МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГТУ»)

Факультет информационных технологий и компьютерной безопасности
(факультет)

Кафедра Систем автоматизированного проектирования и информационных систем

КУРСОВОЙ ПРОЕКТ

по дисциплине Базы данных

тема Проектирование баз данных для конкретных предметных областей

Расчетно-пояснительная записка

Разработал студент _____ Д.В. Тюленев
Подпись, дата Инициалы, фамилия

Руководитель _____ Д.В. Иванов
Подпись, дата Инициалы, фамилия

Члены комиссии _____
Подпись, дата Инициалы, фамилия

_____
Подпись, дата Инициалы, фамилия

Нормоконтролер _____
Подпись, дата Инициалы, фамилия

Защищена _____ Оценка _____
дата

2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГТУ»)

Кафедра Систем автоматизированного проектирования и информационных систем

ЗАДАНИЕ
на курсовой проект

по дисциплине Базы данных

Тема работы Проектирование баз данных для конкретных предметных областей

Студент группы ИСТ-214 Тюленев Данил Вячеславович
Фамилия, имя, отчество

Вариант 22. Разработать информационную систему «Записная книжка»

Технические условия процессор Intel® Core™ i3-8145U CPU @ 2,10 ГГц, операционная система Windows 10, ОЗУ 8192 МБ

Содержание и объем проекта (графические работы, расчеты и прочее):
анализ и описание предметной области (10 страницы); проектирование базы данных (8 страниц); разработка и реализация клиентского приложения (15 страниц); 24 рисунков, 1 приложение.

Сроки выполнения этапов анализ и описание предметной области (07.02 – 12.03); проектирование базы данных (14.03 – 16.04); разработка клиентского приложения и реализация программы (18.04 – 21.05); оформление расчтено-пояснительной записки (23.05 – 28.05)

Срок защиты курсового проекта 07.06.2023

Руководитель Д.В. Иванов
Подпись, дата Инициалы, фамилия

Задание принял студент 02.02.2023 Д.В. Тюленев
Подпись, дата Инициалы, фамилия

Замечания руководителя

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

ЭВМ помогает человеку реализовать трудоёмкие работы в различных сферах его деятельности. Универсальность различных научных приложений заключается в многократности их использования с различными входными данными для решения задачи в заданной области. Каждый день мы получаем большой объем информации. Если бы человек не умел обрабатывать полученную информацию, то наша голова была бы забита огромным количеством бесполезной информации, и любые мыслительные процессы были бы крайне затруднительны.

Электронные записные книжки довольно давно появились на рынке. Они обладают набором функций, позволяющим избавиться от ведения бумажных записных книжек. Основными достоинствами первых являются количество возможных записей, удобство навигации и поиска, парольная защита.

Поздравительные письма в день рождения клиента – не просто формальность, это отличный ход коммуникации, который напомнит ваших знакомых. А чтобы не тратить каждый раз много времени на поиск именинников и составления текста — такие рассылки лучше автоматизировать. Это уменьшит риск ошибиться, а также сэкономит время.

Данный курсовой проект посвящён разработке автоматизированной информационной системы для хранения анкетных данных о знакомых с функциональностью уведомления пользователя о приближающихся событиях. Грамотное организованное хранение документов, содержащих персональные данные, залог того, что конфиденциальная информация о клиентах и в целом деятельности организации не попадёт в руки посторонних лиц.

В курсовом проекте будет проанализирована поставленная задача на основе теоретического материала, а также программного и аппаратного обеспечения. В результате исследования будет представлены результаты проектирования информационной системы и базы данных для сбора и обработки персональных данных пользователей решения. Также будет предоставлено клиентское решение для которого в приложении приведём листинг программы.

1 Анализ и описание информационной системы «Зарплата»

1.1 Особенности предметной области

Предметная область – это часть реального мира, которая подлежит изучению с целью автоматизации организации управления. Предметной областью информационной системы является совокупность объектов, свойства которых и отношения между которыми представляют интерес для пользователей ИС.

Записная книга - платформа, позволяющая в удобном виде хранить телефоны, адреса электронной почты, дни рождения и прочие личные данные своих друзей, знакомых и коллег по работе. Список контактов можно разбивать на группы и сортировать необходимых способом. Объектом исследования является создание электронной записной книжки. Предметом исследования является процесс систематизации личной информации. В работе использована различная методологическая база, в том числе Государственные Стандарты по информационным технологиям, учебная литература базам данных, разработке программного обеспечения и менеджменту.

Также разрабатываемая система будет в состоянии генерировать автоматические поздравления для участников списка, с использованием триггерной рассылки. Триггерная рассылка – это письмо или несколько писем, которые отправляются в ответ на определённое действие или бездействие пользователя. Зачастую данный вид взаимодействия применяется в интернет магазинах. Приведём простой пример: потенциальный покупатель добавил товар в корзину, но не завершил покупку (эта ситуация называется «брошенная корзина») и покинул сайт. Спустя некоторое время срабатывает триггер – и клиенту направляется e-mail с сообщением о том, что он забыл свои покупки в корзине. Таким образом, триггерная рассылка призывает его завершить покупку.

Для защиты анкетных данных пользователей системы необходимо проговорить про понятие персональных данных. Персональные данные – любая информация, относящаяся прямо или косвенно к определенному или определяемому лицу (п. 1 ст. 3 Федерального закона от 27 июля 2006 г. № 152-ФЗ

"О персональных данных"; далее – закон о персональных данных). Такое широкое толкование позволяет относить к персональным данным практически любую информацию о человеке: сведения о его ФИО, поле и возрасте, образовании, месте жительства, семейном положении и др. Помимо этого к персональным данным относится и изображение человека, с помощью которого можно установить его личность – например, фотография, видеозапись или портрет (разъяснения Роскомнадзора от 30 августа 2013 г. "Разъяснения по вопросам отнесения фото-, видеоизображений, дактилоскопических данных и иной информации к биометрическим персональным данным и особенностей их обработки").

Обработка персональных данных – любое действие с персональными данными, включая сбор, запись, систематизацию, накопление, хранение, уточнение (обновление, изменение), извлечение, использование, передачу (распространение, предоставление, доступ), обезличивание, блокирование, удаление, уничтожение персональных данных (п. 3 ст. 3 закона о персональных данных). Оператор – государственный орган, муниципальный орган, юридическое или физическое лицо, самостоятельно или совместно с другими лицами организующие и (или) осуществляющие обработку персональных данных, а также определяющие цели обработки персональных данных, состав персональных данных, подлежащих обработке, действия (операции), совершаемые с персональными данными;

Трансграничная передача персональных данных – передача персональных данных на территорию иностранного государства органу власти иностранного государства, иностранному физическому лицу или иностранному юридическому лицу (п. 11 ст. 3 закона о персональных данных).

Настоящим Федеральным законом регулируются отношения, связанные с обработкой персональных данных, осуществляемой федеральными органами государственной власти, органами государственной власти субъектов Российской Федерации, иными государственными органами, органами местного самоуправления, иными муниципальными органами, юридическими лицами и физическими лицами с использованием средств автоматизации, в том числе в

информационно-телекоммуникационных сетях, или без использования таких средств, если обработка персональных данных без использования таких средств соответствует характеру действий (операций), совершаемых с персональными данными с использованием средств автоматизации, то есть позволяет осуществлять в соответствии с заданным алгоритмом поиск персональных данных, зафиксированных на материальном носителе и содержащихся в картотеках или иных систематизированных собраниях персональных данных, и (или) доступ к таким персональным данным. (В редакции Федерального закона от°25.07.2011 261-ФЗ). Целью настоящего Федерального закона является обеспечение защиты прав и свобод человека и гражданина при обработке его персональных данных, в том числе защиты прав на неприкосновенность частной жизни, личную и семейную тайну.

Рынок программного обеспечения России довольно богат системами, позволяющими выполнить операции автоматизации хранения контактных данных и автоматизации поздравления с праздниками записанных людей. Рассмотрим аналогичное приложение разрабатываемого решения «iSender – Умный календарь». Приложение объединяет информацию о дне рождении друзей сразу из трёх мест: контакты на вашем смартфоне, друзья из ВКонтакте, друзья из Facebook. Синхронизация происходит на уровне грамотного процесса: если приложение увидело совпадающие имена, оно предложит вам объединить их, чтобы исключить повторения в календаре. Если есть необходимость, каждому контакту вы сможете задать отдельные параметры напоминания: для начала выберите время напоминания, активируйте автоматическое поздравление и выберите способ, как это сделать. Приложение может выслать теплые слова по почте, сообщениями ВКонтакте или Facebook, через iMessage или SMS, либо просто вспомнить вам обычным напоминанием на смартфоне или совсем не напоминать, если не хотите поздравлять человека. Можно заранее придумать красивый шаблон с приятными словами и задать его для рассылки несколькими контактами или всем.

Разрабатываемая автоматизированная информационная система «Записная книжка» предназначена для хранения контактной и анкетной информация о близких людях, а также необходима для автоматизации формирования и доставки поздравительной открытки на день рождения выбранным людям. Пользователь может добавлять в список контактов новые записи с возможностью импортировать часть информации из социальных сетей. Для удобной навигации по списку знакомых людей будет доступен функционал сортировки записей по разным категориям: дата рождения, ФИО, номер телефона и т.д. На основании анализа предметной области были выделены основные объекты информационной системы, представленные в таблице 1.

Таблица 1 – Информационные объекты.

Название объекта системы	Обозначение в базе данных	Описание
Контакт	Contact	Представляет первичную информацию о знакомом.
Электронная почта	EmailAddress	Содержит данные о электронных почтах пользователей.
Местоположение	Location	Хранит информацию о местах проживания пользователей.
Место работы	WorkPlace	Содержит информацию о рабочих местах и местах учёбы.
Увлечения	Hobby	Представляет информацию о увлечениях пользователей системы.

Объект системы «Контакт» является одним из главных внутри проектируемого решения; он содержит всю необходимую информацию о человеке, который был занесён в записную книгу: ФИО, дата рождения, хобби, место работы, место проживания, номер телефона и адрес электронной почты. С

его помощью будут составляться поздравления с праздниками к нужным пользователю людям.

1.2 Описание бизнес-процессов

Под проектированием автоматизированных информационных систем понимается процесс разработки технической документации, связанный с организацией системы получения и преобразования исходной информации в результатную, т.е. с организацией автоматизированной информационной технологии.

Нотации – графические модели, которые используются, чтобы фиксировать бизнес-процессы, анализировать их и оптимизировать. В настоящее время для описания бизнес-процессов существует множество методологий, к примеру, такие как UML, IDEF0, DFD, и инструментальных средств для их реализации. В структурном и объектно-ориентированном анализе используются средства, моделирующие в форме диаграмм определённого вида деловые процессы и отношения между данными в системе.

— IDEF – применяется для построения функциональной/концептуальной модели системы, отображают функции/процедуры, а также потоки информации и материальных объектов;

— DFD – отображает потоки данных между системами, базами данных; ключевыми элементами являются входные/выходные данные, системы, точки хранения и сбора данных.

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

входящие – (вводные) ставят определённую задачу; исходящие – выводящие результат деятельности; управляющие (сверху вниз) – механизмы управления (положения, инструкции и пр); механизмы (снизу вверх) – что используется для того, чтобы произвести необходимую работу.

Модель IDEF0 может содержать несколько видов диаграмм, одними из которых являются контекстная диаграмма и диаграмма декомпозиции. Контекстная диаграмма является вершиной древовидной структуры диаграмм и представляет собой самое общее описание системы и её взаимодействия с внешней средой.

Для составления контекстной диаграммы информационной системы «Записная книжка» необходимо выделить механизмы, выполняющие работу, средства управления, влияющие на выполнение работы, информацию, которая подаётся на вход системы и используются или преобразуются работой для получения результата (выхода), материал, получающийся в результате функционирования системы(выход).

В качестве входных данных подаются следующие элементы: сведения о добавляемом пользователе (анкетные данные: ФИО, телефон, email и т.д); временной интервал для проверки праздников (отрезок времени, на котором будут проверяться грядущие события); данные для поиска необходимого контакта (фильтры). Системы имеет возможность возвращать отчёты (документы или таблицы), уведомления о событиях, данные о контактах, а также отправлять формы с поздравлениями. Выделяются следующие средства управления: Федеральный закон «О персональных данных» от 27.07.2006 N 152-ФЗ; пользовательское соглашение на рассылку смс и email сообщений; а также ГОСТ для поздравительных шаблонов. В качестве механизмов системы применяются пользователи системы и SMTP сервер для взаимодействия с функцией отправки электронных писем.

На основании выделенных компонентов диаграммы была составлена контекстная диаграмма, представленная на рисунке 1.

Рисунок 1 – Контекстная диаграмма IDEF0

На рисунке 1 можно увидеть, что диаграмма содержит все вышеуказанные выделенные компоненты.

После описания системы в целом проводится разбиение её на крупные фрагменты. Этот процесс называется функциональной декомпозицией, а диаграммы, которые описывают каждый фрагмент и взаимодействие фрагментов, называются диаграммами декомпозиции. Декомпозиция позволяет постепенно и структурированно представлять модель системы в виде иерархической структуры отдельных диаграмм, что делает её менее перегруженной и легко усваиваемой.

Для составления диаграммы декомпозиции необходимо выделить процессы, которые выполняются в системе, и их взаимосвязь между собой:

1. Добавление нового контакта – заполнение специальной формы, предполагающая получение контактной и анкетной информации о пользователе и его знакомых.

2. Проверка данных для сохранения на диск – на этом этапе производится проверка валидности введённых внутрь формы данных

3. Чтение данных из списка с применением фильтра – позволяет просматривать записи с контактной информацией с использованием фильтров и с возможностью сортировки по необходимым аттрибутам.

4. Оформление напоминания о праздниках – на данном этапе производится поиск и обработка информации о пользователях, дни рождения которых попадают под указанный временной интервал.

5. Формирование отчёта в виде документа – данный этап предполагает формирование определённого вида документа (текстовый или табличный) на основе выбранных данных из списка контакта.

6. Формирование поздравления с использованием конструктора – производится создание поздравления, с возможностью редактирования через специальный конструктор формы, а также указать за время отправки.

7. Отправка электронного письма по почте – на данном этапе происходит отправка сформированного письма по адресу найденного в процессе обработки контакта

8. Уведомление пользователя о событии – производится системное уведомление о грядущем событии.

На основании выделенных процессов диаграммы была составлена диаграмма декомпозиции, представленная на рисунке 2.

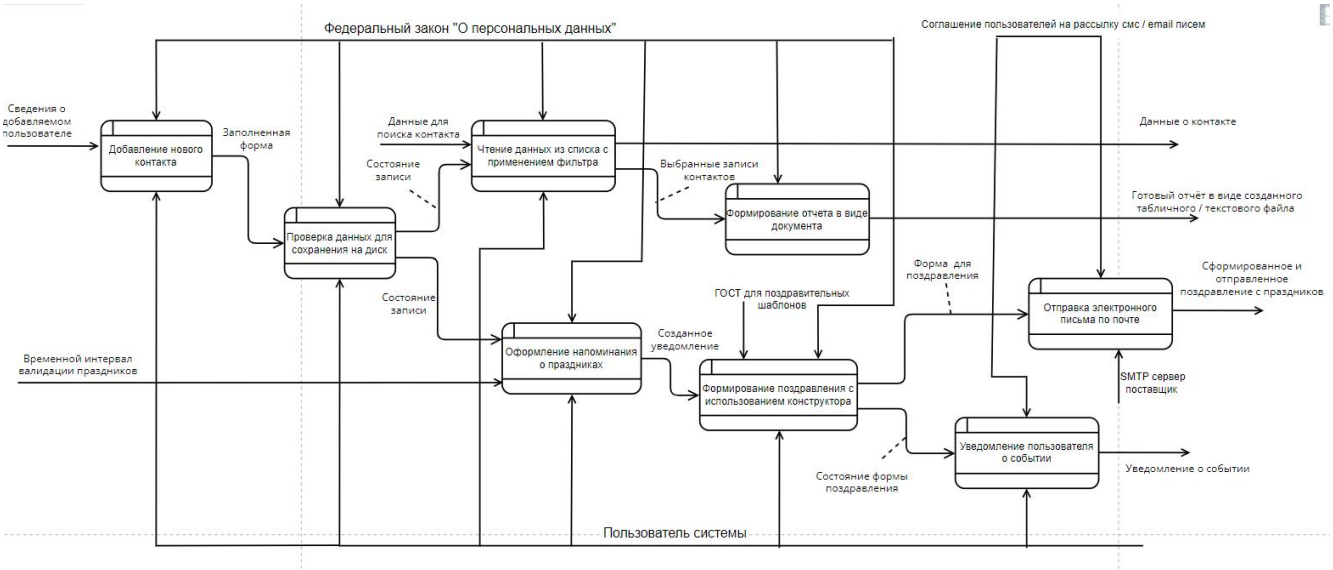


Рисунок 2 – Диаграмма декомпозиции IDEF0

Диаграммы потоков данных (Data flow diagramming, DFD) используются для описания документооборота и обработки информации. Подобно IDEF0, DFD представляет систему как сеть связанных между собой работ.

Для составления диаграммы DFD необходимо выделить процессы, которые выполняются в системе, и их взаимосвязь между собой: На основании выделенных процессов диаграммы была составлена диаграмма DFD, представленная на рисунке 3.

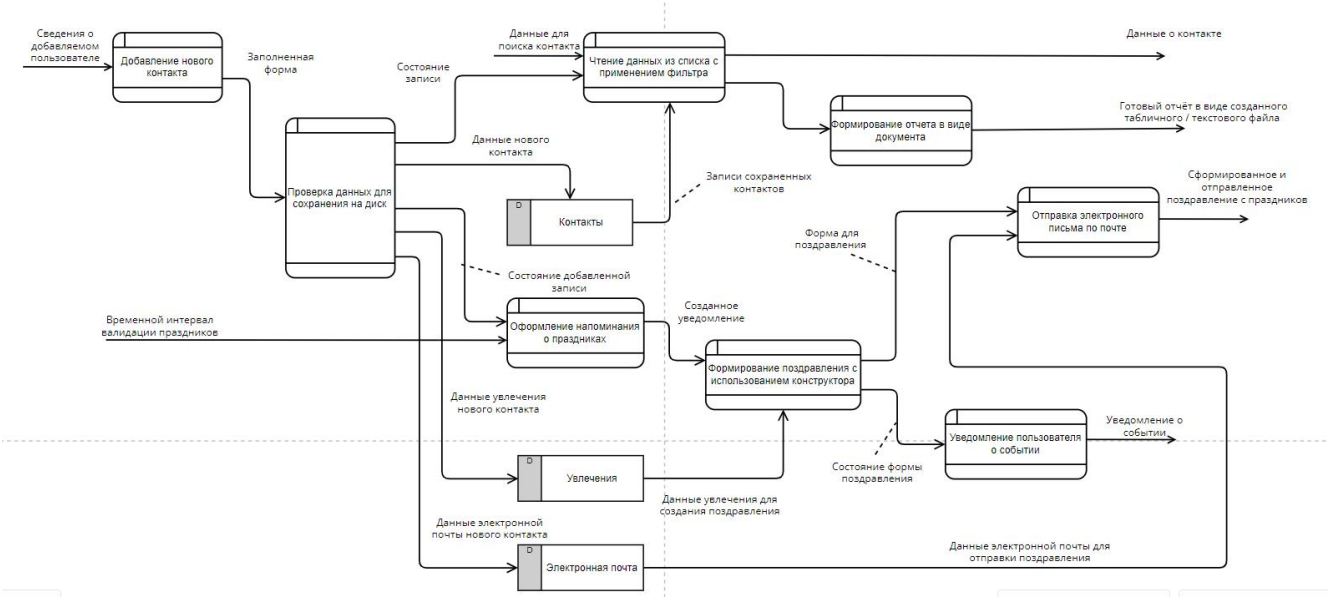


Рисунок 3 – Диаграмма потоков данных DFD

Транзакции в базе данных разрабатываемой информационной системы играют ключевую роль в обеспечении надёжности данных и предотвращении ошибок при выполнении операций в базе данных.

Транзакция – это логически связанная последовательность операций, которые выполняются в базе данных как единое целое. Транзакция должна быть успешно завершена или отменена целиком, чтобы сохранить целостность данных. Примеры транзакций в базе данных могут включать следующие группы действий: получение данных от пользователя, проверка и регистрация их в системе; чтение данных о контакте, формирование поздравления с последующей отправкой на электронную почту и т.д.

1.3 Постановка цели и задач системы

Целью курсового проектирования является закрепление теоретических знаний, а также навыков проектирования баз данных, полученных при изучении курса «Управление данными».

Цель работы заключается в разработке информационной системы «Записная книжка» и проектировании для неё базы данных, позволяющей автоматизировать работу по поиску необходимой контактной информации, получению уведомлений о приближающихся праздниках близких людей и упростить процесс оформления поздравления с применением конструктора и шаблонов. Для разработки программного обеспечения были поставлены следующие задачи:

1. Возможность вносить изменения в персональные данные контактов, и базу данных в целом;
2. Формирование результирующего документа с отобранными контактными записями;
3. Реализация интуитивно понятного пользователю интерфейса;
4. Обеспечение ввода данных в БД;
5. Обеспечение проверки корректности вносимых данных в БД;
6. Обеспечение функционирования системы как автоматизированному процессу поздравления с праздниками.

Таким образом, поставленная цель и выдвинутые задачи представляют собой явный список требований, который должен быть реализован в данном курсовом проекте.

2 Проектирование баз данных для системы «Записная книжка»

2.1 Особенности предметной области

Концептуальное проектирование начинается с анализа предметной области, включает анализ концептуальных требований и информационных потребностей, выявление информационных объектов и связей между ними, построение концептуальной модели (схемы) данных.

Концептуальная модель — один из видов моделей данных, формируемых на ранней стадии разработки баз данных; которая отображает сущности, их атрибуты и связи между ними, а также описывает основные бизнес-процессы и правила работы с данными. Эта модель не зависит от конкретной технологии хранения данных и служит основой для разработки логической и физической моделей данных.

Формирование концептуальной модели данных заключается в подготовке модели данных в рамках конкретного объекта.

Для проектирования концептуальной модели необходимо придерживаться следующей последовательности действий: определение основных типов сущностей, присутствующих в представлении предметной области приложения; документирование выделенных типов сущностей.

1. Определение основных типов сущностей, присутствующих в представлении предметной области приложения. Документирование выделенных типов сущностей.

В рамках сформулированной цели и решения конкретных задач можно выделить следующие типы сущностей:

— Контакт. Представляет собой главную сущность, которая отвечает за определение состояний отдельных записей в контактном списке: имя, фамилия, дата рождения, пол, электронный адрес, семейное положение, хобби и т.д.

— Местоположение. Для каждого пользователя и контакта, необходимо отображать информацию о месте, где он находится (поживает), однако данные о местоположении не обязательны для формирования контакта.

— Город. Сущность, которая хранит информацию о доступных внутри системы населённых пунктов, представленную в виде пары: страна, город (посёлок).

— Работник. Представляет дополнительную характеристику контакта, которая отвечает за информацию о место работы (учёбы): название учреждения или компании, статус пребывания внутри.

— Должность. Описывает доступные внутри системы, возможные должности контакта на рабочем месте.

— Человеческие качества. Сущность, которая хранит доступные в системе человеческие характеристики, которые можно предоставить для конкретного контакта.

— Хобби. Описывает различные увлечения, которые можно ассоциировать с контактом: тип хобби и название.

— Пол. Сущность, идентифицирующая пол человека (контакта).

— Друзья. Представляет собой сущность, которая определяет связь между двумя контактами, регламентирующая дружественные отношения. В качестве атрибутов сущности можно выделить следующее: время определения отношения, а также характер знакомства.

— Сообщение. Сущность, определяющая составные части диалога двух контактов, между которыми определены дружественные отношения. Сущность хранит следующие атрибуты: тело сообщения, время отправления, контакт отправителя и ссылка на отношение, соединяющее два контакта.

2. Определение первичного ключа для каждого типа сущности.

Каждый тип сущности внутри модели имеет первичный ключ, название ключа составляется из названия таблицы, поэтому обозначения всех первичных ключей в БД уникальны.

3. Проверка модели на избыточность, т.е. внимательная проверка связей «один к одному» и удаление избыточных связей.

4. Определение важнейших типов связей, существующих между сущностями, выделенными на предыдущем этапе. Применение ER-моделирования для наглядного отображения сущностей и связей. Выявление дефектов разветвления и разрывов. Проверка того, участвует ли каждая сущность, по меньшей мере, в одной связи.

После выполнения всех вышеописанных этапов, происходит формирование концептуальной модели, которая представлена на рисунке 4.

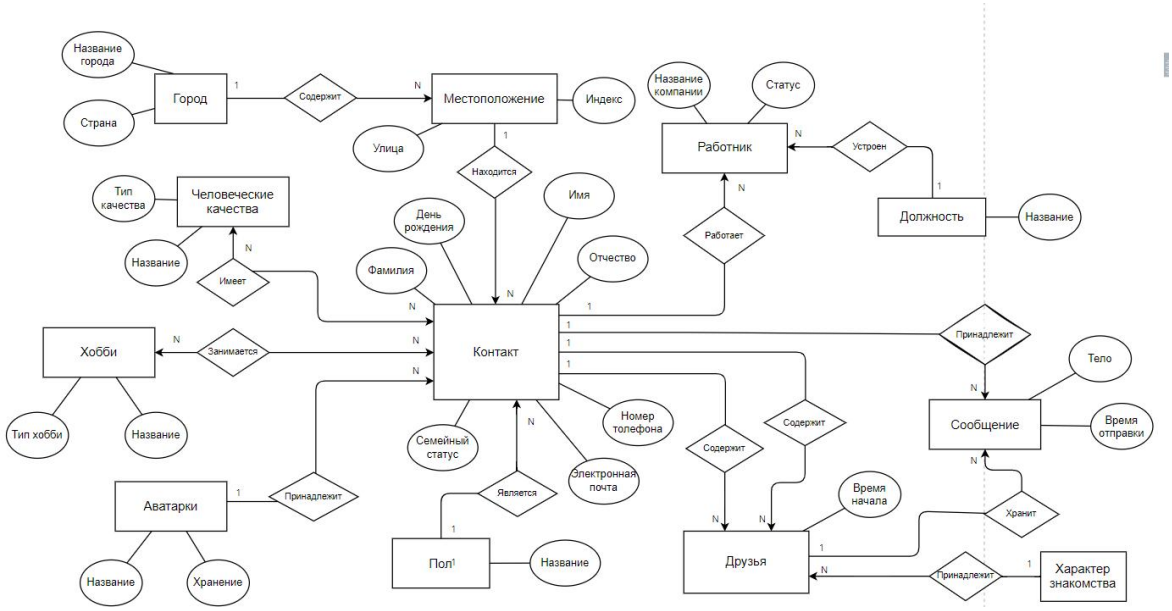


Рисунок 4 – Концептуальная модель данных

2.2 Построение логической модели данных

Логическая модель данных используется для создания схемы базы данных и служит основой для проектирования физической модели данных. Она включает в себя определения сущностей, атрибутов и связей между ними. Для проектирования логической модели данных необходимо убедится что все отношения между сущности корректны, для этого воспользуемся методом нормализации.

Нормализация данных - процесс организации и структурирования данных в базе данных таким образом, чтобы избежать дублирования информации и снизить вероятность ошибок при обработке данных. Она включает в себя разделение

данных на логически связанные таблицы и определение связей между таблицами. Целью нормализации данных является минимизация избыточности данных, упрощении структуры базы данных, а также предотвращении аномальных ситуаций и потерей данных при изменении, вставке или удалении строк из таблицы.

Внесём основные положения для определения записной книжки. В данном виде данные находятся в «Ненормализированной нормальной форме» (UNF) так, чтобы он отвечал базовым принципам реляционной теории. На рисунке 5 представлена ненормализованная таблица.

Контакт								
Ф. И. О.	ID Пол	Телефон	Адрес	Дата рождения	Место работы / учебы	Данные знакомства	Хобби	
Иванов Иван Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23, Липецк, Россия, 398059	03.12.1996	ОАО "Лепесток", Директор, 30000 (руб)	Коллега, 03.03.2020	Музыка, Пассивный	Добро
Иванов Иван Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23, Липецк, Россия, 398059	03.12.1996	ОАО "Лепесток", Директор, 30000 (руб)	Коллега, 03.03.2020	Литература, Пассивный	Добро
Иванов Иван Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23, Липецк, Россия, 398059	03.12.1996	ОАО "Лепесток", Директор, 30000 (руб)	Коллега, 03.03.2020	Животные, Активный	Добро
Иванов Иван Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23, Липецк, Россия, 398059	03.12.1996	ОАО "Лепесток", Директор, 30000 (руб)	Коллега, 03.03.2020	Музыка, Активный	Завис
Иванов Иван Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23, Липецк, Россия, 398059	03.12.1996	ОАО "Лепесток", Директор, 30000 (руб)	Коллега, 03.03.2020	Литература, Пассивный	Завис
Иванов Иван Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23, Липецк, Россия, 398059	03.12.1996	ОАО "Лепесток", Директор, 30000 (руб)	Коллега, 03.03.2020	Животные, Активный	Завис
Сергеев Сергей Сергеевич	Мужской	+7(920)-521-92-14	ул. Космонавтов д.21, Воронеж, Россия, 394005	10.06.2000	ВГТУ, Студент, 2500 (руб)	Одногруппник, 10.12.2015	Аниме, Активный	Завис
Сергеев Сергей Сергеевич	Мужской	+7(920)-521-92-14	ул. Космонавтов д.21, Воронеж, Россия, 394005	10.06.2000	ВГТУ, Студент, 2500 (руб)	Одногруппник, 10.12.2015	Литература, Пассивный	Завис
Сергеев Сергей Сергеевич	Мужской	+7(920)-521-92-14	ул. Космонавтов д.21, Воронеж, Россия, 394005	10.06.2000	ВГТУ, Студент, 2500 (руб)	Одногруппник, 10.12.2015	Аниме, Активный	Щедр
Сергеев Сергей Сергеевич	Мужской	+7(920)-521-92-14	ул. Космонавтов д.21, Воронеж, Россия, 394005	10.06.2000	ВГТУ, Студент, 2500 (руб)	Одногруппник, 10.12.2015	Литература, Пассивный	Щедр
Сидоров Николай Петрович	Мужской	+7(906)-602-42-05	пр. Победы д. 5, Москва, Россия, 125009	23.01.1980	Сбербанк, Менеджер, 18000 (руб)	Коллега, 23.05.2012	Фильмы, Пассивный	Добро
Власова Анна Михайловна	Женский	+7(950)-150-12-94	ул. Шишкова д. 85, Воронеж, Россия, 394036	17.09.2002	Вкусно и точка, Менеджер, 18000 (руб)	Друг, 17.08.2021	Рукоделие, Активный	Верно

Рисунок 5 – Ненормализованная таблица

После анализа таблицы, необходимо устранить избыточность данных, сделав преобразование в следующую форму – «Первая нормальная форма» (1NF). Чтобы реализовать переход необходимо произвести разбиение отдельных свойств. Таблица в первой нормальной форме продемонстрирована на рисунке 6.

Контакт														
Фамилия	Имя	Отчество	ID Пол	Телефон	Адрес	Город	Страна	Индекс	Дата рождения	Место работы / учебы	Должность		Дата знакомства	Тип знакомства	Хо
Иванов	Иван	Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23	Липецк	Россия	398059	03.12.1996	ОАО "Лепесток"	Директор		03.03.2020	Коллега	Музы
Иванов	Иван	Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23	Липецк	Россия	398059	03.12.1996	ОАО "Лепесток"	Директор		03.03.2020	Коллега	Литер
Иванов	Иван	Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23	Липецк	Россия	398059	03.12.1996	ОАО "Лепесток"	Директор		03.03.2020	Коллега	Живо
Иванов	Иван	Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23	Липецк	Россия	398059	03.12.1996	ОАО "Лепесток"	Директор		03.03.2020	Коллега	Музы
Иванов	Иван	Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23	Липецк	Россия	398059	03.12.1996	ОАО "Лепесток"	Директор		03.03.2020	Коллега	Литер
Иванов	Иван	Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23	Липецк	Россия	398059	03.12.1996	ОАО "Лепесток"	Директор		03.03.2020	Коллега	Живо
Сергеев	Сергей	Сергеевич	Мужской	+7(920)-521-92-14	ул. Космонавтов д.21	Воронеж	Россия	394005	10.06.2000	ВГТУ	Студент		10.12.2015	Одногруппник	Аним
Сергеев	Сергей	Сергеевич	Мужской	+7(920)-521-92-14	ул. Космонавтов д.21	Воронеж	Россия	394005	10.06.2000	ВГТУ	Студент		10.12.2015	Одногруппник	Литер
Сергеев	Сергей	Сергеевич	Мужской	+7(920)-521-92-14	ул. Космонавтов д.21	Воронеж	Россия	394005	10.06.2000	ВГТУ	Студент		10.12.2015	Одногруппник	Аним
Сергеев	Сергей	Сергеевич	Мужской	+7(920)-521-92-14	ул. Космонавтов д.21	Воронеж	Россия	394005	10.06.2000	ВГТУ	Студент		10.12.2015	Одногруппник	Литер
Сидоров	Николай	Петрович	Мужской	+7(906)-602-42-05	пр. Победы д. 5	Москва	Россия	125009	23.01.1980	Сбербанк	Менеджер		23.05.2012	Коллега	Филь
Власова	Анна	Михайловна	Женский	+7(950)-150-12-94	ул. Шишкова д. 85	Воронеж	Россия	394036	17.09.2002	Вкусно и точка	Менеджер		17.08.2021	Друг	Руко

Рисунок 6 – Первая нормальная форма

В итоге должна получиться таблица с данными, каждая ячейка которой хранит атомарное значение (не составное), тем самым таблица будет находится в состоянии «Первой нормальной формы».

После приведения в первую нормальную форму, наблюдаем лишнюю избыточность, поэтому для её устранения необходимо привести модель во вторую нормальную форму.

Для трансформирования таблицы необходимо выявить группу атрибутов, которые будут представлять собой ключевыми атрибутами. Из представленной структуры видно: чтобы уникально идентифицировать отдельную запись в качестве ключа необходимо выбрать группу атрибутов: «Фамилия», «Имя», «Отчество», «Хобби», «Качество». Проанализировав данный ключ, можно понять, что условие «Второй нормальной формы» не соблюдается: все неключевые столбцы таблицы должны зависеть от полного ключа. Чтобы привести к необходимой форме, стоит произвести декомпозицию (разбить одну таблицу на несколько). В качестве новых отношений выделим «Хобби», «Контакт», «Качества человека», «Связь контакта и хобби», «Связь контакта и качества»

Напишем отдельные таблицы для хобби и человеческих качеств. На рисунке 7 продемонстрирована вторая нормальная форма на примере таблиц «Контакты», «Хобби» и «Человеческие качества».

Контакт

ID	Фамилия	Имя	Отчество	ID Пол	Телефон	Адрес	Город	Страна	Индекс	Дата рождения	Место работы / учебы	Должность		Дата знакомства	Тип знакомства
1	Иванов	Иван	Иванович	Мужской	+7(952)-234-63-32	ул. Гагарина д. 23	Липецк	Россия	398059	03.12.1996	ОАО "Лепесток"	Директор		03.03.2020	Коллега
2	Сергеев	Сергей	Сергеевич	Мужской	+7(920)-521-92-14	ул. Космонавтов д.21	Воронеж	Россия	394005	10.06.2000	ВГТУ	Студент		10.12.2015	Одногруппник
3	Сидоров	Николай	Петрович	Мужской	+7(906)-602-42-05	пр. Победы д. 5	Москва	Россия	125009	23.01.1980	Сбербанк	Менеджер		23.05.2012	Коллега
4	Власова	Анна	Михайловна	Женский	+7(950)-150-12-94	ул. Шишкова д. 85	Воронеж	Россия	394036	17.09.2002	Вкусно и точка	Менеджер		17.08.2021	Друг

Хобби

ID	Хобби	Тип
1	Музыка	Пассивный
2	Спорт и танцы	Активный
3	Литература	Пассивный
4	Аниме	Активный
5	Животные	Активный
6	Рукоделие	Активный
7	Фильмы	Пассивный

Соединение хобби и контакта

ID Контакта	ID Хобби
1	1
1	3
1	5
2	4
2	3
3	7
4	6

Качества человека

ID	Качество	Тип
1	Доброта	Положительные
2	Зависть	Отрицательное
3	Щедрость	Положительные
4	Юмор	Положительные
5	Эгоизм	Отрицательное
6	Верность	Положительные

Связь качество и контакт

ID Контакт	ID Качество
1	1
1	2
2	3
2	2
3	1
4	6

Рисунок 7 – Вторая нормальная форма

Для выведение из текущей нормальной формы в третью, необходимо устранить транзитивную зависимость неключевых аттрибутов. Для этого необходимо рассмотреть структуру сущности «Контакт». Значения атрибута

«Должность» функционально зависят от атрибута «Место работы», поэтому необходимо выделить новое отдельное отношение и оставить на неё ссылку в главной. Тоже самое необходимо сделать в строну атрибутов: «Дата знакомства» и «Тип знакомства»; «Адрес», «Город», «Страна» и т. д. Модель, приведённая в третью нормальную форму продемонстрирована на рисунке 8.

