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

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)
КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

Студент Варин Дмитрий Владимирович ИУ7-26Б

Студент Ивахненко Дмитрий Александрович ИУ7-26Б

Студент Ларин Владимир Николаевич ИУ7-24Б

Студент Сироткина Полина Юрьевна ИУ7-26Б

Студент Трошкин Николай Романович ИУ7-26Б

Тип практики Проектно-технологическая практика

Название предприятия МГТУ им. Н.Э. Баумана

2020 г.

Оглавление

Введение

Формулировка цели работы: Разработать автоматизированную систему распределения сообщений от клиентов по менеджерам посредством телеграмм бота.

Участники практики:

- Ковалева Дарья - ментор
- Ларин Владимир - тимлид, тестировщик
- Трошкин Николай - разработчик интерфейсов взаимодействия с БД
- Ивахненко Дмитрий - разработчик интерфейсов взаимодействия с БД
- Сироткина Полина - разработчик интерфейсов взаимодействия с пользователем
- Варин Дмитрий - разработчик интерфейсов взаимодействия с пользователем

Аналитика рынка

На рынке есть острая потребность в простой коммуникации бизнеса и клиентов компании. Любой руководитель компании хочет сделать процесс общения достаточном простым и прозрачным, как для себя, так и для своего клиента. При этом вопрос упрощения общения может способствовать решению ряду других ключевых проблем бизнеса. Простота и удобство общения повысит лояльность клиентов, а следовательно будет виден небольшой прирост продаж. Такая система может решить вопрос хранения «тикетов» клиентов (проблем, вопросов), что в будущем поможет более полно проанализировать потребности клиентов компании и улучшить свой бизнес-процесс.

Использование и применение чат-систем очень логично из-за высокой популярности мессенджеров и потребности оказания услуг онлайн. Тем более в режиме «удалёнки» прямое контактирование невозможно, поэтому проект решает еще и эту проблему.

Проанализировав потребности рынка, мы выделили основные направления развития проекта.

Основные функции нашего проекта заключаются в:

- Обработке сообщений клиентов – менеджеры отвечают на разные вопросы клиентов
- Распределении сообщений между менеджерами – упрощаем процесс поиска свободного менеджера
- Хранении и обработке кейсов компании – администратор сможет проанализировать обращения клиентов

Идеальный покупатель системы: развивающийся бизнес проект в сфере услуг. Наша система должна решать следующие кейсы бизнеса: продажи услуг, ведения клиентов, поддержки продуктов бизнеса.

Для просты разработки конкретизируем процесс взаимодействия с продуктом:

- Клиент пишет в чат-бот
- Система находит свободного менеджера
- Менеджер отвечает на вопрос клиента
- Проблема клиента решена

Конструкторские решения

Данный проект рассчитан для использования как клиентами системы, так и менеджерами решающих их проблемы и вопросы. Система делит пользователей на три основные роли

- Клиент - целевой пользователь системы, задает вопросы и сообщает о своих проблемах
- Менеджер - сотрудник компании, занимающийся поддержкой и ведением клиентов. Основная задача сотрудника - отвечать на вопросы клиентов и решать их проблемы.
- Администратор системы - сотрудник компании, занимающийся настройкой системы

Назначение ролей

Перед началом работы с ботом, необходимо назначить хотя бы одного администратора. Поэтому приняли решение о назначение администратором первого зарегистрировавшегося пользователя.

