблицы

Чаще всего задействуются обе коллекции – CustomUser и Record. Также есть запросы с использованием только коллекции CustomUser, если администратору нужен просто список всех врачей/пациентов.

### 3.2 Реляционная модель данных

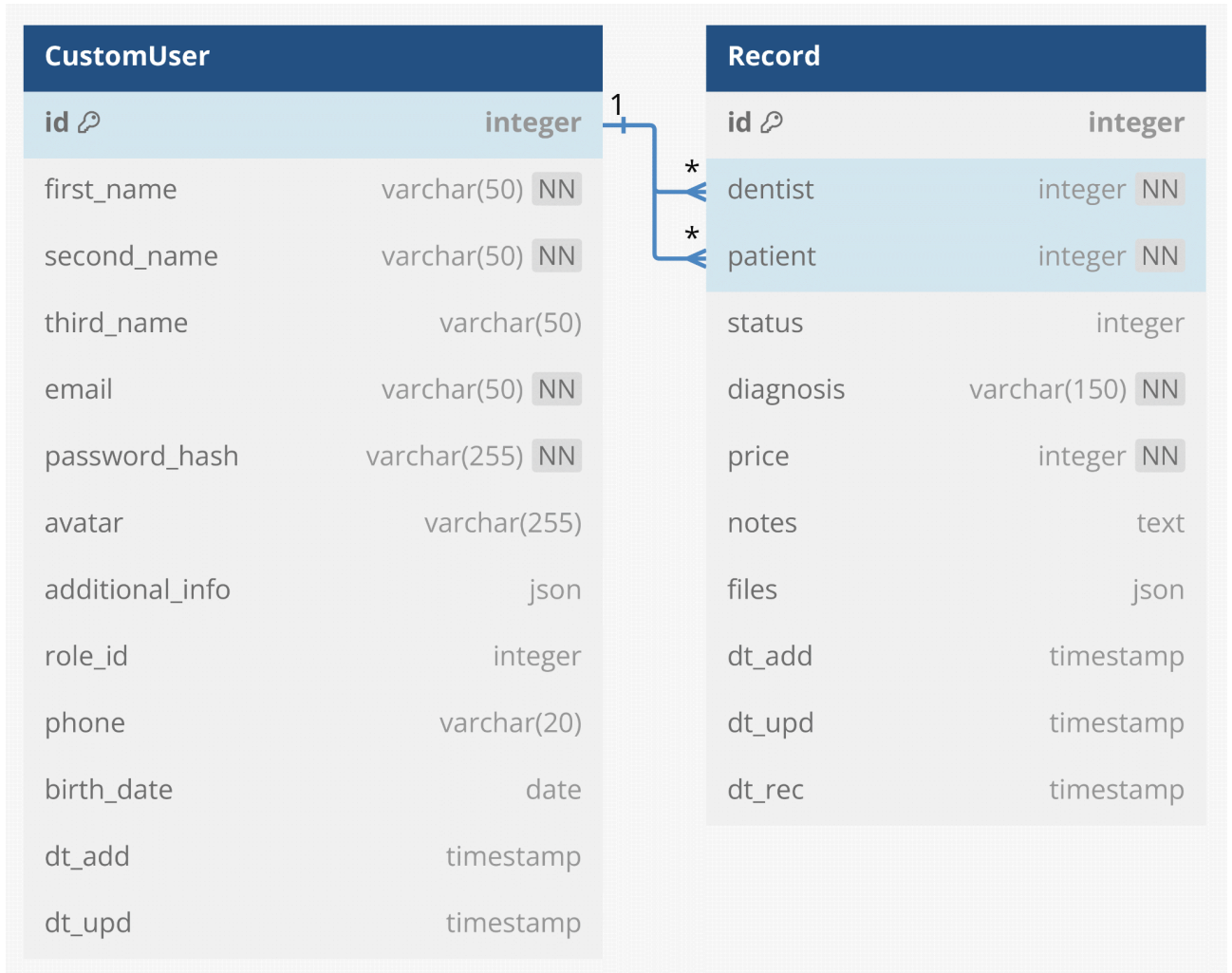


Рисунок 3 — Графическое представление реляционной модели данных

#### 3.2.1. Описание назначений коллекций, типов данных и сущностей

Таблица CustomUser

Поле	Тип данных	Размер (байт)	Описание
id	integer	4	Первичный ключ, автоинкремент
first_name	varchar(50)	50	Имя (обязательное)
second_name	varchar(50)	50	Фамилия (обязательная)
third_name	varchar(50)	50	Отчество
email	varchar(50)	50	Уникальный email (обязательный)
password_hash	varchar(255)	255	Хэш пароля (обязательный)

avatar	varchar(255)	255	Ссылка на аватар
additional_info	json	переменный	Дополнительная информация
role_id	integer	4	Роль пользователя (1, 2, 3)
phone	varchar(11)	11	Телефон (пример: +7 900 000 00 00)
birth_date	date	3	Дата рождения
dt_add	timestamp	8	Дата добавления (по умолчанию: now())
dt_upd	timestamp	8	Дата обновления (по умолчанию: now())

Таблица Record

Поле	Тип данных	Размер (байт)	Описание
id	integer	4	Первичный ключ, автоинкремент
dentist	integer	4	Врач-стоматолог (ссылка на CustomUser.id, обязателен)
patient	integer	4	Пациент (ссылка на CustomUser.id, обязателен)
status	integer	4	Статус записи
appointment_date	timestamp	8	Назначенное время (обязательное)
diagnosis	varchar(150)	150	Диагноз
notes	text	переменный	Примечания
files	json	переменный	Прикреплённые файлы
dt_add	timestamp	8	Дата создания записи
dt_upd	timestamp	8	Дата обновления записи
dt_rec	timestamp	8	Фактическая дата приема

CustomUser: Хранит информацию о пользователях

Record: Хранит информацию о записях

### 3.2.2. Оценка объема информации, хранимой в модели

- $N$  — количество пользователей
- $M$  — количество записей
- $U$  — средний размер документа CustomUser 948 байта
- $R$  — средний размер документа Record 598 байт
- Пусть доля врачей среди всех пользователей — 5%, пациентов — 94%, админов — 1%. Допустим средний врач работает 20 дней в месяц и принимает в день 6 пациентов. Записи хранятся 5 лет, затем архивируются. Тогда зависимость количества записей от количества пользователей  $M(N) = N * 0.05 * 6 * 20 * 12 * 5 = 360 * N$ . Общий объем базы данных:  $V(N) = N * U + M(N) * R = 216\,228 * N$  байт

### 3.2.3. Избыточность данных

В реляционной модели таблица CustomUser имеет средний размер строки 948 байт. При исключении избыточных элементов, таких как поле `id` (4 байта), поле `role_id` (4 байта) и поле `additional_info` (в среднем 200 байт), остаётся 740 байт значимой информации, относящейся к конкретному пользователю.