Контакт

ID	Фамилия	Имя	Отчество	ID Пол	Телефон	ID Адрес	Дата рождения	ID Место работы	ID Характер знакомства
1	Иванов	Иван	Иванович	1	+7(952)-234-63-32	1	03.12.1996	1	1
2	Сергеев	Сергей	Сергеевич	1	+7(920)-521-92-14	2	10.06.2000	2	2
3	Сидоров	Николай	Петрович	1	+7(906)-602-42-05	3	23.01.1980	3	3
4	Власова	Анна	Михайловна	2	+7(950)-150-12-94	4	17.09.2002	4	4

Местоположение

ID	Улица	ID Город	ID Страна	Индекс
1	ул. Гагарина д. 23	4	1	398059
2	ул. Космонавтов д.21	2	1	394005
3	пр. Победы д. 5	3	1	125009
4	ул. Шишкова д. 85	2	1	394036

Хобби

ID	Хобби	Тип
1	Музыка	Пассивный
2	Спорт и танцы	Активный
3	Литература	Пассивный
4	Аниме	Активный
5	Животные	Активный
6	Рукоделие	Активный
7	Фильмы	Пассивный

Соединение хобби и контакта

ID Контакта	ID Хобби
1	1
1	3
1	5
2	4
2	3
3	7
4	6

Пол

ID	Пол
1	Мужской
2	Женский

Место работы

ID	Место работы/ учебы	Должность
1	ОАО "Лепесток"	1
2	ВГТУ	2
3	Сбербанк	3
4	Вкусно и точка	3

Качества человека

ID	Качество	Тип
1	Доброта	Положительные
2	Зависть	Отрицательное
3	Щедрость	Положительные
4	Юмор	Положительные
5	Эгоизм	Отрицательное
6	Верность	Положительные

Связь качество и контакт

ID Контакт	ID Качество
1	1
1	2
2	3
2	2
3	1
4	6

Характер знакомства

ID	Дата знакомства	ID Тип знакомства
1	03.03.2020	2
2	10.12.2015	3
3	23.05.2012	2
4	17.08.2021	4

Рисунок 8 – Третья нормальная форма

На основе вышеприведённых таблиц смоделируем логическую модель для отображения связей между сущностями. Логическая модель представлена на рисунке 9.

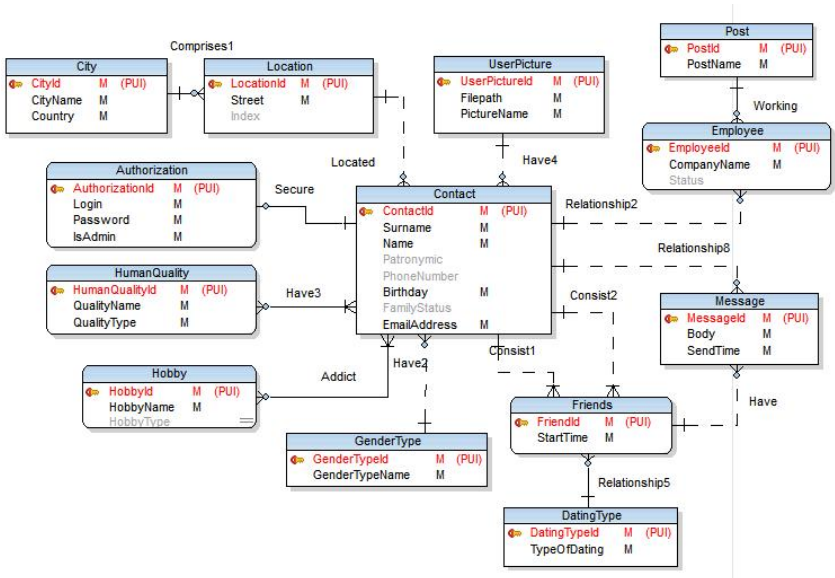


Рисунок 9 – Логическая модель данных

2.3 Построение физической модели данных

Построение физической модели данных - это процесс создания физической структуры базы данных, которая определяет, как данные будут храниться на устройствах хранения данных и как они будут доступны для использования приложениями.

Физическая модель БД определяет способ размещения данных на носителях (устройствах внешней памяти), а также способ эффективного доступа к ним. Поскольку СУБД функционирует в составе и под управлением операционной системы, то организация хранения данных и доступа к ним зависит от принципов и методов управления данными операционной системы. Способ хранения БД определяется механизмами СУБД автоматически по умолчанию на основе спецификаций концептуальной схемы БД, и внутренняя схема в явном виде в таких системах не используется. Внешние схемы БД обычно конструируются на стадии разработки приложений.

В целях ускорения обработки базы данных необходимо продумать, какие поля базы данных будут проиндексированы, и далее обосновать свой выбор индексированных полей. Необходимо обеспечить целостность базы данных (все ограничения).

Подготовим схему реляционной базы данных, которая может быть реализована в целевой СУБД на основе ранее описанной логической модели. Используя CASE-средство Toad Data Modeler мы можем конвертировать логическую модель в физическую, что продемонстрировано на рисунке 10.

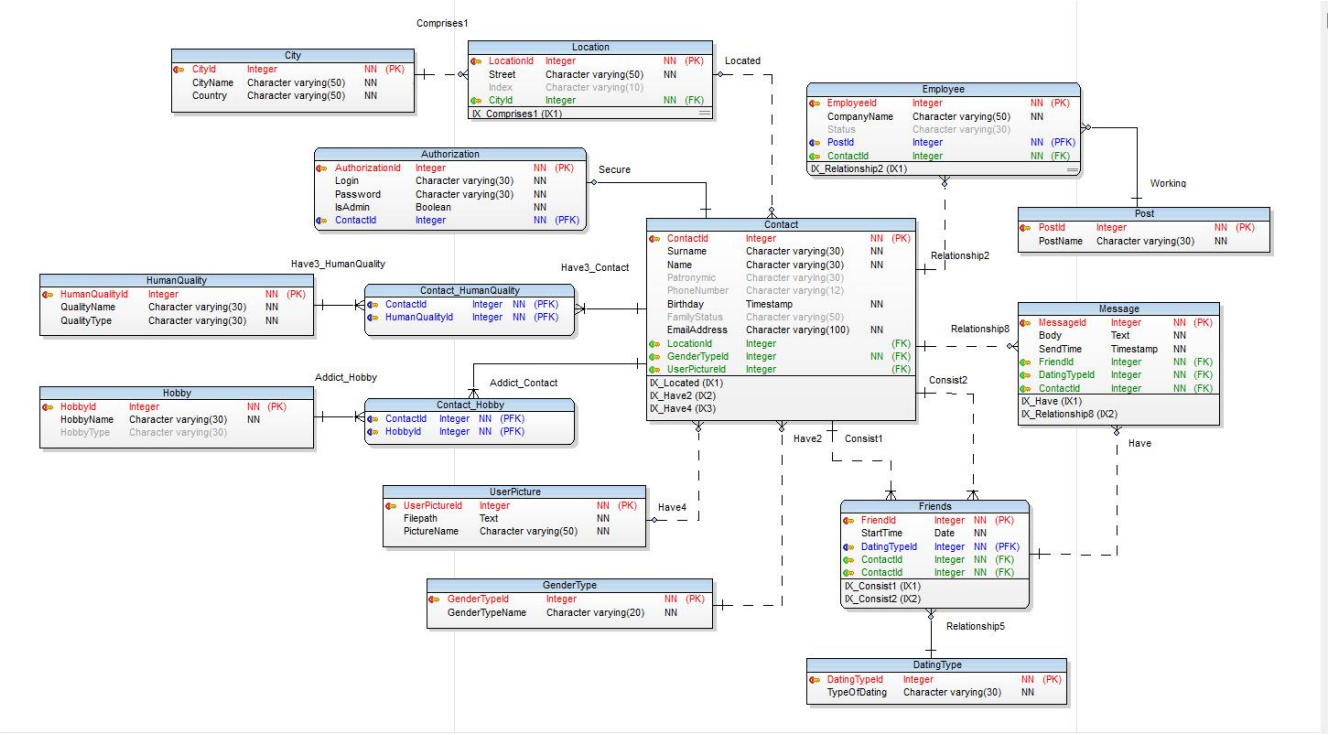


Рисунок 10 – Физическая модель данных

3 Разработка и реализация клиентского приложения записной книжки

3.1 Выбор средства реализации

Выбор средства реализации проекта является ключевой задачей в процессе разработки и реализации проекта. Корректный выбор средства позволяет повысить эффективность работы, улучшить качество конечного продукта и снизить затраты на реализацию проекта. Средства разработки программного обеспечения – совокупность приёмов, методов, методик, а также набор инструментальных программ (компиляторы, прикладные/системные библиотеки и т.д.), используемых разработчиком для создания программного кода Программы, отвечающего заданным требованиям.

ASP.NET Core 7 и Entity Framework Core 7 – это современные технологии, используемые для разработки веб-приложений, которые могут работать на любой операционной системе и платформе. Они предоставляют широкий набор функций и инструментов, которые обеспечивают безопасность, масштабируемость, производительность и простоту в использовании.

Язык программирования – это основной инструмент для разработки приложения на платформе ASP.NET Core 7 и Entity Framework Core 7. В качестве языка программирования предлагается использовать C#, который является одним из стандартных языков для разработки приложений на платформе .NET. C# позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в .NET. При выполнении программы C# сборка загружается в среду CLR. Среда CLR выполняет JIT-компиляцию из кода на языке IL в инструкции машинного языка. Среда CLR также выполняет другие операции, например, автоматическую сборку мусора, обработку исключений и управление ресурсами.

В качестве фреймворка для разработки веб-приложения предлагается использовать ASP.NET Core 7 – это платформа для создания веб-приложений, предназначенная для работы как на Windows, так и на Linux и macOS. ASP.NET Core 7 основан на современных технологиях, таких как .NET 6 и C# 10, и

включает в себя новые функции и улучшения по сравнению с предыдущими версиями. Некоторые из этих функций и улучшений включают: поддержка HTTP/3 и gRPC, улучшенная поддержка WebSocket и SignalR, интерниционализация и локализация, улучшенная обработка ошибок и журналирование, улучшенная поддержка Docker и Kubernetes

ASP.NET Core 7 также поддерживает различные типы веб-приложений, включая веб-приложения MVC, веб-приложения API, одностраничные приложения, веб-приложения Blazor и многие другие. Он также интегрируется с другими технологиями и инструментами, такими как Entity Framework, Azure, Visual Studio, и т.д., что делает его ещё более гибким и удобным для разработки веб-приложений. В целом, ASP.NET Core 7 - это мощный и гибкий фреймворк для разработки веб-приложений, который использует современные технологии и предоставляет множество возможностей для создания высокопроизводительных, масштабируемых и безопасных веб-приложений. Текущую архитектуру платформы ASP.NET Core можно выразить следующим образом:

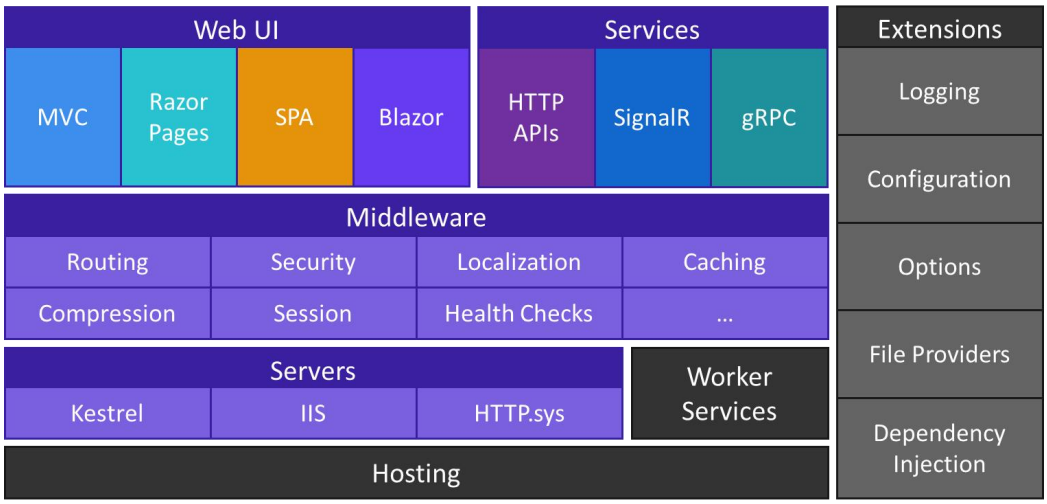


Рисунок 11 – платформа ASP.NET Core

В качестве СУБД для разработки веб-приложения на платформе ASP.NET Core 7 и Entity Framework Core 7 предлагается использовать PostgreSQL – свободная объектно-реляционная система управления базами данных, которая поддерживает большое количество типов данных и имеет множество

инструментов для работы с SQL-запросами и является одним из самых популярных решений для хранения и управления данными в веб-приложениях.

Для взаимодействия с базой данных используется ORM (Object-Relational Mapping) технология, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». В качестве ORM для работы с базой данных предлагается использовать Entity Framework Core 7 (EF7)– это ORM-фреймворк для работы с базами данных. Он предоставляет возможность работать с данными в виде объектов, а не непосредственно с SQL-запросами. Это делает работу с базами данных более удобной, быстрой и безопасной. Entity Framework 7 была разработана с учетом модульной архитектуры, что делает её более легкой и масштабируемой. EF7 включает инструменты для миграции данных, которые упрощают процесс обновления базы данных. Также в EF7 были внесены изменения в API для работы с запросами, что повышает взаимодействие и уменьшает время выполнения запросов.

Одним из ключевых изменений, внесённых в EF7, является поддержка множества баз данных, включая SQL Server, MySQL, PostgreSQL, SQLite и др. Это позволяет выбрать наиболее подходящую базу данных для конкретного проекта. EF7 является основой для создания современных приложений, использующих массу новых технологий, таких как .NET Core, ASP.NET Core, Xamarin и .NET Standard. Поэтому, используя EF7, разработчики могут создавать мощные и высокопроизводительные приложения для различных платформ.

В качестве среды разработки для разработки веб-приложения на платформе ASP.NET Core 7 и Entity Framework Core 7 предлагается использовать Microsoft Visual Studio 2022. – интегрированная среда разработки для создания приложений на платформе .NET. Она предоставляет множество инструментов и шаблонов для создания веб-приложений на ASP.NET Core 7 и инструменты для работы с базами данных.. Таким образом, для разработки клиентского приложения на платформе ASP.NET Core 7 и Entity Framework Core 7 рекомендуется использовать следующие средства:

- Язык программирования: C#
- Фреймворк: ASP.NET Core 7
- СУБД: PostgreSQL
- ORM: Entity Framework Core 7
- Среда разработки: Microsoft Visual Studio 2022

Эти средства обеспечивают широкий набор функций и инструментов, которые необходимы для разработки качественного и масштабируемого веб-приложения на платформе ASP.NET Core 7 и Entity Framework Core 7.

3.2 Архитектура программного обеспечения

Архитектура программного обеспечения (Application Architecture) - это структура и организация компонентов программного обеспечения, которые определяют как эти компоненты взаимодействуют друг с другом, работают вместе и решают поставленные задачи. Она включает в себя не только код программы, но и прикладные инфраструктуры, интеграцию с другими приложениями и системами, а также инфраструктуру управления данными и конфигурации.

Архитектура программного обеспечения позволяет разработчикам создавать высококачественное программное обеспечение, безопасное, надёжное, легко поддерживаемое и масштабируемое. Она способствует повторному использованию кода и ускоряет разработку и реализацию программы. Кроме того, архитектура программного обеспечения играет ключевую роль в создании интерфейсов для пользователя и обеспечении удобной работы с программным обеспечением.

Для разработки данной работы было утверждено решение архитектурный паттерн Model-View-Controller (MVC) – это архитектурный паттерн программирования, описывающий способ организации структуры приложения. Он разделяет приложение на три основных компонента:

1. Модель (Model) - описывает используемые в приложении данные, а также логику, которая связана непосредственно с данными, например, логику валидации данных. Как правило, объекты моделей хранятся в базе данных.

В MVC модели представлены двумя основными типами: модели представлений, которые используются представлениями для отображения и передачи данных, и модели домена, которые описывают логику управления данными. Модель может содержать данные, хранить логику управления этими данными. В то же время модель не должна содержать логику взаимодействия с пользователем и не должна определять механизм обработки запроса. Кроме того, модель не должна содержать логику отображения данных в представлении.

2. Представление (View) - слой, который отображает данные пользователю. отвечают за визуальную часть или пользовательский интерфейс, нередко html-страница, через который пользователь взаимодействует с приложением. Также представление может содержать логику, связанную с отображением данных. В то же время представление не должно содержать логику обработки запроса пользователя или управления данными.

3. Контроллер (Controller) - слой, который обрабатывает пользовательский ввод, взаимодействует с моделью и передаёт данные в представление. Каждый запрос, поступающий в приложение, обрабатывается контроллером. И в зависимости от результатов обработки отправляет пользователю определённый вывод, например, в виде представления, наполненного данными моделей. Контроллер может обрабатывать запрос произвольным образом до тех пор, пока он не пересекает границу ответственности модели и представления.

Разделение приложения на эти три компонента упрощает его разработку и поддержку, позволяет разработчикам работать над отдельными компонентами независимо друг от друга, а также повышает его гибкость и масштабируемость. Диаграмма классов контроллеров и сервисов продемонстрирована на рисунке 12.

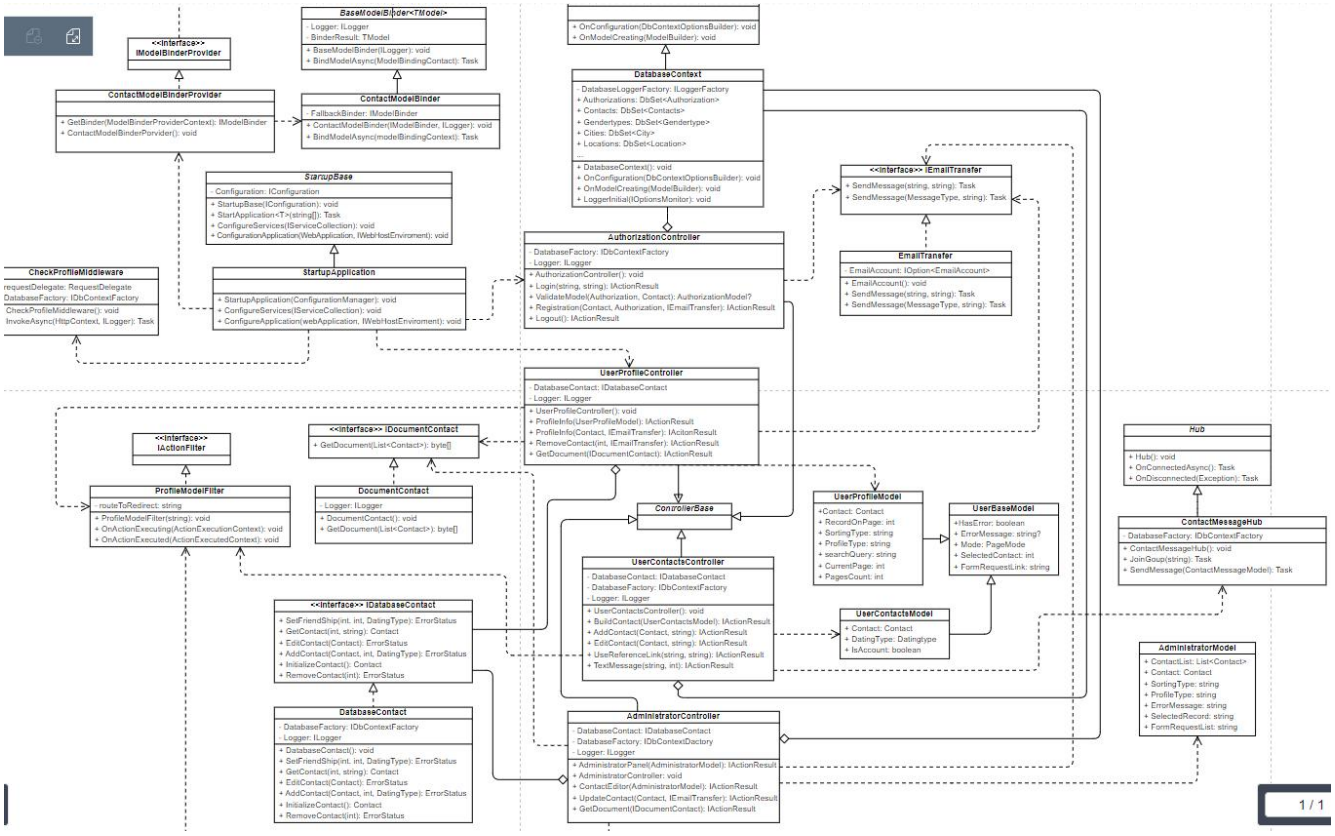


Рисунок 12 – Диаграмма классов UML

3.3 Функциональные возможности

При запуске программы пользователю будет предложено авторизоваться в систему, если прежде он этого не совершал (куки браузера связанные с данными авторизации очищены).

Если у пользователя уже существует аккаунт, он может совершить вход, используя свои данные авторизации. Иначе для взаимодействия с приложением ему необходимо будет зарегистрировать новый профиль. Страницы для совершения входа в систему представлены на рисунках 13 и 14.

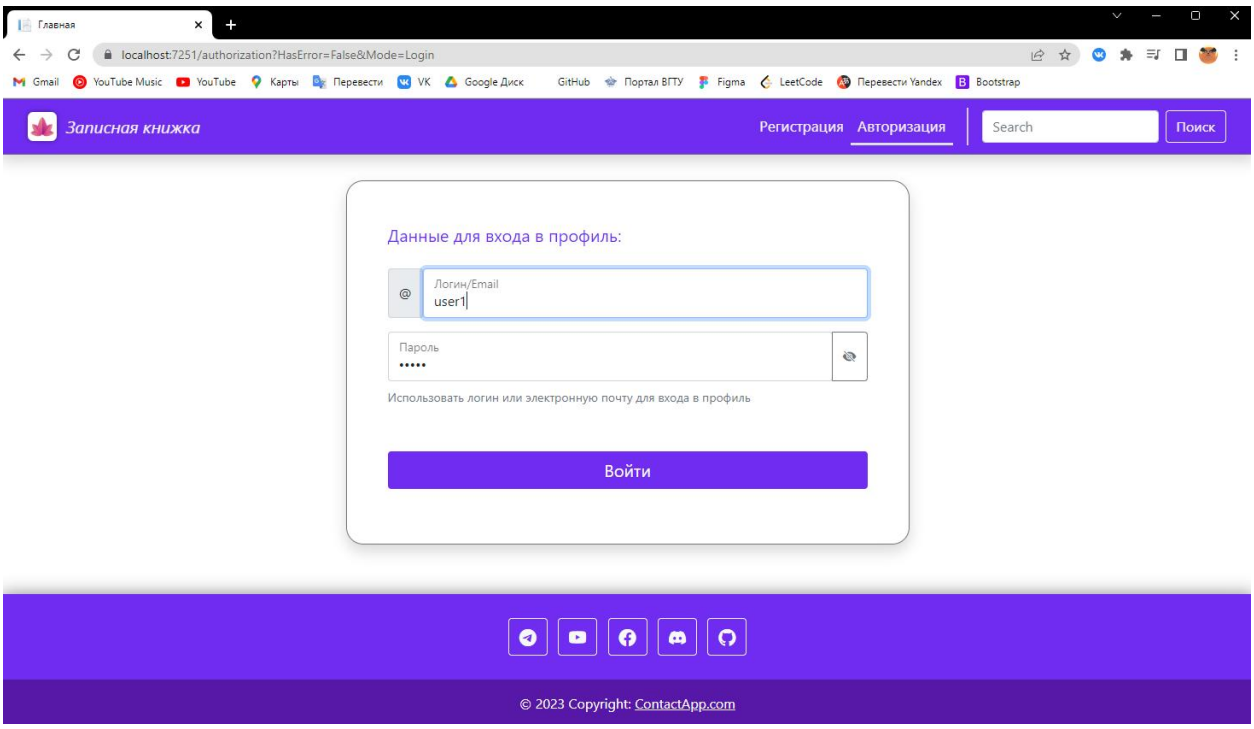


Рисунок 13 – Страница авторизации

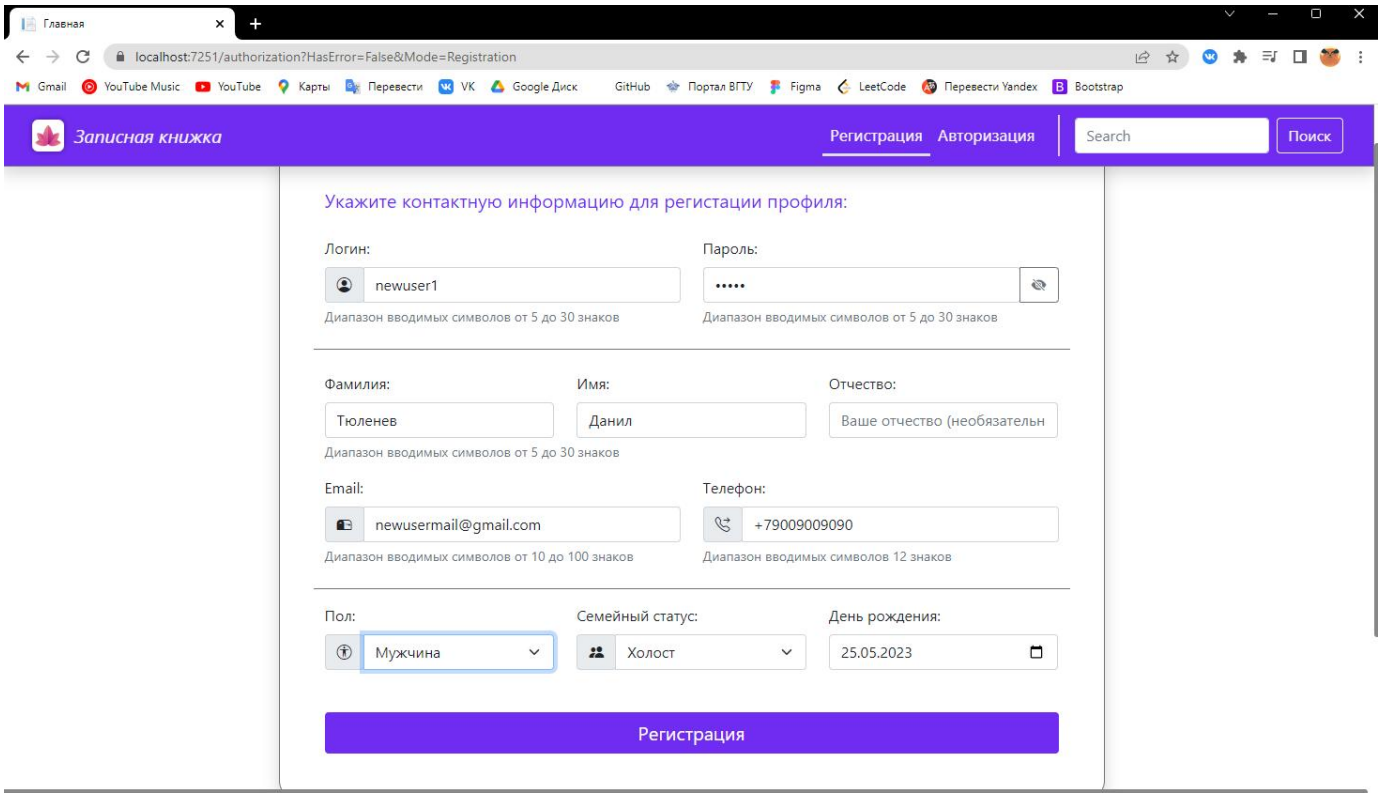


Рисунок 13 – Страница авторизации

После совершения входа в приложение у нового пользователя системы будет отображаться сообщение о том, что его контактная книга на данный момент пуста, продемонстрировано на рисунке 14.

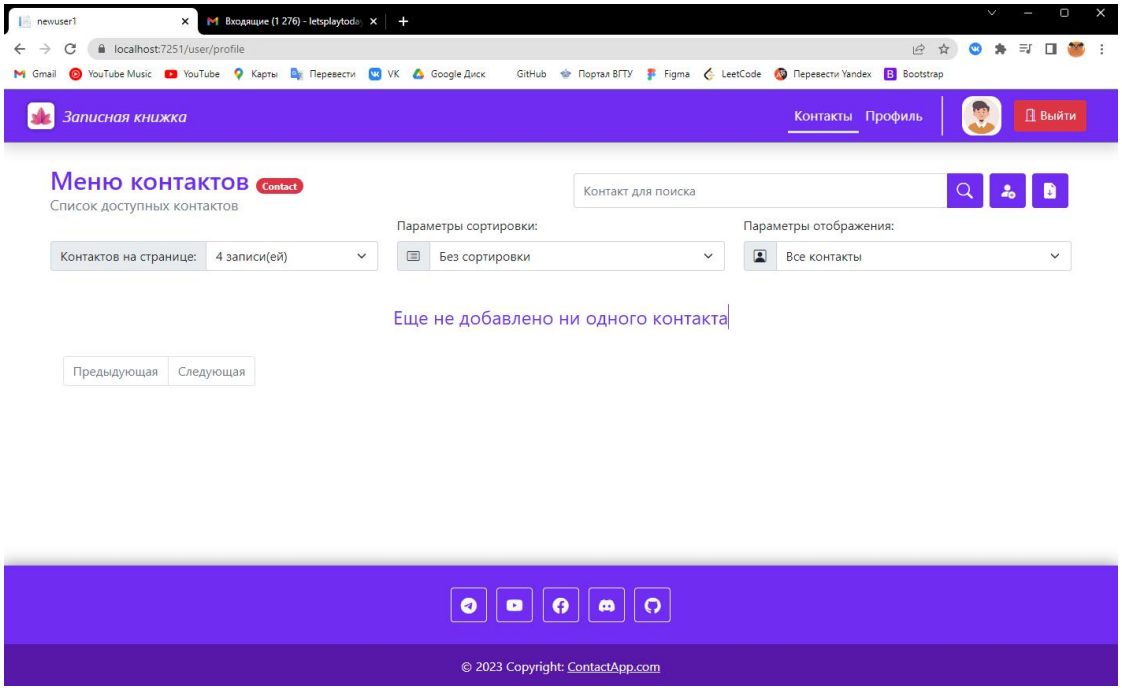


Рисунок 14 – Страница с пустым списком контактов

Чтобы добавить нового контакта, необходимо воспользоваться выпадающим меню «Создания нового контакта», которое активируется при нажатии на соответствующую кнопку в панели инструментов. После чего пользователь может выбрать: добавить уже существующего пользователя системы в качестве своего нового контакта, используя для этого специально сгенерированный GUID-код или Email почту (указав при этом характер знакомства); создать новый контакт, заполнив форму с контактными данными.

На рисунке 15 показано выпадающее меню для добавления новой записи в список контактов.

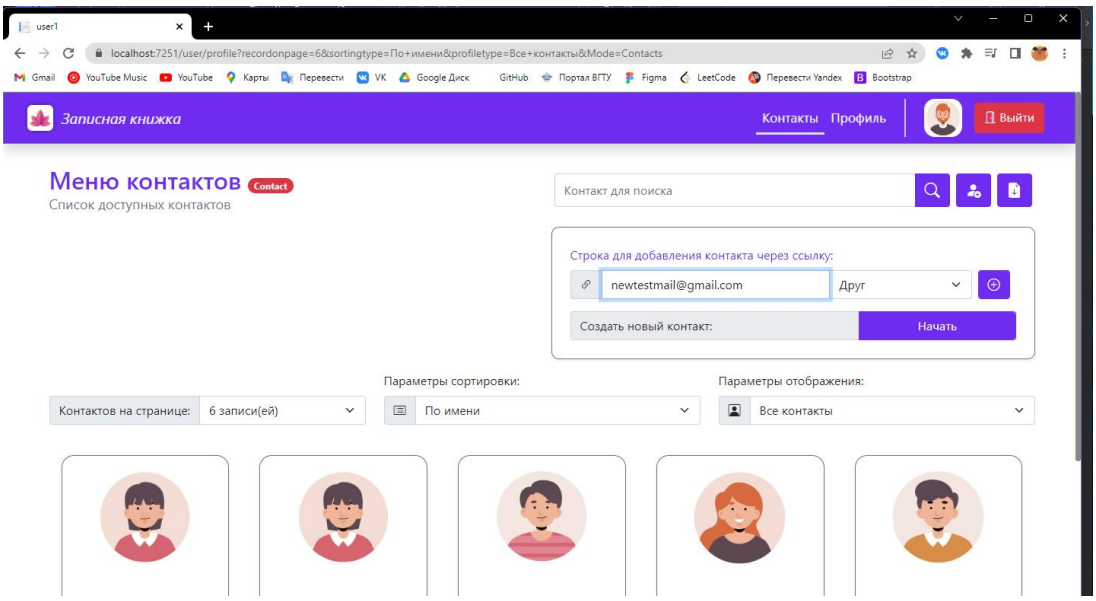


Рисунок 15 – Меню добавления контакта

Для создания нового контакта, пользователю будет доступна форма для заполнения, которая содержит контактную информацию, данные и месте проживания, семейном статусе, работе (учёбе), а также о качествах человека и списке хобби. Для сохранения заполненных данных необходимо указать характер знакомства. Форма создания контакта показана на рисунке 16.

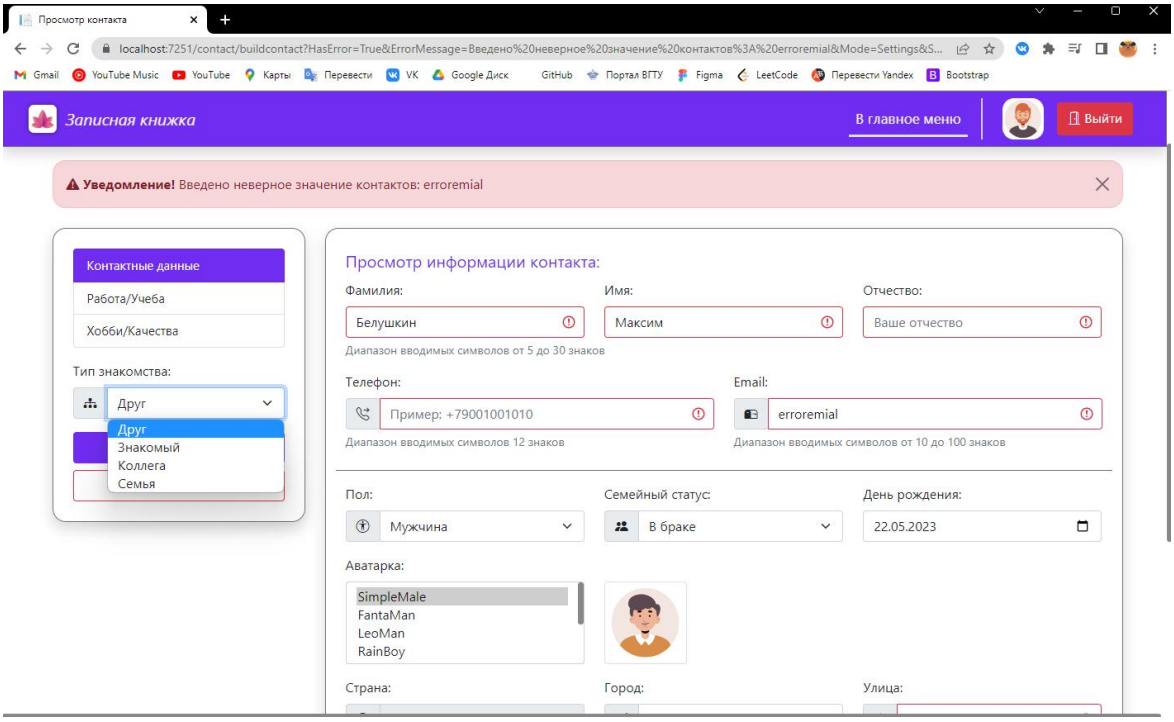


Рисунок 16 - Форма создания контакта

Если данные были введены в неверном формате, то пользователю будет отображено уведомление с сообщением о совершённой ошибке.

После создания или добавления записей на вкладке «Контакты» пользователю будет доступен список сохранённых контактов, с которым можно совершать следующие операции: отображать некоторое количество записей, с возможностью переключать страницы; сортировать по различным параметрам, фильтрация по типу контактов.

