



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

*«Разработка веб-сайта для мониторинга ситуации с
больными коронавирусной инфекцией COVID-19»*

Студент ИУ7-63Б
(Группа)

(Подпись, дата)

А. А. Наместник
(И.О.Фамилия)

Руководитель курсового проекта

(Подпись, дата)

Ю.М. Гаврилова
(И.О.Фамилия)

Москва, 2021 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой _____
(Индекс)

(И.О.Фамилия)
« ____ » _____ 20 ____ г.

ЗАДАНИЕ на выполнение курсового проекта

по дисциплине _____ «Базы данных» _____

Студент группы _____ ИУ7-63Б _____

Наместник Анастасия Андреевна
(Фамилия, имя, отчество)

Тема курсового проекта Разработка веб-сайта для мониторинга ситуации с больными
коронавирусной инфекцией COVID-19

Направленность КП (учебный, исследовательский, практический, производственный, др.)

учебный
Источник тематики (кафедра, предприятие, НИР) _____ кафедра _____

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание: Разработать веб-сайт для анализа базы данных, содержащей информацию о больных
коронавирусной инфекцией. Веб-сайт должен предоставлять возможность авторизации как с
правами администратора, так и с правами пользователя. Администратор имеет доступ к базе данных
с правом вносить туда изменения. Пользователю позволено формировать запросы к базе данных.
Также должен быть предусмотрен вывод статистической информации.

Оформление курсового проекта:

Расчетно-пояснительная записка на 25-30 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку задачи, введение, аналитическую часть,
конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение,
список литературы и приложения.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

На защиту проекта должна быть предоставлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть
отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура
комплекса программ, интерфейс, результаты проведенных исследований.

Дата выдачи задания « ____ » _____ 20 ____ г.

Руководитель курсового проекта

Студент

(Подпись, дата)

(Подпись, дата)

Ю. М. Гаврилова

(И.О.Фамилия)

А. А. Наместник

(И.О.Фамилия)

Оглавление

Введение	4
1. Аналитическая часть.....	5
1.1 Постановка задачи.....	5
1.2 Формализация данных.....	5
1.3 Виды пользователей	6
1.4 Определение понятия СУБД.....	8
1.4.1 Основные функции СУБД	8
1.4.2 Классификация СУБД по модели данных.....	9
1.5 Выводы из аналитического раздела.....	12
2. Конструкторская часть.....	13
2.1 Проектирование базы данных	13
2.2 Требования к программе	19
2.3 Проектирование приложения	20
2.4 Выводы из конструкторского раздела	28
3. Технологическая часть	29
3.1 Выбор и обоснование используемых технологий	29
3.2 Структура и состав классов	30
3.3 Взаимодействие с базой данных	34
3.4 Триггер.....	35
3.5 Сведения о модулях программы	36
3.6 Графический интерфейс пользователя	36
3.6.1 Авторизация	38
3.6.2 Регистрация	39
3.6.3 Личный кабинет	42
3.6.4 База данных	44
3.6.5 Запросы	45
3.6.6 Создание запроса	47
3.7 Выводы из технологического раздела	49
Заключение.....	51
Список использованной литературы	52

Введение

В современном мире одной из основных проблем всего человечества являются вирусы и осложнения, которые они вызывают. Вирусы являются неотъемлемой частью почти каждой экосистемы на Земле, их многообразие и диапазон биохимического воздействия на инфицированную клетку живого организма очень велики, но в большинстве случаев, вирус приводит к гибели клеток-хозяев. Из-за быстрой приспособляемости и жизнеспособности вирусы становятся большой угрозой для здоровья и долголетия человека, и эта проблема особенно важна сейчас, когда к уже существующим активным вирусам в 2019 году добавился новый коронавирус, вызвавший вспышку заболевания с различными последствиями по всему миру. COVID-19 — заболевание, вызываемое новым коронавирусом, который называется SARS-CoV-2 [13]. ВОЗ впервые узнала об этом новом вирусе 31 декабря 2019 г., получив сообщение о группе случаев заболевания «вирусной пневмонией» в городе Ухане, Китайская Народная Республика.

Цель данной работы – разработать базу данных, содержащую информацию о больных коронавирусной инфекцией и пользователях веб-сайта, и веб-приложение для ее анализа.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи.

1. Проанализировать предметную область, сформулировать ограничения предметной области.
2. Провести анализ существующих СУБД.
3. Спроектировать базу данных, содержащую информацию о больных коронавирусной инфекцией и пользователях веб-сайта.
4. Реализовать веб-приложение для анализа спроектированной базы данных.

1. Аналитическая часть

В данном разделе будет проанализирована поставленная задача, формализованы данные, необходимые для ее решения, а также приведены теоретические сведения, на основании которых будет сделан выбор способа представления данных и управления ими.

1.1 Постановка задачи

В рамках поставленной цели необходимо разработать базу данных, содержащую информацию о больных коронавирусной инфекцией и пользователях веб-сайта, и веб-приложение для анализа этой базы данных. Веб-сайт должен предоставлять возможность авторизации как с правами обычного пользователя, так и с правами, предоставляющими больший доступ к базе данных. Пользователь может обращаться к базе данных посредством выбора запроса из предложенных, а также фиксировать свое текущее состояние здоровья в личном кабинете и наблюдать за статистикой больных COVID-19 пользователей веб-приложения, проживающих на его улице. Администратор (специалист) имеет те же возможности, и дополнительно ему предоставляется право создавать запросы к базе данных и вносить в нее изменения.

1.2 Формализация данных

База данных должна содержать информацию о следующих объектах.

1. Пользователи сайта:

1.1. Данные для аутентификации.

2.2. Персональные данные для личного кабинета.

2. Улицы Москвы.

3. Запросы к базе данных, имеющиеся в системе и созданные специалистами.

4. Пациенты с зарегистрированными случаями заражения COVID-19.

5. Местоположение пациентов.

6. Симптомы, наблюдающиеся при COVID-19.

7. Статус состояния здоровья пациентов.

Таблица 1.1 – категории и сведения о данных

Категория	Сведения
Пользователь	Email-адрес, пароль
Личный кабинет	Владелец кабинета, имя, фамилия, дата рождения, пол, адрес проживания (название улицы), состояние здоровья
Роль	Название, описание
Улица	Название, численность населения, количество больных COVID-19 на этой улице
Запросы	Название, описание, SQL-запрос, названия выходных столбцов
Пациенты	Дата регистрации в базе данных, пол, возраст, дата начала симптомов, дата посещения больницы, дата начала наблюдения, дата окончания наблюдения, посещал ли пациент Ухань, является ли пациент жителем Уханя
Местоположение	Провинция/штат, страна, широта, долгота
Симптомы	Название
Статус состояния здоровья	Название

1.3 Виды пользователей

Для создания личного кабинета, который позволит следить за текущей ситуацией на конкретной улице и здоровьем самого пользователя, и для просмотра запросов к базе данных нужна авторизация пользователей. Таким образом, существуют авторизованные и неавторизованные пользователи.

Для внесения изменений в базу данных и составления релевантных SQL-запросов введена роль администратора, или специалиста. Предполагается, что специалист владеет информацией о течении заболевания в мире и знает основы

языка SQL [8], так чтобы его корректировки и созданные им запросы отражали реальную картину происходящего.

Просмотр таблиц базы данных доступен для всех видов пользователей, однако только специалист может видеть развязочную таблицу, названия таблиц и схем в базе данных и ER-диаграмму сущностей базы данных, так как эта информация необходима для составления запросов и внесения изменений. Для обычного пользователя такого рода информация не несет смысловой нагрузки.

В таблице 1.2 и на рисунке 1.1 представлены виды пользователей и их возможности, а также описание системы на концептуальном уровне в виде диаграммы прецедентов, соответственно.

Таблица 1.2 – Виды пользователей и доступные им возможности

Вид пользователя	Возможности
Неавторизованный	Регистрация, авторизация Просмотр таблиц базы данных (всех, кроме развязочной)
Авторизованный (обычный пользователь)	Создание личного кабинета, фиксация статуса состояния здоровья (Здоров/Болен COVID-19) Просмотр статистики заболевших на своей улице проживания Просмотр запросов (выбор любого из списка для получения результата) Просмотр таблиц базы данных (всех, кроме развязочной)
Авторизованный (специалист)	Создание личного кабинета, фиксация статуса состояния здоровья (Здоров/Болен COVID-19) Просмотр статистики заболевших на своей улице проживания Просмотр запросов (выбор любого из списка для получения результата)

	<p>Создание запросов</p> <p>Просмотр таблиц базы данных и ER-диаграммы</p> <p>Внесение изменений в базу данных</p>
--	--

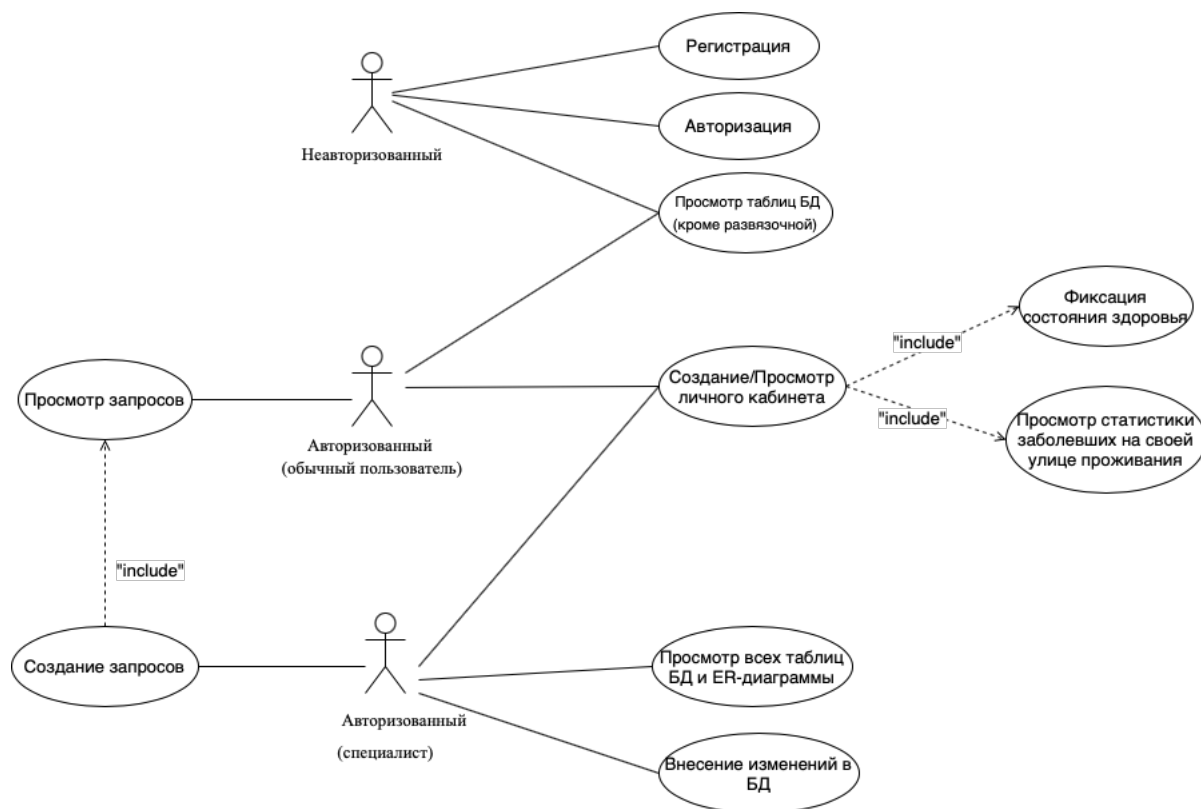


Рис. 1.1 – Диаграмма прецедентов

1.4 Определение понятия СУБД

Система управления базами данных, сокр. СУБД — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных. Главные задачи СУБД – это создание базы данных и манипуляция данными. Такая система позволяет обеспечить надежность хранения данных, их целостность и неизбыточность [14].

1.4.1 Основные функции СУБД

Основными функциями СУБД считаются следующие функции [14].

1. Управление данными во внешней памяти.

2. Управление данными в оперативной памяти с использованием дискового кэша.
3. Журнализация изменений, резервное копирование и восстановление базы данных после сбоев.
4. Поддержка языков БД.

1.4.2 Классификация СУБД по модели данных

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь [14]. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных.

По модели данных СУБД подразделяются на три типа [14].

1. Дореляционные.
2. Реляционные.
3. Постреляционные.

СУБД, основанные на дореляционной модели, в свою очередь, имеют три типа организации: системы, основанные на инвертированных списках, иерархические и сетевые системы управления базами данных.

В системах, которые основаны на инвертированных списках, таблицы и пути доступа к ним видны пользователям. При этом строки таблиц упорядочены в определенной физической последовательности, и для каждой таблицы определяется произвольное число ключей поиска, для которых создаются и поддерживаются системой индексы, которые видны пользователю. Главная особенность СУБД, основанных на инвертированных списках, заключается в отсутствии общих правил определения целостности базы данных. Примерами такой организации СУБД являются ранние реализации СУБД Datacom / DB и Adabas до перехода к реляционной технологии.

