**Тема 2. Ознакомление со структурой и характером деятельности предприятия**

Практика проходит на предприятии «ООО Главстрой-СПб специализированный застройщик» которая является одной из крупнейших строительных компаний Северо-Западного региона РФ. Структура компании представлена на рисунке 1.

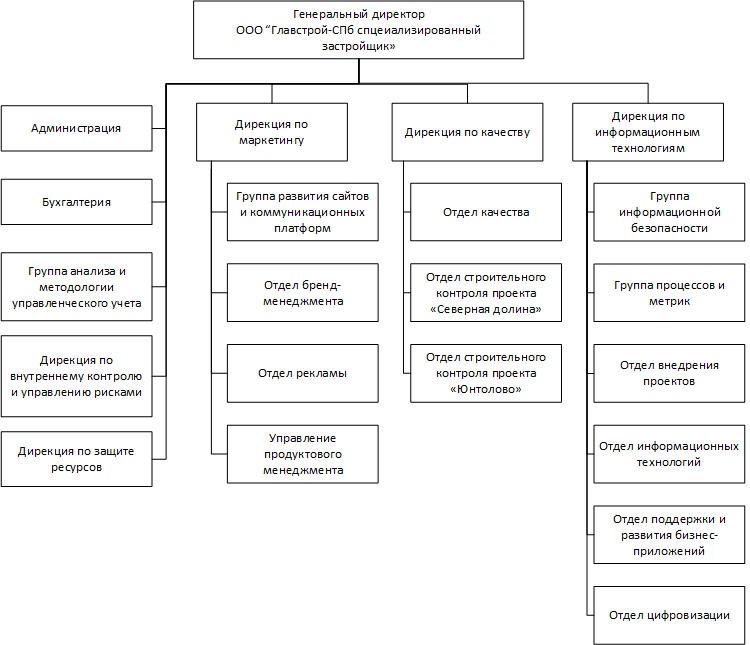


Рисунок 1 – Структура предприятия

Компания имеет линейно-функциональную структуру управления, где каждый отдел выполняет свои определённые функции. Отделы взаимодействуют друг с другом через функциональные связи. Такая структура позволяет обеспечить чёткое распределение ответственности между сотрудниками и эффективное управление процессами.

Во главе компании стоит генеральный директор в лице Сурвилло Татьяны Геннадиевны. В компании есть множество подразделений, таких как администрация, бухгалтерия, дирекции по маркетингу, качеству, защите ресурсов и т. д. Во главе каждого из подразделений стоит директор. Некоторые подразделения делятся на отделы. Например, дирекция по информационным технологиям делится на 6 отделов, в числе которых отдел цифровизации, в котором проходит данная практика.

«ООО Главстрой-СПб специализированный застройщик» по характеру деятельности является строительной компанией. Она занимается реализацией проектов жилищного строительства в сегментах комфорт- и бизнес-класса. География присутствия компании охватывает все районы Санкт-Петербурга и Ленинградскую область.

Основными направлениями деятельности компании являются:

* Строительство жилых комплексов комфорт- и бизнес-класса;
* Реализация объектов коммерческой недвижимости;
* Разработка проектной документации для строительства зданий и сооружений различного назначения.

В настоящее время компания реализует ряд крупных проектов жилищного строительства в различных районах Санкт-Петербурга и Ленинградской области. Среди наиболее значимых проектов можно выделить: ЖК «Северная долина», ЖК «Юнтолово», ЖК «Панорама 360».

**Тема 3. Сбор материалов для составления технического задания по теме дипломного проекта**

3.1 Анализ деятельности предприятия

ООО «Главстрой-СПб специализированный застройщик» – это строительная компания, специализирующаяся на разработке и реализации строительных проектов в городе Санкт-Петербурге и его окрестностях. Компания была основана в 2006 году и с тех пор успешно развивается, реализуя проекты жилищного строительства в сегментах комфорт- и бизнес-класса. Предприятие занимается возведением жилых и коммерческих объектов, а также инфраструктурных сооружений.

География деятельности компании охватывает все районы Санкт-Петербурга и Ленинградскую область. Основные направления деятельности включают строительство жилых комплексов комфорт- и бизнес-класса, реализацию объектов коммерческой недвижимости и разработку проектной документации для строительства зданий и сооружений различного назначения.

Так как компания занимается также и обслуживанием реализованных объектов, то для повышения качества взаимодействия с клиентами руководством было принято решение о разработке программного модуля по интеграции в бизнес-процессы компании сервиса голосовых рассылок. Подобное программное решение может быть полезно во многих сценариях, например:

* Уведомления о задолженности: Сервис голосовых рассылок может использоваться для автоматического уведомления жильцов о задолженности по оплате коммунальных услуг. Это поможет управляющей компании сократить количество просроченных платежей и улучшить финансовое состояние организации.
* Информирование о проведении ремонтных работ: Сервис голосовых рассылок может использоваться для информирования жильцов о проведении ремонтных работ в доме или на территории комплекса. Это позволит избежать конфликтов между управляющей компанией и жителями из-за неожиданного шума или других неудобств.
* Организация опросов и голосований: Сервис голосовых рассылок может использоваться для проведения опросов среди жителей дома или комплекса по различным вопросам (например, опрос об удовлетворенности качеством обслуживания).

Таким образом, интеграция сервиса голосовых рассылок будет являться эффективным инструментом для компании, который поможет улучшить коммуникацию с клиентами и повысить качество обслуживания.

3.2 Анализ программных средств, использованных для разработки

Для разработки проекта был выбран набор программных средств, включающий в себя интегрированную среду разработки (IDE), систему управления базами данных, систему контроля версий, инструменты тестирования.

В качестве системы управления базами данных была выбрана Microsoft SQL Management Studio. Данный инструмент разработан специально для работы с Microsoft SQL Server, используемом в качестве сервера баз данных на предприятии, что обеспечивает высокую совместимость и эффективность при работе с этой базой данных. Также, SSMS предоставляет широкий набор инструментов для администрирования баз данных, написания запросов, создания отчетов и управления безопасностью. Среда предоставляет инструменты для мониторинга производительности баз данных, выявления узких мест и оптимизации запросов, что позволяет повысить эффективность работы с базой данных.

В качестве языка разработки был выбран Python, а в качестве интегрированной среды разработки – PyCharm. Python прост в использовании и позволяет вести разработку с высокой скоростью, а так же отлично подходит для веб-разработки. У Python множество библиотечных инструментов, пакетов услуг и удобных фреймворков, в частности – flask, легкий и мощный фреймворк для разработки веб-приложений, который будет использоваться в разрабатываемом проекте. В свою очередь PyCharm, как среда разработки на Python, предлагает широкий набор мощных функций и инструментов, которые помогут увеличить производительность и улучшить качество вашего кода. Например, автодополнение кода, подсветка синтаксиса, отладчик, рефакторинг кода и многое другое. PyCharm интегрируется с различными инструментами и сервисами, такими как системы контроля версий (например, Git), виртуальные окружения, базы данных, фреймворки и т.д. Это облегчает работу с различными аспектами разработки.

В качестве системы контроля версий на сервере компании развернут Bonobo Git Server. Это простой и лёгкий git-сервер, являющийся отличным решением для управления git-репозиториями в организации. Также он является бесплатным и имеет открытый исходный код, что позволяет изменять и адаптировать его под свои потребности.

В качестве инструмента для тестирования бэкенда (в частности, для работы с API сервиса «Звонобот») был выбран Postman. Данный инструмент предлагает бесплатную версию со множеством полезных функций, которые покрывают большинство потребностей при тестировании API. Postman позволяет организовывать запросы в коллекции, что упрощает их использование и повторное использование, создавать и выполнять автотесты, проверять ответы на соответствие ожидаемым результатам и мониторить производительность API.

3.3 Описание этапов разработки

На данный момент, в компании уже начата разработка модуля для взаимодействия с сервисом «Звонобот». Доработка данного модуля будет включать следующие этапы:

1. Доработка веб-интерфейса исходя из требований заказчиков: внесение изменений в фронтенд проекта согласно требованиями. Работы на данном этапе будет включать в себя добавление/удаление полей на страницах веб-интерфейса, реализацию нового функционала, в том числе добавление новых веб-страниц в проект.
2. Рефакторинг кода бэкенда имеющегося сервиса и исправление багов в нем: улучшение структуры и читаемости кода, оптимизация производительности, устранение возможных проблем и улучшение архитектуры приложения.
3. Изучение и предоставление описания части API функционала голосового сервиса «Звонобот», отвечающую за создание и управление роботизированными диалоговыми прозвонами: изучение соответствующих эндпоинтов API сервиса «Звонобот», предоставление описания их параметров, возвращаемых результатов и других особенностей взаимодействия.
4. Анализ решений в области синтеза речи с целью дальнейшего внедрения наиболее зарекомендовавших себя в работу сервиса: изучение сервисов, предоставляющих возможность синтеза речи из текста, для создания с их помощью аудиороликов, которые в дальнейшем будут использоваться в роботизированных звонках.
5. Интеграция функции роботизированного диалогового прозвона в имеющийся сервис перезвонов: добавление требуемых заказчиком функций и сценариев диалоговых звонков на основании изученного API-функционала, обработка результатов прозвонов.

**Тема 4. Разработка программного обеспечения на основе технического задания по теме дипломного проекта**

4.1 Доработка веб-интерфейса

Структура и логика работы веб-интерфейса, который имеется на данный момент, представлены на wireframe-диаграмме (приложение А). Как понятно из неё, интерфейс имеет проблемы с навигацией (нет возможности возвращаться на предыдущие страницы), в нем отсутствуют некоторые страницы (например, нет страницы с результатами прозвона), а также требуется внесение некоторых изменений в соответствии с требованиями заказчика.

4.1.1 Страницы выбора типа обзвона

Для обеспечения навигации по интерфейсу, на страницу выбора типа обзвона добавлена кнопка «выйти», по нажатию на которую пользователь может вернуться на страницу авторизации. Итоговый вариант страницы представлен на рисунке 3.

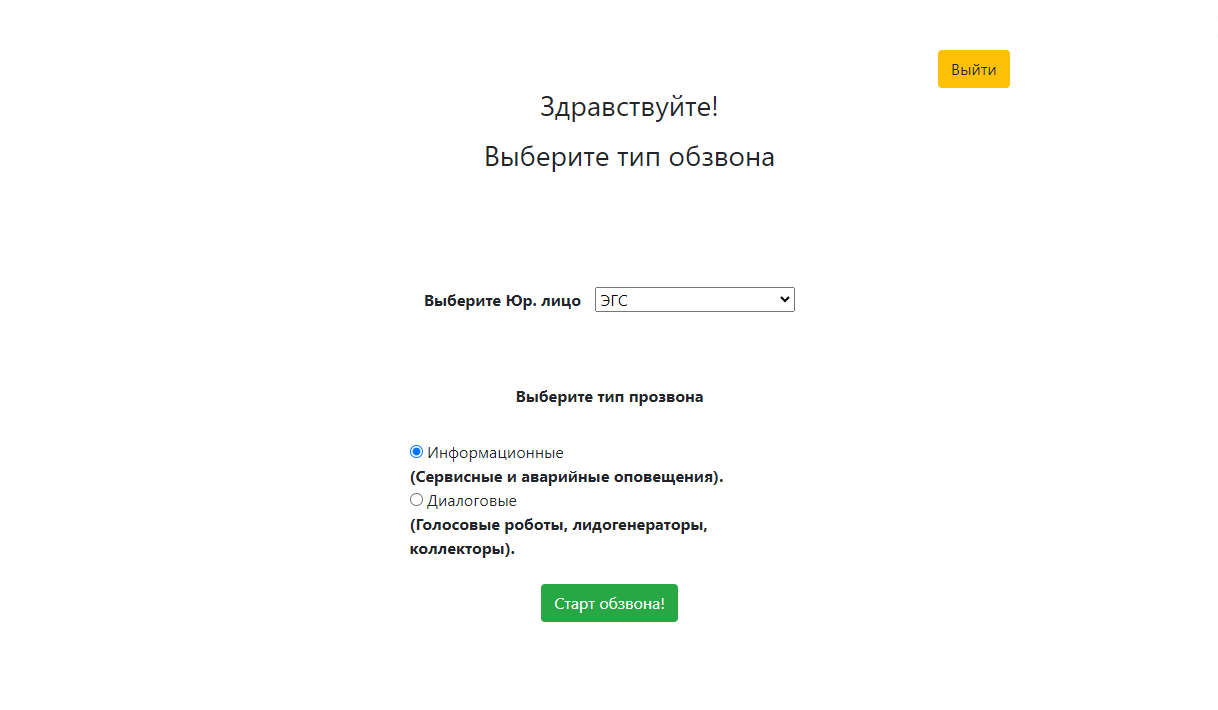


Рисунок 3 – Страница выбора типа обзвона

4.1.2 Страница информационных прозвонов

Сервис «звонобот» позволяет для голосовых рассылок использовать либо уже созданный и сохраненный в личном кабинете аудиоролик, либо текст, который будет преобразован сервисом в голос. Таким образом, на странице информационных прозвонов в выпадающем списке пользователю необходимо выбрать тип рингтона («Текст» / «ID из звонобота»). Был добавлен скрипт, реализующий логику отображения полей ввода ID рингтона или текста в зависимости от выбранной опции (изначально оба поля отображались всегда). Код скрипта представлен на рисунке 4.

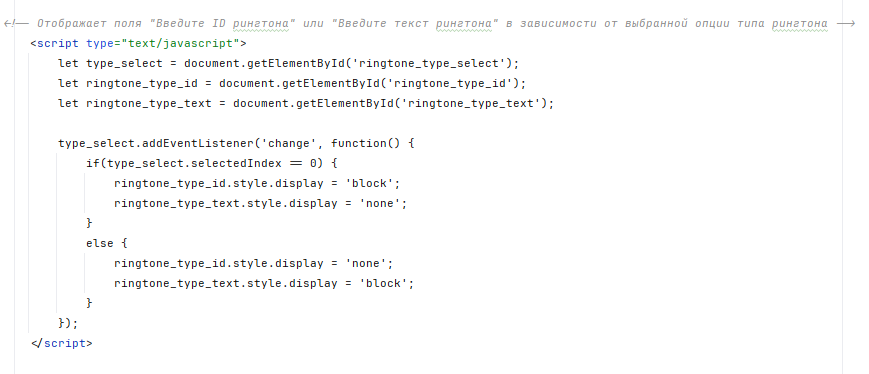


Рисунок 4 – Скрипт для отображения полей «Текст» / «ID из звонобота»

В соответствии с требованиями заказчика, в секции «что считать дозвоном» был оставлен только пункт «прослушал более N секунд ролика», и убран чекбокс «автоответчик». На страницу добавлено отображение баланса на счету личного кабинета в сервисе «звонобот».

Было замечено, что загрузка страницы занимает продолжительное время – около 10 секунд. Это происходит из-за получения исходящих номеров телефонов из «звонобота». До тех пор, пока все номера не будут получены, страница не будет отображена. Для исправления этой проблемы логика работы сайта была изменена следующим образом: страница теперь возвращается мгновенно с пустым списком, и посылает Ajax-запрос на получение номеров (код скрипта, реализующего данную логику на стороне клиента представлен на рисунке 5).

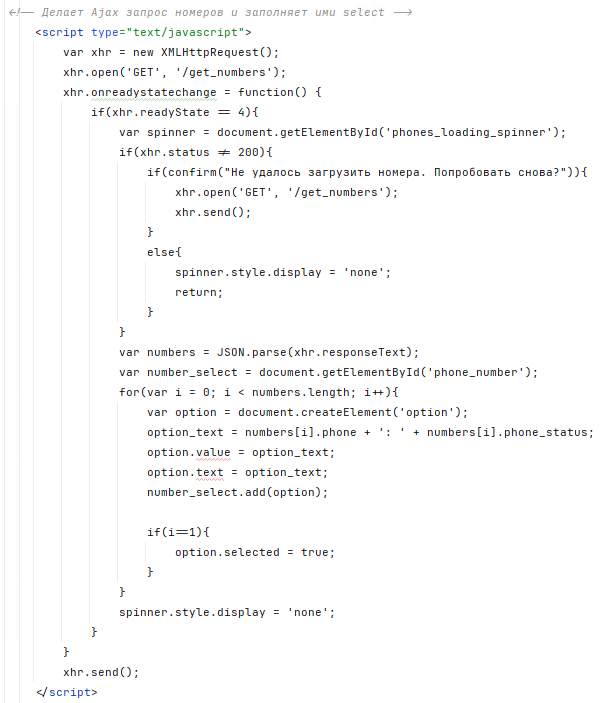


Рисунок 5 – Скрипт для получения исходящих номеров

Пока ожидается ответ от сервера, рядом с выпадающим списком вращается спиннер. В это время пользователь может начать заполнять другие поля. Как только ответ будет получен, выпадающий список будет заполен, и пользователь сможет выбрать исходящий номер.

Для удобства пользователя было добавлено отображение количества номеров в загруженном Excel-файле. Для работы с файлами .xlsx/.xls с помощью языка JavaScript была использована библиотека SheetJS. Код скрипта для подсчета количества номеров в файле представлен на рисунке 6.



Рисунок 6 – Скрипт для подсчета номеров в Excel-файле

В случае, если выбран 1 круг прозвона, поле для ввода расстояния между перезвонами в минутах не должно отображаться. Код скрипта, реализующего эту логику, представлен на рисунке 7.

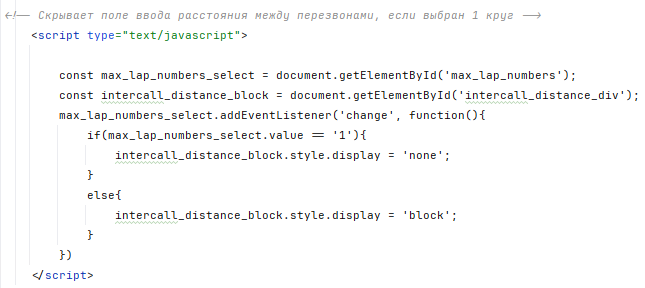


Рисунок 7 – Скрипт для скрытия поля ввода расстояния между перезвонами

Также, на страницу информационных прозвонов была добавлена кнопка «назад», нажав на которую пользователь может попасть обратно на страницу выбора типа прозвона. Итоговый вариант страница информационных прозвонов представлена на рисунке 8.

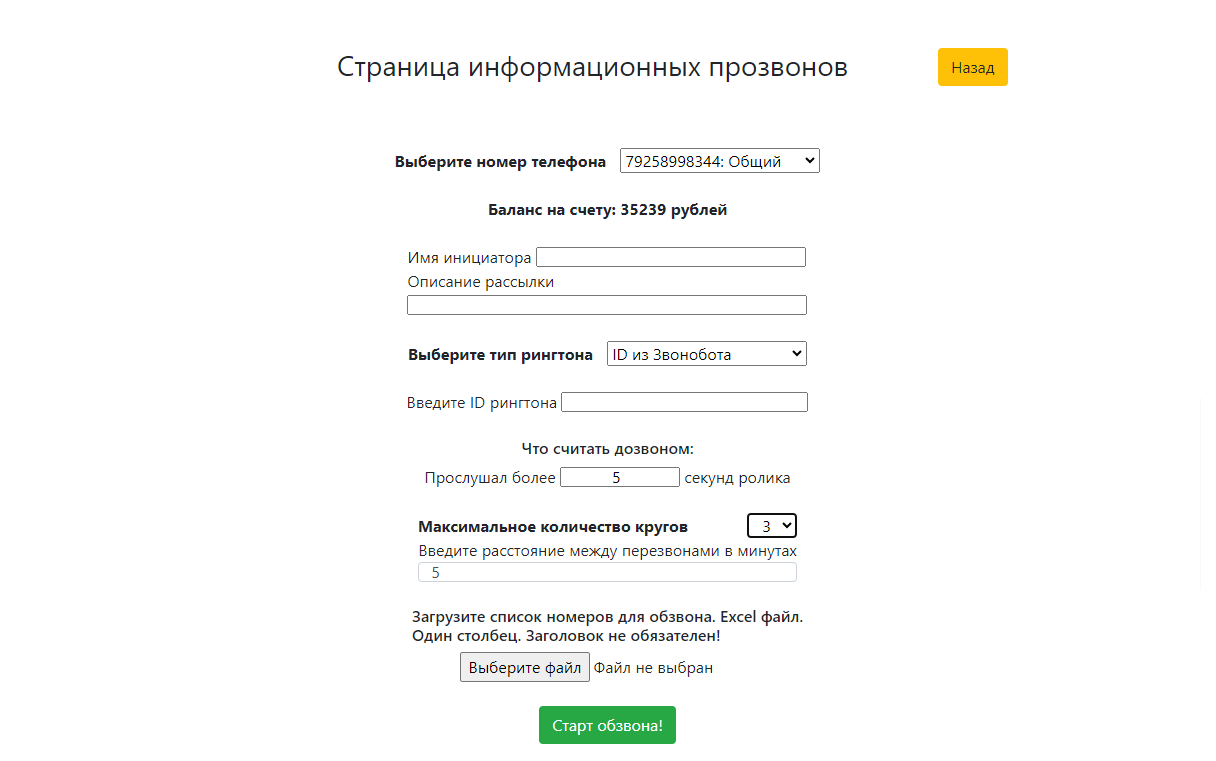


Рисунок 8 – Страница информационных прозвонов

4.1.3 Страница уведомления о незавершенном прозвоне

В интерфейс добавлена страница, оповещающая пользователя о том, что последний начатый им прозвон не был завершен (рисунок 9).