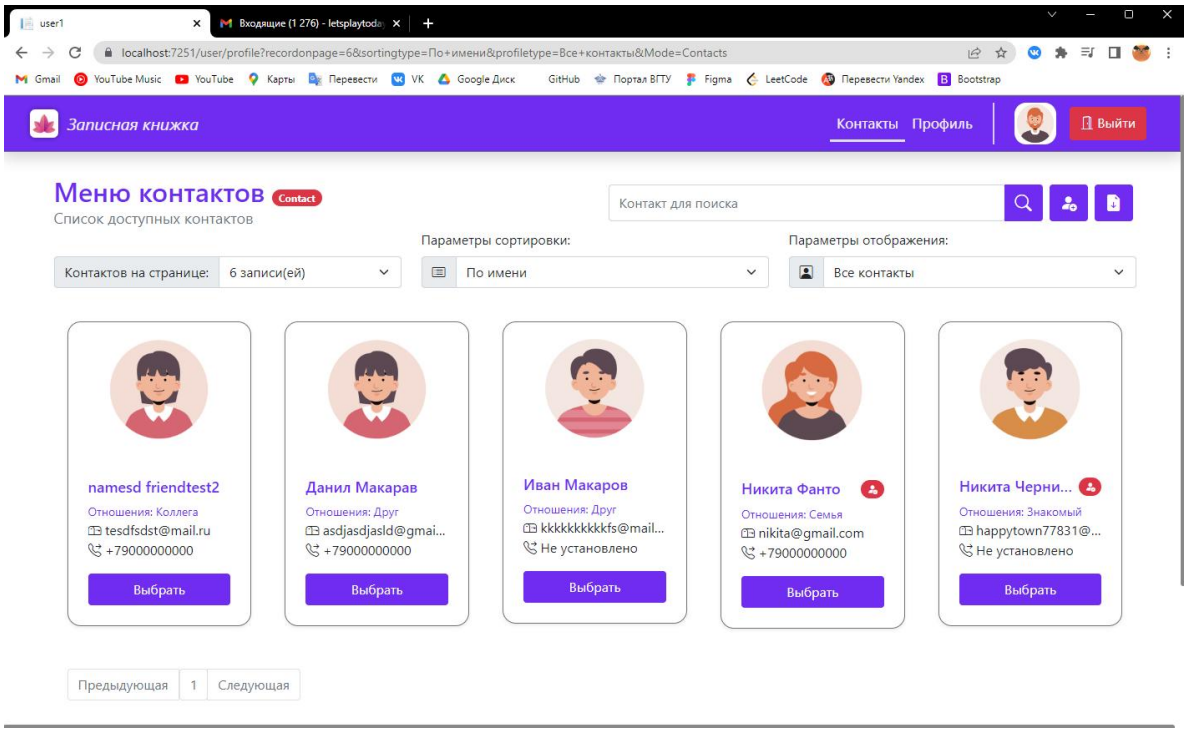


Рисунок 16 - Вкладка «Контакты»

Контакты, которые являются также другими пользователями системы помечены «красным знаком». При выборе данного типа контактов, будет открываться страница с информацией о пользователе: контактные данные, место работы, хобби и качества, а также будет доступен список сохранённых контактов и панель мессендрежа, через который имеется возможность вести «переписку», отправляя друг другу сообщения в реальном времени.

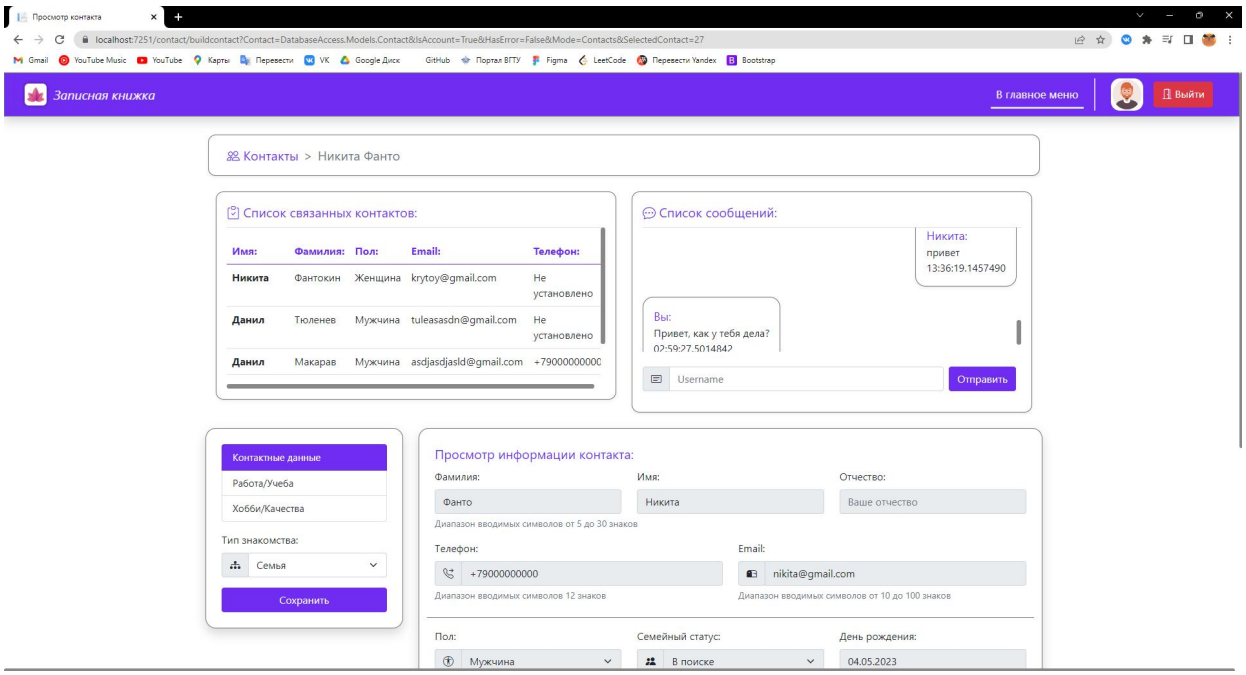


Рисунок 16 - Страница сохранённого контакта пользователя

Также пользователю доступна вкладка «Профиль» для настройки собственных контактных данных, информации о месте работы (учёбы), хобби и человеческие качества. Данные настройки можно увидеть на рисунках 17, 18, 19.

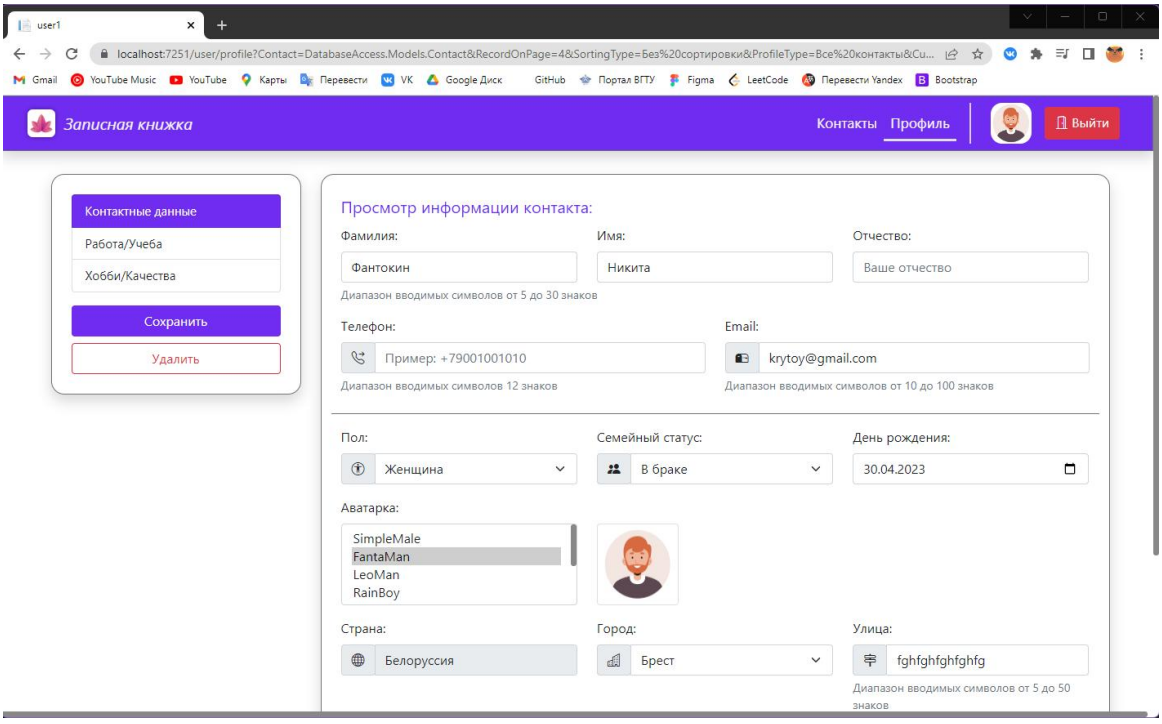


Рисунок 17 - Настройка профиля (часть 1)

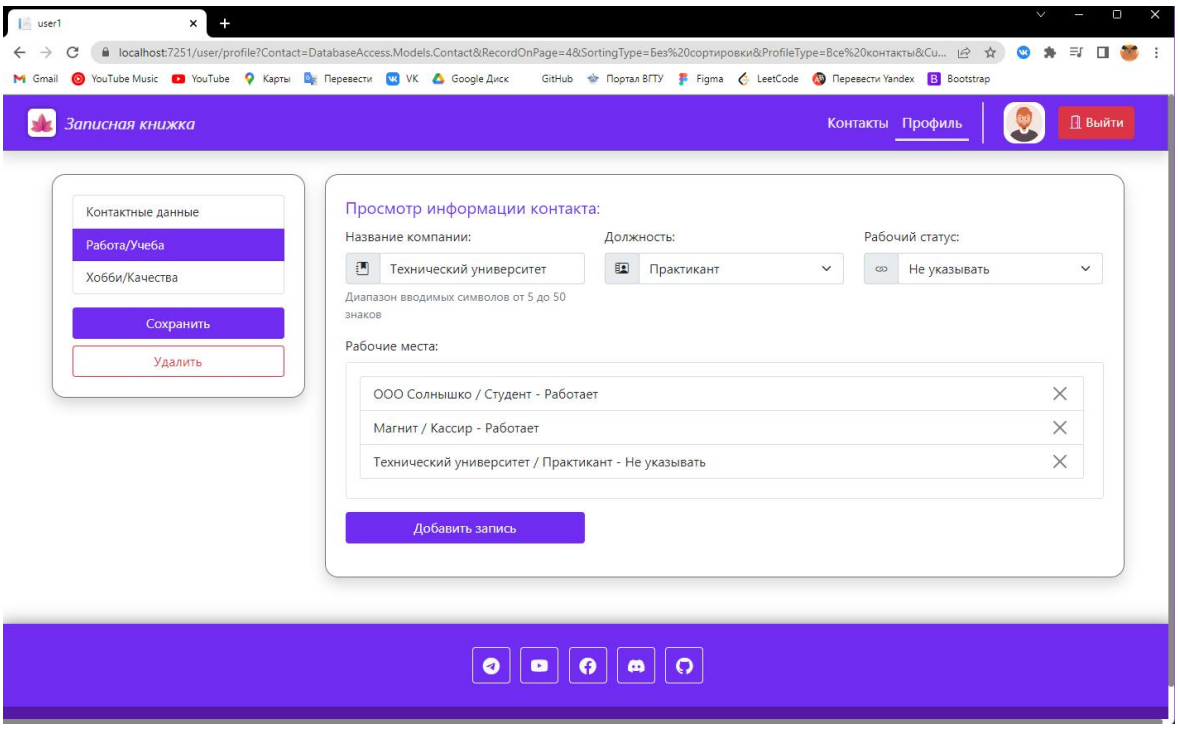


Рисунок 18 - Настройка профиля (часть 2)

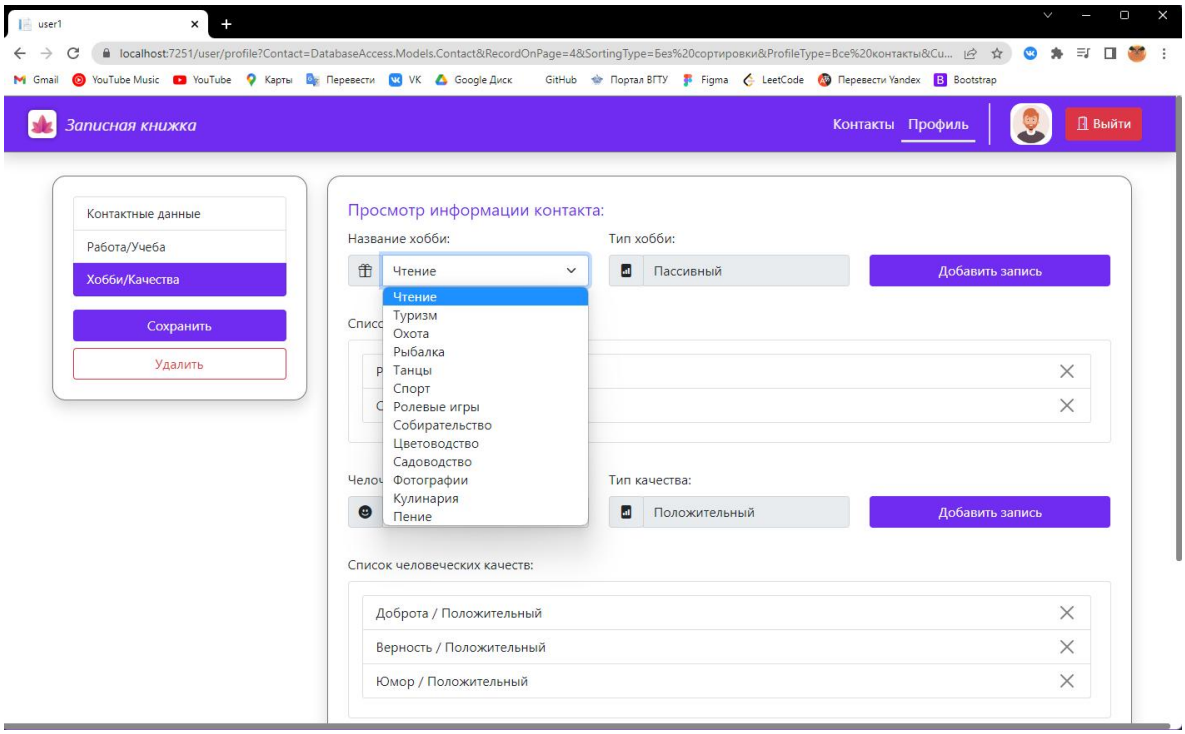


Рисунок 19 - Настройка профиля (часть 3)

После того, как данные будут введены в форму, необходимо нажать кнопку «Сохранить», далее будет отображено диалоговое окно для подтверждения изменения. Если пользователь подтвердит изменения, то вместе с сохранёнными изменениями пользователю будет отправлено письмо на указанную электронную

почту с уведомлением об совершённых изменениях. Вышеописанные события показаны на рисунках 20 и 21.

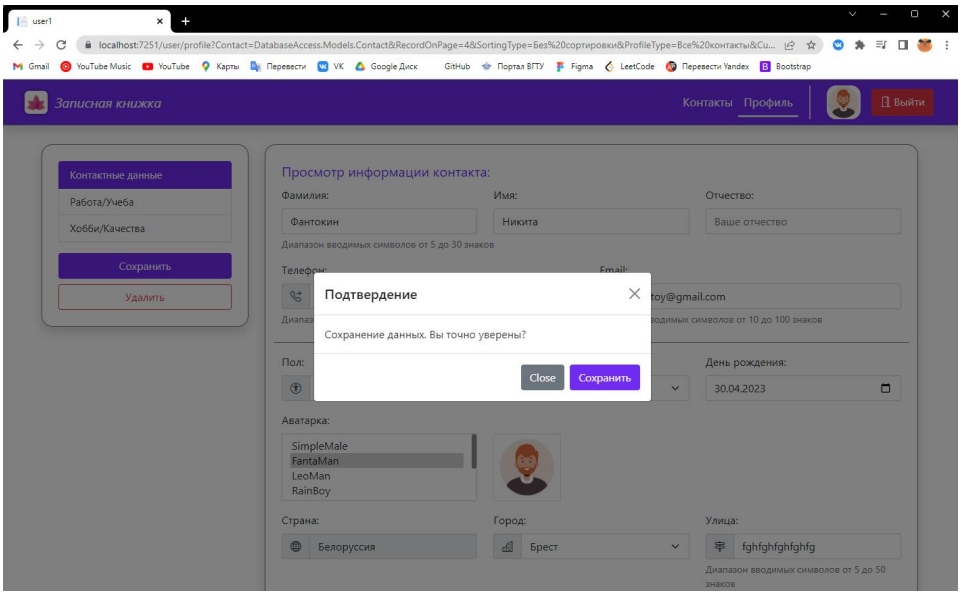


Рисунок 20 - Подтверждение изменения данных

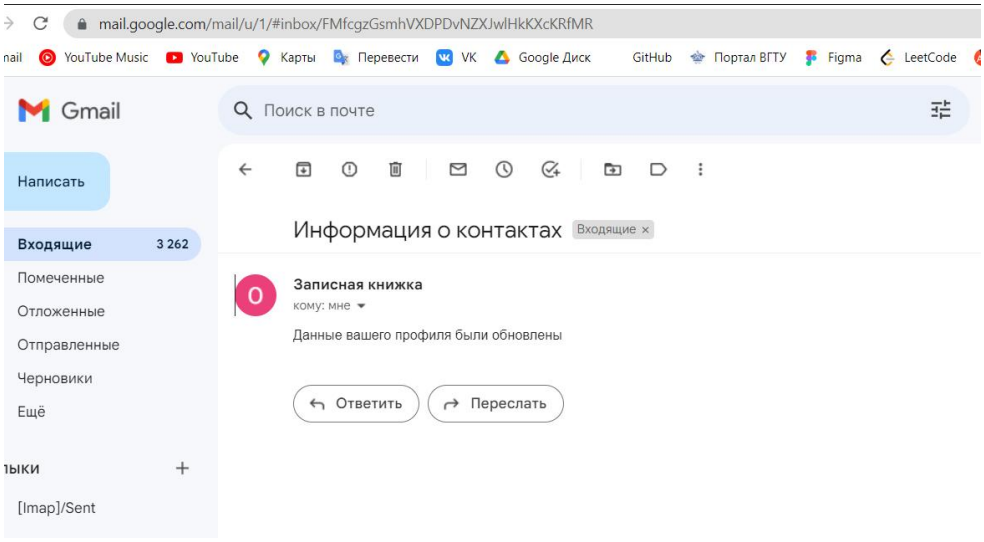


Рисунок 21 - Уведомление на электронную почту

Для доступа к панели администратора, пользователь должен выполнить авторизацию в аккаунт с привилегиями системного администратора. Вкладка «Администратор» позволяет просматривать список всех созданных контактов внутри системы, а также при необходимости редактировать и удалять отдельные записи, что продемонстрировано на рисунке 22.

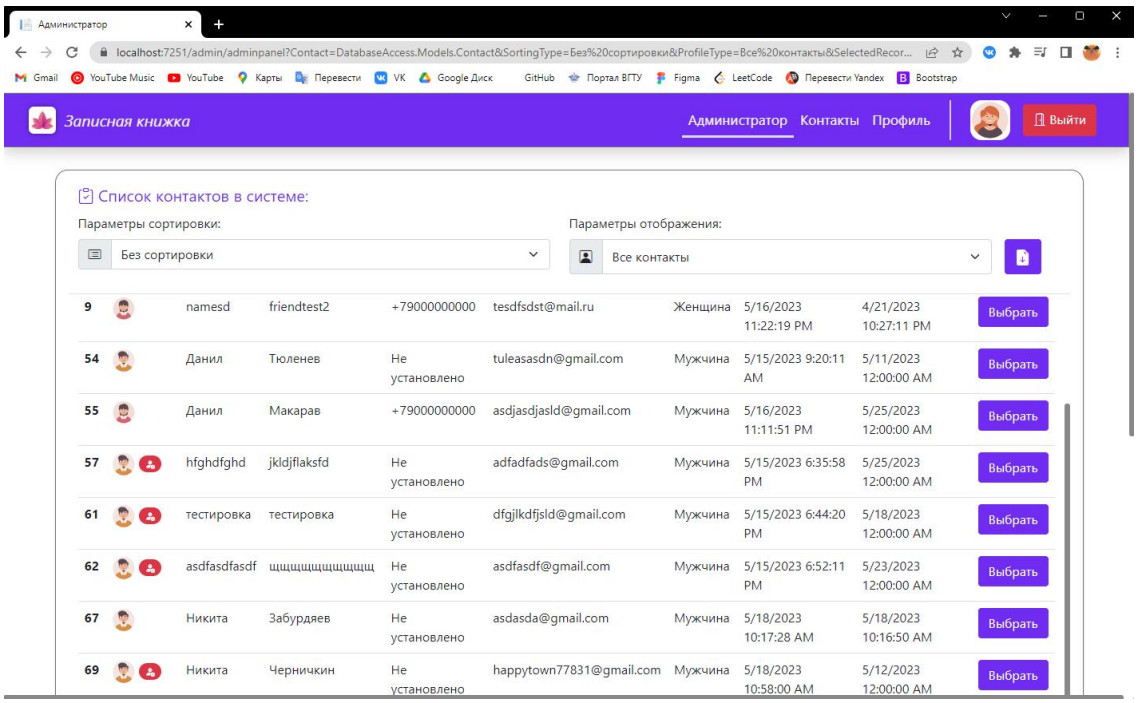


Рисунок 22 - Панель администратора

Также на вкладке администратора доступна сгруппированная информация по разным параметрам для вычисления суммарного количества с использованием функций агрегации, что показано на рисунке 23.

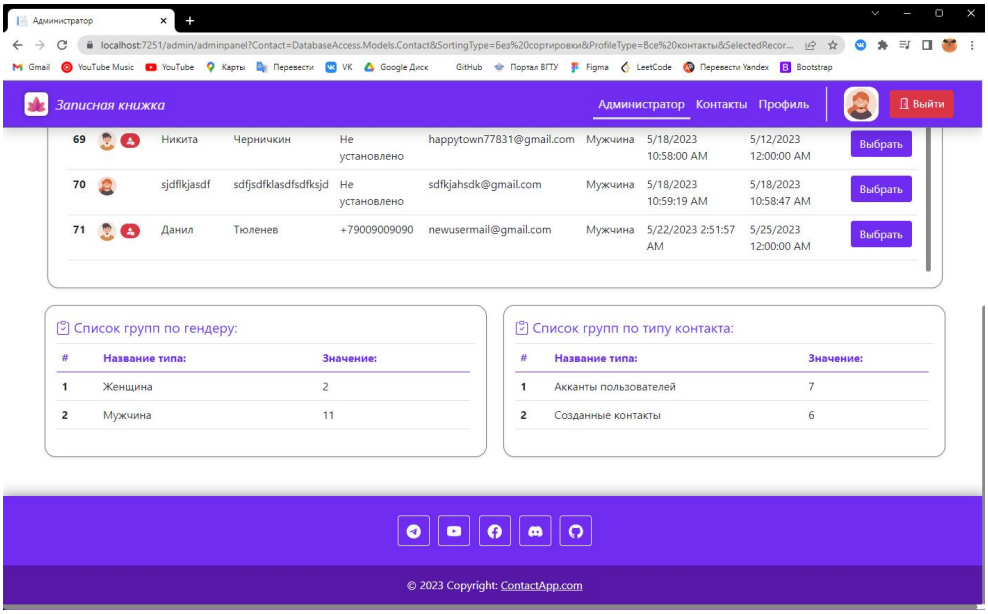
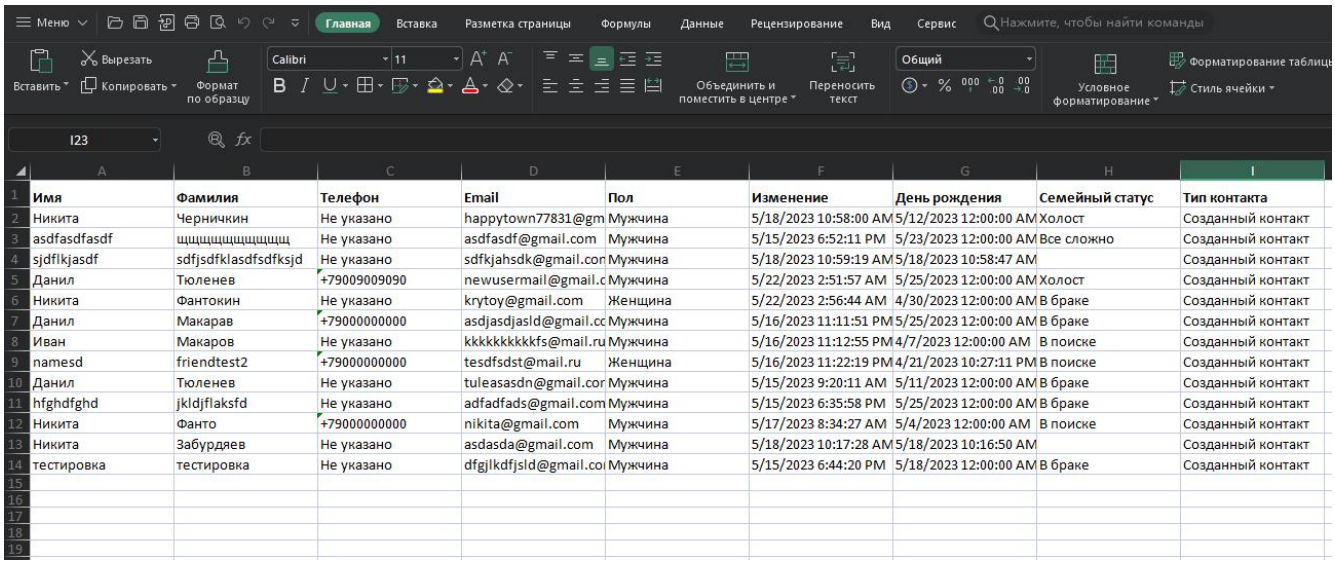


Рисунок 23 - Группировка данных по параметрам

Также при нажатии на кнопку «Экспорта» в панели инструментов пользователю системы будет сохранён файл электронных таблиц с

продублированными данными о созданных контактах системы. Данная возможность показана на рисунке 24.



Имя	Фамилия	Телефон	Email	Пол	Изменение	День рождения	Семейный статус	Тип контакта
Никита	Черничкин	Не указано	happytown77831@gm	Мужчина	5/18/2023 10:58:00 AM	5/12/2023 12:00:00 AM	Холост	Созданный контакт
asdfasdfasdf	щщщщщщщщщщ	Не указано	asdfasdf@gmail.com	Мужчина	5/15/2023 6:52:11 PM	5/23/2023 12:00:00 AM	Все сложно	Созданный контакт
sjdflkjasdf	sdfjsdfklasdfsdfksjd	Не указано	sdfkjahsdk@gmail.com	Мужчина	5/18/2023 10:59:19 AM	5/18/2023 10:58:47 AM		Созданный контакт
Данил	Тюленев	+79009009090	newusermail@gmail.c	Мужчина	5/22/2023 2:51:57 AM	5/25/2023 12:00:00 AM	Холост	Созданный контакт
Никита	Фантокин	Не указано	krytoy@gmail.com	Женщина	5/22/2023 2:56:44 AM	4/30/2023 12:00:00 AM	В браке	Созданный контакт
Данил	Макарав	+79000000000	asdjasdjasld@gmail.cc	Мужчина	5/16/2023 11:11:51 PM	5/25/2023 12:00:00 AM	В браке	Созданный контакт
Иван	Макаров	Не указано	kkkkkkkkkkfs@mail.ru	Мужчина	5/16/2023 11:12:55 PM	4/7/2023 12:00:00 AM	В поиске	Созданный контакт
namesd	friendtest2	+79000000000	tesdfsdst@mail.ru	Женщина	5/16/2023 11:22:19 PM	4/21/2023 10:27:11 PM	В поиске	Созданный контакт
Данил	Тюленев	Не указано	tuleasasdn@gmail.cor	Мужчина	5/15/2023 9:20:11 AM	5/11/2023 12:00:00 AM	В браке	Созданный контакт
hfghdfghd	jkldjflaksfd	Не указано	adfadfads@gmail.com	Мужчина	5/15/2023 6:35:58 PM	5/25/2023 12:00:00 AM	В браке	Созданный контакт
Никита	Фанто	+79000000000	nikita@gmail.com	Мужчина	5/17/2023 8:34:27 AM	5/4/2023 12:00:00 AM	В поиске	Созданный контакт
Никита	Забурдяев	Не указано	asdasda@gmail.com	Мужчина	5/18/2023 10:17:28 AM	5/18/2023 10:16:50 AM		Созданный контакт
тестировка	тестировка	Не указано	dfgjlkdfjsld@gmail.co	Мужчина	5/15/2023 6:44:20 PM	5/18/2023 12:00:00 AM	В браке	Созданный контакт

Рисунок 24 - Экспорт электронных таблиц

ЗАКЛЮЧЕНИЕ

Все большую актуальность и широкое распространение получают базы данных (БД) и системы управления базами данных (СУБД), использующиеся для обработки больших объёмов разного рода информации, в том числе экономической. БД способны хранить информацию о десятках, сотнях тысяч и миллионах различных объектов.

В рамках данной курсовой работы было разработано приложение "Записная книга", реализованное на платформе ASP.NET с использованием базы данных PostgreSQL. Данное приложение предназначено для хранения контактной информации о людях и предоставляет пользователю возможность удобно вести свой список контактов.

Основными достоинствами приложения являются: простой и интуитивно понятный интерфейс, который позволяет пользователям легко добавлять, редактировать, удалять и просматривать список контактов; использование базы данных PostgreSQL, что гарантирует сохранность и доступность данных; обработка ошибок ввода данных, что позволяет предотвратить возможные ошибки при заполнении контактной информации; доступность приложения из любой точки мира с помощью интернет-браузера.

Кроме того, были освоены основные принципы разработки приложений на языке программирования ASP.NET, такие как разделение приложения на слои, использование моделей представления и использование технологии Razor для формирования представлений.

Таким образом, создание приложения «Записная книга» на языке программирования ASP.NET с использованием базы данных PostgreSQL было успешно завершено. Были получены новые знания и навыки в области разработки веб-приложений и баз данных, которые могут быть использованы при дальнейшей разработке приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Язык программирования C# 7 и платформы .NET и .NET Core, 8-е изд. : Пер. с англ. / Эндрю Троелсен, Филипп Джепкинс – СПб. : ООО “Диалектика”, 2018 – 1328 с. : ил.

2. Программирование на СИ# : учеб. пособие / М. А. Медведев, А. Н. Медведев. – Екатеринбург : Изд-во Урал. ун-та, 2015 — 64 с.

3. Оптимизация запросов PostgreSQL — Г. Домбровская, Б. Новиков, А. Бейликова, 2022 - 112 с.

4. «Изучаем PostgreSQL 10» — С. Джуба, А. Волков, 2019 - 36 с.

5. Саймон, Ригс Администрирование PostgreSQL 9. Книга рецептов / Ригс Саймон. - М.: ДМК Пресс, 2018. - 806 с

6. Основы алгоритмизации и программирования на языке C# : учеб. пособие для бакалавриата и специалитета / Е. В. Кудрина, М. В. Огнева. – М. : Издательство Юрайт, 2019 — 322 с.

7. SQL для простых смертных / Грабер М. : Издательство Лори, 2014 — 375 с.

8. Чарльз Петцольд Программирование для Microsoft Windows на C# - В 2-х томах – Том 1. Переведено с английского – М.: Издательско-торговый дом “Русская Редакция”, 2002. – 576 с.

ПРИЛОЖЕНИЕ А

Листинг программы на C#

Файл «StartupBase.cs»

```
using DatabaseAccess;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authorization;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Options;
using System.Runtime.CompilerServices;
using System.Security.Claims;

namespace ClientApplication.Configurations
{
    public sealed class StartupException : System.Exception
    {
        public Type StartupType { get; private set; } = default!;
        public StartupException(string message, Type @class) : base(message)
{ StartupType = @class; }
    }

    public abstract class StartupBase : System.Object
    {
        protected virtual IConfiguration Configuration { get; private set; } = default!;
        public StartupBase(IConfiguration configuration) : base() { Configuration =
configuration; }

        public static Task StartApplication<TStartup>(string[] args) where TStartup :
StartupBase
        {
            var builder = WebApplication.CreateBuilder(args);
            var startup = Activator.CreateInstance(typeof(TStartup), new[]
{ builder.Configuration }) as TStartup;

            if (startup == null) throw new StartupException("Невозможно создать объект
загрузчика", typeof(TStartup));
            startup.ConfigureServices(builder.Services);

            WebApplication web_application = builder.Build();
            startup.ConfigureApplication(web_application, builder.Environment);

            return web_application.RunAsync();
        }
        protected abstract void ConfigureServices(IServiceCollection services);
        protected abstract void ConfigureApplication(WebApplication application,
IWebHostEnvironment env);
    }
}
```

Файл «StartupClient.cs»

```
using DatabaseAccess;
using ClientApplication.Infrastructure;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;
using Microsoft.AspNetCore.Authorization;
using System.Text.Json.Serialization;
using Newtonsoft.Json;
using Microsoft.Extensions.FileProviders;
using Microsoft.Extensions.Options;
```

```
using ClientApplication.Services;
using ClientApplication.Middleware;

namespace ClientApplication.Configurations
{
    public partial class StartupApplication : Configurations.StartupBase
    {
        public StartupApplication(ConfigurationManager configuration) :
base(configuration)
        {

configuration.SetBasePath(AppDomain.CurrentDomain.BaseDirectory).AddJsonFile("dbsettings.
json");
        }
        protected override void ConfigureServices(IServiceCollection services)
        {
            services.AddRazorPages(); services.AddServerSideBlazor();
services.AddHttpClient(); services.AddSignalR();

services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).AddCookie((
options) =>
            {
                options.AccessDeniedPath = new PathString("/authorization");
                options.LoginPath = new PathString("/authorization");
            });

services.Configure<DatabaseLoggerConfiguration>(Configuration.GetSection("DatabaseLoggerC
onfiguration"));

services.Configure<IEmailTransfer.EmailAccount>(Configuration.GetSection("EmailAccess"));
            services.AddAuthorization((AuthorizationOptions options) =>
            {
                options.AddPolicy("Administrator", policy =>
policy.RequireClaim(ClaimTypes.Role, "Admin"));
                options.AddPolicy("DefaultUser", policy =>
policy.RequireClaim(ClaimTypes.Role, "User"));
            });
            services.AddControllersWithViews((MvcOptions options) =>
            {
                options.ModelBinderProviders.Insert(0, new
ContractModelBinder.ContractModelBinderProvider());
            });
            services.AddDbContextFactory<DatabaseContext>((DbContextOptionsBuilder
options) =>
            {
                if (options is DbContextOptionsBuilder<DatabaseContext> converted_options)
                { new
DatabaseContextFactory.DatabaseConfigure(converted_options!).ConfigureOptions(); }
            });
            services.AddSignalR(options => options.MaximumReceiveMessageSize =
102400000L);
            services.AddTransient<Services.IDatabaseContact, Services.DatabaseContact>()
                .AddTransient<Services.IEmailTransfer, Services.EmailTransfer>()
                .AddTransient<Services.IDocumentContact, Services.DocumentContact>();
        }
        protected override void ConfigureApplication(WebApplication application,
IWebHostEnvironment env)
        {
            application.Map("/", (HttpContext context) =>
Results.RedirectToRoute("profile"));
            application.UseStaticFiles();
            application.UseStaticFiles(new StaticFileOptions()
            {
                FileProvider = new
PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), @"Icons")),
```

```
                RequestPath = new PathString("/Icons")
            });
            application.UseRouting().UseAuthentication().UseAuthorization();

            application.UseMiddleware<Middleware.CheckProfileMiddleware>();
            application.UseEndpoints((IEndpointRouteBuilder route_builder) =>
            {
                route_builder.MapControllers();
                route_builder.MapControllerRoute("default",
"{controller=Home}/{action=Index}/{id?}");
            });
            application.MapBlazorHub();
application.MapHub<ContactMessagerHub>("/chathub");
        }
    }
}
```

