Пермский филиал федерального государственного автономного
образовательного учреждения высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»

Факультет экономики, менеджмента и бизнес-информатики

Пономарёв Михаил Владимирович

РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ ПРЕДПРИЯТИЯ ДЛЯ ПРИНЯТИЯ И ОБРАБОТКИ ЗАЯВОК КЛИЕНТОВ

Курсовая работа

студента образовательной программы бакалавриата «Программная инженерия» по направлению подготовки *09.03.04 Программная инженерия*

Руководитель
к.т.н., доцент кафедры
информационных
технологий в бизнесе
О.Л. Викентьева

Пермь, 2022 год

АННОТАЦИЯ

Курсовая работа на тему «Разработка информационной системы предприятия для принятия и обработки заявок клиентов».

Автор работы: Пономарёв Михаил Владимирович.

Во введении обоснована актуальность работы, поставлена цель работы, определены задачи работы, указаны объект и предмет исследования.

В первой главе разобраны теоретические основы систем принятия и обработки заявок клиентов, найдены и проанализированы существующие системы. Во второй главе сформулированы функциональные требования разрабатываемой системы и спроектировано будущее решение. В третьей главе описаны этапы разработки системы и ее тестирование.

Заключение содержит описание результатов, полученных в ходе выполнения работы.

Работа состоит из 68 страниц основного текста и 5 страниц приложений, содержит 34 иллюстрации, 21 таблиц, библиография - 13 наименований, 11 приложений.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Многим организациям требуется система принятия и обработки заявок от клиентов. В зависимости от потребностей организации, перед такой системой могут стоять различные задачи, например:

- техническая поддержка клиентов;
- предварительная запись на услуги;
- обратная связь с клиентами.

Классическим вариантом такой системы являются взаимосвязь посредством телефонных звонков, когда клиент может позвонить по определенному номеру, задать вопрос, оставить заявку или решить свою проблему. Альтернативным вариантом является электронная система, например, на основе веб-сайта, которая может предоставить клиенту больше возможностей, например прикрепить какие-либо электронные документы к заявке, для подобной системы не нужен сотрудник, который будет принимать заявки в режиме реального времени. Как правило, организации предоставляют своим клиентам оба варианта. Система для принятия и обработки заявок значительно упрощает дистанционное взаимодействие с клиентами и является одним из инструментов для анализа и контроля работы организации. Из вышеперечисленного следует, что данная тема курсовой работы актуальна.

Объектом исследования работы является процесс взаимодействия организации со своими клиентами.

Предметом исследования работы является процесс принятия и обработки заявок клиентов.

Целью исследования является разработка информационной системы для принятия и обработки заявок клиентов на основе веб-приложения.

Для выполнения поставленной цели необходимо выполнить следующие задачи:

- исследовать теоретические аспекты вопроса;
- проанализировать существующие системы принятия и обработки заявок клиентов;
- сформировать функциональные требования;

- спроектировать приложение и базу данных;
- реализовать приложение и базу данных.

Курсовая работа состоит из трех глав. В первой главе разобраны теоретические основы систем принятия и обработки заявок клиентов, найдены и проанализированы существующие системы. Во второй главе сформулированы функциональные требования разрабатываемой системы и спроектировано будущее решение. В третьей главе описаны этапы разработки базы данных, серверного и клиентского приложения.

Практическая значимость работы заключается в том, что разработанная информационная система будет достаточно универсальным решением для принятия и обработки заявок клиентов, что позволит ее внедрить в существующую информационную структуру организации с минимальными усилиями.

ГЛАВА 1 АНАЛИЗ СУЩЕСТВУЮЩИХ ИНФОРМАЦИОННЫХ СИСТЕМ ПРИНЯТИЯ И ОБРАБОТКИ ЗАЯВОК КЛИЕНТОВ

1.1 Классификация информационных систем принятия и обработки заявок клиентов

В сегодняшних реалиях связь потребителей услуги с ее поставщиком, либо связь представителя службы технической поддержки с клиентом, почти всегда осуществляется через программную оболочку. Это позволяет сделать процесс коммуникации более удобным для обеих сторон, но что не менее важно - сделать его прозрачным, отслеживаемым и контролируемым. В качестве программных оболочек могут выступать следующие типы систем:

- CRM;
- Help Desk.

CRM

CRM-система (Customer Relationship Management или Управление отношениями с клиентами) - это прикладное программное обеспечение для организаций, предназначенное для автоматизации стратегий взаимодействия с заказчиками (клиентами), в частности, для повышения уровня продаж, оптимизации маркетинга и улучшения обслуживания клиентов путем сохранения информации о клиентах и истории взаимоотношений с ними, установления и улучшения бизнес-процессов и последующего анализа результатов.

Подавляющее большинство CRM систем нацелены на решение 3 основных задач:

- учет всех лидов (контактов, так или иначе заинтересованных в вашем товаре/услуге);
- учет всех взаимодействий с лидом (звонки, письма, ответы в социальных сетях и чатах и т.д.);
- автоматизация процессов взаимодействия с целью максимизации перехода лидов в продажи (конверсия).

Таким образом CRM система в первую очередь является основным инструментом для «продажников» и их руководителей.

Как следствие, основным объектом большинства CRM являются сделки и задачи, которые необходимо выполнять «продажнику» для прохождения сделки по воронке. В рамках процессов продаж могут возникать задачи для коллег из смежных отделов: маркетинга, бухгалтерии или даже внедрения, но это, скорее, вторичные активности.

Основная работа в CRM завершается в момент перехода лида в клиента, то есть в момент оплаты Вашего товара или услуги. Далее в зависимости от бизнеса и его специфики, с клиентом может:

- не быть никаких взаимодействий;
- быть передача в отдел внедрения - если это проект;
- передача клиента на сопровождение (абонентское обслуживание).

Help Desk

Help Desk системы можно назвать CRM, но только ориентированы не на продажи, а на допродажи и снижение оттока. Из-за этого может возникнуть путаница, так как и там и там есть термин «заявка от клиента / клиентская заявка», который одинаково подходит как для первичного обращения и «входа» в процесс продаж, а значит в CRM, так и для постпродажного обслуживания и «входа» в Help Desk. На этом этапе основная работа происходит не с лидом (гипотетическим клиентом), а заказчиком на договорной основе в случае B2B (Business to Business). Именно в этот момент появляются количественные и качественные обязательства, которые часто называются SLA (Service Level Agreement) и которые мы должны перед заказчиком соблюдать. Цель этих обязательств - поддерживать работоспособность предоставляемых и поддерживаемых услуг клиента таким образом, чтобы минимизировать влияние негативных последствий на бизнес. Починить то, что сломалось, заменить то, что вышло из строя, помочь выполнять свою работу в «типовом» режиме.

В рамках решения клиентских заявок процессы становятся более сложными, чем процесс продажи. В том числе, потому что ответственное лицо не аккаунт, а специалисты различных квалификаций, а иногда и привлекаемые сторонние подрядчики, вендоры и т.д. И, главное, у этих процессов есть обязательные временные параметры решения - те самые SLA.

Упрощенно, цель любой Help Desk системы, помимо исключения потери заявок от клиентов, максимально быстро решить возникшую проблему. Это в первую очередь влияет и на конечную удовлетворенность клиента. Если его заявки решаются в соответствии с его ожиданиями и, в идеале, обязательствами исполнителя, он будет доволен и не будет пытаться найти другого поставщика на постпродажное обслуживание.

Help Desk системы можно разделить на 3 основных класса:

- Решения для автоматизации процессов предоставления ИТ услуг «внутри» компании (их еще называют ITSM решения).
- Решения для автоматизации массовой поддержки физических лиц (так называемые Customer Service или Customer Support решения).
- Решения для автоматизации внешней поддержки в сервисных компаниях, работающих по договорам абонентского и постпродажного обслуживания (B2B решения).

ITSM решения

Управление ИТ-услугами (или ITSM) - это принятый ИТ-командой способ управления комплексным предоставлением ИТ-услуг клиентам. Он охватывает все процессы и действия по проектированию, созданию, предоставлению и поддержке ИТ-услуг.

Основной принцип ITSM заключается в том, что ИТ-продукты должны предоставляться в виде услуг. Типичный сценарий ITSM может включать запрос на новое оборудование, например ноутбук. Пользователь отправляет запрос через портал, заполняя заявку с указанием всей необходимой информации, и тем самым запускает воспроизводимый рабочий процесс. Заявка помещается в очередь ИТ-команды, где входящие запросы сортируются и рассматриваются в соответствии с их важностью.

Customer Service решения

Customer Service инструменты ориентированы на массовое обслуживание по типовым вопросам. Там, где важно фиксировать обращения от кого угодно по всем удобным пользователю каналам и предлагать инструменты массового решения

(шаблоны ответов). Такие системы очень популярны при взаимодействии с физическими лицами в банках, интернет-магазинах и т.д.

B2B решения

Эти системы подходят для компаний, в которых важно:

- фиксировать клиентские заявки, в том числе, в привязке к локациям или объектам обслуживания;
- соответствовать договорным обязательствам и SLA;
- вести учет поддерживаемого клиентского оборудования и ПО;
- уметь оказывать и разовые работы или услуги за дополнительную плату по прайс-листу;
- строить прозрачные отношения с заказчиком (особенность b2b поддержки) и т.д.

Компании, где сервисные услуги и есть бизнес, или как минимум его большая и важная часть, например:

- Обслуживание специализированного оборудования и ПО (оргтехника, видеонаблюдение, домофоны, шлагбаумы, лифты и так далее).
- Разработчики и вендоры.
- ИТ-аутсорсинг.
- Эксплуатация коммерческой недвижимости (бизнес- и торговых центров).
- Digital-агентства и веб-студии.

Разработанная система относится к типу Help Desk и к классу Customer Service, потому что нацелена на решение типовых вопросов от физических лиц.

Среди готовых систем типа Help Desk можно выделить следующие примеры:

- Еадеск;
- Юздеск;
- Kayako.

1.2 Анализ информационной системы Еадеск

Это инструмент для работы с входящими обращениями, и он полезен небольшим командам и любым компаниям, работающим с клиентами. Помогает организовать нормальную поддержку клиентов: перестать сливать заявки, увеличить повторные продажи за счёт качественного сервиса, сократить расходы на сотрудников за счёт автоматизации и управлять репутацией в сети.

Еадеск объединяет обращения из всех источников в одном окне: мессенджеры, соцсети, почта и телефон. Распределит обращения и даст инструменты совместной работы. Поможет контролировать сотрудников и отслеживать показатели работы. Официальный сайт https://yeahdesk.ru/. Требуется платная лицензия. Поддерживаемые платформы для работы: веб-приложение, Android, iOS. Список возможностей:

- шаблоны ответов;
- интеграция E-mail;
- вложения файлов;
- приоритеты заявок;
- статусы заявок;
- управление доступом;
- комментарии;
- отчёты и аналитика эффективности;
- отслеживание активности клиентов;
- управление проблемами.

Еадеск зафиксирует все обращения из любого канала, покажет новые и неотвеченные чаты. Поможет найти, отфильтровать и сортировать переписку «рис. 1.1»:

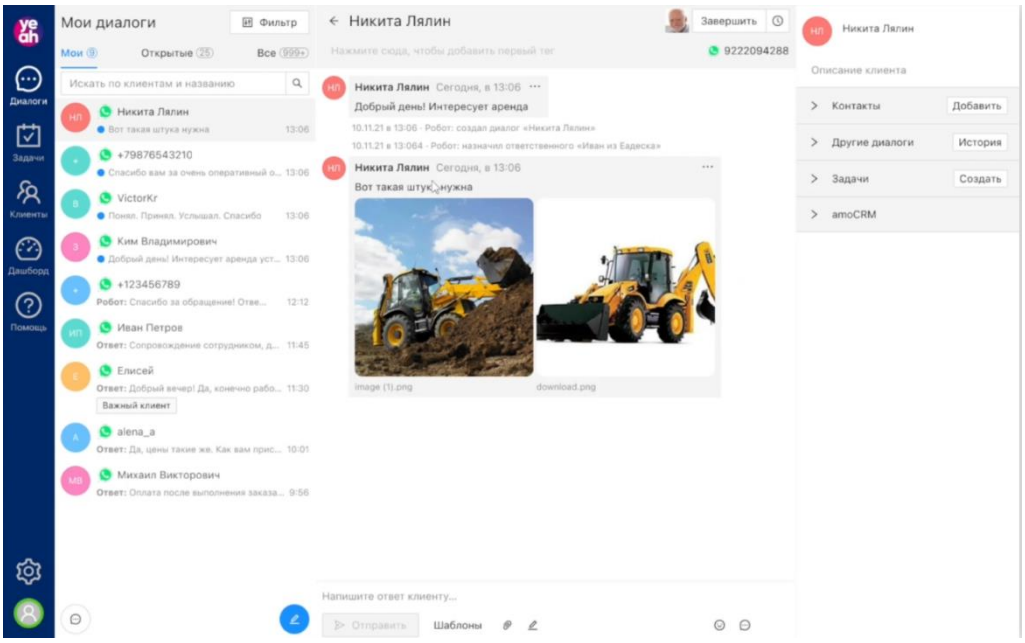


Рисунок 1.1 Главная страница в информационной системе Еадеск

Еадеск распределит все обращения между менеджерами, которые сейчас на связи, и отправит им уведомления о новом диалоге. Менеджеры быстро ответят клиентам с помощью шаблонов «Приложение А». Простые отчёты помогут отследить, какие менеджеры работают, а какие редко на связи и долго отвечают клиентам «Приложение Б». Фильтрация данных «Приложение В».

1.3 Анализ информационной системы Юздеск

Это единая система поддержки клиентов во всех каналах, где пишут вам и о вас. Выбор канала коммуникации, передача файлов, всплывающие уведомления, звуковые оповещения, база знаний - все в одном окне у вас на сайте. Официальный сайт https://usedesk.ru/. Требуется платная лицензия. Поддерживаемые платформы для работы: веб-приложение, Android, iOS. Список возможностей:

- приоритеты заявок;
- база знаний;
- статусы заявок;
- комментарии;
- портал самообслуживания;
- вложения файлов;
- теги / категории;
- опросы;
- отчёты и аналитика эффективности;

- уведомления;
- шаблоны ответов;
- интеграция E-mail;
- управление доступом;
- фильтры.

Автоматическое и ручное назначение - сообщение автоматически поступает ответственной группе. Сотрудник в любой момент может передать диалог вручную «Приложение Г».

Автоответы - Юздеск отправляет автоответы на стандартные вопросы клиентов. Операторы решают вопросы посложнее «Приложение Д».