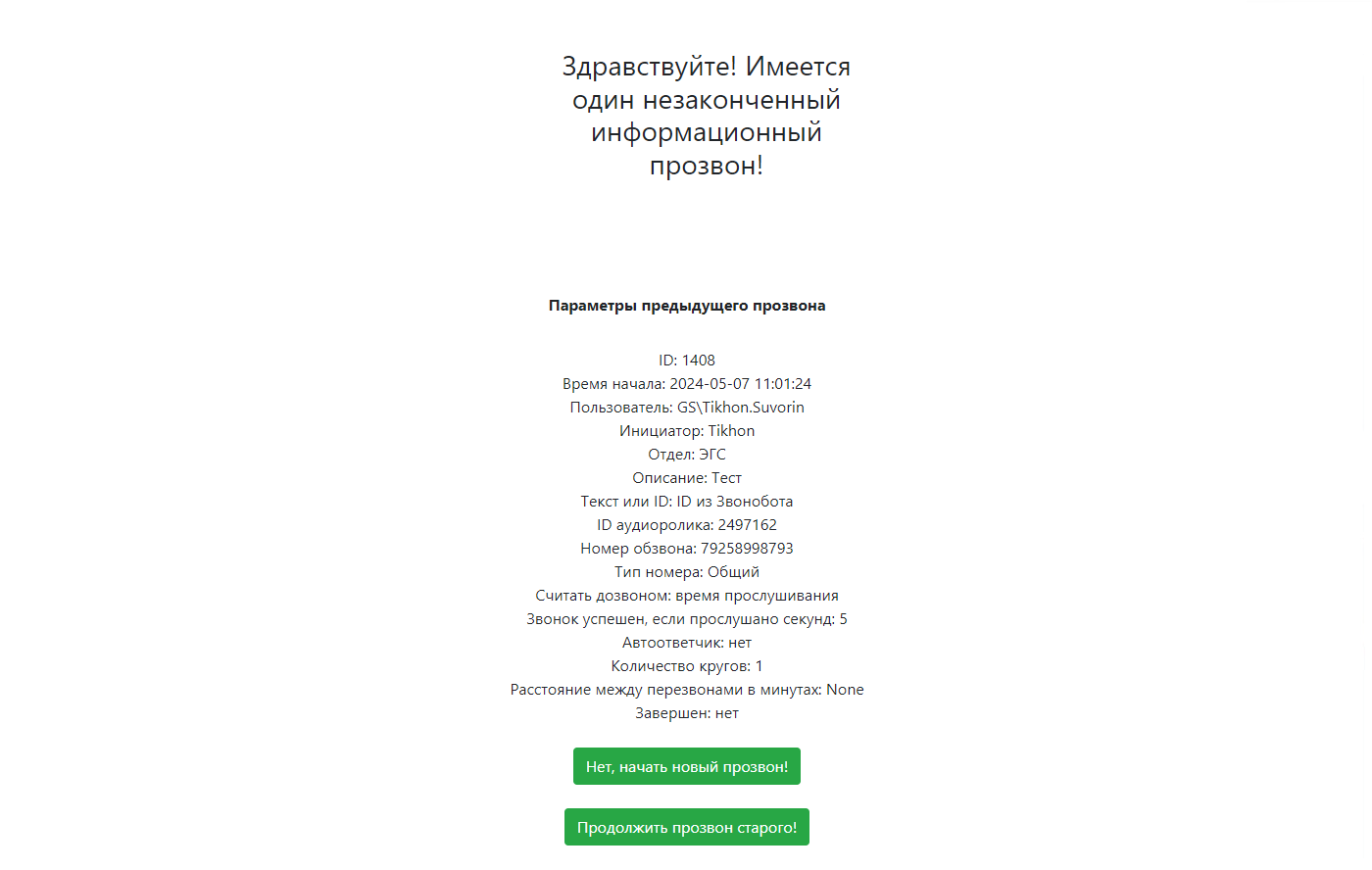


Рисунок 9 – Страница с информацией о незаконченном прозвоне

На данной странице отображаются параметры незавершенного прозвона, а также предлагается продолжить его, либо начать новый.

4.1.4 Страница вывода ошибок

В интерфейс добавлена страница, которая будет служить для отображения различного рода ошибок (рисунок 10). На ней содержится описание произошедшей ошибки, а также кнопка «назад», возвращающая пользователя на страницу выбора типа прозвона.

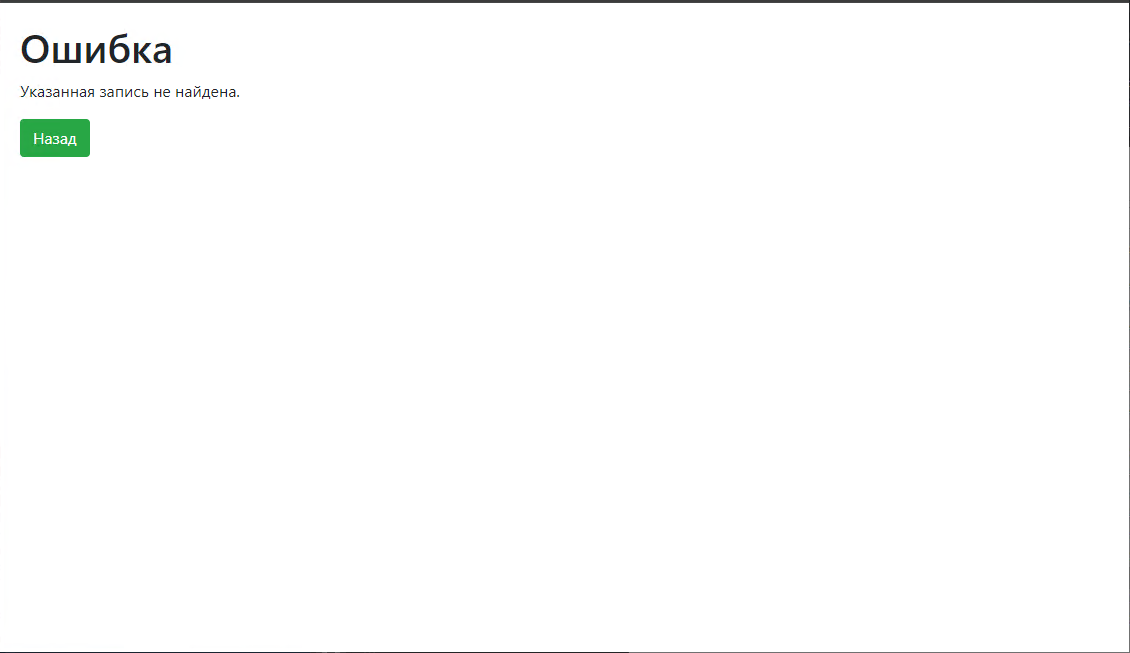


Рисунок 10 – Страница отображения ошибок

В частности, на данной странице отображается уведомление о некорректном ID аудиоролика.

4.1.5 Страница результатов прозвона

В интерфейс добавлена страница с информацией о результатах прозвона, которая открывается после его завершения (рисунок 11).

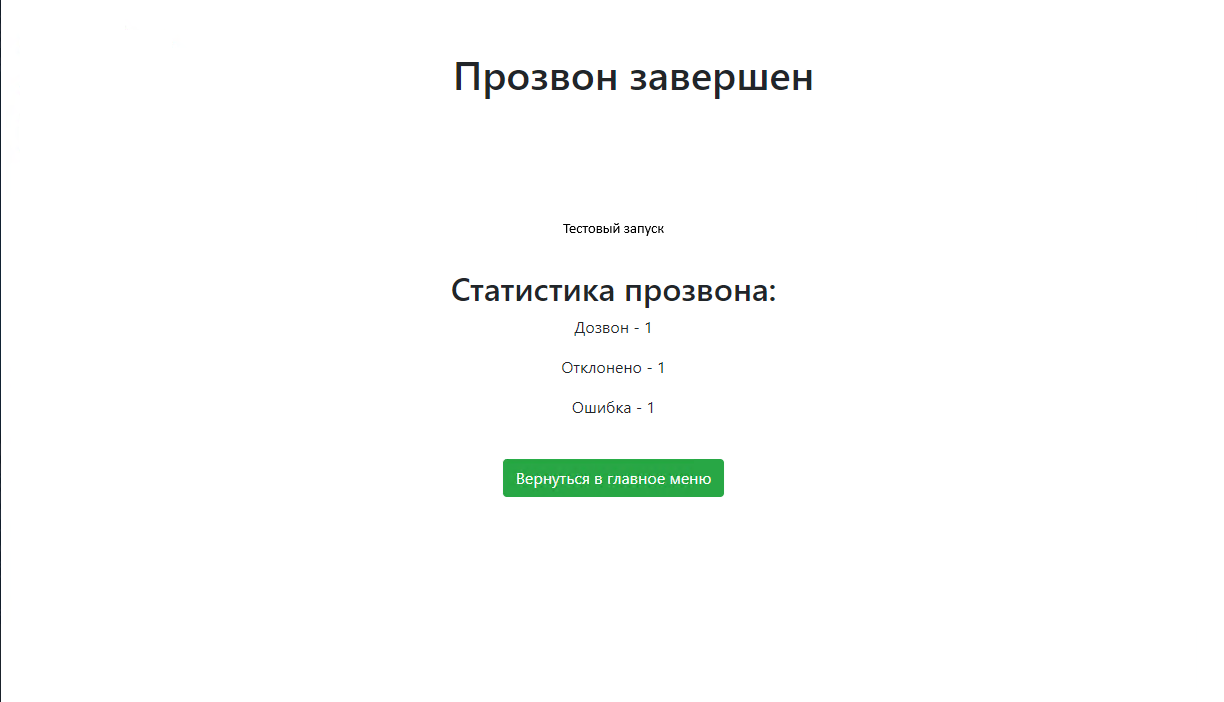


Рисунок 11 – Страница результатов прозвона

На данной странице отображается описание прозвона, а также его статистика: количество успешных, отклоненных и завершившихся с ошибкой звонков. Пользователь может вернуться на страницу выбора типа прозвона, нажав на соответствующую кнопку.

4.1.6 Страница диалоговых прозвонов

Так как на данный момент вне зависимости от выбора типа прозвона пользователь попадает на страницу информационных прозвонов, в интерфейс добавлена страница, предназначенная для создания диалоговых прозвонов (рисунок 12).

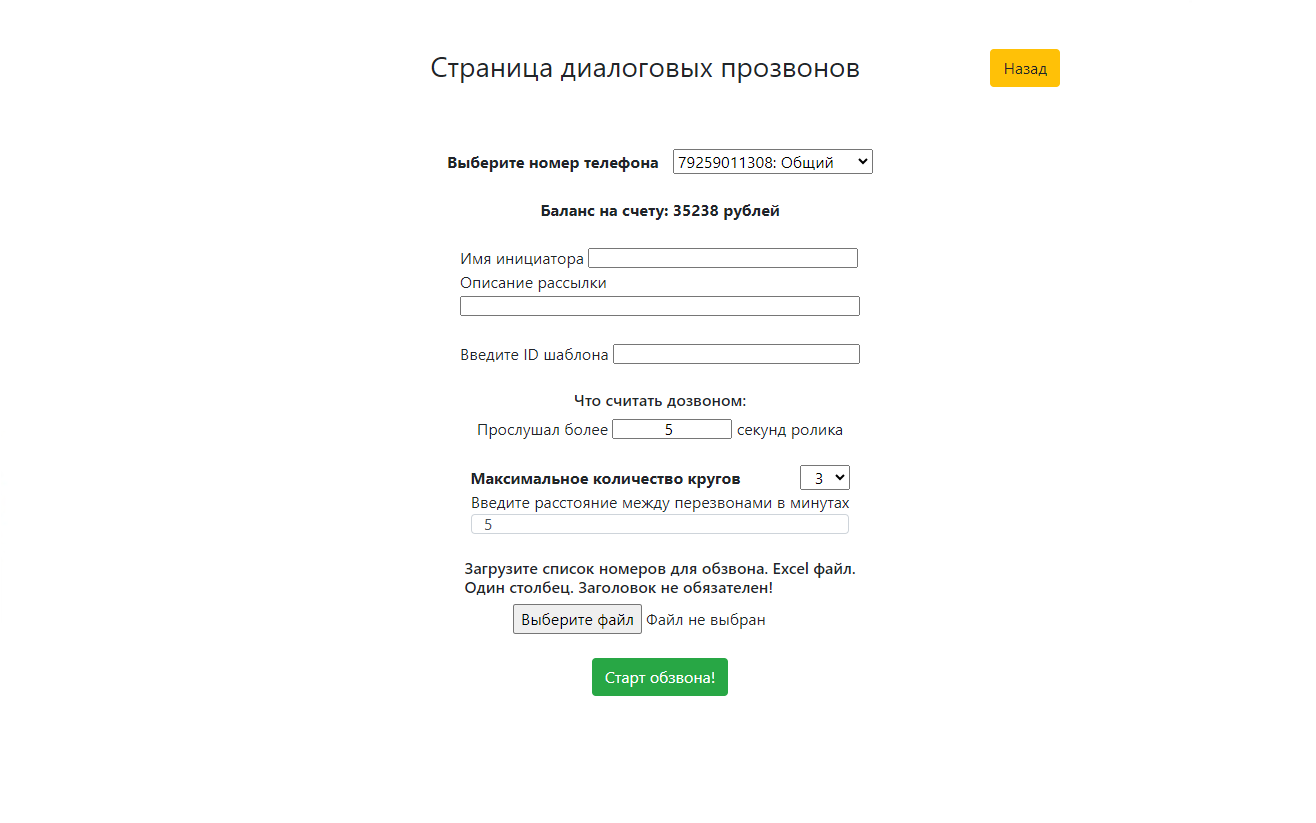


Рисунок 12 – Страница диалоговых прозвонов

Данная страница во многом использует тот же функционал, что и страница информационных прозвонов, но, вместо выбора ввода текста или ID аудиоролика, предлагается вести ID шаблона рассылки в сервисе «звонобот».

Wireframe-диаграмма интерфейса, получившегося в результате всех доработок, представлена в приложении Б.

4.2 Рефакторинг кода бэкенда

В ходе рефакторинга кода серверной части в первую очередь были исправлены ошибки, нарушающие стандарт PEP 8, убраны закомментированные строки кода, а параметры конфигурации flask-приложения из кода вынесены в конфигурационный файл (.yaml). Затем, были внесены изменения в структуру проекта. Итоговая структура представлена на рисунке 13.

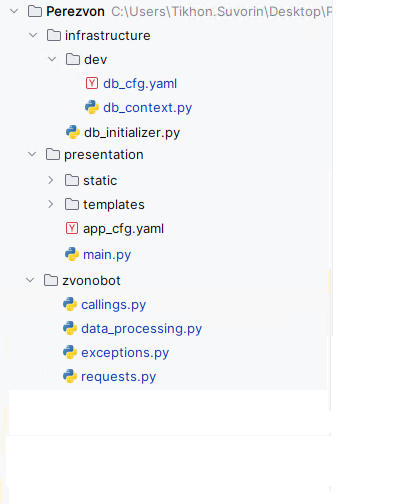


Рисунок 13 – Структура проекта после рефакторинга

Функционал по работе с базой данных был вынесен в уровень infrastructure. Модуль db\_initializer предназначен для получения объектов pyodbc connection и sqlachemy engine. В папке «dev» располагается модуль контекста, содержащий функции для работы с базой данных «Dev», а также конфигурационный файл с параметрами этой базы данных.

В папке zvonobot находятся файлы, относящиеся к уровню взаимодействия с внешним API сервиса «звонобот». Здесь содержаться модули, содержащие специфичные для работы с сервисом искючения, позволяющие осуществлять запросы к сервису, обрабатывать полученные данные. Модуль zvonobot\_callings.py содержит основную функцию launch\_calling(), позволяющую запустить рассылку, передав в неё словарь с параметрами и датафрейм с номерами телефонов.

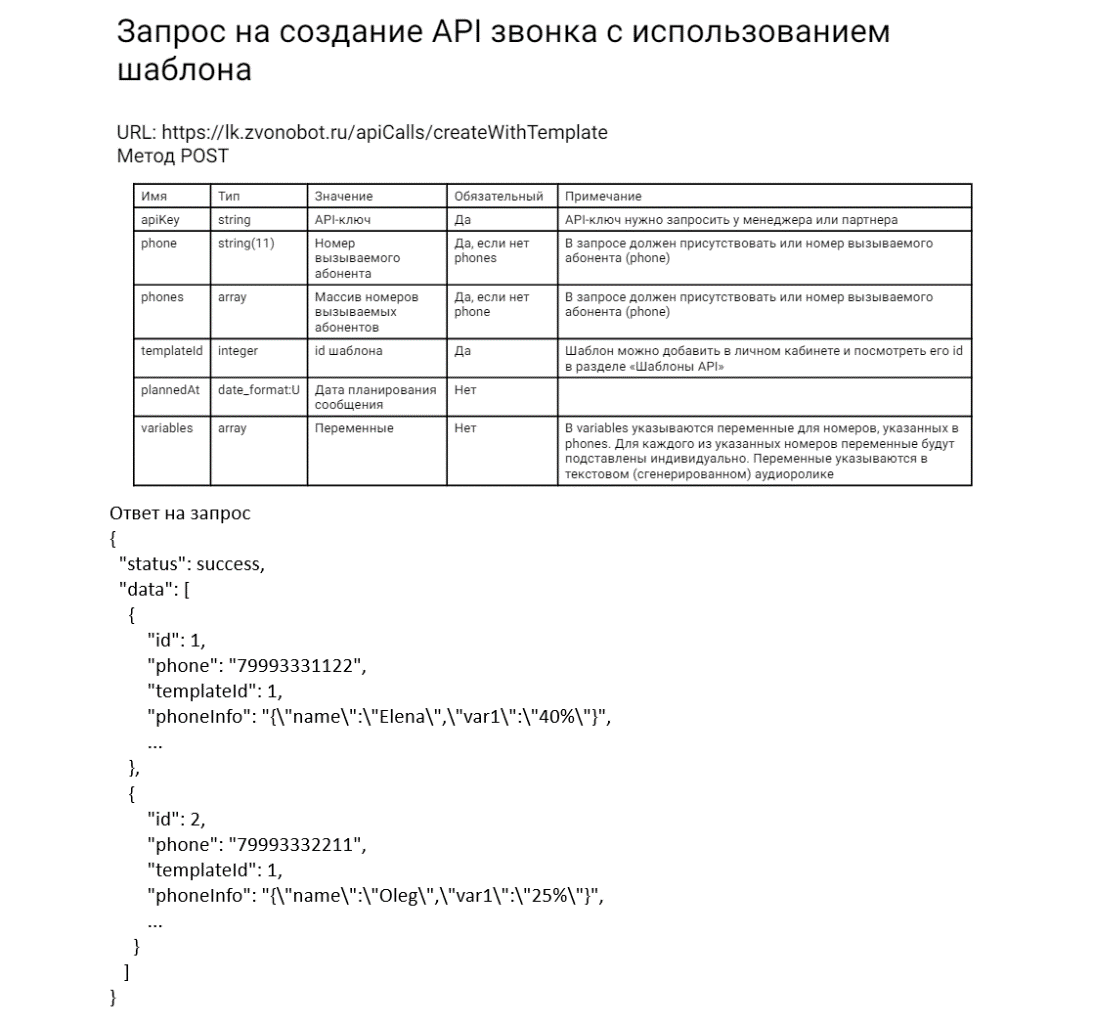
Уровень presentation – содержит само веб-приложение на flask. В папке templates хранятся html-страницы, в static – файлы стилей (.css) и скприптов (.js). В main.py содержаться контроллеры, принимающие и обрабатывающие запросы.