Файл «AdministratorController.cs»

```
using ClientApplication.Filters;
using ClientApplication.Services;
using ClientApplication.ViewModels;
using DatabaseAccess;
using DatabaseAccess.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Storage;
using Newtonsoft.Json;
using System.Security.Claims;

namespace ClientApplication.Controllers
{
    using ControllerLogger = ILogger<AdministratorController>;

    [ControllerAttribute, RouteAttribute(ControllerRoute), AuthorizeAttribute(Policy =
"Administrator")]
    public partial class AdministratorController : Controller
    {
        public const string ControllerRoute = "admin";
        protected Services.IDatabaseContact DatabaseContact { get; private set; } =
default!;
        protected IDbContextFactory<DatabaseContext> DatabaseFactory { get; private set; }
= default!;

        protected readonly ILogger<AdministratorController> _logger = default!;
        public AdministratorController(Services.IDatabaseContact dbcontact,
ControllerLogger logger,
            IDbContextFactory<DatabaseContext> factory) : base()
        { (this.DatabaseContact, this._logger, this.DatabaseFactory) = (dbcontact, logger,
factory); }

        [HttpGetAttribute, RouteAttribute("adminpanel", Name = "adminpanel")]
        public async Task<IActionResult> AdministratorPanel(AdministratorModel? model)
        {
            var profileId =
int.Parse(this.HttpContext.User.FindFirst(ClaimTypes.PrimarySid)!.Value);

            if (model == null) model = new AdministratorModel();
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                model.ContactsList = dbcontext.Contacts.Include(item => item.Gendertype)
                    .Include(item => item.Authorization).Include(item => item.Userpicture)
                    .Where(item => item.Contactid != profileId).OrderBy(item =>
item.Contactid).ToList();
```

```
                model.ContactsList = model.ContactsList.Where(item => model.ProfileType
switch
                {
                    "Все контакты" => true, "Созданные" => item.Authorization == null,
                    "Аккаунты" => item.Authorization != null, _ => true,
                }).ToList();
                model.ContactsList = (model.SortingType switch
                {
                    "Без сортировки" => model.ContactsList.OrderBy(item =>
item.Contactid),
                    "По дате изменения" => model.ContactsList.OrderBy(item =>
item.Lastupdate),
                    "По дате рождения" => model.ContactsList.OrderBy(item =>
item.Birthday),

                    "По имени" => model.ContactsList.OrderBy(item => item.Name),
                    _ => model.ContactsList.OrderBy(item => item.Contactid)
                }).ToList();
            }
            return base.View(model);
        }

        [HttpGetAttribute, RouteAttribute("contacteditor", Name = "contacteditor")]
        public async Task<IActionResult> ContactEditor(AdministratorModel model)
        {
            model.Contact = (await this.DatabaseContact.GetContact(model.SelectedRecord,
string.Empty))!;
            model.FormRequestLink = $"admin/updatecontact";

            if (model.Contact == null) return base.RedirectToRoute("profile", new
ViewModels.UserProfileModel()
            { Mode = UserBaseModel.PageMode.Contacts, ErrorMessage = "Контакт не
найден" });

            string modelSerialize = JsonConvert.SerializeObject(model, new
JsonSerializerSettings()
            { ReferenceLoopHandling = ReferenceLoopHandling.Ignore });

            return
base.View(JsonConvert.DeserializeObject<AdministratorModel>(modelSerialize));
        }

        [HttpPostAttribute, RouteAttribute("updatecontact", Name = "updatecontact")]
        [ProfileModelFilter("contacteditor")]
        public async Task<IActionResult> UpdateContact(Contact contactModel,
[FromServices] IEmailTransfer email)
        {
            IDatabaseContact.ErrorStatus? errorMessage = default!;
            var resultModel = new AdministratorModel() { };

            if ((errorMessage = await this.DatabaseContact.EditContact(contactModel)) !=
null)
            {
                resultModel.ErrorMessage = errorMessage.Message;
            }
            await email.SendMessage(IEmailTransfer.MessageType.Update,
contactModel.Emailaddress);

            return base.RedirectToRoute("contacteditor", new AdministratorModel()
            { SelectedRecord = contactModel.Contactid });
        }

        [HttpGetAttribute, RouteAttribute("getusers", Name = "getusers")]
        public async Task<IActionResult> GetDocument([FromServices] IDocumentContact
generator)
```

```
        {
            List<Contact> contacsList = default!;
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                contacsList = await dbcontext.Contacts.Include(item =>
item.Gendertype).ToListAsync();
            }
            return base.File(await generator.GetDocument(contacsList),
"application/octet-stream", "report.xlsx");
        }
    }
}
```

Файл «AuthorizationController.cs»

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore.Internal;
using DatabaseAccess;
using ClientApplication.ViewModels;
using Microsoft.EntityFrameworkCore;
using System.Runtime.CompilerServices;
using Microsoft.AspNetCore.Authentication;
using System.Security.Claims;
using Microsoft.AspNetCore.Authentication.Cookies;
using System.Text.RegularExpressions;
using ClientApplication.Services;

namespace ClientApplication.Controllers
{
    using DAModels = DatabaseAccess.Models;
    using ControllerLogger = ILogger<AuthorizationController>;

    using static ClientApplication.ViewModels.AuthorizationModel;

    [ControllerAttribute]
    public partial class AuthorizationController : Controller
    {
        protected IDbContextFactory<DatabaseContext> DatabaseFactory { get; private set; }
= default!;

        private readonly ILogger<AuthorizationController> _logger;
        public AuthorizationController(IDbContextFactory<DatabaseContext> factory,
ControllerLogger logger)
            : base() { (this.DatabaseFactory, this._logger) = (factory, logger); }

        [RouteAttribute("authorization", Name = "authorization")]
        [HttpGetAttribute]
        public async Task<IActionResult> Authorization(ViewModels.AuthorizationModel?
model)
        {
            using var dbcontext = await this.DatabaseFactory.CreateDbContextAsync();
            this.ViewBag.GenderTypes = dbcontext.Gendertypes.Select(item =>
item.Gendertypename).ToList();

            if (this.HttpContext.User.Identity != null &&
this.HttpContext.User.Identity.IsAuthenticated)
            {
                return base.LocalRedirect("/user/profile");
            }
            return base.View(model ?? new ViewModels.AuthorizationModel());
        }

        [RouteAttribute("login", Name = "login"), HttpPostAttribute]
        public async Task<IActionResult> Login([FromForm]string login, [FromForm]string
password)
        {
```

```
            this._logger.LogInformation($"Login: {login}; Password: {password}");
            DAModels::Authorization? userProfile = default;
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                userProfile = dbcontext.Authorizations.Include((item) => item.Contact)
                    .Where((item) => (item.Login == login || item.Contact.Emailaddress ==
login)
                    && item.Password == password).FirstOrDefault();

                if (userProfile == null) return base.RedirectToRoute("authorization", new
AuthorizationModel
                { ErrorCause = "", Mode = AuthorizationMode.Login, HasError = true });
            }
            var profilePrinciple = new ClaimsIdentity(new List<Claim>()
            {
                new Claim(ClaimTypes.Role, "User"),
                new Claim(ClaimTypes.PrimarySid, userProfile.Contact.Contactid.ToString())
            },
            CookieAuthenticationDefaults.AuthenticationScheme);
            if(userProfile.Isadmin) profilePrinciple.AddClaim(new Claim(ClaimTypes.Role,
"Admin"));

            await this.HttpContext.SignInAsync(new ClaimsPrincipal(profilePrinciple));
            return base.RedirectToAction("ProfileInfo", "UserProfile");
        }
        private AuthorizationModel? ValidateModel(DAModels::Authorization auth,
DAModels::Contact user)
        {
            foreach (var propertyValue in new[] { auth.Login, auth.Password, user.Surname,
user.Name })
            {
                if (propertyValue != null && Regex.IsMatch(propertyValue, @"[А-Яа-
я\w]{4,}")) continue;
                return new ViewModels.AuthorizationModel(true)
                {
                    ErrorCause = @$"Данные неверно введены{(propertyValue == null ? "" :
$": {propertyValue}")}",
                    Mode = ViewModels.AuthorizationModel.AuthorizationMode.Registration
                };
            }
            var emailMatch = Regex.IsMatch(user.Emailaddress,
@"^\w{6,}@(mail|gmail|yandex){1}.(ru|com){1}$");
            var phoneMatch = Regex.IsMatch(user.Phonenumber!, @"^\+7[0-9]{10}$");

            user.Phonenumber = (user.Phonenumber == string.Empty) ? null! :
user.Phonenumber;
            if (!emailMatch || (!phoneMatch && user.Phonenumber != null))
            {
                return new ViewModels.AuthorizationModel(true)
                {
                    ErrorCause = (!emailMatch ? "Неверный адрес почты" : (!phoneMatch ?
"Неверный телефон" : null)),
                    Mode = ViewModels.AuthorizationModel.AuthorizationMode.Registration,
                };
            }
            return default(AuthorizationModel);
        }
        [RouteAttribute("registration", Name = "registration")]
        [HttpPostAttribute]
        public async Task<IActionResult> Registration([FromForm] DAModels::Contact
profile,
            [FromForm, Bind("Login", "Password")] DAModels::Authorization authorization,
[FromServices] IEmailTransfer email)
        {
```

```
            if (profile is null) return base.RedirectToAction("Authorization", new
RouteValueDictionary()
            {["haserror"] = true, ["mode"] = AuthorizationMode.Registration,
["errorcause"] = "Данные не установлены"});

            var modelValidation = this.ValidateModel(authorization, profile);
            if (modelValidation != null) return base.RedirectToAction("Authorization",
new RouteValueDictionary()
            {
                ["haserror"] = modelValidation.HasError, ["mode"] = modelValidation.Mode,
                ["errorcause"] = modelValidation.ErrorCause
            });

            var referenceCollision = default(int) + 1;
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                profile.Gendertype = (await
dbcontext.Gendertypes.Where((DAModels::Gendertype item)
                    => item.Gendertypename ==
profile.Gendertype.Gendertypename).FirstOrDefaultAsync())!;

                profile.Lastupdate = DateTime.Now;
                profile.Userpicture = await dbcontext.Userpictures.FirstAsync();
                while (referenceCollision != 0)
                {
                    authorization!.Referenceguid = Guid.NewGuid().ToString();

                    referenceCollision = dbcontext.Authorizations.Where(
                        (DAModels::Authorization item) => item.Referenceguid ==
authorization.Referenceguid).Count();
                }
                profile.Authorization = default(DAModels::Authorization);
                await dbcontext.Contacts.AddAsync(authorization.Contact = profile);

                await dbcontext.Authorizations.AddAsync(authorization);
                try { await dbcontext.SaveChangesAsync(); } catch (System.Exception error)
                {
                    return base.RedirectToAction("Authorization", new
RouteValueDictionary()
                    { ["haserror"] = true, ["mode"] = AuthorizationMode.Registration,
["errorcause"] = error.Message });
                }
            }
            var profile_principle = new ClaimsIdentity(new List<Claim>()
            {
                new Claim(ClaimTypes.Role, authorization.Isadmin ? "Admin": "User"),
                new Claim(ClaimTypes.PrimarySid,
authorization.Contact.Contactid.ToString())
            },
            CookieAuthenticationDefaults.AuthenticationScheme);
            this._logger.LogInformation($"Account was registered: {authorization.Login}");

            await this.HttpContext.SignInAsync(new ClaimsPrincipal(profile_principle));
            await email.SendMessage(IEmailTransfer.MessageType.Registration,
profile.Emailaddress);
            return base.RedirectToAction("ProfileInfo", "UserProfile");
        }

        [RouteAttribute("logout", Name = "logout"), HttpGetAttribute]
        public async Task<IActionResult> Logout()
        {
            await
this.HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
            return base.LocalRedirect("/");
        }
```

```
    }
}
```

Файл «UserContactsController.cs»

```
using ClientApplication.Filters;
using ClientApplication.Services;
using ClientApplication.ViewModels;
using DatabaseAccess;
using DatabaseAccess.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Routing;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Query;
using Microsoft.EntityFrameworkCore.Storage;
using Newtonsoft.Json;
using System.Collections.Immutable;
using System.Linq;
using System.Security.Claims;
using System.Text.RegularExpressions;

namespace ClientApplication.Controllers
{
    using DAModels = DatabaseAccess.Models;
    using ControllerLogger = ILogger<UserContactsController>;

    [ControllerAttribute, RouteAttribute(ControllerRoute), AuthorizeAttribute(Policy =
"DefaultUser")]
    public partial class UserContactsController : Controller
    {
        public const string ControllerRoute = "contact";
        protected Services.IDatabaseContact DatabaseContact { get; private set; } =
default!;
        protected IDbContextFactory<DatabaseContext> DatabaseFactory { get; private set; }
= default!;

        protected readonly ILogger<UserContactsController> _logger = default!;
        public UserContactsController(Services.IDatabaseContact dbcontact,
ControllerLogger logger,
            IDbContextFactory<DatabaseContext> factory) : base()
        { (this.DatabaseContact, this._logger, this.DatabaseFactory) = (dbcontact, logger,
factory); }

        [HttpGetAttribute, RouteAttribute("buildcontact", Name = "buildcontact")]
        public async Task<IActionResult> BuildContact(UserContactsModel model)
        {
            var authorizatedProfile =
this.HttpContext.User.FindFirst(ClaimTypes.PrimarySid)!;

            if (model.SelectedContact == default) model.Contact = await
DatabaseContact.InitializeContact();
            else model.Contact = (await
this.DatabaseContact.GetContact(model.SelectedContact, string.Empty))!;

            model.FormRequestLink = string.Format("{0}/{1}",
UserContactsController.ControllerRoute,
                model.SelectedContact == default ? "addcontact" : "editcontact");

            if (model!.Contact == null) return base.RedirectToRoute("profile", new
UserProfileModel()
            { ErrorMessage = "Контакт не найден" });
            else model.IsAccount = model.Contact.Authorization != null;

            var friend = model.Contact.FriendContactid1Navigations.FirstOrDefault(item =>
```

```
                item.Contactid2Navigation.Contactid ==
int.Parse(authorizatedProfile.Value));
            if (friend != null) model.DatingType = friend.Datingtype!;

            var modelSerialize = JsonConvert.SerializeObject(model, new
JsonSerializerSettings()
                { ReferenceLoopHandling = ReferenceLoopHandling.Ignore });
            return
base.View(JsonConvert.DeserializeObject<UserContactsModel>(modelSerialize));
        }

        [HttpPostAttribute, RouteAttribute("addcontact", Name = "addcontact")]
        [ProfileModelFilter("buildcontact")]
        public async Task<IActionResult> AddContact([FromForm] DAModels::Contact
contactModel,
            [FromForm]string datingtype)
        {
            var authorizatedProfile =
this.HttpContext.User.FindFirst(ClaimTypes.PrimarySid)!;

            Console.WriteLine($"\nid1:{contactModel}, id2:
{int.Parse(authorizatedProfile.Value)}");
            var resultModel = new ViewModels.UserProfileModel()
            {
                Mode = UserBaseModel.PageMode.Contacts, ErrorMessage = default!, HasError
= default,
            };
            var error = await this.DatabaseContact.AddContact(contactModel,
int.Parse(authorizatedProfile.Value),
                new DAModels::Datingtype() { Typeofdating = datingtype });

            if ((resultModel.ErrorMessage = error?.Message) != null) return
base.RedirectToRoute("profile", resultModel);
            resultModel.ErrorMessage = string.Format("Контакт успешно создан");

            return base.RedirectToRoute("profile", resultModel);
        }

        [HttpPostAttribute, RouteAttribute("editcontact", Name = "editcontact")]
        [ProfileModelFilter("buildcontact")]
        public async Task<IActionResult> EditContact([FromForm] DAModels::Contact
contactModel,
            [FromForm] string datingtype)
        {
            var profileId = this.HttpContext.User.FindFirst(ClaimTypes.PrimarySid)!;
            var record = await this.DatabaseContact.GetContact(contactModel.Contactid,
string.Empty);

            if (record == null) return base.RedirectToRoute("profile", new
ViewModels.UserProfileModel()
            { Mode = UserBaseModel.PageMode.Contacts, ErrorMessage = "Контакт не
найден" });

            var errorModel = default(IDatabaseContact.ErrorStatus);
            if(record.Authorization == null && (errorModel = await
this.DatabaseContact.EditContact(contactModel)) != null)
            {
                return base.RedirectToRoute("profile", new ViewModels.UserProfileModel()
                { Mode = UserBaseModel.PageMode.Contacts, ErrorMessage =
errorModel.Message });
            }
            if(datingtype != null) errorModel = await
this.DatabaseContact.SetFriendShip(int.Parse(profileId.Value),
                record.Contactid, new DAModels::Datingtype() { Typeofdating =
datingtype });
```

```
            if (errorModel != null) base.RedirectToRoute("buildcontact", new
UserContactsModel()
            { SelectedContact = contactModel.Contactid, ErrorMessage =
errorModel.Message });

            return base.RedirectToRoute("buildcontact", new UserContactsModel()
            { SelectedContact = contactModel.Contactid });
        }

        [HttpPostAttribute, RouteAttribute("referencelink", Name = "referencelink")]
        public async Task<IActionResult> UseReferenceLink([FromForm]string link,
[FromForm]string datingtype)
        {
            var profileId = this.HttpContext.User.FindFirst(ClaimTypes.PrimarySid)!;
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                var record = await dbcontext.Contacts.Include(item => item.Authorization)
                    .Where(item => item.Authorization != null).FirstOrDefaultAsync(
                    item => item.Emailaddress == link ||
item.Authorization!.Referenceguid == link);

                if(record == null) return base.RedirectToRoute("profile", new
UserProfileModel()
                { Mode = UserProfileModel.PageMode.Contacts, HasError = true,
ErrorMessage = "Контакт не найден" });

                var error1 = await
this.DatabaseContact.SetFriendShip(int.Parse(profileId.Value), record.Contactid,
                    new DAModels::Datingtype() { Typeofdating = datingtype });
                var error2 = await this.DatabaseContact.SetFriendShip(record.Contactid,
int.Parse(profileId.Value),
                    new DAModels::Datingtype() { Typeofdating = datingtype });

                if (error1 != null || error2 != null) return
base.RedirectToRoute("profile", new UserProfileModel()
                { Mode = UserProfileModel.PageMode.Contacts, HasError = true,
ErrorMessage = error1?.Message ?? error2?.Message });
            }
            return base.RedirectToRoute("profile", new UserProfileModel()
            { Mode = UserProfileModel.PageMode.Contacts, HasError = false, ErrorMessage =
"Связь установлена" });
        }

        [HttpPostAttribute, RouteAttribute("textmessage", Name = "textmessage")]
        public async Task<IActionResult> TextMessage([FromForm] string text, int friend)
        {
            if(text == null) return base.RedirectToRoute("buildcontact", new
UserContactsModel() { SelectedContact = friend });

            var profileId =
int.Parse(this.HttpContext.User.FindFirst(ClaimTypes.PrimarySid)!.Value);
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                var friendRecord = await dbcontext.Friends.FirstAsync(item =>
(item.Contactid1 == profileId
                    && item.Contactid2 == friend) || (item.Contactid2 == profileId &&
item.Contactid1 == friend));

                dbcontext.Messages.AddRange(new DAModels::Message()
                {
                    Sendtime = DateTime.Now, Messagebody = text, Friendid =
friendRecord.Friendid, Contactid = profileId
                });
                await dbcontext.SaveChangesAsync();
```

```
            }
            return base.RedirectToRoute("buildcontact", new UserContactsModel()
{ SelectedContact = friend });
        }
    }
}
```

Файл «UserProfileController.cs»

```
using ClientApplication.Filters;
using ClientApplication.Services;
using ClientApplication.ViewModels;
using DatabaseAccess;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Query;
using Newtonsoft.Json;
using Npgsql.EntityFrameworkCore.PostgreSQL.Storage.Internal.Mapping;
using System.Collections.Immutable;
using System.Linq;
using System.Security.Claims;
using System.Text.RegularExpressions;

namespace ClientApplication.Controllers
{
    using DAModels = DatabaseAccess.Models;
    using ControllerLogger = ILogger<UserProfileController>;

    [ControllerAttribute, RouteAttribute(ControllerRoute), AuthorizeAttribute(Policy =
"DefaultUser")]
    public partial class UserProfileController : Controller
    {
        public const string ControllerRoute = "user";
        protected Services.IDatabaseContact DatabaseContact { get; private set; } =
default!;

        protected readonly ILogger<UserProfileController> _logger = default!;
        public UserProfileController(Services.IDatabaseContact dbcontact,
ControllerLogger logger) : base()
        { (this.DatabaseContact, this._logger) = (dbcontact, logger); }

        [HttpGetAttribute, RouteAttribute("profile", Name = "profile")]
        public async Task<IActionResult> ProfileInfo(UserProfileModel? model)
        {
            var authorizatedProfile =
this.HttpContext.User.FindFirst(ClaimTypes.PrimarySid)!;
            this.ViewBag.IsAdmin = HttpContext.User.FindAll(ClaimTypes.Role).Where(item
=> item.Value == "Admin").Count() > 0;
            if (model == null) model = new UserProfileModel();

            model.SearchQuery = model.SearchQuery == default ? default! :
model.SearchQuery.Replace('+', ' ').Trim();
            model.FormRequestLink =
$"{UserProfileController.ControllerRoute}/profileupdate";
            model.Contact = (await
this.DatabaseContact.GetContact(int.Parse(authorizatedProfile.Value),
model.SearchQuery))!;

            if (model.Contact == null) { return base.LocalRedirect("/logout"); }
            model.Contact.FriendContactid1Navigations =
model.Contact.FriendContactid1Navigations
                .Where(delegate (DAModels::Friend record)
            {
                var friendContact = record.Contactid2Navigation;
                return model.ProfileType switch
```

```
                {
                    "Все контакты" => true, "Созданные" => friendContact.Authorization ==
null,
                    "Аккаунты" => friendContact.Authorization != null,  _ => true,
                };
            }).ToImmutableList();
            model.PagesCount =
(int)Math.Ceiling(model.Contact.FriendContactid1Navigations.Count()
                / (double)model.RecordOnPage);

            DAModels::Contact GetContact(DAModels::Friend record) =>
record.Contactid2Navigation;
            model.Contact.FriendContactid1Navigations = (model.SortingType switch
            {
                "Без сортировки" =>
                    model.Contact.FriendContactid1Navigations.OrderBy(item =>
GetContact(item).Contactid),
                "По дате изменения" =>
                    model.Contact.FriendContactid1Navigations.OrderBy(item =>
GetContact(item).Lastupdate.ToString()),
                "По дате рождения" =>
                    model.Contact.FriendContactid1Navigations.OrderBy(item =>
GetContact(item).Birthday.ToString()),

                "По имени" => model.Contact.FriendContactid1Navigations.OrderBy(item =>
GetContact(item).Name),
                _ => model.Contact.FriendContactid1Navigations.OrderBy(item =>
GetContact(item).Contactid)
            })
            .Skip(model.CurrentPage *
model.RecordOnPage).Take(model.RecordOnPage).ToImmutableList();

            string modelSerialize = JsonConvert.SerializeObject(model,
                 new JsonSerializerSettings() { ReferenceLoopHandling =
ReferenceLoopHandling.Ignore });
            return
base.View(JsonConvert.DeserializeObject<UserProfileModel>(modelSerialize));
        }

        [HttpPostAttribute, RouteAttribute("profileupdate", Name = "profileupdate")]
        [ProfileModelFilter("profile")]
        public async Task<IActionResult> ProfileInfo([FromForm] DAModels::Contact
contactModel,
            [FromServices] IEmailTransfer email)
        {
            if (contactModel is null) return base.RedirectToAction("ProfileInfo",
"UserProfile");
            var errorMessage = await this.DatabaseContact.EditContact(contactModel);

            if(errorMessage != null) return base.RedirectToRoute("profile", new
UserProfileModel()
            {
                HasError = true, ErrorMessage = errorMessage.Message, Mode =
UserProfileModel.PageMode.Settings
            });
            await email.SendMessage(IEmailTransfer.MessageType.Update,
contactModel.Emailaddress);
            return base.RedirectToAction("ProfileInfo", "UserProfile", new
UserProfileModel()
            { Mode = UserProfileModel.PageMode.Settings });
        }

        [HttpGetAttribute, RouteAttribute("deletecontact/{contactid}", Name =
"deletecontact")]
```

```
        public async Task<IActionResult> RemoveContact(int contactid, [FromServices]
IEmailTransfer email)
        {
            IDatabaseContact.ErrorStatus? errorMessage = default!;
            var emailName = (await this.DatabaseContact.GetContact(contactid,
string.Empty))?.Emailaddress;
            if(emailName == null) return base.RedirectToRoute("profile");
            try {
                if ((errorMessage = await
this.DatabaseContact.RemoveContact(contactid)) != null)
                { base.RedirectToRoute("profile", new UserProfileModel() { ErrorMessage =
errorMessage.Message }); }
            }
            catch(System.Exception error)
            {
                return base.RedirectToRoute("profile", new UserProfileModel()
{ ErrorMessage = error.Message });
            }
            await email.SendMessage(IEmailTransfer.MessageType.Deleted, emailName);
            return base.RedirectToRoute("profile", new UserProfileModel() { });
        }

        [HttpGetAttribute, RouteAttribute("getdocument", Name = "getdocument")]
        public async Task<IActionResult> GetDocument([FromServices] IDocumentContact
generator)
        {
            var authorizationId =
int.Parse(this.HttpContext.User.FindFirstValue(ClaimTypes.PrimarySid));
            var profileModel = await this.DatabaseContact.GetContact(authorizationId,
string.Empty);
            if (profileModel == null) throw new Exception("При получении отчета: не
найден профиль");

            var friendsList = profileModel.FriendContactid1Navigations.Select(item =>
item.Contactid2Navigation);
            return base.File(await generator.GetDocument(friendsList.ToList()),
"application/octet-stream", "report.xlsx");
        }
    }
}
```

Файл «ProfileModelFilter.cs»

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using System.Text.RegularExpressions;

namespace ClientApplication.Filters
{
    public partial class ProfileModelFilter : System.Attribute, IActionFilter
    {
        protected readonly string routeToRedirect = default!;
        public ProfileModelFilter(string routeToRedirect) : base() { this.routeToRedirect
= routeToRedirect; }
        public virtual void OnActionExecuting(ActionExecutingContext context)
        {
            var idValue = context.HttpContext.Request.Form["id"].FirstOrDefault();
            if (context.HttpContext.Request.Form["validate"].FirstOrDefault() == null)
return;
            var phoneValue = context.HttpContext.Request.Form["phone"].FirstOrDefault();
            var emailValue = context.HttpContext.Request.Form["email"].FirstOrDefault();

            var surnameValue =
context.HttpContext.Request.Form["surname"].FirstOrDefault();
            var nameValue = context.HttpContext.Request.Form["name"].FirstOrDefault();
            var streetValue = context.HttpContext.Request.Form["street"].FirstOrDefault();
```

```
            var errorModel = new ViewModels.UserBaseModel()
            {
                HasError = true, Mode = ViewModels.UserProfileModel.PageMode.Settings,
ErrorMessage = string.Empty,
                SelectedContact = int.Parse(idValue ?? "0")
            };
            foreach (var propertyValue in new[] { surnameValue, nameValue })
            {
                if (propertyValue != null && Regex.IsMatch(propertyValue, @"[А-Яа-
я\w]{4,}")) continue;

                errorModel.ErrorMessage = $"Введено неверное значение ФИО: " +
                        $"{propertyValue?.Substring(0, propertyValue.Length > 20 ? 20 :
propertyValue.Length)}";

                Console.WriteLine($"\nerrorModel: {errorModel.ErrorMessage}");

                context.Result = new RedirectToRouteResult(this.routeToRedirect,
errorModel); return;
            }
            if (streetValue != null && !Regex.IsMatch(streetValue, @"[А-Яа-я\w]{5,}"))
            {
                errorModel.ErrorMessage = $"Введено неверное значение улицы:
{streetValue}";
                context.Result = new RedirectToRouteResult(this.routeToRedirect,
errorModel); return;
            }
            var emailMatch = emailValue != null && Regex.IsMatch(emailValue,
@"^\w{6,}@(mail|gmail|yandex){1}.(ru|com){1}$");
            var phoneMatch = phoneValue != null && Regex.IsMatch(phoneValue, @"^\+7[0-
9]{10}$");

            if (!emailMatch || (!phoneMatch && phoneValue != null))
            {
                errorModel.ErrorMessage = $"Введено неверное значение контактов:
{(!emailMatch ? emailValue : phoneValue)}";
                context.Result = new RedirectToRouteResult(this.routeToRedirect,
errorModel); return;
            }
        }
        public virtual void OnActionExecuted(ActionExecutedContext context) { }
    }
}
```

Файл «ContactModelBinder.cs»

```
using Microsoft.AspNetCore.Mvc.ModelBinding;
using Microsoft.AspNetCore.Mvc.ModelBinding.Binders;
using System.Reflection;
using System.Runtime.InteropServices;

namespace ClientApplication.Infrastructure
{
    using DAModels = DatabaseAccess.Models;
    public abstract class BaseModelBinder<TModel, TBinder> : object, IModelBinder where
TModel : new()
    {
        protected ILogger<TBinder> Logger { get; private set; } = default!;
        public TModel BindingResult { get; protected set; } = new();
        public BaseModelBinder(ILogger<TBinder> logger) : base() { this.Logger = logger; }
        public abstract Task BindModelAsync(ModelBindingContext bindingContext);
    }

    public partial class ContractModelBinder : BaseModelBinder<DAModels::Contact,
ContractModelBinder>, IModelBinder
```

```
    {
        protected IModelBinder FallbackBinder { get; private set; } = default!;

        private readonly BaseModelBinder<List<DAModels::Hobby>, HobbyModelBinder>
hobbyBinder = default!;
        private readonly BaseModelBinder<List<DAModels::Humanquality>,
QualityModelBinder> qualityBinder = default!;

        private readonly BaseModelBinder<List<DAModels::Employee>, EmployeeModelBinder>
employeeBinder = default!;
        public ContractModelBinder(IModelBinder fallbackBinder,
ILogger<ContractModelBinder> logger)
            : base(logger) { this.FallbackBinder = fallbackBinder; }

        public ContractModelBinder(IModelBinder fallbackBinder, ILoggerFactory factory)
            : base(factory.CreateLogger<ContractModelBinder>())
        {
            this.qualityBinder = new
QualityModelBinder(factory.CreateLogger<QualityModelBinder>());
            this.hobbyBinder = new
HobbyModelBinder(factory.CreateLogger<HobbyModelBinder>());

            this.employeeBinder = new
EmployeeModelBinder(factory.CreateLogger<EmployeeModelBinder>());
            this.FallbackBinder = fallbackBinder;
        }
        public sealed class ContractModelBinderProvider : System.Object,
IModelBinderProvider
        {
            public IModelBinder? GetBinder(ModelBinderProviderContext context)
            {
                var loggerFactory = context.Services.GetService<ILoggerFactory>();
                var fallbackBinder = new SimpleTypeModelBinder(typeof(DAModels::Contact),
loggerFactory!);

                var modelBinder = new ContractModelBinder(fallbackBinder, loggerFactory!);
                return context.Metadata.ModelType == typeof(DAModels::Contact) ?
modelBinder : null;
            }
            public ContractModelBinderProvider() : base() { }
        }
        public override Task BindModelAsync(ModelBindingContext bindingContext)
        {
            ValueProviderResult GetProvider(string name) =>
bindingContext.ValueProvider.GetValue(name);
            string GetValue(string name) => GetProvider(name).FirstValue!;
            foreach (var item in new[] {"email", "gender", "birthday", "surname", "name",
"patronymic", "family", "id" })
            {
                if (GetProvider(item) == ValueProviderResult.None) return
this.FallbackBinder.BindModelAsync(bindingContext);
            }
            if (!DateTime.TryParse(GetProvider("birthday").FirstValue, out var
birthdayValue)
                || !int.TryParse(GetProvider("id").FirstValue, out var idValue))
            {
                return Task.FromResult(bindingContext.Result =
ModelBindingResult.Failed());
            }
            this.employeeBinder.BindModelAsync(bindingContext).Wait();
            this.BindingResult = new DAModels::Contact()
            {
                Surname = GetValue("surname"), Name = GetValue("name"), Patronymic =
GetValue("patronymic"),
                Birthday = birthdayValue, Emailaddress = GetValue("email"),
```

```
                Contactid = idValue, Employees = this.employeeBinder.BindingResult,
Familystatus = GetValue("family"),
                Gendertype = new DAModels::Gendertype() { Gendertypename =
GetValue("gender") },
            };
            this.BindingResult.Phonenumber = (GetProvider("phone") ==
ValueProviderResult.None) ? null : GetValue("phone");
            this.Logger.LogInformation("Contact binder create model");

            this.BindingResult.Userpicture = (GetProvider("picture") ==
ValueProviderResult.None) ? null
                : new DAModels::Userpicture() { Filepath = GetValue("picture") };
            var locationModelState = true;
            foreach(var locationItem in new string[] { "country", "city", "street" })
            {
                if (GetProvider(locationItem) == ValueProviderResult.None)
{ locationModelState = false; break; }
            }
            this.BindingResult.Location = (locationModelState == default) ? null : new
DAModels::Location()
            {
                City = new DAModels::City() { Cityname = GetValue("city"), Country =
GetValue("country") },
                Street = GetValue("street")
            };
            this.hobbyBinder.BindModelAsync(bindingContext).Wait();
            this.BindingResult.Hobbies = this.hobbyBinder.BindingResult;

            this.qualityBinder.BindModelAsync(bindingContext).Wait();
            this.BindingResult.Humanqualities = this.qualityBinder.BindingResult;

            return Task.FromResult(bindingContext.Result =
ModelBindingResult.Success(this.BindingResult));
        }
    }
}
```

Файл «EmployeeModelBinder.cs»

```
using Microsoft.AspNetCore.Mvc.ModelBinding;
using Microsoft.EntityFrameworkCore.Query.Internal;
using Npgsql.EntityFrameworkCore.PostgreSQL.Storage.Internal.Mapping;

namespace ClientApplication.Infrastructure
{
    using Employees = List<DatabaseAccess.Models.Employee>;
    using DAModels = DatabaseAccess.Models;
    public partial class EmployeeModelBinder : BaseModelBinder<Employees,
EmployeeModelBinder>, IModelBinder
    {
        public EmployeeModelBinder(ILogger<EmployeeModelBinder> logger) : base(logger) { }
        public sealed class EmployeeModelBinderProvider : System.Object,
IModelBinderProvider
        {
            public EmployeeModelBinderProvider() : base() { }
            public IModelBinder? GetBinder(ModelBinderProviderContext context)
            {
                var loggerFactory = context.Services.GetService<ILoggerFactory>();
                var modelBinder = new
EmployeeModelBinder(loggerFactory!.CreateLogger<EmployeeModelBinder>());

                return context.Metadata.BinderType == typeof(Employees) ? modelBinder :
null;
            }
        }
```

```
        public override Task BindModelAsync(ModelBindingContext bindingContext)
        {
            var employeeLength = bindingContext.ValueProvider.GetValue("employee_length");
            if (employeeLength == ValueProviderResult.None
|| !int.TryParse(employeeLength.FirstValue, out var length))
            {
                return Task.FromResult(bindingContext.Result =
ModelBindingResult.Failed());
            }
            this.BindingResult.Clear();
            for (var index = default(int); index < length; index++)
            {
                var employeeValues = new Dictionary<string, ValueProviderResult>()
                {
                    ["company"] =
bindingContext.ValueProvider.GetValue($"employee[{index}].company"),
                    ["post"] =
bindingContext.ValueProvider.GetValue($"employee[{index}].post"),

                    ["status"] =
bindingContext.ValueProvider.GetValue($"employee[{index}].status"),
                };
                foreach(KeyValuePair<string, ValueProviderResult> providerResult in
employeeValues)
                {
                    if(providerResult.Value == ValueProviderResult.None)
                    { return Task.FromResult(bindingContext.Result =
ModelBindingResult.Failed()); }
                }
                this.BindingResult.Add(new DAModels::Employee()
                {
                    Companyname = employeeValues["company"].FirstValue!, Status =
employeeValues["status"].FirstValue!,
                    Post = new DAModels.Post() { Postname =
employeeValues["post"].FirstValue! }
                });
            }
            bindingContext.Result = ModelBindingResult.Success(this.BindingResult);
            return Task.CompletedTask;
        }
    }
}
```

Файл «HobbyModelBinder.cs»

```
using DatabaseAccess.Models;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using System.Threading.Tasks;

namespace ClientApplication.Infrastructure
{
    using Hobbies = List<DatabaseAccess.Models.Hobby>;
    using DAModels = DatabaseAccess.Models;
    public partial class HobbyModelBinder : BaseModelBinder<Hobbies, HobbyModelBinder>,
IModelBinder
    {
        public HobbyModelBinder(ILogger<HobbyModelBinder> logger) : base(logger) { }
        public sealed class HobbyModelBinderProvider : System.Object,
IModelBinderProvider
        {
            public HobbyModelBinderProvider() : base() { }
            public IModelBinder? GetBinder(ModelBinderProviderContext context)
            {
                var loggerFactory = context.Services.GetService<ILoggerFactory>();
                var modelBinder = new
QualityModelBinder(loggerFactory!.CreateLogger<QualityModelBinder>());
```

```
                return context.Metadata.BinderType == typeof(Hobbies) ? modelBinder :
null;
            }
        }
        public override Task BindModelAsync(ModelBindingContext bindingContext)
        {
            var hobbyLength = bindingContext.ValueProvider.GetValue("hobby_length");
            if (hobbyLength == ValueProviderResult.None
|| !int.TryParse(hobbyLength.FirstValue, out var length))
            {
                return Task.FromResult(bindingContext.Result =
ModelBindingResult.Failed());
            }
            this.BindingResult.Clear();
            for (var index = default(int); index < length; index++)
            {
                var hobbyValues =
bindingContext.ValueProvider.GetValue($"hobby[{index}]");
                if (hobbyValues == ValueProviderResult.None)
                {
                    return Task.FromResult(bindingContext.Result =
ModelBindingResult.Success(this.BindingResult));
                }
                this.BindingResult.Add(new Hobby() { Hobbyname =
hobbyValues.FirstValue! });
            }
            return Task.FromResult(bindingContext.Result =
ModelBindingResult.Success(this.BindingResult));
        }
    }
}
```