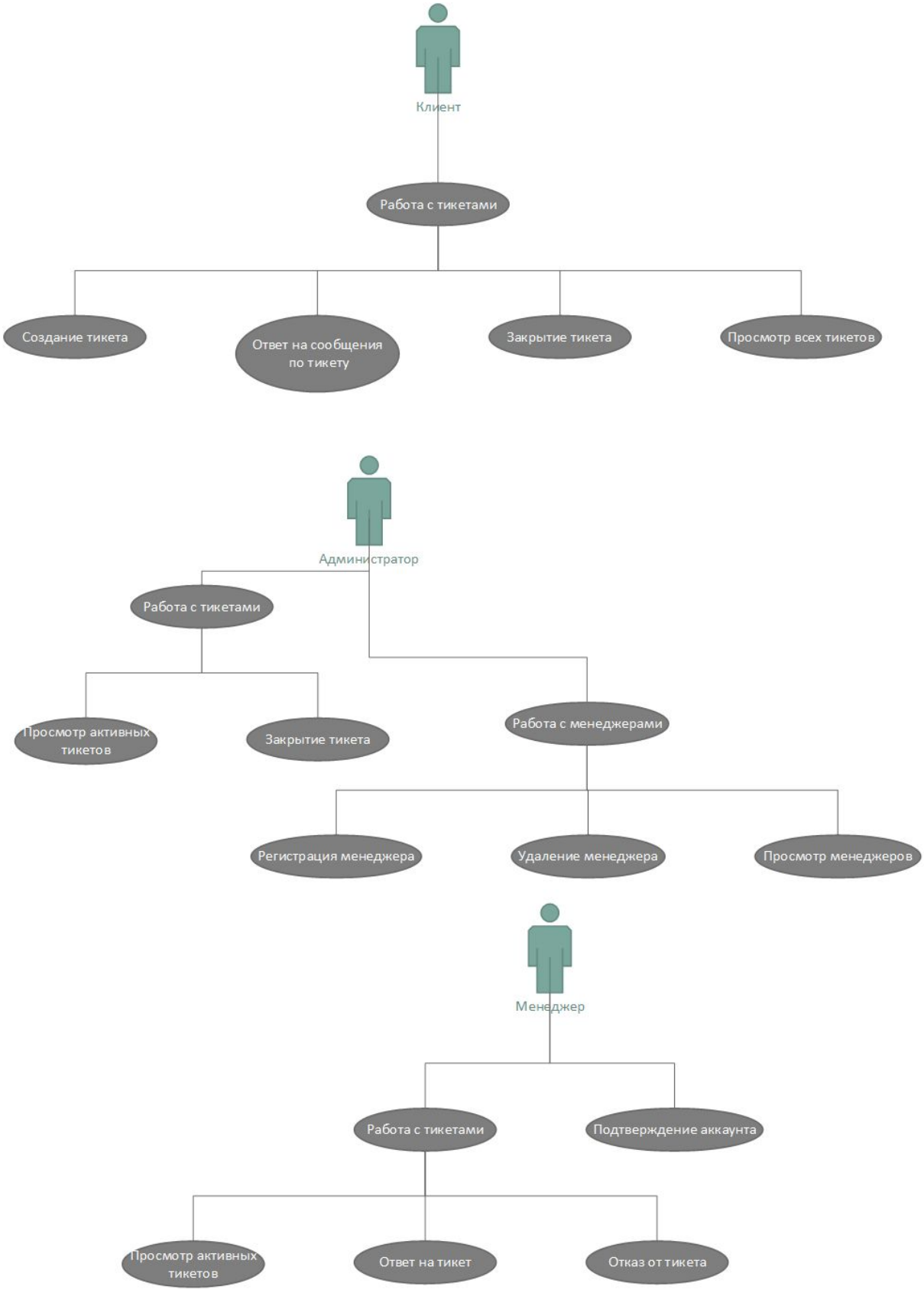
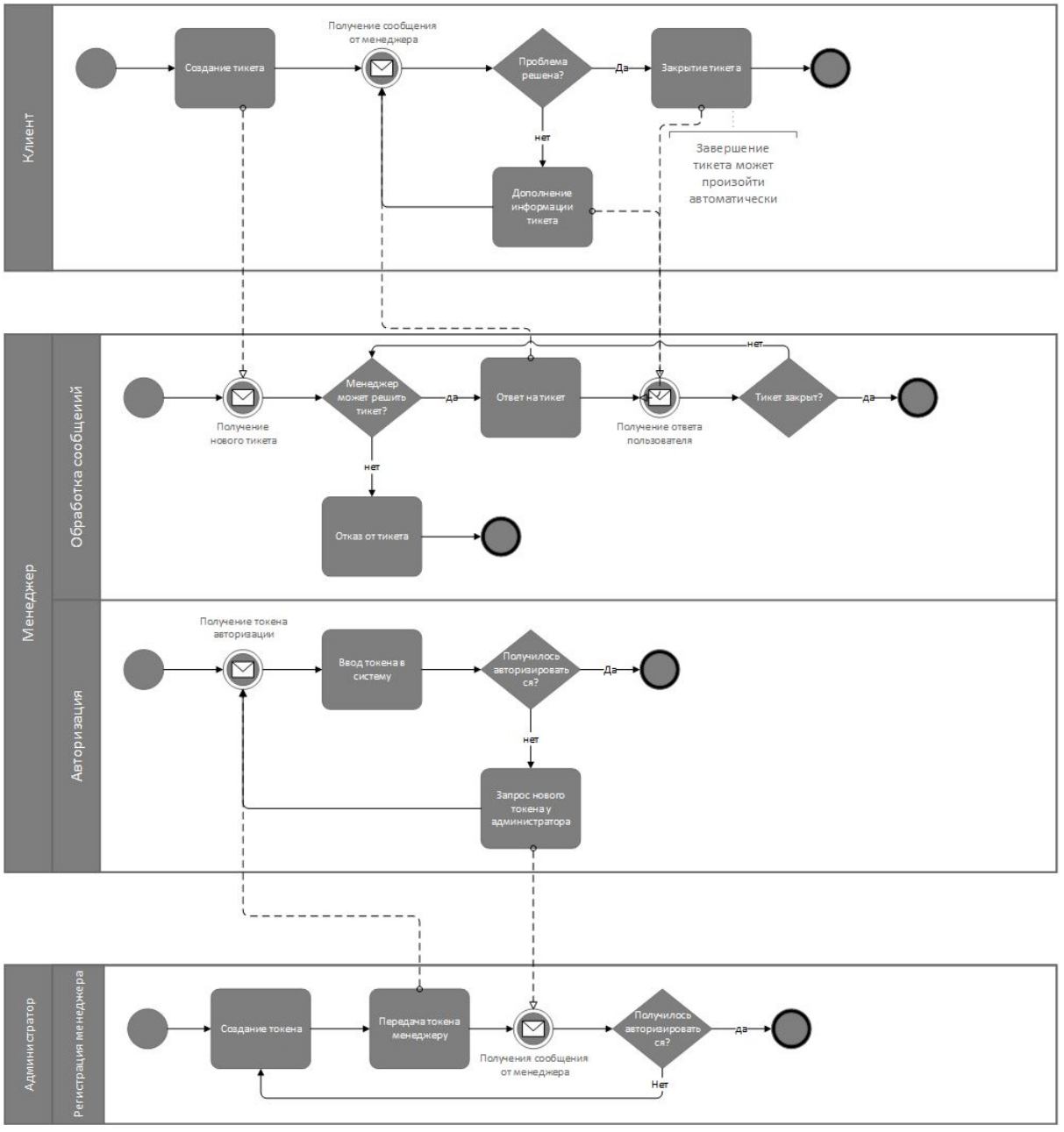