Иерархическая модель данных — логическая модель данных в виде древовидной структуры, представляющая собой совокупность элементов,

расположенных в порядке их подчинения от общего к частному [15]. В иерархических моделях основная структура представления данных имеет форму дерева. На самом высшем (первом) уровне иерархии находится только одна вершина, которая называется корнем дерева. Эта вершина имеет связи с вершинами второго уровня, вершины второго уровня имеют связи с вершинами третьего уровня и т.д. Связи между вершинами одного уровня отсутствуют. Следовательно, данные в иерархической структуре не равноправны – одни жестко подчинены другим. Доступ к информации возможен только по вертикальной схеме, начиная с корня, так как каждый элемент связан только с одним элементом на верхнем уровне и с одним или несколькими на низком. К недостаткам такой модели относятся: дублирование данных, негибкость, большой расход ресурсов памяти, относительная сложность модели по сравнению с реляционной моделью, возможность потери данных [15].

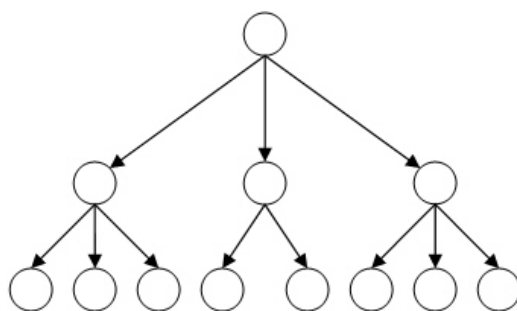


Рис. 1.1 – Концептуальная схема иерархической модели данных

Сетевая модель данных — логическая модель данных, являющаяся расширением иерархического подхода, строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных [15]. Основное отличие этих моделей заключается в том, что в иерархических структурах запись-потомок должна иметь в точности одного предка, а в сетевой структуре данных у потомка может иметься любое число предков. Главным недостатком сетевой модели данных являются жесткость и высокая сложность схемы базы данных, построенной на основе этой модели [15]. Так как логика процедуры выбора данных зависит от физической организации этих данных, то эта модель не является полностью

независимой от приложения. Иначе говоря, если будет необходимо изменить структуру данных, то нужно будет изменять и приложение.

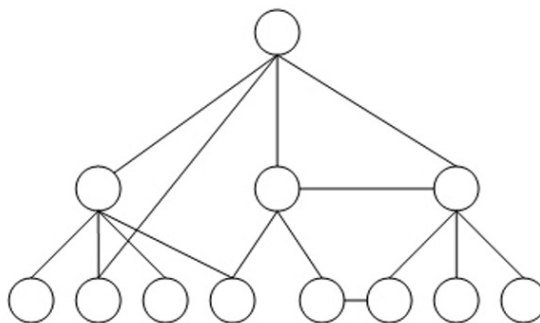


Рис. 1.2 – Концептуальная схема сетевой модели данных

Реляционная модель представляет собой совокупность данных, состоящую из набора двумерных таблиц [12]. В теории множеств таблице соответствует термин отношение (relation), физическим представлением которого является таблица, отсюда и название модели – реляционная. Реляционная модель является удобной и наиболее привычной формой представления данных. В реляционной модели, то есть при табличной организации данных, отсутствует иерархия элементов. Строки и столбцы могут быть просмотрены в любом порядке, поэтому высока гибкость выбора любого подмножества элементов в строках и столбцах. Любая таблица в реляционной базе состоит из строк, которые называются записями (кортежами), и столбцов, которые называются полями (атрибутами). На пересечении строк и столбцов находятся конкретные значения данных.

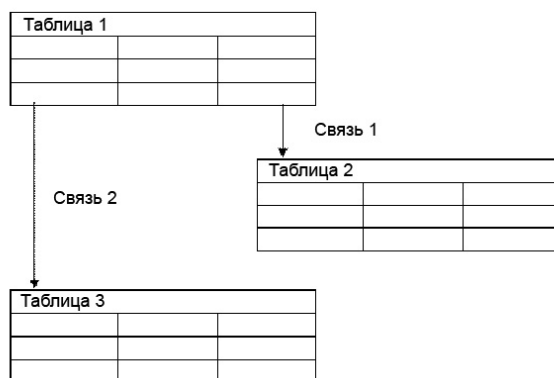


Рис. 1.3 – Концептуальная схема реляционной модели данных

Наиболее популярными реляционными СУБД являются Oracle, Microsoft SQL Server и PostgreSQL [4].

Постреляционная модель – это расширение реляционной модели [11]. Она использует трехмерные структуры, позволяя хранить в полях таблицы другие таблицы, расширяя таким образом возможности по описанию сложных объектов реального мира. Постреляционные СУБД поддерживают множественные группы, которые являются ассоциированными множественными полями, совокупность которых называется ассоциацией. За счет отсутствия требований на длину и количество полей в записях структура таблиц представляется более наглядной. Главное достоинство постреляционных СУБД состоит в возможности представления совокупности связанных реляционных таблиц в виде одной постреляционной таблицы, что делает их очень удобными для сложных объектов данных, таких как мультимедийные данные, данные для географических информационных систем и др. Недостатком является сложность обеспечения целостности данных [11]. Постреляционными СУБД являются такие системы, как uniVerse, Pick, Bubba и др.

1.5 Выводы из аналитического раздела

В данном разделе была проанализирована предметная область и сформулированы ограничения предметной области. Также были рассмотрены основные теоретические сведения о СУБД, их функциях и типах. Так как задача предполагает использование разнообразных запросов, включающее в себя выборки элементов строк и столбцов различной сложности, приоритетным свойством модели данных является гибкость, простота использования и независимость от приложения, поэтому в качестве модели организации данных была выбрана реляционная модель.

2. Конструкторская часть

В данном разделе будут рассмотрены основные структурные элементы базы данных, определена модель предметной области, сформулированы требования к программе, выделены основные компоненты веб-приложения и их функции, а также приведены схемы некоторых методов.

2.1 Проектирование базы данных

В соответствии с таблицей 1.1, содержащей данные, которые должны находиться в базе данных, можно выделить следующие таблицы и схемы.

1. Схема для серверной информации - server_data:

- 1.1. Таблица пользователей - users.
- 1.2. Таблица личного кабинета – private_office.
- 1.3. Таблица адресов - address.
- 1.4. Таблица ролей – roles.
- 1.5. Таблица запросов - query.
- 1.6. Развязочная таблица (пользователи - роли) – users_roles.

2. Схема для объекта анализа – информации о течении коронавирусной инфекции по всему миру - public:

- 2.2. Таблица пациентов - _person.
- 2.2. Таблица местоположения пациентов – _location.
- 2.3. Таблица симптомов - _symptom.
- 2.4. Таблица состояния здоровья – _status.
- 2.5. Развязочная таблица (пациенты - симптомы) – person_symptom.

На основании полученного описания основных понятий базы данных можно выделить модель предметной области больных коронавирусной инфекцией. Модель предметной области в классической нотации Питера Чена представлена на рисунке 2.1.

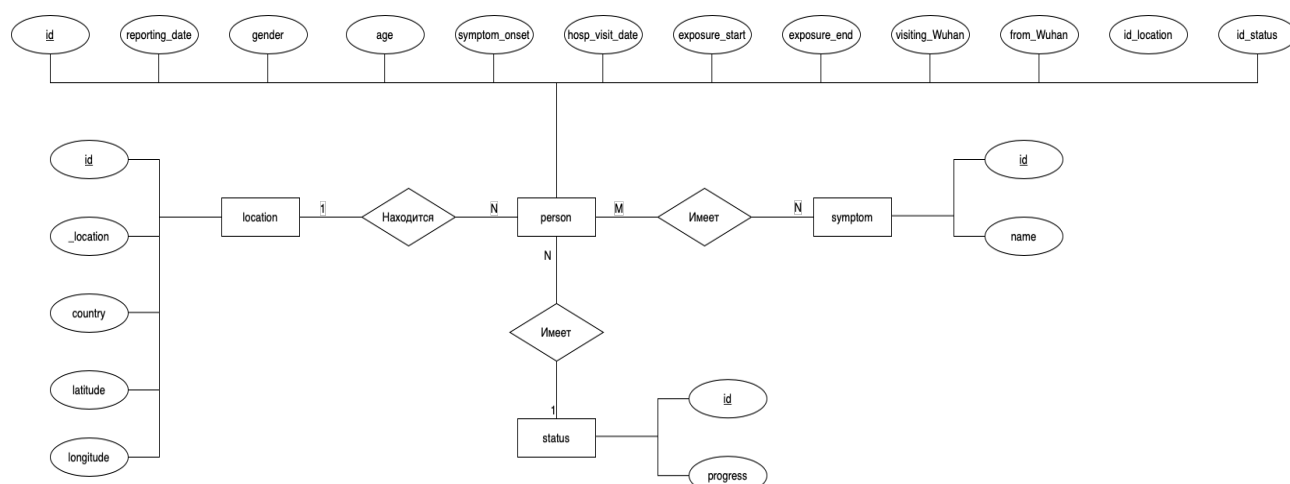


Рис. 2.1 – Модель предметной области в нотации Питера Чена

Таблица **_person** хранит информацию о пациентах с зарегистрированным случаем заражения COVID-19.

1. id – уникальный идентификатор пациента: int, Primary Key.
2. reporting_date – дата регистрации пациента в базе данных: varchar(64).
3. gender – пол пациента: varchar(64).
4. age – возраст пациента: int, ≥ 0 , ≤ 110 .
5. symptom_onset – дата начала симптомов: varchar(64).
6. hosp_visit_date – дата первого посещения больницы: varchar(64).
7. exposure_start – дата начала наблюдения: varchar(64).
8. exposure_end – дата окончания наблюдения: varchar(64).
9. visiting_Wuhan – посещал ли пациент Ухань: bool.
10. from_Wuhan – является ли пациент жителем Ухани: bool.
11. id_location – идентификатор местоположения пациента: Foreign Key (_location).
12. id_status – идентификатор состояния здоровья пациента: Foreign Key (_status).

Таблица **_location** хранит информацию о местоположении пациентов с зарегистрированным случаем заражения COVID-19.

1. id – уникальный идентификатор локации: int, Primary Key.
2. _location – название провинции/штата: varchar(64).

3. country название страны: varchar(64).
4. latitude –широта: float(7).
5. longitude – долгота: float(7).

Таблица **_symptom** хранит информацию о симптомах, характерных при COVID-19.

1. id – уникальный идентификатор симптома: int, Primary Key.
2. name – название симптома: varchar(64).

Таблица **_status** хранит информацию о возможных состояниях здоровья пациентов.

3. id – уникальный идентификатор локации: int, Primary Key.
4. progress – название состояния: varchar(64).

Если в приведенных выше таблицах какое-то из полей, кроме полей, содержащих идентификаторы, является пустым, предполагается, что в базе данных такое поле имеет значение NULL.

Таблица **person_symptom** – это развязочная таблица, позволяющая установить связь многие – ко – многим между таблицами **_person** и **_symptom**:

1. id_person – идентификатор пациента: Foreign Key (_person).
2. id_symptom - идентификатор симптома: Foreign Key (_symptom).

Данные для таблиц схемы **_public** содержат информацию для анализа в соответствии с основной тематикой веб-приложения. Это информация непосредственно о зарегистрированных случаях коронавирусной инфекции в мире. Соответствующие данные не генерируются, а берутся из открытых источников. В данной версии проекта используется публичная веб-платформа, предоставляющая наборы данных – Kaggle [3].

На рисунке 2.2 представлена диаграмма «сущность-связь», представляющая модель рассматриваемой предметной области (случаи заражения COVID-19 в мире).

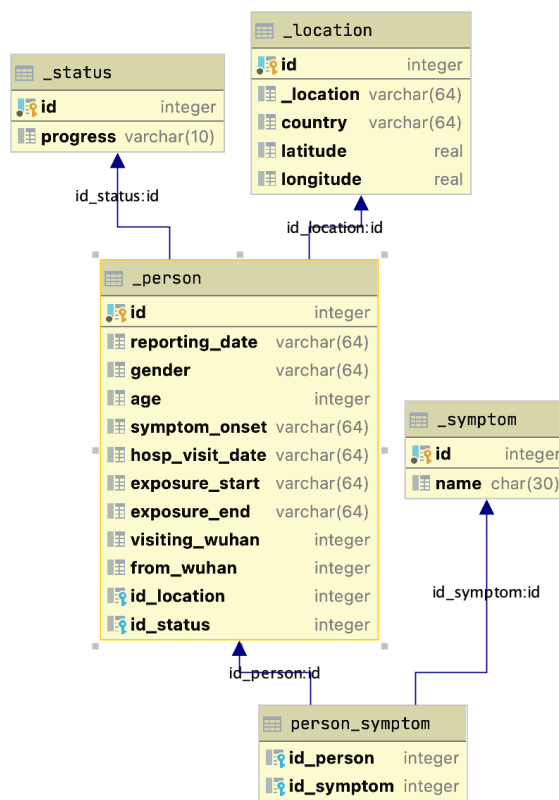


Рис. 2.2 – Диаграмма «сущность-связь» для схемы public

Таблица **users** хранит информацию о пользователях веб-приложения.

1. id - уникальный идентификатор пользователя: int, Primary Key.
2. email – адрес электронной почты пользователя, используется для регистрации и аутентификации, должен быть уникальным: varchar(64).
3. password – пароль пользователя, используется для регистрации и аутентификации, должен храниться в хешированном состоянии: varchar(100).

Таблица **roles** хранит информацию о ролях пользователей.

1. id - уникальный идентификатор роли: int, Primary Key.
2. name – название роли: varchar(64).
3. description – описание роли: varchar(255).