Таблица Record имеет средний размер строки 598 байт. Из неё исключаются поля `id` (4 байта), `dentist` (4 байта), `patient` (4 байта), `status` (4 байта) и `diagnosis` (150 байт), поскольку они либо являются дублирующими ссылками на другие таблицы, либо содержат повторяющиеся значения, заменяемые внешними ключами. После этого остаётся 432 байта полезной нагрузки на одну запись.

Если учитывать, что на одного пользователя приходится в среднем 360 записей (на основании расчёта по средней загруженности врача в течение 5 лет), то объём базы данных без избыточных данных составит:

$$V^*(N) = 740 \times N + 432 \times 360 \times N = 156220 \times N \text{ байт}$$

где  $N$  — общее количество пользователей.

Полный же объём данных в модели до оптимизации составляет:

$$V(N) = 216228 \times N \text{ байт}$$

Таким образом, коэффициент избыточности можно рассчитать как:

$$216228 / 156220 \approx 1.38$$

Это означает, что примерно 38% от общего объёма данных в исходной SQL-модели составляют избыточную или структурную информацию, которую можно исключить без потери значимых сведений.

#### **3.2.4. Направление роста модели при увеличении количества объектов каждой сущности**

Когда возникает необходимость масштабировать модель, одним из главных аспектов является понимание того, как именно растёт система при увеличении количества объектов каждой сущности. В рассматриваемом случае расширение модели происходит за счёт линейного увеличения количества записей в соответствующих таблицах по мере роста числа пользователей или записей. Это означает, что структура базы данных так организована, что она сохраняет стабильность работы и своих характеристик при увеличении нагрузки, добавляя новые записи в пропорциональном масштабе.

Такой линейный характер роста обеспечивает предсказуемое поведение модели, позволяя системным администраторам и разработчикам заранее планировать необходимые ресурсы и меры для поддержания производительности. Это особенно важно в условиях быстро увеличивающихся объёмов данных, где ясное понимание динамики роста базы данных позволяет своевременно принимать меры для поддержания её эффективности и надёжности. Это способствует устойчивости системы и её способности справляться с возрастающими запросами и увеличивающейся нагрузкой без потерь в производительности и стабильности.

#### **3.2.5. Примеры данных**

Таблица CustomUser

id	first_name	second_name	third_name	email	password_hash	avatar	additional_info	role_id	phone	birth_date	dt_added	dt_updated
----	------------	-------------	------------	-------	---------------	--------	-----------------	---------	-------	------------	----------	------------



1	Анна	Козлова	Сергеевна	admin@clinic.ru	\$2b\$12\$examplehash1	null	{"position": "Главный админ"}	1	+79111111	1980-04-12 00:00:00	2025-04-01 10:00:00	2025-04-01 10:00:00
2	Иван	Смирнов	Алексеевич	ivan.smirnov@mail.ru	\$2b\$12\$examplehash2	/images/patient1.jpg	{"allergies": ["анестезия"]}	2	+7912222222	1992-09-03 00:00:00	2025-04-01 11:15:00	2025-04-01 11:15:00
3	Мария	Белова	Викторовна	belova.dentist@mail.ru	\$2b\$12\$examplehash3	/images/dentist1.jpg	{"education": "МГМСУ", "experience": "10 лет"}	3	+7913333333	1985-01-27 00:00:00	2025-04-01 11:30:00	2025-04-01 11:30:00

Таблица Record

id	dentist	patient	status	diagnosis	price	notes	dt_add	dt_upd	dt_rec	files
1	3	2	1	Кариес зуба 2.7	5000	Пациент жалуется на боль	2025-04-03 09:00:00	2025-04-03 09:00:00	2025-04-05 14:30:00	["photo1.jpg", "xray_27.png"]

### 3.2.6. Примеры запросов

#### 1. Просмотр записей к врачам

Для admin доступны все записи:

```
SELECT r.id, r.dentist, r.patient, r.status, r.diagnosis,
r.dt_rec, r.dt_add, r.dt_upd
FROM Record r;
```

Для patient:

```
SELECT r.id, r.dentist, r.patient, r.status, r.diagnosis,
r.dt_rec, r.dt_add, r.dt_upd
FROM Record r
WHERE r.patient = 3;
```

Для dentist:

```
SELECT r.id, r.dentist, r.patient, r.status, r.diagnosis,
r.dt_rec, r.dt_add, r.dt_upd
FROM Record r
WHERE r.dentist = 5;
```

## 2. Просмотр пациентом своей медицинской карты

```
SELECT r.id, r.dentist, r.patient, r.status, r.diagnosis,  
r.dt_rec, r.dt_add, r.dt_upd  
FROM Record r  
WHERE r.patient = 3;
```

## 3. Информация о конкретном враче

```
SELECT u.first_name, u.second_name, u.third_name, u.email,  
u.phone, u.additional_info  
FROM CustomUser u  
WHERE u.id = 5;
```

## 4. Запись на прием (пациент выбирает специалиста, услугу (диагноз), дату, время)

Запись на прием для пациента с ID 3, выбирающего врача с ID 5:

```
INSERT INTO Record (dentist, patient, status, diagnosis,  
price, notes, dt_rec, dt_add, dt_upd)  
VALUES (5, 3, 1, 'Кариес', 5000, 'Процедура по пломбированию',  
'2025-04-10 10:00:00', NOW(), NOW());
```