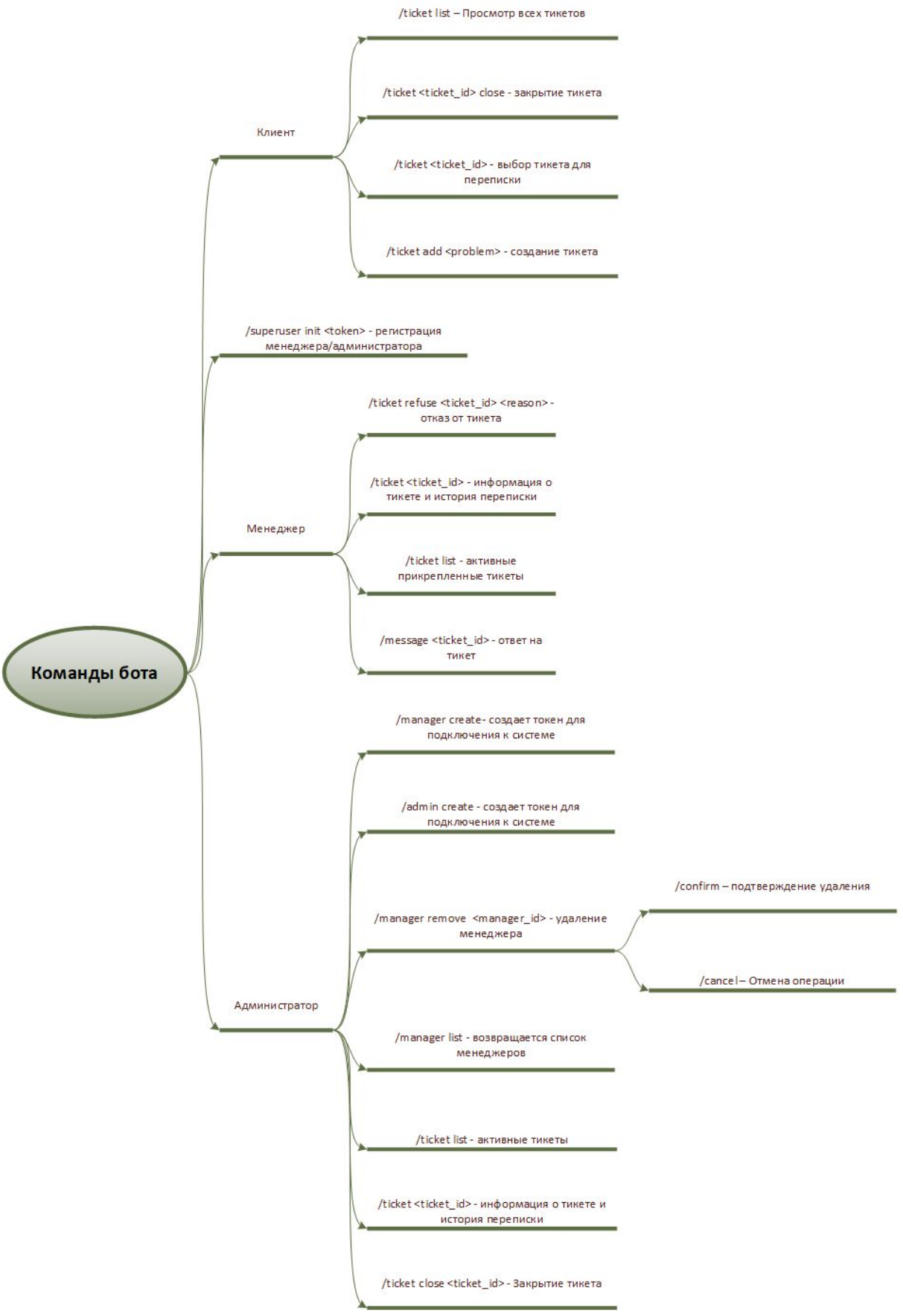