Всего предусмотрено 2 роли: user – обычный пользователь, specialist - специалист (допустимый функционал для каждого типа пользователей представлен в таблице 1.2).

Таблица **address** хранит информацию об улицах Москвы.

1. id - уникальный идентификатор улицы: int, Primary Key.
2. name – название улицы: varchar(64).
3. population – численность населения улицы: int.
4. infected – количество больных COVID-19 пользователей, живущих на этой улице: int.

Таблица **private_office** хранит информацию о личных кабинетах пользователей.

1. id - уникальный идентификатор личного кабинета: int, Primary Key.
2. name – имя пользователя: varchar(20).
3. lastname – фамилия пользователя: varchar(20).
4. gender – пол пользователя: varchar(20).
5. day_of_birth – день рождения пользователя: int.
6. month_of_birth – месяц рождения пользователя: varchar(20).
7. year_of_birth – год рождения пользователя: int.
8. address – название улицы, на которой живет пользователь: varchar(64).
9. health_status – состояние здоровья пользователя (1 – болен COVID-19, 0 - здоров): int.
3. id_user - идентификатор пользователя: Foreign Key (users).
10. id_address - идентификатор адреса: Foreign Key (address).

Таблица **users_roles** хранит информацию о существующих в системе и созданных специалистами запросах:

4. id_user – идентификатор за: Foreign Key (users).

5. id_role - идентификатор роли: Foreign Key (roles).

Таблица **query** – это развязочная таблица, позволяющая установить связь многие – ко – многим между таблицами _person и _symptom:

1. id - уникальный идентификатор запроса: int, Primary Key.
2. name – название запроса (необходимо для стороны сервера): varchar(64).
3. description – описание запроса (необходимо для стороны клиента): varchar(255).
4. sql_query – SQL-текст запроса: varchar(500).
5. columns_name – названия выходных столбцов (результатов запроса): varchar(255).

На рисунке 2.3 представлена диаграмма «сущность-связь» для сущностей схемы server_data.

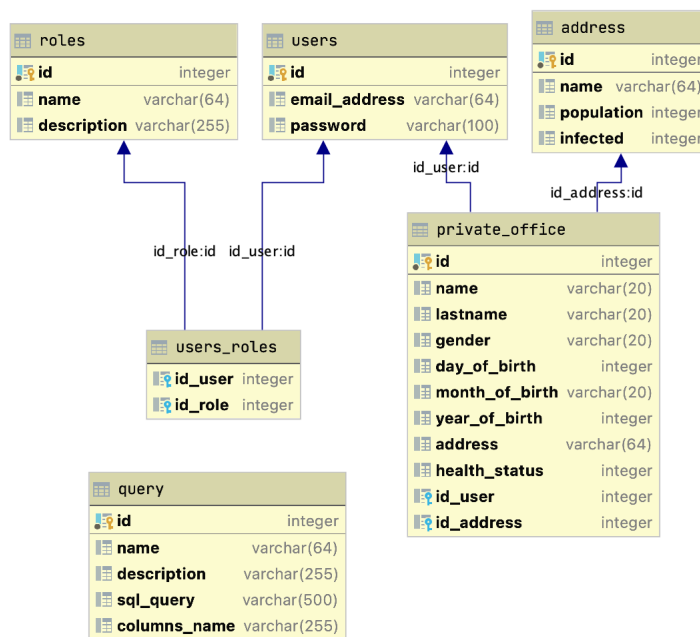


Рис. 2.3 – Диаграмма «сущность-связь» для схемы server_data

Триггер DML **change_percentage** – это триггер, который вступает в силу после любого успешного действия с таблицей `private_office` (добавления, изменения или удаления).

Функция **change_infected** – это триггерная функция, изменяющая значение атрибута `infected` таблицы `address` в зависимости от того, с каким статусом здоровья был добавлен/удален пользователь или на какой статус здоровья был изменен текущий. С помощью атрибутов `infected` и `population` таблицы `address` рассчитывается статистика заболевших COVID-19 на определенной улице.

2.2 Требования к программе

Для реализации поставленной задачи программа должна предоставлять следующие возможности.

1. Регистрация.
2. Аутентификация/Авторизация.
3. Просмотр таблиц базы данных, относящихся к зарегистрированным случаям заражения COVID-19.
4. Выбор запроса к базе данных из предложенного списка запросов для получения результата.
5. Создание личного кабинета зарегистрированного пользователя.
6. Фиксация состояния здоровья в личном кабинете (Здоров/Болен COVID-19).
7. Просмотр статистики заболевших COVID-19 пользователей, проживающих на той же улице, что и авторизованный пользователь, в его личном кабинете.
8. Дополнительные возможности специалиста:
 - 8.1. Просмотр дополнительных сведений о таблицах базы данных: название таблицы, схема таблицы.
 - 8.2. Просмотр развязочной таблицы.

8.3. Просмотр диаграммы «сущность-связь» (рисунок 2.1).

8.4. Создание запроса к базе данных.

8.5. Внесение изменений в базу данных.

2.3 Проектирование приложения

В рамках поставленной задачи курсового проекта необходимо реализовать веб-приложение с доступом к данным и удобным пользовательским интерфейсом. В соответствии с этим, в архитектуре программного обеспечения разделяются front-end и back-end.

front-end – это клиентская сторона веб-приложения, с которой взаимодействует пользователь через пользовательский интерфейс.

back-end – это программно-аппаратная часть сервиса, реализующая внутреннее функционирование веб-приложения.

Согласно принятой архитектуре, следует выделить три основных компонента проекта:

1. Компонент пользовательского интерфейса (UI).
2. Компонент бизнес-логики (серверная часть приложения).
3. Компонент доступа к данным.

Компонент пользовательского интерфейса предполагает использование шаблонов и статических файлов, располагающихся в поддиректориях проекта templates и static, соответственно.

Компонент бизнес-логики реализует методы API, предоставляемые пользовательским интерфейсом пользователю и осуществляет проверку данных, возвращаемых методами. Все файлы, реализующие бизнес-логику проекта, размещаются в поддиректории Backend.

Компонент доступа к данным отвечает за определение моделей уровня базы данных (сущности, дублирующие таблицы базы данных) и взаимодействие с базой данных, которое подразумевает:

- получение результата запроса;

- добавление записи в таблицу базы данных;
- обновление записей таблицы базы данных;
- выполнение запроса, созданного пользователем-специалистом (любые изменения в базе данных).

Все файлы, реализующие доступ к данным, размещаются в поддиректории Database.

Точка входа, конфигурационный файл и контроллер, обрабатывающий запросы пользователя, находятся в корневой директории проекта.

В соответствии с требованиями к программе, описанными в таблице 2.2, необходимы следующие методы.

1. Регистрация.

1.1. `signup_email` (POST, GET) – ввод пользователем email-адреса.

1.2. `signup_password` (POST, GET) – ввод пользователем пароля.

1.3. `signup_code` (POST, GET) – отправка на email-адрес пользователя кода для подтверждения регистрации и проверка введенного пользователем кода на совпадение с отправленным.

1.4. `create_privateoffice` (POST, GET) – создание личного кабинета.

2. Авторизация: `authorization` (POST, GET) – ввод пользователем данных для аутентификации: email-адрес, пароль.

3. `privateoffice_page` (POST, GET) - просмотр личного кабинета, смена статуса состояния здоровья, просмотр статистики (требуется авторизация).

4. `database` (POST, GET) - просмотр базы данных (для всех видов пользователей). Изменение базы данных (требуется роль специалиста).

5. `query` (POST, GET) - просмотр запросов к базе данных (требуется авторизация).

6. `create_query` (POST, GET) - создание запроса: ввод названия, описания, sql-текста запроса и названий результирующих столбцов (требуется роль специалиста).

7. `logout` (POST, GET) - выход из системы (требуется авторизация).

8. index (GET) - просмотр главной страницы веб-сайта.

Для проверки данных следует определить следующие функции.

1. check_signup_email – проверка введенного email-адреса на корректность и на то, не находится ли этот email-адрес в базе данных (не был ли пользователь ранее зарегистрирован).
2. check_signup_password – проверка пароля на соответствие требованиям. Пароль должен содержать цифры, строчные символы, прописные символы, а также иметь длину не менее 8 символов.
3. check_private_office – проверка введенных пользователем персональных данных.
4. check_query – проверка созданного запроса.

На рисунках 2.4 – 2.8 представлены схемы методов регистрации и авторизации.