Гибкие отчеты - Юздеск покажет продуктивность, время ответа, удовлетворенность клиентов, отчет по отделу или по сотруднику «рис. 1.2»:

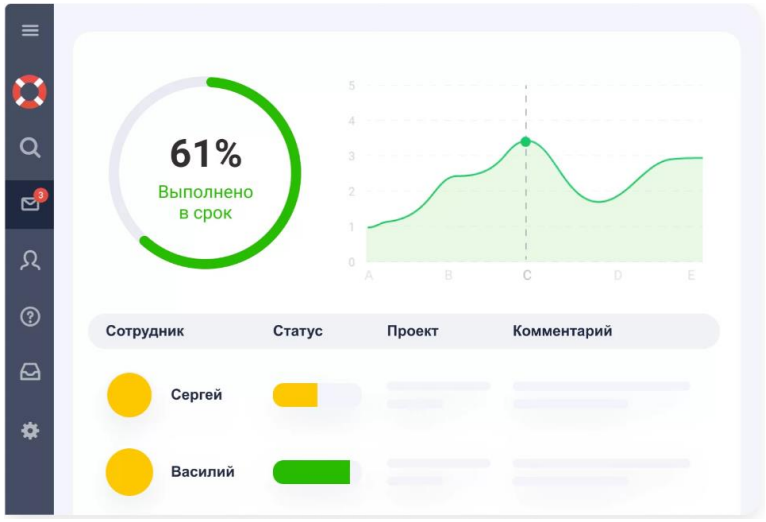


Рисунок 1.7 – Отчеты в информационной системе Юздеск

Контроль качества - клиенты оценивают каждый диалог, а руководитель отслеживает ошибки сотрудников «Приложение Е».

История переписки - диалог уже сохранился как заявка в истории клиента. Просто передайте вопрос ответственным, а после решения напишите клиенту на почту или в мессенджер «Приложение Ж».

База знаний - все для помощи клиентам в одном месте: скрипты, инструкции, регламенты разных подразделений. Статьи структурированы по разделам и категориям для еще более удобного поиска «Приложение З».

1.4 Анализ информационной системы Kayako

Это веб-сервис для обслуживания клиентов с опытом в сфере технической поддержки. Он способен вести поддержку по электронной почте и принимать заявки, через чат, звонки или центр самообслуживания, в нём же хранится вся история поддержка клиентов. Официальный сайт https://kayako.com/. Требуется платная лицензия. Поддерживаемые платформы для работы: веб-приложение, Android, iOS. Список возможностей:

- Поддержка приёма заявок с помощью электронной почты.
- Полнофункциональный чат для службы поддержки и веб-сайта.
- Мониторинг посетителей в реальном времени.
- Отслеживание и регистрация телефонных звонков.
- Центр самообслуживания.
- Расширенные рабочие процессы и правила.
- Пользовательские поля в заявках, чатах и карточках клиентов.
- Профили клиентов и организаций в CRM.
- Поддержка нескольких тарифов.
- Управление ответами на заявки и сроками разрешения.
- Определение бизнес-часов для точного отслеживания производительности.
- Автоматические правила эскалации.
- Более сотни готовых отчётов службы поддержки.
- Создание пользовательских отчетов.
- Расписание с автоотправкой на e-mail.
- Многоязычная поддержка клиентов.
- Полный контроль над внешним видом.
- REST API для интеграции с приложениями и серверными системами.

Обеспечивает удобный клиентский опыт, будучи всегда доступным, где бы клиенты ни задавали вопросы. Живой чат, социальные сети, электронная почта и служба поддержки на одной платформе «рис. 1.3»:

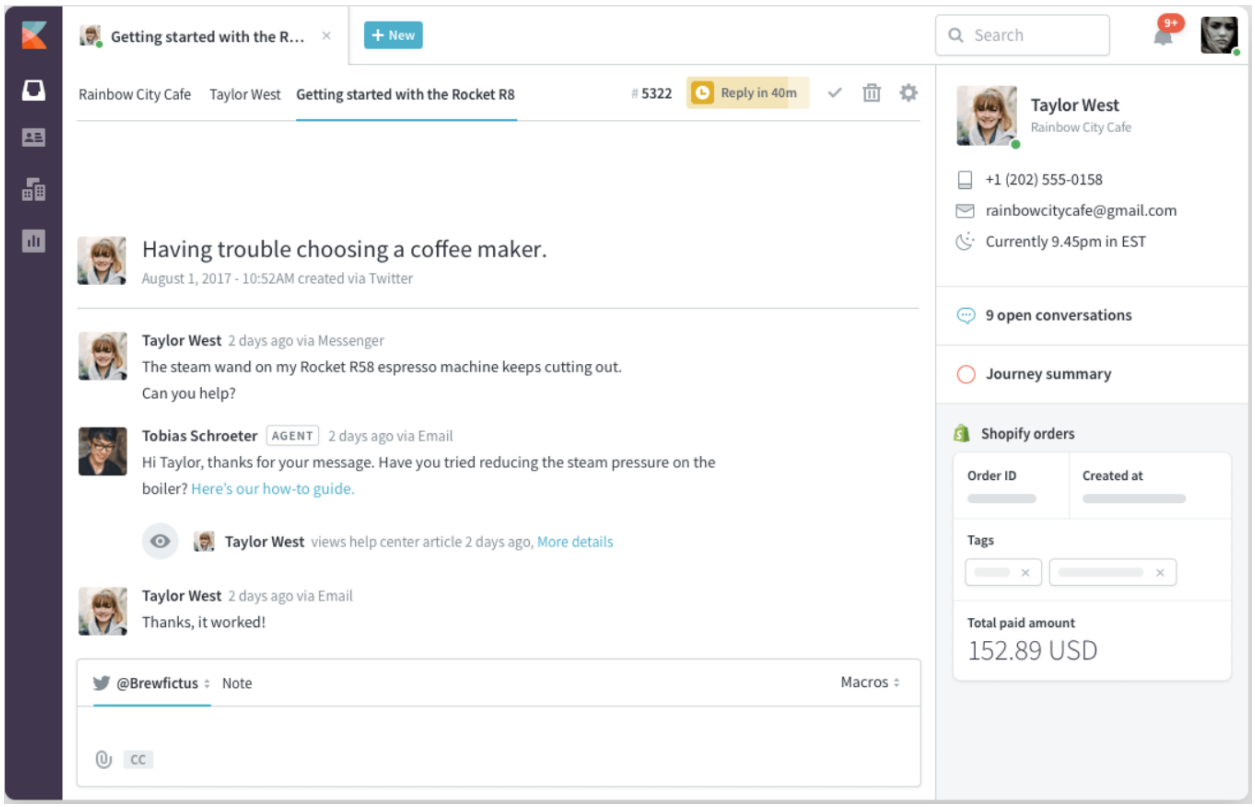


Рисунок 1.11 Главная страница в информационной системе Kayako

Позволяет визуализировать весь путь клиента в одном месте «Приложение И». Предоставляет удобный для поиска справочный центр для клиентов (статьи по самопомощи, видео, часто задаваемые вопросы и сообщество самообслуживания), позволяет отслеживать условия поиска, чтобы найти пробелы в своем портале самообслуживания «Приложение К».

Предоставляет отчеты для отслеживания время отклика на уровне обслуживания в чате, электронной почте и социальных сетях. Мгновенный доступ к ключевым показателям или создание настраиваемых информационных панелей для детализированных отчетов «Приложение Л».

1.5 Сравнение функциональных возможностей информационных систем

Сравнение функциональных возможностей проанализированных информационных систем представлено в таблице 1.1:

Таблица 1.1 Функциональные возможности информационных систем

	Еадеск	**Юздеск**	**Kayako**
База знаний	нет	да	да
Шаблоны ответов	да	да	да
Интеграция E-mail	да	да	да
Вложения файлов	да	да	нет
Приоритеты заявок	да	да	да
Статусы заявок	да	да	да
Управление доступом	да	да	да
Комментарии	да	да	да
Портал самообслуживания	нет	да	да
Отчёты и аналитика эффективности	да	да	да
Отслеживание активности клиентов	да	нет	нет
Управление проблемами	да	нет	нет

ГЛАВА 2 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ ПРИНЯТИЯ И ОБРАБОТКИ ЗАЯВОК

2.1 Требования к системе

Согласно проведенному анализу существующих систем в прошлой главе сформируем следующие функциональные возможности, которыми должна обладать проектируемая система:

- шаблоны ответов;
- вложения файлов;
- приоритеты заявок;
- статусы заявок;
- управление доступом;
- комментарии;
- портал самообслуживания;
- отчеты и аналитика эффективности.

Роли пользователей системы разделяются следующим образом:

1) Администратор - авторизованный пользователь с правом заполнения системных данных необходимых для функционирования системы и управления другими пользователями.

2) Клиент – авторизованный пользователь с правами на создание заявок по определенным категориям и дальнейшего отслеживания хода обработки созданных заявок.

3) Сотрудник – авторизованный пользователь с правом просмотра, фильтрации и обработки заявок. Под обработкой имеется ввиду изменение атрибутов заявки.

Описание операций

Для корректной работы система должна иметь как минимум одного пользователя с правами администратора, предполагается, что такой пользователь будет создан автоматически при первом запуске системы. Список операций приведен ниже в таблице (Таблица 2.1).

Таблица 2.1 Список операций

Название операции	**Исполнитель**	**Периодичность выполнения**	**Входные данные (документы)**	**Результат выполнения операции (выходные данные, документы)**
Заполнение системных данных	Администратор	Вначале и далее по мере необходимости	Внутренние нормативные документы	В случае успеха - новая строка в базе данных. В случае неудачи – сообщение об ошибке
Создание учетной записи сотрудника	Администратор	По усмотрению пользователя	Список сотрудников	В случае успеха - новая строка в базе данных. В случае неудачи – сообщение об ошибке
Блокировка / разблокировка пользователя	Администратор	По усмотрению пользователя	Нет	В случае успеха – обновленная строка в базе данных. В случае неудачи – сообщение об ошибке
Создание учетной записи клиента	Клиент	По усмотрению пользователя	Нет	В случае успеха - новая строка в базе данных. В случае неудачи – сообщение об ошибке
Создание заявки	Клиент	По усмотрению пользователя	Нет	В случае успеха - новая строка в базе данных. В случае неудачи – сообщение об ошибке
Просмотр списка заявок с фильтрацией и сортировкой	Клиент, Сотрудник	По усмотрению пользователя	Нет	Список заявок в табличном виде
Просмотр заявки	Клиент, Сотрудник	По усмотрению пользователя		Содержимое заявки
Обновление заявки клиентом	Клиент	По усмотрению пользователя	Нет	Обновленное содержимое заявки
Обновление заявки сотрудником	Сотрудник	По усмотрению пользователя	Нет	Обновленное содержимое заявки
Подписаться на	Клиент,	По усмотрению	Нет	В случае успеха -

Название операции	**Исполнитель**	**Периодичность выполнения**	**Входные данные (документы)**	**Результат выполнения операции (выходные данные, документы)**
отслеживание изменений заявки	Сотрудник	пользователя		новая строка в базе данных. В случае неудачи – сообщение об ошибке
Получение уведомлений об изменениях в отслеживаемых заявках	Клиент, Сотрудник	Автоматически	Нет	Сообщения об изменениях

Требования к входным данным по операции заполнения системных данных

На основании внутренних нормативных документов администратор заполняет системные данные, которые будут использоваться при создании и обработки заявок, к ним относятся:

1) Статусы заявок, например: «Новая», «В работе», «Закрыта», «Переоткрыта». В базу данных вносится следующая информация:

- название статуса, допустимая длина 100 символов, уникальное значение;
- признак того, что статус назначается новой заявки по умолчанию, только один статус может быть статусом по умолчанию;
- признак того, что статус является финальным;
- признак того, что статус является активным и может использоваться сотрудниками в пользовательском интерфейсе.

2) Правила смены статусов, используются, чтобы указать из какого статуса в какой можно перевести заявки, например в статус «Переоткрыта» можно перевести только из статуса «Закрыта». В базу данных вносится следующая информация:

- название правила, допустимая длина 100 символов, уникальное значение;
- признак того, что правило является активным и будет использоваться для валидации смены статуса;

- статус, из которого выполняется перевод, если пустое значение, то из любого;
- статус, в который выполняется перевод, если пустое значение, то в любой.

3) Очереди заявок, используются для указания ответственного, от которого требуется дальнейшее действие по заявки, например: «Клиент», «Отдел продаж», «Отдел доставки», «Отсутствует». В базу данных вносится следующая информация:

- название очереди, допустимая длина 100 символов, уникальное значение;
- признак того, что очередь назначается новой заявки по умолчанию, только одна очередь может быть очередью по умолчанию;
- признак того, что очередь является активной и может использоваться сотрудниками в пользовательском интерфейсе.

4) Категории заявок, например «Товар не был получен», «Записаться на обслуживание». Возможно создание двух уровневой иерархии категорий. В базу данных вносится следующая информация:

- название категории, допустимая длина 100 символов, уникальное значение;
- признак, что категория является корневой;
- признак того, что категория является активной и может использоваться сотрудниками и клиентами в пользовательском интерфейсе;
- родительская категория, пустое значение, если категория является корневой.

5) Приоритеты заявок, например: «Низкий», «Нормальный», «Высокий». В базу данных вносится следующая информация:

- название приоритета, допустимая длина 100 символов, уникальное значение;
- уровень приоритета, числовое значение, чем больше число, тем выше приоритет;
- признак того, что приоритет является активным и может использоваться сотрудниками в пользовательском интерфейсе.

6) Причины закрытия заявок, например: «Дубликат», «Выполнена». В базу данных вносится следующая информация:

- название причины закрытия, допустимая длина 100 символов, уникальное значение;
- признак того, что причина закрытия является активной и может использоваться сотрудниками и клиентами в пользовательском интерфейсе.

7) Шаблоны ответов на заявки, например «Заявка взята в работу, приблизительное время обработки - 2 часа». В базу данных вносится следующая информация:

- название шаблона, допустимая длина 100 символов, уникальное значение;
- текст шаблона, допустимая длина 2000 символов;
- тип шаблона, допустимая длина 100 символов, используется для поиска подходящих шаблонов в пользовательском интерфейсе;
- признак того, что шаблон является активным и может использоваться сотрудниками в пользовательском интерфейсе.

Требования к входным данным по операции создания учетной записи сотрудника

На основании списка сотрудников администратор вносит следующую информацию о пользователях в базу данных:

- электронный адрес пользователя, допустимая длина 250 символов, уникальное значение;
- имя пользователя, допустимая длина 50 символов;
- фамилия пользователя, допустимая длина 50 символов;
- разрешения, которые определяют, что пользователь обладает правами доступа сотрудника;
- признак того, что учетная запись является активной и может использоваться в пользовательском интерфейсе.

Требования к входным данным по операции блокировки или разблокировки пользователя

Администратор может заблокировать/разблокировать учетную запись пользователя, для этого он должен будет передать следующие данные:

- идентификатор учетной записи;
- признак блокировки или разблокировки.

Требования к входным данным по операции создания учетной записи клиента

Во время регистрации пользователя в базу данных вносится следующая информация:

- электронный адрес пользователя, допустимая длина 250 символов, уникальное значение;
- имя пользователя, допустимая длина 50 символов;
- фамилия пользователя, допустимая длина 50 символов.

Требования к входным данным по операции создания заявки

Во время создания заявки в базу данных вносится следующая информация:

- название заявки, допустимая длина 100 символов;
- категория заявки, выбирается из списка (системные данные);
- комментарий, допустимая длина 2000 символов;
- прикрепленные файлы, (опциональная возможность прикрепить файлы к комментарию);
- статус заявки по умолчанию (системные данные);
- очередь заявки по умолчанию (системные данные).