В результате рефакторинга код проекта стал более структурирован, что в будущем позволит удобнее вносить в него изменения и доработки.

4.3 Изучение части API-функционала сервиса «звонобот», отвечающей за диалоговые прозвоны

В ходе изучения официальной документации API сервиса «звонобот», с которой можно ознакомится в личном кабинете пользователя, было выяснено, что для управления диалоговыми прозвонами понадобится использовать 3 конечных точки.

1. https://lk.zvonobot.ru/apiCalls/createWithTemplate – используется для создания звонка с использованием шаблона (описание запроса на рисунке 14).

Рисунок 14 – Описание запроса на создание звонка с использованием шаблона

1. https://lk.zvonobot.ru/apiCalls/getWithTemplates – используется для получения информации о звонке (описание запроса на рисунке 15).

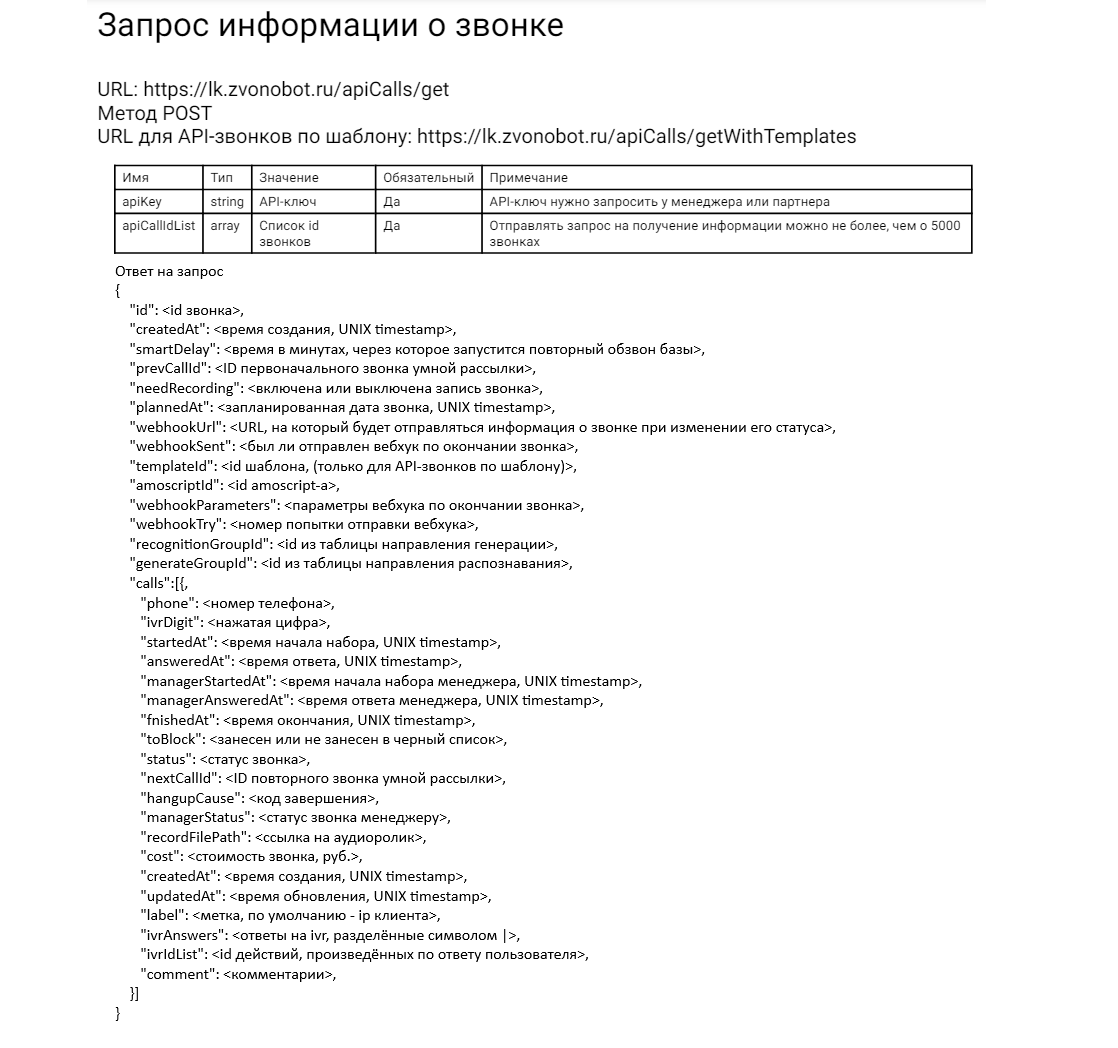


Рисунок 15 – Описание запроса на информацию о звонке с использованием шаблона

1. https://lk.zvonobot.ru/apiCalls/getTemplates – используется для получения списка шаблонов, созданных в личном кабинете (описание запроса на рисунке 16).

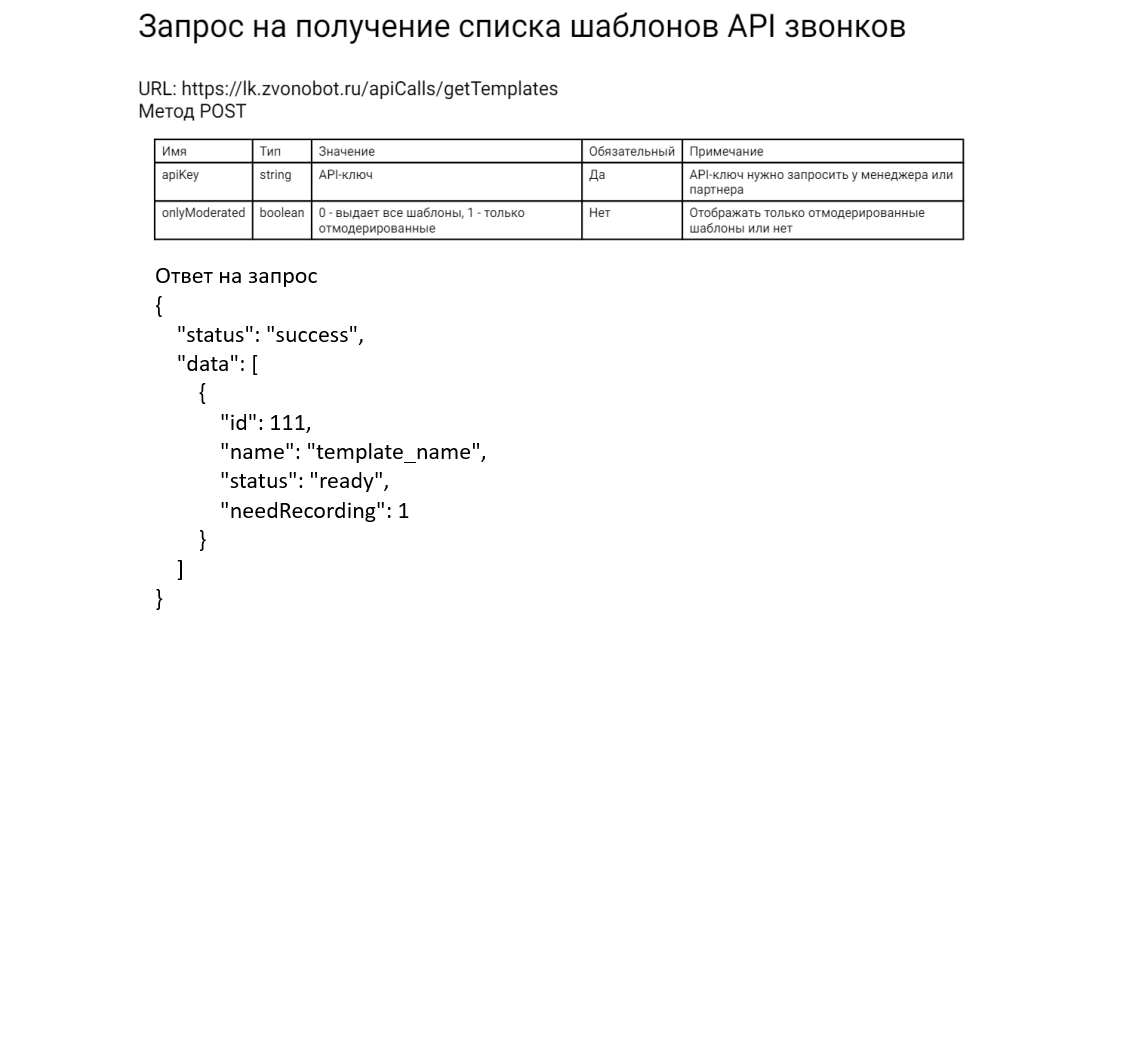


Рисунок 16 – Описание запроса на список шаблонов

Полученной информации о функционале API сервиса будет достаточно, чтобы внедрить возможность создания диалоговых прозвонов в разрабатываемый программный модуль.

4.4 Анализ решений в области синтеза речи

Синтез речи — это процесс преобразования текста в речь с использованием компьютерных программ. Существует несколько подходов к синтезу речи, таких как использование естественного языка (Natural Language Processing, NLP) и глубокого обучения (Deep Learning), или использование конкатенативного синтеза, который основан на записи отдельных фонем или звуков и их последующем объединении в слова и фразы.

На сегодняшний день существует множество сервисов в области синтеза речи, предлагающих различные функции и возможности. Например, сервис «звонобот» предлагает собственные возможности по синтезу речи. Однако, для этой цели было решено выбрать сервис Jay Copilot, из-за более низкой стоимости озвучки.

Jay Copilot – это умный помощник для бизнеса, сотрудников компаний и индивидуальных пользователей. В интерфейсе все сервисы представлены в виде удобных приложений, позволяющих решать конкретные бизнес-задачи. В основе приложений лежат как лучшие мировые нейросетевые модели, так и собственная разработка Just AI – большая языковая модель JustGPT. Интерфейс сервиса «диктор» в Jay Copilot представлен на рисунке 17, он понятен и прост в использовании. Для озвучки текста его достаточно напечатать и отправить в виде сообщения, при этом предоставляется возможность выбрать подходящий голос для озвучки из множества доступных.

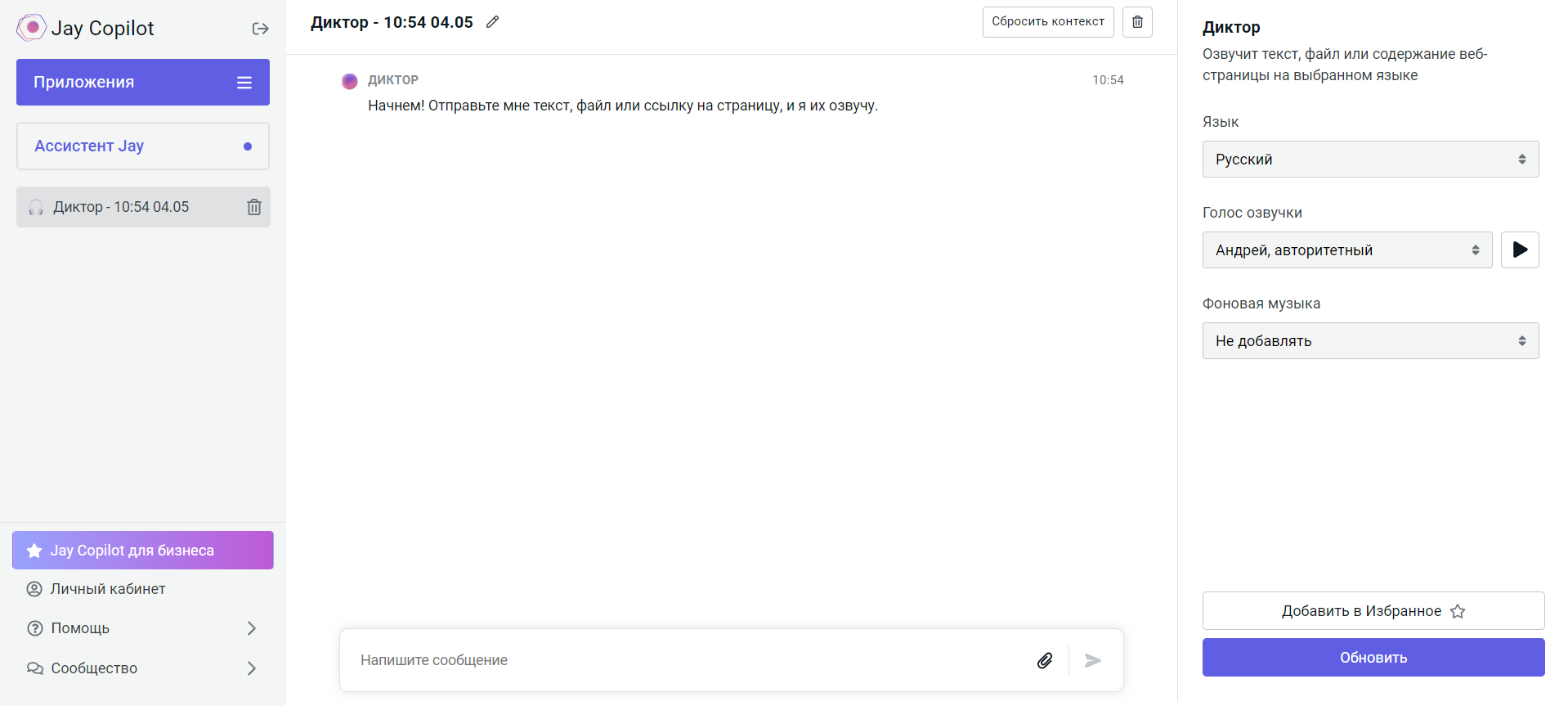


Рисунок 17 – Интерфейс сервиса «диктор» Jay Copilot

4.5 Интеграция диалоговых прозвонов в программный модуль

Согласно требованиям заказчика (управляющей компании «Эксплуатация Главстрой СПб»), в разрабатываемом сервисе необходимо реализовать 3 сценария диалоговых прозвонов, которые будут представлены в качестве отдельных модулей для последующего запуска на сервере по расписанию, с установленной дискретностью:

* Сценарий изучения клиентской лояльности (NPS): необходим для изучения отношения клиентов к компании.
* Сценарий изучения клиентской удовлетворенности (CSAT): предназначен для изучения удовлетворенности клиентов, обращавшихся ранее в диспетчерскую службу компании.
* Сценарий изучения удовлетворенности жильцов администратором дома (ADMIN): служит для оценки качества работы администраторов домов.

Реализация данных сценариев в сервисе будет проводится в 3 этапа: в первую очередь необходимо создать шаблоны рассылок в сервисе «звонобот», затем реализовать структуру хранения новых данных, получаемых по результатам выполнения рассылок, и разработать сами скрипты для запуска рассылок, интегрировав их в разрабатываемый сервис.

4.5.1 Создание шаблонов рассылок в сервисе «звонобот»

Первый – сценарий изучения клиентской лояльности (NPS – Net Promoter Score). Для этого сценария был создан шаблон рассылки в сервисе «Звонобот» (рисунок 18). После того, как клиент отвечает на звонок, ему проигрывается основной аудиоролик: «Добрый день. Оцените, пожалуйста, работу управляющей компании Эксплуатация Главстрой Санкт-Петербург по шкале от 0 до 10. Насколько вероятно, что Вы порекомендуете нашу компанию друзьям и коллегам, где 10 – это максимум, обязательно порекомендую. Благодарим за ответ.». Клиент ставит оценку голосом, после чего проигрывается завершающий аудиоролик «благодарим за ответ».

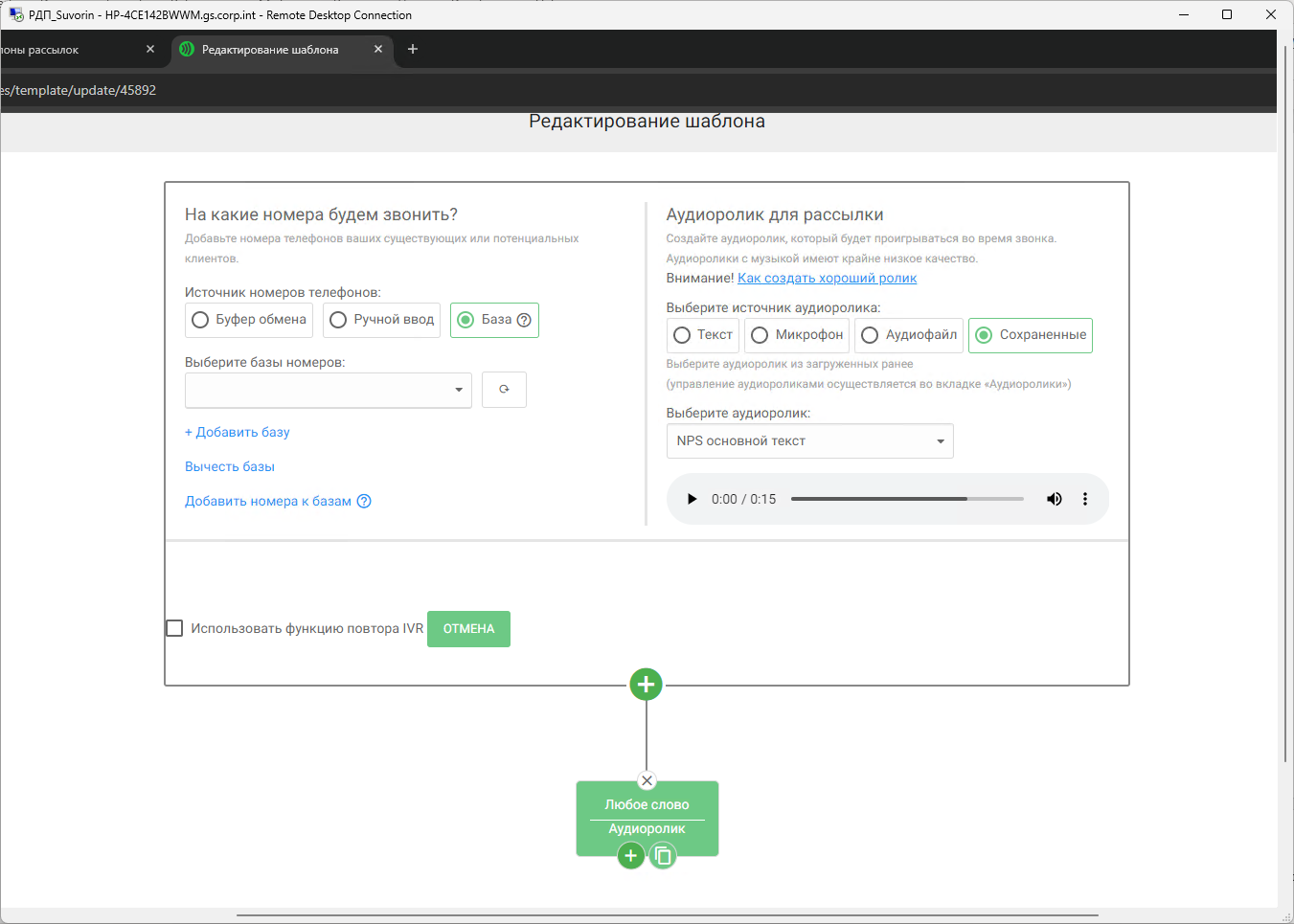


Рисунок 18 – Шаблон сценария NPS

Логика двух следующих сценариев (изучения клиентской удовлетворенности и удовлетворенности администратором дома), шаблон которых представлен на рисунке 19, в целом схожа: после ответа на звонок клиент в первую очередь слышит вопрос. В случае CSAT это: «Добрый день, Вы обращались в диспетчерскую службу управляющей компании Эксплуатация Главстрой Санкт-Петербург. Оцените, пожалуйста, качество работы диспетчера по шкале от 0 до 10, где 10 – это максимальная оценка его работы», а в случае ADMIN – «Добрый день. Оцените, пожалуйста, качество работы администратора вашего дома по шкале от 0 до 10, где 10 – это максимальная оценка его работы».