## 5. Перенос записи

Перенос записи на новую дату:

```
UPDATE Record  
SET dt_rec = '2025-04-12 10:00:00', dt_upd = NOW()  
WHERE id = 2;
```

## 6. Отмена записи

```
UPDATE Record  
SET status = 3, dt_upd = NOW() -- статус 3 - "canceled"  
WHERE id = 2;
```

## 7. Просмотр медицинской карты пациента и редактирование в ней записи

Просмотр медицинской карты пациента с ID 3:

```
SELECT r.id, r.dentist, r.patient, r.status, r.diagnosis,  
r.dt_rec, r.dt_add, r.dt_upd  
FROM Record r  
WHERE r.patient = 3;
```

Редактирование записи с ID 2 для врача с ID 5:

```
UPDATE Record
```

```
SET diagnosis = 'Новый диагноз', price = 6000, notes =  
'Дополнительные заметки', dt_upd = NOW()  
WHERE id = 2 AND dentist = 5;
```

## 8. Просмотр ближайших приемов с пациентами

Для врача с ID 5:

```
SELECT r.id, r.dentist, r.patient, r.status, r.diagnosis,  
r.dt_rec  
FROM Record r  
WHERE r.dentist = 5 AND r.dt_rec >= NOW()  
ORDER BY r.dt_rec ASC;
```

## 9. Добавление нового врача (пациента) в базу

```
INSERT INTO CustomUser (first_name, second_name, third_name,  
email, password_hash, role_id, phone, birth_date, dt_add, dt_upd)  
VALUES ('Иван', 'Иванов', 'Иванович', 'ivanov@clinic.com',  
'password_hash', 3, '+7 900 123 45 67', '1980-05-15', NOW(),  
NOW());
```

## 10. Массовый импорт базы пациентов (врачей)

```
LOAD DATA INFILE 'patients_and_doctors.csv' INTO TABLE  
CustomUser  
FIELDS TERMINATED BY ','  
ENCLOSED BY '"'  
LINES TERMINATED BY '\n'  
(first_name, second_name, third_name, email, password_hash,  
role_id, phone, birth_date, dt_add, dt_upd);
```

## 11. Массовый экспорт базы пациентов (врачей)

Для экспорта в CSV:

```
SELECT first_name, second_name, third_name, email, role_id,  
phone, birth_date  
FROM CustomUser  
INTO OUTFILE '/path/to/exported_patients.csv'  
FIELDS TERMINATED BY ','  
ENCLOSED BY '"'  
LINES TERMINATED BY '\n';
```

## 12. Удаление из базы пациента

Удаление пациента с ID 3:

```
DELETE FROM CustomUser WHERE id = 3; -- ID пациента или врача
```

## 13. Просмотр загруженности врача по дням

Для врача с ID 5, загруженность на апрель 2025 года:

```
SELECT DATE(dt_rec) AS date, COUNT(*) AS appointments_count
FROM Record
WHERE dentist = 5 AND dt_rec BETWEEN '2025-04-01' AND
'2025-04-30'
GROUP BY DATE(dt_rec)
ORDER BY date ASC;
```

Для всех врачей в апреле 2025 года:

```
SELECT dentist, COUNT(*) AS appointments_count
FROM Record
WHERE dt_rec BETWEEN '2025-04-01' AND '2025-04-30'
GROUP BY dentist
ORDER BY appointments_count DESC;
```

### 3.2.7. Количество запросов

Как правило, для каждого сценария нужен один запрос для показа страницы по умолчанию (без фильтров), далее запрос с выставленными фильтрами и/или добавление/редактирование записи в базе данных. Итого в среднем по 2 запроса на каждый сценарий с использованием JOIN.

#### Задействованные таблицы

Чаще всего используются обе таблицы — CustomUser и Record. Они связаны с помощью JOIN, поскольку, например, врач может получить список своих пациентов только при условии просмотра ID пациентов в таблице Record в записях, соответствующих конкретному врачу.

Также есть запросы, в которых используется только таблица CustomUser. Например, администратору может потребоваться список всех врачей или пациентов.

С мед. картой ситуация аналогичная: чтобы её получить, нужно использовать JOIN, поскольку в таблице Record хранятся только ID пациентов и врачей без ФИО.

### **3.3. Сравнение моделей**

#### **3.3.1. Удельный объём информации**

NoSQL: ~523 байт на пользователя, ~568 байт на запись. Общий объем:  $205\,003 \times N$  байт

SQL: ~948 байт на пользователя, ~598 байт на запись. Общий объем:  $216\,228 \times N$  байт

NoSQL-модель требует меньше места при масштабировании.

#### **3.3.2. Запросы и их сложность**

NoSQL: Простые запросы к одной коллекции выполняются быстро. Для сложных сценариев может понадобиться несколько запросов и использование обеих коллекций, что сложнее.

SQL: Гибкие запросы, легко использовать JOIN'ы, агрегацию и фильтрацию. Лучше для сложных аналитических сценариев.

SQL мощнее при аналитике, NoSQL проще для CRUD.

#### **3.3.3. Итоговое сравнение по объёму и запросам**

С точки зрения удельного объёма информации, NoSQL-модель показывает лучшую эффективность: она занимает ~523 байта на одного пользователя и ~568 байт на одну запись, тогда как SQL — ~948 и ~598 байт соответственно. При масштабировании, когда количество пользователей и записей быстро растёт, это даёт значительное преимущество по экономии дискового пространства. Общий объём данных в NoSQL составляет  $205\,003 \times N$  байт, в то время как SQL —  $216\,228 \times N$  байт. Таким образом, NoSQL требует меньше памяти при сохранении той же бизнес-логики и набора данных.

Что касается запросов, NoSQL отлично справляется с базовыми операциями чтения и записи (CRUD), особенно при работе внутри одной коллекции. Однако при необходимости объединения данных из разных сущностей (например, врач + пациент + запись) могут потребоваться дополнительные усилия и логика на уровне приложения. В противоположность этому, SQL предоставляет мощные инструменты для работы со сложными структурами данных — благодаря JOIN'ам, агрегациям и продвинутой

фильтрации. Это делает его более удобным и наглядным решением для аналитических и отчётных задач.