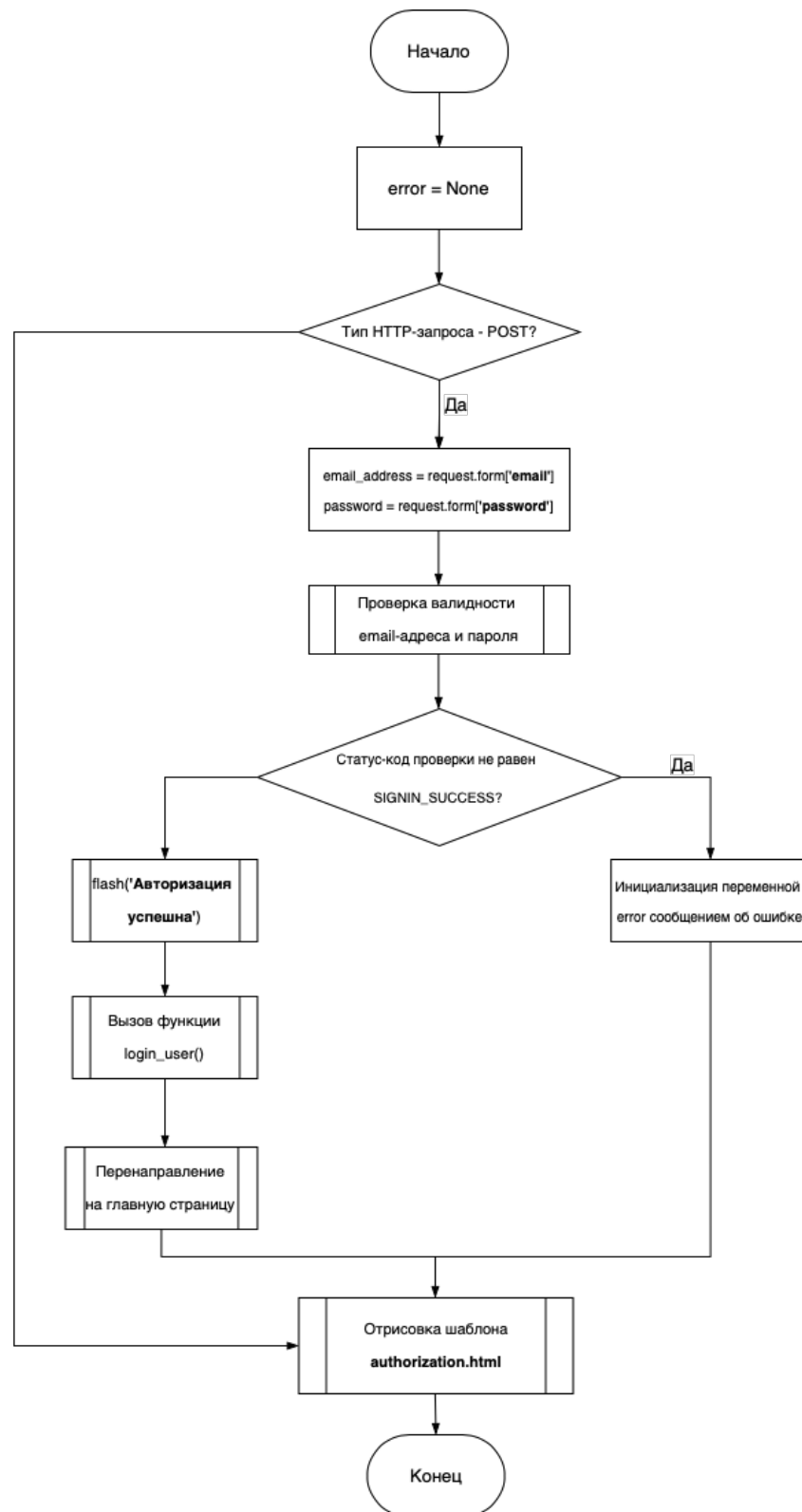


Рис. 2.4 – Схема метода авторизации

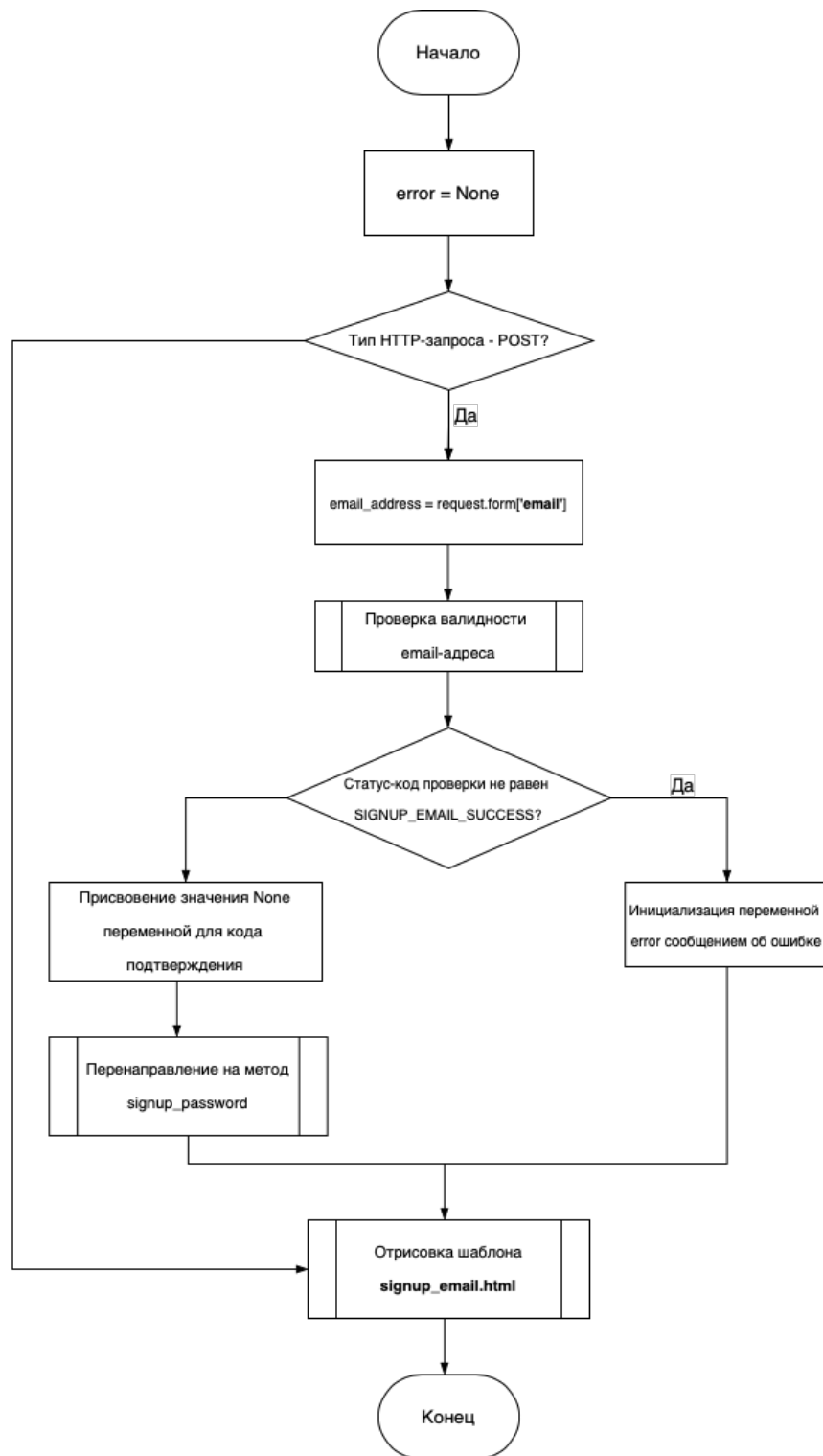


Рис. 2.5 – Схема метода регистрации (ввод email-адреса)

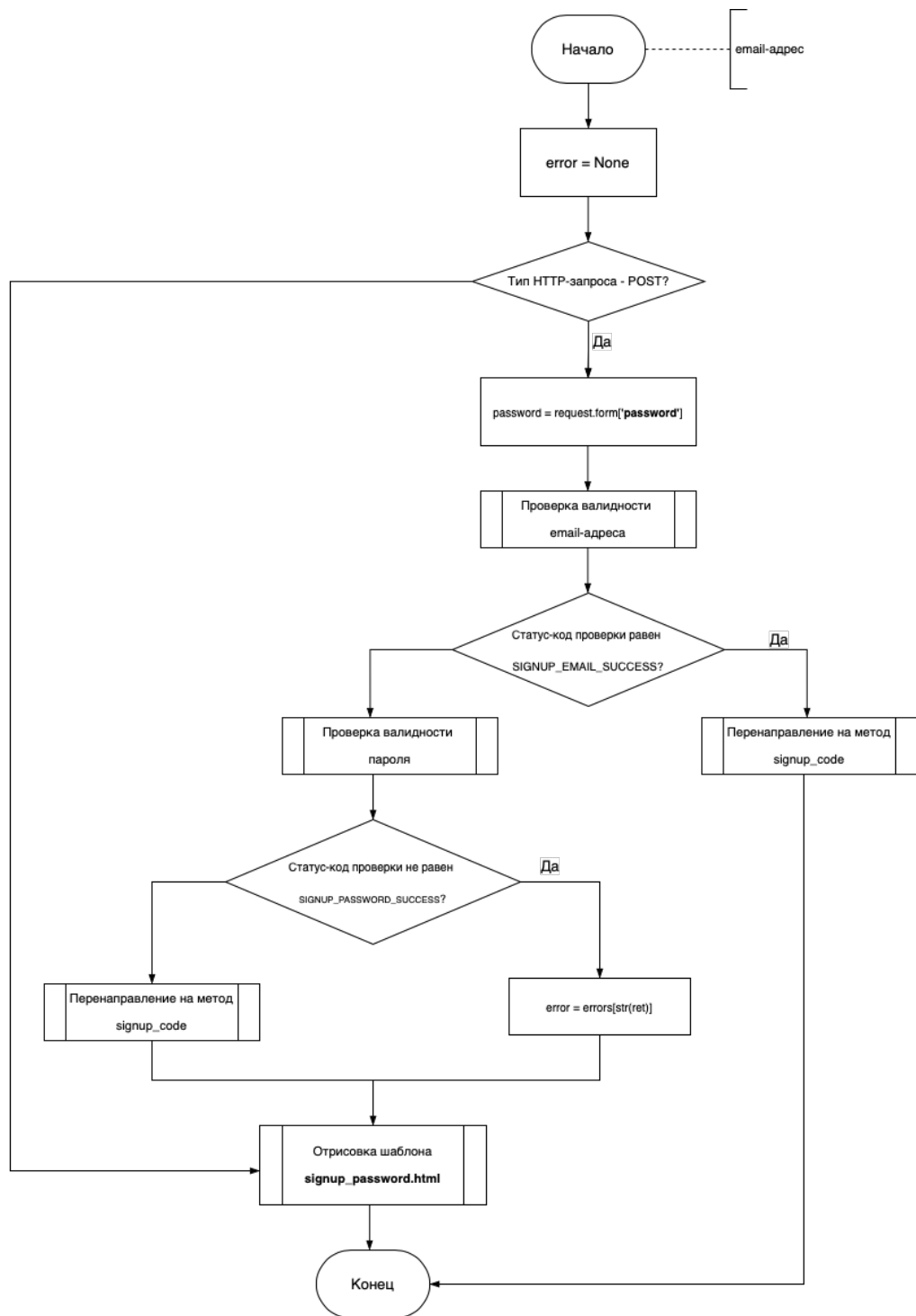


Рис. 2.6 – Схема метода регистрации (ввод пароля)

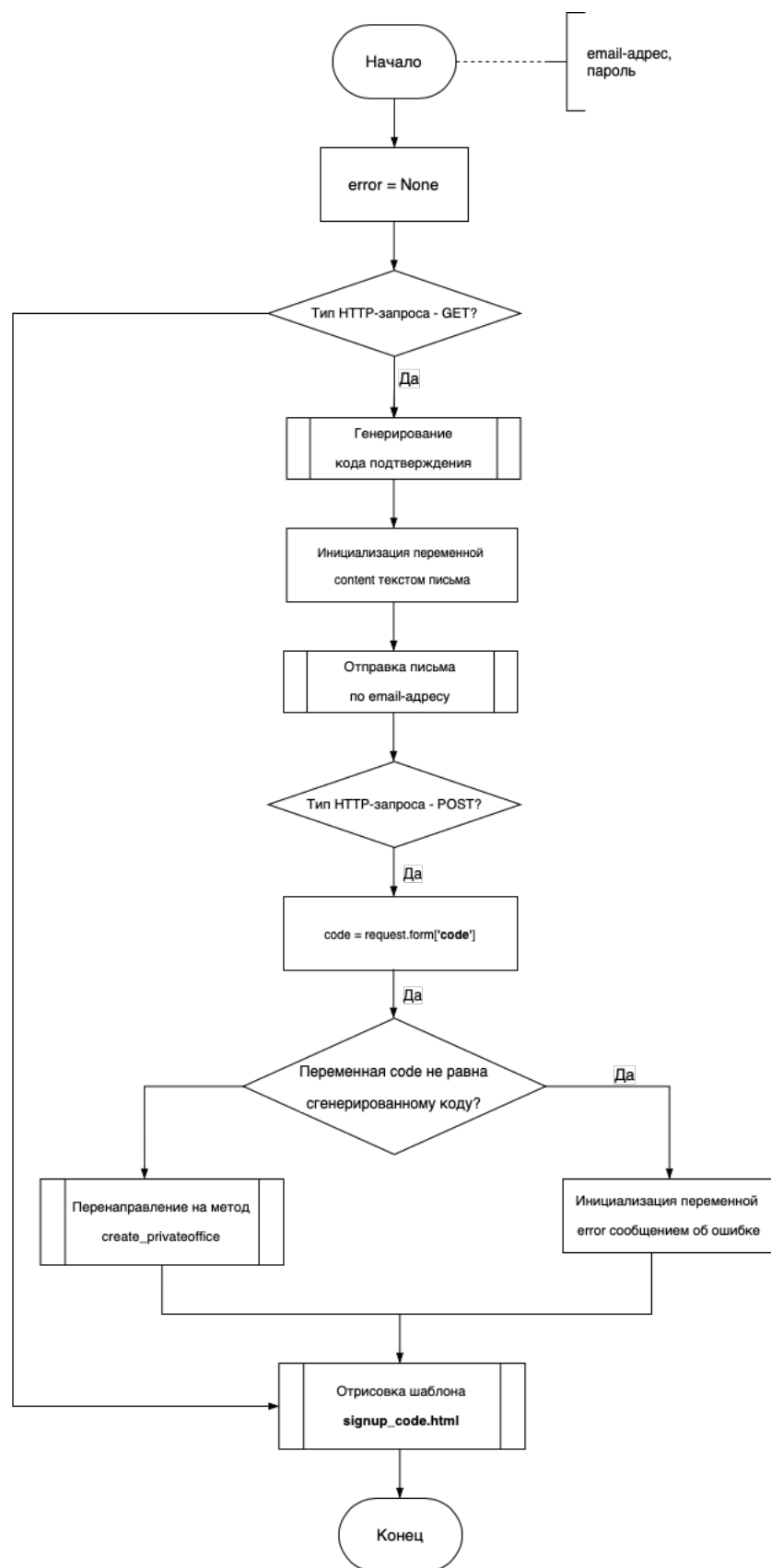


Рис. 2.7 – Схема метода регистрации (ввод кода подтверждения email-адреса)

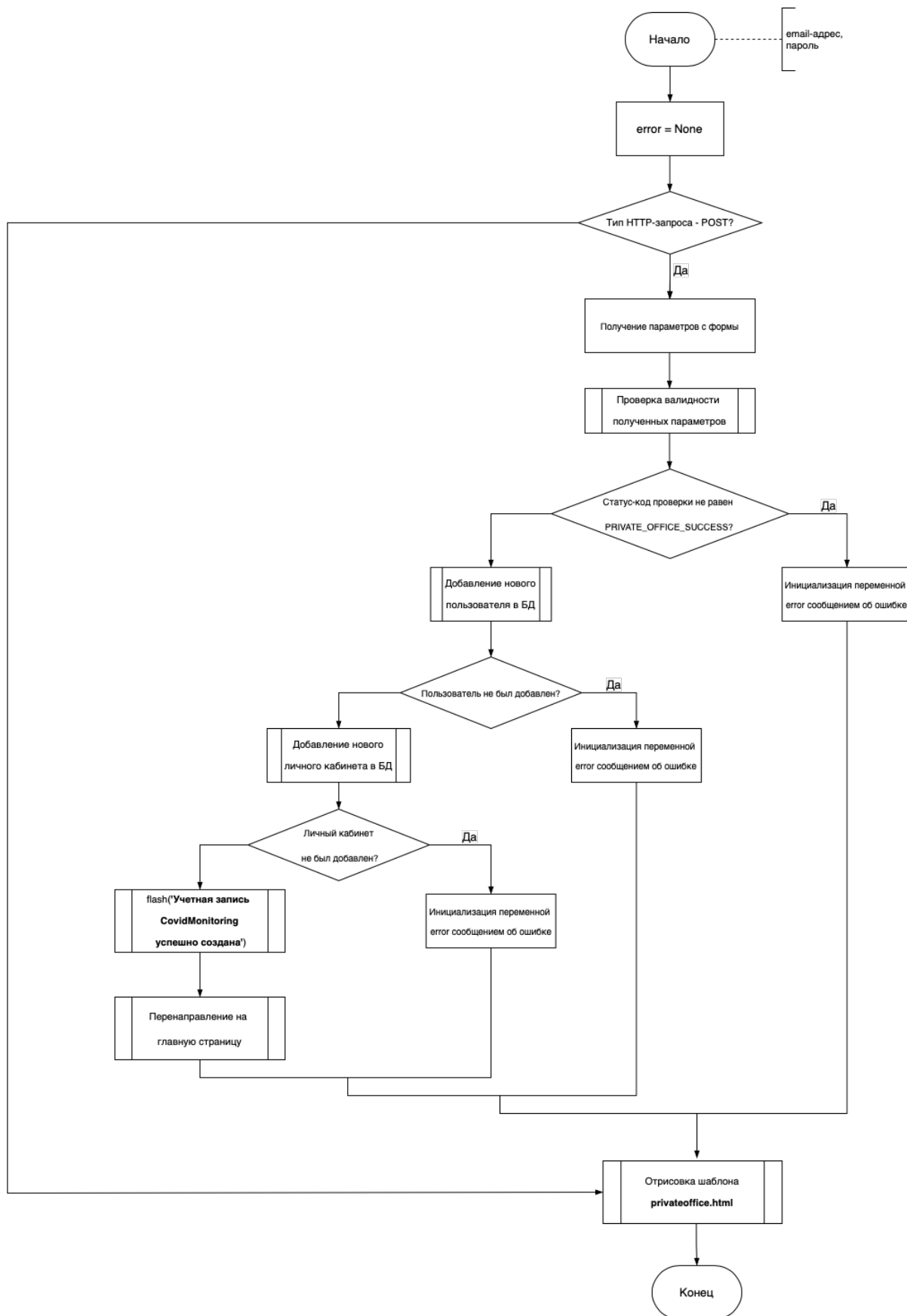


Рис. 2.8 – Схема метода регистрации (создание личного кабинета)

2.4 Выводы из конструкторского раздела

В данном разделе были рассмотрены основные структурные элементы базы данных, определена модель предметной области, сформулированы требования к программе, выделены основные компоненты веб-приложения и их функции, а также приведены схемы некоторых методов.