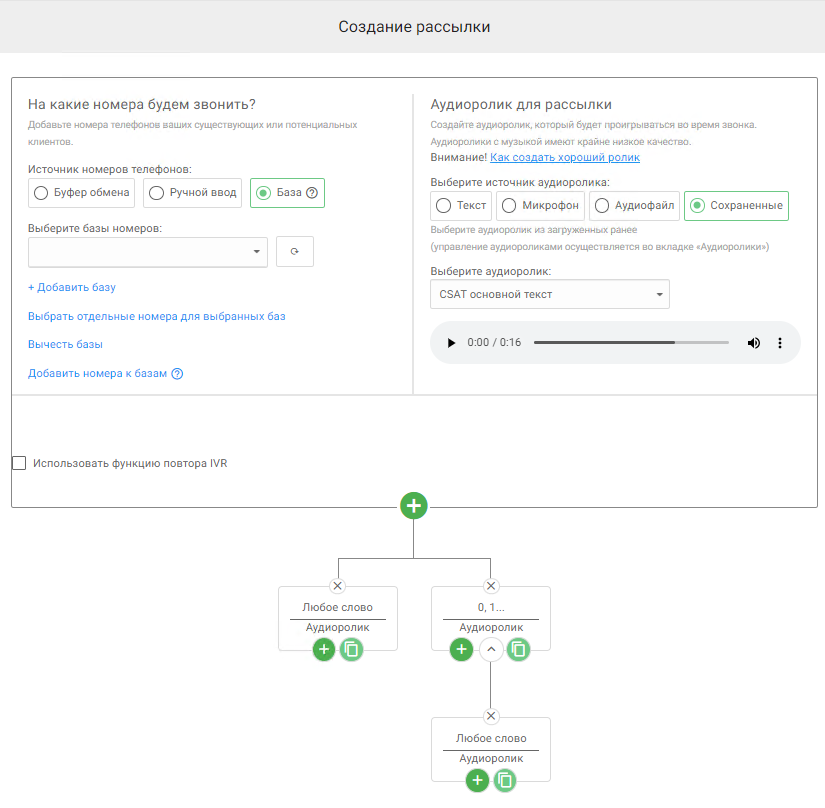


Рисунок 19 – Шаблон сценариев CSAT и ADMIN

После того, как клиент ставит оценку голосом, в зависимости от ней проигрывается либо аудиоролик «благодарим за оценку», в случае если оценка от 5 до 10, либо задаётся открытый вопрос «что повлияло на вашу оценку», если оценка от 0 до 4. В этом случае пользователю даётся 10 секунд для изложения своего мнения, после чего проигрывается аудиоролик «благодарим за ответ». Так как в данных сценариях ожидается получение развернутой обратной связи клиентов, включена функция сохранения записей разговоров для последующего их анализа.

4.5.2 Проектирование и реализация структуры хранения новых данных

Для хранения новых данных в базу данных было добавлено несколько новых таблиц (рисунок 20). Для результатов опроса по каждому из сценариев созданы таблицы Zvonobot\_NPS, Zvonobot\_CSAT и Zvonobot\_ADMIN. В каждой из них присутствуют атрибуты GlobalCallingID (глобальный индекс рассылки), PhoneNumber (телефон клиента), Score (выставленная оценка) и InsDate (время вставки). При этом первые три атрибута объединены в составной первичный ключ.

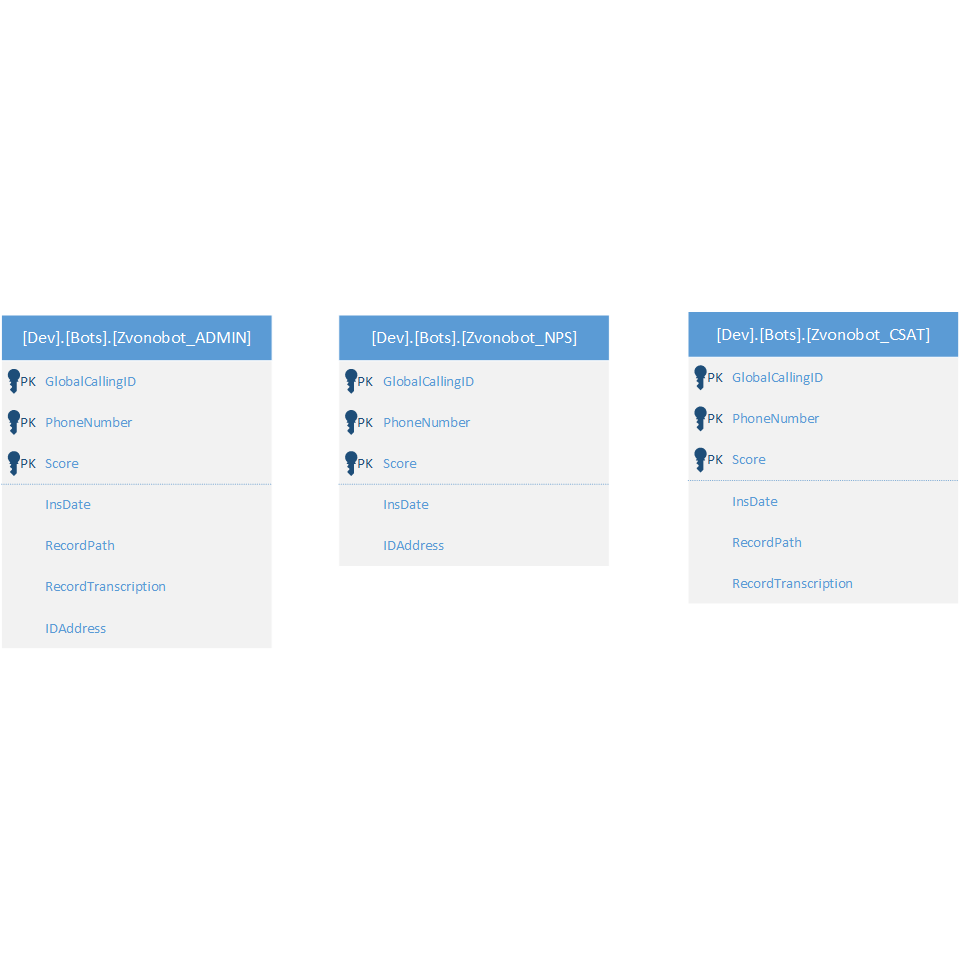


Рисунок 20 – Структура добавленных таблиц

Так как в сценариях ADMIN и CSAT ожидается получение развернутой обратной связи от клиентов, в соответствующие таблицы добавлены также атрибуты RecordTranscription (транскрибация ответа на открытый вопрос), и RecordPath (путь, по которому сохранена запись разговора).

Помимо этого, в сценариях ADMIN и NPS требуется знать, с какого адреса поступила оценка, поэтому в соответствующие таблицы также добавлен столбец IDAddress.

4.5.3 Реализация сценариев в проекте

Для реализации сценариев в проекте в первую очередь в модуль zvonobot\_requests были интегрированы функции по взаимодействию с ранее изученными конечными точками API «звонобот». Это функции для создания рассылки с шаблоном (run\_calling\_with\_template()) и получения информации о состоянии рассылки с шаблоном (check\_calling\_state\_with\_template()).

Далее, в структуру проекта добавлена папка scripts, в которой заранее подготовлены папка common (с модулями для общего использования всеми сценариями) и папки для трех разрабатываемых сценариев. Структура директории scripts представлена на рисунке 21.

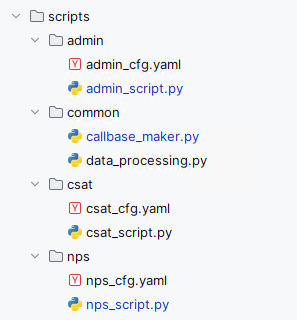


Рисунок 21 – Структура директории scripts

В каждой из папок сценариев были созданы конфигурационные файлы (.yaml), которые позволяют настраивать параметры прозвона. На рисунке 22 представлена конфигурация сценария CSAT. Конфигурации остальных сценариев схожи, за исключением параметров answer\_start\_time и record\_dir, которые отсутствуют в конфигурации NPS.

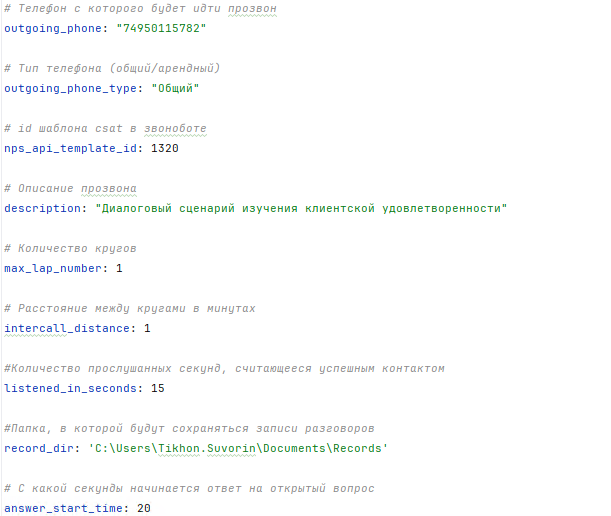


Рисунок 22 – Конфигурация сценария CSAT

В папку common были добавлены модули для получения базы номеров для прозвона (callbase\_maker) и обработки данных (callbase\_maker).

Завершающим этапом была разработка самих сценариев. Они работают по схожей логике: получают базу номеров для прозвона, затем формируют датафреймы с параметрами рассылки и номерами телефонов, после чего запускают прозвон. Когда тот завершается, оценки клиентов собираются в датафрейм и вносятся в соответствующую таблицу. Сценарии CSAT и ADMIN также записывают в таблицы транскрибации развернутых ответов и сохраняют аудиозаписи звонков по указанному пути.

**Тема 5. Разработка эксплуатационной документации**

5.1 Разработка документа «текст программы»

Разработка документа «текст программы» необходима для предоставления подробного описания исходного кода программы, его структуры, логики работы и алгоритмов, что обеспечивает полное понимание программы разработчиками, тестировщиками и другими участниками проекта.

После завершения разработки программного модуля, в соответствии с ГОСТ 19.401-78 был составлен документ «текст программы», в котором код на языке Python был структурирован и описан. Текст программы представлен в приложении В.

5.2 Разработка документа «руководство пользователя»

Руководство пользователя предназначен для конечных пользователей разрабатываемой системы и описывает, как эффективно использовать программное обеспечение, предоставляя подробные инструкции, советы и рекомендации.

По завершении разработки программного модуля было составлено руководство пользователя, с которым перед использованием системы смогут ознакомиться сотрудники компании. В руководстве описаны назначение и условия применения системы, подготовка к работе, описание операций, а также возможные ошибки и рекомендации по их устранению. Документ «руководство пользователя» представлен в приложении Г.

**Тема 6. Проведение испытаний, отладка и внедрение программного продукта на предприятии**

4.1 Выбор стратегии тестирования

В процессе разработки программного модуля ключевым этапом являлось обеспечение его правильной работы. Для этого была продумана подходящая стратегия, учитывающая особенности проекта, его цели и контекст разработки.

Исходя из того, что проект содержит большое количество функционала, отвечающего за обработку данных, наиболее важным является проведение модульного (unit) тестирования. Данный тип тестирования проверяет работу каждого отдельного модуля системы. Модульные тесты помогут удостовериться в правильности выполнения логики программы, а также предотвратить ошибки при дальнейшей доработке системы и добавлении нового функционала.

4.2 Разработка тестов

Для разработки модульных тестов в проект была добавлена папка tests. В данной папке были созданы модули тестовых сценариев (рисунок 23).

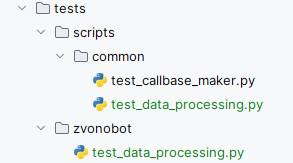


Рисунок 23 – Структура директории tests

Для проведения модульного тестирования используется модуль unittest. Пример тестирующего модуля показан на рисунке 24.



Рисунок 24 – Класс TestCallbaseMaker

На рисунке выше частично показан класс TestCallbaseMaker, тестирующий функции модуля callbase\_maker. Для написания тестов используется паттерн ААА (Arrange-Act-Assert). Согласно данному паттерну, тестовый сценарий разбивается на 3 части: подготовку данных, выполнение действия и проверку результата. Таким образом были протестированы функции модулей callbase\_maker, data\_processing (scripts) и data\_processing (zvonobot). Результат успешного выполнения тестов, представленный на рисунке 25, свидетельствует о соответствии разрабатываемого программного продукта установленным требованиям, а также готовности программы к внедрению и использованию.

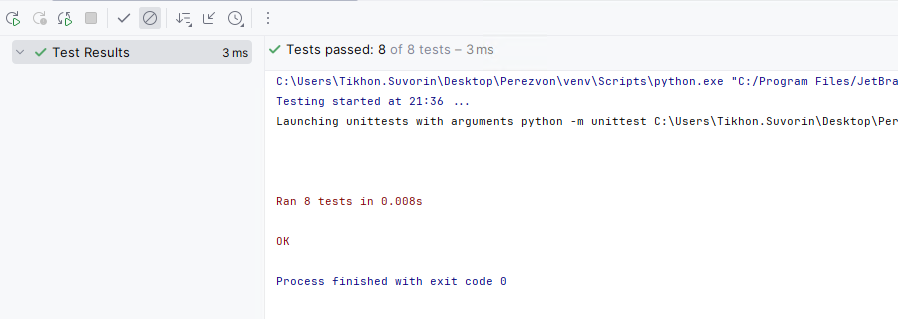


Рисунок 25 – Результат успешного выполнения тестов

4.3 Внедрение программного продукта на предприятии

Для внедрения диалоговых сценариев необходимо из скриптов, написанных на языке Python, создать исполняемые файлы, которые в дальнейшем будут с определенной дискретностью запускаться через планировщик задач на сервере.

Для сборки из файлов .py файлов .exe была использована библиотека PyInstaller. PyInstaller собирает в один пакет Python-приложение и все необходимые ему библиотеки следующим образом:

1. Считывает файл скрипта.
2. Анализирует код для выявления всех зависимостей, необходимых для работы.
3. Создает файл spec, который содержит название скрипта, библиотеки-зависимости, любые файлы, включая те параметры, которые были переданы в команду PyInstaller.
4. Собирает копии всех библиотек и файлов вместе с активным интерпретатором Python.
5. Создает папку BUILD в папке со скриптом и записывает логи вместе с рабочими файлами в BUILD.
6. Создает папку DIST в папке со скриптом, если она еще не существует.
7. Записывает все необходимые файлы вместе со скриптом или в одну папку, или в один исполняемый файл.

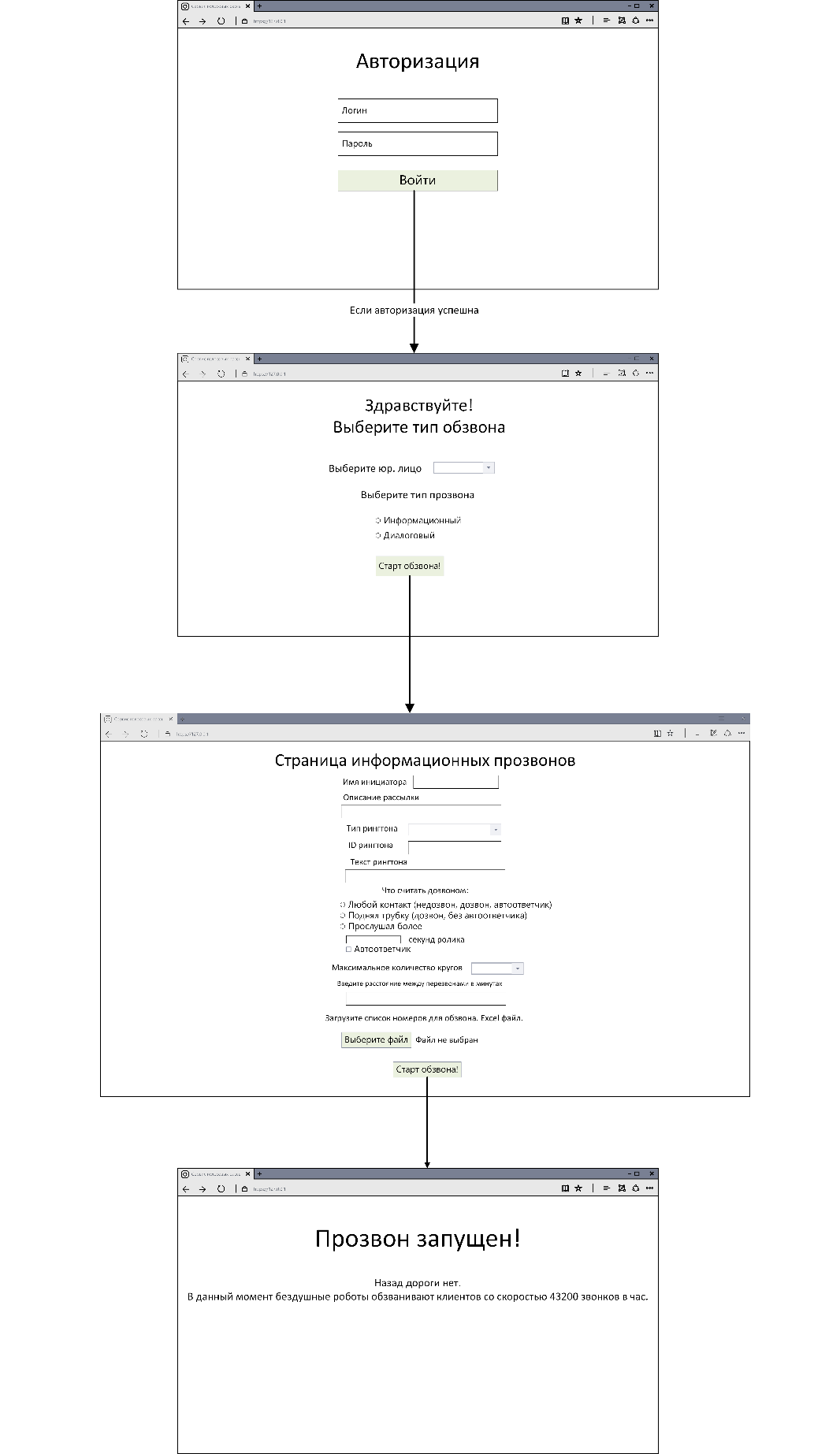
После сборки исполняемых файлов, они были добавлены в планировщик задач сервера с указанием нужной дискретности.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Чистый код: создание, анализ и рефакторинг, / Роберт Мартин – 2020 г. - 150 с.
2. Приёмы объектно-ориентированного проектирования, / Эрих Гамма, Ральф Джонсон – 2019 г. -213с.
3. Грокаем алгоритмы / Адитья Бхаргава – 2019 г. - 378с.
4. Изучаем SQL / Алан Болье – 2020 г. - 145 с.
5. SQL. Сборник рецептов / Энтони Молинаро – 2019 г. - 145 с.
6. Совершенный алгоритм. Основы / Тим Рафгарден – 2020 г. - 240 с.
7. Автоматное программирование / Надежда Поликарпова – 2019 г. - 140 с.
8. SQL: Руководство по изучению языка / Крис Фиайли – 2020 г. - 190 с.
9. Чистая архитектура / Роберт Мартин – 2020 г. - 290 с.
10. Разработка интеллектуальных систем / Джефф Хултен – 2021 г. - 303 с.
11. Структура компьютерных программ / Сергей Васильев – 2019 г. - 120с.
12. Искусство гибкой разработки ПО / Джефф Паттон – 2019 г. - 210 с.
13. Алгоритмы. Построение и анализ / Томас Кармен – 2021 г. - 291с.
14. Git для профессионального программиста / Бен Штрауб – 2022 г. - 156 с.
15. Изучаем Python, 5-е издание / Лутц Марк – 2019 г. – 1280 с.
16. Программирование на языке высокого уровня Python: учеб. пособие для прикладного бакалавриата / Д. Ю. Федоров. — 2019 г. — 161 с.