В итоге, NoSQL выигрывает в сценариях с частыми изменениями структуры данных и большим объёмом операций записи, в то время как SQL предпочтительнее для аналитики и сложных выборок.

#### **3.3.4. Вывод**

Для системы управления стоматологической клиникой использование NoSQL-модели представляется более целесообразным и оправданным по следующим причинам.

Во-первых, структура данных клиники подвержена частым изменениям и дополнениям: например, в карточке пациента могут появляться новые поля, связанные с индивидуальными особенностями лечения, прикреплённые файлы, фотографии, история взаимодействий и т.д. NoSQL предоставляет гибкость в работе с такими данными — коллекции легко расширяются без необходимости изменения схемы, как это было бы в реляционной модели.

Во-вторых, предполагается активная запись данных: каждый врач создаёт десятки записей ежедневно, и система должна быстро обрабатывать их. В NoSQL такие CRUD-операции выполняются быстрее, особенно при работе в рамках одной коллекции. Кроме того, линейная масштабируемость и простота горизонтального расширения делают NoSQL выгодным выбором при росте числа пользователей и объёма записей.

В-третьих, анализ показал, что при равном количестве пользователей общий объём данных в NoSQL после устранения избыточности оказывается меньше, чем в SQL ( $154387 \times N$  против  $156220 \times N$  байт), что дополнительно подтверждает её эффективность в данном случае.

SQL-модель, безусловно, была бы более удобной для систем, где требуется сложная аналитика, кросс-табличные связи и формализованные схемы. Однако в рамках стоматологической системы приоритетом являются гибкость, производительность и минимальная избыточность — и по этим критериям NoSQL выигрывает.

Таким образом, NoSQL-модель является более подходящей архитектурной основой для текущей задачи — создания гибкой, масштабируемой и производительной информационной системы для стоматологической клиники.

## 4. Разработанное приложение

### Краткое описание

Проект представляет собой веб-приложение для стоматологической клиники. Backend реализован на Python с использованием фреймворка Django. Frontend написан на HTML, JavaScript и использует Vite для сборки. В качестве базы данных используется MongoDB. Приложение развёртывается с помощью Docker и docker-compose. Архитектура проекта включает модули: users, dentists, clients, records и core.

### Использованные технологии

Django, MongoDB, Docker, Vite, Git.