3. Технологическая часть

В технологической части будут выбраны язык программирования, среда разработки, СУБД и вспомогательные технологии с указанием причины выбора, приведены структура классов программы и листинги кода, а также продемонстрированы возможности веб-интерфейса.

3.1 Выбор и обоснование используемых технологий

В качестве языка программирования был выбран Python [7], так как:

1. Данный язык программирования удобен и прост в использовании.
2. Python хорошо совместим с различными фреймворками, упрощающими разработку.
3. Python содержит множество библиотек для работы с базами данных, например, flask_sqlalchemy, которая содержит класс SQLAlchemy [9], – набор инструментов, обеспечивающих работу с реляционными СУБД с применением технологии ORM.

Для создания веб-приложения в рамках учебного проекта был выбран Flask [2], потому что этот веб-фреймворк удобен для написания небольших проектов с возможностью выбора того, какие системы ORM и другие технологии будут использоваться. Flask реализуется с минимальными настройками, что позволяет лучше контролировать используемые компоненты и в учебных целях самостоятельно реализовывать некоторый функционал.

Передача данных в клиент-серверной модели приложения организована в соответствии с протоколом HTTP.

Для отправки письма пользователю на этапе регистрации по протоколу SMTP использовалась библиотека Python – yagmail [10].

В качестве среды разработки была выбрана PyCharm [6] так как:

1. Данная среда разработки обладает удобным редактором кода и графическим отладчиком.
2. PyCharm доступна в бесплатном формате для студентов.
3. PyCharm поддерживает новейшие версии библиотек Python и различных фреймворков.

4. PyCharm – кросс-платформенная среда разработки, что позволяет использовать ее на Mac OS.

В качестве СУБД была выбрана PostgreSQL [4], так как она основана на реляционной модели данных и имеет следующие достоинства.

1. Наличие открытого исходного кода.
2. Отсутствие ограничений на количество записей в таблицах и размер самой базы данных.
3. Поддержка большого количества типов данных.
4. Высокая производительность.
5. Поддержка операционных систем семейства Unix.

3.2 Структура и состав классов

В этом разделе будут рассмотрена структура и состав классов. Так как технология ORM, поддерживаемая классом SQLAlchemy ORM, позволяет использовать данные в терминах классов, а не таблиц данных, были описаны следующие классы:

- User – класс пользователей;
- Role – класс ролей пользователей;
- PrivateOffice – класс личных кабинетов пользователей;
- Address – класс адресов пользователей;
- Query – класс запросов;
- Person – класс пациентов, больных COVID-19;
- Status – класс состояния здоровья пациентов;
- Location – класс местоположения пациентов;
- Symptom – класс симптомов при COVID-19.

Структура каждого класса соответствует данным таблиц базы данных, описанным в разделе 2.1, и представлена в листингах 3.1 - 3.9.

Листинг 3.1 – класс User

```
class User(db.Model, UserMixin):  
    __table_args__ = {"schema": "server_data"}
```

```

__tablename__ = "users"

id = db.Column(db.Integer, primary_key=True)
email_address = db.Column(db.String(64), unique=True)
password = db.Column(db.String(100), unique=False)
private_office = db.relationship('PrivateOffice',
                                backref='user',
                                uselist=False)

roles = db.relationship('Role',
                        secondary=users_roles,
                        backref=db.backref('users', lazy='dynamic'))

def __repr__(self):
    return '<User %r>' % self.id

def set_password(self, password):
    self.password = generate_password_hash(password)

def check_password(self, password):
    return check_password_hash(self.password, password)

# Flask-Security
def has_role(self, *args):
    return set(args).issubset({role.name for role in self.roles})

```

ЛИСТИНГ 3.2 – класс Role

```

class Role(db.Model, RoleMixin):
    __table_args__ = {"schema": "server_data"}
    __tablename__ = "roles"

    id = db.Column(db.Integer(), primary_key=True)
    name = db.Column(db.String(64), unique=True)
    description = db.Column(db.String(255))

    def __str__(self):
        return self.name

```

ЛИСТИНГ 3.3 – класс PrivateOffice

```

class PrivateOffice(db.Model):
    __table_args__ = {"schema": "server_data"}
    __tablename__ = "private_office"

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(20), unique=False)
    lastname = db.Column(db.String(20), unique=False)
    gender = db.Column(db.String(20), unique=False)
    day_of_birth = db.Column(db.Integer, unique=False)
    month_of_birth = db.Column(db.String(20), unique=False)
    year_of_birth = db.Column(db.Integer, unique=False)
    address = db.Column(db.String(64), unique=False)
    health_status = db.Column(db.Integer, unique=False)
    id_user = db.Column(db.Integer, db.ForeignKey('server_data.users.id'),
nullable=False)
    id_address = db.Column(db.Integer, db.ForeignKey('server_data.address.id'),
nullable=False)

    def __repr__(self):
        return '<PrivateOffice %r>' % self.id

```

Листинг 3.4 – класс Address

```
class Address(db.Model):
    __table_args__ = {"schema": "server_data"}
    __tablename__ = "address"

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=False)
    population = db.Column(db.Integer, unique=False)
    infected = db.Column(db.Integer, unique=False)
    pr_offices = db.relationship('PrivateOffice', backref='addr',
lazy='dynamic')

    def __repr__(self):
        return '<Address %r>' % self.id
```

Листинг 3.5 – класс Query

```
class Query(db.Model):
    __table_args__ = {"schema": "server_data"}
    __tablename__ = "query"

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True)
    description = db.Column(db.String(255), unique=True)
    sql_query = db.Column(db.String(500), unique=True)
    columns_name = db.Column(db.String(255), unique=False)

    def __repr__(self):
        return '<Query %r>' % self.id
```

Листинг 3.6 – класс Person

```
class Person(db.Model):
    __table_args__ = {"schema": "public"}
    __tablename__ = "_person"

    id = db.Column(db.Integer, primary_key=True)
    reporting_date = db.Column(db.String(10), unique=False)
    gender = db.Column(db.String(10), unique=False)
    age = db.Column(db.Integer, unique=False)
    symptom_onset = db.Column(db.String(10), unique=False)
    hosp_visit_date = db.Column(db.String(10), unique=False)
    exposure_start = db.Column(db.String(10), unique=False)
    exposure_end = db.Column(db.String(10), unique=False)
    visiting_wuhan = db.Column(db.Integer, unique=False)
    from_wuhan = db.Column(db.Integer, unique=False)
    id_location = db.Column(db.Integer, db.ForeignKey('public._location.id'),
nullable=False)
    id_status = db.Column(db.Integer, db.ForeignKey('public._status.id'),
nullable=False)
    symptoms = db.relationship('Symptom', secondary=PersonSymptom,
backref=db.backref('people', lazy='dynamic'))

    def __init__(self, id, reporting_date, gender, age, symptom_onset,
hosp_visit_date,
exposure_start, exposure_end, visiting_Wuhan, from_Wuhan,
```



```

id_location, id_status):
    self.id = id
    self.reporting_date = reporting_date
    self.gender = gender
    self.age = age
    self.symptom_onset = symptom_onset
    self.hosp_visit_date = hosp_visit_date
    self.exposure_start = exposure_start
    self.exposure_end = exposure_end
    self.visiting_Wuhan = visiting_Wuhan
    self.from_Wuhan = from_Wuhan
    self.id_location = id_location
    self.id_status = id_status

    def __repr__(self):
        return '<Person %r>' % self.id

```

Листинг 3.7 – класс Status

```

class Status(db.Model):
    __table_args__ = {"schema": "public"}
    __tablename__ = "_status"

    id = db.Column(db.Integer, primary_key=True)
    progress = db.Column(db.String(10), unique=True)
    people = db.relationship('Person', backref='status', lazy='dynamic')

    def __init__(self, id, progress):
        self.id = id
        self.progress = progress

    def __repr__(self):
        return '<Status %r>' % self.id

```

Листинг 3.8 – класс Location

```

class Location(db.Model):
    __table_args__ = {"schema": "public"}
    __tablename__ = "_location"

    id = db.Column(db.Integer, primary_key=True)
    _location = db.Column(db.String(64), unique=False)
    country = db.Column(db.String(64), unique=False)
    latitude = db.Column(db.Float(7), unique=False)
    longitude = db.Column(db.Float(7), unique=False)
    people = db.relationship('Person', backref='location', lazy='dynamic')

    def __init__(self, id, location, country, latitude, longitude):
        self.id = id
        self.location = location
        self.country = country
        self.latitude = latitude
        self.longitude = longitude

    def __repr__(self):
        return '<Location %r>' % self.id

```

Листинг 3.9 – класс Symptom

```

class Symptom(db.Model):
    __table_args__ = {"schema": "public"}
    __tablename__ = "_symptom"

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(30), unique=True)

    def __init__(self, id, name):
        self.id = id
        self.name = name

    def __repr__(self):
        return '<Symptom %r>' % self.id

```

3.3 Взаимодействие с базой данных

Класс SQLAlchemy позволяет работать с движком базы данных PostgreSQL, проинициализировав объект этого класса и задав определенные параметры, например, URI базы данных, отвечающий за доступ к ней, в конфигурации веб-приложения.

Для запросов к базе данных и любых других операций над ней используются классы Session и Query, предоставляемые SQLAlchemy, а также модуль Python - psycopg2 [5], имеющий курсоры.

Пример использования функционала SQLAlchemy и psycopg2 представлены в листингах 3.10 и 3.11, соответственно.

Листинг 3.10 – применение классов Session и Query для создания запроса

```

def get_qty_by_status():
    quantityByStatus = db.session.query(Status.progress,
db.func.count(Person.id)). \
        join(Status.people). \
        group_by(Status.progress). \
        all()
    return quantityByStatus

```

Листинг 3.11 – применение psycopg2 для создания запроса

```

def init_con():
    try:
        con = psycopg2.connect(...) #Данные для подключения к базе данных
    except:
        return PSYCOPG2_ERROR

    cur = con.cursor()
    return cur, con

def sql_query_execute(sql_query):
    cur, con = init_con()

```

```

if cur == PSYCOPG2_ERROR:
    return PSYCOPG2_ERROR

try:
    cur.execute(sql_query)
except:
    return SQL_ERROR

try:
    con.commit()
except:
    return PSYCOPG2_ERROR

con.close()

return PSYCOPG2_SUCCESS

```

3.4 Триггер

Для отслеживания количества больных коронавирусной инфекцией на определенной улице используется триггер. Действиями триггера являются инструкции DML (INSERT, UPDATE и DELETE). Триггер DML и триггерная функция представлены в листингах 3.12 и 3.13, соответственно.

Листинг 3.12 – Триггер DML

```

CREATE TRIGGER change_percentage
AFTER INSERT OR UPDATE OR DELETE ON server_data.private_office
FOR EACH ROW
EXECUTE PROCEDURE change_infected();

```

Листинг 3.13 – Триггерная функция

```

CREATE OR REPLACE FUNCTION change_infected() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        IF NEW.health_status = 1 THEN
            UPDATE server_data.address
            SET infected = infected + 1
            WHERE id = NEW.id_address;
            RETURN NEW;
        END IF;
        RETURN NEW;
    ELSEIF TG_OP = 'UPDATE' THEN
        IF NEW.health_status = 0 THEN
            UPDATE server_data.address
            SET infected = infected - 1
            WHERE id = NEW.id_address;
            RETURN NEW;
        ELSEIF NEW.health_status = 1 THEN
            UPDATE server_data.address
            SET infected = infected + 1
            WHERE id = NEW.id_address;
            RETURN NEW;
        END IF;
    ELSIF TG_OP = 'DELETE' THEN

```

```

        IF OLD.health_status = 1 THEN
            UPDATE server_data.address
            SET infected = infected - 1
            WHERE id = OLD.id_address;
            RETURN OLD;
        END IF;
    RETURN OLD;
END IF;
END;
$$ LANGUAGE plpgsql;

```

3.5 Сведения о модулях программы

Проект состоит из 4 директорий.

1. Backend – основная бизнес-логика программы.
2. Database – содержит модули с ORM-представлениями таблиц данных и непосредственным взаимодействием с базой данных.
3. templates – содержит шаблоны представлений для клиентской стороны.
4. static – содержит статические файлы для шаблонизатора.