Требования к входным данным по операции обновления заявки клиентом

Клиент может закрыть заявку. Клиент может добавить следующие данные:

- комментарий, допустимая длина 2000 символов;
- прикрепленные файлы, (опциональная возможность прикрепить файлы к комментарию);

Требования к входным данным по операции обновления заявки сотрудником

Сотрудник может обновить следующие данные:

- категория заявки, выбирается из списка (системные данные);
- статус заявки, выбирается из списка (системные данные) в соответствии с определенными администратором правилами;
- очередь заявки, выбирается из списка (системные данные);
- приоритет заявки, выбирается из списка (системные данные);
- причину закрытия при закрытии, выбирается из списка (системные данные);
- назначить заявку на себя или снять с себя назначение;
- свои комментарии (есть возможность использовать готовые шаблоны ответов из системных данных) и свои прикрепленные файлы.

Сотрудник может добавить следующие данные:

- комментарий, допустимая длина 2000 символов (комментарий может быть внутренним и тогда он будет виден только другим сотрудникам, есть возможность использовать готовые шаблоны ответов из системных данных);
- прикрепленные файлы, (опциональная возможность прикрепить файлы к комментарию).

Требования к выходным данным по операции просмотра списка заявок с фильтрацией и сортировкой

Клиент может просматривать только свои заявки, сотрудник может просматривать все заявки. Результат выводиться в табличном виде с пагинацией.

Результат для клиента включает следующие данные:

- название заявки;
- категория заявки;
- признак, что заявка закрыта (доступна фильтрация);
- дата и время создания заявки (доступна сортировка и фильтрация);
- дата и время обновления заявки (доступна сортировка и фильтрация);
- статус заявки (доступна фильтрация).

Результат для сотрудника включает следующие данные:

- название заявки;
- категория заявки (доступна фильтрация);
- признак, что заявка закрыта (доступна фильтрация);
- дата и время создания заявки (доступна сортировка и фильтрация);
- дата и время обновления заявки (доступна сортировка и фильтрация);
- статус заявки (доступна фильтрация);
- очередь заявки (доступна фильтрация);
- приоритет заявки (доступна сортировка и фильтрация);
- пользователь, который создал заявку (доступна фильтрация);
- пользователь, на которого назначена заявка (доступна фильтрация);

Требования к выходным данным по операции просмотра заявки

Клиент и сотрудник могут просматривать детальное содержимое заявки, содержимое должно включать следующие данные:

- название заявки;
- категория заявки;
- признак, что заявка закрыта;
- дата и время создания заявки;
- дата и время обновления заявки;
- статус заявки;
- очередь заявки (отображается только для сотрудника);
- приоритет заявки (отображается только для сотрудника);
- пользователь, который создал заявку (отображается только сотруднику);
- пользователь, на которого назначена заявка (отображается только сотруднику);
- список комментариев с прикрепленными файлами, отсортированы по дате создания (сперва свежие), с пагинацией, клиент не видит внутренние комментарии;
- история изменений статуса (отображается только для сотрудника);
- история изменений приоритета (отображается только для сотрудника);

– история изменений очереди (отображается только для сотрудника);
– история изменений категории (отображается только для сотрудника).

Требования к входным данным по операции подписаться на отслеживание изменений заявки

Клиент и сотрудник могут подписаться на отслеживание изменений в заявке, для этого пользователь должен открыть детальное содержимое заявки и нажать на переключатель отслеживания изменений, при этом в базу данных создается строка в специальной таблице со следующими данными:

– идентификатор заявки;
– идентификатор пользователя.

В случае отписки от отслеживания изменений, это строка удаляется из базы.

Требования к выходным данным по операции получения уведомлений об изменениях заявки

Пользователи используют отдельную страницу для просмотра уведомлений, страница отображает данные в табличном виде с пагинацией, данные сортируются по дате создания (вначале свежие), страница содержит следующие данные:

– идентификатор заявки;
– тип изменения, например: «Изменение статуса», «Изменение приоритета»;
– предыдущее значение до изменения;
– новое значение;
– дата и время события;
– пользователь, который сделал изменение (отображается только сотруднику).

Описание данных для проектирования БД

Таблица 2.2 Атрибуты статуса заявки

Имя	**Описание**	**Тип данных**	**Значение по умолчанию**	**Формат ввода (маска)**	**Ограничение на значения атрибута**
Id	Идентификатор статуса	Числовой	-	-	Обязательный, авто инкрементируемое значение

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Name	Название статуса	Строковый	-	-	Обязательный, максимальная длина 100 символов, уникальное значение
IsDefault	Статус по умолчанию	Логический	-	-	Обязательный
IsCompletion	Финальный статус	Логический	-	-	Обязательный
IsActive	Активный статус	Логический	-	-	Обязательный
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
DateModified	Дата и время обновления	Дата и время	Текущая дата и время	-	Обязательный

Таблица 2.3 Атрибуты правил смены статусов заявки

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор правила	Числовой	-	-	Обязательный, авто инкрементируемое значение
Name	Название правила	Строковый	-	-	Обязательный, максимальная длина 100 символов, уникальное значение
IsActive	Активный правило	Логический	-	-	Обязательный
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
DateModified	Дата и время обновления	Дата и время	Текущая дата и время	-	Обязательный
FromTicketStatusId	Идентификатор статус, из которого можно	Числовой	NULL	-	Необязательный

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
	перевести				
ToTicketStatusId	Идентификатор статус, в которой можно перевести	Числовой	NULL	-	Необязательный

Таблица 2.4 Атрибуты категорий заявки

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор категории	Числовой	-	-	Обязательный, авто инкрементируемое значение
Name	Название категории	Строковый	-	-	Обязательный, максимальная длина 100 символов, уникальное значение
IsActive	Активная категория	Логический	-	-	Обязательный
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
DateModified	Дата и время обновления	Дата и время	Текущая дата и время	-	Обязательный
IsRoot	Является ли корневой категорией	Логический	-	-	Обязательный
ParentCategoryId	Идентификатор родительской категории	Числовой	NULL	-	Необязательный

Таблица 2.5 Атрибуты приоритетов заявки

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор приоритета	Числовой	-	-	Обязательный, авто инкрементируемое значение
Name	Название приоритета	Строковый	-	-	Обязательный, максимальная длина 100 символов, уникальное значение
Level	Уровень приоритета	Числовой	0	-	Обязательный
IsActive	Активная	Логический	false	-	Обязательный

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
	категория				
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
DateModified	Дата и время обновления	Дата и время	Текущая дата и время	-	Обязательный

Таблица 2.6 Атрибуты очередей заявки

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор очереди	Числовой	-	-	Обязательный, авто инкрементируемое значение
Name	Название очереди	Строковый	-	-	Обязательный, максимальная длина 100 символов, уникальное значение
IsDefault	Очередь по умолчанию	Логический	False	-	Обязательный
IsActive	Активный приоритет	Логический	false	-	Обязательный
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
DateModified	Дата и время обновления	Дата и время	Текущая дата и время	-	Обязательный

Таблица 2.7 Атрибуты причин закрытия заявки

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор причины закрытия	Числовой	-	-	Обязательный, авто инкрементируемое значение
Name	Название причины закрытия	Строковый	-	-	Обязательный, максимальная длина 100 символов, уникальное

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
					значение
IsActive	Активная причина закрытия	Логический	false	-	Обязательный
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
DateModified	Дата и время обновления	Дата и время	Текущая дата и время	-	Обязательный

Таблица 2.8 Атрибуты шаблонов ответов

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор шаблона	Числовой	-	-	Обязательный, авто инкрементируемое значение
Name	Название шаблона	Строковый	-	-	Обязательный, максимальная длина 100 символов, уникальное значение
Text	Текст шаблона	Строковый	-	-	Обязательный, максимальная длина 2000 символов
Type	Тип шаблона	Строковый	-	-	Обязательный, максимальная длина 100 символов
IsActive	Активный шаблон	Логический	false	-	Обязательный
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
DateModified	Дата и время обновления	Дата и время	Текущая дата и время	-	Обязательный

Таблица 2.9 Атрибуты пользователей

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор пользователя	Числовой	-	-	Обязательный, авто инкрементируемое значение
Email	Электронный адрес	Строковый	-	-	Обязательный, максимальная длина 250

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
					символов, уникальное значение
FirstName	Имя пользователя	Строковый	-	-	Обязательный, максимальная длина 50 символов
LastName	Фамилия пользователя	Строковый	-	-	Обязательный, максимальная длина 50 символов
Permissions	Права доступа	Числовой	0	-	Обязательный
IsLocked	Заблокирована ли учетная запись	Логический	false	-	Обязательный
IsActive	Активный пользователь	Логический	false	-	Обязательный
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
DateModified	Дата и время обновления	Дата и время	Текущая дата и время	-	Обязательный

Таблица 2.10 Атрибуты заявки

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор заявки	Числовой	-	-	Обязательный, авто инкрементируемое значение
Name	Название заявки	Строковый	-	-	Обязательный, максимальная длина 100 символов
IsClosed	Закрыта ли заявка	Логический	false	-	Обязательный
ReporterId	Идентификатор пользователя, который создал заявку	Числовой	-	-	Обязательный
AssigneeId	Идентификатор пользователя, на которого назначена заявка	Числовой	NULL	-	Необязательный
TicketPriorityId	Идентификатор приоритета	Числовой	NULL	-	Необязательный
TicketQueueId	Очередь	Числовой	-	-	Обязательный
TicketResolutionId	Идентификатор	Числовой	NULL	-	Необязательный

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
	причины закрытия				
TicketStatusId	Идентификатор статуса	Числовой	-	-	Обязательный
TicketCategoryId	Идентификатор категории	Числовой	-	-	Обязательный
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
DateModified	Дата и время обновления	Дата и время	Текущая дата и время	-	Обязательный
Comment	Комментарий	Строковый	-	-	Обязательный, максимальная длина 2000 символов
IsInternal	Внутренний ли комментарий	Логический	false	-	Обязательный
FileName	Авто сгенерированное имя прикрепленного файла, которое будет использоваться для хранения	Строковый	NULL	-	Необязательный, максимальная длина 50 символов, уникальное значение
Extension	Формат прикрепленного файла	Строковый	NULL	-	Необязательный, максимальная длина 5 символов
OriginalFileName	Оригинальное имя файла для отображения в пользовательском интерфейсе	Строковый	NULL	-	Необязательный

Таблица 2.11 Атрибуты отслеживания изменений

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор отслеживания	Числовой	-	-	Обязательный, авто инкрементируемое значение
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
TicketId	Идентификатор заявки	Числовой	-	-	Обязательный

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
UserId	Идентификатор пользователя	Числовой	-	-	Обязательный

Таблица 2.12 Атрибуты истории смены статусов

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор	Числовой	-	-	Обязательный, авто инкрементируемое значение
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
TicketId	Идентификатор заявки	Числовой	-	-	Обязательный
UserId	Идентификатор пользователя	Числовой	-	-	Обязательный
FromTicketStatusId	Идентификатор статуса, из которого меняли	Числовой	-	-	Обязательный
ToTicketStatusId	Идентификатор статуса, в которой меняли	Числовой	-	-	Обязательный

Таблица 2.13 Атрибуты истории смены категорий

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор	Числовой	-	-	Обязательный, авто инкрементируемое значение
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
TicketId	Идентификатор заявки	Числовой	-	-	Обязательный
UserId	Идентификатор пользователя	Числовой	-	-	Обязательный
FromTicketCategoryId	Идентификатор категории, из которой меняли	Числовой	-	-	Обязательный
ToTicketCategoryId	Идентификатор	Числовой	-	-	Обязательный

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
	категории, в которую меняли				

Таблица 2.14 Атрибуты истории смены приоритетов

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор	Числовой	-	-	Обязательный, авто инкрементируемое значение
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
TicketId	Идентификатор заявки	Числовой	-	-	Обязательный
UserId	Идентификатор пользователя	Числовой	-	-	Обязательный
FromTicketPriorityId	Идентификатор приоритета, из которого меняли	Числовой	NULL	-	Необязательный
ToTicketPriorityId	Идентификатор приоритета, в который меняли	Числовой	-	-	Обязательный

Таблица 2.15 Атрибуты истории смены очередей

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор	Числовой	-	-	Обязательный, авто инкрементируемое значение
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
TicketId	Идентификатор заявки	Числовой	-	-	Обязательный
UserId	Идентификатор пользователя	Числовой	-	-	Обязательный
FromTicketQueueId	Идентификатор очереди, из которой меняли	Числовой	-	-	Обязательный
ToTicketQueueId	Идентификатор	Числовой	-	-	Обязательный

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
	очереди, в которую меняли				

Таблица 2.16 Атрибуты общей истории изменений заявок

Имя	Описание	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута
Id	Идентификатор	Числовой	-	-	Обязательный, авто инкрементируемое значение
IsInternal	Является ли внутренним событием, чтобы отображаться только сотрудникам	Логический	false	-	Обязательный
Type	Тип изменения, заполняется системой автоматически	Строковый	-	-	Обязательный, максимальная длина 100 символов
OldValue	Предыдущее значение	Строковый	NULL	-	Необязательный
NewValue	Новое значение	Строковый	NULL	-	Необязательный
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
TicketId	Идентификатор заявки	Числовой	-	-	Обязательный
UserId	Идентификатор пользователя	Числовой	-	-	Обязательный

При проектировании БД следует учесть следующие функциональные зависимости между атрибутами сущностей:

1) По идентификатору статуса заявки можно однозначно определить:

- название статуса;
- является ли он статусом по умолчанию;
- является ли он финальным;
- является ли он активным;
- дату и время создания и редактирования;

- список заявок с таким статусом;
- список заявок, которые когда-либо имели такой статус;
- список пользователей, которые использовали такой статус для обновления заявки.

2) По идентификатору очереди можно однозначно определить:

- название очереди;
- является ли она очередью по умолчанию;
- является ли она активной;
- дату и время создания и редактирования;
- список заявок с такой очередью;
- список заявок, которые когда-либо находились в такой очереди;
- список пользователей, которые использовали такую очередь для обновления заявки.

3) По идентификатору категории можно однозначно определить:

- название категории;
- является ли она корневой категорией;
- является ли она активной;
- дату и время создания и редактирования;
- её родительскую категорию (при наличии);
- список заявок с такой категорией;
- список заявок, которые когда-либо имели такую категорию;
- список пользователей, которые использовали такую категорию для обновления заявки.

4) По идентификатору приоритета можно однозначно определить:

- название приоритета;
- его уровень;
- является ли он активным;
- дату и время создания и редактирования;
- список заявок с таким приоритетом;
- список заявок, которые когда-либо имели такой приоритет;

– список пользователей, которые использовали такой приоритет для обновления заявки.

5) По идентификатору причины закрытия можно однозначно определить:

– название причины закрытия;
– является ли она активной;
– дату и время создания и редактирования;
– список заявок с такой причиной закрытия.