Клиент
Работа с тикетами
Создание тикета
Ответ на сообщения по тикету
Закрытие тикета
Просмотр всех тикетов
Администратор
Работа с тикетами
Просмотр активных тикетов
Закрытие тикета
Работа с менеджерами
Регистрация менеджера
Удаление менеджера
Просмотр менеджеров
Менеджер
Работа с тикетами
Подтверждение аккаунта
Просмотр активных тикетов
Ответ на тикет
Отказ от тикета

Диаграмма процесса взаимодействия




Команды бота
Клиент
/ticket list – Просмотр всех тикетов
/ticket <ticket_id> close - закрытие тикета
/ticket <ticket_id> - выбор тикета для переписки
/ticket add <problem> - создание тикета
/superuser init <token> - регистрация менеджера/администратора
Менеджер
/ticket refuse <ticket_id> <reason> - отказ от тикета
/ticket <ticket_id> - информация о тикете и история переписки
/ticket list - активные прикрепленные тикеты
/message <ticket_id> - ответ на тикет
Администратор
/manager create- создает токен для подключения к системе
/admin create - создает токен для подключения к системе
/manager remove <manager_id> - удаление менеджера
/confirm – подтверждение удаления
/cancel – Отмена операции
/manager list - возвращается список менеджеров
/ticket list - активные тикеты
/ticket <ticket_id> - информация о тикете и история переписки
/ticket close <ticket_id> - Закрытие тикета