Файл «QualityModelBinder.cs»

```
using DatabaseAccess.Models;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using System.Threading.Tasks;

namespace ClientApplication.Infrastructure
{
    using Qualities = List<DatabaseAccess.Models.Humanquality>;
    using DAModels = DatabaseAccess.Models;
    public partial class QualityModelBinder : BaseModelBinder<Qualities,
QualityModelBinder>, IModelBinder
    {
        public QualityModelBinder(ILogger<QualityModelBinder> logger) : base(logger) { }
        public sealed class QualityModelBinderProvider : System.Object,
IModelBinderProvider
        {
            public QualityModelBinderProvider() : base() { }
            public IModelBinder? GetBinder(ModelBinderProviderContext context)
            {
                var loggerFactory = context.Services.GetService<ILoggerFactory>();
                var modelBinder = new
QualityModelBinder(loggerFactory!.CreateLogger<QualityModelBinder>());

                return context.Metadata.BinderType == typeof(Qualities) ? modelBinder :
null;
            }
        }
        public override Task BindModelAsync(ModelBindingContext bindingContext)
        {
            var qualityLength = bindingContext.ValueProvider.GetValue("quality_length");
            if (qualityLength == ValueProviderResult.None
|| !int.TryParse(qualityLength.FirstValue, out var length))
```

```
            {
                return Task.FromResult(bindingContext.Result =
ModelBindingResult.Failed());
            }
            this.BindingResult.Clear();
            for (var index = default(int); index < length; index++)
            {
                var employeeValues =
bindingContext.ValueProvider.GetValue($"quality[{index}]");
                if (employeeValues == ValueProviderResult.None)
                {
                    return Task.FromResult(bindingContext.Result =
ModelBindingResult.Success(this.BindingResult));
                }
                this.BindingResult.Add(new DAModels::Humanquality() { Qualityname =
employeeValues.FirstValue! });
            }
            return Task.FromResult(bindingContext.Result =
ModelBindingResult.Success(this.BindingResult));
        }
    }
}
```

Файл «CheckProfileMiddleware.cs»

```
using DatabaseAccess;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;

namespace ClientApplication.Middleware
{
    public static class CheckProfileExtensions : System.Object
    {
        public static IApplicationBuilder UseCheckProfile(this IApplicationBuilder
builder)
        {
            return builder.UseMiddleware<Middleware.CheckProfileMiddleware>();
        }
    }
    public partial class CheckProfileMiddleware : System.Object
    {
        protected readonly RequestDelegate requestDelegate = default!;
        protected IDbContextFactory<DatabaseContext> DatabaseFactory { get; set; } =
default!;

        public CheckProfileMiddleware(RequestDelegate next,
IDbContextFactory<DatabaseContext> factory)
            : base() { (this.requestDelegate, this.DatabaseFactory) = (next, factory); }

        public async Task InvokeAsync(HttpContext context,
ILogger<CheckProfileMiddleware> logger)
        {
            var profileId = context.User.FindFirst(ClaimTypes.PrimarySid);
            if (profileId == null || !int.TryParse(profileId.Value, out var
profileIdValue))
            {
                await this.requestDelegate.Invoke(context); return;
            }
            logger.LogWarning($"Check profileId: {profileId.Value}");
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                if(dbcontext.Contacts.Find(profileIdValue) == null)
                {
                    logger.LogWarning($"Remove cookie profileId: {profileId.Value}");
```

```
                    await
context.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
                    context.Response.Redirect("/authorization", true); return;
                }
            }
            await this.requestDelegate.Invoke(context);
        }
    }
}
```

Файл «ContactMeesagerHub.cs»

```
using DatabaseAccess;
using DatabaseAccess.Models;
using Microsoft.AspNetCore.SignalR;
using Microsoft.EntityFrameworkCore;
using System.Text.RegularExpressions;

namespace ClientApplication.Middleware
{
    public partial class ContactMessagerHub : Hub, IAsyncDisposable
    {
        public record class ContactMessagerModel(Message Message, string Group);
        protected IDbContextFactory<DatabaseContext> DatabaseFactory { get; set; } =
default!;
        public ContactMessagerHub(IDbContextFactory<DatabaseContext> factory) : base()
        { this.DatabaseFactory = factory; }

        public async Task JoinGroup(string groupName)
        {
            await this.Groups.AddToGroupAsync(this.Context.ConnectionId, groupName);
            await Clients.All.SendAsync("Notify", $"{Context.ConnectionId} вошел в чат в
группу {groupName}");
        }
        public async Task SendMessage(ContactMessagerModel messageModel)
        {
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                await dbcontext.Messages.AddAsync(messageModel.Message);
                await dbcontext.SaveChangesAsync();
            }
            await this.Clients.Group(messageModel.Group).SendAsync("GetMessage",
messageModel.Message);
        }
        public override Task OnConnectedAsync() => base.OnConnectedAsync();
        public override Task OnDisconnectedAsync(Exception? exception) =>
base.OnDisconnectedAsync(exception);

        public ValueTask DisposeAsync() => ValueTask.CompletedTask;
    }
}
```

Файл «DatabaseContact.cs»

```
using ClientApplication.Controllers;
using ClientApplication.ViewModels;
using DatabaseAccess;
using DatabaseAccess.Models;
using Microsoft.EntityFrameworkCore;
using System.Collections.Immutable;
using System.Text.RegularExpressions;

namespace ClientApplication.Services
{
    using DatabaseContactLogger = ILogger<UserContactsController>;
    using DAModels = DatabaseAccess.Models;
```

```
    public interface IDatabaseContact : System.IDisposable
    {
        public abstract Task<IDatabaseContact.ErrorStatus?> SetFriendShip(int contactId1,
int contactId2,
            DAModels::Datingtype datingtype);
        public abstract Task<DAModels::Contact?> GetContact(int contactId, string search);

        public abstract Task<IDatabaseContact.ErrorStatus?> EditContact(DAModels::Contact
contactModel);
        public abstract Task<IDatabaseContact.ErrorStatus?> AddContact(DAModels::Contact
contactModel,
            int connectId, DAModels::Datingtype datingtype);

        public abstract Task<DAModels::Contact> InitializeContact();
        public abstract Task<IDatabaseContact.ErrorStatus?> RemoveContact(int contactId);
        public record class ErrorStatus(string Message);
    }
    public partial class DatabaseContact : System.Object, Services.IDatabaseContact
    {
        protected IDbContextFactory<DatabaseContext> DatabaseFactory { get; private set; }
= default!;

        protected readonly DatabaseContactLogger Logger = default!;
        public DatabaseContact(IDbContextFactory<DatabaseContext> factory,
DatabaseContactLogger logger)
            : base() { this.DatabaseFactory = factory; this.Logger = logger; }

        public void Dispose() { this.Logger.LogInformation("DatabaseContact Disposed"); }
        public async Task<IDatabaseContact.ErrorStatus?> EditContact(DAModels::Contact
contactModel)
        {
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                var phoneValue = contactModel.Phonenumber == string.Empty ? null :
contactModel.Phonenumber;
                dbcontext.Contacts.Where(record => record.Contactid ==
contactModel.Contactid).ExecuteUpdate(prop => prop
                    .SetProperty(item => item.Surname, item => contactModel.Surname)
                    .SetProperty(item => item.Name, item => contactModel.Name)
                    .SetProperty(item => item.Patronymic, item => contactModel.Patronymic)
                    .SetProperty(item => item.Birthday, item => contactModel.Birthday)

                    .SetProperty(item => item.Emailaddress, item =>
contactModel.Emailaddress)
                    .SetProperty(item => item.Lastupdate, item => DateTime.Now)
                    .SetProperty(item => item.Phonenumber, item => phoneValue)
                    .SetProperty(item => item.Familystatus, item =>
contactModel.Familystatus));

                var userContact = await dbcontext.Contacts.Include(prop =>
prop.Employees).Include(prop => prop.Hobbies)
                    .Include(prop => prop.Humanqualities).FirstAsync(item =>
item.Contactid == contactModel.Contactid);
                if (userContact == null) return new IDatabaseContact.ErrorStatus("Запись
о контакте не найдена");

                userContact.Gendertype = (await
dbcontext.Gendertypes.Where((DAModels::Gendertype item)
                    => item.Gendertypename ==
contactModel.Gendertype.Gendertypename).FirstOrDefaultAsync())!;

                userContact.Userpicture = (await
dbcontext.Userpictures.Where((DAModels::Userpicture item)
                    => item.Filepath ==
contactModel.Userpicture!.Filepath).FirstOrDefaultAsync());
```

```
                var contactLocation = await
dbcontext.Locations.FindAsync(userContact.Locationid ?? 0);
                if (contactModel.Location != null)
                {
                    var contactCity = await dbcontext.Cities.FirstAsync((DAModels::City
city) => city.Cityname
                        == contactModel.Location.City.Cityname);
                    var newLocation = new DAModels::Location() { City = contactCity,
Street = contactModel.Location.Street };

                    if (contactLocation == null) { await
dbcontext.AddAsync(userContact.Location = newLocation); }
                    else { contactLocation.Street = newLocation.Street;
contactLocation.City = contactCity; }
                }
                else if(userContact.Locationid != null)
                {
                    await dbcontext.Locations.Where((DAModels::Location item) =>
item.Locationid ==
                        (userContact.Locationid)).ExecuteDeleteAsync();

                    userContact.Locationid = null; userContact.Location = default!;
                }
                await dbcontext.SaveChangesAsync();

                var result = await dbcontext.Employees.Where((DAModels::Employee item) =>
item.Contactid ==
                    contactModel.Contactid).ExecuteDeleteAsync();
                await dbcontext.SaveChangesAsync();
                foreach (DAModels::Employee employee in contactModel.Employees)
                {
                    var contactPost = await dbcontext.Posts.FirstAsync(city =>
city.Postname == employee.Post!.Postname);
                    var newEmployee = new DAModels::Employee()
                    {
                        Contactid = userContact.Contactid, Post = contactPost, Status =
employee.Status,
                        Companyname = employee.Companyname,
                    };
                    dbcontext.Employees.Add(newEmployee);
                }
                userContact.Hobbies.Clear(); userContact.Humanqualities.Clear();
                foreach (DAModels::Hobby hobby in contactModel.Hobbies)
                {
                    userContact.Hobbies.Add(await dbcontext.Hobbies.FirstAsync(item =>
item.Hobbyname == hobby.Hobbyname));
                }
                dbcontext.Contacts.Update(userContact);
                foreach (DAModels::Humanquality hobby in contactModel.Humanqualities)
                {
                    userContact.Humanqualities.Add(
                        await dbcontext.Humanqualities.FirstAsync(item =>
item.Qualityname == hobby.Qualityname));
                }
                dbcontext.Contacts.Update(userContact); await
dbcontext.SaveChangesAsync();
            }
            return default(IDatabaseContact.ErrorStatus);
        }

public async Task<IDatabaseContact.ErrorStatus?> AddContact(DAModels::Contact
contactModel,
            int connectId, DAModels::Datingtype datingtype)
        {
```

```
            IDatabaseContact.ErrorStatus? errorMessage =
default(IDatabaseContact.ErrorStatus);
            var contactRecord = new Contact() { Surname = string.Empty, Name =
string.Empty, Emailaddress = string.Empty };

            contactModel.Lastupdate = DateTime.Now;
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                contactRecord.Gendertype = await dbcontext.Gendertypes.FirstAsync();
                contactRecord.Userpicture = await dbcontext.Userpictures.FirstAsync();

                await dbcontext.Contacts.AddAsync(contactRecord); await
dbcontext.SaveChangesAsync();
                contactModel.Contactid = contactRecord.Contactid;
            }
            if ((errorMessage = await this.EditContact(contactModel)) != null) return
errorMessage;
            var contactId = contactModel.Contactid;

            if ((errorMessage = await this.SetFriendShip(connectId, contactId,
datingtype)) != null) return errorMessage;
            return default(IDatabaseContact.ErrorStatus);
        }

        public async Task<IDatabaseContact.ErrorStatus?> RemoveContact(int contactId)
        {
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                var result = await dbcontext.Contacts.Where(item => item.Contactid ==
contactId).ExecuteDeleteAsync();
                if (result <= 0) return new IDatabaseContact.ErrorStatus("Данные не были
удалены");
            }
            return default(IDatabaseContact.ErrorStatus);
        }

        public async Task<IDatabaseContact.ErrorStatus?> SetFriendShip(int contactId1,
int contactId2,
            DAModels::Datingtype datingtype)
        {
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                var profile1 = await dbcontext.Contacts.Include(item =>
item.FriendContactid1Navigations)
                    .FirstOrDefaultAsync(item => item.Contactid == contactId1);

                var profile2 = await dbcontext.Contacts.Include(item =>
item.FriendContactid1Navigations)
                    .FirstOrDefaultAsync(item => item.Contactid == contactId2);
                if(profile1 == null || profile2 == null) return new
IDatabaseContact.ErrorStatus("Контакт не найден");

                var collision1 = profile1.FriendContactid1Navigations.Where(item =>
item.Contactid2 == profile2.Contactid);
                var collision2 = profile1.FriendContactid2Navigations.Where(item =>
item.Contactid1 == profile2.Contactid);

                var datingRecord = await dbcontext.Datingtypes.FirstOrDefaultAsync(item
=> item.Typeofdating ==
                    datingtype.Typeofdating);
                if (datingRecord == null || (profile1.Contactid == profile2.Contactid))
                {
                    return new IDatabaseContact.ErrorStatus("Связь не возможно
установить");
                }
```

```
                var resultModel = new DAModels::Friend()
                {
                    Contactid1 = contactId1, Contactid2 = contactId2, Datingtype =
datingRecord,
                    Starttime = DateOnly.FromDateTime(DateTime.Now)
                };
                if (collision1.Count() > 0 || collision2.Count() > 0)
                {
                    resultModel = (await dbcontext.Friends.FirstAsync(item =>
(item.Contactid1 == contactId1
                        && item.Contactid2 == contactId2)
                        || (item.Contactid1 == contactId2 && item.Contactid2 ==
contactId1)));

                    resultModel.Datingtypeid = datingRecord.Datingtypeid;
                    dbcontext.Friends.Update(resultModel); await
dbcontext.SaveChangesAsync();
                    return default(IDatabaseContact.ErrorStatus);
                }
                else await dbcontext.Friends.AddAsync(resultModel); await
dbcontext.SaveChangesAsync();
            }
            return default(IDatabaseContact.ErrorStatus);
        }

        public async Task<DAModels::Contact?> GetContact(int contactId, string
searchQuery)
        {
            DAModels::Contact contactModel = default!;
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                contactModel = (await dbcontext.Contacts.Include(prop => prop.Gendertype)
                    .Include(prop => prop.Location).ThenInclude(prop => prop!.City)
                    .Include(prop => prop.Employees).ThenInclude(prop => prop!.Post)
                    .Include(prop => prop.FriendContactid1Navigations).ThenInclude(prop
=> prop.Contactid2Navigation)
                        .ThenInclude(prop => prop.Authorization)
                    .Include(prop => prop.FriendContactid2Navigations).ThenInclude(prop
=> prop.Contactid1Navigation)
                        .ThenInclude(prop => prop.Authorization)

                    .Include(prop => prop.Userpicture).Include(prop => prop.Authorization)
                    .Include(prop => prop.Humanqualities).Include(prop => prop.Hobbies)
                    .Where(item => item.Contactid == contactId).FirstOrDefaultAsync())!;

                if (contactModel == null) { return default(DAModels::Contact); }
                await dbcontext.Datingtypes.LoadAsync();
                await dbcontext.Gendertypes.LoadAsync();

                contactModel.FriendContactid2Navigations.Select(item => new
DAModels::Friend()
                {
                    Contactid2Navigation = item.Contactid1Navigation, Starttime =
item.Starttime,
                    Datingtype = item.Datingtype, Friendid = item.Friendid
                })
                .ToImmutableList().ForEach(item =>
contactModel.FriendContactid1Navigations.Add(item));
                foreach (DAModels::Friend record in
contactModel.FriendContactid1Navigations)
                {
                    await dbcontext.Userpictures.Where((DAModels::Userpicture item) =>
item.Userpictureid ==
                        record.Contactid2Navigation.Userpictureid).LoadAsync();
                }
```

```
                contactModel.FriendContactid1Navigations =
contactModel.FriendContactid1Navigations
                    .Where(record =>
SelectionExpression(record.Contactid2Navigation)).ToList();
            }
            bool SelectionExpression(DAModels::Contact record)
            {
                return searchQuery == default || Regex.IsMatch(record.Emailaddress,
searchQuery)
                    || Regex.IsMatch($"{record.Name} {record.Surname}", searchQuery)
                    || (record.Phonenumber != null && Regex.IsMatch(record.Phonenumber,
searchQuery)
                );
            }
            return (DAModels::Contact) contactModel;
        }

        public async Task<DAModels::Contact> InitializeContact()
        {
            var resultModel = new DAModels::Contact()
            {
                Birthday = DateTime.Now, Emailaddress = string.Empty, Phonenumber =
string.Empty,
                Surname = string.Empty, Name = string.Empty, Patronymic = string.Empty,
            };
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                resultModel.Gendertype = await dbcontext.Gendertypes.FirstAsync();
                resultModel.Userpicture = await dbcontext.Userpictures.FirstAsync();
            }
            return (DAModels::Contact) resultModel;
        }
    }
}
```

Файл «DocumentContact.cs»

```
using DatabaseAccess.Models;
using Microsoft.AspNetCore.Authorization;
using OfficeOpenXml;

namespace ClientApplication.Services
{
    using DAModels = DatabaseAccess.Models;
    public interface IDocumentContact : System.IDisposable
    {
        public abstract Task<byte[]> GetDocument(List<DAModels::Contact> contacts);
    }
    public partial class DocumentContact : System.Object, IDocumentContact
    {
        protected ILogger<DocumentContact> Logger { get; set; } = default!;
        public DocumentContact(ILogger<DocumentContact> logger) { this.Logger = logger; }
        public void Dispose() => this.Logger.LogInformation("DocumentContact Disposed");

        public async Task<byte[]> GetDocument(List<Contact> contactList)
        {
            using var excelDocument = new ExcelPackage();
            var workSheet = excelDocument.Workbook.Worksheets.Add("Контакты");

            workSheet.DefaultRowHeight = 12;
            workSheet.DefaultColWidth = 20;
            for(var index = 1; index <= 9; index++) workSheet.Column(index).AutoFit();

            workSheet.Cells[1, 1].Value = "Имя"; workSheet.Cells[1, 2].Value = "Фамилия";
            workSheet.Cells[1, 3].Value = "Телефон"; workSheet.Cells[1, 4].Value =
"Email";
```

```
            workSheet.Cells[1, 5].Value = "Пол"; workSheet.Cells[1, 6].Value =
"Изменение";
            workSheet.Cells[1, 7].Value = "День рождения";
            workSheet.Cells[1, 8].Value = "Семейный статус";
            workSheet.Cells[1, 9].Value = "Тип контакта";

            workSheet.Row(1).Style.Font.Bold = true; workSheet.Row(1).Height = 20;
            for(var index = 0; index < contactList.Count; index++)
            {
                workSheet.Cells[index + 2, 1].Value = contactList[index].Name;
                workSheet.Cells[index + 2, 2].Value = contactList[index].Surname;
                workSheet.Cells[index + 2, 3].Value = contactList[index].Phonenumber ??
"Не указано";
                workSheet.Cells[index + 2, 4].Value = contactList[index].Emailaddress;

                workSheet.Cells[index + 2, 5].Value =
contactList[index].Gendertype.Gendertypename;
                workSheet.Cells[index + 2, 6].Value =
contactList[index].Lastupdate.ToString();
                workSheet.Cells[index + 2, 7].Value =
contactList[index].Birthday.ToString();
                workSheet.Cells[index + 2, 8].Value = contactList[index].Familystatus;
                workSheet.Cells[index + 2, 9].Value = contactList[index].Authorization ==
null ?
                    "Созданный контакт" : "Профиль пользователя";
            }
            this.Logger.LogInformation($"Document created\n");
            return await excelDocument.GetAsByteArrayAsync();
        }
    }
}
```

Файл «EmailTransfer.cs»

```
using Microsoft.Extensions.Options;
using System.Net;
using System.Net.Mail;

namespace ClientApplication.Services
{
    public interface IEmailTransfer : System.IAsyncDisposable
    {
        public sealed class EmailAccount : object
        {
            public string Login { get; set; } = default!;
            public string Password { get; set; } = default!;
        }
        public enum MessageType : sbyte { Update, Registration, Deleted }
        public Task SendMessage(string messageText, string address);
        public Task SendMessage(IEmailTransfer.MessageType messageType, string address);
    }

    public partial class EmailTransfer : System.Object, IEmailTransfer
    {
        protected IEmailTransfer.EmailAccount EmailAccount { get; set; } = default!;
        public EmailTransfer(IOptions<IEmailTransfer.EmailAccount> emailAccount) : base()
        { this.EmailAccount = emailAccount.Value; }
        public ValueTask DisposeAsync() => ValueTask.CompletedTask;

        public virtual async Task SendMessage(string messageText, string address)
        {
            var smtpClient = new SmtpClient("smtp.gmail.com", 587)
            {
                Credentials = new NetworkCredential(this.EmailAccount.Login,
this.EmailAccount.Password),
                EnableSsl = true,
```

```
                DeliveryMethod = SmtpDeliveryMethod.Network,
            };
            var transferFrom = new MailAddress(this.EmailAccount.Login, "Записная
книжка");
            using var emailMessage = new MailMessage(transferFrom, new
MailAddress(address))
            {
                Subject = "Информация о контактах", Body = messageText,
            };
            try { await smtpClient.SendMailAsync(emailMessage); }
            catch(System.Exception error) {  }
        }
        public virtual async Task SendMessage(IEmailTransfer.MessageType messageType,
string address)
        {
            await this.SendMessage(messageType switch
            {
                IEmailTransfer.MessageType.Registration => "Вы успешно зарегистрировались
в нашем приложении",
                IEmailTransfer.MessageType.Update => "Данные вашего профиля были
обновлены",
                IEmailTransfer.MessageType.Deleted => "Ваш аккаунт был удалён из системы,
приносим извинения :)",
            _=> "Ошибка" }, address);
        }
    }
}
```

Папка «ViewComponents»

```
using DatabaseAccess;
using DatabaseAccess.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;

namespace ClientApplication.ViewComponents
{
    [ViewComponentAttribute]
    public partial class ContactCardViewComponent : ViewComponent
    {
        protected IDbContextFactory<DatabaseContext> DatabaseFactory { get; set; } =
default!;
        public ContactCardViewComponent(IDbContextFactory<DatabaseContext> factory) :
base()
        { this.DatabaseFactory = factory; }

        public async Task<IViewComponentResult> InvokeAsync(Contact contactModel)
        {
            var profileId =
int.Parse(this.HttpContext.User.FindFirst(ClaimTypes.PrimarySid)!.Value);
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                var friendRecord = dbcontext.Friends.FirstOrDefault(item =>
(item.Contactid2 == profileId
                    && item.Contactid1 == contactModel.Contactid)
                    || (item.Contactid2 == contactModel.Contactid && item.Contactid1 ==
profileId));

                if (friendRecord == null) throw new Exception("Запись о контакте не
найдена");
                await dbcontext.Datingtypes.LoadAsync();
                this.ViewBag.DatingType = friendRecord.Datingtype!.Typeofdating;
            }
            return base.View("ContactCard", contactModel);
        }
```

```
    }
}

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using DatabaseAccess;
using DatabaseAccess.Models;
using System.Collections.Immutable;

namespace ClientApplication.ViewComponents
{
    [ViewComponentAttribute]
    public partial class ContactFriendsViewComponent : ViewComponent
    {
        protected Services.IDatabaseContact DatabaseContact { get; private set; } =
default!;
        public ContactFriendsViewComponent(Services.IDatabaseContact factory) : base()
        { this.DatabaseContact = factory; }
        public async Task<IViewComponentResult> InvokeAsync(int profileId)
        {
            var recordContact = await this.DatabaseContact.GetContact(profileId,
string.Empty);
            return base.View("FriendView", recordContact == null ? new List<Contact>() { }
                : recordContact.FriendContactid1Navigations
                .Select(item => item.Contactid2Navigation).ToList());
        }
    }
}
using DatabaseAccess.Models;
using DatabaseAccess;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;

namespace ClientApplication.ViewComponents
{
    [ViewComponentAttribute]
    public partial class FriendCreatorViewComponent : ViewComponent
    {
        protected IDbContextFactory<DatabaseContext> DatabaseFactory { get; set; } =
default!;
        public FriendCreatorViewComponent(IDbContextFactory<DatabaseContext> factory) :
base()
        { this.DatabaseFactory = factory; }

        public async Task<IViewComponentResult> InvokeAsync()
        {
            List<Datingtype> datingTypes = default!;
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                datingTypes = await dbcontext.Datingtypes.ToListAsync();
            }
            return base.View("FriendCreator", datingTypes);
        }
    }
}
using DatabaseAccess;
using DatabaseAccess.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Security.Claims;

namespace ClientApplication.ViewComponents
```

```
{
    [ViewComponentAttribute]
    public partial class FriendMessagesViewComponent : ViewComponent
    {
        protected IDbContextFactory<DatabaseContext> DatabaseFactory { get; set; } =
default!;
        public FriendMessagesViewComponent(IDbContextFactory<DatabaseContext> factory) :
base()
        { this.DatabaseFactory = factory; }
        public async Task<IViewComponentResult> InvokeAsync(int profileId)
        {
            var currentId =
int.Parse(this.HttpContext.User.FindFirst(ClaimTypes.PrimarySid)!.Value);
            List<Message> viewModel = new();
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                await dbcontext.Contacts.Where(item => item.Contactid ==
profileId).LoadAsync();
                var friendRecord = await dbcontext.Friends.FirstAsync(item =>
(item.Contactid1 == profileId
                    && item.Contactid2 == currentId) || (item.Contactid2 == profileId &&
item.Contactid1 == currentId));

                if (friendRecord == null) return base.View("MessagesView", new
List<Message>() { });
                viewModel = dbcontext.Messages.Where(item => item.Friendid ==
friendRecord.Friendid)
                    .OrderBy(item => item.Sendtime).ToList();
            }
            return base.View("MessagesView", viewModel);
        }
    }
}
using DatabaseAccess;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Query;
using System.Collections.Immutable;

namespace ClientApplication.ViewComponents
{
    [ViewComponentAttribute]
    public partial class GroupbyListViewComponent : ViewComponent
    {
        public record class GroupbyItemModel(string GroupName, int GroupValue);
        public sealed class GroupbyListViewModel : System.Object
        {
            public List<GroupbyItemModel> GenderGroups { get; set; } = new();
            public List<GroupbyItemModel> AccountTypeGroups { get; set; } = new();
            public GroupbyListViewModel() : base() { }
        }
        protected IDbContextFactory<DatabaseContext> DatabaseFactory { get; set; } =
default!;
        public GroupbyListViewComponent(IDbContextFactory<DatabaseContext> factory) :
base()
        { this.DatabaseFactory = factory; }
        public async Task<IViewComponentResult> InvokeAsync()
        {
            var viewModel = new GroupbyListViewModel();
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                var genderGroupRecord = dbcontext.Contacts.Include(item =>
item.Gendertype)
                    .GroupBy(item => item.Gendertype.Gendertypename).Select(item => new
GroupbyItemModel(item.Key, item.Count()))
```

```
                    .ToImmutableList();

                var accountGroupRecord = dbcontext.Contacts.Include(item =>
item.Authorization)
                    .GroupBy(item => item.Authorization == null ? "Созданные контакты" :
"Акканты пользователей")
                    .Select(item => new GroupbyItemModel(item.Key!.ToString()!,
item.Count())).ToImmutableList();

                viewModel.AccountTypeGroups.AddRange(accountGroupRecord);
                viewModel.GenderGroups.AddRange(genderGroupRecord);
            }
            return this.View("GroupbyList", viewModel);
        }
    }
}
using DatabaseAccess;
using DatabaseAccess.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Internal;
using System.Runtime.CompilerServices;
using System.Security.Claims;

namespace ClientApplication.ViewComponents
{
    [ViewComponentAttribute]
    public partial class ProfileIconViewComponent : ViewComponent
    {
        protected IDbContextFactory<DatabaseContext> DatabaseFactory { get; set; } =
default!;
        public ProfileIconViewComponent(IDbContextFactory<DatabaseContext> factory) :
base()
        { this.DatabaseFactory = factory; }

        public async Task<IViewComponentResult> InvokeAsync()
        {
            var profileId =
int.Parse(this.HttpContext.User.FindFirst(ClaimTypes.PrimarySid)!.Value);
            Userpicture? userPicture = default!;
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                userPicture = (await dbcontext.Contacts.Include(item => item.Userpicture)
                    .FirstAsync(item => item.Contactid == profileId)).Userpicture;

                if (userPicture == null) throw new Exception("Изображение профиля не
найдено");
            }
            return base.View("ProfileIcon", userPicture);
        }
    }
}
```

Папка «ViewModels»

```
namespace ClientApplication.ViewModels
{
    using DAModels = DatabaseAccess.Models;
    public partial class AdministratorModel : System.Object
    {
        public List<DAModels::Contact> ContactsList { get; set; } = new();
        public DAModels::Contact Contact { get; set; } = new();
        public System.String SortingType { get; set; } = "Без сортировки";
        public System.String ProfileType { get; set; } = "Все контакты";

        public System.String ErrorMessage { get; set; } = default!;
```

```
        public System.Int32 SelectedRecord { get; set; } = default!;
        public System.String FormRequestLink { get; set; } = default!;
        public AdministratorModel() : base() { }
    }
}
using Microsoft.AspNetCore.Mvc.ModelBinding;
using System.Diagnostics;
using System.Diagnostics.CodeAnalysis;

namespace ClientApplication.ViewModels
{
    public sealed class AuthorizationModel : System.Object
    {
        public enum AuthorizationMode : System.SByte { Login, Registration };

        public System.Boolean HasError { get; set; } = default;
        public System.String? ErrorCause { get; set; } = default!;
        public AuthorizationMode Mode { get; set; } = AuthorizationMode.Login;

        public AuthorizationModel(bool has_error) : base() { this.HasError = has_error; }
        public AuthorizationModel() : this(default(bool)) { }
    }
}
using Microsoft.AspNetCore.Mvc.ModelBinding;

namespace ClientApplication.ViewModels
{
    using static ClientApplication.ViewModels.UserProfileModel;
    using DAModels = DatabaseAccess.Models;
    public partial class UserBaseModel : System.Object
    {
        public enum PageMode : byte { Settings, Contacts, Admin }

        public System.Boolean HasError { get; set; } = default!;
        public System.String? ErrorMessage { get; set; } = default;
        public UserProfileModel.PageMode Mode { get; set; } = PageMode.Contacts;

        public System.Int32 SelectedContact { get; set; } = default!;
        public string FormRequestLink { get; set; } = string.Empty;
        public UserBaseModel() : base() { }
    }

    public partial class UserContactsModel : ViewModels.UserBaseModel
    {
        [BindingBehaviorAttribute(BindingBehavior.Never)]
        public DAModels::Contact Contact { get; set; } = new();
        public DAModels::Datingtype DatingType { get; set; } = default!;

        public System.Boolean IsAccount { get; set; } = false;
        public UserContactsModel() : base() { }
    }
}
namespace ClientApplication.ViewModels
{
    using DAModels = DatabaseAccess.Models;
    public partial class ProfileEditingModel : System.Object
    {
        public List<DAModels::Gendertype> GenderTypes { get; set; } = default!;
        public List<DAModels::Humanquality> QualityTypes { get; set; } = default!;
        public List<DAModels::Hobby> HobbyTypes { get; set; } = default!;

        public List<DAModels::Userpicture> Pictures { get; set; } = default!;
        public List<DAModels::City> Cities { get; set; } = default!;
        public List<DAModels::Post> Postes { get; set; } = default!;
        public List<DAModels::Datingtype> Datingtypes { get; set; } = default!;
```

```
        public ProfileEditingModel() : base() { }
    }
}
namespace ClientApplication.ViewModels
{
    using Microsoft.AspNetCore.Mvc.ModelBinding;
    using System.ComponentModel.DataAnnotations;
    using DAModels = DatabaseAccess.Models;
    public sealed partial class UserProfileModel : ViewModels.UserBaseModel
    {
        [BindingBehaviorAttribute(BindingBehavior.Never)]
        public DAModels::Contact Contact { get; set; } = new();

        public static readonly System.Int32 MaxRecordOnPage = 15, MinRecordOnPage = 4;
        public System.Int32 RecordOnPage { get; set; } = UserProfileModel.MinRecordOnPage;

        public System.String SortingType { get; set; } = "Без сортировки";
        public System.String ProfileType { get; set; } = "Все контакты";
        public System.String SearchQuery { get; set; } = string.Empty;

        public System.Int32 CurrentPage { get; set; } = default;
        public System.Int32 PagesCount { get; set; } = default;

        public UserProfileModel() : base() { }
    }
}
```

Файл «Program.cs»

```
namespace ClientApplication
{
    using Config = ClientApplication.Configurations;
    public sealed class Program : System.Object
    {
        public static void Main(string[] args)
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("[Contact Application has been launched]");

            Console.ForegroundColor = ConsoleColor.White;

            try
{ Config::StartupBase.StartApplication<Config::StartupApplication>(args).Wait(); }
            catch (AggregateException error) when (error.InnerException is
Config::StartupException)
            {
                var startup_error = error.InnerException as Config::StartupException;
                Console.WriteLine($"[{startup_error?.StartupType.Name}]:
{error.Message}");
            }
        }
    }
}
```