6) По идентификатору правила смены статуса можно однозначно определить:

– название правила;
– является ли оно активным;
– дату и время создания и редактирования;
– статус, из которого можно выполнить изменение;
– статус, в который можно выполнить изменение.

7) По идентификатору шаблона ответа можно однозначно определить:

– название шаблона;
– текст;
– тип;
– является ли он активным;
– дату и время создания и редактирования.

8) По идентификатору пользователя можно однозначно определить:

– электронный адрес;
– имя;
– фамилию;
– права доступа;
– является ли учетная запись активной;
– является ли учетная запись заблокированной;
– дату и время создания и редактирования;
– список созданных заявок;
– список назначенных заявок;
– список заявок, по которым отслеживаются изменения;

– список изменений, которые совершал пользователь в заявках.

9) По идентификатору заявки можно однозначно определить:

– название заявки;
– является ли она закрытой;
– дату и время создания и редактирования;
– пользователя, который создал заявку;
– пользователя, на которого назначения заявка;
– приоритет заявки;
– очередь заявки;
– статус заявки;
– причину закрытия заявки;
– категорию заявки;
– комментарии заявки;
– прикрепленные файлы;
– историю изменений статуса/приоритета/категории/очереди/т.д.;

2.2 Проектирование базы данных

Таблица статусов заявок

Таблица будет называться «TicketStatuses». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Таблица правил смены статусов

Таблица будет называться «TicketStatusRules». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Таблица категорий заявок

Таблица будет называться «TicketCategories». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id». Таблица будет иметь связь один ко многим на саму себя с помощью атрибута «ParentCategoryId».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Таблица приоритетов заявок

Таблица будет называться «TicketPriorities». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Таблица очередей заявок

Таблица будет называться «TicketQueues». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Таблица причин закрытия

Таблица будет называться «TicketResolutions». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Таблица шаблонов ответов

Таблица будет называться «TicketResponseTemplates». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица не находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица приведения таблицы к 3НФ требуется избавиться от избыточности в виде атрибута «Type», который приводит к аномалиям обновления (при обновлении типа потребуется обновлять все строки, в которых есть этот тип) и удаления (если только одна строка имеет определенный тип, то при удалении этой строки, мы потеряем и тип). Для устранения этой избыточности выполним декомпозицию, тем самым вынесем тип в отдельную таблицу «TicketResponseTemplateTypes» со следующими атрибутами «см. таблицу 2.17»:

Таблица 2.17 Атрибуты типов шаблонов ответов

Имя	**Описание**	**Тип данных**	**Значение по умолчанию**	**Формат ввода (маска)**	**Ограничение на значения атрибута**
Id	Идентификатор	Числовой	-	-	Обязательный, авто инкрементируемое значение
Name	Название типа	Строковый	-	-	Обязательный, максимальная длина 100 символов, уникальное значение
IsActive	Активный тип	Логический	false	-	Обязательный
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
DateModified	Дата и время обновления	Дата и время	Текущая дата и время	-	Обязательный

В таблицу «TicketResponseTemplates» поместим внешний ключ для связи с таблицей «TicketResponseTemplateTypes» (тип связи один ко многим).

Таблица пользователей

Таблица будет называться «Users». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Атрибут «Permissions» определяется на уровне приложения, поэтому не требуется выносить его в отдельную таблицу.

Таблица заявки

Таблица будет называться «Tickets». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id».

Таблица имеет следующие атрибуты, которые являются внешними ключами:

- ReporterId: связь один ко многим с таблицей «Users»;
- AssigneeId: связь один ко многим с таблицей «Users»;
- TicketPriorityId: связь один ко многим с таблицей «TicketPriorities»;
- TicketQueueId: связь один ко многим с таблицей «TicketQueues»;
- TicketResolutionId: связь один ко многим с таблицей «TicketResolutions»;
- TicketStatusId: связь один ко многим с таблицей «TicketStatuses»;
- TicketCategoryId: связь один ко многим с таблицей «TicketCategories».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Для приведения во 2НФ требуется избавиться от избыточности, которая возникает из-за того, что заявка может содержать несколько комментариев и один комментарий может содержать несколько прикрепленных файлов:

Таблица 2.18 Избыточности заявок

Id	**Name**	**…**	**Comment**	**…**	**FileName**	**…**
1	Возврата билета	…	Прошу вернуть деньги за неиспользуемый билет	…	File_1	…
1	Возврата билета	…	Прошу вернуть деньги за неиспользуемый билет	…	File_2	…
1	Возврата билета	…	Спасибо за обращение, заявка будет обработана в течении 30 минут	…	NULL	…
…	…	…	…	…	…	…

Для устранения данной избыточности выполним декомпозицию, в результате которой получим две дополнительные таблицы «TicketComments» и «TicketCommentAttachments».

Таблица «TicketComments» содержит следующие атрибуты:

Таблица 2.19 Атрибуты комментариев

Имя	**Описание**	**Тип данных**	**Значение по умолчанию**	**Формат ввода (маска)**	**Ограничение на значения атрибута**
Id	Идентификатор	Числовой	-	-	Обязательный, авто инкрементируемое значение
Text	Комментарий	Строковый	-	-	Обязательный, максимальная длина 2000 символов
IsInternal	Является ли комментарий внутренним	Логический	false	-	Обязательный
IsActive	Активный комментарий	Логический	false	-	Обязательный
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
DateModified	Дата и время обновления	Дата и время	Текущая дата и время	-	Обязательный
TicketId	Идентификатор заявки	Числовой	-	-	Обязательный
CreatorId	Идентификатор пользователя	Числовой	-	-	Обязательный

Таблица «TicketComments» содержит атрибут «TicketId», который является внешним ключом с таблицей «Tickets» со связью один ко многим, и атрибут «CreatorId», который является внешним ключом с таблицей «Users» со связью один ко многим. В качестве первичного ключа используется атрибут «Id».

Таблица «TicketCommentAttachments» содержит следующие атрибуты:

Таблица 2.20 Атрибуты прикрепленных файлов

Имя	**Описание**	**Тип данных**	**Значение по умолчанию**	**Формат ввода (маска)**	**Ограничение на значения атрибута**
Id	Идентификатор	Числовой	-	-	Обязательный, авто инкрементируемое значение
FileName	Генерируемое имя файла	Строковый	-	-	Обязательный, максимальная длина 50 символов, уникальное значение
Extension	Формат файла	Строковый	-	-	Обязательный, максимальная длина 5 символов
OriginalFileName	Оригинальное имя файла	Строковый	-	-	Обязательный
IsActive	Активный файл (отображается в пользовательском интерфейсе)	Логический	false	-	Обязательный
DateCreated	Дата и время создания	Дата и время	Текущая дата и время	-	Обязательный
DateModified	Дата и время обновления	Дата и время	Текущая дата и время	-	Обязательный

Таблица «TicketCommentAttachments» содержит атрибут «TicketCommentId», который является внешним ключом с таблицей «TicketComments» со связью один ко многим. В качестве первичного ключа используется атрибут «Id».

Из таблицы «Tickets» удаляем атрибуты, которые переместились в отдельные таблицы. Таким образом таблицы находится во 2НФ, так как они находится в 1НФ и в таблицах есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблицы уже находится в 3НФ, так как они находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Таблица отслеживания изменений

Таблица будет называться «TicketObservations». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id». Таблица будет иметь связь один ко многим с таблицей «Tickets» с помощью атрибута «TicketId» и связь один ко многим с таблицей «Users» с помощью атрибута «UserId».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Таблица истории смены статусов

Таблица будет называться «TicketStatusHistories». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id». Таблица будет иметь связь один ко многим со следующими таблицами:

- таблица «Tickets» с помощью атрибута «TicketId»;
- таблица «Users» с помощью атрибута «UserId»;
- таблица «TicketStatuses» с помощью атрибута «FromTicketStatusId»;
- таблица «TicketStatuses» с помощью атрибута «ToTicketStatusId».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Таблица истории смены категорий

Таблица будет называться «TicketCategoryHistories». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id». Таблица будет иметь связь один ко многим со следующими таблицами:

- таблица «Tickets» с помощью атрибута «TicketId»;
- таблица «Users» с помощью атрибута «UserId»;
- таблица «TicketCategories» с помощью атрибута «FromTicketCategoryId»;
- таблица «TicketCategories» с помощью атрибута «ToTicketCategoryId».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Таблица истории смены приоритетов

Таблица будет называться «TicketPriorityHistories». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id». Таблица будет иметь связь один ко многим со следующими таблицами:

- таблица «Tickets» с помощью атрибута «TicketId»;
- таблица «Users» с помощью атрибута «UserId»;
- таблица «TicketPriorities» с помощью атрибута «FromTicketPriorityId»;
- таблица «TicketPriorities» с помощью атрибута «ToTicketPriorityId».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Таблица истории смены очередей

Таблица будет называться «TicketQueueHistories». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id». Таблица будет иметь связь один ко многим со следующими таблицами:

- таблица «Tickets» с помощью атрибута «TicketId»;
- таблица «Users» с помощью атрибута «UserId»;
- таблица «TicketQueues» с помощью атрибута «FromTicketQueueId»;
- таблица «TicketQueues» с помощью атрибута «ToTicketQueueId».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Таблица истории общих изменений заявок

Таблица будет называться «TicketHistories». В соответствии с функциональными зависимостями в качестве первичного ключа будет использоваться атрибут «Id». Таблица будет иметь связь один ко многим со следующими таблицами:

- таблица «Tickets» с помощью атрибута «TicketId»;
- таблица «Users» с помощью атрибута «UserId».

Таблица уже находится в 1НФ, так как все атрибуты содержат атомарные значения и в таблице не будет повторяющихся строк.

Таблица уже находится во 2НФ, так как она находится в 1НФ и в таблице есть первичный ключ (атрибут Id) и все остальные атрибуты зависят от первичного ключа.

Таблица уже находится в 3НФ, так как она находится в 2НФ и все атрибуты зависят только от первичного ключа и не зависят от других атрибутов.

Схема базы данных

Схема спроектированной базы данных представлены на рисунке 2.1:

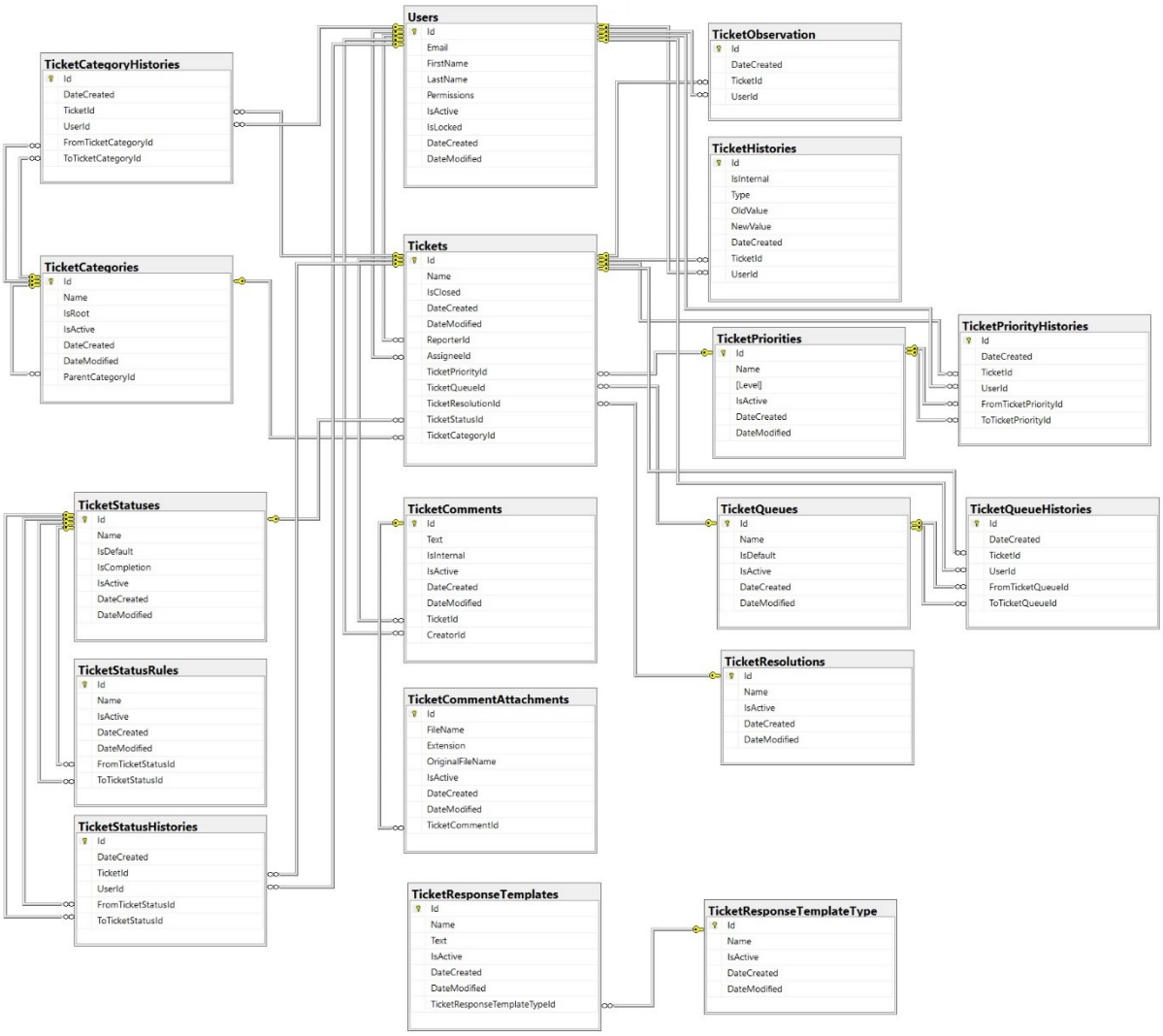


Рисунок 2.1 Схема базы данных

2.3 Проектирование приложения

Приложение будет представлять из себя веб-сайт на основе SPA[1] архитектуры:

– для серверной части будет использоваться язык программирования C# и веб-фреймворк Asp.Net Core. На данный момент язык программирования C# имеет текущую версию 10, веб-фреймворк Asp.Net Core имеет текущую версию 6;

[1] SPA – single page application (одностраничное приложение)

- для работы с базой данных будет использоваться ORM[2] библиотека EntityFramework Core, текущая версия 6.0.5, и ORM библиотека Dapper, текущая версия 2.0.123;
- для клиентской части будет использоваться язык программирования TypeScript, который имеет текущую версию 4.6.3;
- для рендеринга страниц на клиентской стороне будет использоваться фреймворк React.js, который имеет текущую версию 18.1;
- для хранения состояния на клиентской стороне будет использоваться библиотека MobX, текущая версия 6.6;
- для сборки клиентских файлов кода и стилей будет использоваться библиотека Webpack, текущая версия 5.72.1;
- клиентская и серверная сторона будут взаимодействовать по протоколу HTTP2 с использованием шифрования;
- для стилизации клиентской части будет использоваться Material Design;
- система аутентификации и хранение паролей будет делегирована внешнему сервису Google Firebase;
- в качестве сервера базы данных будет использоваться Microsoft SQL Server.