### Снимки всех экранов приложения

Стоматология

ВХОД      РЕГИСТРАЦИЯ

Имя  
введите имя

Фамилия  
введите фамилию

Имя пользователя  
введите имя пользователя

E-mail  
укажите e-mail

Пароль  
введите пароль

Подтвердите пароль  
подтвердите пароль

РЕГИСТРАЦИЯ

Рисунок 4 — Страница регистрации

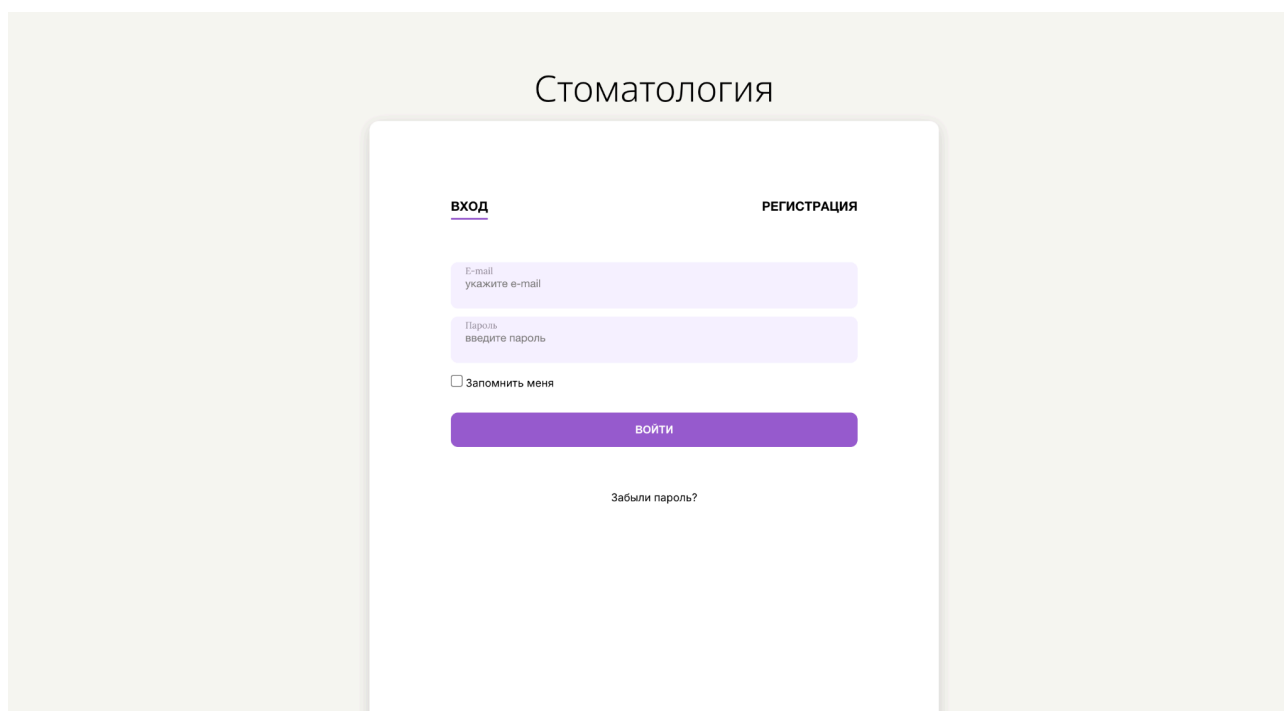


Рисунок 5 — Страница авторизации

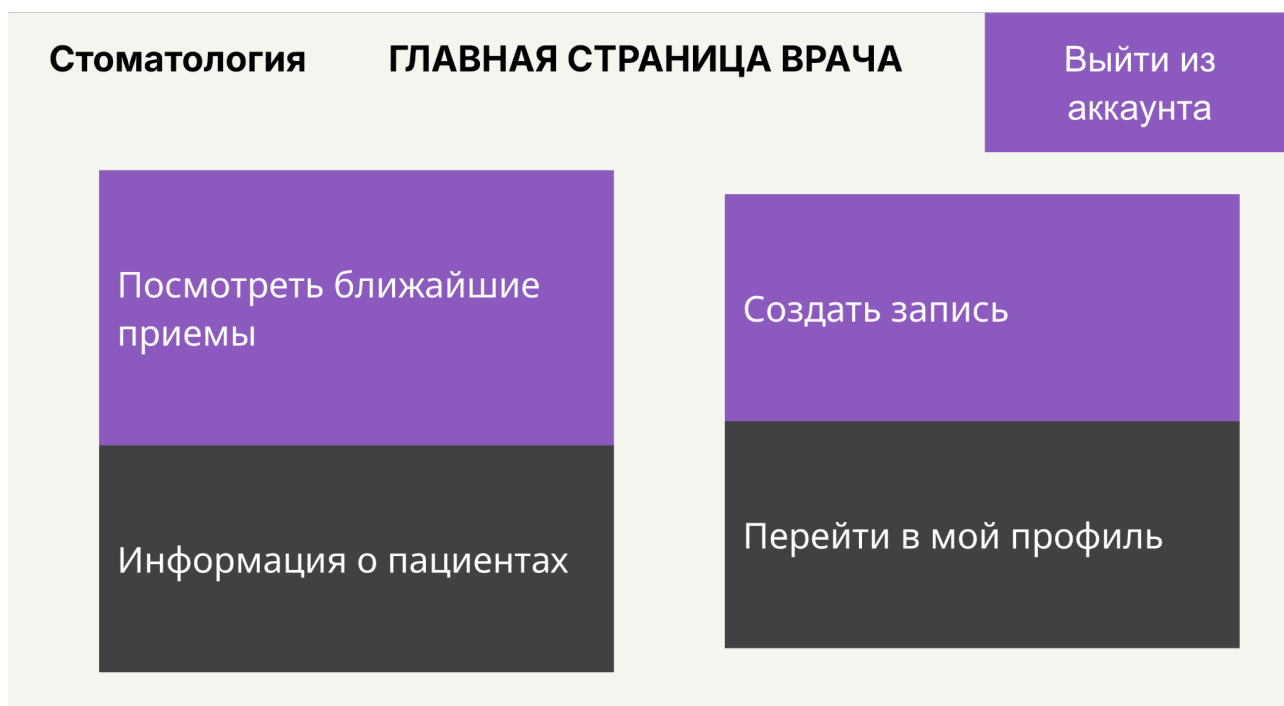


Рисунок 6 — Главная страница врача



Назад

Стоматология

Записи

Пациенты

Специалисты

Мой профиль


Главная

## Записи

Записать пациента

Диапазон с  до

☐ Прошедшие записи



Газукина Дарья

03.06.2025

Рисунок 7 — Страница записей к врачу

Назад

Стоматология

Записи

Пациенты

Специалисты

Мой профиль

Главная

## Создание записи

Услуга

Клиент

Дата и время

Дополнительная информация

Записаться

Рисунок 8 — Создание записи на прием

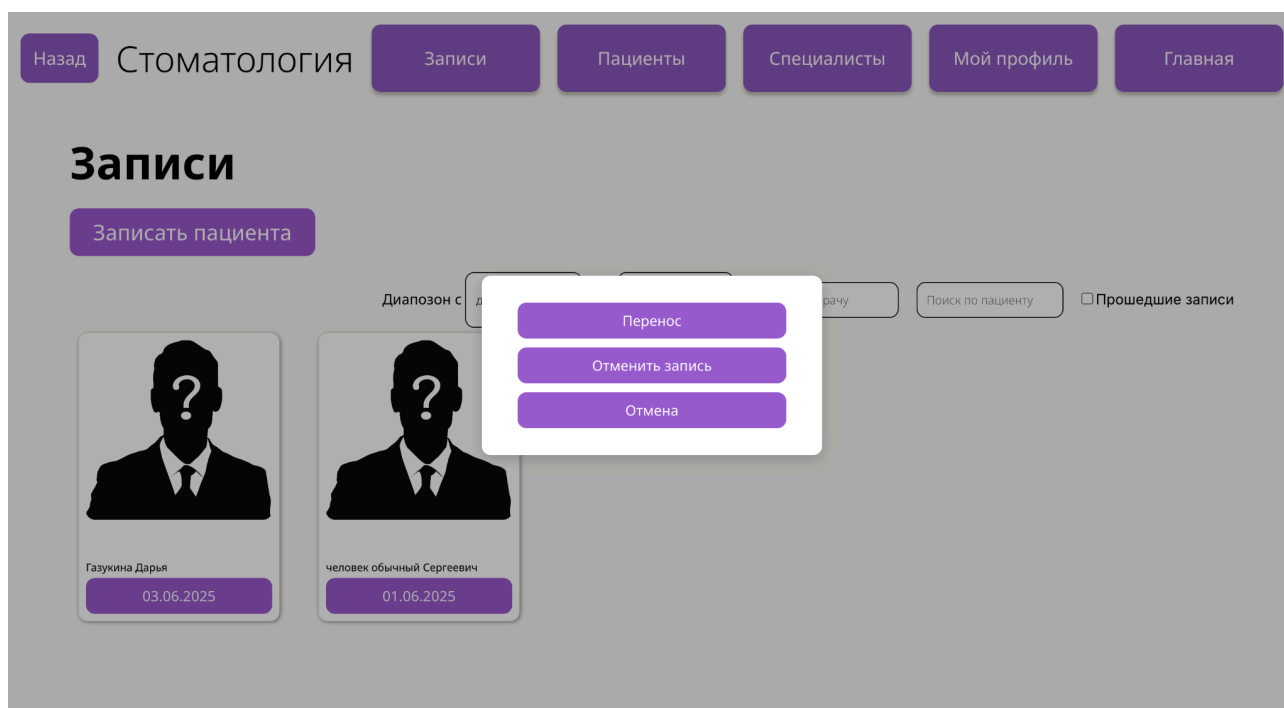


Рисунок 9 — Отмена и перенос записи

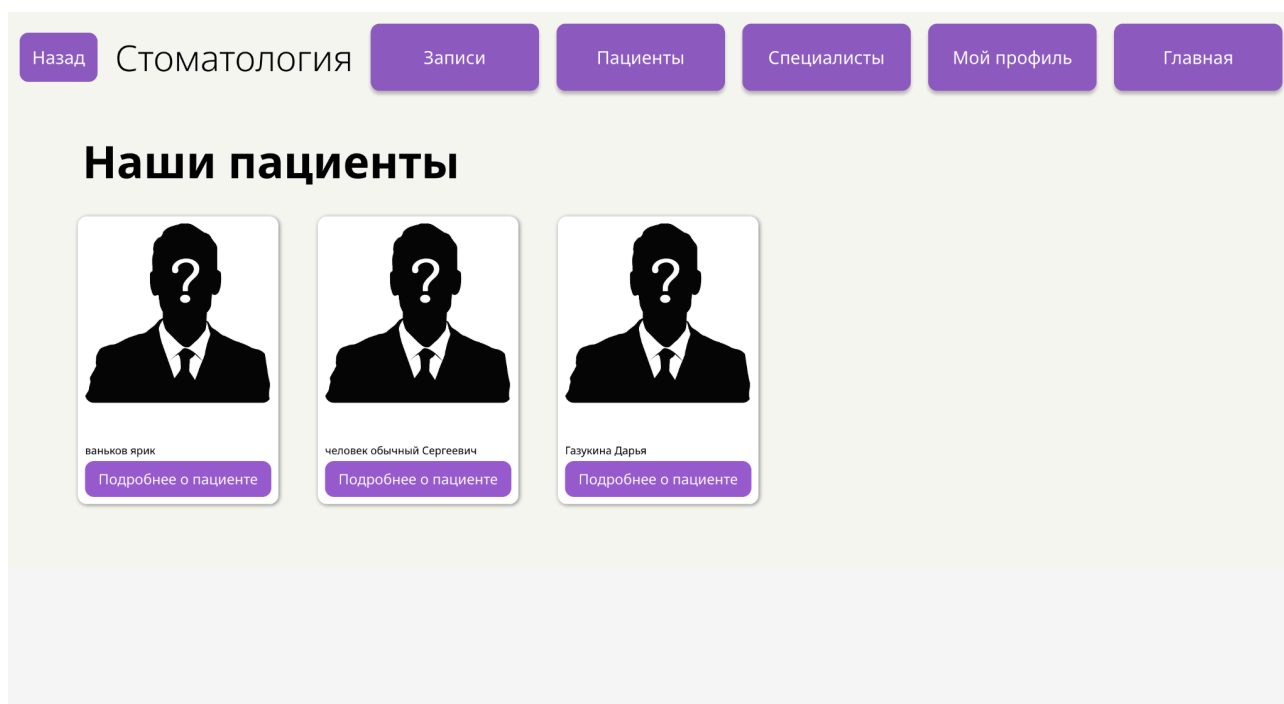


Рисунок 10 — Список пациентов клиники

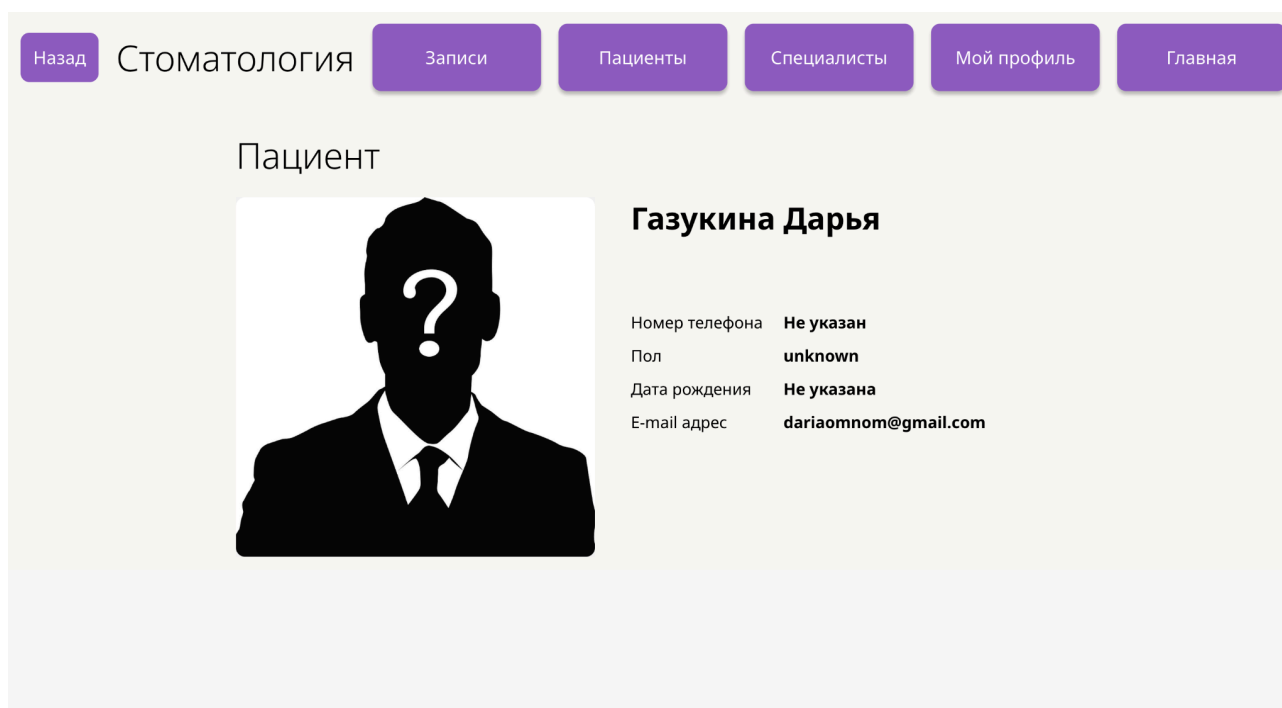