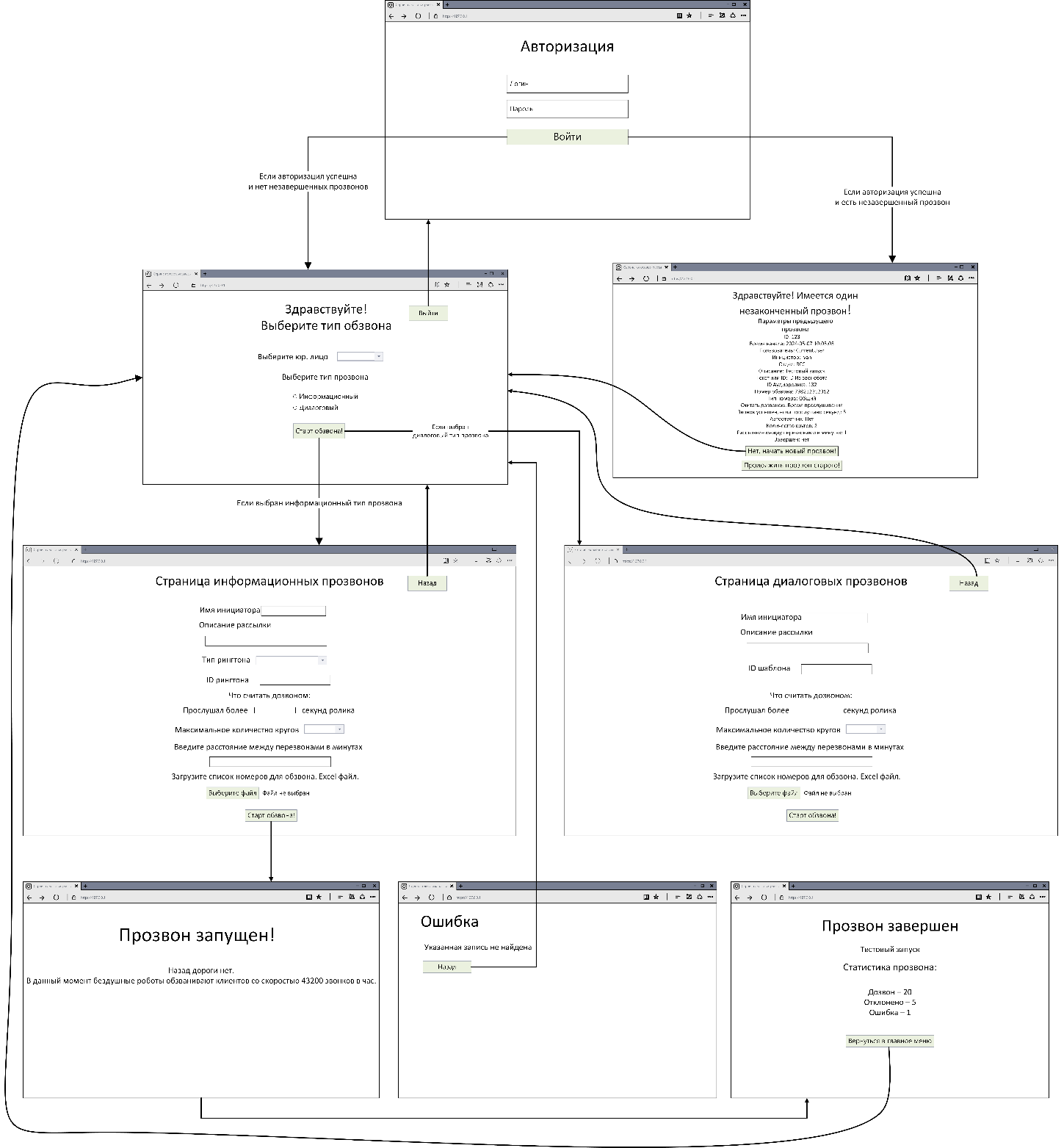
**ПРИЛОЖЕНИЕ А**

**Wireframe-диаграмма веб-интерфейса до доработок**

****

**ПРИЛОЖЕНИЕ Б**

**Wireframe-диаграмма интерфейса после доработок**

****

**ПРИЛОЖЕНИЕ В**

**Текст программы**

1. Уровень взаимодействия с базой данных (infrastructure)

1.1 Класс DbIInitializer

|  |  |
| --- | --- |
| class DbInitializer:  def \_\_init\_\_(self, db\_cfg\_path):  self.\_\_connection\_string = self.\_\_get\_connection\_string(db\_cfg\_path)  def \_\_get\_connection\_string(self, db\_cfg\_path):  with open(db\_cfg\_path, 'r') as f:  db\_config = yaml.load(f, Loader=yaml.FullLoader)  db\_connector = f"DRIVER={db\_config['driver']};\  SERVER={db\_config['server']}; \  DATABASE={db\_config['database\_name']};\  UID={db\_config['user\_id']};\  PWD={db\_config['password']}"  return db\_connector  def get\_connection(self):  conn = pyodbc.connect(self.\_\_connection\_string)  return conn  def get\_engine(self):  params = urllib.parse.quote\_plus(self.\_\_connection\_string)  engine = sqlalchemy.create\_engine("mssql+pyodbc:///?odbc\_connect=%s" % params)  return engine | Класс DbInitializer. В конструкторе принимает в качестве аргумента путь к файлу конфигурации базы данных. Имеет функции для получения объектов pyodbc connection и sqlalchemy engine. |

1.2 Модуль dev\_db\_context

|  |  |
| --- | --- |
| def insert\_calling\_params(df\_init):  with \_engine.begin() as connection:  df\_init.to\_sql(  name='Zvonobot\_init', con=connection, index=False,  if\_exists='append', schema='Bots') | Функция для внесения параметров прозвона в таблицу Zvonobot\_init |
| def get\_calling\_params(global\_calling\_id):  sql = f"""  SELECT \*  FROM [DEV].[Bots].[Zvonobot\_init]  WHERE [ID] = '{global\_calling\_id}'  """  calling\_params = pd.read\_sql\_query(sql, \_engine)  return calling\_params | Функция для получения параметроп прозвона из Zvonobot\_init |
| def get\_last\_user\_calling\_params(login):  sql = f"""  SELECT TOP (1) \*  FROM [DEV].[Bots].[Zvonobot\_init]  WHERE [CurrentUser] = '{login}'  ORDER BY [ID] DESC  """  calling\_params = pd.read\_sql\_query(sql, \_engine)  return calling\_params | Функция для получения параметров последнего прозвона пользователя |
| def get\_last\_calling\_id():  sql = f"""SELECT MAX([ID])  FROM [DEV].[Bots].[Zvonobot\_init]  """  global\_calling\_id = \_cursor.execute(sql).fetchval()  return global\_calling\_id | Функция для получения последнего global\_calling\_id из Zvonobot\_init |
| def finish\_calling(global\_calling\_id):  sql = f"""  UPDATE [DEV].[Bots].[Zvonobot\_init]  SET Finished = 1  WHERE ID = {global\_calling\_id}  """  \_cursor.execute(sql)  \_conn.commit() | Функция для установки флага Finished для прозвона по global\_calling\_id |
| def finish\_last\_user\_calling(login):  sql = f'''  UPDATE Bots.Zvonobot\_init  SET Finished = 1  WHERE ID = (SELECT MAX(ID)  FROM Bots.Zvonobot\_init  WHERE CurrentUser = '{login}')  '''  \_cursor.execute(sql)  \_conn.commit() | Функция для установки флага Finished для последнего прозвона пользователя |
| def insert\_phones\_to\_call(df\_phones):  with \_engine.begin() as connection:  df\_phones.to\_sql(  name='Zvonobot\_Phones\_To\_Call', con=connection, index=False,  if\_exists='append', schema='Bots') | Функция для вносения телефонов для прозвона в Zvonobot\_Phones\_To\_Call |
| def get\_phones\_to\_call(global\_calling\_id):  sql = f"""  SELECT [Numbers]  FROM [DEV].[Bots].[Zvonobot\_Phones\_To\_Call]  WHERE [GlobalCallingID] = '{global\_calling\_id}'  """  phones = [row.Numbers for row in \_cursor.execute(sql).fetchall()]  return phones | Функция для извлечения телефонов для прозвона из Zvonobot\_Phones\_To\_Call |
| def insert\_nps\_scores(scores\_df):  with \_engine.begin() as connection:  scores\_df.to\_sql(  name='Zvonobot\_NPS', con=connection, index=False,  if\_exists='append', schema='Bots') | Функция для внесения оценок клиентов после прозвона по сценарию NPS в Zvonobot\_NPS |
| def get\_nps\_scores():  sql = f"""  SELECT \*  FROM [DEV].[Bots].[Zvonobot\_NPS]  """  nps\_scores = pd.read\_sql\_query(sql, \_engine)  return nps\_scores | Функция для извлечения оценок клиентов из Zvonobot\_NPS |

|  |  |
| --- | --- |
| def insert\_csat\_scores(scores\_df):  with \_engine.begin() as connection:  scores\_df.to\_sql(  name='Zvonobot\_CSAT', con=connection, index=False,  if\_exists='append', schema='Bots') | Функция для внесения оценок клиентов после прозвона по сценарию CSAT в Zvonobot\_CSAT |
| def get\_csat\_scores():  sql = f"""  SELECT \*  FROM [DEV].[Bots].[Zvonobot\_CSAT]  """  csat\_scores = pd.read\_sql\_query(sql, \_engine)  return csat\_scores | Функция для извлечения оценок клиентов из Zvonobot\_ CSAT |
| def insert\_admin\_scores(scores\_df):  with \_engine.begin() as connection:  scores\_df.to\_sql(  name='Zvonobot\_ADMIN', con=connection, index=False,  if\_exists='append', schema='Bots') | Функция для внесения оценок клиентов после прозвона по сценарию ADMIN в Zvonobot\_ADMIN |
| def get\_admin\_scores():  sql = f"""  SELECT \*  FROM [DEV].[Bots].[Zvonobot\_ADMIN]  """  admin\_scores = pd.read\_sql\_query(sql, \_engine)  return admin\_scores | Функция для извлечения оценок клиентов из Zvonobot\_ADMIN |
| def get\_api\_key(department):  sql = f"""  SELECT [API\_key]  FROM [DEV].[Bots].[Zvonobot\_APIs]  WHERE [Department] = '{department}'  """  api\_key = \_cursor.execute(sql).fetchval()  return api\_key | Функция для получения ключа от API звонобота по наименованию департамента |
| def insert\_calling\_results(calling\_results\_df):  with \_engine.begin() as connection:  calling\_results\_df.to\_sql(  name='Zvonobot\_laps', con=connection, index=False,  if\_exists='append', schema='Bots') | Функция для внесения информации о звонках в Zvonobot\_laps |

|  |  |
| --- | --- |
| def get\_calling\_results(global\_calling\_id):  sql = f"""  SELECT [GlobalCallingID], [LapNumber], [PhoneNumber], [HangupCause], [Status], [CallDuration]  FROM [DEV].[Bots].[Zvonobot\_laps]  WHERE [GlobalCallingID] = {global\_calling\_id}  """  results = pd.read\_sql\_query(sql, \_engine)  return results | Функция для извлечения информации о звонках из Zvonobot\_laps |
| def get\_last\_lap\_number(global\_calling\_id):  sql = f"""  SELECT MAX(LapNumber)  FROM [Dev].[Bots].[Zvonobot\_laps]  WHERE GlobalCallingID = {global\_calling\_id}  """  last\_lap\_number = \_cursor.execute(sql).fetchval()  return 1 if last\_lap\_number is None else last\_lap\_number | Функция для получения номера последнего круга прозвона из [Dev].[Bots].[Zvonobot\_laps] |
| def get\_department\_keys(login):  sql = f"""  SELECT API\_list.[API\_key], API\_list.[Department]  FROM [DEV].[Bots].[Zvonobot\_APIs\_rights] as API\_rights  LEFT JOIN [DEV].[Bots].[Zvonobot\_APIs] as API\_list  ON API\_rights.API\_key\_id = API\_list.ID  WHERE API\_rights.[Employee] = '{login}'  """  departments\_keys = pd.read\_sql\_query(sql, \_engine)  return departments\_keys | Функция для получения словаря из названий подразделений и ключей от API звонобота |

1.3 Модуль dwh\_db\_context

|  |  |
| --- | --- |
| def get\_clients(project, quarter=None, queue=None):  sql = f'''  SELECT  PersonalAccounts.ID, PersonalAccounts.ID\_Address,  PersonalAccounts.ID\_PhoneNumber,  PhoneNumbers.PhoneNumber,  AddressObject.IDAdress\_C300, AddressObject.[Quarter],  AddressObject.CRM\_Object,  [Address].IDAdress, [Address].Project, [Address].Street\_Computed,  [Address].House  FROM  [DWH].[EGS].[PersonalAccounts] AS PersonalAccounts LEFT JOIN  [DWH].[Dictionary].[PersonalAccounts\_PhoneNumber] AS  PhoneNumbers  ON PersonalAccounts.ID\_PhoneNumber =  PhoneNumbers.IDPhoneNumber LEFT JOIN  [DWH].[tblCalculated].  [dict\_СопоставлениеАдресаC300\_и\_объектаCRM]  AS AddressObject  ON PersonalAccounts.ID\_Address = AddressObject.IDAdress\_C300  LEFT JOIN  [DWH].[Dictionary].[Address] AS [Address]  ON AddressObject.IDAdress\_C300 = [Address].IDAdress  WHERE  PhoneNumbers.PhoneNumber IS NOT NULL  AND [Address].Project = '{project}'  {f'AND [AddressObject].[Quarter] = {quarter}' if quarter is not None  else ''}  {f'AND [AddressObject].[Queue] = {queue}' if queue is not None else  ''}  ORDER BY  PersonalAccounts.ID'''  clients = pd.read\_sql\_query(sql, \_engine)  return clients | Получает данные о клиентах по ЖК, кварталу/очереди |

|  |  |
| --- | --- |
| def get\_dispatcher\_service\_calls():  sql = '''  WITH enriched\_cdr AS(  SELECT \*,  CASE  WHEN EXISTS (  SELECT 1 FROM [DWH].[Calls].[cdr] t2 WHERE t2.linkedid = [DWH].[Calls].[cdr].linkedid AND t2.dst = '400'  ) THEN 1 ELSE 0  END as call\_to\_egs -- add column that specifies whether the call was to EGS or not  FROM [DWH].[Calls].[cdr]  WHERE disposition = 'ANSWERED'  AND billsec >= 90  AND lastapp = 'Dial'  AND (cnam LIKE '[0-9][0-9][0-9]%' OR cnam LIKE '+7[0-9][0-9][0-9]%')  AND DATEDIFF(day, calldate, GETDATE()) <= 5  )  SELECT cnam, calldate  FROM enriched\_cdr  WHERE call\_to\_egs = 1  '''  calls = pd.read\_sql\_query(sql, \_engine)  return calls | Получает данные о звонках клиентов в диспетчерскую службу за последние 5 дней |

2. Уровень взаимодействия с внешним API (zvonobot)

2.1 Модуль exceptions

|  |  |
| --- | --- |
| class CallingRequestException(Exception):  pass  class StateRequestException(Exception):  pass | Классы исключений, специфичных для работы с API «звонобота». На данный момент не имеют в себе никакой логики или данных. Если возникнет необходимость, их можно дополнить. |

2.2 Модуль requests

|  |  |
| --- | --- |
| def \_post\_request(url, info):  response = \_session.post(  url=url,  data=json.dumps(info),  headers={  'content-type': 'application/json',  'accept': 'application/json'  }, verify=False  )  return response | Приватная функция для отправки post-запроса по указанному url. Предполагается использование только внутри модуля |
| def run\_calling(department, phone\_numbers, outgoing\_phone, ringtone\_id):  api\_key = dev\_db\_context.get\_api\_key(department)  url\_create\_calling = "https://lk.zvonobot.ru/apiCalls/create"  request = {  "apiKey": api\_key,  "phones": phone\_numbers,  "outgoingPhone": outgoing\_phone,  "robotsBlockEnabled": 1,  "record": {'id': ringtone\_id},  "plannedAtTimeZone": 1  }  response = \_post\_request(url\_create\_calling, request)  return response | Функция для запуска прозвона |
| def run\_calling\_with\_template(department, phone\_numbers, outgoing\_phone, template\_id):  api\_key = dev\_db\_context.get\_api\_key(department)  url\_create\_calling = "https://lk.zvonobot.ru/apiCalls/createWithTemplate"  request = {  "apiKey": api\_key,  "phones": phone\_numbers,  "templateId": template\_id,  "outgoingPhone": outgoing\_phone  }  response = \_post\_request(url\_create\_calling, request)  return response | Функция для запуска прозвона с использованием шаблона |
| def check\_calling\_state(department, call\_ids):  api\_key = dev\_db\_context.get\_api\_key(department)  url\_check\_calling\_state = "https://lk.zvonobot.ru/apiCalls/get"  request = {  "apiKey": api\_key,  "apiCallIdList": call\_ids  }  response = \_post\_request(url\_check\_calling\_state, request)  return response | Функция для получения информации о состоянии прозвона |
| def check\_calling\_state\_with\_template(department, call\_ids):  api\_key = dev\_db\_context.get\_api\_key(department)  url\_check\_calling\_state = "https://lk.zvonobot.ru/apiCalls/getWithTemplates"  request = {  "apiKey": api\_key,  "apiCallIdList": call\_ids  }  response = \_post\_request(url\_check\_calling\_state, request)  return response | Функция для получения информации о состоянии прозвона с шаблоном |
| def download\_record(url):  return requests.get(url).content | Функция для скачивания аудиоролика по указанному url |
| def get\_balance(department):  api\_key = dev\_db\_context.get\_api\_key(department)  url\_balance = "https://lk.zvonobot.ru/apiCalls/userInfo"  request = {  "apiKey": api\_key  }  response = \_post\_request(url\_balance, request)  return int(float(response.json()['data']['balance'])) | Функция для получения баланса на счету личного кабинета |
| def get\_outgoing\_phones(department):  api\_key = dev\_db\_context.get\_api\_key(department)  url\_phones = "https://lk.zvonobot.ru/apiCalls/getPhones"  request = {  "apiKey": api\_key,  "all": True  }  response = \_post\_request(url\_phones, request)  return response | Функция для получения исходящих номеров |