[2] ORM - Object-Relational Mapping (объектно-реляционное отображение)

ГЛАВА 3 РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ ДЛЯ ПРИНЯТИЯ И ОБРАБОТКИ ЗАЯВОК КЛИЕНТОВ

3.1 Разработка базы данных

Для реализации базы данных используется среда разработки Visual Studio 2022. Откроем Visual Studio 2022, и создадим новое решение с именем «mvp.tickets», в котором будет проект с именем «mvp.tickets.web» на основе шаблона Asp.Net Core Web API «рис. 3.1»:

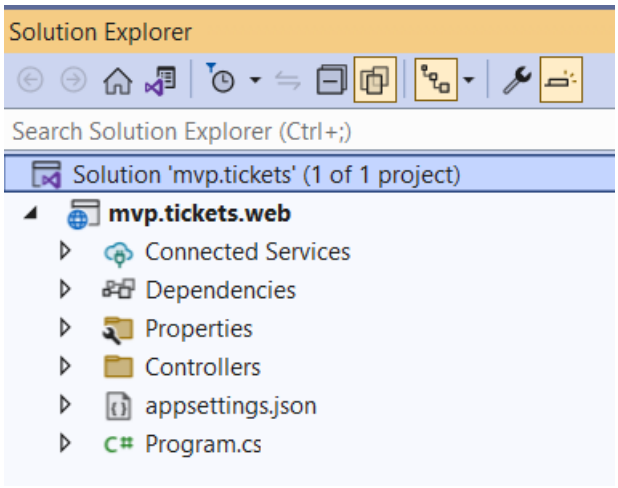


Рисунок 3.1 Обозреватель решений в Visual Studio

Для реализации базы данных используем программный подход Code First, когда сперва создаются классы будущих таблиц на языке C# и потом с помощью миграции на основе этих классов создается сама база данных и таблицы.

Добавим в решение через Visual Studio новый проект «mpv.tickets.data» (тип проекта «Class Library»), он будет содержать классы и логику работы с базой данных «рис. 3.2»:

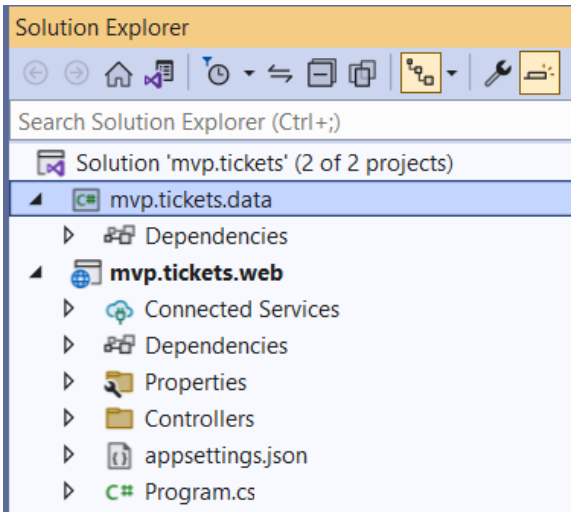


Рисунок 3.2 Проект «mvp.tickets.data»

Откроем менеджер Nuget пакетов и установим в проект «mvp.tickets.data» следующие пакеты «рис. 3.3»:

– Microsoft.EntityFrameworkCore
– Microsoft.EntityFrameworkCore.SqlServer
– Microsoft.EntityFrameworkCore.Design
– Microsoft.EntityFrameworkCore.Tools
– Dapper

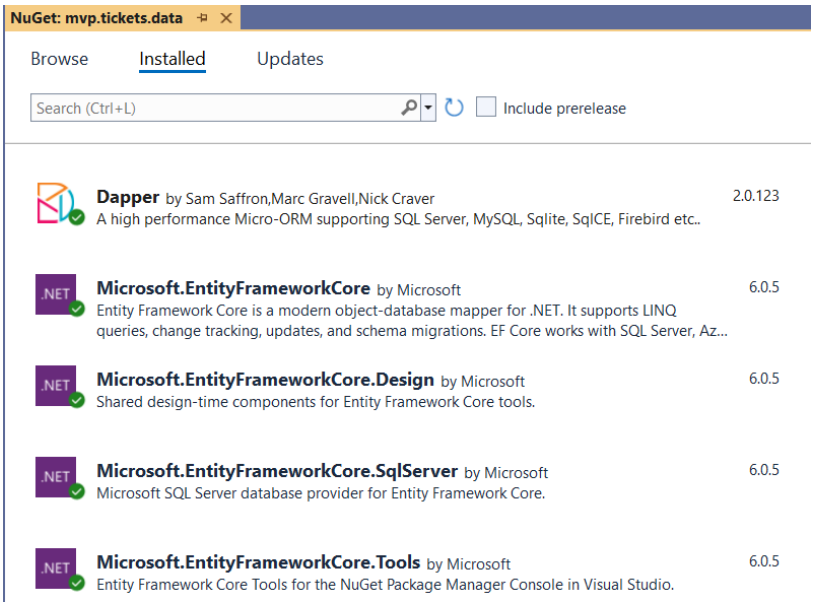


Рисунок 3.3 Установленные пакеты в проекте «mvp.tickets.data»

Библиотека EntityFramework будет использоваться для описания классов, для создания миграций и для простых SQL запросов. Библиотека Dapper будет использоваться для вызова SQL процедур, содержащих сложные запросы.

Создадим C# классы, соответствующие таблицам в базе данных «рис. 3.4»:

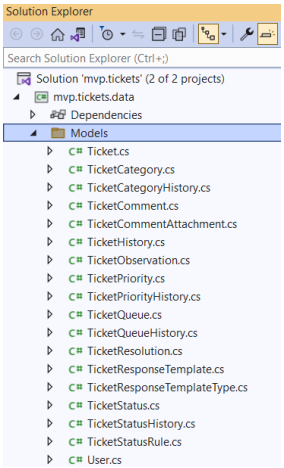


Рисунок 3.4 Классы, описывающие таблицы

В этом же проекте создадим класс с именем «ApplicationDbContext», это будет основной класс для работы с библиотекой EntityFramework, который будет выступать в роли проекции базы данных. Этот класс должен наследоваться от класса «DbContext» из библиотеки EntityFramework и содержать обобщенные свойства с типом «DbSet», которые обобщены типами классов наших моделей «рис. 3.5»:

```
using Microsoft.EntityFrameworkCore;
using mvp.tickets.data.Models;

namespace mvp.tickets.data
{
    6 references
    public class ApplicationDbContext : DbContext
    {
        1 reference
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }

        0 references
        public DbSet<Ticket> Tickets { get; set; }
        0 references
        public DbSet<TicketHistory> TicketHistories { get; set; }
        0 references
        public DbSet<TicketCommentAttachment> TicketCommentAttachments { get; set; }
```

Рисунок 3.5 Фрагмент содержимого класса «ApplicationDbContext»

Для создания миграции откроем консоль менеджера пакетов в Visual Studio и выполним команду «Add-Migration Init», в качестве стартового проекта и местом создания миграции должен быть выбран «mvp.tickets.data», после успешного выполнения команды в проекте «mvp.tickets.data» будет создана папка «Migrations», в которой будут классы миграции с именем «Init» и класс «ApplicationDbContextModelSnapshot», который описывает текущее состояние схемы.

Откроем конфигурационный файл «appsettings.json» в проекте «mvp.tickets.web» и добавим параметр со строкой подключения к базе данных «рис. 3.6»:

appsettings.json

Schema: https://json.schemastore.org/appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost;Database=mvp.tickets;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
```

Рисунок 3.6 Строка подключения к базе данных

Откроем файл «Program.cs» в проекте «mvp.ticket.web», чтобы зарегистрировать класс контекста и добавить логику выполнения миграций, которая будет выполняться при запуске этого проекта «рис. 3.7»:

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

...

using (var scope = app.Services.CreateScope())
{
    var dbContext = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
    dbContext.Database.Migrate();
}
```

Рисунок 3.7 Регистрация контекста и выполнение миграций

Запустим проект «mvp.tickets.web», что приведет к созданию базы данных «рис. 3.8»:

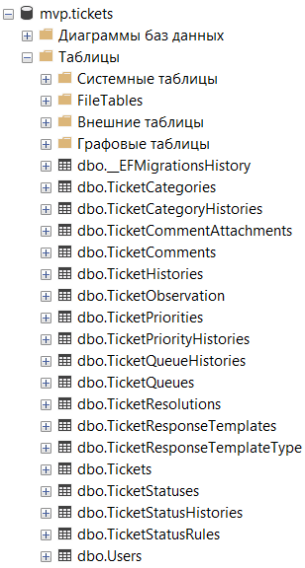


Рисунок 3.8 Обозреватель баз данных в MS SQL Server Management Studio

Таблица «__EFMigrationsHistory» является служебной, в ней хранится информация о выполненных миграциях EntityFramework.

3.2 Разработка серверной части

Разработку серверной части продолжаем в среде Visual Studio 2022. Для выполнения сложных SQL запросов будут использоваться процедуры, которые будут вызываться с помощью библиотеки Dapper. Для разработки и поддержки процедур удобнее использовать C#, для этого потребуется реализовать систему управления версиям процедур. В проекте «mvp.tickets.data» создадим новую модель «ProcedureVersion», добавим эту модель в класс «ApplicationDbContext» и создадим миграцию базы данных, модель «ProcedureVersion» будет описывать таблицу для хранения названий процедур и их текущих версий «рис. 3.9»:

```
namespace mvp.tickets.data.Models
{
    3 references
    public class ProcedureVersion
    {
        0 references
        public int Id { get; set; }
        1 reference
        public string Name { get; set; }
        0 references
        public int Version { get; set; }
        0 references
        public DateTimeOffset DateCreated { get; set; }
    }

    0 references
    public static class ProcedureVersionExtension
    {
        public const string TABLE_NAME = "ProcedureVersions";
        1 reference
        public static void DescribeProcedureVersion(this ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<ProcedureVersion>(c =>
            {
                c.Property(p => p.Name).IsRequired(true);
            });
            modelBuilder.Entity<ProcedureVersion>().ToTable(TABLE_NAME);
        }
    }
}
```

Рисунок 3.9 Класс «ProcedureVersion»

В этом же проекте создадим класс «ProcedureAttribute», который будет использоваться для нахождения процедур «рис. 3.10»:

```
namespace mvp.tickets.data.Procedures
{
    0 references
    public class ProcedureAttribute : Attribute
    {
    }
}
```

Рисунок 3.10 Класс «ProcedureAttribute»

В этом же проекте создадим класс «InitializationHelper», который будет отвечать за создание и обновление процедур в базе данных «рис. 3.11», этот класс нужно будет использовать один раз при запуске приложения:

```
public static class InitializationHelper
{
    0 references
    public static void ProceduresInit(ApplicationDbContext context)
    {
        Assembly currentAssem = Assembly.GetExecutingAssembly();
        var procedures = currentAssem.GetTypes().Where(s => s.CustomAttributes.Any(a => a.AttributeType == typeof(ProcedureAttribute))).ToList();
        foreach (var procedure in procedures)
        {
            var name = (string)procedure.GetProperty("Name").GetValue(null);
            var version = (int)procedure.GetProperty("Version").GetValue(null);
            var text = (string)procedure.GetProperty("Text").GetValue(null);
            var procedureVersion = context.ProcedureVersions.FirstOrDefault(pv => pv.Name == name);
            bool needCreate = true;
            if (procedureVersion != null)
            {
                if (version > procedureVersion.Version)
                {
                    string dropProcedure =
                        $@"IF EXISTS (select * from sysobjects where id = object_id(N'[{name}]')
                                and OBJECTPROPERTY(id, N'IsProcedure') = 1)
                            DROP PROCEDURE [{name}]";
                    context.Database.ExecuteSqlRaw(dropProcedure);
                }
                else
                    needCreate = false;
            }
            else
            {
                procedureVersion = new ProcedureVersion { Name = name, Version = 0, DateCreated = DateTimeOffset.Now };
                context.ProcedureVersions.Add(procedureVersion);
                context.SaveChanges();
            }
            if (needCreate)
            {
                context.Database.ExecuteSqlRaw(text);
                procedureVersion.Version = version;
                context.SaveChanges();
            }
        }
```

Рисунок 3.11 Класс «InitializationHelper»

Добавим дополнительных проект «mvp.tickets.domain» с типом «Class Library», в котором будет содержаться основная бизнес логика. Существующие проекты будут зависеть от этого проекта «рис. 3.12»:

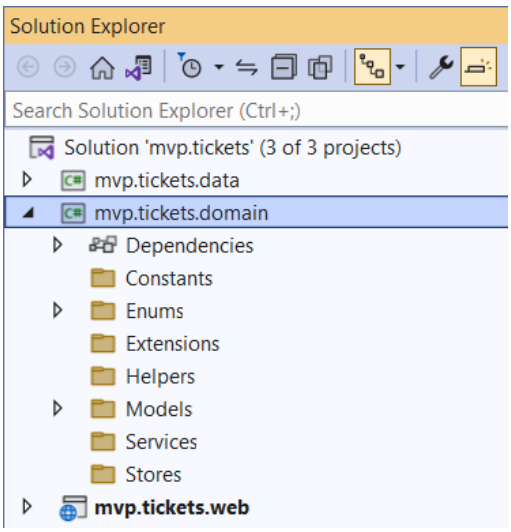


Рисунок 3.12 Структура проекта «mvp.tickets.domain»

Создадим в этом проекте перечисление «ResponseCode», которое будет использоваться в результирующих моделях для обозначения статуса операций «рис. 3.13»:

```
namespace mvp.tickets.domain.Enums
{
    0 references
    public enum ResponseCode
    {
        Unknown = 0,
        Success = 200,
        BadRequest = 400,
        Unauthorized = 401,
        NotFound = 404,
        Error = 500,
    }
}
```

Рисунок 3.13 Перечисление «ResponseCode»

В этом же проекте создадим базовые модели запросов, отдельно на получение данных и на изменение данных «рис. 3.14»:

```
namespace mvp.tickets.domain.Models
{
    3 references
    public interface IBaseRequest
    {
        1 reference
        Guid RequestId { get; set; }
    }
    2 references
    public record BaseRequest : IBaseRequest
    {
        1 reference
        public Guid RequestId { get; set; }
    }

    1 reference
    public interface IBaseCommandRequest : IBaseRequest { }
    0 references
    public record BaseCommandRequest : BaseRequest, IBaseCommandRequest { }

    2 references
    public interface IBaseQueryRequest : IBaseRequest { }
    1 reference
    public record BaseQueryRequest : BaseRequest, IBaseQueryRequest { }

    5 references
    public interface IBaseReportQueryRequest : IBaseQueryRequest
    {
        2 references
        int? Offset { get; set; }
        2 references
        int? Limmit { get; set; }
    }
    1 reference
    public record BaseReportQueryRequest : BaseQueryRequest, IBaseReportQueryRequest
    {
        2 references
        public int? Offset { get; set; }
        2 references
        public int? Limmit { get; set; }
    }
}
```