Рисунок 11 — Страница одного пациента

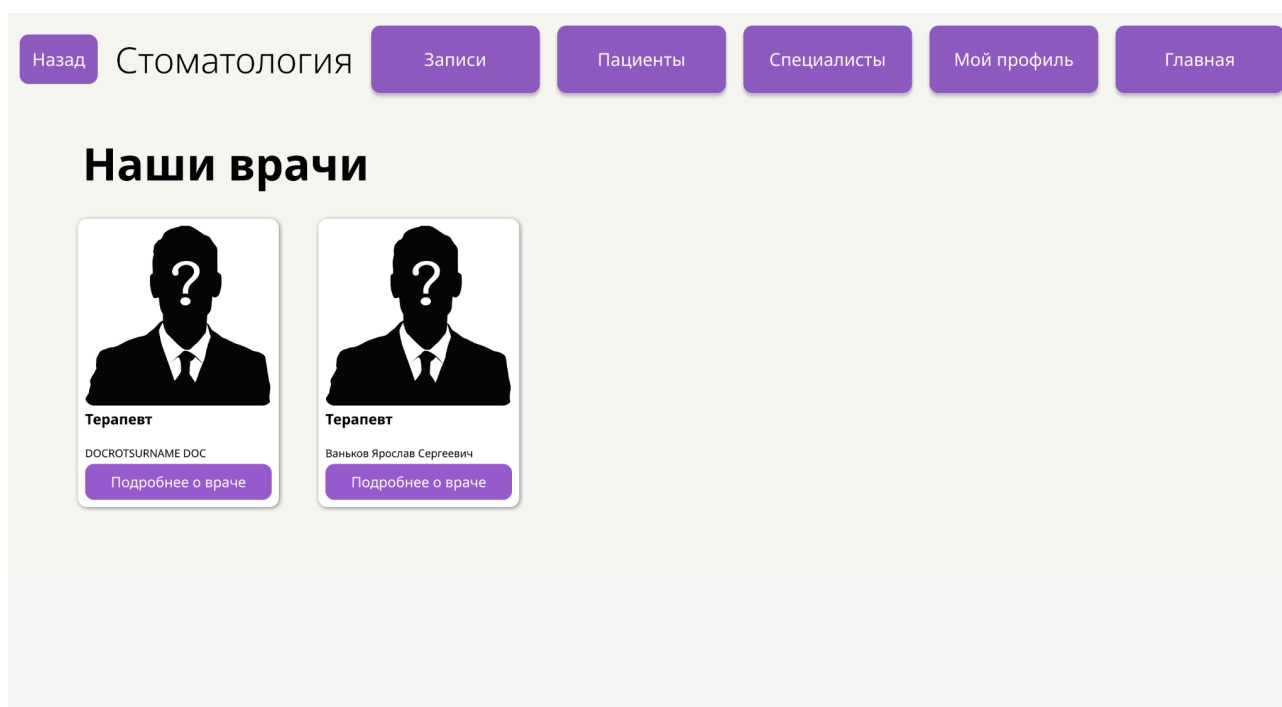


Рисунок 12 — Список специалистов клиники

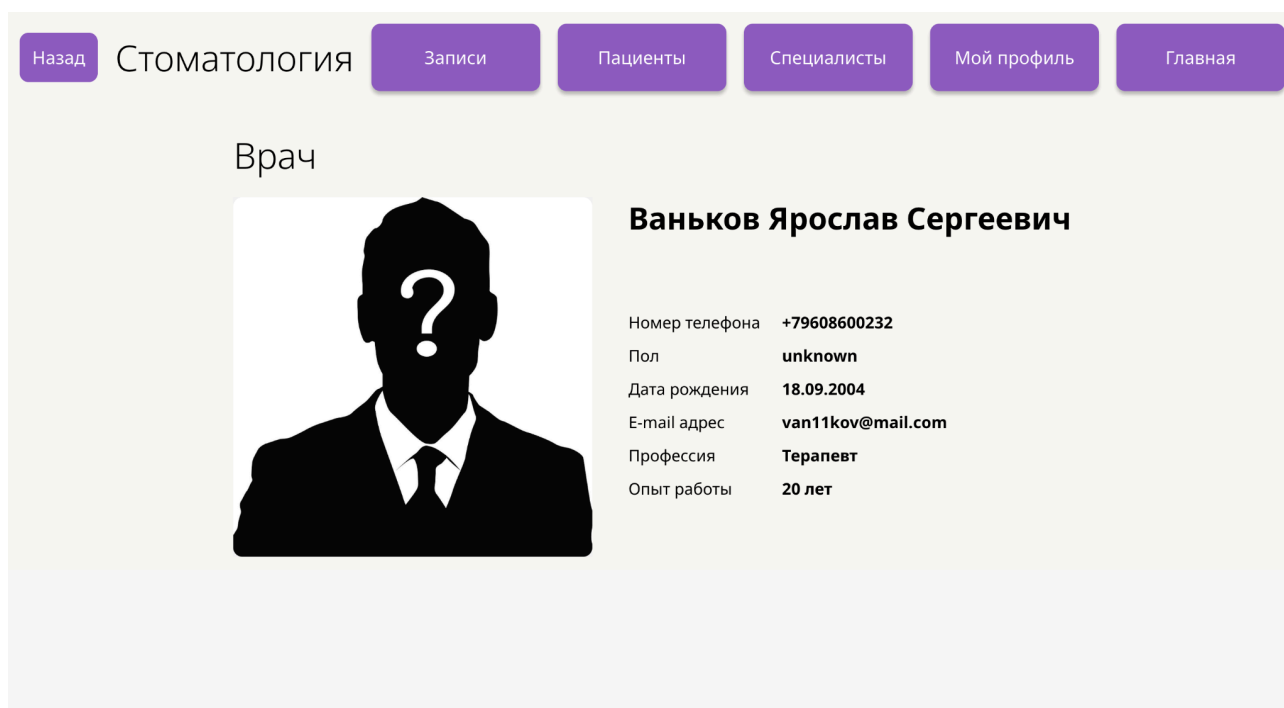


Рисунок 13 — Страница одного специалиста

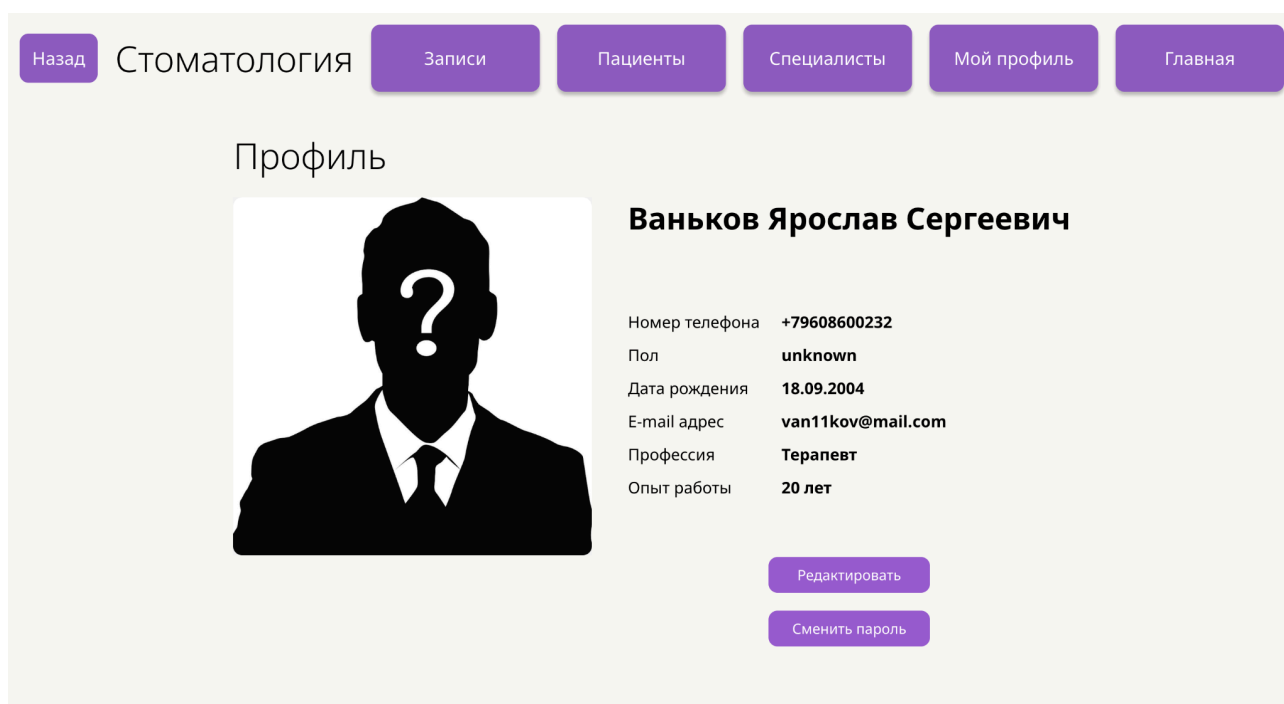


Рисунок 14 — Профиль врача

Назад

Стоматология

Записи

Пациенты

Специалисты

Мой профиль

Главная

Профиль



Редактировать профиль

Фамилия

Ваньков

Имя

Ярослав

Отчество

Сергеевич

Номер телефона

+79608600232

Пол

Не указан

Дата рождения

18.09.2004

Е-mail адрес

van11kov@mail.com

Профессия

Терапевт

Опыт работы

20 лет

Сохранить

Отмена

Рисунок 15 — Редактирование личного профиля

Стоматология

ГЛАВНАЯ СТРАНИЦА ПАЦИЕНТА

Выйти из аккаунта

Посмотреть мои записи

Посмотреть врачей

Записаться на прием

Открыть медицинскую карту

Перейти в мой профиль

Рисунок 16 — Главная страница пациента

## **5. Выводы**

### **Достигнутые результаты**

В рамках проекта было разработано веб-приложение для стоматологической клиники, предназначенное для упрощения взаимодействия пациентов и врачей.