Папка «Views»

```
@using ClientApplication.Views.Shared;
@using DatabaseAccess.Models;
@model ClientApplication.ViewModels.AdministratorModel;
@{
    this.ViewBag.Title = "Администратор";

    const UserProfileModel.PageMode settingsMode = UserProfileModel.PageMode.Settings;
    const UserProfileModel.PageMode contactMode = UserProfileModel.PageMode.Contacts;
```

```
    var sortingTypes = new string[] { "Без сортировки", "По имени", "По дате изменения",
"По дате рождения" };
    var profileTypes = new string[] { "Все контакты", "Аккаунты", "Созданные", };

    var buttonStyle = "background-color:var(--main-color);border-color:var(--main-
color);";
}
<div class="container my-5 p-0">
    <div class="row mb-4 shadow-sm border border-secondary p-3 mx-1" style="border-
radius:16px;">
        <label class="form-label mb-2 fs-5 fw-semibold" style="color:var(--main-color);">
            <i class="bi bi-clipboard-check"></i> Список контактов в системе:
        </label>
        @using (this.Html.BeginForm("AdministratorPanel", "Administrator", FormMethod.Get,
new { @class="row mb-3 justify-content-end" }))
        {
        <div class="col-12 сщд-sm-6 col-lg-6 mb-2">
            <label class="form-label">Параметры сортировки:</label>
            <div class="input-group fs-5">
                <span class="input-group-text"><i class="bi bi-card-list"></i></span>
                <select id="order" name="sortingtype" class="form-select"
onchange="this.form.submit()">
                    @foreach(var selectedItem in sortingTypes)
                    {
                    if (selectedItem == this.Model.SortingType) { <option
value="@selectedItem" selected>@selectedItem</option> }
                    else { <option value="@selectedItem">@selectedItem</option> }
                    }
                </select>
            </div>
        </div>
        <div class="col-12 сщд-sm-6 col-lg-6 mb-2">
            <label class="form-label">Параметры отображения:</label>
            <div class="d-flex flex-row">
                <div class="input-group flex-grow-1">
                    <span class="input-group-text"><i class="bi bi-person-
square"></i></span>
                    <select id="profiletype" name="profiletype" class="form-select"
onchange="this.form.submit()">
                        @foreach(var selectedItem in profileTypes)
                        {
                        if (selectedItem == this.Model.ProfileType) { <option
value="@selectedItem" selected>@selectedItem</option> }
                        else { <option value="@selectedItem">@selectedItem</option> }
                        }
                    </select>
                </div>
                <a class="btn btn-success fs-5 ms-3" style="background-color:var(--main-
color);border-color:var(--main-color)"
                    asp-asp-controller="Administrator" asp-action="GetDocument"> <i
class="bi bi-file-earmark-arrow-down-fill"></i>
                </a>
            </div>
        </div>
        }
        <div class="border-top" style="overflow:scroll;max-height:600px;">
            <table class="table">
                <thead style="color:var(--main-color);">
                    <tr>
                        <th scope="col">#</th> <th scope="col">Аватарка:</th> <th
scope="col">Имя:</th>
                        <th scope="col">Фамилия:</th> <th scope="col">Телефон:</th> <th
scope="col">Email:</th>
                        <th scope="col">Пол:</th> <th scope="col">Изменение:</th>
                        <th scope="col">День рождения:</th> <th scope="col"> </th>
```

```
                    </tr>
                </thead>
                <tbody>
                @foreach(var item in this.Model.ContactsList)
                {
                    <tr>
                        <th scope="row">@item.Contactid</th>
                        <td>
                            <img src="@item.Userpicture!.Filepath" alt="..."
style="width:30px;height:30px;">
                            @if(item.Authorization != null)
                            {
                            <span class="badge rounded-pill bg-danger flex-grow-1"
style="font-size:12px;">
                                <i class="bi bi-person-fill-check"></i>
                            </span>
                            }
                        </td>
                        <td>@item.Name</td> <td>@item.Surname</td>
                        <td>@((item.Phonenumber == null ? "Не установлено" :
item.Phonenumber))</td>
                        <td>@item.Emailaddress</td>
<td>@item.Gendertype.Gendertypename</td>
                        <td>@item.Lastupdate</td> <td>@item.Birthday</td>
                        <td>
                            <div class="w-100 d-inline-flex input-group" >
                                @this.Html.ActionLink("Выбрать", "ContactEditor", new
AdministratorModel()
                                {
                                    SelectedRecord = item.Contactid
                                },
                                new { style=buttonStyle, @class="btn btn-primary flex-
grow-1"})
                            </div>
                        </td>
                    </tr>
                }
                </tbody>
            </table>
        </div>

    </div>
    @await this.Component.InvokeAsync("GroupbyList")
</div>

@*@section Scripts {
    <script src="/_framework/blazor.server.js"></script>
}*@
@section Supports {
<ul class="navbar-nav my-3 my-sm-0 my-lg-0 navbar-nav-scroll"
    style="--bs-scroll-height:84px;font-size:13pt;">

    <li class="nav-item">
        @this.Html.ActionLink("Администратор", "AdministratorPanel", new UserProfileModel
() { },
            new { @class=$"nav-link text-light border-bottom border-3" })
    </li>

    <li class="nav-item">
        @this.Html.ActionLink("Контакты", "ProfileInfo", "UserProfile", new
UserProfileModel { Mode = contactMode },
            new { @class=$"nav-link text-light" })
    </li>
    <li class="nav-item">
```

```
        @this.Html.ActionLink("Профиль", "ProfileInfo", "UserProfile", new
UserProfileModel { Mode = settingsMode },
            new { @class=$"nav-link text-light" })
    </li>
</ul>
<div class="vr mx-3 bg-white opacity-75 d-none d-md-block" style="width:2px;"></div>
<div class="d-flex flex-row my-2 my-md-0 mb-3 mb-md-0 align-items-center">
    @await this.Component.InvokeAsync("ProfileIcon")
    <a class="btn btn-danger mx-2 flex-grow-1" style="height:40px;" href="~/logout"
role="button">
        <i class="bi bi-door-open"></i> Выйти
    </a>
</div>
}
```

```
@using ClientApplication.Views.Shared;
@model AdministratorModel
@{ }

<div class="mb-5">
<component type="typeof(ProfilePage)" render-mode="ServerPrerendered" param-
Contact="this.Model.Contact"
    param-ErrorMessage="this.Model.ErrorMessage" param-SetDatingType="false"
    param-FormRequestLink="this.Model.FormRequestLink" param-IsReadonly="false"/>
</div>
@section Scripts {
    <script src="/_framework/blazor.server.js"></script>
}
@section Supports {
<ul class="navbar-nav my-3 my-sm-0 my-lg-0 navbar-nav-scroll"
    style="--bs-scroll-height:84px;font-size:13pt;">
    <li class="nav-item">
        @this.Html.ActionLink("В главное меню", "ProfileInfo", "UserProfile", new
UserProfileModel { },
            new { @class=$"nav-link text-light border-bottom border-3" })
    </li>
</ul>
<div class="vr mx-3 bg-white opacity-75 d-none d-md-block" style="width:2px;"></div>
<div class="d-flex flex-row my-2 my-md-0 mb-3 mb-md-0 align-items-center">
    @await this.Component.InvokeAsync("ProfileIcon")
    <a class="btn btn-danger mx-2 flex-grow-1" style="height:40px;" href="~/logout"
role="button">
        <i class="bi bi-door-open"></i> Выйти
    </a>
</div>
}
```

```
@using ClientApplication.ViewModels;
@model AuthorizationModel;
@{
    const string underlineClass = "border-bottom border-3";

    var genderList = new SelectList(this.ViewBag.GenderTypes);
    var familyStatusList = new SelectList(new string[] { "В браке", "В поиске", "Холост",
"Все сложно", "Не указывать" });
    this.ViewData["Title"] = "Главная";

    var registrationMode = AuthorizationModel.AuthorizationMode.Registration;
    var loginMode = AuthorizationModel.AuthorizationMode.Login;
}
@section Supports {
<ul class="navbar-nav my-3 my-sm-0 my-lg-0 navbar-nav-scroll"
    style="--bs-scroll-height:84px;font-size:13pt;">
    <li class="nav-item">
        @this.Html.ActionLink("Регистрация", "Authorization", new AuthorizationModel
{ Mode = registrationMode },
```

```
            new { @class=$"nav-link text-light {(this.Model.Mode == registrationMode ?
underlineClass : "")}" })
    </li>
    <li class="nav-item">
        @this.Html.ActionLink("Авторизация", "Authorization", new AuthorizationModel
{ Mode = loginMode },
            new { @class=$"nav-link text-light {(this.Model.Mode == loginMode ?
underlineClass : "")}" })
    </li>
</ul>
<div class="vr mx-3 bg-white opacity-75 d-none d-md-block" style="width:2px;"></div>

<form class="d-flex my-2 my-md-0 mb-3 mb-md-0" role="search">
    <input class="form-control me-2" type="search" placeholder="Search" aria-
label="Search">
    <button class="btn btn-outline-light" type="submit">Поиск</button>
</form>
}
<div class="container p-0 my-5">
@switch (this.Model.Mode)
{
    case AuthorizationModel.AuthorizationMode.Login:
    @using(Html.BeginForm("Login", "Authorization", FormMethod.Post, new { @class="form-
floating" }))
    {
    <div class="row align-items-center justify-content-center">
        <div class="col-lg-6 col-md-8 col-sm-12 shadow border border-secondary p-5"
style="border-radius: 20px;">

            <label class="form-label mb-4 fs-5 fw-semibold" style="color:var(--main-
color);">
                Данные для входа в профиль:
            </label>
            <div class="input-group mb-3 has-validation">
                <span class="input-group-text" id="basic-addon">@@</span>
                <div class="form-floating flex-grow-1 mx-0">
                    @this.Html.TextBox("login", "", new {@class=$"form-control
{(this.Model.HasError ? "is-invalid" : "")}",
                        placeholder="Логин/Email", maxlength="30"})
                    <label for="login">Логин/Email</label>
                </div>
            </div>
            <div class="input-group mb-2 has-validation">
                <div class="form-floating flex-grow-1 @(this.Model.HasError ? "is-
invalid" : "")">
                    @this.Html.TextBox("password", "", new {@class=$"form-control
{(this.Model.HasError ? "is-invalid" : "")}",
                        type="password", placeholder="Пароль", maxlength="30"})
                    <label for="password">Пароль</label>
                </div>
                <button class="btn btn-outline-secondary" type="button" id="button-
addon">
                    <i id="password-icon" class="bi bi-eye-slash-fill"></i>
                </button>
            </div>
            <div id="passwordHelpBlock" class="form-text mb-4">
                Использовать логин или электронную почту для входа в профиль
            </div>
            <div class="form-text mb-2 text-danger">
                  @if(this.Model.HasError) { <text>@(this.Model.ErrorCause ??
"Неверные данные")</text> }
            </div>
            <div class="d-flex justify-content-center mx-auto mb-3">
                <input class="btn btn-success w-100 fs-5" type="submit"
```

```
                    style="background-color:var(--main-color);border-color:var(--main-
color)" value="Войти"/>
            </div>
        </div>
    </div>
    }
    break;
    case AuthorizationModel.AuthorizationMode.Registration:
    @using(Html.BeginForm("Registration", "Authorization", FormMethod.Post))
    {
    <div class="row align-items-center justify-content-center">
        <div class="col-lg-10 col-xl-8 col-md-12 col-12 shadow border border-secondary p-
5"
            style="border-radius: 20px;">

            <label class="form-label mb-4 fs-5 fw-semibold" style="color:var(--main-
color);">
                Укажите контактную информацию для регистации профиля:
            </label>
            <div class="row mb-4">
                <div class="col-12 col-md-6">
                    <label for="login" class="form-label">Логин:</label>
                    <div class="input-group has-validation">
                        <span class="input-group-text"><i class="bi bi-person-
circle"></i></span>
                        @this.Html.TextBox("login", "", new { @class=$"form-control
{(this.Model.HasError ? "is-invalid" : "")}",
                            maxlength="30", placeholder="Логин для вашего профиля"})
                    </div>
                    <div class="form-text">Диапазон вводимых символов от 5 до 30
знаков</div>
                </div>
                <div class="col-12 col-md-6">
                    <label for="password" class="form-label">Пароль:</label>
                    <div class="input-group has-validation">
                        @this.Html.TextBox("password", "", new { @class=$"form-control
{(this.Model.HasError ? "is-invalid" : "")}",
                            type="password", placeholder="Пароль для вашего профиля",
maxlength="30"})
                        <button class="btn btn-outline-secondary" type="button"
id="button-addon">
                            <i id="password-icon" class="bi bi-eye-slash-fill"></i>
                        </button>
                    </div>
                    <div class="form-text">Диапазон вводимых символов от 5 до 30
знаков</div>
                </div>
                <div class="col-md-12 col-12 bg-secondary mt-4 my-st-4"
style="height:1px;"></div>
            </div>
            <div class="row mb-3">
                <div class="col-12 col-md-6 col-lg-4">
                    <label for="surname" class="form-label">Фамилия:</label>
                    @this.Html.TextBox("surname", "", new { @class=$"form-control
{(this.Model.HasError ? "is-invalid" : "")}",
                        placeholder="Ваша фамилия", maxlength="30"})
                </div>
                <div class="col-12 col-md-6 col-lg-4">
                    <label for="name" class="form-label">Имя:</label>
                    @this.Html.TextBox("name", "", new { @class=$"form-control
{(this.Model.HasError ? "is-invalid" : "")}",
                        placeholder="Ваше имя", maxlength="30"})
                </div>
                <div class="col-12 col-md-6 col-lg-4">
```

```
                    <label for="patronymic" class="form-label">Отчество:</label>
                    @this.Html.TextBox("patronymic", "", new { @class=$"form-control
{(this.Model.HasError ? "is-invalid" : "")}",
                        placeholder="Ваше отчество (необязательно)", maxlength="30"})
                </div>
                <div class="form-text">Диапазон вводимых символов от 5 до 30 знаков</div>
            </div>
            <div class="row mb-3">
                <div class="col-12 col-md-6">
                    <label for="email" class="form-label">Email:</label>
                    <div class="input-group has-validation">
                        <span class="input-group-text"><i class="bi bi-
mailbox2"></i></span>
                        @this.Html.TextBox("email", "", new { @class=$"form-control
{(this.Model.HasError ? "is-invalid" : "")}",
                            maxlength="100", placeholder="Пример: example@gmail.com"})
                    </div>
                    <div class="form-text">Диапазон вводимых символов от 10 до 100
знаков</div>
                </div>
                <div class="col-12 col-md-6">
                    <label for="phone" class="form-label">Телефон:</label>
                    <div class="input-group has-validation">
                        <span class="input-group-text"><i class="bi bi-telephone-
forward"></i></span>
                        @this.Html.TextBox("phone", "", new { @class=$"form-control
{(this.Model.HasError ? "is-invalid" : "")}",
                            maxlength="12", placeholder="Пример: +79001001010"})
                    </div>
                    <div class="form-text">Диапазон вводимых символов 12 знаков</div>
                </div>
                <div class="col-md-12 col-12 bg-secondary mt-4 my-st-4"
style="height:1px;"></div>
            </div>
            <div class="row mb-3">
                <div class="col-12 col-md-6 col-lg-4">
                    <label for="gender" class="form-label">Пол:</label>
                    <div class="input-group has-validation">
                        <span class="input-group-text"><i class="bi bi-universal-access-
circle"></i></span>
                        @this.Html.DropDownList("gender", genderList, new { @class="form-
select" })
                    </div>
                </div>
                <div class="col-12 col-md-6 col-lg-4">
                    <label for="family" class="form-label">Семейный статус:</label>
                    <div class="input-group has-validation">
                        <span class="input-group-text"><i class="bi bi-people-
fill"></i></span>
                        @this.Html.DropDownList("family", familyStatusList, new
{ @class="form-select" })
                    </div>
                </div>
                <div class="col-12 col-md-6 col-lg-4">
                    <label for="birthday" class="form-label">День рождения:</label>
                    <input id="birthday" class="form-control" type="date"
name="birthday"/>
                </div>
            </div>
            <div class="form-text mb-2 text-danger">
                  @if(this.Model.HasError) { <text>@(this.Model.ErrorCause ??
"Неверные данные")</text> }
            </div>
            @this.Html.Hidden("id", 0)
            <div class="d-flex justify-content-center mx-auto">
```

```
                <input class="btn btn-success w-100 fs-5" type="submit"
                    style="background-color:var(--main-color);border-color:var(--main-
color)" value="Регистрация"/>
            </div>
        </div>
    </div>
    }
        break;
}
</div>

@section Scripts {
<script id="password-hideshow" type="text/javascript">
    $(this.document).ready(function() {
        var changeClass = (added, deleted) => {
            $('#password-icon').addClass(added); $('#password-icon').removeClass(deleted);
        };
        var isPasswordHide = false;
        $('#button-addon').click(function(){
            if(isPasswordHide == false) {
                $('#password').attr('type', 'text'); changeClass('bi-eye-fill', 'bi-eye-
slash-fill');
                }
                else { $('#password').attr('type', 'password'); changeClass('bi-eye-slash-
fill', 'bi-eye-fill'); }
                isPasswordHide = !isPasswordHide;
        });
    });
</script>
}

@functions { }


@using ClientApplication.Views.Shared;
@using System.Security.Claims;
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

@model ClientApplication.ViewModels.UserContactsModel
@{
    this.ViewBag.Title = "Просмотр контакта";
    const UserProfileModel.PageMode contactMode = UserProfileModel.PageMode.Contacts;

    var profileId = int.Parse(this.Context.User.FindFirst(ClaimTypes.PrimarySid)!.Value);
}
<div class="container p-0 mt-5 mb-4">
    <div class="row shadow-sm border border-secondary p-3 mx-1 mb-4" style="border-
radius:12px;">
        <nav style="--bs-breadcrumb-divider: '>';" aria-label="breadcrumb">

            <ol class="breadcrumb m-0 flex-nowrap" style="overflow:hidden;white-
space:nowrap;text-overflow:ellipsis">
                <li class="breadcrumb-item fs-5" style="color:var(--main-color);">
                    <i class="bi bi-people"></i> Контакты
                </li>
                <li class="breadcrumb-item fs-5 active flex-grow-1">
                    @this.Model.Contact.Name @this.Model.Contact.Surname
                </li>
            </ol>
        </nav>
    </div>
    @if (this.Model.IsAccount)
    {
    <div class="row mx-1 mb-4">
        <div class="col-12 col-md-6">
```

```
            <div class="shadow-sm border border-secondary p-3 mb-4 mb-md-0"
style="border-radius:12px;">
                <label class="form-label mb-2 fs-5 fw-semibold" style="color:var(--main-
color);">
                    <i class="bi bi-clipboard-check"></i> Список связанных контактов:
                </label>
                @await this.Component.InvokeAsync("ContactFriends", new { profileId =
this.Model.Contact.Contactid })
            </div>
        </div>
        <div class="col-12 col-md-6">
            @*<div class="shadow-sm border border-secondary p-3 d-flex flex-column h-100"
                style="border-radius:12px;">
                <label class="form-label mb-2 fs-5 fw-semibold" style="color:var(--main-
color);">
                    <i class="bi bi-chat-dots"></i> Список сообщений:
                </label>
                @await this.Component.InvokeAsync("FriendMessages", new { profileId =
this.Model.Contact.Contactid })
                @using(this.Html.BeginForm("TextMessage", "UserContacts", FormMethod.Post,
                    new { @class="d-flex flex-row my-3 " }))
                {
                <input type="hidden" name="friend" value="@this.Model.SelectedContact"/>
                <div class="input-group">
                    <span class="input-group-text"><i class="bi bi-card-text"></i></span>
                    <input type="text" name="text" class="form-control"
placeholder="Username"/>
                </div>
                <input type="submit" class="btn btn-success mx-2" value="Отправить"
                    style="background-color:var(--main-color);border-color:var(--main-
color);"/>
                }
            </div>*@
            <component type="typeof(ContactMessager)" render-mode="ServerPrerendered"
param-Contactid="this.Model.Contact.Contactid"
                param-Profileid="profileId" param-ContactName="this.Model.Contact.Name"/>
        </div>
    </div>

    }
</div>
<div class="mb-5">
<component type="typeof(ProfilePage)" render-mode="ServerPrerendered" param-
Contact="this.Model.Contact"
    param-ErrorMessage="this.Model.ErrorMessage" param-SetDatingType="true" param-
FormRequestLink="this.Model.FormRequestLink"
    param-IsReadonly="@this.Model.IsAccount" param-DatingType="@this.Model.DatingType"/>
</div>
@section Scripts {
    <script src="/_framework/blazor.server.js"></script>
    <script type="text/javascript">
        function scrollToBottom() {
            var objDiv = document.getElementById("messanger"); objDiv.scrollTop =
objDiv.scrollHeight;
        }
        scrollToBottom();
    </script>
}
@section Supports {
<ul class="navbar-nav my-3 my-sm-0 my-lg-0 navbar-nav-scroll"
    style="--bs-scroll-height:84px;font-size:13pt;">
    <li class="nav-item">
        @this.Html.ActionLink("В главное меню", "ProfileInfo", "UserProfile", new
UserProfileModel { Mode = contactMode },
            new { @class=$"nav-link text-light border-bottom border-3" })
```

```
    </li>
</ul>
<div class="vr mx-3 bg-white opacity-75 d-none d-md-block" style="width:2px;"></div>
<div class="d-flex flex-row my-2 my-md-0 mb-3 mb-md-0 align-items-center">
    @await this.Component.InvokeAsync("ProfileIcon")
    <a class="btn btn-danger mx-2 flex-grow-1" style="height:40px;" href="~/logout"
role="button">
        <i class="bi bi-door-open"></i> Выйти
    </a>
</div>
}


@using DAModels = DatabaseAccess.Models;
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@model IEnumerable<DAModels::Contact>
@{
    var sortingTypes = new string[] { "Без сортировки", "По имени", "По дате изменения",
"По дате рождения" };
    var profileTypes = new string[] { "Все контакты", "Аккаунты", "Созданные", };
    const string cardStyle = "card m-2 m-sm-3 shadow-sm border border-secondary flex-row
flex-sm-column";

    var currentPage = int.Parse(this.ViewData["CurrentPage"]!.ToString()!);
    var recordOnPage = int.Parse(this.ViewData["RecordOnPage"]!.ToString()!);
    var pagesCount = int.Parse(this.ViewData["PagesCount"]!.ToString()!);

    void RenderPageActionLink(string text, bool isDisabled, int page)
    {
        <li class="page-item @((isDisabled ? "disabled" : ""))">
            @this.Html.ActionLink(text, "ProfileInfo", new UserProfileModel()
            {
                ProfileType = this.ViewData["ProfileType"]!.ToString()!, RecordOnPage =
recordOnPage,
                CurrentPage = page, SortingType =
this.ViewData["SortingType"]!.ToString()!,
            },
            new { @class="page-link", style=$"color:{(isDisabled ? "" : "var(--main-
color)")};" })
        </li>
    }
}
<div class="container p-0 my-5">
    @if(this.ViewData["Message"] != null)
    {
        <div class="alert alert-warning alert-dismissible fade show" role="alert"
style="border-radius:16px;">
            <strong>Уведомление!</strong> @this.ViewData["Message"]!.ToString()!
            <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
        </div>
    }
    <div class="row mb-0 justify-content-between">

        <div class="col-12 col-md-6 text-center text-md-start mb-3 mb-md-0">
            <h1 class="fs-2 m-0" style="color:var(--main-color);font-weight: 600;">
                Меню контактов <span class="badge rounded-pill bg-danger"
style="font-size:12px;">Contact</span>
            </h1>
            <p class="form-text m-0" style="font-size:18px;">Список доступных
контактов</p>
        </div>
        <div class="col-12 col-md-6 d-flex flex-row align-items-center">
            @using (this.Html.BeginForm("ProfileInfo", "UserProfile", FormMethod.Get, new
{ @class="d-inline flex-grow-1 mx-1" }))
            {
```

```
            <div class="d-flex input-group has-validation">
                @this.Html.TextBox("searchquery", "", new {@class=$"form-control",
placeholder="Контакт для поиска", maxlength="50"})
                <button class="btn btn-success fs-5" type="submit"
                    style="background-color:var(--main-color);border-color:var(--main-
color)" value="Найти">
                    <i class="bi bi-search"></i>
                </button>
            </div>
            }
            <div class="d-flex justify-content-center mx-1">
                <button class="btn btn-success w-100 fs-5" style="background-color:var(--
main-color);border-color:var(--main-color)"
                    data-bs-toggle="collapse" href="#collapseSearch">
                    <i class="bi bi-person-fill-add"></i>
                </button>
            </div>
            <div class="d-flex justify-content-center mx-1">
                <a class="btn btn-success w-100 fs-5" style="background-color:var(--main-
color);border-color:var(--main-color)"
                    asp-asp-controller="UserProfile" asp-action="GetDocument"> <i
class="bi bi-file-earmark-arrow-down-fill"></i>
                </a>
            </div>
        </div>
    </div>
    <div class="row collapse my-3 justify-content-end" id="collapseSearch">
        <div class="col-12 col-sm-6"> @await this.Component.InvokeAsync("FriendCreator")
</div>
    </div>
    @using (this.Html.BeginForm("ProfileInfo", "UserProfile", FormMethod.Get, new
{ @class="row mb-3 justify-content-end" }))
    {
    <div class="col-12 сщд-sm-6 col-lg-4">
        <label class="form-label"> </label>

        <div class="input-group mb-2">
            <span class="input-group-text">Контактов на странице:</span>
            <select id="count" name="recordonpage" class="form-select"
onchange="this.form.submit()">
                @for(var index = UserProfileModel.MinRecordOnPage; index <
UserProfileModel.MaxRecordOnPage; index++)
                {
                if (index.ToString() == this.ViewData["RecordOnPage"]!.ToString())
{ <option value="@index" selected>@(index) записи(ей)</option> }
                else { <option value="@index">@index записи(ей)</option> }
                }
            </select>
        </div>
    </div>
    <div class="col-12 сщд-sm-6 col-lg-4">
        <label class="form-label">Параметры сортировки:</label>
        <div class="input-group mb-2">

            <span class="input-group-text"><i class="bi bi-card-list"></i></span>
            <select id="order" name="sortingtype" class="form-select"
onchange="this.form.submit()">
                @foreach(var selectedItem in sortingTypes)
                {
                if (selectedItem == this.ViewData["SortingType"]!.ToString()) { <option
value="@selectedItem" selected>@selectedItem</option> }
                else { <option value="@selectedItem">@selectedItem</option> }
                }
            </select>
        </div>
```

```
    </div>
    <div class="col-12 сщд-sm-6 col-lg-4">
        <label class="form-label">Параметры отображения:</label>
        <div class="input-group mb-2">

            <span class="input-group-text"><i class="bi bi-person-square"></i></span>
            <select id="profiletype" name="profiletype" class="form-select"
onchange="this.form.submit()">
                @foreach(var selectedItem in profileTypes)
                {
                if (selectedItem == this.ViewData["ProfileType"]!.ToString()) { <option
value="@selectedItem" selected>@selectedItem</option> }
                else { <option value="@selectedItem">@selectedItem</option> }
                }
            </select>
        </div>
    </div>
    @this.Html.Hidden("Mode", UserProfileModel.PageMode.Contacts);
    }
    <div class="row mb-3 gx-2 justify-content-center row-cols-1 row-cols-md-3 row-cols-
sm-2 row-cols-lg-4 row-cols-xl-5">
        @foreach(var friendRecord in this.Model)
        {
        <div class="col mb-1 mb-sm-3">
            <div class="@cardStyle d-none d-sm-flex" style="border-radius:16px;">
                <img src="@friendRecord.Userpicture!.Filepath" class="card-img-top w-75
p-3 align-self-center" alt="...">

                @*@{await RenderCardBody(friendRecord.Value, $"{friendRecord.Value.Name}
{friendRecord.Value.Surname}",
                    friendRecord.Value.Authorization != null,
friendRecord.Key.Typeofdating);}*@
                @await this.Component.InvokeAsync("ContactCard", new { contactModel =
friendRecord })
            </div>
            <div class="@cardStyle d-sm-none d-inline-flex" style="border-radius:14px;">
                <img src="@friendRecord.Userpicture!.Filepath" class="card-img-top w-25
p-3 align-self-center" alt="...">

                @*@{await RenderCardBody(friendRecord.Value, $"{friendRecord.Value.Name}
{friendRecord.Value.Surname}",
                    friendRecord.Value.Authorization != null,
friendRecord.Key.Typeofdating);}*@
                @await this.Component.InvokeAsync("ContactCard", new { contactModel =
friendRecord })
            </div>
        </div>
        }
        @if(this.Model.Count() <= 0) {
        <div class="col w-100 m-3 text-center">
            <span class="fs-4" style="color:var(--main-color)">Еще не добавлено ни одного
контакта</span>
        </div>
        }
    </div>
    <nav class="table-responsive row justify-content-start mx-3">
        <ul class="pagination col-12" style="overflow-x: scroll;">
            @{RenderPageActionLink("Предыдующая", (currentPage <= 0), currentPage <= 0 ?
0 : currentPage - 1);}

            <div class="d-inline-flex p-0" style="overflow-x:scroll;">
                @for(var page = 0; page < pagesCount; page++) RenderPageActionLink((page
+ 1).ToString(), (page == currentPage), page);
            </div>
            @{RenderPageActionLink("Следующая", (currentPage >= pagesCount - 1),
```

```
                currentPage >= pagesCount - 1 ? currentPage : currentPage + 1);}
        </ul>
    </nav>
</div>
@using ClientApplication.Views.Shared;
@using DatabaseAccess.Models;
@model ClientApplication.ViewModels.UserProfileModel;
@{
    const UserProfileModel.PageMode settingsMode = UserProfileModel.PageMode.Settings;
    const UserProfileModel.PageMode contactMode = UserProfileModel.PageMode.Contacts;
    const string underlineClass = "border-bottom border-3";

    this.ViewBag.Title = this.Model.Contact.Authorization!.Login;
    var contactId = this.Model.Contact.Contactid;
    var searchingParameters = new ViewDataDictionary(this.ViewData)
    {
        ["SortingType"] = this.Model.SortingType, ["ProfileType"] =
this.Model.ProfileType,
        ["RecordOnPage"] = this.Model.RecordOnPage, ["CurrentPage"] =
this.Model.CurrentPage,
        ["PagesCount"] = this.Model.PagesCount, ["Message"] = this.Model.ErrorMessage,
    };
    var friendsList = this.Model.Contact.FriendContactid1Navigations.Select(item =>
item.Contactid2Navigation);
}
@switch(this.Model.Mode)
{
    case UserProfileModel.PageMode.Settings:
    <div class="my-5">
    <component type="typeof(ProfilePage)" render-mode="ServerPrerendered" param-
Contact="this.Model.Contact"
        param-ErrorMessage="this.Model.ErrorMessage" param-
FormRequestLink="this.Model.FormRequestLink"/>
    </div>
        break;
    case UserProfileModel.PageMode.Contacts: @await this.Html.PartialAsync("ContactList",
friendsList, searchingParameters);
        break;
}
@section Scripts {
    <script src="/_framework/blazor.server.js"></script>
}
@section Supports {
<ul class="navbar-nav my-3 my-sm-0 my-lg-0 navbar-nav-scroll"
    style="--bs-scroll-height:84px;font-size:13pt;">
    @if (this.ViewBag.IsAdmin)
    {
    <li class="nav-item">
        @this.Html.ActionLink("Администратор", "AdministratorPanel", "Administrator", new
AdministratorModel () { },
            new { @class=$"nav-link text-light"})
    </li>
    }
    <li class="nav-item">
        @this.Html.ActionLink("Контакты", "ProfileInfo", new UserProfileModel { Mode =
contactMode },
            new { @class=$"nav-link text-light {(this.Model.Mode == contactMode ?
underlineClass : "")}" })
    </li>
    <li class="nav-item">
        @this.Html.ActionLink("Профиль", "ProfileInfo", new UserProfileModel { Mode =
settingsMode },
            new { @class=$"nav-link text-light {(this.Model.Mode == settingsMode ?
underlineClass : "")}" })
    </li>
```

```
</ul>
<div class="vr mx-3 bg-white opacity-75 d-none d-md-block" style="width:2px;"></div>
<div class="d-flex flex-row my-2 my-md-0 mb-3 mb-md-0 align-items-center">
    @await this.Component.InvokeAsync("ProfileIcon")
    <a class="btn btn-danger mx-2 flex-grow-1" style="height:40px;" href="~/logout"
role="button">
        <i class="bi bi-door-open"></i> Выйти
    </a>
</div>

}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@this.ViewData["Title"]</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.10.5/font/bootstrap-icons.css">
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/ClientApplication.styles.css" asp-append-
version="true" />
    <base href="~/"/>
    @await RenderSectionAsync("Style", required: false)
</head>
<body>
    <nav class="navbar navbar-expand-md fixed-top border border-bottom border-secondary
        shadow py-2 bg-body-tertiary" style="background-color:var(--main-color);">
        <div class="container-fluid px-4">
            <a class="navbar-brand p-0 d-inline-flex text-light" href="#">

                <img src="/icons/logo.png" alt="Logo" width="48" height="48" class="me-1
d-inline-block align-text-top"/>
                <i class="align-self-center fs-5 text-decoration-none">Записная
книжка</i>
            </a>
            <button class="navbar-toggler border border-2" type="button"
style="outline:none;box-shadow:none;"
                data-bs-toggle="collapse" data-bs-target="#navbar-dropdown" >
                <span class="text-light fs-2 bi bi-list"></span>
            </button>
            <div class="collapse navbar-collapse justify-content-end px-2" id="navbar-
dropdown">
                @await RenderSectionAsync("Supports")
            </div>
        </div>
    </nav>
    <div class="container mt-5" style="flex-grow: 1;">
        <main role="main" class="mb-3 mb-sm-3 h-100"> @RenderBody() </main>
    </div>
    <footer class="text-center text-white mt-sm-2 mt-2" id="page-footer">
        <div class="container p-4 pb-0">
            <section class="mb-4">
                <a class="btn btn-outline-light btn-floating m-1" href="#!"
role="button">
                    <i class="fs-5 bi bi-telegram text-light"></i>
                </a>
                <a class="btn btn-outline-light btn-floating m-1" href="#!"
role="button">
                    <i class="fs-5 bi bi-youtube text-light"></i>
                </a>
                <a class="btn btn-outline-light btn-floating m-1" href="#!"
role="button">
```

```
                    <i class="fs-5 bi bi-facebook text-light"></i>
                </a>
                <a class="btn btn-outline-light btn-floating m-1" href="#!"
role="button">
                    <i class="fs-5 bi bi-discord text-light"></i>
                </a>
                <a class="btn btn-outline-light btn-floating m-1"
href="https://github.com/mo0nchild" role="button">
                    <i class="fs-5 bi bi-github text-light"></i>
                </a>
            </section>
        </div>
        <div class="text-center fs-6 p-3" style="background-color:var(--secondary-
color);">
            © 2023 Copyright:
            <a class="text-white" href="#!">ContactApp.com</a>
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
@using ClientApplication.Middleware;
@using Microsoft.EntityFrameworkCore;
@using DatabaseAccess;
@using DatabaseAccess.Models;
@using Microsoft.JSInterop;
@using Microsoft.AspNetCore.Components.Forms;
@using Microsoft.AspNetCore.Components.Web;
@using Microsoft.AspNetCore.SignalR.Client;

@implements IAsyncDisposable

@inject NavigationManager Navigation
@inject IDbContextFactory<DatabaseContext> DatabaseFactory
@inject IJSRuntime JS

<div class="shadow-sm border border-secondary p-3 d-flex flex-column h-100"
style="border-radius:12px;">
    <label class="form-label mb-2 fs-5 fw-semibold" style="color:var(--main-color);">
        <i class="bi bi-chat-dots"></i> Список сообщений:
    </label>
    <div id="messanger" class="border-top py-3 flex-grow-1" style="overflow:scroll;max-
height:200px;">
        @foreach(var item in this.MessagesList)
        {
        <div class="mb-3 d-flex flex-row flex-nowrap @(ChooseSide(item.Contactid !=
Profileid))">
            <div class="shadow-sm border border-secondary p2 p-3" style="border-
radius:16px;max-width:60%;">

                <p class="m-0" style="color:var(--main-color);font-size:18px;">
                    @((item.Contactid == Profileid ? "Вы:" : $"{ContactName}: "))
                </p>
                <p class="m-0">@item.Messagebody</p>
                <p class="m-0">@item.Sendtime.TimeOfDay.ToString()</p>
            </div>
        </div>
        }
    </div>
    <div class="d-flex flex-row my-3">
        <div class="input-group">
            <span class="input-group-text"><i class="bi bi-card-text"></i></span>
```

```
            <input type="text" name="text" class="form-control" placeholder="Username"
@bind-value="this.MessageText"/>
        </div>
        <button class="btn btn-success mx-2" style="background-color:var(--main-
color);border-color:var(--main-color);"
            @onclick="SendButtonClickHandler">Отправить</button>
    </div>
</div>
@code {
    [ParameterAttribute, EditorRequired] public int Contactid { get; set; } = default;
    [ParameterAttribute, EditorRequired] public int Profileid { get; set; } = default;
    [ParameterAttribute, EditorRequired] public string ContactName { get; set; } =
string.Empty;

    protected virtual string MessageText { get; set; } = string.Empty;
    protected virtual List<Message> MessagesList { get; set; } = new();

    private HubConnection hubConnection = default!;
    private Friend friendRecord = default!;
    private string ChooseSide(bool check) => (check ? "justify-content-end" : "justify-
content-start");

    protected virtual async Task SendButtonClickHandler()
    {
        var messageModel = new ContactMessagerHub.ContactMessagerModel(new Message()
        {
            Friendid = friendRecord.Friendid,
            Contactid = Profileid, Sendtime = DateTime.Now, Messagebody =
this.MessageText,
        },
        this.friendRecord.Friendid.ToString());

        if (hubConnection is not null) await hubConnection.SendAsync("SendMessage",
messageModel);
        this.MessageText = string.Empty;
    }

    protected override async Task OnInitializedAsync()
    {
        using(var dbcontext = this.DatabaseFactory.CreateDbContext())
        {
            dbcontext.Contacts.Where(item => item.Contactid == Contactid).Load();
            this.friendRecord = dbcontext.Friends.First(item =>  (item.Contactid1 ==
Contactid && item.Contactid2 == Profileid)
                || (item.Contactid2 == Contactid && item.Contactid1 == Profileid));

            if (this.friendRecord == null) throw new Exception("Запись о связях контактов
не найдена");
            this.MessagesList = dbcontext.Messages.Where(item => item.Friendid ==
friendRecord.Friendid)
                .OrderBy(item => item.Sendtime).ToList();
        }
        hubConnection = new
HubConnectionBuilder().WithUrl(Navigation.ToAbsoluteUri("/chathub")).Build();

        hubConnection.On<string>("Notify", (item) => Console.WriteLine(item));
        hubConnection.On<Message>("GetMessage", (item) =>
        {
            this.MessagesList.Add(item);
            Console.WriteLine($"\nitem.Messagebody: {item.Messagebody}");
            this.InvokeAsync(StateHasChanged);
        });
        await hubConnection.StartAsync();
        await this.hubConnection.InvokeAsync("JoinGroup",
this.friendRecord.Friendid.ToString());
```

```
    }

    protected override async Task OnAfterRenderAsync(bool firstRender) => await
JS.InvokeVoidAsync("scrollToBottom");

    public async ValueTask DisposeAsync() => await hubConnection.DisposeAsync();
}
@using DAModels = DatabaseAccess.Models;
@using DatabaseAccess;
@using Microsoft.AspNetCore.Components.Forms;
@using Microsoft.AspNetCore.Components.Web;
@using Microsoft.EntityFrameworkCore;
@using ViewModels;
@using ClientApplication.Views.Shared.Components;

@inject NavigationManager NavigationManager
@inject ILogger<ProfilePage> Logger
@inject IHttpClientFactory HttpFactory
@inject IDbContextFactory<DatabaseContext> DatabaseFactory

<div class="container p-0">
    @if(this.ErrorMessage != null)
    {
    <div class="mb-4 alert alert-danger alert-dismissible fade show" role="alert"
style="border-radius:16px;">
        <i class="bi bi-exclamation-triangle-fill"></i>
        <strong>Уведомление!</strong> @this.ErrorMessage
        <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
    </div>
    }
    <div class="row mb-4 justify-content-md-between">

        <div class="col-md-3 col-12 mb-4">
            <div class="shadow border border-secondary p-4" style="border-radius: 16px;">

                <div class="d-md-none mb-3">
                    <button class="btn btn-primary w-100" data-bs-toggle="collapse" data-
bs-target="#collapseGroup"
                        style="background-color:transparent;border-color:var(--main-
color);color:var(--main-color);">
                        Показать разделы настройки
                    </button>
                    <div class="collapse" id="collapseGroup">
                        <div class="card card-body py-4"> <SettingsTabs @bind-
ComponentValue="@this.SelectedTab"/></div>
                    </div>
                </div>
                <div class="d-sm-none d-none d-md-block mb-3">
                    <SettingsTabs @bind-ComponentValue="@this.SelectedTab" />
                </div>
                @if(this.SetDatingType == true)
                {
                <div class="mb-3">
                    <TextSelect ItemId="typeofdating" @bind-
ComponentValue="@this.DatingType.Typeofdating"
                        LabelTitle="Тип знакомства" GroupIcon="bi bi-diagram-3-fill"
                        ElementList="this.EditingModel.Datingtypes.Select(item =>
item.Typeofdating).ToList()"/>
                </div>
                }
                <ModalDialog ButtonText="Сохранить"
FormRequestLink="@this.FormRequestLink">
                @*@if(this.IsReadonly != true)
                {*@
                    <input type="hidden" name="id" value="@this.Contact.Contactid"/>
```

```
                    <input type="hidden" name="surname" value="@this.Contact.Surname"/>
                    <input type="hidden" name="name" value="@this.Contact.Name"/>
                    <input type="hidden" name="validate" value="true"/>

                    <input type="hidden" name="patronymic"
value="@this.Contact.Patronymic"/>
                    <input type="hidden" name="email"
value="@this.Contact.Emailaddress"/>
                    @if(this.Contact.Phonenumber != "" && this.Contact.Phonenumber !=
null)
                    { <input type="hidden" name="phone"
value="@this.Contact.Phonenumber"/> }

                    <input type="hidden" name="family"
value="@this.Contact.Familystatus"/>
                    <input type="hidden" name="birthday" value="@this.Contact.Birthday"/>
                    <input type="hidden" name="gender"
value="@this.Contact.Gendertype.Gendertypename"/>
                    <input type="hidden" name="picture"
value="@this.Contact.Userpicture!.Filepath"/>

                    @if(this.Contact.Location!.City.Cityname != "Не установлено")
                    {
                        <input type="hidden" name="city"
value="@this.Contact.Location!.City.Cityname"/>
                        <input type="hidden" name="country"
value="@this.Contact.Location!.City.Country"/>
                        <input type="hidden" name="street"
value="@this.Contact.Location!.Street"/>
                    }
                    <input type="hidden" name="employee_length"
value="@this.Contact.Employees.Count()"/>
                    @for(var index = 0; index < this.Contact.Employees.Count(); index++)
                    {
                        <input type="hidden" name="employee[@index].company"

value="@this.Contact.Employees.ElementAt(index).Companyname"/>
                        <input type="hidden" name="employee[@index].post"

value="@this.Contact.Employees.ElementAt(index).Post!.Postname"/>
                        <input type="hidden" name="employee[@index].status"
                            value="@this.Contact.Employees.ElementAt(index).Status"/>
                    }
                    <input type="hidden" name="hobby_length"
value="@this.Contact.Hobbies.Count()"/>
                    @for(var index = 0; index < this.Contact.Hobbies.Count(); index++)
                    {
                        <input type="hidden" name="hobby[@index]"
                            value="@this.Contact.Hobbies.ElementAt(index).Hobbyname"/>
                    }
                    <input type="hidden" name="quality_length"
value="@this.Contact.Humanqualities.Count()"/>
                    @for(var index = 0; index < this.Contact.Humanqualities.Count();
index++)
                    {
                        <input type="hidden" name="quality[@index]"

value="@this.Contact.Humanqualities.ElementAt(index).Qualityname"/>
                    }
                @*}*@
                    @if(this.SetDatingType)
                    { <input type="hidden" name="datingType"
value="@this.DatingType.Typeofdating"/> }
                </ModalDialog>
                @if(this.IsReadonly != true)
```

```
                { <a class="btn btn btn-outline-danger mt-2 w-100" role="button"
href="/user/deletecontact/@(this.Contact.Contactid)">Удалить</a> }
            </div>
        </div>
        <div class="col-md-9 col-12">
            <div class="shadow border border-secondary p-4" style="border-radius: 16px;">
                <label class="form-label mb-2 fs-5 fw-semibold" style="color:var(--main-
color);">
                    Просмотр информации контакта:
                </label>
               @* <div class="form-text mb-2 text-danger">
                      @if(this.ErrorMessage != null)
{ <text>@(this.ErrorMessage)</text> }
                </div>*@

@switch(this.SelectedTab)
            {
            case SelectedTabType.Details:
            <div class="row mb-3">
                <div class="col-12 col-lg-4 col-md-6">
                    <TextField ItemId="contact-surname" LabelTitle="Фамилия" @bind-
ComponentValue="@this.Contact.Surname"
                               Placeholder="Ваша фамилия" IsValid="@(this.ErrorMessage == null)"
IsReadonly="@this.IsReadonly"/>
                </div>
                <div class="col-12 col-lg-4 col-md-6">
                    <TextField ItemId="contact-name" LabelTitle="Имя" @bind-
ComponentValue="@this.Contact.Name"
                               Placeholder="Ваше имя" IsValid="@(this.ErrorMessage == null)"
IsReadonly="@this.IsReadonly"/>
                </div>
                <div class="col-12 col-lg-4 col-md-6">
                    <TextField ItemId="contact-patronymic" LabelTitle="Отчество" @bind-
ComponentValue="@this.Contact.Patronymic"
                               Placeholder="Ваше отчество" IsValid="@(this.ErrorMessage ==
null)" IsReadonly="@this.IsReadonly"/>
                </div>
                <div class="form-text">Диапазон вводимых символов от 5 до 30 знаков</div>
            </div>
            <div class="row mb-3">
                <div class="col-12 col-md-6">
                    <TextField ItemId="contact-phone" LabelTitle="Телефон" @bind-
ComponentValue="@this.Contact.Phonenumber"
                               Placeholder="Пример: +79001001010" GroupIcon="bi bi-telephone-
forward" MaxLength="12"
                               IsValid="@(this.ErrorMessage == null)"
IsReadonly="@this.IsReadonly"/>
                    <div class="form-text">Диапазон вводимых символов 12 знаков</div>
                </div>
                <div class="col-12 col-md-6">
                    <TextField ItemId="contact-email" LabelTitle="Email" @bind-
ComponentValue="@this.Contact.Emailaddress"
                               Placeholder="Пример: example@gmail.com" GroupIcon="bi bi-
mailbox2" MaxLength="100"
                               IsValid="@(this.ErrorMessage == null)"
IsReadonly="@this.IsReadonly"/>
                    <div class="form-text">Диапазон вводимых символов от 10 до 100
знаков</div>
                </div>
                <div class="col-md-12 col-12 bg-secondary mt-4 my-st-4"
style="height:1px;"></div>

            </div>
            <div class="row mb-3">
                <div class="col-12 col-lg-4 col-md-6">
```

```
                    <TextSelect ItemId="contact-gender" @bind-
ComponentValue="@this.Contact.Gendertype.Gendertypename"
                                LabelTitle="Пол" GroupIcon="bi bi-universal-access-circle"
IsReadonly="@this.IsReadonly"
                                ElementList="this.EditingModel.GenderTypes.Select(item =>
item.Gendertypename).ToList()"/>
                </div>
                <div class="col-12 col-lg-4 col-md-6">
                    <TextSelect ItemId="contact-family" @bind-
ComponentValue="@this.Contact.Familystatus"
                                LabelTitle="Семейный статус" GroupIcon="bi bi-people-fill"
IsReadonly="@this.IsReadonly"
                                ElementList="@((new [] { "В браке", "В поиске", "Холост", "Все
сложно", "Не указывать" }).ToList())"/>
                </div>
                <div class="col-12 col-lg-4 col-md-6">
                    <label for="birthday" class="form-label">День рождения:</label>
                    <input id="birthday" class="form-control" type="date" name="birthday"
@bind-value="this.Contact.Birthday"
                           readonly="@this.IsReadonly"/>
                </div>
            </div>
            <div class="row mb-3">
                <label for="user-picture" class="form-label">Аватарка:</label>
                <div class="col-6 col-lg-4 col-md-6 gx-4" id="user-picture">
                    <select class="form-select" size="3"
@onchange="this.UserPictureChangeHandler" style="height:100px"
                            disabled="@this.IsReadonly">
                        @foreach(var item in this.EditingModel.Pictures)
                        {
                            @if(item.Filepath != this.Contact.Userpicture!.Filepath)
{ <option value="@item.Filepath">@item.Picturename</option> }
                            else { <option value="@item.Filepath"
selected>@item.Picturename</option> }
                        }
                    </select>
                </div>
                <div class="col-6 col-lg-4 col-md-6">
                    <img src="@(this.Contact.Userpicture?.Filepath)" class="rounded img-
thumbnail"
                         style="width:100px;height:100px;">
                </div>
            </div>
            <div class="row mb-3">
                <div class="col-12 col-lg-4 col-md-6">
                    <label for="contact-country" class="form-label">Страна:</label>
                    <div class="input-group has-validation">

                        <span class="input-group-text"><i class="bi bi-globe"></i></span>
                        <input class="form-control" type="text"
@bind="@this.ContactCountry" readonly/>
                    </div>
                </div>
                <div class="col-12 col-lg-4 col-md-6">
                    <TextSelect ItemId="contact-city" @bind-
ComponentValue="this.ContactLocation"
                                LabelTitle="Город" GroupIcon="bi bi-buildings"
IsReadonly="@this.IsReadonly"
                                ElementList="this.EditingModel.Cities.Select(item =>
item.Cityname).ToList()"/>
                </div>
                <div class="col-12 col-lg-4 col-md-6">
                    <TextField ItemId="contact-street" LabelTitle="Улица" @bind-
ComponentValue="@this.Contact.Location!.Street"
```

```
                        Placeholder="Ваша улица" GroupIcon="bi bi-signpost-2"
MaxLength="50" IsValid="@(this.ErrorMessage == null)"
                        IsReadonly="@this.IsReadonly"/>
                    <div class="form-text">Диапазон вводимых символов от 5 до 50
знаков</div>
                </div>
            </div>
                break;
case SelectedTabType.Employee:
            <div class="row mb-3">
                <div class="col-12 col-lg-4 col-md-6">
                    <TextField ItemId="contact-company" LabelTitle="Название компании"
Placeholder="Название вашей компании"
                        @bind-ComponentValue="@this.CurrentEmployee.Companyname"
IsValid="@this.CurrentEmployeeIsValid"
                        GroupIcon="bi bi-journal-bookmark-fill" MaxLength="50"
IsReadonly="@this.IsReadonly"/>
                    <div class="form-text">Диапазон вводимых символов от 5 до 50
знаков</div>
                </div>
                <div class="col-12 col-lg-4 col-md-6">
                    <TextSelect ItemId="contact-city" @bind-
ComponentValue="@this.CurrentEmployee.Post!.Postname"
                        LabelTitle="Должность" GroupIcon="bi bi-person-video2"
IsReadonly="@this.IsReadonly"
                        ElementList="this.EditingModel.Postes.Select(item =>
item.Postname).ToList()"/>
                </div>
                <div class="col-12 col-lg-4 col-md-6">
                    <TextSelect ItemId="contact-status" @bind-
ComponentValue="@this.CurrentEmployee.Status"
                        LabelTitle="Рабочий статус" GroupIcon="bi bi-link"
IsReadonly="@this.IsReadonly"
                        ElementList="@(new() { "Не указывать", "Работает", "Уволен" })"/>
                </div>
            </div>
            <div class="row mb-3">
                <div class="col-12 mb-3">
                    <label for="contact-emploees" class="form-label">Рабочие
места:</label>
                    <div class="card card-body" id="contact-emploees">
                        <ul class="list-group" style="overflow:scroll;max-height:160px;">
                        @foreach(var item in this.Contact.Employees)
                        {
                            <li class="list-group-item d-flex justify-content-between">
                                <label class="form-check-label" for="firstRadio">
                                    @($"{item.Companyname} / {item.Post!.Postname} -
{item.Status}")
                                </label>
                                <button type="button" class="btn-close"
                                    @onclick="@(value => { if(!this.IsReadonly)
this.Contact.Employees.Remove(item); })" />
                            </li>
                        }
                        </ul>
                    </div>
                </div>
                <div class="col-12 col-lg-6 col-md-6 col-xl-4">
                    <button style="background-color:var(--main-color);border-color:var(--
main-color)"
                        class="btn btn-success w-100"
@onclick="@EmployeeClickHandler">Добавить запись</button>
                </div>
            </div>
```

```
                break;
            case SelectedTabType.QualityHobby:
            <div class="row mb-3">
                <div class="col-12 col-lg-4 col-md-6 mb-3">
                    <TextSelect ItemId="contact-hobby" @bind-
ComponentValue="@this.CurrentHobbyValue"
                                LabelTitle="Название хобби" GroupIcon="bi bi-gift"
IsReadonly="@this.IsReadonly"
                                ElementList="@(this.EditingModel.HobbyTypes.Select(item =>
item.Hobbyname).ToList())"/>
                </div>
                <div class="col-12 col-lg-4 col-md-6 mb-3">
                    <label for="contact-hobby" class="form-label">Тип хобби:</label>
                    <div class="input-group has-validation">

                        <span class="input-group-text"><i class="bi bi-file-bar-graph-
fill"></i></span>
                        <input class="form-control" type="text"
@bind="@this.CurrentHobby.Hobbytype" readonly/>
                    </div>
                </div>
                <div class="col-12 col-lg-4 col-md-6 mb-3">
                    <label for="contact-hobby" class="form-label"> </label>

                    <button style="background-color:var(--main-color);border-color:var(--
main-color)"
                            class="btn btn-success w-100"
@onclick="@HobbyClickHandler">Добавить запись</button>
                </div>
            </div>
            <div class="row mb-3">
                <div class="col-12 mb-3">
                    <label for="contact-hobby" class="form-label">Список хобби:</label>
                    <div class="card card-body" id="contact-hobby">
                        <ul class="list-group" style="overflow:scroll;max-height:160px;">
                        @foreach(var item in this.Contact.Hobbies)
                        {
                            <li class="list-group-item d-flex justify-content-between">
                                <label class="form-check-label" for="firstRadio">
@($"{item.Hobbyname} / {item.Hobbytype}")</label>
                                <button type="button" class="btn-close"
                                    @onclick="@(value => { if(!this.IsReadonly)
this.Contact.Hobbies.Remove(item); })" />
                            </li>
                        }
                        </ul>
                    </div>
                </div>
            </div>
            <div class="row mb-3">
                <div class="col-12 col-lg-4 col-md-6 mb-3">
                    <TextSelect ItemId="contact-quality" @bind-
ComponentValue="@this.CurrentQualityValue"
                                LabelTitle="Челоческое качество" GroupIcon="bi bi-emoji-smile-
fill" IsReadonly="@this.IsReadonly"
                                ElementList="@(this.EditingModel.QualityTypes.Select(item =>
item.Qualityname).ToList())"/>
                </div>
                <div class="col-12 col-lg-4 col-md-6 mb-3">
                    <label for="contact-qualitytype" class="form-label">Тип
качества:</label>
                    <div class="input-group has-validation">

                        <span class="input-group-text"><i class="bi bi-file-bar-graph-
fill"></i></span>
```

```
                        <input class="form-control" type="text"
@bind="@this.CurrentQuality.Qualitytype" readonly/>
                    </div>
                </div>
                <div class="col-12 col-lg-4 col-md-6 mb-3">
                    <label for="contact-qualitytype" class="form-label"> </label>

                    <button style="background-color:var(--main-color);border-color:var(--
main-color)"
                        class="btn btn-success w-100"
@onclick="@QualityClickHandler">Добавить запись</button>
                </div>
            </div>
            <div class="row mb-3">
                <div class="col-12 mb-3">
                    <label for="contact-quality" class="form-label">Список человеческих
качеств:</label>
                    <div class="card card-body" id="contact-quality">
                        <ul class="list-group" style="overflow:scroll;max-height:160px;">
                        @foreach(var item in this.Contact.Humanqualities)
                        {
                            <li class="list-group-item d-flex justify-content-between">
                                <label class="form-check-label"
for="firstRadio">@($"{item.Qualityname} / {item.Qualitytype}")</label>
                                <button type="button" class="btn-close"
                                    @onclick="@(value => { if(!this.IsReadonly)
this.Contact.Humanqualities.Remove(item); })" />
                            </li>
                        }
                        </ul>
                    </div>
                </div>
            </div>
                break;
            }
            </div>
        </div>

    </div>
</div>

@code {
    [ParameterAttribute] public DAModels::Contact Contact { get; set; } = default!;
    protected virtual ProfileEditingModel EditingModel { get; set; } = default!;

    [ParameterAttribute] public string? ErrorMessage { get; set; } = default!;
    [ParameterAttribute] public bool IsReadonly { get; set; } = default!;
    [ParameterAttribute] public bool SetDatingType { get; set; } = default!;

    [ParameterAttribute, EditorRequiredAttribute]
    public string FormRequestLink { get; set; } = default!;

    public enum SelectedTabType : sbyte { Details, Employee, QualityHobby  }
    public SelectedTabType SelectedTab { get; set; } = default;

    protected virtual void UserPictureChangeHandler(ChangeEventArgs args)
    {
        this.Contact.Userpicture = new DAModels::Userpicture() { Filepath =
args.Value?.ToString() ?? "" };
        this.Logger.LogInformation($"Picture: {args.Value?.ToString()}");
    }
    protected string ContactCountry
    {
        get => this.Contact.Location!.City.Country; set
{ this.Contact.Location!.City.Country = value; }
```

```
    }
    protected string ContactLocation
    {
        get => this.Contact.Location!.City.Cityname;
        set {
            this.Contact.Location!.City = new DAModels::City()
            {
                Country = this.EditingModel.Cities.First(item => item.Cityname ==
value).Country,
                Cityname = value,
            };
            this.Logger.LogInformation($"City: {ContactLocation}; Country:
{this.Contact.Location!.City.Country}");
        }
    }
    [ParameterAttribute]
    public virtual DAModels::Datingtype DatingType { get; set; } = default!;

    protected virtual bool CurrentEmployeeIsValid { get; set; } = true;
    protected DAModels::Employee CurrentEmployee { get; set; } = default!;

    protected virtual void EmployeeClickHandler()
    {
        if (this.IsReadonly) return;
        var validation = this.Contact.Employees.Where(item => item.Companyname ==
this.CurrentEmployee.Companyname
            && item.Post!.Postname == this.CurrentEmployee.Post!.Postname).Count() <= 0;

        if (this.CurrentEmployee.Companyname.Length < 5 || !validation)
{ CurrentEmployeeIsValid = false; return; }
        this.Contact.Employees.Add(new DAModels::Employee()
        {
            Companyname = this.CurrentEmployee.Companyname, Status =
this.CurrentEmployee.Status,
            Post = new() { Postname = this.CurrentEmployee.Post!.Postname }
        });
        this.CurrentEmployeeIsValid = true;
    }
    protected DAModels::Humanquality CurrentQuality { get; set; } = default!;
    private string CurrentQualityValue
    {
        get => this.CurrentQuality.Qualityname;
        set {
            this.CurrentQuality = new DAModels::Humanquality()
            {
                Qualitytype = this.EditingModel.QualityTypes.First(item =>
item.Qualityname == value).Qualitytype,
                Qualityname = value,
            };
            this.Logger.LogInformation($"Quality: {CurrentQuality.Qualityname}; Type:
{CurrentQuality.Qualitytype}");
        }
    }
    protected virtual void QualityClickHandler()
    {
        if (this.IsReadonly) return;
        if(this.Contact.Humanqualities.Where(item => item.Qualityname ==
this.CurrentQualityValue).Count() <= 0)
        {
            this.Contact.Humanqualities.Add(new DAModels::Humanquality()
            { Qualityname = this.CurrentQualityValue, Qualitytype =
this.CurrentQuality.Qualitytype });
        }
    }
```

```
    protected DAModels::Hobby CurrentHobby { get; set; } = default!;
    private string CurrentHobbyValue
    {
        get => this.CurrentHobby.Hobbyname;
        set {
            this.CurrentHobby = new DAModels::Hobby()
            {
                Hobbytype = this.EditingModel.HobbyTypes.First(item => item.Hobbyname ==
value).Hobbytype,
                Hobbyname = value,
            };
            this.Logger.LogInformation($"Hobby: {CurrentHobby.Hobbyname}; Type:
{CurrentHobby.Hobbytype}");
        }
    }
    protected virtual void HobbyClickHandler()
    {
        if (this.IsReadonly) return;
        if(this.Contact.Hobbies.Where(item => item.Hobbyname ==
this.CurrentHobbyValue).Count() <= 0)
        {
            this.Contact.Hobbies.Add(new DAModels::Hobby()
            { Hobbyname = this.CurrentHobbyValue, Hobbytype =
this.CurrentHobby.Hobbytype });
        }
    }

    protected override void OnInitialized()
    {
        base.OnInitialized();
        this.EditingModel = new ProfileEditingModel();
        using (var dbcontext = this.DatabaseFactory.CreateDbContext())
        {
            this.EditingModel.GenderTypes = dbcontext.Gendertypes.ToList();
            this.EditingModel.Cities = dbcontext.Cities.ToList();
            this.EditingModel.Pictures = dbcontext.Userpictures.ToList();

            this.EditingModel.QualityTypes = dbcontext.Humanqualities.ToList();
            this.EditingModel.HobbyTypes = dbcontext.Hobbies.ToList();
            this.EditingModel.Postes = dbcontext.Posts.ToList();
            this.EditingModel.Datingtypes = dbcontext.Datingtypes.ToList();
        }
        this.EditingModel.Cities.Insert(0, new() { Cityname = "Не установлено", Country =
"Не установлено" });

        this.Contact.Location ??= new DAModels.Location() { City =
this.EditingModel.Cities[0] };
        this.CurrentEmployee = new DAModels::Employee()
        {
            Post = new() { Postname = this.EditingModel.Postes.First().Postname },
            Companyname = string.Empty, Status = "Не указывать",
        };
        this.CurrentQuality = this.EditingModel.QualityTypes.First();
        this.CurrentHobby = this.EditingModel.HobbyTypes.First();

        this.DatingType ??= this.EditingModel.Datingtypes.First();
        this.Logger.LogInformation($"OnInitializedAsync was called");
    }
}
```

```
@using Microsoft.AspNetCore.Components.Forms;
@using Microsoft.AspNetCore.Components.Web;

<button type="button" class="btn btn-success w-100" data-bs-toggle="modal" data-bs-
target="#modal"
```

```
    style="background-color:var(--main-color);border-color:var(--main-
color)">@this.ButtonText</button>

<div class="modal fade" id="modal" tabindex="-1" aria-labelledby="exampleModalLabel"
aria-hidden="true">
    <div class="modal-dialog modal-dialog-centered">
        <div class="modal-content">
            <div class="modal-header">
                <h1 class="modal-title fs-5" id="exampleModalLabel">Подтвердение</h1>
                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
            </div>
            <div class="modal-body"> Сохранение данных. Вы точно уверены? </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Close</button>
                <form method="post" action="@this.FormRequestLink">
                    @this.ChildContent
                    <input type="submit" value="Сохранить" class="btn btn-success w-100"
                        style="background-color:var(--main-color);border-color:var(--
main-color)"/>
                </form>
            </div>
        </div>
    </div>
</div>

@code {
    [ParameterAttribute, EditorRequired] public string ButtonText { get; set; } =
default!;
    [ParameterAttribute] public RenderFragment? ChildContent { get; set; } = default!;

    [ParameterAttribute, EditorRequiredAttribute]
    public string FormRequestLink { get; set; } = default!;
}
@using Microsoft.AspNetCore.Components.Forms;
@using Microsoft.AspNetCore.Components.Web;

<div class="list-group">
    <a @onclick="@(async() => await
ValueChangeHandling(ProfilePage.SelectedTabType.Details))"
        class="list-group-item list-group-item-action"
        style="@this.SetSelectedStyle(ProfilePage.SelectedTabType.Details);font-
size:15px;cursor:pointer;">Контактные данные</a>

    <a @onclick="@(async() => await
ValueChangeHandling(ProfilePage.SelectedTabType.Employee))"
        class="list-group-item list-group-item-action"
        style="@this.SetSelectedStyle(ProfilePage.SelectedTabType.Employee);font-
size:15px;cursor:pointer;">Работа/Учеба</a>

    <a @onclick="@(async() => await
ValueChangeHandling(ProfilePage.SelectedTabType.QualityHobby))"
        class="list-group-item list-group-item-action"
        style="@this.SetSelectedStyle(ProfilePage.SelectedTabType.QualityHobby);font-
size:15px;cursor:pointer;">Хобби/Качества</a>
</div>

@code {
    [ParameterAttribute] public ProfilePage.SelectedTabType ComponentValue { get; set; }
= default!;
    [ParameterAttribute] public EventCallback<ProfilePage.SelectedTabType>
ComponentValueChanged { get; set; }

    protected async Task ValueChangeHandling(ProfilePage.SelectedTabType value)
```

```
    {
        await this.ComponentValueChanged.InvokeAsync(this.ComponentValue = value);
    }
    private string SetSelectedStyle(ProfilePage.SelectedTabType type)
    {
        return this.ComponentValue == type ? "background-color:var(--main-
color);color:white" : string.Empty;
    }
}
@using Microsoft.AspNetCore.Components.Forms;
@using Microsoft.AspNetCore.Components.Web;

<label for="@this.ItemId" class="form-label">@this.LabelTitle:</label>
<div class="@((this.GroupIcon != null ? "input-group" : "")) has-validation">
    @if(this.GroupIcon != null)
    {
        <span class="input-group-text"><i class="@this.GroupIcon"></i></span>
    }
    <input id="@this.ItemId" class="form-control @((this.IsValid ? "" : "is-invalid"))"
maxlength="@this.MaxLength"
        placeholder="@this.Placeholder" type="@this.InputType"
value="@this.componentValue"
        @onchange="@this.ChangeValue" readonly="@this.IsReadonly"/>
</div>

@code {
    [ParameterAttribute] public string LabelTitle { get; set; } = string.Empty;
    [ParameterAttribute] public string ItemId { get; set; } = string.Empty;

    [ParameterAttribute] public string? GroupIcon { get; set; } = default;
    [ParameterAttribute] public string Placeholder { get; set; } = string.Empty;
    [ParameterAttribute] public string InputType { get; set; } = "text";

    [ParameterAttribute] public bool IsReadonly { get; set; } = default;
    [ParameterAttribute] public int MaxLength { get; set; } = 30;
    [ParameterAttribute] public bool IsValid { get; set; } = true;

    protected string componentValue = string.Empty;
    [ParameterAttribute] public string ComponentValue
    {
        get => this.componentValue; set { this.componentValue = value; }
    }
    [ParameterAttribute] public EventCallback<string> ComponentValueChanged { get; set; }

    private async Task ChangeValue(ChangeEventArgs args)
    {
        this.componentValue = args.Value!.ToString()!;
        await this.ComponentValueChanged.InvokeAsync(componentValue);
    }
}
@using Microsoft.AspNetCore.Components.Forms;
@using Microsoft.AspNetCore.Components.Web;

<label for="family" class="form-label">@this.LabelTitle:</label>
<div class="@((this.GroupIcon != null ? "input-group" : "")) has-validation">
    @if(this.GroupIcon != null)
    {
        <span class="input-group-text"><i class="@this.GroupIcon"></i></span>
    }
    <select id="@this.ItemId" class="form-select" @onchange="@(this.ValueChangeHandler)"
disabled="@this.IsReadonly">
        @foreach(var selectedItem in this.ElementList)
        {
            if (selectedItem == this.ComponentValue) { <option value="@selectedItem"
selected>@selectedItem</option> }
```

```
            else { <option value="@selectedItem">@selectedItem</option> }
        }
    </select>
</div>
@code {
    [ParameterAttribute] public string? GroupIcon { get; set; } = default;
    [ParameterAttribute] public string LabelTitle { get; set; } = string.Empty;
    [ParameterAttribute] public string ItemId { get; set; } = string.Empty;

    [ParameterAttribute] public bool IsReadonly { get; set; } = default;
    [ParameterAttribute, EditorRequiredAttribute]
    public List<string> ElementList { get; set; } = new();

    protected string componentValue = string.Empty;
    [ParameterAttribute] public string ComponentValue
    {
        get => this.componentValue; set { this.componentValue = value; }
    }
    [ParameterAttribute] public EventCallback<string> ComponentValueChanged { get; set; }

    protected async Task ValueChangeHandler(ChangeEventArgs args)
    {
        this.componentValue = args.Value!.ToString()!;
        await this.ComponentValueChanged.InvokeAsync(this.componentValue);
    }
}
@model DatabaseAccess.Models.Contact;
@{
    var textOverflow = "overflow:hidden;white-space:nowrap;text-overflow:ellipsis";
    var buttonStyle = "background-color:var(--main-color);border-color:var(--main-
color);";
    var userModel = new UserContactsModel()
    {
        SelectedContact = this.Model.Contactid, IsAccount = (this.Model.Authorization !=
null)
    };
}
<div class="card-body p-3 p-sm-4">
    <h5 class="card-title w-100 d-inline-flex flex-nowrap" style="color:var(--main-
color);font-size:18px;">
        <span class="w-100"
style="@(textOverflow)">@(this.Model.Name) @(this.Model.Surname)</span>
        @if(this.Model.Authorization != null) {
            <span class="badge rounded-pill bg-danger flex-grow-1" style="font-
size:12px;">
                <i class="bi bi-person-fill-check"></i>
            </span>
        }
    </h5>
    <p class="card-text w-100 m-0" style="color:var(--main-color);font-
size:14px;@(textOverflow)">
        Отношения: @this.ViewBag.DatingType
    </p>
    <p class="card-text w-100 mb-0" style="@(textOverflow)"><i class="bi bi-
mailbox"></i> @(this.Model.Emailaddress)</p>
    <p class="card-text w-100 mb-3" style="@(textOverflow)">
        <i class="bi bi-telephone-forward"></i> @((this.Model.Phonenumber ==
default ? "Не установлено" : this.Model.Phonenumber))
    </p>
    <div class="w-100 d-inline-flex input-group" >
        @this.Html.ActionLink("Выбрать", "BuildContact", "UserContacts", userModel, new
{ style=buttonStyle,
            @class="btn btn-primary flex-grow-1"})
    </div>
</div>
```

```
@model List<DatabaseAccess.Models.Contact>
@{ }

<div class="border-top py-3" style="overflow:scroll;max-height:250px;">
    <table class="table">
        <thead>
            <tr style="color: var(--main-color);">
                <th scope="col">Имя:</th> <th scope="col">Фамилия:</th> <th
scope="col">Пол:</th>
                <th scope="col">Email:</th> <th scope="col">Телефон:</th>
            </tr>
        </thead>
        <tbody>
            @foreach(var item in this.Model)
            {
            <tr>
                <th scope="row">@item.Name</th> <td>@item.Surname</td>
                <td>@item.Gendertype.Gendertypename</td> <td>@item.Emailaddress</td>
                <td>@((item.Phonenumber == null ? "Не установлено" :
item.Phonenumber))</td>
            </tr>
            }
        </tbody>
    </table>
</div>

@model List<DatabaseAccess.Models.Datingtype>
@{
    var datingTypes = new SelectList(this.Model.Select(item =>
item.Typeofdating).ToList());
}
<div class="shadow-sm border border-secondary p-4" style="border-radius: 10px;">

    <label class="form-label" style="color:var(--main-color);">Строка для добавления
контакта через ссылку:</label>
    @using (this.Html.BeginForm("UseReferenceLink", "UserContacts", FormMethod.Post, new
{ @class="d-flex flex-row mb-3"}))
    {
    <div class="d-flex input-group flex-grow-1">
        <span class="input-group-text"><i class="bi bi-link-45deg"></i></span>
        @this.Html.TextBox("link", "", new { @class="form-control", placeholder="Ссылка
контакта/Email", maxlength="100", style="flex:2;" })
        @this.Html.DropDownList("datingtype", datingTypes, new { @class="form-select",
style="flex:1;" })
    </div>
    <div class="d-flex mx-2">
        <button type="submit" class="btn btn-success form-control"
            style="background-color:var(--main-color);border-color:var(--main-color);">
<i class="bi bi-plus-circle"></i>
        </button>
    </div>
    }
    <div class="d-flex input-group flex-grow-1">
        <span class="input-group-text flex-grow-1">Создать новый контакт: </span>
        @this.Html.ActionLink("Начать", "BuildContact", "UserContacts", null, new
{ @class="form-control btn btn-primary",
            placeholder="Ссылка контакта/Email", style="background-color:var(--main-
color);border-color:var(--main-color);" })
    </div>
</div>

@model List<DatabaseAccess.Models.Message>
@{
    string ChooseSide(bool check) => (check ? "justify-content-end" : "justify-content-
start");
```

```
}
<div id="messanger" class="border-top py-3" style="overflow:scroll;max-height:200px;">
     @foreach(var item in this.Model)
    {
    <div class="mb-3 d-flex flex-row flex-nowrap @(ChooseSide(item.Contact != null))">
        <div class="shadow-sm border border-secondary p2 p-3" style="border-
radius:16px;max-width:60%;">

            <p class="m-0" style="color:var(--main-color);font-size:18px;">
                @((item.Contact == null ? "Вы:" : ${item.Contact.Name}: "))
            </p>
            <p class="m-0">@item.Messagebody</p>
            <p class="m-0">@item.Sendtime.TimeOfDay.ToString()</p>
        </div>
    </div>
    }
</div>
@using ClientApplication.ViewComponents;
@model GroupbyListViewComponent.GroupbyListViewModel
@{ }
<div class="row">
    <div class="col-12 col-md-6 mb-4 mb-md-0">
        <div class="shadow-sm border border-secondary p-3" style="border-radius: 16px;">
            <label class="form-label mb-2 fs-5 fw-semibold" style="color:var(--main-
color);">
                <i class="bi bi-clipboard-check"></i> Список групп по гендеру:
            </label>
            <div class="border-top" style="overflow:scroll;max-height:600px;">
                <table class="table">
                    <thead style="color:var(--main-color);">
                        <tr><th scope="col">#</th> <th scope="col">Название типа:</th><th
scope="col">Значение:</th></tr>
                    </thead>
                    <tbody>
                    @for(var index = 0; index < this.Model.GenderGroups.Count; index++)
                    {
                        var record = this.Model.GenderGroups[index];
                        <tr> <th scope="row">@(index + 1)</th> <td> @record.GroupName
</td> <td> @record.GroupValue </td> </tr>
                    }
                    </tbody>
                </table>
            </div>
        </div>
    </div>
    <div class="col-12 col-md-6">
        <div class="shadow-sm border border-secondary p-3" style="border-radius: 16px;">
            <label class="form-label mb-2 fs-5 fw-semibold" style="color:var(--main-
color);">
                <i class="bi bi-clipboard-check"></i> Список групп по типу контакта:
            </label>
            <div class="border-top" style="overflow:scroll;max-height:600px;">
                <table class="table">
                    <thead style="color:var(--main-color);">
                        <tr><th scope="col">#</th> <th scope="col">Название типа:</th><th
scope="col">Значение:</th></tr>
                    </thead>
                    <tbody>
                    @for(var index = 0; index < this.Model.AccountTypeGroups.Count;
index++)
                    {
                        var record = this.Model.AccountTypeGroups[index];
                        <tr> <th scope="row">@(index + 1)</th> <td> @record.GroupName
</td> <td> @record.GroupValue </td> </tr>
                    }
```

```
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>
```

```
@model DatabaseAccess.Models.Userpicture
@{ }
<div class="mx-2" style="border-radius:14px;overflow:hidden;">
    <a href="/"> <img src="@this.Model.Filepath" alt="..."
style="width:50px;height:50px;" /> </a>
</div>
```