Техническое описание команд

T1	Команды бота
T1.1	Администратор
T1.1.1	/manager list - возвращается список менеджеров
T1.1.2	/manager remove <manager_id> - удаление менеджера Данная операция происходит с подтверждением, при удалении менеджера все его открытые тикеты передаются другим менеджерам.
T1.1.3	/manager create - создает токен для подключения к системе Токен уникальный и действует 24 часа
T1.1.4	/ticket list - активные тикеты Тикеты должны быть оформлены в виде слайдера
T1.1.5	/ticket <ticket_id> - информация о тикете и история переписки
T1.1.6	/ticket close <ticket_id> - Закрытие тикета Если <ticket_id> не существует, то выводится ошибка
T1.2	Клиент
T1.2.1	/ticket add <problem> - создание тикета
T1.2.2	/ticket list – Просмотр всех тикетов Тикеты должны быть оформлены в виде слайдера
T1.2.3	/ticket <ticket_id> close - закрытие тикета
T1.2.4	/ticket <ticket_id> - выбор тикета для переписки. Произошел ОТКАЗ ОТ ДАННОЙ КОМАНДЫ. Подробнее в отчете о разработки взаимодействия с бд

T1.3	Менеджер
T1.3.1	/manager init <token> - регистрация менеджера
T1.3.2	/message <ticket_id> - ответ на тикет
T1.3.3	/ticket list - активные прикрепленные тикеты
T1.3.4	/ticket <ticket_id> - информация о тикете и история переписки
T1.3.5	/ticket refuse <ticket_id> <reason> - отказ от тикета

Распределение нагрузки

Чтобы определить нагрузку менеджера используются следующие критерии:

- Критерий №1
  - k = количество необработанных тикетов (последнее сообщение в тикете было от клиента)
  - q = 2
- Критерий №2
  - k = количество открытых тикетов
  - q = 1
- Критерий №3
  - k = количество обработанных (закрытых) тикетов за 24 часа, деленное на продолжительность этого периода
  - q = 1
- Критерий №4
  - k = количество отказов от тикетов за последние 7 дней
  - q = -1