На текущем этапе реализованы следующие функции:

1. Регистрация и авторизация пользователей;
2. Просмотр списков врачей и пациентов с возможностью фильтрации;
3. Запись на приём к врачу;
4. Возможность редактирования профиля пользователя;
5. Разграничение прав доступа (пациент, врач, администратор);
6. Реализация backend-части на Django и frontend на Vite с использованием HTML/JavaScript;
7. Хранение данных в MongoDB и контейнеризация с помощью Docker.

### **Недостатки и пути для улучшения полученного решения**

1. Отсутствует адаптивная верстка: интерфейс не до конца оптимизирован для мобильных устройств;
2. Недостаточная валидация данных на фронтенде;
3. Отсутствует система уведомлений (например, напоминание о приёме);
4. Ограниченные возможности по аналитике и статистике.

Для улучшения проекта следует внедрить адаптивный дизайн, расширить валидацию данных и добавить функциональность, ориентированную на администраторов и улучшение пользовательского опыта.

### **Будущее развитие решения**

- Онлайн-оплата приёмов через банковские карты, QR-коды или платёжные сервисы
- Добавление напоминаний и уведомлений (по SMS, email, push) о предстоящих приёмах или необходимости профилактического осмотра
- Улучшение пользовательского интерфейса
- Разработка адаптивного дизайна для мобильных устройств
- Улучшение UX/UI (быстрая запись, удобные фильтры, интуитивная

- навигация)
- Поддержка нескольких языков интерфейса (например, русский, английский)
  - Развитие аналитики и автоматизации – частота визитов, загрузка врачей, популярные услуги
  - Система рекомендаций услуг на основе диагноза или истории приёмов.

## **6. Приложения**

Для запуска проекта используется:

```
docker compose build --no-cache && docker compose up
```

Сервис будет доступен по адресам:

➔ Local: `http://localhost:3000/`

➔ Network: `http://172.19.0.4:3000/`

## 7. Литература

1. Ссылка на репозиторий: [github.com/moevm/nosql1h25-dentistry](https://github.com/moevm/nosql1h25-dentistry)
2. Документация MongoDB: [docs.mongodb.com/manual/](https://docs.mongodb.com/manual/)
3. Документация Django: [docs.djangoproject.com/en/5.2/](https://docs.djangoproject.com/en/5.2/)