Рисунок 3.14 Базовые модели запросов

Таким же образом создадим базовые модели ответов «рис. 3.15»:

```
namespace mvp.tickets.domain.Models
{

    public interface IBaseResponse
    {

        bool IsSuccess { get; set; }
        4 references
        ResponseCodes Code { get; set; }
        2 references
        string ErrorMessage { get; set; }
    }
    2 references
    public record BaseResponse : IBaseResponse
    {
        4 references
        public bool IsSuccess { get; set; }
        4 references
        public ResponseCodes Code { get; set; }
        2 references
        public string ErrorMessage { get; set; }
    }

    1 reference
    public interface IBaseCommandResponse : IBaseResponse { }
    0 references
    public record BaseCommandResponse : BaseResponse, IBaseCommandResponse { }

    2 references
    public interface IBaseQueryResponse : IBaseResponse { }
    1 reference
    public record BaseQueryResponse : BaseResponse, IBaseQueryResponse { }

    7 references
    public interface IBaseReportQueryResponse<T> : IBaseQueryResponse
    {
        2 references
        IEnumerable<T> Data { get; set; }
        2 references
        int Total { get; set; }
    }
    3 references
    public record BaseReportQueryResponse<T> : BaseQueryResponse, IBaseReportQueryResponse<T>
    {
        2 references
        public IEnumerable<T> Data { get; set; }
        2 references
        public int Total { get; set; }
    }
}
```

Рисунок 3.15 Базовые модели ответов

Получение списка пользователей

Создадим в проекте «mvp.tickets.data» класс «UsersReportProcedure», который будет содержать логику SQL запросов для получения списка пользователей с пагинацией «рис. 3.16»:

```
namespace mvp.tickets.data.Procedures
{
    [Procedure]
    3 references
    public static class UsersReportProcedure
    {
        2 references
        public static string Name => "procUsersReport";
        1 reference
        public static int Version => 1;
        6 references
        public static class Params
        {
            3 references
            public static string Offset => "@offset";
            3 references
            public static string Limit => "@limit";
        }

        0 references
        public static string Text => $@"
/* version={Version} */
CREATE PROCEDURE [{Name}]
    {Params.Offset} INT,
    {Params.Limit} INT
AS
BEGIN
    SET NOCOUNT ON;
    SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

    SELECT
        [{nameof(User.Id)}] AS [{nameof(UserReportModel.Id)}]
        ,[{nameof(User.Email)}] AS [{nameof(UserReportModel.Email)}]
        ,[{nameof(User.FirstName)}] AS [{nameof(UserReportModel.FirstName)}]
        ,[{nameof(User.LastName)}] AS [{nameof(UserReportModel.LastName)}]
        ,[{nameof(User.Permissions)}] AS [{nameof(UserReportModel.Permissions)}]
        ,[{nameof(User.IsActive)}] AS [{nameof(UserReportModel.IsActive)}]
        ,[{nameof(User.IsLocked)}] AS [{nameof(UserReportModel.IsLocked)}]
        ,[{nameof(User.DateCreated)}] AS [{nameof(UserReportModel.DateCreated)}]
        ,[{nameof(User.DateModified)}] AS [{nameof(UserReportModel.DateModified)}]
    FROM [{UserExtension.TableName}]
    ORDER BY [{nameof(User.Id)}]
    OFFSET {Params.Offset} ROWS FETCH NEXT {Params.Limit} ROWS ONLY

    SELECT COUNT(*)
    FROM [{UserExtension.TableName}]
END";
    }
}
```

Рисунок 3.16 Класс «UsersReportProcedure»

В проекте «mvp.tickets.domain» создадим интерфейс IUserStore «рис. 3.17»:

```
namespace mvp.tickets.domain.Stores
{
    4 references
    public interface IUserStore
    {
        2 references
        Task<IBaseReportQueryResponse<IUserReportModel>> GetUsersReport(IBaseReportQueryRequest request);
    }
}
```

Рисунок 3.17 Интерфейс «IUserStore»

В этом же проекте создадим класс «UserService» и его интерфейс «рис. 3.18»:

```
namespace mvp.tickets.domain.Services
{
    4 references
    public class UserService : IUserService
    {
        private readonly IUserStore _userStore;
        private readonly ILogger<UserService> _logger;

        0 references
        public UserService(IUserStore userStore, ILogger<UserService> logger)
        {
            _userStore = userStore;
            _logger = logger;
        }

        2 references
        public async Task<IBaseReportQueryResponse<IUserReportModel>> GetUsersReport(IBaseReportQueryRequest request)
        {
            if (request == null)
            {
                return new BaseReportQueryResponse<IUserReportModel>
                {
                    IsSuccess = false,
                    Code = ResponseCodes.BadRequest
                };
            }

            IBaseReportQueryResponse<IUserReportModel> response = default;

            try
            {
                response = await _userStore.GetUsersReport(request).ConfigureAwait(false);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, ex.Message);
                response = new BaseReportQueryResponse<IUserReportModel>();
                response.HandleException(ex);
            }
            return response;
        }
    }
}
```

Рисунок 3.18 Класс «UserService»

В проекте «mvp.tickets.data» создадим реализацию интерфейса IUserStore «рис. 3.19»:

```
namespace mvp.tickets.data.Stores
{
    2 references
    public class UserStore : IUserStore
    {
        private readonly IConnectionStrings _connectionStrings;

        0 references
        public UserStore(IConnectionStrings connectionStrings)
        {
            _connectionStrings = connectionStrings ?? ThrowHelper.NullArgument<IConnectionStrings>();
        }

        2 references
        public async Task<IBaseReportQueryResponse<IUserReportModel>> GetUsersReport(IBaseReportQueryRequest request)
        {
            using (var connection = new SqlConnection(_connectionStrings.DefaultConnection))
            {
                DynamicParameters parameter = new DynamicParameters();
                parameter.Add(UsersReportProcedure.Params.Offset, request.Offset ?? 0, DbType.Int32);
                parameter.Add(UsersReportProcedure.Params.Limit, request.Limmit ?? ReportConstants.DEFAULT_LIMIT, DbType.Int32);

                using (var multi = await connection.QueryMultipleAsync(UsersReportProcedure.Name, param: parameter,
                    commandType: CommandType.StoredProcedure).ConfigureAwait(false))
                {
                    return new BaseReportQueryResponse<IUserReportModel>
                    {
                        Data = multi.Read<UserReportModel>().ToList(),
                        Total = multi.Read<int>().FirstOrDefault(),
                        IsSuccess = true,
                        Code = ResponseCodes.Success
                    };
                }
            }
        }
    }
}
```

Рисунок 3.19 Класс «UserStore»

В проекте «mvp.tickets.web» создадим класс UserController «рис. 3.20»:

```
namespace mvp.tickets.web.Controllers
{
    [ApiController]
    [Route("api/users")]
    1 reference
    public class UserController : ControllerBase
    {
        private readonly IUserService _userService;

        0 references
        public UserController(IUserService userService)
        {
            _userService = userService;
        }

        [HttpGet("report")]
        0 references
        public async Task<IBaseReportQueryResponse<IUserReportModel>> GetUsersReport([FromQuery] BaseReportQueryRequest request)
        {
            return await _userService.GetUsersReport(request);
        }
    }
}
```

Рисунок 3.20 Класс «UserController»

Зарегистрируем используемые классы в системе внедрения зависимостей, чтобы система автоматически создавала нужные экземпляры классов «рис. 3.21»:

```
#region Data
var connectionsStrings = new ConnectionStrings
{
    DefaultConnection = config.GetConnectionString("DefaultConnection")
};
services.AddSingleton<IConnectionStrings>(connectionsStrings);
services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionsStrings.DefaultConnection));
services.AddTransient<IUserStore, UserStore>();
#endregion

#region Domain
services.AddTransient<IUserService, UserService>();
#endregion
```

Рисунок 3.21 Регистрация зависимостей

Таким образом у нас есть класс «UserController», который обрабатываем HTTP GET вызовы для маршрута «/api/users/report», при вызове этого маршрута класс «UserController» будет использовать метод «GetUsersReport» из класса «UserService», в свою очередь этот класс будет использовать аналогичный метод из класса «UserStore», который в свою очередь вызовет процедуру «procUsersReport», описанную в классе «UsersReportProcedure». Все эти абстрактные слои позволяют изолировать зону ответственности классов между собой и упросить поддержку кода в будущем. Далее реализовываем оставшеюся логику в схожем стиле.

3.3 Разработка клиентской части

Для реализации клиентской части используется среда разработки Visual Code. Создадим в проекте «mvp.tickets.web» директорию «wwwroot», в которую

буду собираться готовые файлы клиентского приложения, и директорию «ClientApp», в которой будут находится исходные файлы клиентского приложения. Откроем директорию «ClientApp» с помощью Visual Code. Откроем в текущей папке терминал и выполним команду «yarn init» для инициализации клиентского приложения, в качестве имени приложения укажем «mvp.tickets».

С помощью менеджера пакетов «yarn» установим следующие зависимости «рис. 3.22», среди них основные это «react» и «mobx»:

```
"dependencies": {
  "axios": "^0.27.2",
  "mobx": "^6.3.12",
  "mobx-react-lite": "^3.2.3",
  "react": "^18.1.0",
  "react-dom": "^18.1.0"
},
"devDependencies": {
  "@babel/core": "^7.12.16",
  "@babel/plugin-transform-runtime": "^7.13.10",
  "@babel/preset-env": "^7.12.16",
  "@babel/preset-react": "^7.12.13",
  "@babel/preset-typescript": "^7.12.16",
  "@babel/runtime": "^7.13.10",
  "@pmmmwh/react-refresh-webpack-plugin": "^0.5.7",
  "@types/react": "^18.0.9",
  "@types/react-dom": "^18.0.5",
  "@typescript-eslint/eslint-plugin": "^5.26.0",
  "@typescript-eslint/parser": "^5.26.0",
  "babel-loader": "^8.2.2",
  "clean-webpack-plugin": "^4.0.0",
  "css-loader": "^6.7.1",
  "dotenv": "^16.0.1",
  "eslint": "^8.16.0",
  "eslint-config-prettier": "^8.5.0",
  "eslint-plugin-eslint-comments": "^3.2.0",
  "eslint-plugin-import": "^2.22.1",
  "eslint-plugin-jsx-a11y": "^6.4.1",
  "eslint-plugin-prettier": "^4.0.0",
  "eslint-plugin-react": "^7.22.0",
  "eslint-plugin-react-hooks": "^4.2.0",
  "html-webpack-plugin": "^5.1.0",
  "husky": "8.0.1",
  "lint-staged": "^12.4.2",
  "prettier": "^2.2.1",
  "react-refresh": "^0.13.0",
  "style-loader": "^3.3.1",
  "typescript": "^4.1.5",
  "webpack": "^5.21.2",
  "webpack-cli": "^4.5.0",
  "webpack-dev-server": "^4.9.0",
  "webpack-merge": "^5.7.3"
}
```

Рисунок 3.22 Зависимости клиентского приложения

Создадим в директории «ClientApp» следующие файлы:

- «tsconfig.json» - содержит настройки компиляции TypeScript;
- «aspnetcore-react.js» и «aspnetcore-https.js» - содержат настройки использования HTTPS на этапе разработки;
- «.prettierrc.js» - содержит настройка для форматирования кода с помощью библиотеки Prettier;
- «.eslintrc.js» - содержит правила корректности написания кода для ESLint;
- «.env.development.local» - содержит путь к SSL сертификату для локальной разработки;
- «.babelrc» - содержит настройки для библиотеки Babel, которая будет транслировать свежий синтаксис в старый для поддержки большего количества браузеров;
- «webpack.common.js», «webpack.config.js», «webpack.dev.js» и «webpack.prod.js» - содержит настройки сборки исходного кода с помощью библиотеки Webpack.

Добавим в файл «package.json» секцию «scripts» и дополнительные настройки компиляции «рис. 3.23»:

```
"scripts": {
  "prestart": "node aspnetcore-https && node aspnetcore-react",
  "start": "webpack serve --config webpack/webpack.config.js --env env=dev",
  "build": "webpack --config webpack/webpack.config.js --env env=prod",
  "lint": "eslint --fix \"./src/**/*.{js,jsx,ts,tsx,json}\"",
  "format": "prettier --write \"./src/**/*.{js,jsx,ts,tsx,json,css,scss,md}\"",
  "test": "echo \"Error: no test specified\" && exit 1"
},
"husky": {
  "hooks": {
    "pre-commit": "lint-staged"
  }
},
"lint-staged": {
  "src/**/*.{js,jsx,ts,tsx,json}": [
    "eslint --fix"
  ],
  "src/**/*.{js,jsx,ts,tsx,json,css,scss,md}": [
    "prettier --write"
  ]
},
```

Рисунок 3.23 Команды компиляции клиентского приложения

Создадим вложенную директорию «src» и в ней еще одну вложенную директорию «Store», в которой будут находится файлы с логикой хранения состояния, в директории «Store» создадим файл «UserStore.ts», который будет хранить загруженных с сервера пользователей «рис. 3.24»:

```
import axios from "axios"
import { observable, action, makeObservable } from 'mobx';
import { RootStore } from "./RootStore";

export class UserStore {
    private rootStore: RootStore;
    usersReport: IUserReportModel[]

    constructor(rootStore: RootStore) {
        this.rootStore = rootStore;
        this.usersReport = []
        makeObservable(this, {
            usersReport: observable,
            getUsersReport: action,
        })
    }

    getUsersReport(): void {
        axios.get<IUserReportModel[]>('/api/users/report')
            .then(response => {
                this.usersReport = response.data
            })
            .catch(error => {
                alert(error)
            })
    }
}

export interface IUserReportModel {
    id: number
    email: string
    firstName: string
    lastName: string
    permissions: number
    isActive: boolean
    isLocked: boolean
    dateCreated: Date
    dateModified: Date
}
```

Рисунок 3.23 Файл «UserStore.ts»

Создадим в этой же директории файл RootStore.ts «рис. 3.24»:

```
import { createContext, useContext } from 'react';
import { UserStore } from './UserStore';

export class RootStore {
  userStore: UserStore;

  constructor() {
    this.userStore = new UserStore(this);
  }
}

const RootStoreContext = createContext(new RootStore());

export function useRootStore() {
  return useContext(RootStoreContext)
}
```

Рисунок 3.24Файл «RootStore.ts»

В директории «src» создадим вложенную директорию «Components», в которой будут находиться UI[3] компоненты, внутри директории «Components» создадим файл UsersReportPage.tsx «рис. 3.25»:

```
import { observer } from 'mobx-react-lite';
import { FC, useEffect } from 'react';
import { useRootStore } from '../Store/RootStore';

interface IUsersReportPageProps {
}

const UsersReportPage: FC<IUsersReportPageProps> = (props) => {
  const store = useRootStore()
  useEffect(() => {
    store.userStore.getUsersReport()
  }, []);
  return <>
    {store.userStore.usersReport.map(user => <div key={user.id}>{user.email}</div>)}
  </>
}

export default observer(UsersReportPage)
```

Рисунок 3.25 Файл «UsersReportPage.tsx»

В директории «src» создадим файл «App.tsx», который будем корневым UI компонентом, и внутри которого будут находится все остальные «рис. 3.26»:

[3] UI – User Interface (пользовательский интерфейс)

```
import UsersReportPage from './Components/UsersReportPage'

export const App = () => {
  return (
    <>
      <h1>MVP Tickets</h1>
      <UsersReportPage />
    </>
  )
}
```

Рисунок 3.26 Файл «App.tsx»

В директории «src» создадим файл «index.tsx», который будет являться точкой входа в клиентском приложении «рис. 3.27»:

```
import ReactDOM from 'react-dom'
import { App } from './App'

ReactDOM.render(
    <App />,
    document.getElementById('root')
)
```

Рисунок 3.27 Файл «index.tsx»

В директории «src» создадим файл «index.html», который будет являться шаблоном страницы с подключенными ресурсами клиентского приложения, этот файл будет приходить клиенту с сервера при открытии приложения в браузере «рис. 3.28»:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>MVP Tickets</title>
</head>
<body>
    <div id="root"></div>
</body>
</html>
```

Рисунок 3.28 Файл «index.html»

Таким образом у нас есть компонент «UsersReportPage», который использует класс «UserStore» для загрузки пользователей с веб API маршрута «/api/users/report» и отображает их электронную почту. Для компиляции клиентского приложения нужно выполнить в терминале команду «yarn build».

После этого в директории «build» будут созданы выходные файлы клиентского приложения, требуется настроить серверную часть, чтобы при публикации приложения эти файлы копировались в серверную директорию «wwwroot» и файл «index.html» использовался по умолчанию, для этого в Visual Studio 2022 откроем файл «Program.cs» из проекта «mvp.tickets.web» и добавим вызов «app.MapFallbackToFile("index.html");» для того, чтобы использовался файл «index.html», когда ни один другой обработчик не может обработать текущий запрос. Далее для проекта «mvp.tickets.web» установим Nuget пакет «Microsoft.AspNetCore.SpaProxy», который обеспечивает взаимосвязь клиентского и серверного приложения, после установки требуется настроить файл проекта «рис. 3.29»:

```
  <ImplicitUsings>enable</ImplicitUsings>
  <SpaRoot>ClientApp\</SpaRoot>
  <DefaultItemExcludes>$(DefaultItemExcludes);$(SpaRoot)node_modules\**</DefaultItemExcludes>
  <SpaProxyServerUrl>https://localhost:8080</SpaProxyServerUrl>
  <SpaProxyLaunchCommand>yarn start</SpaProxyLaunchCommand>
</PropertyGroup>

<ItemGroup>
  <Content Remove="$(SpaRoot)**" />
  <None Remove="$(SpaRoot)**" />
  <None Include="$(SpaRoot)**" Exclude="$(SpaRoot)node_modules\**" />
</ItemGroup>

<Target Name="DebugEnsureNodeEnv" BeforeTargets="Build" Condition=" '$(Configuration)' == 'Debug' And !Exists('$(SpaRoot)node_modules') ">
  <Exec Command="node --version" ContinueOnError="true">
    <Output TaskParameter="ExitCode" PropertyName="ErrorCode" />
  </Exec>
  <Exec WorkingDirectory="$(SpaRoot)" Command="yarn install" />
</Target>

<Target Name="PublishRunWebpack" AfterTargets="ComputeFilesToPublish">
  <Exec WorkingDirectory="$(SpaRoot)" Command="yarn install" />
  <Exec WorkingDirectory="$(SpaRoot)" Command="yarn build" />

  <ItemGroup>
    <DistFiles Include="$(SpaRoot)build\**" />
    <ResolvedFileToPublish Include="@(DistFiles->'%(FullPath)')" Exclude="@(ResolvedFileToPublish)">
      <RelativePath>wwwroot\%(RecursiveDir)%(FileName)%(Extension)</RelativePath>
      <CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
      <ExcludeFromSingleFile>true</ExcludeFromSingleFile>
    </ResolvedFileToPublish>
  </ItemGroup>
</Target>
```

Рисунок 3.29 Файл «mvp.tickets.web.csproj»

Таким образом при публикации приложения скомпилированные файлы клиентского приложения будут скопированы в папку «wwwroot», так же во время запуска серверного приложения из Visual Studio будет автоматически выполняться команда «yarn start», которая скомпилирует и запустит клиентское приложение отдельным процессом, запросы от клиентского приложения будут проксироваться в серверное приложение. Добавим в серверном приложении логику создания пользователя с правами администратора «рис. 3.30», после этого разрабатываем UI компоненты:

```
context.Database.Migrate();