2.3 Модуль data\_processing

|  |  |
| --- | --- |
| def chunks(lst, n):  for i in range(0, len(lst), n):  yield lst[i:i + n] | Функция для разбиения списка на куски с размером n и получения генератора |
| def make\_state\_df(info\_request\_result\_data, calls\_df, listening\_seconds):  # Столбцы датафрейма  column\_names = [  "CallId", "StartDt", "FinishDt",  "LapInitTime", "Status", "CallDuration",  "HangupCause", "Cost"  ]  # Данные для формирования ДатаФрейма  data = []  # Из для каждого id в нашей id книге  for unique\_call\_id in calls\_df.CallId:  for unique\_responce in info\_request\_result\_data:  # Отыщем информацию в ответе о статуте звонков  if unique\_responce["id"] == unique\_call\_id:  # Время поднятия трубки  answered\_ts = unique\_responce["calls"][0]["answeredAt"]  # Время как положили трубку  finish\_ts = unique\_responce["calls"][0]["finishedAt"]  # Время начала рассылки  started\_ts = unique\_responce["calls"][0]["startedAt"]  # Код завершения  hangupCause = unique\_responce["calls"][0]["hangupCause"]  # Стоимость  cost = unique\_responce["calls"][0]["cost"]  # Переводим из unix timestamp в datetime  answered\_ts = pd.to\_datetime(answered\_ts, unit='s') + pd.Timedelta(hours=3)  finish\_ts = pd.to\_datetime(finish\_ts, unit='s') + pd.Timedelta(hours=3)  started\_ts = pd.to\_datetime(started\_ts, unit='s') + pd.Timedelta(hours=3)  # Проверка: если человек не поднял трубку,  # то время окончания разговора отсутствует,  # тогда время разговора выставляем -1  try:  time\_diff = (finish\_ts-answered\_ts).total\_seconds()  except TypeError:  time\_diff = -1  # Статус звонка  status = unique\_responce["calls"][0]["status"]  # Если у звонка статус 'finished', но он длился меньше заданного количества секунд, то принудительно выставляем ему статус canceled  if status == 'finished' and time\_diff < listening\_seconds:  status = 'canceled'  # Формируем tuple из собранных данных  unique\_info = (  unique\_call\_id, answered\_ts, finish\_ts,  started\_ts, status, time\_diff,  hangupCause, cost  )  data.append(unique\_info)  return pd.DataFrame(data=data, columns=column\_names) | Функция для получения датафрейма с состояниями по каждому звонку |
| def make\_calling\_results\_df(global\_calling\_id, calls\_df, states\_df, lap\_number):  round\_finished\_df = calls\_df.merge(states\_df, on="CallId")  round\_finished\_df["HangupCause"].fillna(value=0, inplace=True)  round\_finished\_df["HangupCause"] = round\_finished\_df["HangupCause"].astype(int)  round\_finished\_df["GlobalCallingID"] = [global\_calling\_id] \* round\_finished\_df.shape[0]  round\_finished\_df["LapNumber"] = lap\_number  return round\_finished\_df | Функция для получения датафрейма с результатами прозвона |
| def get\_phones\_to\_call\_again(calling\_results\_df, ignore\_states, listening\_seconds):  # Выделение записей с состояниями, не равными 16 (успех)  unsuccessive\_lap\_states = calling\_results\_df[  (calling\_results\_df["HangupCause"] != 16) |  (calling\_results\_df["Status"] != "finished") |  (calling\_results\_df["CallDuration"] < listening\_seconds)  ]  for ignore\_state in ignore\_states:  unsuccessive\_lap\_states = unsuccessive\_lap\_states[unsuccessive\_lap\_states["Status"] != ignore\_state]  phones\_to\_call\_again = list(unsuccessive\_lap\_states['PhoneNumber'])  return phones\_to\_call\_again | Функция для получения списка номеров, которые переходят на следующий круг |
| def make\_init\_df(init\_dict):  df\_init = pd.DataFrame(init\_dict, index=[0])  return df\_init | Функция для получения датафрейма с параметрами прозвона из словаря |

2.4 Модуль callings

|  |  |
| --- | --- |
| def \_launch\_lap(phones\_to\_call, outgoing\_phone, ringtone\_id, calling\_type):  global error\_calls\_cnt  phone\_number\_id\_arr = []  splits = zvonobot\_data\_processing.chunks(phones\_to\_call, max\_phones\_in\_one\_request) # Сплитованный массив-генератор с номерами  for split in splits:  if calling\_type == 'информационный':  response = zvonobot\_requests.run\_calling(\_department, split, outgoing\_phone, ringtone\_id)  else:  response = zvonobot\_requests.run\_calling\_with\_template(\_department, split, outgoing\_phone, ringtone\_id)  response\_json = response.json()  if response.ok: # Проверка корректности post запроса (ответ от сервака)  print("Autocalls were submitted and ran!")  if response\_json["status"] == "success": # Проверка корректности запроса на стороне Звонобота  response\_data = response\_json["data"]  # Из ответа от сервака получим номера телефонов + id звонка  # звонок на каждый номер имеет свою уникальную id  for recording in response\_data:  print(recording)  if 'id' in recording:  recording\_phone\_id = recording["id"]  else:  error\_calls\_cnt += 1  print(f'С номером телефона {recording["phone"]} возникла ошибка: {recording["message"]} ')  continue  recording\_phone\_number = recording["phone"]  phone\_number\_id\_arr.append(  [recording\_phone\_id, recording\_phone\_number])  else:  print("Error in autocall post-request!")  raise zvonobot\_exceptions.CallingRequestException(response\_json['data']['message'])  else:  print("Autocalls were not submitted! Error!")  raise zvonobot\_exceptions.CallingRequestException(response\_json['data']['message'])  # Создаем датаФрейм ["ID звонка", "Номер телефона"]  calls\_df = pd.DataFrame(data=phone\_number\_id\_arr, columns=["CallId", "PhoneNumber"])  print(calls\_df)  return calls\_df | Приватная функция для запуска нового круга прозвона. Предполагается использование только внутри модуля |
| def \_launch\_lap\_state\_check(calls\_df, listening\_seconds, calling\_type):  call\_ids = list(calls\_df["CallId"]) # Выделяем IDs звонков и делим их на порции  splits = list(zvonobot\_data\_processing.chunks(call\_ids, max\_phones\_in\_one\_request))  print("Начинаем проверку окочания рассылки по одному кругу.")  is\_finished\_local = False  while not is\_finished\_local:  info\_state\_responses\_array = []  for call\_ids in splits:  if calling\_type == 'информационный': # Запрос о состоянии рассылки  response = zvonobot\_requests.check\_calling\_state(\_department, call\_ids)  else:  response = zvonobot\_requests.check\_calling\_state\_with\_template(\_department, call\_ids)  response\_json = response.json()  if response.ok: # Проверка корректности post запроса (ответ от сервака)  print("Info was loaded!")  if response\_json["status"] == "success": # Проверка корректности запроса на стороне Звонобота  response\_data = response\_json["data"]  info\_state\_responses\_array += response\_data  else:  print("Something wrong with the autocall post-request!")  raise zvonobot\_exceptions.StateRequestException(response\_json['data']['message'])  else:  print("Info was not loaded! ERROR!!!")  raise zvonobot\_exceptions.StateRequestException(response\_json['data']['message'])  print("Собранные request responces info were collected: ", info\_state\_responses\_array)  state\_df = zvonobot\_data\_processing.make\_state\_df(info\_state\_responses\_array, calls\_df, listening\_seconds) # Создаем датафрейм с состояниями звонков  print(state\_df)  callings\_states = state\_df.Status.unique() # Проверяем статусы звонков  if set(callings\_states).issubset(set(lap\_finish\_states)):  is\_finished\_local = True  print("Прозвон по одному кругу закончен!")  else:  print(time.ctime(), "Прозвон не закончен, имеются состояния:", callings\_states)  time.sleep(10)  return info\_state\_responses\_array | Приватная функция для запуска проверки состояний круга прозвона. Предполагается использование только внутри модуля |
| def launch\_calling(init\_dict, numbers\_df):  global global\_calling\_id, error\_calls\_cnt, global\_state\_responses\_array, \_department  global\_state\_responses\_array = []  error\_calls\_cnt = 0  init\_df = zvonobot\_data\_processing.make\_init\_df(init\_dict)  dev\_db\_context.insert\_calling\_params(init\_df)  global\_calling\_id = dev\_db\_context.get\_last\_calling\_id() # Получаем глобальный индекс прозвона  print("Global calling ID: ", global\_calling\_id)  numbers\_df["GlobalCallingID"] = [global\_calling\_id] \* numbers\_df.shape[0] # Получаем датафрейм с номерами для прозвона  dev\_db\_context.insert\_phones\_to\_call(numbers\_df)  phones\_to\_call = list(numbers\_df['Numbers'])  # Исходящий номер  outgoing\_phone = init\_dict["OutgoingPhone"]  # Отдел  \_department = init\_dict["InitiatorDepartment"]  # Максимальный номер круга == максимальное количество  # обзвонов по данной глобальной рассылке.  max\_lap\_number = init\_dict["MaxLapNumber"]  # ID рингтона  ringtone\_id = init\_dict["RingtoneID"]  # Расстояние между перезвонами  if init\_dict["InterCallDistance"]:  intercall\_distance = int(init\_dict["InterCallDistance"]) \* 60  # Длина ролика в секундах, которая будет считаться успешным контактом  listening\_seconds = init\_dict["ListenedInSeconds"]  # Тип прозвона (информационный/диалоговый)  calling\_type = init\_dict['CallingType']  # Завершен ли прозвон  is\_finished = False  while not is\_finished:  # Запуск рассылки  print(f"В данном кругу участвует {len(phones\_to\_call)} номеров.")  # Запуск рассылки по одному кругу  try:  calls\_df = \_launch\_lap(phones\_to\_call, outgoing\_phone, ringtone\_id, calling\_type)  except zvonobot\_exceptions.CallingRequestException:  raise  # Запуск проверки состояния рассылки  try:  state\_responses\_array = \_launch\_lap\_state\_check(calls\_df, listening\_seconds, calling\_type)  global\_state\_responses\_array += state\_responses\_array  except zvonobot\_exceptions.StateRequestException:  raise  # Номер последнего круга  last\_lap\_number = dev\_db\_context.get\_last\_lap\_number(global\_calling\_id)  states\_df = zvonobot\_data\_processing.make\_state\_df(state\_responses\_array, calls\_df, listening\_seconds)  calling\_results\_df = zvonobot\_data\_processing.make\_calling\_results\_df(  global\_calling\_id, calls\_df, states\_df, last\_lap\_number)  # Состояние в последнем круге  calling\_states\_ok = set(states\_df['Status']) == {'finished'}  print("ДатаФрейм состояний")  print(calling\_results\_df)  if dev\_db\_context.get\_calling\_results(global\_calling\_id).shape[0] > 0:  last\_lap\_number += 1  print("Global calling index: ", global\_calling\_id,  "Lap: ", last\_lap\_number)  # Загружаем данные с результатами прозвона в БД  # с присвоением круга и индексом глабальной рассылки  dev\_db\_context.insert\_calling\_results(calling\_results\_df)  # Завершен ли прозвон кнопкой 'Нет, начать новый прозвон!'  calling\_finished\_manually = dev\_db\_context.get\_calling\_params(global\_calling\_id).loc[0, 'Finished']  if calling\_finished\_manually:  print(f'Прозвон {global\_calling\_id} завершен пользовтелем!')  # Проверка на окончание обзвона  if last\_lap\_number >= max\_lap\_number or calling\_states\_ok or calling\_finished\_manually:  is\_finished = True  # Загрузка в БД что обзвон успешно завершен  dev\_db\_context.finish\_calling(global\_calling\_id)  print("Рассылка закончена")  else:  print("Запускаем следующий круг")  # Выдлеляем номера на следующий круг  phones\_to\_call = zvonobot\_data\_processing.get\_phones\_to\_call\_again(calling\_results\_df, ignore\_states, listening\_seconds)  # Ожидание для запуска следующей рассылки  time.sleep(intercall\_distance) | Основная функция для запуска прозвона |

3 Уровень представления (presentation)

|  |  |
| --- | --- |
| @app.route('/get\_numbers')  @login\_required  def get\_free\_phone\_numbers():  # Заправшиваем номера телефонов  try:  request\_response = zvonobot\_requests.get\_outgoing\_phones(APIRights.chosen\_department).json()  except requests.exceptions.ConnectionError:  return "Ошибка при получении номеров", 502  # Создаем датасет из ответа Звонобота  df\_phones = pd.DataFrame.from\_records(request\_response['data'])  # Ранжируем, в начале списка остаются арендные номера, затем общие\бесплатные  df\_phones\_sorted = df\_phones.sort\_values(by="overall")  overall\_dict = {0: "Арендный", 1: "Общий"}  df\_phones\_sorted["phone\_status"] = df\_phones\_sorted["overall"].map(overall\_dict)  # Оставляем только [НомерТелефона:Статус]  phone\_statuses = df\_phones\_sorted[["phone", "phone\_status"]]  # Переводим обработанные данные в json  phones\_json = phone\_statuses.to\_json(orient='records', force\_ascii=False)  return phones\_json | Обработчик запроса списка номеров |
| @app.route('/login', methods=["POST", "GET"])  def login():  global is\_calling\_finished  if current\_user.is\_authenticated:  if 'next' in request.args:  return redirect(url\_for(request.args['next']))  return redirect(url\_for('choose\_calling\_type',  departmets=APIRights.departments\_list,  calling\_aims=calling\_aims))  if request.method == 'POST':  login = request.form['login']  password = request.form['password']  APIRights.name = login  department\_keys = dev\_db\_context.get\_department\_keys(login)  APIRights.departments\_list = list(department\_keys['Department'])  APIRights.api\_dict = department\_keys.set\_index('Department').to\_dict()['API\_key']  user = User()  user.id = login  User.id = login  try:  try\_login(login, password)  flask\_login.login\_user(user)  except (ldap3.core.exceptions.LDAPInvalidCredentialsResult,  ldap3.core.exceptions.LDAPUserNameIsMandatoryError,  ldap3.core.exceptions.LDAPPasswordIsMandatoryError):  flash('Неверный логин и/или пароль')  return render\_template('login.html')  was\_last\_run\_successful\_bool, last\_global\_calling = get\_if\_last\_run\_was\_success()  if was\_last\_run\_successful\_bool:  return redirect(url\_for('choose\_calling\_type'))  else:  is\_calling\_finished = False  return redirect(url\_for('last\_calling\_error'))  return render\_template('login.html') | Контроллер страницы авторизации |
| @app.route('/last\_calling\_error', methods=['POST', 'GET'])  @login\_required  def last\_calling\_error():  last\_run\_df = dev\_db\_context.get\_last\_user\_calling\_params(User.id).iloc[0]  if request.method == 'POST': # Нажата кнопка "Продолжить прозвон старого!"  return render\_template("informative\_calling\_started.html",  sending\_description=last\_run\_df['Description'])  else:  calling\_type, last\_unfinished\_run\_info = get\_last\_unfinished\_run\_array(last\_run\_df)  return render\_template('last\_calling\_error.html',  calling\_type=calling\_type,  data=last\_unfinished\_run\_info) | Контроллер страницы незавершенного прозвона |
| @app.route('/', methods=['POST', 'GET'])  @login\_required  def choose\_calling\_type():  if request.method == 'POST':  department = request.form.get('choose\_department')  calling\_type = request.form.get('options')  APIRights.chosen\_department = department  CallingParams.calling\_type = calling\_type  if department is not None:  APIRights.chosen\_api = APIRights.api\_dict[department]  print(department, calling\_type, APIRights.chosen\_api)  if calling\_type == 'информационный':  return redirect(url\_for('informative\_main'))  else:  return redirect(url\_for('dialog\_main'))  else:  # Если department не выбран, значит запрос пришел со страницы /last\_calling\_error  # по нажатию кнопки 'Нет, начать новый прозвон!'. В этом случае завершаем старый прозвон.  dev\_db\_context.finish\_last\_user\_calling(User.id)  if not hasattr(APIRights, 'departments\_list'):  return redirect(url\_for('logout'))  return render\_template('choose\_calling\_type.html',  departmets=APIRights.departments\_list,  calling\_aims=calling\_aims) | Контроллер страницы выбора типа прозвона |
| @app.route("/informative\_main", methods=['GET', 'POST'])  @login\_required  def informative\_main():  global is\_calling\_finished  is\_calling\_finished = False  account\_balance = zvonobot\_requests.get\_balance(APIRights.chosen\_department)  response = render\_template(  'informative\_main.html',  ringtone\_types=ringtone\_types,  max\_lap\_numbers=max\_lap\_numbers,  max\_lap\_numbers\_default=max\_lap\_numbers\_default,  balance=account\_balance)  return response | Контроллер страницы информационных прозвонов |
| @login\_required  @app.route('/begin\_calling', methods=['POST'])  def begin\_calling():  global is\_calling\_finished, global\_calling\_id  if is\_calling\_finished:  print('Прозвон уже завершен!')  finished, canceled, error = get\_calling\_results()  return render\_template('informative\_calling\_finished.html',  finished=finished, canceled=canceled, error=error,  sending\_description=CallingParams.sending\_description)  if not is\_calling\_finished and check\_is\_calling\_ongoing():  print('Прозвон уже запущен!')  while check\_is\_calling\_ongoing():  time.sleep(10)  is\_calling\_finished = True  finished, canceled, error = get\_calling\_results()  return render\_template('informative\_calling\_finished.html',  finished=finished, canceled=canceled, error=error,  sending\_description=CallingParams.sending\_description)  print('Начинаем новый прозвон!')  outgoing\_phone\_number = CallingParams.chosen\_phone\_number.split(":")[0]  phone\_number\_type\_temp = CallingParams.chosen\_phone\_number.split(":")[1]  init\_dict = {  "CurrentUser": User.id,  "InitiatorName": CallingParams.initiator\_name,  "InitiatorDepartment": APIRights.chosen\_department,  "Description": CallingParams.sending\_description,  "TextOrID": CallingParams.ringtone\_type,  "RingtoneID": CallingParams.ringtone\_id,  "Text": CallingParams.ringtone\_text,  "OutgoingPhone": outgoing\_phone\_number,  "OutgoingPhoneType": phone\_number\_type\_temp,  "SuccessCallType": CallingParams.success\_call\_type,  "ListenedInSeconds": CallingParams.seconds\_listened,  "AnsweringMachine": CallingParams.check\_box\_auto\_reply,  "MaxLapNumber": CallingParams.max\_lap\_numbers,  "InterCallDistance": CallingParams.intercall\_distance,  "CallingType": CallingParams.calling\_type  }  try:  # Запуск прозвона  global\_calling\_id = dev\_db\_context.get\_last\_calling\_id() + 1  zvonobot\_callings.launch\_calling(init\_dict, CallingParams.df\_numbers)  except (zvonobot\_exceptions.CallingRequestException,  zvonobot\_exceptions.StateRequestException) as e:  global\_calling\_id = 0  return render\_template('error.html', err=e,  return\_url='/')  finally:  is\_calling\_finished = True  finished, canceled, error = get\_calling\_results()  return render\_template('informative\_calling\_finished.html',  finished=finished, canceled=canceled, error=error,  sending\_description=CallingParams.sending\_description) | Обрабочик запроса на запуск прозвона |
| @app.route('/logout')  def logout():  flask\_login.logout\_user()  return redirect(url\_for('login')) | Обработчик запроса на выход из системы |