В основной директории располагаются следующие модули:

1. __init.py__ - точка входа в программу;
2. main.py – контроллер, содержащий маршрутизаторы;
3. config.py – файл конфигурации веб-приложения.

3.6 Графический интерфейс пользователя

Пользовательский интерфейс реализуется клиентской частью – браузером - посредством языка разметки HTML и формального языка описания внешнего вида веб-страницы CSS и представляет собой совокупность веб-страниц.

Для размещения и реализации различных элементов веб-интерфейса, включая иконки, стилизованное оформление кнопок и ссылок, был использован свободный набор инструментов для создания веб-приложений – Bootstrap [1].

На рисунке 3.1 представлена главная веб-страница для неавторизованного пользователя, отслеживаемая по URL <http://127.0.0.1:5000/>. Главная веб-страница содержит основную информацию для пользователя о функционале веб-приложения, а также выдает текстовый блок с главной информацией о коронавирусе при наведении курсора на изображение в левой части экрана. В

верхней части экрана пользователю предоставляются три ссылки: при переходе по ссылке «Информация» пользователю возвращается главная страница, по ссылке «База данных» - страница, отображающая базу данных пациентов, больных COVID-19. Однако при переходе по ссылке «Запросы» пользователь получит отказ в доступе, так как для этого ресурса необходима авторизация. Чтобы авторизоваться или зарегистрироваться, необходимо использовать кнопку «Войти».

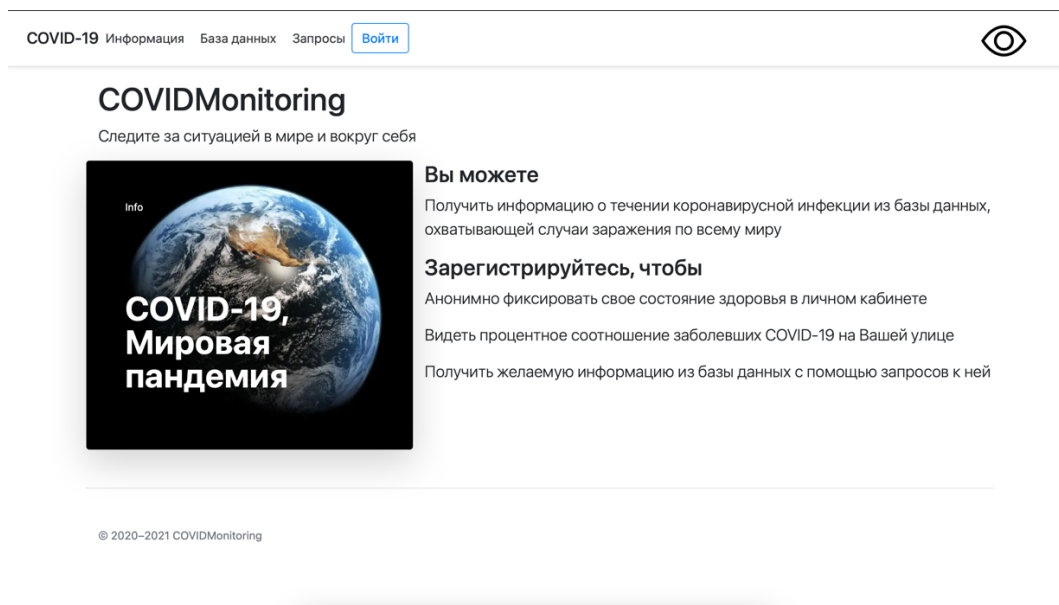
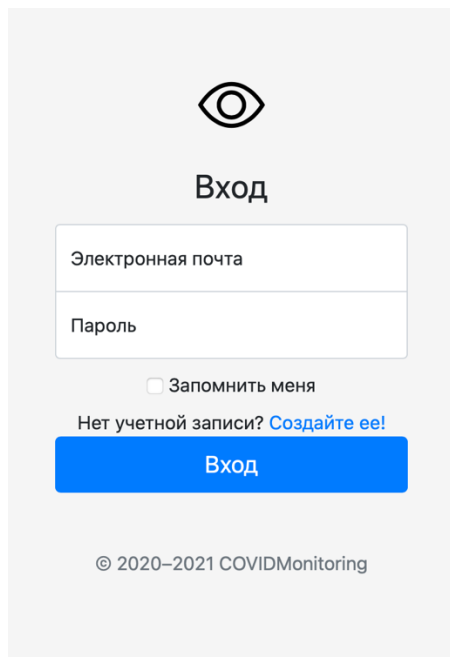


Рис. 3.1 – Главная веб-страница для неавторизованного пользователя

На рисунке 3.2 представлена веб-страница для авторизации или регистрации пользователя.

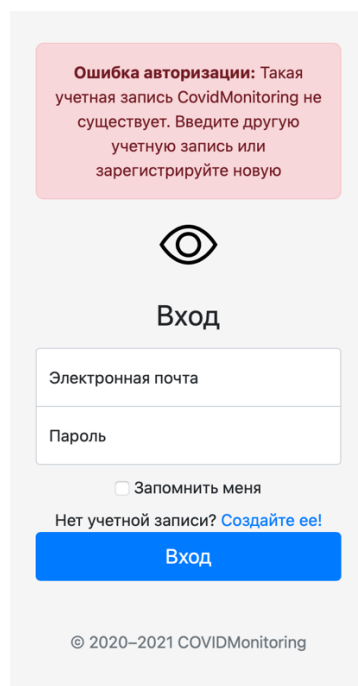


The image shows a login form for the COVIDMonitoring system. At the top is an eye icon. Below it is the title "Вход" (Login). The form contains two input fields: "Электронная почта" (Email) and "Пароль" (Password). Below these fields is a checkbox labeled "Запомнить меня" (Remember me). Under the checkbox is a link: "Нет учетной записи? [Создайте ее!](#)". At the bottom of the form is a blue button labeled "Вход". At the very bottom of the page is the copyright notice "© 2020–2021 COVIDMonitoring".

Рис. 3.2 – Веб-страница для авторизации или регистрации

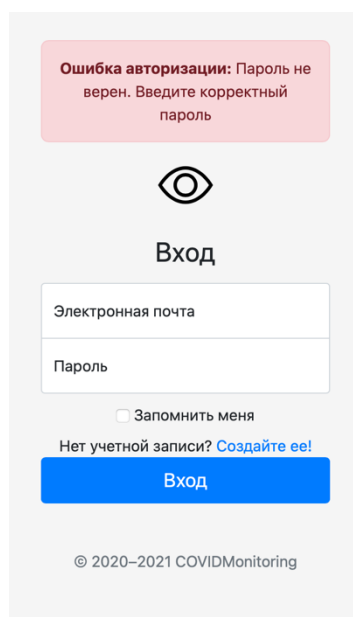
3.6.1 Авторизация

Для того чтобы авторизоваться, достаточно ввести email-адрес и пароль пользователя, который есть в системе. На рисунке 3.3 представлено предупреждение системы о попытке авторизации с незарегистрированными email-адресом, а на рисунке 3.4 – уведомление о попытке входа с неверным паролем. В обоих случаях пользователь не проходит аутентификацию.



The image shows the same login form as in Figure 3.2, but with an error message displayed at the top. The error message is in a red box and reads: "Ошибка авторизации: Такая учетная запись CovidMonitoring не существует. Введите другую учетную запись или зарегистрируйте новую". Below the error message is the eye icon, the title "Вход", the input fields for "Электронная почта" and "Пароль", the "Запомнить меня" checkbox, the link "Нет учетной записи? [Создайте ее!](#)", the blue "Вход" button, and the copyright notice "© 2020–2021 COVIDMonitoring".

Рис. 3.3 – Предупреждение о попытке входа незарегистрированного пользователя



Ошибка авторизации: Пароль не верен. Введите корректный пароль

Вход

Электронная почта

Пароль

☐ Запомнить меня

Нет учетной записи? [Создайте ее!](#)

Вход

© 2020–2021 COVIDMonitoring

Рис. 3.4 – Предупреждение о попытке входа с неверным паролем

Если данные успешно валидированы, пользователь перенаправляется на главную веб-страницу, получает уведомление об успешной авторизации, и теперь ему доступна веб-страница «Личный кабинет», как представлено на рисунке 3.5. Подробнее о личном кабинете пользователя описано в разделе 3.5.3.

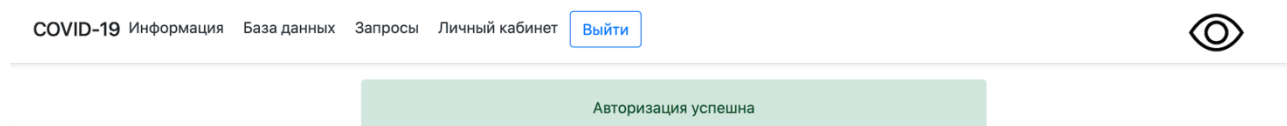


Рис. 3.5 – Главная веб-страница после успешной авторизации

3.6.2 Регистрация

Чтобы зарегистрироваться, пользователю нужно перейти по ссылке «Создайте ее!» (рис. 3.2). Регистрация осуществляется в три этапа.

1. Ввод email-адреса.
2. Ввод пароля.
3. Ввод кода безопасности из письма, отправленного на указанный email-адрес.
4. Создание личного кабинета.

Каждому этапу соответствует своя веб-страница. Обработчики запросов делают проверку вводимых данных на корректность и соответствие требованиям системы. Добавление пользователя в базу данных происходит при условии успешного завершения всех четырех этапов.

На рисунке 3.6 представлена веб-страница, на которую будет направлен пользователь после перехода по ссылке для регистрации, а также пример ввода email-адреса пользователя.

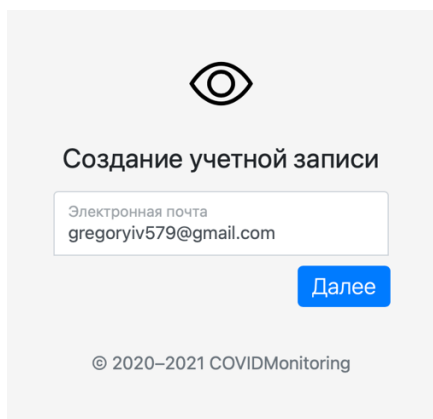


Рис. 3.6 – Пример ввода email-адреса пользователя при регистрации

После ввода email-адреса пользователю будет предложено придумать пароль для учетной записи, который должен соответствовать требованиям по длине и составу токена: длина не менее 8 символов, обязательно наличие строчных, прописных символов и цифр. Пример несоответствия требованиям приведен на рисунке 3.7.

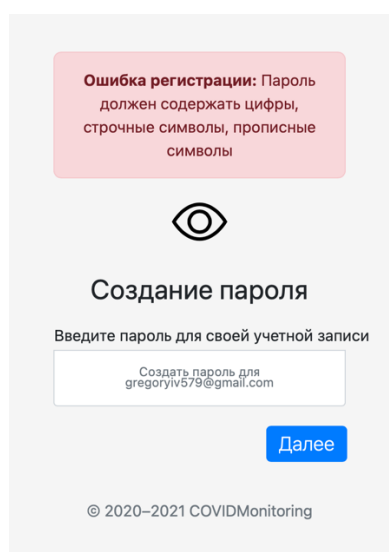


Рис. 3.7 – Пример неверного ввода пароля при регистрации

При успешном вводе пароля на указанный email-адрес пользователя будет отправлено письмо для подтверждения адреса электронной почты. Содержание письма представлено на рисунке 3.8.

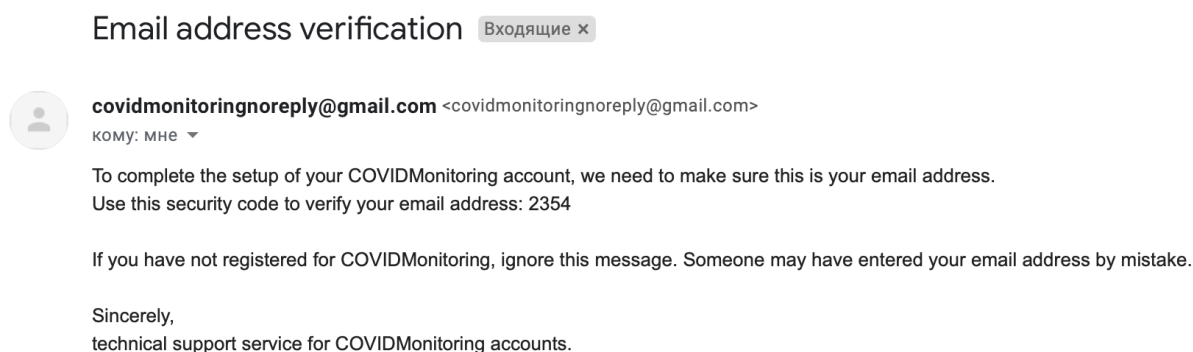


Рис. 3.8 – Содержание письма, отправляемого на email-адрес пользователя

Пример ввода кода безопасности из текста письма приведен на рисунке 3.9.

The image shows a web form titled "Проверка электронной почты" (Email verification) with an eye icon. The text on the form says: "Введите код, отправленный на gregoryiv579@gmail.com". There is a text input field containing the number "2354". Below the input field is a blue button labeled "Далее" (Next). At the bottom of the form, it says "© 2020–2021 COVIDMonitoring".