Файл «DatabaseConfigure.cs»

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Options;
using Microsoft.VisualBasic.FileIO;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;

[assembly: InternalsVisibleToAttribute("ClientApplication")]
namespace DatabaseAccess
{
    public sealed class InitialOptionsMonitor<TOption> : IOptionsMonitor<TOption> where
TOption : class, new()
    {
        public TOption CurrentValue { get; private set; } = default!;
        public InitialOptionsMonitor(TOption currentValue) => this.CurrentValue =
currentValue;

        TOption IOptionsMonitor<TOption>.Get(string? name) => this.CurrentValue;
        public IDisposable OnChange(Action<TOption, string> listener) { return default!; }
    }

    internal partial class DatabaseContextFactory :
IDesignTimeDbContextFactory<DatabaseContext>
    {
        public DatabaseContextFactory() : base() { }
        public interface DatabaseOptionsConfigure<TContext> where TContext : DbContext
        {
            public abstract DbContextOptionsBuilder<TContext> ContextOptions { get; set; }
            public abstract DbContextOptions<TContext> ConfigureOptions(System.String
config_name);
        }
        public partial class DatabaseConfigure :
DatabaseOptionsConfigure<DatabaseContext>
        {
            public DbContextOptionsBuilder<DatabaseContext> ContextOptions { get; set; }
= default!;
            public DatabaseConfigure(DbContextOptionsBuilder<DatabaseContext> options) :
base() => this.ContextOptions = options;

            public DbContextOptions<DatabaseContext> ConfigureOptions(string config_name
= "dbsettings.json")
            {
                var builder = new ConfigurationBuilder();
```