4 Диалоговые сценарии

4.1 Общие модули

4.1.1 Модуль callbase\_maker

|  |  |
| --- | --- |
| def \_get\_phones(clients\_df):  clients\_data = {'PhoneNumber': [], 'IDAdress': []}  for index, record in clients\_df.iterrows():  if record['PhoneNumber'] is not None:  client\_phones = record['PhoneNumber'].split('; ')  client\_address = record['IDAdress']  clients\_data['PhoneNumber'].extend(client\_phones)  clients\_data['IDAdress'].extend([client\_address] \* len(client\_phones))  return clients\_data | Приватная функция для получения датафрейма с номерами |
| def \_process\_phones(phones\_list):  for i in range(len(phones\_list)):  if phones\_list[i][:2] == '+7':  phones\_list[i] = phones\_list[i][1:]  elif phones\_list[i][0] != '7':  phones\_list[i] = '7' + phones\_list[i] | Приватная функция для приведения номеров к виду 7XXXXXXXXXX |
| def \_remove\_invalid\_phones(phones\_dict):  i = 0  while i < len(phones\_dict['PhoneNumber']):  if len(phones\_dict['PhoneNumber'][i]) != 11 or len(set(phones\_dict['PhoneNumber'][i][1:])) == 1:  phones\_dict['PhoneNumber'].pop(i)  phones\_dict['IDAdress'].pop(i)  i -= 1  i += 1 | Приватная функция для удаления некорректных номеров |

|  |  |
| --- | --- |
| def \_remove\_symbols(phones\_dict):  for i in range(len(phones\_dict['PhoneNumber'])):  phones\_dict['PhoneNumber'][i] = ''.join(c if c.isdigit() else '' for c in phones\_dict['PhoneNumber'][i]) | Приватная функция для удаления всех символов из номеров |
| def \_remove\_duplicate\_phones(phones\_dict):  phones\_df = pd.DataFrame.from\_dict(phones\_dict)  phones\_df.drop\_duplicates(inplace=True, subset='PhoneNumber')  return phones\_df | Приватная функция для удаления номеров, на которые уже был совершен звонок |
| def \_remove\_repeat\_phones(phones\_df, called\_phones, min\_amount=None):  new\_phones\_df = phones\_df[~phones\_df['PhoneNumber'].isin(called\_phones)]  if min\_amount is not None and len(new\_phones\_df) < min\_amount:  repeat\_phones\_df = phones\_df[phones\_df['PhoneNumber'].isin(called\_phones)]  repeat\_phones\_df.index = range(len(repeat\_phones\_df))  new\_phones\_df = pd.concat([new\_phones\_df, repeat\_phones\_df.loc[:min\_amount-len(new\_phones\_df)-1, :]])  new\_phones\_df.index = range(len(new\_phones\_df))  return new\_phones\_df | Приватная функция для удаления дублирующихся в датафрейме номеров |
| def \_select\_n\_phones(phones\_df, n):  return phones\_df.loc[:n-1, :] | Приватная функция для выбора определенного количества номеров |
| def \_get\_project\_callbase(project\_name, quarters, queues, amounts, called\_phones):  project\_callbase = []  if quarters is not None: # Если передан список кварталов, выбираем по кварталам  for quarter, amount in zip(quarters, amounts):  quarter\_clients\_df = dwh\_db\_context.get\_clients(project\_name, quarter, None)  phones\_dict = \_get\_phones(quarter\_clients\_df)  \_remove\_symbols(phones\_dict)  \_remove\_invalid\_phones(phones\_dict)  phones\_df = \_remove\_duplicate\_phones(phones\_dict)  phones\_df = \_remove\_repeat\_phones(phones\_df, called\_phones)  phones\_df = \_select\_n\_phones(phones\_df, amount)  project\_callbase.append(phones\_df)  else: # Если передан список очередей, выбираем по очередям  for queue, amount in zip(queues, amounts):  queue\_clients\_df = dwh\_db\_context.get\_clients(project\_name, None, queue)  phones\_dict = \_get\_phones(queue\_clients\_df)  \_remove\_symbols(phones\_dict)  \_remove\_invalid\_phones(phones\_dict)  phones\_df = \_remove\_duplicate\_phones(phones\_dict)  phones\_df = \_remove\_repeat\_phones(phones\_df, called\_phones)  phones\_df = \_select\_n\_phones(phones\_df, amount)  project\_callbase.append(phones\_df)  return pd.concat(project\_callbase) | Приватная функция для получения базы номеров по проекту (ЖК) |
| def \_get\_houses\_callbase(project\_name, quarters, called\_phones):  project\_callbase = []  for quarter in quarters:  quarter\_clients\_df = dwh\_db\_context.get\_clients(project\_name, quarter, None)  houses = set(quarter\_clients\_df['House'])  for house in houses:  house\_clients\_df = quarter\_clients\_df[quarter\_clients\_df['House'] == house]  phones\_dict = \_get\_phones(house\_clients\_df)  \_remove\_symbols(phones\_dict)  \_remove\_invalid\_phones(phones\_dict)  phones\_df = \_remove\_duplicate\_phones(phones\_dict)  phones\_df = \_remove\_repeat\_phones(phones\_df, called\_phones)  phones\_df = \_select\_n\_phones(phones\_df, 30)  project\_callbase.append(phones\_df)  return pd.concat(project\_callbase) | Приватная функция для получения базы номеров по домам |
| def get\_full\_nps\_callbase():  called\_phones = list(dev\_db\_context.get\_nps\_scores()['PhoneNumber']) # Получаем список уже обзвоненых номеров  sd\_callbase\_df = \_get\_project\_callbase('Северная долина', [15, 19, 20, 21], None, [250] \* 4, called\_phones)  yuntolovo\_callbase\_df = \_get\_project\_callbase('Юнтолово', None, [1, 2, 3, 4, 5, 6], [50]\*6, called\_phones)  full\_callbase = pd.concat([sd\_callbase\_df, yuntolovo\_callbase\_df])  return full\_callbase | Функция для получения полной базы номеров для сценария NPS |

|  |  |
| --- | --- |
| def get\_full\_csat\_callbase():  callbase = []  called\_phones = list(dev\_db\_context.get\_csat\_scores()['PhoneNumber']) # Получаем список уже обзвоненых номеров  phones\_df = dwh\_db\_context.get\_dispatcher\_service\_calls() # Получаем датафрейм звонков в диспетчерскую службу  if len(phones\_df) == 0:  return pd.DataFrame(columns=['PhoneNumber'])  for day\_of\_week in range(5): # Формируем базу по дням недели с Пн по Пт  phones\_list = list(phones\_df[phones\_df['calldate'].dt.weekday == day\_of\_week]['cnam'])  \_process\_phones(phones\_list)  day\_phones\_df = \_remove\_duplicate\_phones({'PhoneNumber': phones\_list})  day\_phones\_df = \_remove\_repeat\_phones(day\_phones\_df, called\_phones)  day\_phones\_df = \_select\_n\_phones(day\_phones\_df, 30)  callbase.append(day\_phones\_df)  return pd.concat(callbase) | Функция для получения полной базы номеров для сценария CSAT |
| def get\_full\_admin\_callbase():  called\_phones = list(dev\_db\_context.get\_admin\_scores()['PhoneNumber']) # Получаем список уже обзвоненых номеров  sd\_callbase\_df = \_get\_houses\_callbase('Северная долина', [15, 19, 20, 21], called\_phones)  yuntolovo\_callbase\_df = \_get\_houses\_callbase('Юнтолово', [1, 2, 3, 4], called\_phones)  p360\_callbase\_df = \_get\_houses\_callbase('Панорама 360', [None], called\_phones)  full\_callbase = pd.concat([sd\_callbase\_df, yuntolovo\_callbase\_df, p360\_callbase\_df])  return full\_callbase | Функция для получения полной базы номеров для сценария ADMIN |

4.1.2 Модуль data\_processing

|  |  |
| --- | --- |
| def make\_init\_dict(config):  new\_init\_data = {  "CurrentUser": None,  "InitiatorName": None,  "InitiatorDepartment": 'ЭГС',  "Description": config['description'],  "TextOrID": "ID из Звонобота",  "RingtoneID": config['nps\_api\_template\_id'],  "Text": None,  "OutgoingPhone": config['outgoing\_phone'],  "OutgoingPhoneType": config['outgoing\_phone\_type'],  "SuccessCallType": None,  "ListenedInSeconds": config['listened\_in\_seconds'],  "AnsweringMachine": None,  "MaxLapNumber": config['max\_lap\_number'],  "InterCallDistance": config['intercall\_distance'],  "CallingType": "диалоговый",  }  return new\_init\_data | Функция для создания словаря с параметрами прозвона из файла конфигурации |
| def make\_scores\_df(info\_request\_result\_data, global\_calling\_id, config):  scores\_dict = {'GlobalCallingID': [], 'PhoneNumber': [], 'Score': [], 'RecordPath': [], 'RecordTranscription': []}  for data in info\_request\_result\_data:  for call in data['calls']:  if call['status'] in ['error', 'canceled']: # Пропускаем звонок, если он закончился с ошибкой или отклонен  continue  ivr\_answers = call['ivrAnswers'].split('|')  phone = call['phone']  score = ivr\_answers[1] if len(ivr\_answers) > 1 else -1  score = int(score) if score == -1 or score.isdigit() else -1  score = -1 if score < 0 or score > 10 else score  record\_path = path.join(  config['record\_dir'], f'{global\_calling\_id}\_{phone}.wav') if len(ivr\_answers) > 2 else None  transcription = ivr\_answers[2] if len(ivr\_answers) > 2 else None  if record\_path is not None:  save\_record(record\_path, call['recordFilePath'], config)  if phone not in scores\_dict['PhoneNumber']:  scores\_dict['GlobalCallingID'].append(global\_calling\_id)  scores\_dict['PhoneNumber'].append(phone)  scores\_dict['Score'].append(score)  scores\_dict['RecordPath'].append(record\_path)  scores\_dict['RecordTranscription'].append(transcription)  else:  phone\_index = scores\_dict['PhoneNumber'].index(phone)  prev\_score = scores\_dict['Score'][phone\_index]  # Если телефон уже есть в scores\_dict, устанавливаем максимальную из двух оценок  if score > prev\_score:  scores\_dict['Score'][phone\_index] = score  scores\_dict['RecordPath'][phone\_index] = record\_path  scores\_dict['RecordTranscription'][phone\_index] = transcription  return pd.DataFrame(scores\_dict) | Функция для создания датафреймов с оценками клиентов из ответов сервера |
| def save\_record(record\_path, download\_url, config):  answer\_start\_time = config['answer\_start\_time']  record = zvonobot\_requests.download\_record(download\_url)  with open(record\_path, 'wb') as file:  file.write(record)  audio\_segment = AudioSegment.from\_file(record\_path)[answer\_start\_time \* 1000:]  audio\_segment.export(record\_path, format='wav') | Функция для сохранения записи разговора по указанному пути |

4.2 Скрипты

|  |  |
| --- | --- |
| # Файл конфигурации  with open('nps\_cfg.yaml', 'r', encoding='utf8') as f:  nps\_config = yaml.load(f, Loader=yaml.FullLoader)  print('Сценарий NPS запущен')  phones\_to\_call = callbase\_maker.get\_full\_nps\_callbase() # Получаем базу номеров для прозвона  df\_init = data\_processing.make\_init\_dict(nps\_config) # Создаем датафрейм с параметрами прозвона  print(df\_init)  # Получаем датафрейм с номерами для прозвона  numbers\_df = data\_processing.make\_numbers\_df(phones\_to\_call)  zvonobot\_callings.launch\_calling(df\_init, numbers\_df)  # Получаем датафрейм с оценками клиентов  scores\_df = data\_processing.make\_nps\_scores\_df(zvonobot\_callings.global\_state\_responses\_array, zvonobot\_callings.global\_calling\_id)  print(scores\_df)  dev\_db\_context.insert\_nps\_scores(scores\_df)  print('Сценарий NPS завершен') | Скрипт NPS |
| # Файл конфигурации  with open('csat\_cfg.yaml', 'r', encoding='utf8') as f:  csat\_config = yaml.load(f, Loader=yaml.FullLoader)  print('Сценарий CSAT запущен')  phones\_to\_call = callbase\_maker.get\_full\_csat\_callbase() # Получаем базу номеров для прозвона  df\_init = data\_processing.make\_init\_dict(csat\_config) # Создаем датафрейм с параметрами прозвона  print(df\_init)  numbers\_df = data\_processing.make\_numbers\_df(phones\_to\_call) # Получаем датафрейм с номерами для прозвона  zvonobot\_callings.launch\_calling(df\_init, numbers\_df) # Запускаем прозвон  # Получаем датафрейм с оценками клиентов  scores\_df = data\_processing.make\_scores\_df(  zvonobot\_callings.global\_state\_responses\_array, zvonobot\_callings.global\_calling\_id, csat\_config)  print(scores\_df)  dev\_db\_context.insert\_csat\_scores(scores\_df)  print('Сценарий CSAT завершен') | Скрипт CSAT |
| # Файл конфигурации  with open('admin\_cfg.yaml', 'r', encoding='utf8') as f:  admin\_config = yaml.load(f, Loader=yaml.FullLoader)  print('Сценарий ADMIN запущен')  phones\_to\_call = callbase\_maker.get\_full\_admin\_callbase() # Получаем базу номеров для прозвона  df\_init = data\_processing.make\_init\_dict(admin\_config) # Создаем датафрейм с параметрами прозвона  print(df\_init)  # Получаем датафрейм с номерами для прозвона  numbers\_df = data\_processing.make\_numbers\_df(phones\_to\_call)  zvonobot\_callings.launch\_calling(df\_init, numbers\_df) # Запускаем прозвон  # Получаем датафрейм с оценками клиентов  scores\_df = data\_processing.make\_scores\_df( zvonobot\_callings.global\_state\_responses\_array, zvonobot\_callings.global\_calling\_id, admin\_config)  print(scores\_df)  dev\_db\_context.insert\_admin\_scores(scores\_df)  print('Сценарий ADMIN завершен') | Скрипт ADMIN |

**ПРИЛОЖЕНИЕ Г**

**Руководство пользователя**

**1 Введение**

1.1 Область применения

Пользовательский интерфейс системы обеспечивает информационную поддержку деятельности сотрудника при выполнении следующих операций:

1. Запуск информационных и диалоговых прозвонов с указанием параметров;
2. Продолжение или завершение уже начатых прозвонов;
3. Получение информации о результатах прозвона.

1.2 Краткое описание возможностей

Система обеспечивает выполнение следующих функций:

1. Управление информационными голосовыми рассылками;
2. Управление диалоговыми голосовыми рассылками;
3. Предоставление информации о результатах голосовых рассылок.

1.3 Уровень подготовки пользователя

Для эксплуатации системы определена одна роль − сотрудник компании.

Пользователь должен:

1. Иметь общие сведения о системе и ее назначении;
2. Владеть информацией о системе в объеме эксплуатационной документации;
3. Владеть информацией о работе в интерфейсе;

1.4 Перечень эксплуатационной документации, с которыми необходимо ознакомиться пользователю

Перед использованием системы, пользователю необходимо ознакомиться с руководством пользователя (настоящий документ).

**2 Назначение и условия применения**

2.1 Назначение

Веб-приложение предназначено для повышения эффективности работы сотрудников ООО «Главстрой-СПб специализированный застройщик».

Основным назначением приложения является автоматизация процессов управления голосовыми рассылками.

В рамках проекта автоматизируется информационно аналитическая деятельность в следующих бизнес – процессах:

1. Управление информационными и диалоговыми прозвонами;
2. Получение информации о результатах прозвонов

2.2 Условия применения

Приложение может эксплуатироваться и выполнять заданные функции при соблюдении требований, предъявляемых к техническому, системному и прикладному программному обеспечению.

**3. Описание операций**

В данном разделе приводится описание всех операций, существующих в приложении.

3.1 Авторизация

После запуска веб-интерфейса открывается страница авторизации, представленная на рисунке 26. На данной странице необходимо ввести логин и пароль пользователя.

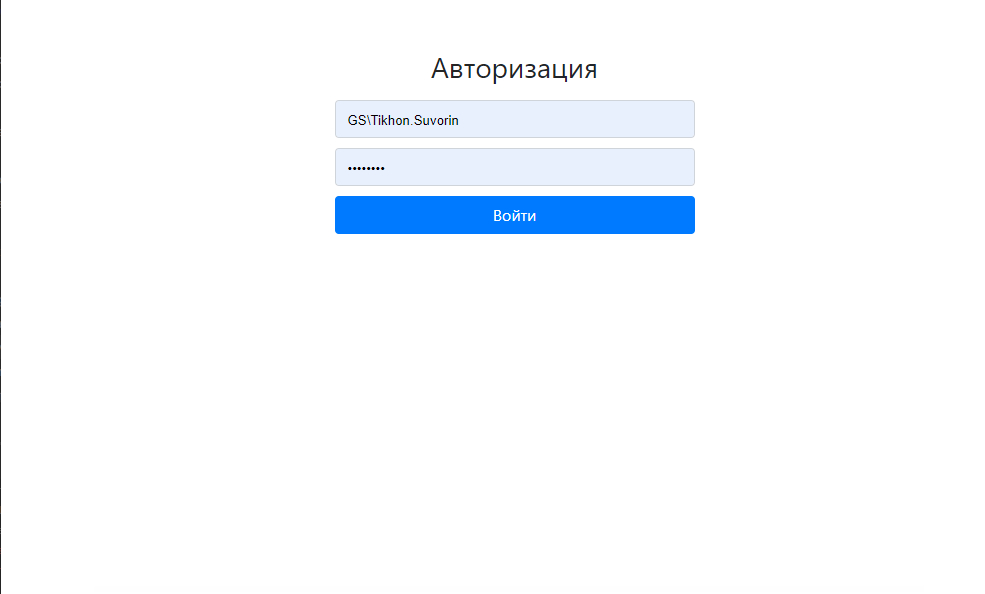


Рисунок 26 – Страница авторизации

После ввода авторизационных данных, следует нажать кнопку «войти». В результате откроется доступ к функционалу системы. В результате откроется либо страница выбора типа обзвона, либо страница уведомления о незавершенном прозвоне.

3.2 Страница выбора типа обзвона

Если незавершенные прозвоны отсутствуют, то после авторизации откроется страница выбора типа обзвона (рисунок 27).

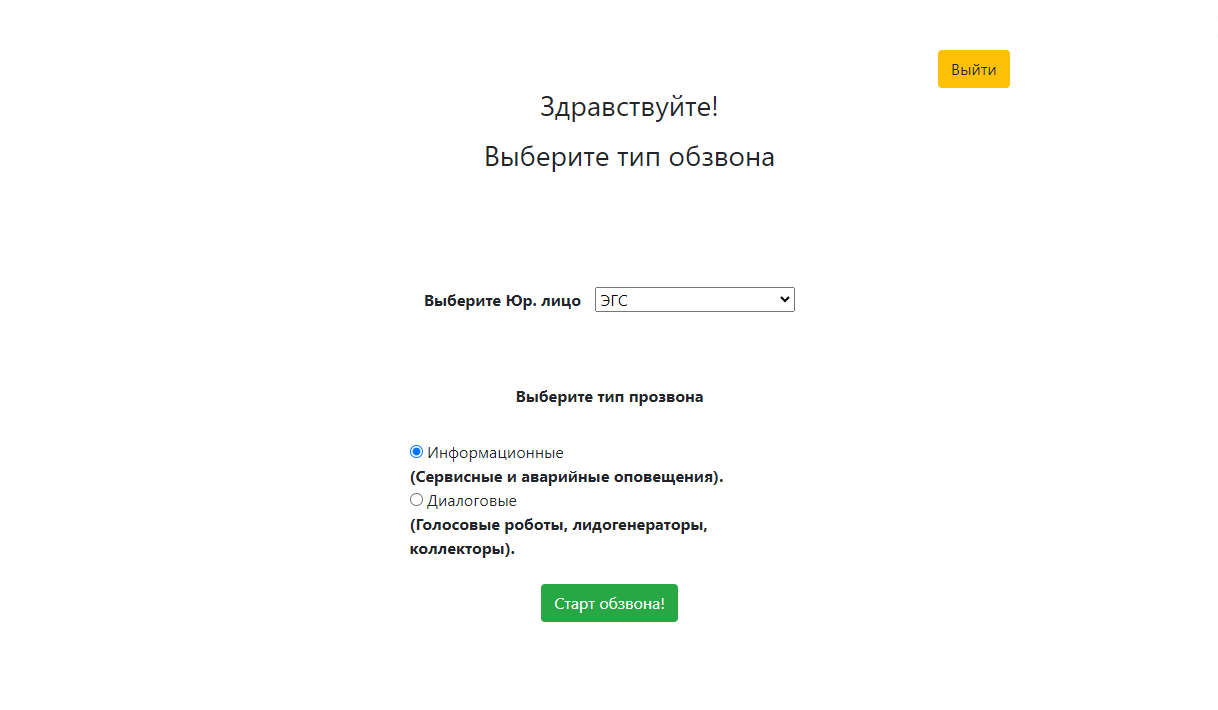


Рисунок 27 – Страница выбора типа обзвона

На данной странице необходимо выбрать юридическое лицо, от имени которого будет производиться прозвон: ЭГС («Эксплуатация Главстрой СПб»), или АН («Агентство недвижимости Главстрой СПб»). Далее следует выбрать нужный тип прозвона: информационный (клиенту будет проигран аудиоролик с определенной информацией, без диалога) или диалоговый (звонок будет проходить по заданному сценарию, от клиента будет получена обратная связь). После выбора типа для перехода к указанию параметров прозвона следует нажать кнопку «Старт обзвона!». Также, при необходимости выйти из системы, можно нажать кнопку «Выйти».

3.3 Страница уведомления о незавершенном прозвоне

В случае, если у пользователя имеется незавершенный прозвон, то после авторизации откроется страница уведомления о незавершенном прозвоне (рисунок 28).