Рис. 3.9 – Пример ввода кода безопасности

Если код, введенный пользователем, совпал с тем, что был отправлен в тексте письма, отображается веб-страница для создания личного кабинета, которая приведена на рисунках 3.10, 3.11. Каждое поле так же валидируется обработчиком запроса на создание личного кабинета. На этом этапе пользователь выбирает тип аккаунта: «Пользователь» или «Специалист», - что определит его права и доступные ему ресурсы.

Рис. 3.10 – Веб-страница создания личного кабинета (часть 1)

Рис. 3.11 – Веб-страница создания личного кабинета (часть 2)

При успешном заполнении полей формы пользователь с указанными персональными данными и данными для авторизации будет добавлен в базу данных пользователей (таблица users схемы server_data).

3.6.3 Личный кабинет

Личный кабинет содержит информацию о следующих данных пользователя: имя, фамилия, дата рождения, пол, адрес (название улицы), статус состояния здоровья. В нижней части экрана отображается статистика больных COVID-19 пользователей веб-приложения, проживающих на данной улице. На рисунках 3.12, 3.13 представлен пример личного кабинета пользователя.

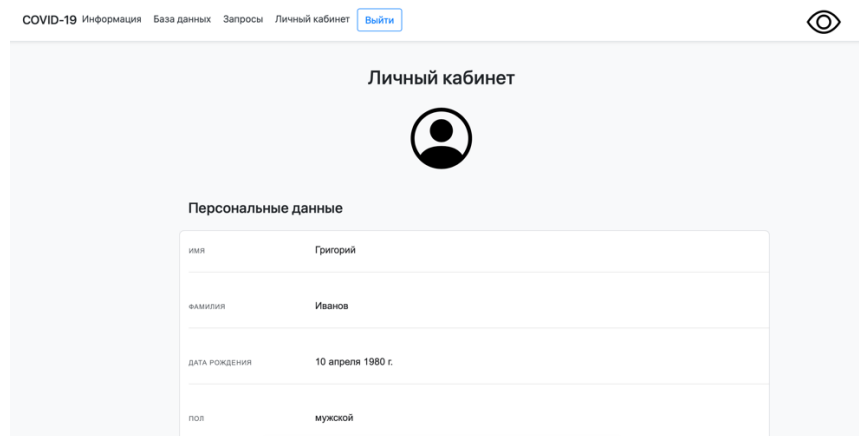


Рис. 3.12 – Пример личного кабинета пользователя (часть 1)

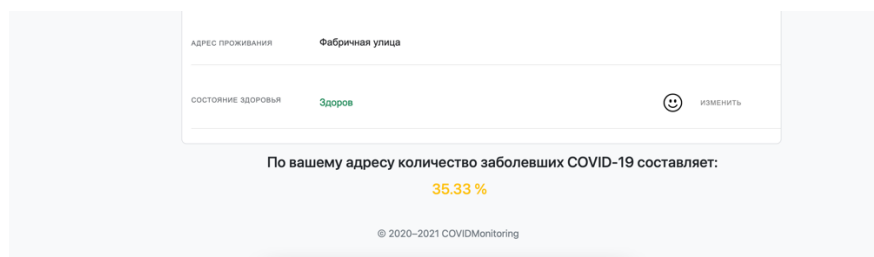


Рис. 3.13 – Пример личного кабинета пользователя (часть 2)

Пользователь имеет возможность изменить текущий статус состояния здоровья, нажав на кнопку «Изменить». Процесс смены статуса здоровья и результат показаны на рисунках 3.14, 3.15.

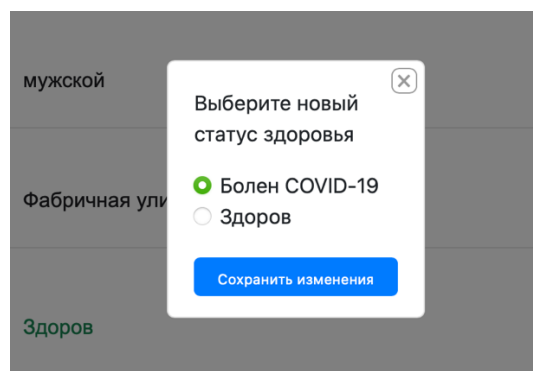


Рис. 3.14 – Смена статуса состояния здоровья

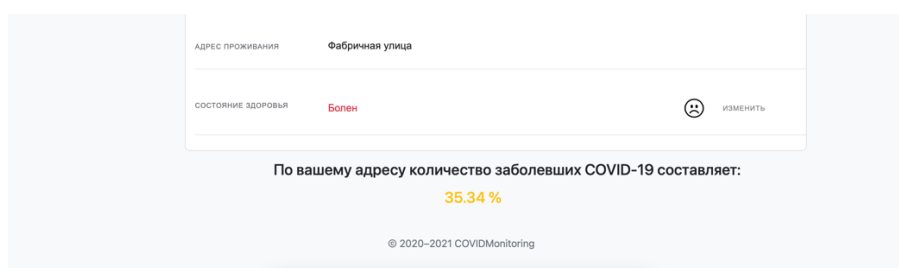


Рис. 3.15 – Результат смены статуса состояния здоровья со «Здоров» на «Болен COVID-19»

3.6.4 База данных

Пользователь с любой ролью может просматривать таблицы базы данных для получения необходимой информации и анализа. Однако, только для специалиста будет отображаться окно для создания SQL-запроса к базе данных для внесения изменений в нее. Помимо этого, специалисту доступно более подробное описание таблиц (название и схема таблицы), развязочная таблица и ER-диаграмма, за отображение которой отвечает последовательность графических элементов «Дополнительно» -> «ER-диаграмма». Взаимодействовать с таблицами с веб-страницы нельзя, информация только для ознакомления.

На рисунках 3.16 и 3.17 представлена веб-страница базы данных для обычного пользователя и специалиста, соответственно.

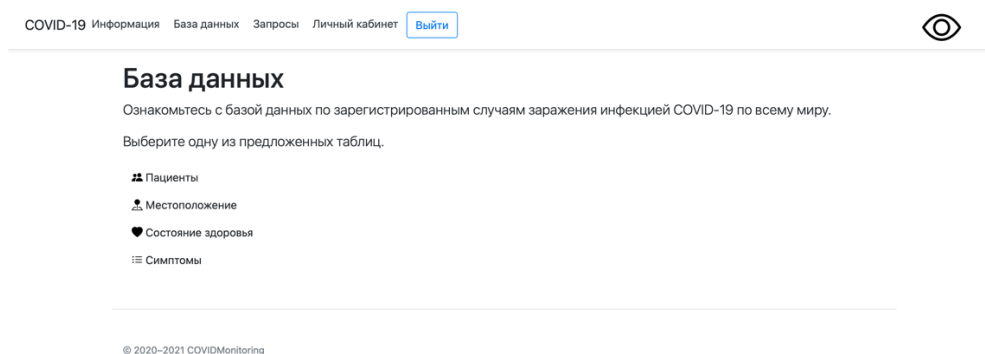


Рис. 3.16 – Веб-страница базы данных для обычного пользователя

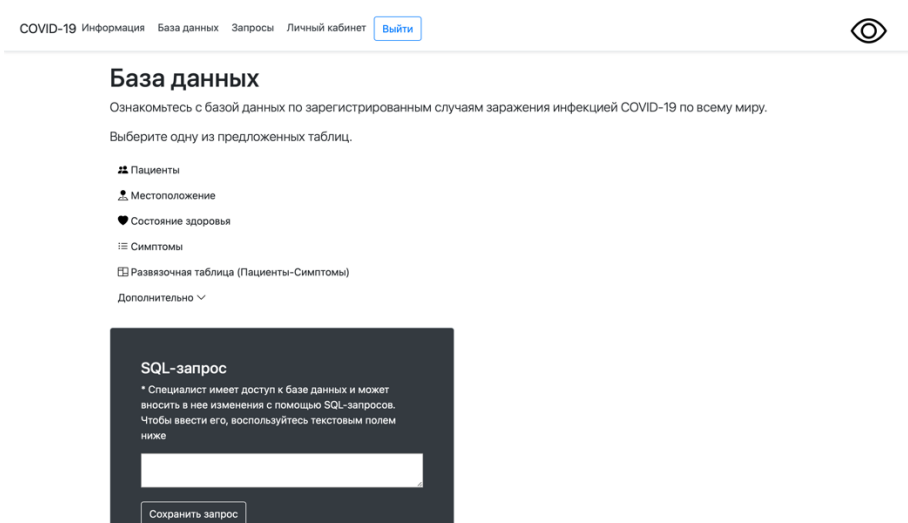


Рис. 3.17 – Веб-страница базы данных для специалиста

Чтобы открыть любую таблицу, нужно нажать на кнопку, соответствующую содержанию интересующей таблицы: «Пациенты», «Местоположение», «Состояние здоровья», «Симптомы» или «Развязочная таблица (Пациенты-Симптомы)» (для специалиста). Чтобы убрать таблицу, нужно воспользоваться кнопкой «Скрыть» над правым верхним углом таблицы.

Пример просмотра некоторых таблиц базы данных для специалиста приведен на рисунках 3.18, 3.19.

Пациенты

Таблица содержит информацию о людях с подтвержденным случаем COVID-19

Название таблицы в базе данных: _person

Название схемы: public

Скрыть

id	reporting_date	gender	age	symptom_onset	hosp_visit_date	exposure_start	exposure_end	visiting_wuhan	from_wuhan
1	1/20/2020	male	66	01/03/20	01/11/20	12/29/2019	01/04/20	1	0
2	1/20/2020	female	56	1/15/2020	1/15/2020	None	01/12/20	0	1
3	1/21/2020	male	46	01/04/20	1/17/2020	None	01/03/20	0	1
4	1/21/2020	female	60	None	1/19/2020	None	None	1	0
5	1/21/2020	male	58	None	1/14/2020	None	None	0	0
6	1/21/2020	female	44	1/15/2020	None	None	None	0	1
7	1/21/2020	male	34	01/11/20	None	None	None	0	1
8	1/21/2020	male	27	1/11/2020	1/20/2020	01/01/20	01/11/20	1	0

Рис. 3.18 – Просмотр таблицы «Пациенты» специалистом

Состояние здоровья

Таблица содержит информацию о состоянии здоровья людей с подтвержденным случаем COVID-19

Пояснения:
RECOVERED - выздоровевшие
UNKNOWN - состояние пациента на данный момент неизвестно
DEATH - летальный исход

Название таблицы в базе данных: _status

Название схемы: public

Скрыть

id	progress
1	RECOVERED
2	UNKNOWN
3	DEATH

☰ Симптомы

☐ Развязочная таблица (Пациенты-Симптомы)

Дополнительно ▾

Рис. 3.18 – Просмотр таблицы «Состояние здоровья» специалистом

3.6.5 Запросы

Авторизованному пользователю доступна ссылка «Запросы» в верхней панели. Данная веб-страница будет отличаться для специалиста только новым элементом «Создать запрос». Список запросов находится в левой части экрана,

при этом запросы, уже существующие в системе на момент запуска сервера, помечаются меткой «Admin», а запросы, созданные специалистом, – меткой «Specialist». Результат выбранного запроса появится в правой части экрана после нажатия на кнопку в средней части экрана.

На рисунках 3.19, 3.20 представлена веб-страница запросов для обычного пользователя и специалиста, соответственно.

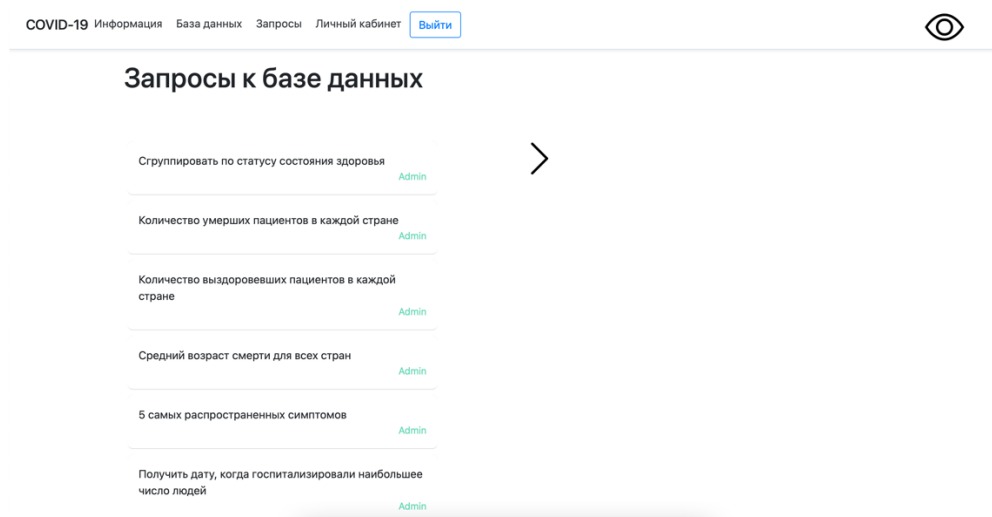


Рис. 3.19 – Веб-страница запросов для обычного пользователя

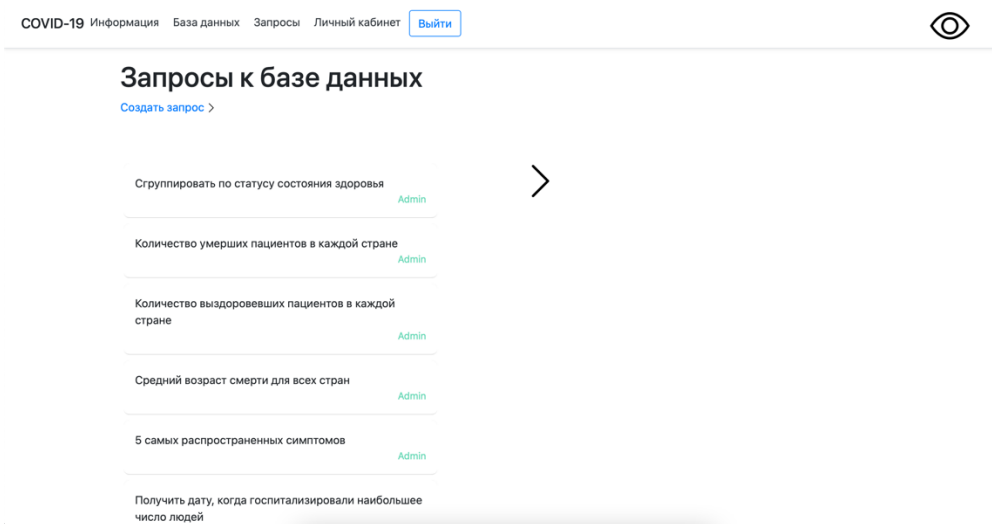


Рис. 3.20 – Веб-страница запросов для специалиста

На рисунке 3.21 приведен пример результата для запроса «5 самых распространенных симптомов».