```
                var config =
builder.SetBasePath(AppDomain.CurrentDomain.BaseDirectory).AddJsonFile(config_name).Build
();

                return
this.ContextOptions.UseNpgsql(config.GetConnectionString("DefaultConnection")).Options;
            }
        }
        public virtual DatabaseContext CreateDbContext(string[] args)
        {
            var logger_option = new
InitialOptionsMonitor<DatabaseLoggerConfiguration>(new DatabaseLoggerConfiguration());
            var options_builder = new
DbContextOptionsBuilder<DatabaseAccess.DatabaseContext>();

            var database_options = (new
DatabaseConfigure(options_builder)).ConfigureOptions();
            return new DatabaseContext(database_options, logger_option);
        }
    }
}
```

Файл «DatabaseContext.cs»

```
using System;
using System.Collections.Generic;
using DatabaseAccess.Models;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;

namespace DatabaseAccess;

using LoggerConfiguration = IOptionsMonitor<DatabaseLoggerConfiguration>;
public partial class DatabaseContext : Microsoft.EntityFrameworkCore.DbContext
{
    protected static ILoggerFactory DatabaseLoggerFactory = default(ILoggerFactory)!;
    public DatabaseContext(LoggerConfiguration configuration)
        : base() { base.Database.EnsureCreated(); this.LoggerInitial(configuration); }

    [ActivatorUtilitiesConstructorAttribute()]
    public DatabaseContext(DbContextOptions<DatabaseContext> options, LoggerConfiguration
configuration)
        : base(options) { base.Database.EnsureCreated();
this.LoggerInitial(configuration); }

    public virtual DbSet<Models.Authorization> Authorizations { get; set; } = null!;
    public virtual DbSet<Models.Contact> Contacts { get; set; } = null!;
    public virtual DbSet<Models.Gendertype> Gendertypes { get; set; } = null!;

    public virtual DbSet<Models.City> Cities { get; set; } = null!;
    public virtual DbSet<Models.Location> Locations { get; set; } = null!;

    public virtual DbSet<Models.Post> Posts { get; set; } = null!;
    public virtual DbSet<Models.Employee> Employees { get; set; } = null!;

    public virtual DbSet<Models.Datingtype> Datingtypes { get; set; } = null!;
    public virtual DbSet<Models.Friend> Friends { get; set; } = null!;
    public virtual DbSet<Models.Message> Messages { get; set; } = null!;

    public virtual DbSet<Models.Userpicture> Userpictures { get; set; } = null!;
    public virtual DbSet<Models.Hobby> Hobbies { get; set; } = null!;
    public virtual DbSet<Models.Humanquality> Humanqualities { get; set; } = null!;
```

```
    protected void LoggerInitial(IOptionsMonitor<DatabaseLoggerConfiguration>
configuration)
    {
        DatabaseContext.DatabaseLoggerFactory = LoggerFactory.Create((ILoggingBuilder
builder) =>
        {
            builder.AddFilter((category, level) => category ==
DbLoggerCategory.Database.Command.Name)
                .AddProvider(new DatabaseLoggerProvider(configuration));
        });
    }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseLoggerFactory(DatabaseContext.DatabaseLoggerFactory);
        if (optionsBuilder.IsConfigured == true) return;


optionsBuilder.UseNpgsql("Server=localhost;Database=db_course_work;Username=postgres;Pass
word=prolodgy778;");
    }
    protected override void OnModelCreating(ModelBuilder modelBuilder) =>
this.OnModelCreatingPartial(modelBuilder);
    partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}
```

Файл «DatabaseLogger.cs»

```
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess
{
    public sealed class DatabaseLoggerConfiguration : System.Object
    {
        public DatabaseLoggerConfiguration() : base() { }

        public System.String FilePath { get; set; } = "database.log";
        public Dictionary<LogLevel, System.ConsoleColor> LogLevelColorMap { get; set; } =
new()
        {
            [LogLevel.Information] = ConsoleColor.Green, [LogLevel.Warning] =
ConsoleColor.Yellow,
            [LogLevel.Error] = ConsoleColor.DarkRed
        };
    }
    internal partial class DatabaseLoggerProvider : System.Object, ILoggerProvider
    {
        protected readonly System.IDisposable? _onChangeToken;
        protected virtual DatabaseLoggerConfiguration Configuration { get; private set; }
= new();
        public DatabaseLoggerProvider(IOptionsMonitor<DatabaseLoggerConfiguration>
config) : base()
        {
            this.Configuration = config.CurrentValue;
            this._onChangeToken = config.OnChange((updated_config) => this.Configuration
= updated_config);
        }
        public virtual ILogger CreateLogger(string category_name) => new
DatabaseLoggerProvider.DatabaseLogger(
            category_name, this.GetLoggerConfiguration);
```

```
        protected DatabaseLoggerConfiguration GetLoggerConfiguration() =>
this.Configuration;

        protected class DatabaseLogger : ILogger, System.IDisposable
        {
            protected System.String CategoryName { get; private set; } = default!;
            protected Func<DatabaseLoggerConfiguration> CurrentConfig { get; private
set; } = default!;

            public DatabaseLogger(string category_name, Func<DatabaseLoggerConfiguration>
config) : base()
                => (this.CategoryName, this.CurrentConfig) = (category_name, config);

            public IDisposable? BeginScope<TState>(TState state) where TState : notnull
=> this;
            public bool IsEnabled(LogLevel logLevel) =>
this.CurrentConfig().LogLevelColorMap.ContainsKey(logLevel);
            public void Dispose() { }

            public async void Log<TState>(LogLevel logLevel, EventId eventId, TState
state, Exception? exception,
                Func<TState, Exception?, string> formatter)
            {
                var timestamp_value = string.Format("\n[DateTime]: [{0}]",
DateTime.UtcNow);
                Console.WriteLine(timestamp_value);

                var filepath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
this.CurrentConfig().FilePath);
                using (var file_writer = new StreamWriter(new FileStream(filepath,
FileMode.Append)))
                {
                    await file_writer.WriteLineAsync(timestamp_value);
                    await file_writer.WriteLineAsync($"{formatter(state, exception)}\n");
                }
                Console.ForegroundColor = this.CurrentConfig().LogLevelColorMap[logLevel];

                Console.WriteLine(formatter(state, exception));
                Console.ForegroundColor = ConsoleColor.White; Console.WriteLine();
            }
        }
        public virtual void Dispose() { this._onChangeToken?.Dispose(); }
    }
}
```

Папка «Configurations»

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess.Configurations
{
    public class AuthorizationConfiguration :
IEntityTypeConfiguration<Models.Authorization>
    {
        public AuthorizationConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.Authorization>
entity_builder)
        {
            entity_builder.HasKey(e => e.Authorizationid).HasName("authorization_pkey");
            entity_builder.ToTable("authorization");
```

```
            entity_builder.HasIndex(e => e.Contactid,
"authorization_contactid_key").IsUnique();
            entity_builder.HasIndex(e => e.Login, "authorization_login_key").IsUnique();
            entity_builder.HasIndex(e => e.Referenceguid,
"authorization_referenceguid_key").IsUnique();

            entity_builder.Property(e =>
e.Authorizationid).HasColumnName("authorizationid");
            entity_builder.Property(e => e.Contactid).HasColumnName("contactid");
            entity_builder.Property(e => e.Isadmin).HasColumnName("isadmin");
            entity_builder.Property(e => e.Login)
                .HasMaxLength(30)
                .HasColumnName("login");
            entity_builder.Property(e => e.Password)
                .HasMaxLength(30)
                .HasColumnName("password");
            entity_builder.Property(e => e.Referenceguid)
                .HasMaxLength(40)
                .HasColumnName("referenceguid");

            entity_builder.HasOne(d => d.Contact).WithOne(p => p.Authorization)
                .HasForeignKey<DatabaseAccess.Models.Authorization>(d => d.Contactid)
                .HasConstraintName("authorization_contactid_fkey");
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess.Configurations
{
    public class CityConfiguration : IEntityTypeConfiguration<Models.City>
    {
        public CityConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.City> entity_builder)
        {
            entity_builder.HasKey(e => e.Cityid).HasName("city_pkey");

            entity_builder.ToTable("city");

            entity_builder.HasIndex(e => e.Cityname, "city_cityname_key").IsUnique();

            entity_builder.Property(e => e.Cityid).HasColumnName("cityid");
            entity_builder.Property(e => e.Cityname)
                .HasMaxLength(30)
                .HasColumnName("cityname");
            entity_builder.Property(e => e.Country)
                .HasMaxLength(30)
                .HasColumnName("country");
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace DatabaseAccess.Configurations
{
    public class ContactConfiguration : IEntityTypeConfiguration<Models.Contact>
    {
        public ContactConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.Contact> entity_builder)
        {
            entity_builder.HasKey(e => e.Contactid).HasName("contact_pkey");
            entity_builder.ToTable("contact");

            entity_builder.Property(e => e.Contactid).HasColumnName("contactid");
            entity_builder.Property(e => e.Birthday)
                .HasColumnType("timestamp without time zone")
                .HasColumnName("birthday");
            entity_builder.Property(e => e.Emailaddress)
                .HasMaxLength(100)
                .HasColumnName("emailaddress");
            entity_builder.Property(e => e.Familystatus)
                .HasMaxLength(50)
                .HasDefaultValueSql("NULL::character varying")
                .HasColumnName("familystatus");
            entity_builder.Property(e => e.Gendertypeid).HasColumnName("gendertypeid");
            entity_builder.Property(e => e.Lastupdate)
                .HasDefaultValueSql("'1999-01-08 04:05:06'::timestamp without time zone")
                .HasColumnType("timestamp without time zone")
                .HasColumnName("lastupdate");
            entity_builder.Property(e => e.Locationid).HasColumnName("locationid");
            entity_builder.Property(e => e.Name).HasMaxLength(30);
            entity_builder.Property(e => e.Patronymic)
                .HasMaxLength(30)
                .HasDefaultValueSql("NULL::character varying")
                .HasColumnName("patronymic");
            entity_builder.Property(e => e.Phonenumber)
                .HasMaxLength(12)
                .HasDefaultValueSql("NULL::character varying")
                .HasColumnName("phonenumber");
            entity_builder.Property(e => e.Surname)
                .HasMaxLength(30)
                .HasColumnName("surname");
            entity_builder.Property(e => e.Userpictureid).HasColumnName("userpictureid");

            entity_builder.HasOne(d => d.Gendertype).WithMany(p => p.Contacts)
                .HasForeignKey(d => d.Gendertypeid)
                .HasConstraintName("contact_gendertypeid_fkey");

            entity_builder.HasOne(d => d.Location).WithMany(p => p.Contacts)
                .HasForeignKey(d => d.Locationid)
                .HasConstraintName("contact_locationid_fkey");

            entity_builder.HasOne(d => d.Userpicture).WithMany(p => p.Contacts)
                .HasForeignKey(d => d.Userpictureid)
                .HasConstraintName("contact_userpictureid_fkey");

            entity_builder.HasMany(d => d.Hobbies).WithMany(p => p.Contacts)
                .UsingEntity<Dictionary<string, object>>(
                    "ContactHobby",
                    r => r.HasOne<Models.Hobby>().WithMany()
                        .HasForeignKey("Hobbyid")
                        .HasConstraintName("contact_hobby_hobbyid_fkey"),
                    l => l.HasOne<Models.Contact>().WithMany()
                        .HasForeignKey("Contactid")
                        .HasConstraintName("contact_hobby_contactid_fkey"),
                    j =>
                    {
```

```
                        j.HasKey("Contactid", "Hobbyid").HasName("contact_hobby_pkey");
                        j.ToTable("contact_hobby");
                        j.IndexerProperty<int>("Contactid").HasColumnName("contactid");
                        j.IndexerProperty<int>("Hobbyid").HasColumnName("hobbyid");
                    });
            entity_builder.HasMany(d => d.Humanqualities).WithMany(p => p.Contacts)
                .UsingEntity<Dictionary<string, object>>(
                    "ContactHumanquality",
                    r => r.HasOne<Models.Humanquality>().WithMany()
                        .HasForeignKey("Humanqualityid")
                        .HasConstraintName("contact_humanquality_humanqualityid_fkey"),
                    l => l.HasOne<Models.Contact>().WithMany()
                        .HasForeignKey("Contactid")
                        .HasConstraintName("contact_humanquality_contactid_fkey"),
                    j =>
                    {
                        j.HasKey("Contactid",
"Humanqualityid").HasName("contact_humanquality_pkey");
                        j.ToTable("contact_humanquality");
                        j.IndexerProperty<int>("Contactid").HasColumnName("contactid");

j.IndexerProperty<int>("Humanqualityid").HasColumnName("humanqualityid");
                    });
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess.Configurations
{
    public class DatingTypeConfiguration : IEntityTypeConfiguration<Models.Datingtype>
    {
        public DatingTypeConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.Datingtype> entity_builder)
        {
            entity_builder.HasKey(e => e.Datingtypeid).HasName("datingtype_pkey");

            entity_builder.ToTable("datingtype");

            entity_builder.HasIndex(e => e.Typeofdating,
"datingtype_typeofdating_key").IsUnique();

            entity_builder.Property(e => e.Datingtypeid).HasColumnName("datingtypeid");
            entity_builder.Property(e => e.Typeofdating)
                .HasMaxLength(30)
                .HasColumnName("typeofdating");
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess.Configurations
{
```

```
    public class EmployeeConfiguration : IEntityTypeConfiguration<Models.Employee>
    {
        public EmployeeConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.Employee> entity_builder)
        {
            entity_builder.HasKey(e => e.Employeeid).HasName("employee_pkey");

            entity_builder.ToTable("employee");

            entity_builder.Property(e => e.Employeeid).HasColumnName("employeeid");
            entity_builder.Property(e => e.Companyname)
                .HasMaxLength(50)
                .HasColumnName("companyname");
            entity_builder.Property(e => e.Contactid).HasColumnName("contactid");
            entity_builder.Property(e => e.Postid).HasColumnName("postid");
            entity_builder.Property(e => e.Status)
                .HasMaxLength(30)
                .HasDefaultValueSql("NULL::character varying")
                .HasColumnName("status");

            entity_builder.HasOne(d => d.Contact).WithMany(p => p.Employees)
                .HasForeignKey(d => d.Contactid)
                .HasConstraintName("employee_contactid_fkey");

            entity_builder.HasOne(d => d.Post).WithMany(p => p.Employees)
                .HasForeignKey(d => d.Postid)
                .HasConstraintName("employee_postid_fkey");
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess.Configurations
{
    public class FriendConfiguration : IEntityTypeConfiguration<Models.Friend>
    {
        public FriendConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.Friend> entity_builder)
        {
            entity_builder.HasKey(e => new { e.Friendid, e.Contactid1,
e.Contactid2 }).HasName("friends_pkey");

            entity_builder.ToTable("friends");

            entity_builder.HasIndex(e => e.Friendid, "friends_friendid_key").IsUnique();

            entity_builder.Property(e => e.Friendid)
                .ValueGeneratedOnAdd()
                .HasColumnName("friendid");
            entity_builder.Property(e => e.Contactid1).HasColumnName("contactid1");
            entity_builder.Property(e => e.Contactid2).HasColumnName("contactid2");
            entity_builder.Property(e => e.Datingtypeid).HasColumnName("datingtypeid");
            entity_builder.Property(e => e.Starttime).HasColumnName("starttime");

            entity_builder.HasOne(d => d.Contactid1Navigation).WithMany(p =>
p.FriendContactid1Navigations)
                .HasForeignKey(d => d.Contactid1)
                .HasConstraintName("friends_contactid1_fkey");
```

```
            entity_builder.HasOne(d => d.Contactid2Navigation).WithMany(p =>
p.FriendContactid2Navigations)
                .HasForeignKey(d => d.Contactid2)
                .HasConstraintName("friends_contactid2_fkey");

            entity_builder.HasOne(d => d.Datingtype).WithMany(p => p.Friends)
                .HasForeignKey(d => d.Datingtypeid)
                .HasConstraintName("friends_datingtypeid_fkey");
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess.Configurations
{
    public class GenderTypeConfiguration : IEntityTypeConfiguration<Models.Gendertype>
    {
        public GenderTypeConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.Gendertype> entity_builder)
        {
            entity_builder.HasKey(e => e.Gendertypeid).HasName("gendertype_pkey");

            entity_builder.ToTable("gendertype");

            entity_builder.HasIndex(e => e.Gendertypename,
"gendertype_gendertypename_key").IsUnique();

            entity_builder.Property(e => e.Gendertypeid).HasColumnName("gendertypeid");
            entity_builder.Property(e => e.Gendertypename)
                .HasMaxLength(20)
                .HasColumnName("gendertypename");
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess.Configurations
{
    public class HobbyConfiguration : IEntityTypeConfiguration<Models.Hobby>
    {
        public HobbyConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.Hobby> entity_builder)
        {
            entity_builder.HasKey(e => e.Hobbyid).HasName("hobby_pkey");

            entity_builder.ToTable("hobby");

            entity_builder.HasIndex(e => e.Hobbyname, "hobby_hobbyname_key").IsUnique();

            entity_builder.Property(e => e.Hobbyid).HasColumnName("hobbyid");
            entity_builder.Property(e => e.Hobbyname)
                .HasMaxLength(30)
                .HasColumnName("hobbyname");
```

```
            entity_builder.Property(e => e.Hobbytype)
                .HasMaxLength(30)
                .HasDefaultValueSql("NULL::character varying")
                .HasColumnName("hobbytype");
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess.Configurations
{
    public class HumanQualityConfiguration :
IEntityTypeConfiguration<Models.Humanquality>
    {
        public HumanQualityConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.Humanquality>
entity_builder)
        {
            entity_builder.HasKey(e => e.Humanqualityid).HasName("humanquality_pkey");

            entity_builder.ToTable("humanquality");

            entity_builder.HasIndex(e => e.Qualityname,
"humanquality_qualityname_key").IsUnique();

            entity_builder.Property(e =>
e.Humanqualityid).HasColumnName("humanqualityid");
            entity_builder.Property(e => e.Qualityname)
                .HasMaxLength(30)
                .HasColumnName("qualityname");
            entity_builder.Property(e => e.Qualitytype)
                .HasMaxLength(30)
                .HasColumnName("qualitytype");
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess.Configurations
{
    public class LocationConfiguration : IEntityTypeConfiguration<Models.Location>
    {
        public LocationConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.Location> entity_builder)
        {
            entity_builder.HasKey(e => e.Locationid).HasName("location_pkey");
            entity_builder.ToTable("location");
            entity_builder.HasIndex(e => e.Street, "location_street_key").IsUnique();

            entity_builder.Property(e => e.Locationid).HasColumnName("locationid");
            entity_builder.Property(e => e.Cityid).HasColumnName("cityid");
            entity_builder.Property(e => e.Street)
                .HasMaxLength(50)
```

```
                .HasDefaultValueSql("NULL::character varying")
                .HasColumnName("street");

            entity_builder.HasOne(d => d.City).WithMany(p => p.Locations)
                .HasForeignKey(d => d.Cityid)
                .HasConstraintName("location_cityid_fkey");
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess.Configurations
{
    public class MessageConfiguration : IEntityTypeConfiguration<Models.Message>
    {
        public MessageConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.Message> entity_builder)
        {
            entity_builder.HasKey(e => e.Messageid).HasName("message_pkey");

            entity_builder.ToTable("message");

            entity_builder.Property(e => e.Messageid).HasColumnName("messageid");
            entity_builder.Property(e => e.Contactid).HasColumnName("contactid");
            entity_builder.Property(e => e.Friendid).HasColumnName("friendid");
            entity_builder.Property(e => e.Messagebody).HasColumnName("messagebody");
            entity_builder.Property(e => e.Sendtime)
                .HasColumnType("timestamp without time zone")
                .HasColumnName("sendtime");

            entity_builder.HasOne(d => d.Contact).WithMany(p => p.Messages)
                .HasForeignKey(d => d.Contactid)
                .HasConstraintName("message_contactid_fkey");

            entity_builder.HasOne(d => d.Friend).WithMany(p => p.Messages)
                .HasPrincipalKey(p => p.Friendid)
                .HasForeignKey(d => d.Friendid)
                .HasConstraintName("message_friendid_fkey");
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess.Configurations
{
    public class PostConfiguration : IEntityTypeConfiguration<Models.Post>
    {
        public PostConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.Post> entity_builder)
        {
            entity_builder.HasKey(e => e.Postid).HasName("post_pkey");

            entity_builder.ToTable("post");
```

```
            entity_builder.HasIndex(e => e.Postname, "post_postname_key").IsUnique();

            entity_builder.Property(e => e.Postid).HasColumnName("postid");
            entity_builder.Property(e => e.Postname)
                .HasMaxLength(50)
                .HasColumnName("postname");
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseAccess.Configurations
{
    public class UserPictureConfiguration : IEntityTypeConfiguration<Models.Userpicture>
    {
        public UserPictureConfiguration() : base() { }
        public virtual void Configure(EntityTypeBuilder<Models.Userpicture>
entity_builder)
        {
            entity_builder.HasKey(e => e.Userpictureid).HasName("userpicture_pkey");

            entity_builder.ToTable("userpicture");

            entity_builder.Property(e => e.Userpictureid)
                .ValueGeneratedNever()
                .HasColumnName("userpictureid");
            entity_builder.Property(e => e.Filepath).HasColumnName("filepath");
            entity_builder.Property(e => e.Picturename)
                .HasMaxLength(50)
                .HasColumnName("picturename");
        }
    }
}
```

Папка «Models»

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using DatabaseAccess.Configurations;

namespace DatabaseAccess.Models;

[EntityTypeConfiguration(typeof(AuthorizationConfiguration))]
public partial class Authorization : System.Object
{
    public int Authorizationid { get; set; } = default;
    public string Login { get; set; } = default!;

    public string Password { get; set; } = default!;
    public int Contactid { get; set; } = default;
    public bool Isadmin { get; set; } = default;

    public string Referenceguid { get; set; } = null!;
    public virtual Contact Contact { get; set; } = null!;
}
using DatabaseAccess.Configurations;
using Microsoft.EntityFrameworkCore;
using System;
```

```
using System.Collections.Generic;

namespace DatabaseAccess.Models;

[EntityTypeConfiguration(typeof(CityConfiguration))]
public partial class City : System.Object
{
    public int Cityid { get; set; } = default;

    public string Cityname { get; set; } = null!;
    public string Country { get; set; } = null!;

    public virtual ICollection<Location> Locations { get; set; } = new List<Location>();
}
using DatabaseAccess.Configurations;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;

namespace DatabaseAccess.Models;

[EntityTypeConfiguration(typeof(ContactConfiguration))]
public partial class Contact : System.Object
{
    public int Contactid { get; set; } = default;

    public string Surname { get; set; } = null!;
    public string Name { get; set; } = null!;
    public string? Patronymic { get; set; } = default;

    public string? Familystatus { get; set; } = default;
    public DateTime Birthday { get; set; } = default;

    public string Emailaddress { get; set; } = null!;
    public string? Phonenumber { get; set; } = default;
    public DateTime Lastupdate { get; set; } = default;

    public int? Locationid { get; set; } = default;
    public int Gendertypeid { get; set; } = default;
    public int? Userpictureid { get; set; } = default;

    public virtual ICollection<Employee> Employees { get; set; } = new List<Employee>();
    public virtual ICollection<Friend> FriendContactid1Navigations { get; set; } = new
List<Friend>();
    public virtual ICollection<Friend> FriendContactid2Navigations { get; set; } = new
List<Friend>();

    public virtual Gendertype Gendertype { get; set; } = null!;
    public virtual Location? Location { get; set; }

    public virtual Authorization? Authorization { get; set; }
    public virtual Userpicture? Userpicture { get; set; }

    public virtual ICollection<Message> Messages { get; set; } = new List<Message>();
    public virtual ICollection<Hobby> Hobbies { get; set; } = new List<Hobby>();
    public virtual ICollection<Humanquality> Humanqualities { get; set; } = new
List<Humanquality>();
}
using DatabaseAccess.Configurations;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;

namespace DatabaseAccess.Models;
```

```
[EntityTypeConfiguration(typeof(DatingTypeConfiguration))]
public partial class Datingtype : System.Object
{
    public int Datingtypeid { get; set; } = default;
    public string Typeofdating { get; set; } = null!;

    public virtual ICollection<Friend> Friends { get; set; } = new List<Friend>();
}
using DatabaseAccess.Configurations;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;

namespace DatabaseAccess.Models;

[EntityTypeConfiguration(typeof(EmployeeConfiguration))]
public partial class Employee : System.Object
{
    public int Employeeid { get; set; } = default;

    public string Companyname { get; set; } = null!;
    public string? Status { get; set; } = default;

    public int? Postid { get; set; } = default;
    public int Contactid { get; set; } = default;

    public virtual Contact Contact { get; set; } = null!;
    public virtual Post? Post { get; set; }
}
using DatabaseAccess.Configurations;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;

namespace DatabaseAccess.Models;

[EntityTypeConfiguration(typeof(FriendConfiguration))]
public partial class Friend : System.Object
{
    public int Friendid { get; set; } = default;
    public DateOnly Starttime { get; set; } = default;

    public int? Datingtypeid { get; set; } = default;
    public int Contactid1 { get; set; } = default;
    public int Contactid2 { get; set; } = default;

    public virtual Contact Contactid1Navigation { get; set; } = null!;
    public virtual Contact Contactid2Navigation { get; set; } = null!;

    public virtual Datingtype? Datingtype { get; set; } = default;
    public virtual ICollection<Message> Messages { get; set; } = new List<Message>();
}
using DatabaseAccess.Configurations;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;

namespace DatabaseAccess.Models;

[EntityTypeConfiguration(typeof(GenderTypeConfiguration))]
public partial class Gendertype : System.Object
{
    public int Gendertypeid { get; set; } = default;

    public string Gendertypename { get; set; } = null!;
```

```
    public virtual ICollection<Contact> Contacts { get; set; } = new List<Contact>();
}
using DatabaseAccess.Configurations;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;

namespace DatabaseAccess.Models;

[EntityTypeConfiguration(typeof(HobbyConfiguration))]
public partial class Hobby : System.Object
{
    public int Hobbyid { get; set; } = default;

    public string Hobbyname { get; set; } = null!;
    public string? Hobbytype { get; set; } = default;

    public virtual ICollection<Contact> Contacts { get; set; } = new List<Contact>();
}
using DatabaseAccess.Configurations;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;

namespace DatabaseAccess.Models;

[EntityTypeConfiguration(typeof(HumanQualityConfiguration))]
public partial class Humanquality : System.Object
{
    public int Humanqualityid { get; set; } = default;

    public string Qualityname { get; set; } = null!;
    public string Qualitytype { get; set; } = null!;

    public virtual ICollection<Contact> Contacts { get; set; } = new List<Contact>();
}
using DatabaseAccess.Configurations;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Diagnostics;

namespace DatabaseAccess.Models;

[EntityTypeConfiguration(typeof(LocationConfiguration))]
public partial class Location : System.Object
{
    public int Locationid { get; set; } = default!;
    public string? Street { get; set; } = default!;

    public int Cityid { get; set; } = default!;
    public virtual City City { get; set; } = null!;
    public virtual ICollection<Contact> Contacts { get; set; } = new List<Contact>();
}
using DatabaseAccess.Configurations;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;

namespace DatabaseAccess.Models;

[EntityTypeConfiguration(typeof(MessageConfiguration))]
public partial class Message : System.Object
{
    public int Messageid { get; set; } = default;
```

```
    public string Messagebody { get; set; } = null!;
    public DateTime Sendtime { get; set; }

    public int Friendid { get; set; } = default;
    public int Contactid { get; set; } = default;

    public virtual Contact Contact { get; set; } = null!;
    public virtual Friend Friend { get; set; } = null!;
}
using DatabaseAccess.Configurations;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;

namespace DatabaseAccess.Models;

[EntityTypeConfiguration(typeof(PostConfiguration))]
public partial class Post : System.Object
{
    public int Postid { get; set; } = default;
    public string Postname { get; set; } = null!;

    public virtual ICollection<Employee> Employees { get; set; } = new List<Employee>();
}
using DatabaseAccess.Configurations;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;

namespace DatabaseAccess.Models;

[EntityTypeConfiguration(typeof(UserPictureConfiguration))]
public partial class Userpicture : System.Object
{
    public int Userpictureid { get; set; } = default;

    public string Filepath { get; set; } = null!;
    public string Picturename { get; set; } = null!;

    public virtual ICollection<Contact> Contacts { get; set; } = new List<Contact>();
}
```

ПРИЛОЖЕНИЕ Б

Листинг программы на SQL

```
-- Создание таблиц базы данных
CREATE TABLE IF NOT EXISTS public.City (
    CityId      SERIAL      PRIMARY KEY NOT NULL,
    CityName    VARCHAR(30) NOT NULL UNIQUE,
    Country     VARCHAR(30) NOT NULL
);

CREATE TABLE IF NOT EXISTS public.Location (
    LocationId  SERIAL      PRIMARY KEY NOT NULL,
    Street      VARCHAR(50) DEFAULT NULL,
    CityId      INTEGER     NOT NULL,
    FOREIGN KEY(CityId) REFERENCES public.City(CityId) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS public.GenderType (
    GenderTypeId    SERIAL      PRIMARY KEY NOT NULL,
    GenderTypeName  VARCHAR(20) NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS public.UserPicture (
    UserPictureId   SERIAL     NOT NULL PRIMARY KEY,
    FilePath        TEXT        NOT NULL,
    pictureName     VARCHAR(50) NOT NULL
);

CREATE TABLE IF NOT EXISTS public.Contact (
    ContactId       SERIAL          PRIMARY KEY NOT NULL,
    Surname         VARCHAR(30)     NOT NULL,
    "Name"          VARCHAR(30)     NOT NULL,
    Patronymic      VARCHAR(30)     DEFAULT NULL,
    PhoneNumber     VARCHAR(12)     DEFAULT NULL,
    Birthday        TIMESTAMP       NOT NULL,
    FamilyStatus    VARCHAR(50)     DEFAULT NULL,
    EmailAddress    VARCHAR(100)    NOT NULL,
    UserPictureId   INTEGER         DEFAULT NULL,
    LocationId      INTEGER         DEFAULT NULL,
    GenderTypeId    INTEGER         NOT NULL,
    FOREIGN KEY (LocationId)    REFERENCES public.Location(LocationId) ON DELETE SET
DEFAULT,
```

```
    FOREIGN KEY (GenderTypeId)  REFERENCES public.GenderType(GenderTypeId) ON DELETE
CASCADE,
    FOREIGN  KEY  (UserPictureId)  REFERENCES  public.UserPicture(UserPictureId)  ON
DELETE SET DEFAULT
);

ALTER TABLE public.Contact ADD COLUMN LastUpdate TIMESTAMP NOT NULL DEFAULT '1999-
01-08 04:05:06';

CREATE TABLE IF NOT EXISTS public.Authorization (
    AuthorizationId SERIAL      NOT NULL PRIMARY KEY,
    Login           VARCHAR(30) NOT NULL UNIQUE,
    Password        VARCHAR(30) NOT NULL,
    ContactId       INTEGER     NOT NULL UNIQUE,
    IsAdmin         BOOLEAN     NOT NULL,
    ReferenceGuid   VARCHAR(40) NOT NULL UNIQUE,
    FOREIGN KEY (ContactId) REFERENCES public.Contact(ContactId) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS public.Post (
    PostId      SERIAL      PRIMARY KEY NOT NULL,
    PostName    VARCHAR(50) NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS public.Employee (
    EmployeeId  SERIAL      NOT NULL PRIMARY KEY,
    CompanyName VARCHAR(50) NOT NULL,
    Status      VARCHAR(30) DEFAULT NULL,
    PostId      INTEGER     DEFAULT NULL,
    ContactId   INTEGER     NOT NULL,
    FOREIGN KEY (PostId)    REFERENCES public.Post(PostId) ON DELETE SET DEFAULT,
    FOREIGN KEY (ContactId) REFERENCES public.Contact(ContactId) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS public.DatingType (
    DatingTypeId    SERIAL      NOT NULL PRIMARY KEY,
    TypeOfDating    VARCHAR(30) NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS public.Friends (
    FriendId        SERIAL  NOT NULL UNIQUE,
```

```
    StartTime       DATE    NOT NULL,
    DatingTypeId    INTEGER DEFAULT NULL,
    ContactId1      INTEGER NOT NULL,
    ContactId2      INTEGER NOT NULL,
    PRIMARY KEY (FriendId, ContactId1, ContactId2),
    FOREIGN KEY (DatingTypeId) REFERENCES public.DatingType(DatingTypeId) ON DELETE
SET DEFAULT,
    FOREIGN KEY (ContactId1)      REFERENCES public.Contact(ContactId) ON DELETE
CASCADE,
    FOREIGN KEY (ContactId2)      REFERENCES public.Contact(ContactId) ON DELETE
CASCADE
);

CREATE TABLE IF NOT EXISTS public.Message (
    MessageId   SERIAL      NOT NULL PRIMARY KEY,
    MessageBody TEXT        NOT NULL,
    SendTime    TIMESTAMP   NOT NULL,
    FriendId    INTEGER     NOT NULL,
    ContactId   INTEGER     NOT NULL,
    FOREIGN KEY (FriendId)  REFERENCES public.Friends(FriendId) ON DELETE CASCADE,
    FOREIGN KEY (ContactId) REFERENCES public.Contact(ContactId) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS public.HumanQuality (
    HumanQualityId  SERIAL      NOT NULL PRIMARY KEY,
    QualityName     VARCHAR(30) NOT NULL UNIQUE,
    QualityType     VARCHAR(30) NOT NULL
);

CREATE TABLE IF NOT EXISTS public.Hobby (
    HobbyId     SERIAL      NOT NULL PRIMARY KEY,
    HobbyName   VARCHAR(30) NOT NULL UNIQUE ,
    HobbyType   VARCHAR(30) DEFAULT NULL
);

CREATE TABLE IF NOT EXISTS public.Contact_HumanQuality (
    ContactId       INTEGER NOT NULL,
    HumanQualityId  INTEGER NOT NULL,
    PRIMARY KEY (ContactId, HumanQualityId),
    FOREIGN KEY (ContactId)         REFERENCES public.Contact(ContactId) ON DELETE
CASCADE,
```

```
    FOREIGN KEY (HumanQualityId) REFERENCES public.HumanQuality(HumanQualityId) ON
DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS public.Contact_Hobby (
    ContactId   INTEGER NOT NULL,
    HobbyId     INTEGER NOT NULL,
    PRIMARY KEY (ContactId, HobbyId),
    FOREIGN KEY (ContactId) REFERENCES public.Contact(ContactId) ON DELETE CASCADE,
    FOREIGN KEY (HobbyId)   REFERENCES public.Hobby(HobbyId) ON DELETE CASCADE
);
-- Заполнение таблиц данными
INSERT INTO public.GenderType(GenderTypeName) VALUES ('Мужчина'), ('Женщина'),
('Неизвестно');

INSERT INTO public.City(CityName, Country) VALUES ('Воронеж', 'Россия'), ('Липецк',
'Россия'),
    ('Москва', 'Россия'), ('Санкт-Петербург', 'Россия'), ('Екатеринбург', 'Россия'),
    ('Минск', 'Белоруссия'), ('Брест', 'Белоруссия'), ('Алматы', 'Казахстан'),
('Уральск', 'Казахстан'),
    ('Самара', 'Россия'), ('Астана', 'Казахстан'), ('Полоцк', 'Белоруссия');

INSERT INTO public.Post(PostName) VALUES ('Менеджер'), ('Администратор'),
('Студент'),
    ('Практикант'), ('Кладовщик'), ('Уборщик'), ('Кассир'), ('Учитель'), ('Охраник'),
    ('Программист'), ('Диспетчер'), ('Механик'), ('Актер');

INSERT INTO public.DatingType(TypeOfDating) VALUES ('Друг'), ('Знакомый'),
('Коллега'), ('Семья');

INSERT INTO public.HumanQuality(QualityName, QualityType) VALUES ('Доброта',
'Положительный'),
    ('Честность', 'Положительный'), ('Верность', 'Положительный'), ('Отзывчивость',
'Положительный'),
    ('Щедрость', 'Положительный'), ('Юмор', 'Положительный'), ('Злость',
'Отрицательный'),
    ('Зависть', 'Отрицательный'), ('Высокомерие', 'Отрицательный'), ('Лицемерие',
'Отрицательный');

INSERT INTO public.Hobby(HobbyName, HobbyType) VALUES ('Чтение', 'Пассивный'),
('Туризм', 'Активный'),
```

```
    ('Охота', 'Активный'), ('Рыбалка', 'Активный'), ('Танцы', 'Активный'), ('Спорт',
'Активный'),
    ('Ролевые игры', 'Активный'), ('Собирательство', 'Пассивный'), ('Цветоводство',
'Пассивный'),
    ('Садоводство',   'Пассивный'),   ('Фотографии',   'Пассивный'),   ('Кулинария',
'Пассивный'), ('Пение', 'Пассивный');
```