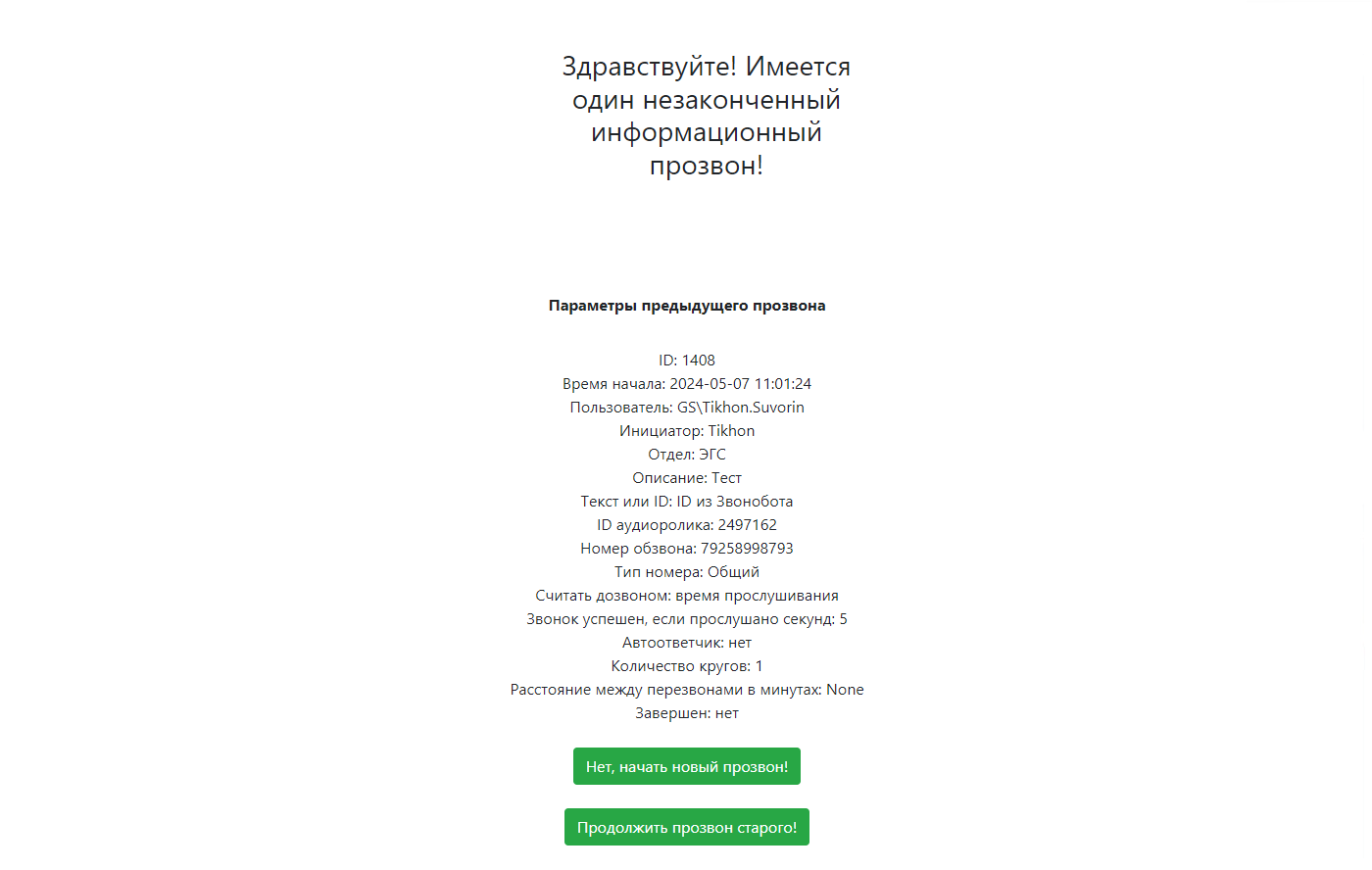


Рисунок 28 – Страница с информацией о незаконченном прозвоне

На данной странице можно ознакомиться с параметрами предыдущего прозвона, а затем либо продолжить его, нажав на кнопку «Продолжить прозвон старого», либо начать новый прозвон, нажав на кнопку «Нет, начать новый прозвон!».

3.4 Страница информационных прозвонов

Если на странице выбора типа обзвона выбран «информационный», то откроется страница конфигурации информационного прозвона (рисунок 29). Для удобства пользователя на странице отображается баланс на счету личного кабинета в сервисе «звонобот».

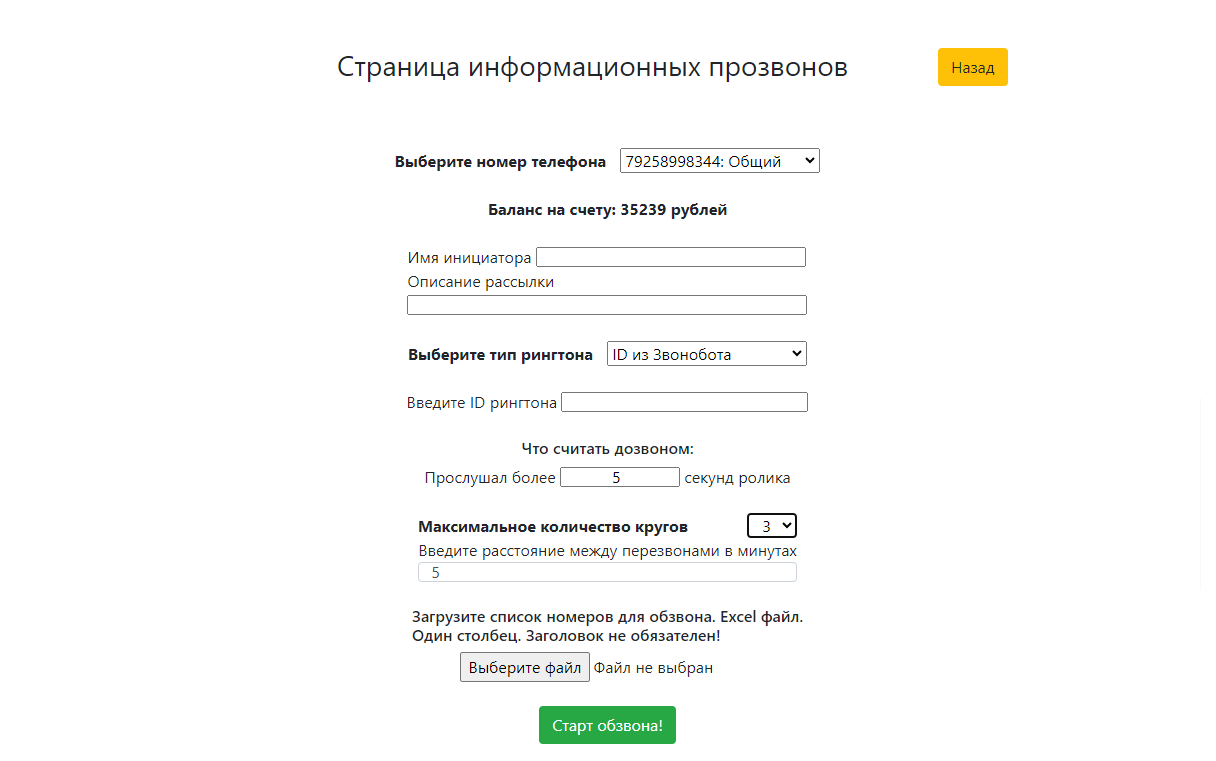


Рисунок 29 – Страница информационных прозвонов

Сразу после запуска страницы некоторое время займет получение исходящих номеров телефонов из сервиса «звонобот». Как только номера будут загружены и отображены в списке, необходимо выбрать один из них. Далее следует ввести необходимую информацию: имя инициатора прозвона, описание прозвона, выбрать тип рингтона (ID из звонобота или текст). В зависимости от выбора ввести либо ID аудиоролика, либо текст рассылки. Затем нужно указать, сколько секунд ролика должен прослушать клиент, чтобы звонок считался успешным. Далее следует указать максимальное количество кругов (попыток) прозвона, и указать расстояние между ними в минутах. После указания всех параметров необходимо загрузить файл в формате .xlsx/.xls с одним столбцом, содержащим номера для прозвона, и нажать кнопку «Старт обзвона!». При необходимости, также можно вернуться на страницу выбора типа обзвона, нажав на кнопку «Назад».

3.5 Страница диалоговых прозвонов

Если на странице выбора типа обзвона выбран «диалоговый», то откроется страница конфигурации диалогового прозвона (рисунок 30).

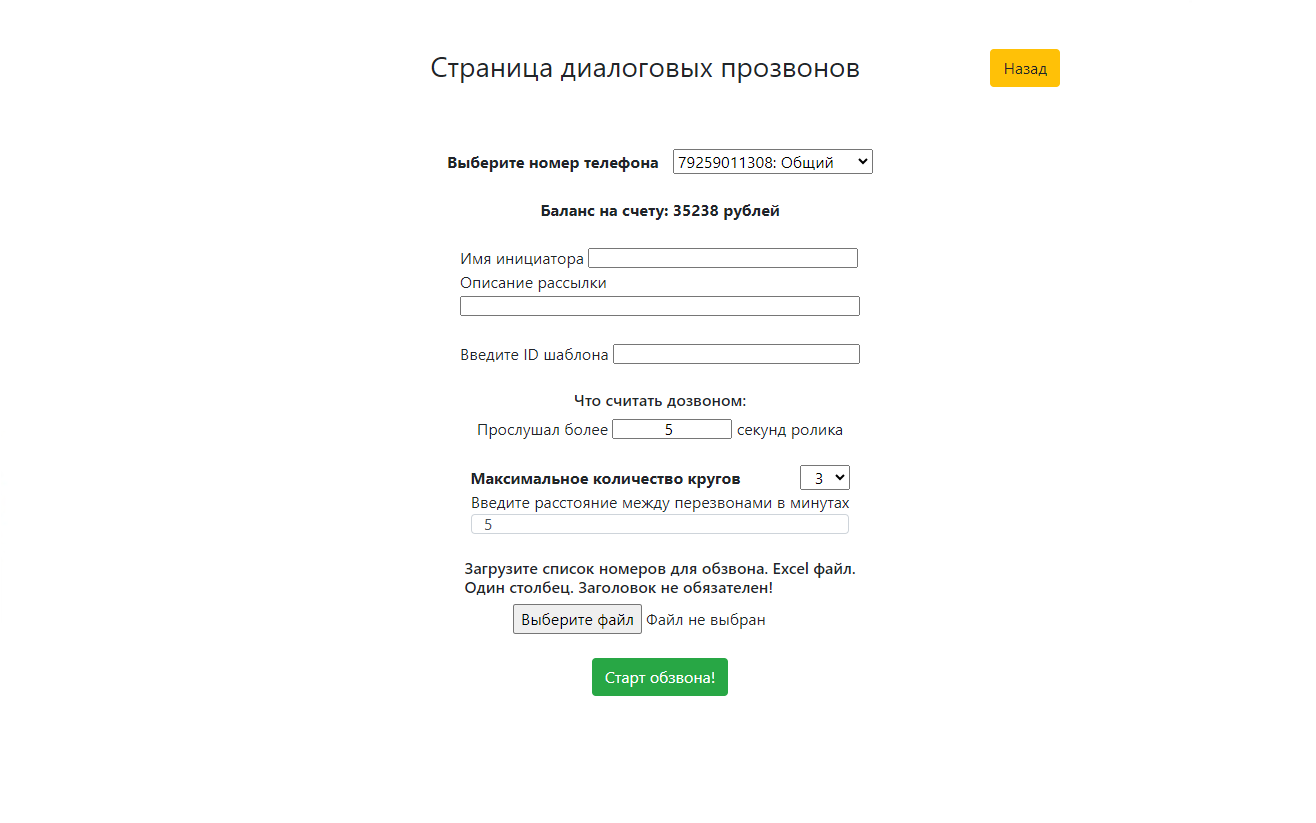


Рисунок 30 – Страница диалоговых прозвонов

Конфигурация диалогового прозвона во многом аналогична информационному, описанному в предыдущем пункте. Отличие заключается лишь в отсутствии возможности типа рингтона. Вместо этого необходимо ввести ID шаблона из сервиса «звонобот». Запуск рассылки осуществляется также кнопкой «Старт обзвона!».

3.6 Страница «Прозвон запущен!»

После нажатия кнопки «Старт обзвона!» на странице информационных или диалоговых прозвонов открывается страница, информирующая о том, что прозвон был запущен (рисунок 28):

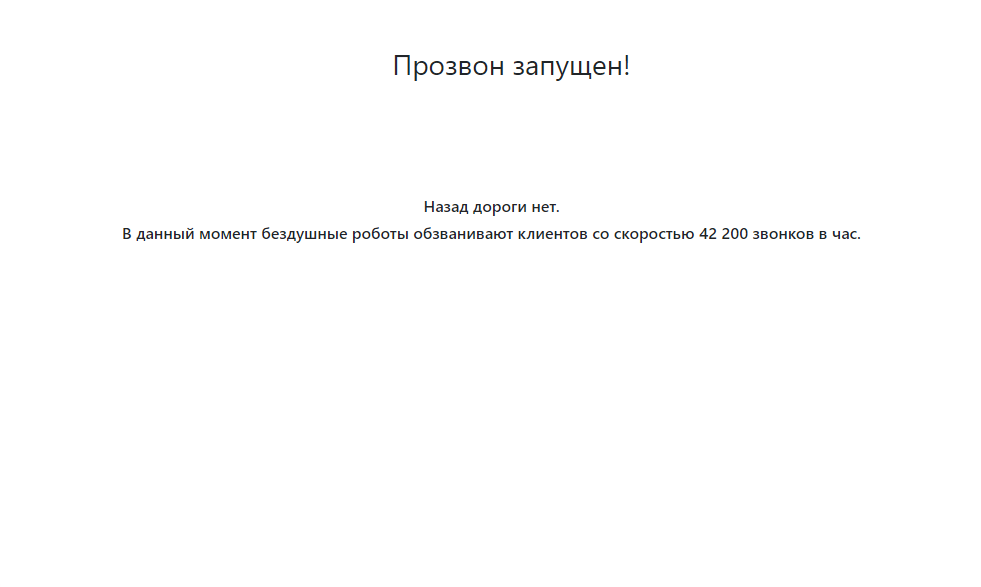


Рисунок 28 – Страница «Прозвон запущен!»

Данная страница не предполагает никакого взаимодействия, она будет отображаться до тех пор, пока прозвон не будет завершен.

3.7 Страница результатов прозвона

После того, как прозвон завершается, открывается страница с результатами прозвона (рисунок 29):

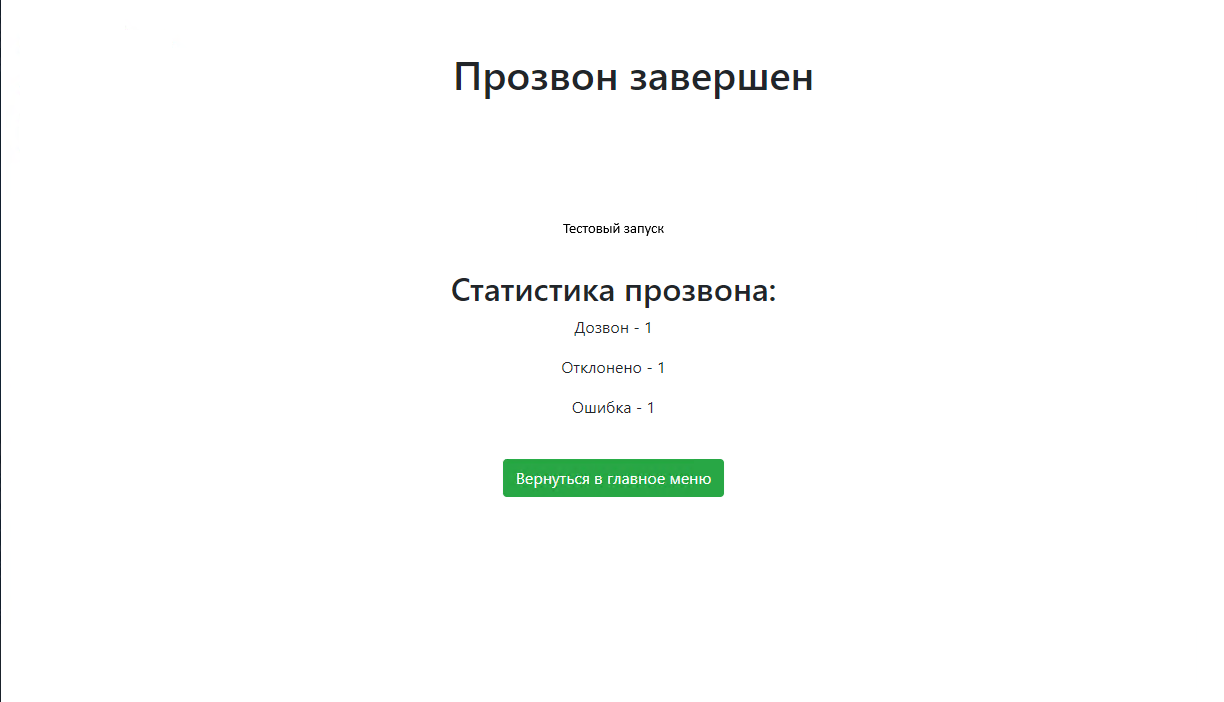


Рисунок 29 – Страница результатов прозвона

На данной странице отображается описание, введенное пользователем при конфигурации, и статистика прозвона (количество успешных, отклоненных и завершившихся с ошибкой звонков). После ознакомления с результатами прозвона вернуться на страницу выбора типа прозвона можно, нажав на кнопку «Вернуться в главное меню».

3.8 Страница вывода ошибок

Данная страница может быть отображена в случае возникновения каких-либо нештатных ситуаций. Например, на рисунке 30 представлено отображение ошибки о попытке запуска прозвона с несуществующим ID аудиоролика.

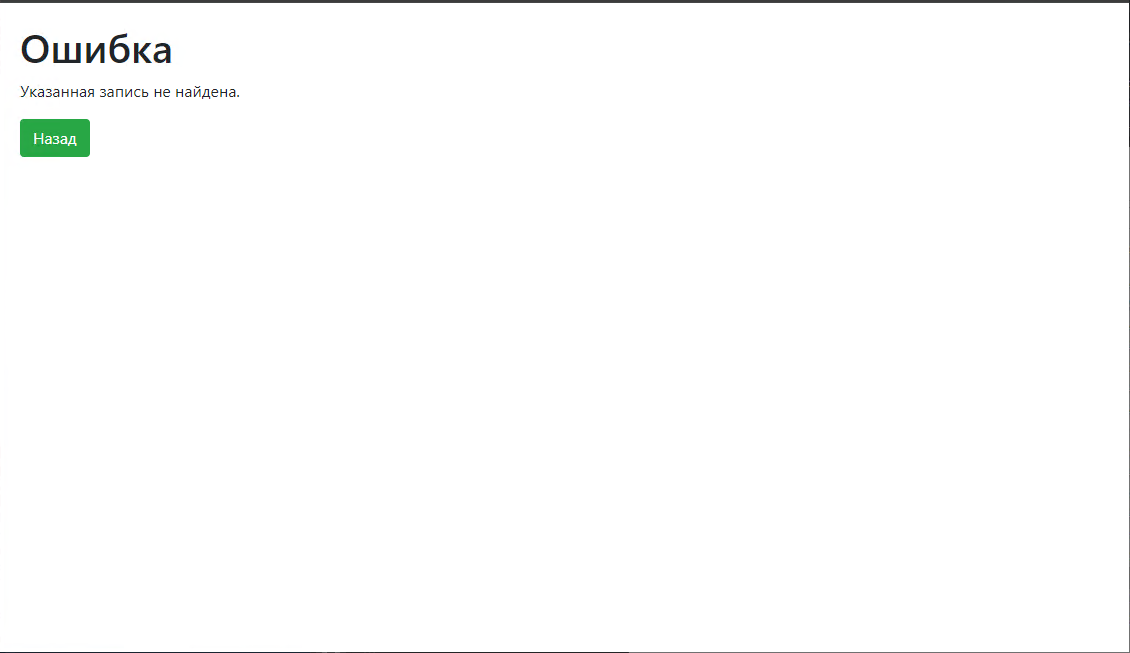


Рисунок 30 – Страница отображения ошибок

На странице указывается информация о возникшей ошибке, после ознакомления с которой можно нажать кнопку «назад» и вернуться на страницу выбора типа прозвона.

**4 Возможные ошибки и рекомендации по их устранению**

В случае возникновения ошибок, пользователю необходимо выполнять действия, описанные в таблице 2.

Таблица 1 – Рекомендации по устранению ошибок

|  |  |
| --- | --- |
| Описание ошибки | Рекомендации по устранению |
| Указанная запись не найдена | Проверить правильность введенного ID записи в сервисе «звонобот». |
| Всплывающее уведомление с текстом: «Не удалось загрузить исходящие номера. Попробовать снова?» | Нажать кнопку ОК на уведомлении. Если после нескольких попыток номера не будут загружены, следует обратиться к системному администратору. |