if (!context.Users.Any())
{
    context.Users.Add(new User
    {
        Email = "tickets@mvp-stack.com",
        FirstName = "Admin",
        LastName = "Admin",
        IsActive = true,
        IsLocked = false,
        Permissions = domain.Enums.Permissions.Admin,
        DateCreated = DateTimeOffset.Now,
        DateModified = DateTimeOffset.Now
    });
    context.SaveChanges();
}
```

Рисунок 3.30 Создание администратора

Исходные коды приложения размещены на Github и доступны по ссылке https://github.com/vfelinis/mvp.tickets

ЗАКЛЮЧЕНИЕ

Целью исследования являлась разработка информационной системы для принятия и обработки заявок клиентов на основе веб-приложения.

Задачами исследования были:

- исследовать теоретические аспекты вопроса;
- проанализировать существующие системы принятия и обработки заявок клиентов;
- сформировать функциональные требования;
- спроектировать приложение и базу данных;
- реализовать приложение и базу данных.

В ходе исследования цель и задачи были достигнуты.

Курсовая работа состояла из трех глав. В первой главе были изучены теоретические основы систем принятия и обработки заявок клиентов, найдены и проанализированы существующие системы. Во второй главе были сформулированы функциональные требования разрабатываемой системы и спроектировано будущее решение. В третьей главе были описаны этапы разработки базы данных, серверного и клиентского приложения.

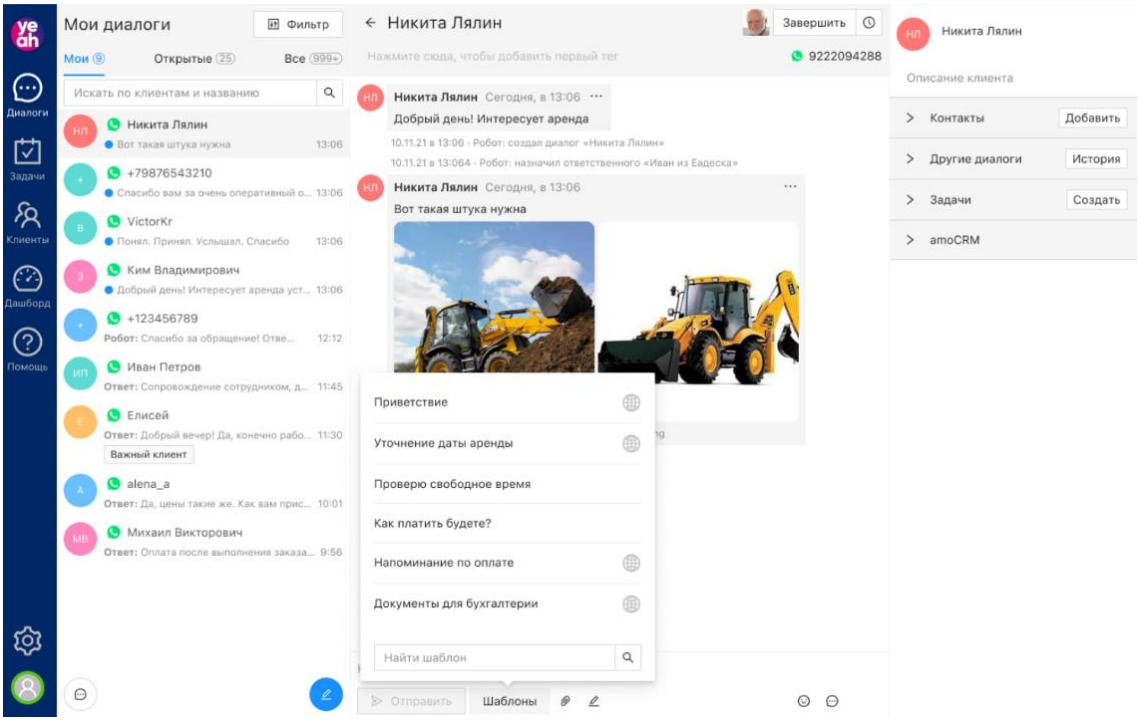
Результатом работы является информационная система для принятия и обработки заявок клиентов на основе веб-приложения. Исходные коды приложения размещены на Github. Планируется, что в будущем система будет дорабатываться, появятся следующие новые функции: база знаний, которая предоставит пользователям возможность искать информацию по своей проблеме; интеграция с email, что позволит автоматически создавать заявки через обработку электронных писем; интеграция с чат приложениями, что так же позволит автоматически создавать заявки через обработку сообщений; чат-боты для автоматических ответов и т.д.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

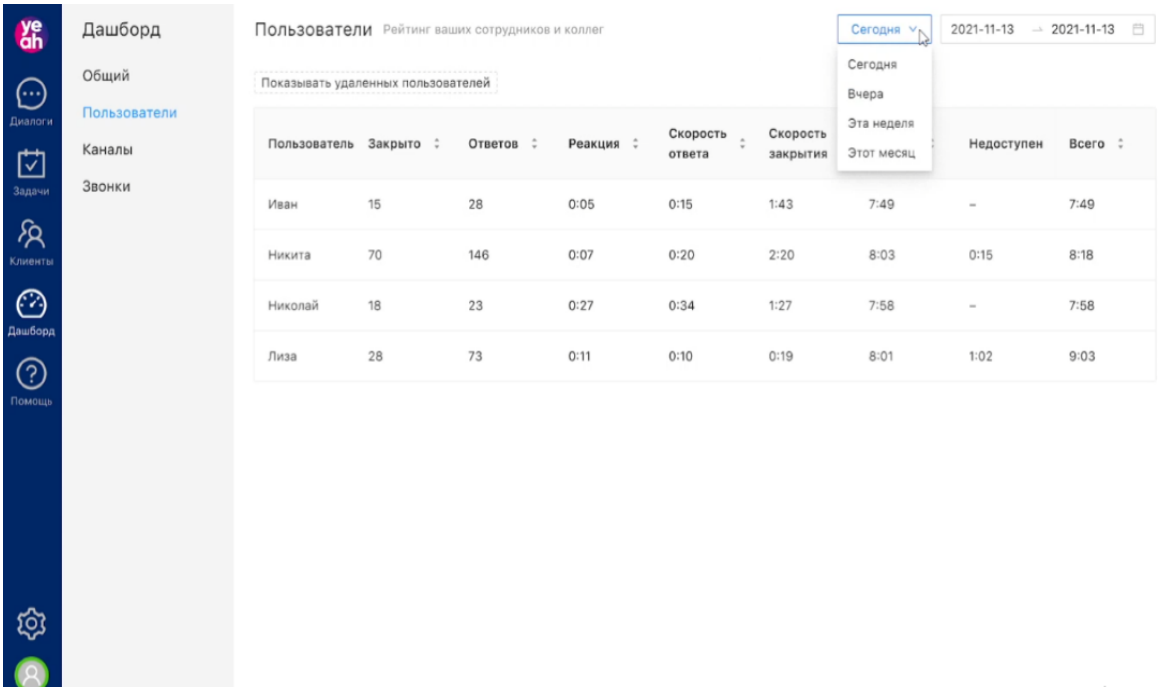
1. Гринберг, Пол. CRM со скоростью света. — СПб.: Символ Плюс, 2007. — 528 с.
2. Help Desk система: что это и зачем она нужна вашей компании?. URL: https://okdesk.ru/blog/chto-takoe-help-desk (дата обращения: 10.05.2022).
3. Официальный сайт системы Еадеск. URL: https://yeahdesk.ru/ (дата обращения: 14.05.2022).
4. Официальный сайт системы Юздеск. URL: https://usedesk.ru/ (дата обращения: 16.05.2022).
5. Официальный сайт системы Kayako. URL: https://kayako.com/ (дата обращения: 16.05.2022).
6. ГОСТ 34.602-89 Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы (дата обращения: 20.05.2022)
7. Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика — М.: Вильямс И.Д., 2017. — 1440 с.
8. Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель. SQL: полное руководство, 3-е издание. — М.: «Вильямс», 2014. — 960 с.
9. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению (дата обращения: 01.05.2022)
10. Алистер, Коберн Современные методы описания функциональных требований к системам — М.: ЛОРИ, 2014. — 706 с.
11. Роберт Э. Уолтерс, Майкл Коулс. SQL Server 2008: ускоренный курс для профессионалов, 2008. — М.: «Вильямс», 2008. — 768 с.
12. Джон Скит. C# для профессионалов: тонкости программирования, 3-е издание. — М.: «Вильямс», 2014. — 608 с.
13. Томас Марк Тиленс. React в действии. — СПб.: «Питер», 2019. — 368 с.

ПРИЛОЖЕНИЯ

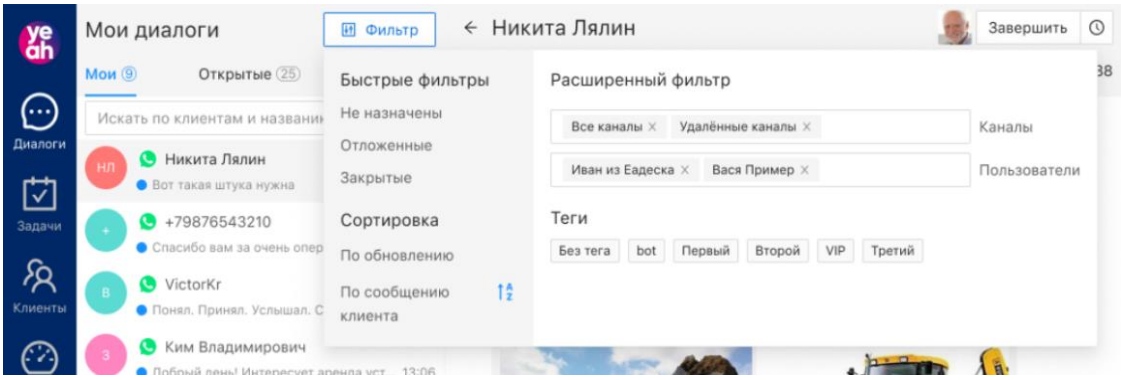
ПРИЛОЖЕНИЕ А. ШАБЛОНЫ ОТВЕТОВ В ИНФОРМАЦИОННОЙ СИСТЕМЕ ЕАДЕСК



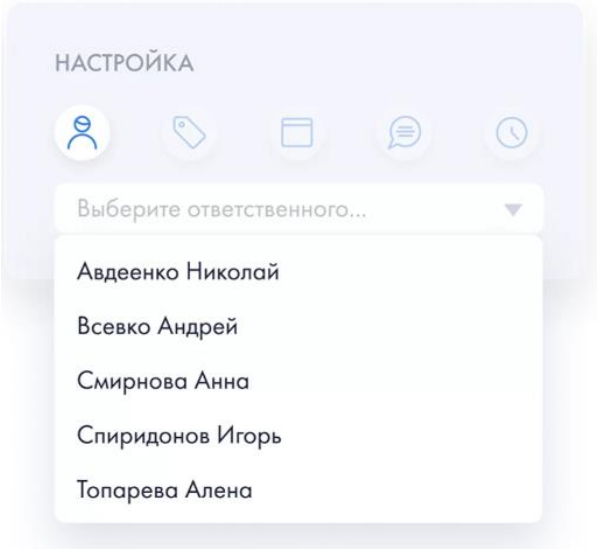
ПРИЛОЖЕНИЕ Б. СТРАНИЦА ОТЧЕТОВ В ИНФОРМАЦИОННОЙ СИСТЕМЕ ЕАДЕСК



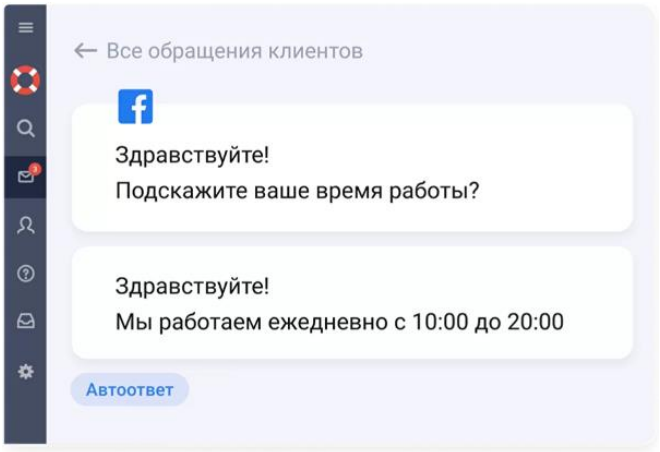
ПРИЛОЖЕНИЕ В. ФИЛЬТРАЦИЯ В ИНФОРМАЦИОННОЙ СИСТЕМЕ ЕАДЕСК



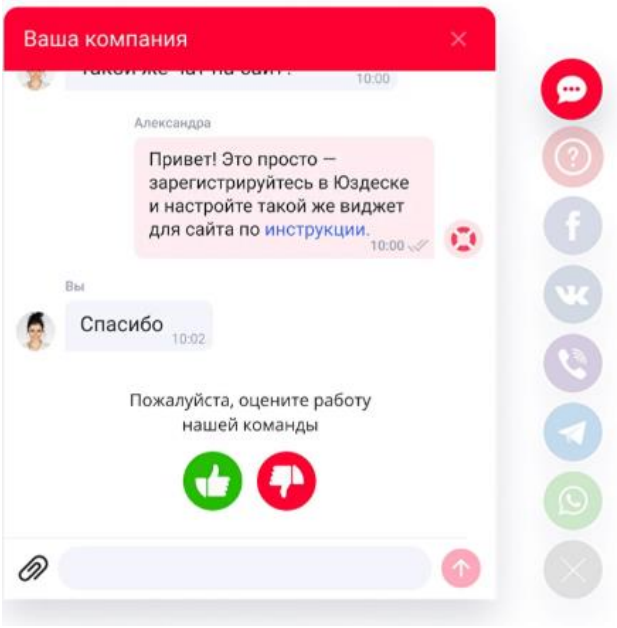
ПРИЛОЖЕНИЕ Г. СМЕНА НАЗНАЧЕНИЯ В ИНФОРМАЦИОННОЙ СИСТЕМЕ ЮЗДЕСК



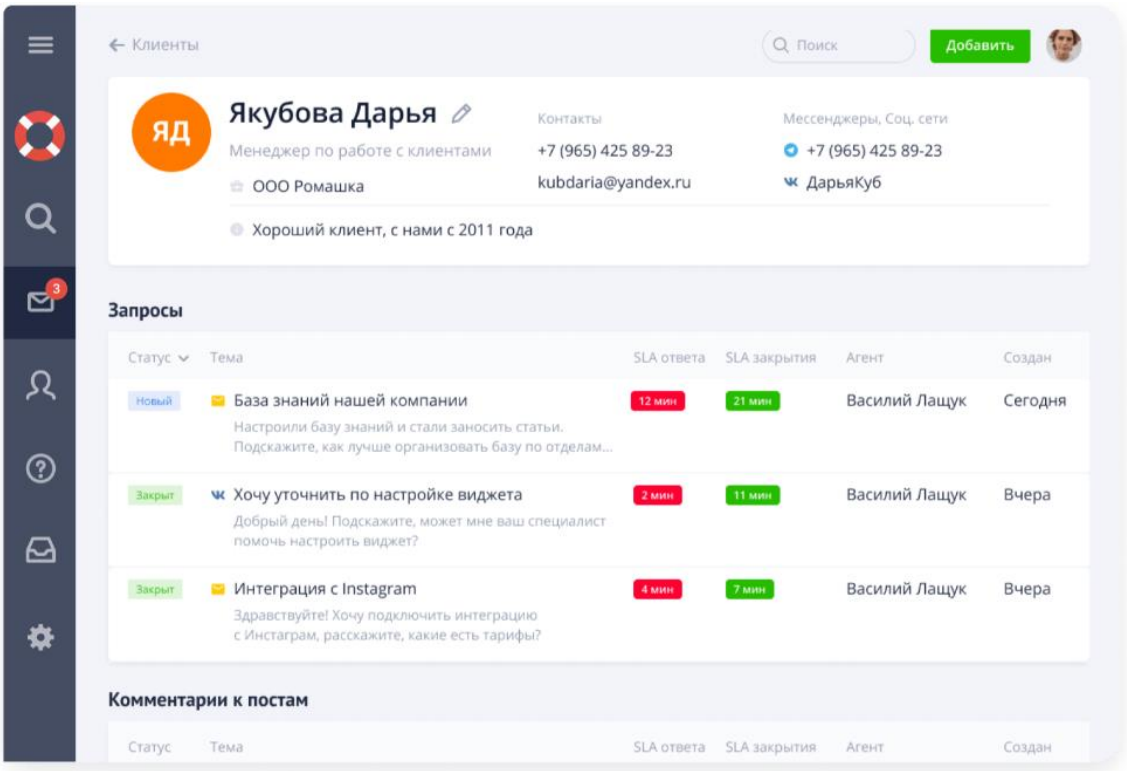
ПРИЛОЖЕНИЕ Д. АВТООТВЕТЫ В ИНФОРМАЦИОННОЙ СИСТЕМЕ ЮЗДЕСК



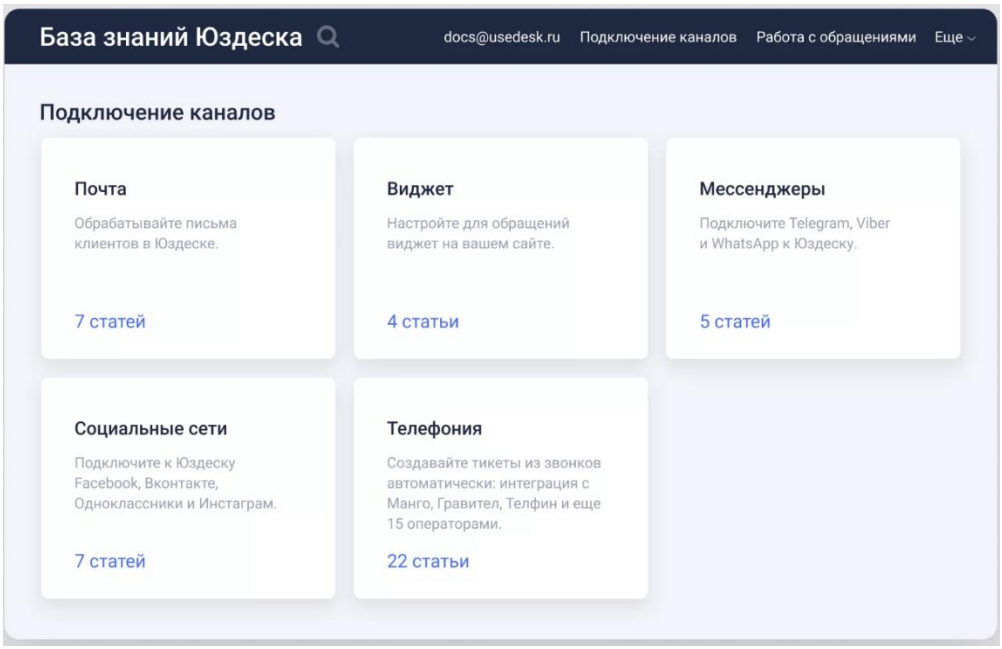
ПРИЛОЖЕНИЕ Е. ОЦЕНКА КАЧЕСТВА В ИНФОРМАЦИОННОЙ СИСТЕМЕ ЮЗДЕСК



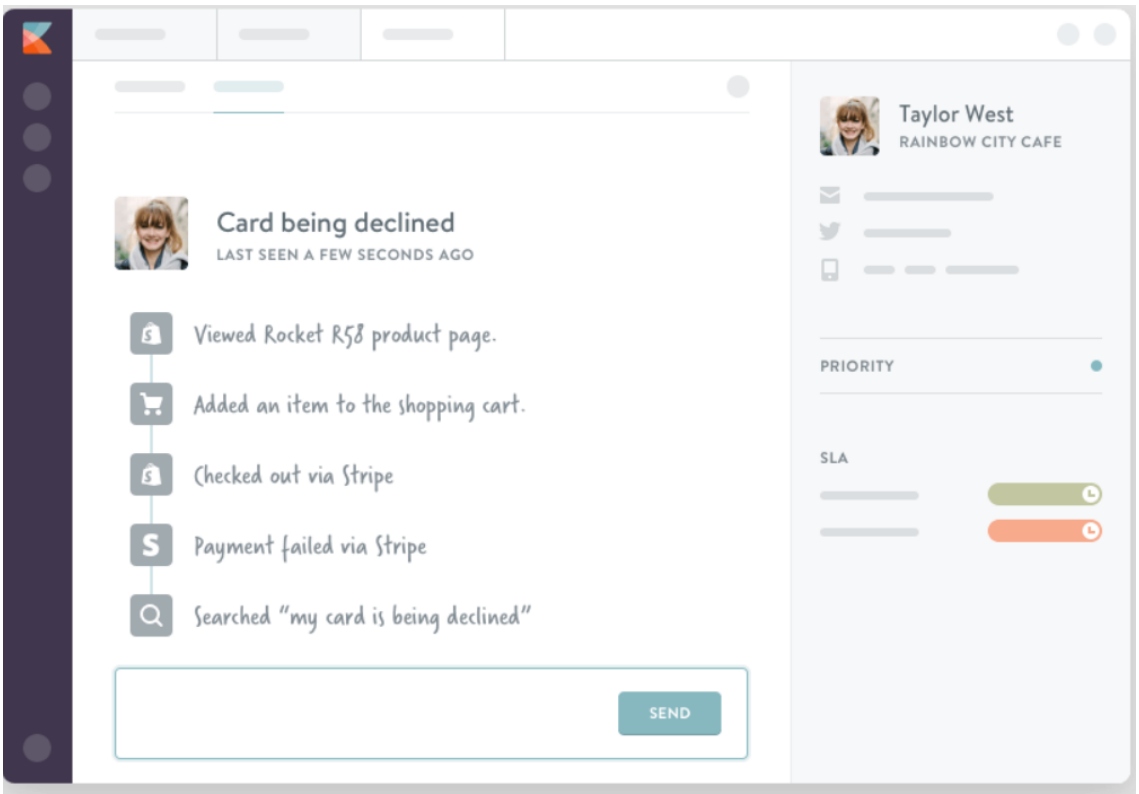
ПРИЛОЖЕНИЕ Ж. ИСТОРИЯ В ИНФОРМАЦИОННОЙ СИСТЕМЕ ЮЗДЕСК



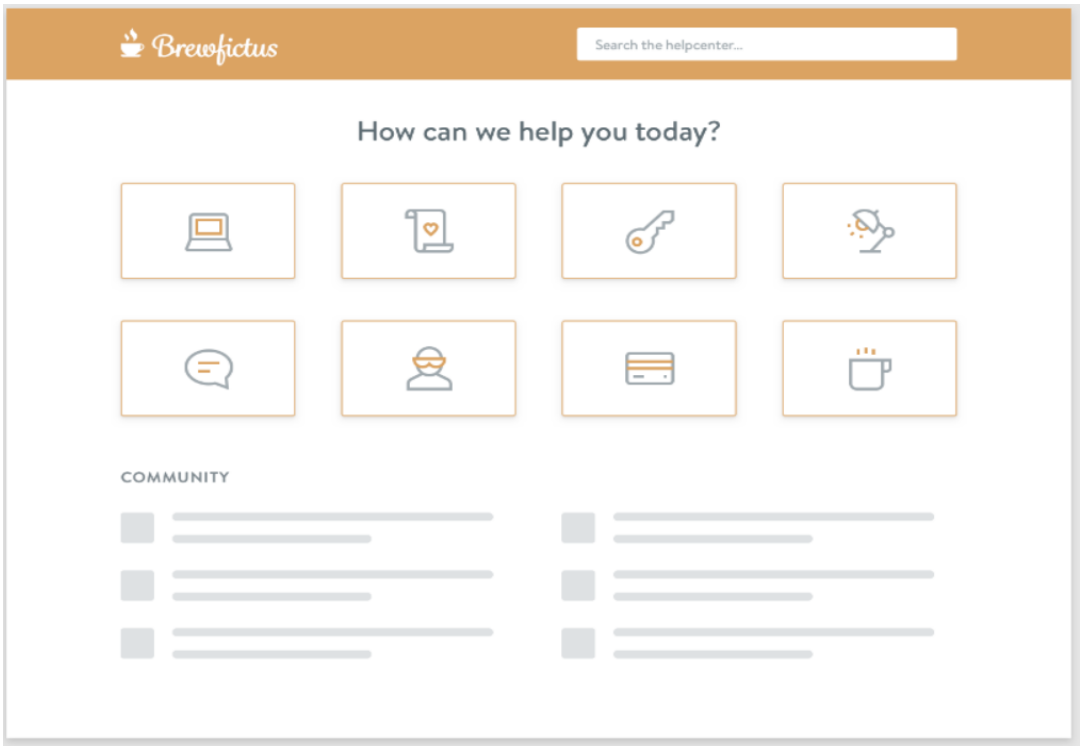
ПРИЛОЖЕНИЕ З. БАЗА ЗНАНИЙ В ИНФОРМАЦИОННОЙ СИСТЕМЕ ЮЗДЕСК



ПРИЛОЖЕНИЕ И. ИСТОРИЯ АКТИВНОСТИ ПОЛЬЗОВАТЕЛЯ В ИНФОРМАЦИОННОЙ СИСТЕМЕ KAYAKO



ПРИЛОЖЕНИЕ К. БАЗА ЗНАНИЙ В ИНФОРМАЦИОННОЙ СИСТЕМЕ KAYAKO



ПРИЛОЖЕНИЕ Л. ОТЧЕТЫ В ИНФОРМАЦИОННОЙ СИСТЕМЕ KAYAKO