Средний возраст смерти для всех стран
Admin

5 самых распространенных симптомов
Admin

Получить дату, когда госпитализировали наибольшее число людей
Admin

Получить название симптома, который чаще всего наблюдался у пациентов с летальным исходом
Admin

Максимальный возраст пациента
Specialist

Название симптома

itchy throat	+
high fever	+
conjunctivitis	+
flu	+
difficulty breathing	+

Рис. 3.21 – Пример результата запроса

3.6.6 Создание запроса

После перехода по ссылке «Создать запрос» пользователь-специалист будет перенаправлен на веб-страницу создания запроса, представленную на рисунке 3.22.

COVID-19 Информация База данных Запросы Личный кабинет Выйти

Создание запроса

Чтобы создать новый запрос, укажите название и текст запроса

Пример названия: recoveredQuantity

Пример текста: Получить число выздоровевших пациентов

Сохранить запрос

Введите SQL-запрос

Результат представляется в виде таблицы. Чтобы сделать ответ на ваш запрос понятным пользователям, укажите в поле ниже название каждого столбца через знак ;

Пример названий столбцов: ID; Пол; Возраст

© 2020–2021 COVIDMonitoring

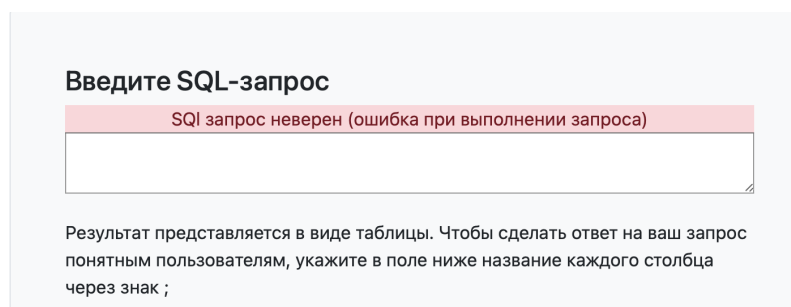
Рис. 3.22 – Веб-страница создания запроса

Данная веб-страница содержит 4 текстовых поля для ввода данных запроса:

1. Название запроса.
2. Текст запроса (то, что видят пользователи).
3. SQL-запрос.
4. Названия результирующих столбцов.

Если в одном из этих полей допущена ошибка, например, поле осталось незаполненным или SQL-запрос неверен, пользователь получит сообщение

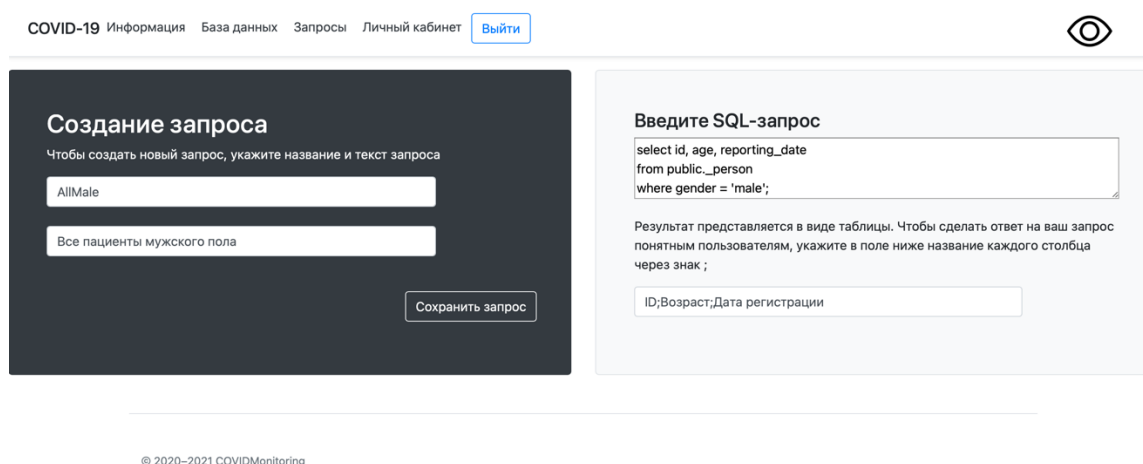
об ошибке. На рисунке 3.23 представлен пример уведомления об ошибке при неверном SQL-тексте запроса.



The screenshot shows a web interface for entering an SQL query. At the top, it says "Введите SQL-запрос". Below this is a red error message: "SQL запрос неверен (ошибка при выполнении запроса)". There is an empty text input field below the error message. At the bottom, there is a paragraph of text: "Результат представляется в виде таблицы. Чтобы сделать ответ на ваш запрос понятным пользователям, укажите в поле ниже название каждого столбца через знак ;".

Рис. 3.24 – Уведомление об ошибке в SQL-тексте запроса

На рисунке 3.25 представлен пример верно заполненных полей. Орфографические ошибки в названии и тексте запроса, а также смысловые ошибки (например, несоответствие текста запроса результату) не контролируются.



The screenshot shows a web interface for creating a new SQL query. At the top, there is a navigation bar with links: "COVID-19", "Информация", "База данных", "Запросы", "Личный кабинет", and a "Выйти" button. On the left, there is a dark grey box titled "Создание запроса" with the subtitle "Чтобы создать новый запрос, укажите название и текст запроса". It contains two text input fields: the first contains "AllMale" and the second contains "Все пациенты мужского пола". Below these fields is a "Сохранить запрос" button. On the right, there is a light grey box titled "Введите SQL-запрос". It contains a text input field with the SQL query: "select id, age, reporting_date from public__person where gender = 'male';". Below this field is a paragraph of text: "Результат представляется в виде таблицы. Чтобы сделать ответ на ваш запрос понятным пользователям, укажите в поле ниже название каждого столбца через знак ;". Below this text is a text input field containing "ID;Возраст;Дата регистрации". At the bottom of the page, there is a copyright notice: "© 2020–2021 COVIDMonitoring".

Рис. 3.25 – Пример верно заполненных полей для создания нового запроса

Если все поля заполнены верно, запрос будет добавлен в таблицу query схемы server_data, а пользователь будет перенаправлен на веб-страницу запросов и получит уведомление об успешном создании запроса, как представлено на рисунке 3.26.

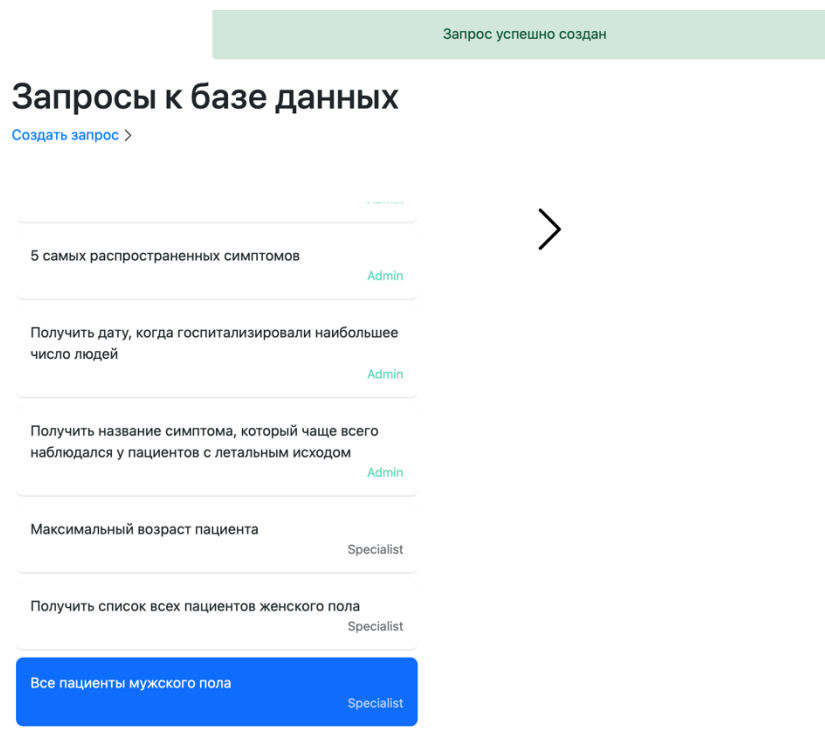


Рис. 3.26 – Пример успешно созданного запроса

Результат созданного запроса приведен на рисунке 3.27.

ID	Возраст	Дата регистрации
1	66	1/20/2020
3	46	1/21/2020
5	58	1/21/2020
7	34	1/21/2020
8	37	1/21/2020
9	39	1/21/2020
10	56	1/21/2020
13	37	1/21/2020
14	51	1/21/2020
15	57	1/22/2020
16	56	1/22/2020
17	50	1/22/2020

Рис. 3.27 – Результат созданного запроса

3.7 Выводы из технологического раздела

В данном разделе был выбран язык программирования Python и среда разработки PyCharm, для реализации веб-технологий был выбран фреймворк Flask. Также были выбраны технологии для взаимодействия с базой данных,

описаны структура и состав реализованных классов, предоставлены сведения о модулях программы и листинги кода и продемонстрированы возможности веб-интерфейса.

Заключение

В ходе выполнения данного проекта были рассмотрены существующие модели данных, на основании чего был сделан и обоснован выбор наиболее подходящей под поставленную задачу СУБД. Также были спроектированы база данных, серверная часть веб-приложения и веб-интерфейс, реализующий основной функционал приложения.

Для реализации базы данных, содержащей информацию о больных коронавирусной инфекцией и пользователях веб-сайта, и веб-приложения для ее анализа были решены следующие задачи.

1. Проанализирована предметная область, сформулированы ограничения предметной области.
2. Проведен анализ существующих СУБД.
3. Спроектирована база данных, содержащая информацию о больных коронавирусной инфекцией и пользователях веб-сайта.
4. Реализовано веб-приложение для анализа спроектированной базы данных.

Список использованной литературы

1. Bootstrap [Электронный ресурс] Режим доступа: <https://getbootstrap.com>, (дата обращения 18.04.2021)
2. Flask [Электронный ресурс] Режим доступа: <https://flask.palletsprojects.com/en/2.0.x/>, (дата обращения 18.04.2021)
3. Kaggle [Электронный ресурс] Режим доступа: <https://www.kaggle.com>, (дата обращения 05.04.2021)
4. Postgresql [Электронный ресурс] Режим доступа: <https://www.postgresql.org>, (дата обращения 05.04.2021)
5. Psycopg2 [Электронный ресурс] Режим доступа: <https://www.psycopg.org/docs/>, (дата обращения 15.04.2021)
6. PyCharm [Электронный ресурс] Режим доступа: <https://www.jetbrains.com/ru-ru/pycharm/>, (дата обращения 05.04.2021)
7. Python [Электронный ресурс] Режим доступа: <https://www.python.org>, (дата обращения 05.04.2021)
8. SQL [Электронный ресурс] Режим доступа: <https://www.sql.ru>, (дата обращения 04.04.2021)
9. SQLAlchemy [Электронный ресурс] Режим доступа: <https://www.sql.ru>, (дата обращения 14.04.2021)
10. yagmail [Электронный ресурс] Режим доступа: <https://pypi.org/project/yagmail/>, (дата обращения 07.04.2021)
11. Бородина А. И. Постреляционная, многомерная и объектно-ориентированная модели данных [Электронный ресурс] Режим доступа: http://www.bseu.by/it/tohod/lekci2_4.htm (дата обращения 10.05.2020)
12. Бородина А. И. Реляционная модель данных [Электронный ресурс] Режим доступа: http://www.bseu.by/it/tohod/lekci2_3.htm (дата обращения 10.05.2020)
13. Всемирная организация здравоохранения [Электронный ресурс] Режим доступа: <https://www.who.int/ru>, (дата обращения 15.04.2021)

14. Кузнецов С. Д. Система управления базами данных [Электронный ресурс]
Режим доступа: https://bigenc.ru/technology_and_technique/text/3666497,
(дата обращения 10.05.2020)
15. Сатиндер Б. Г. Introduction Database Management System [Электронный
ресурс] Режим доступа:
https://books.google.ru/books?id=NGlSs8YFs3EC&pg=PA77&redir_esc=y#v=onepage&q&f=false,
(дата обращения 16.05.2021)