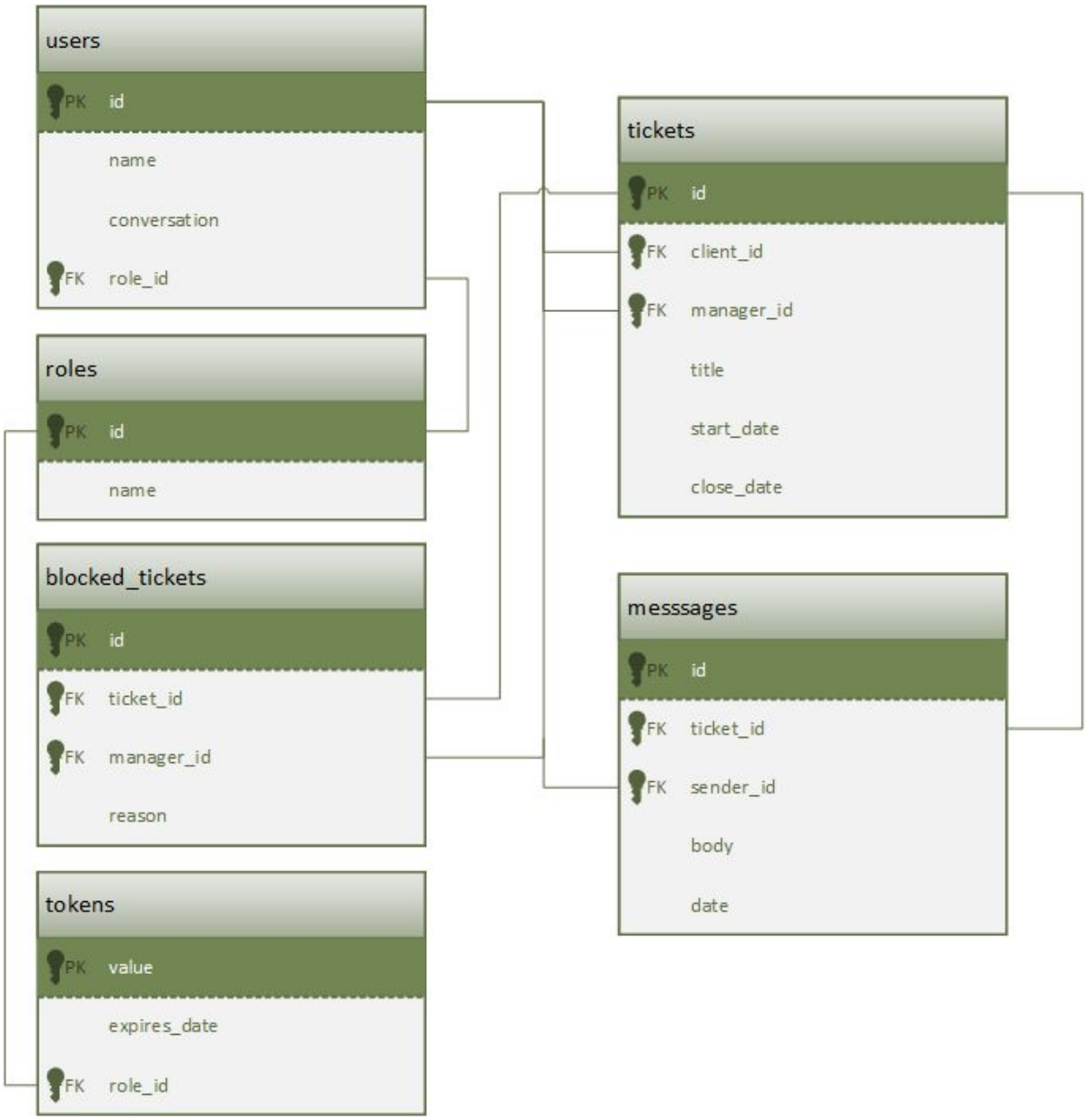
На основании этих критериев система выбирает нужного менеджера.

Выбирается менеджер с наименьшим результирующим коэффициентом, рассчитываемый по формуле

$$K = \prod_{i=1}^{n} (k_i)^{q_i}$$

, где k - критерий, q - важность критерия, n - количество критериев

Структура таблиц баз данных



Концепция тестирования бота

Так как тестирование - очень важный этап разработки любого программного продукта, подошли к этому вопросу с разных сторон. Бот - это в первую очередь взаимодействие с пользователем, а не программным интерфейсом. Варианты ответа бота нельзя зафиксировать (нельзя выделить эталон), так как бот отвечает в зависимости от имеющихся данных БД. Выделили два основных способа тестирования:

- Пользовательское тестирование

    - Тестирование интерфейсов взаимодействия
    - Поиск орфографических ошибок и проблем взаимодействия
- Программное Тестирование
    - Написание тестовых данных
    - Написание программы для вызова команд телеграмм бота

В пользовательском тестировании визуально происходит проверка работоспособности бота и анализ простоты взаимодействия. А программно вызываем команды бота согласно набору тестовых данных:

- вход - команда, подаваемая боту
- выход - ключевая фраза, которая должна содержаться в ответе

Технология разработки проекта

Используемые технологии

- Bot API by telegram
- Python 3
- SQL DB (преимущественно MySQL или MariaDB)
- telebot https://github.com/eternnoir/pyTelegramBotAPI
- SQLAlchemy ORM

Используем мессенджер Телеграмм, потому что он обладает удобными и гибкими интерфейсами проектирования ботов и высокой популярностью мессенджера. В качестве языка для ведения разработки приняли Python 3, потому что с ним знаком каждый участник практики. Чтобы упростить разработку логики бота использовали библиотеку pyTelegramBotAPI, имеющую множество примеров и достаточно информативную документацию. Использовали MySQL систему управления базами данных, так как она работает на языке запросов SQL, который достаточно распространенный и имеет достаточную скорость обработки запросов. Для упрощения взаимодействия с бд и автоматической генерацией SQL запросов использовали SQLAlchemy ORM, которая помогла создать удобные классы сущностей БД.

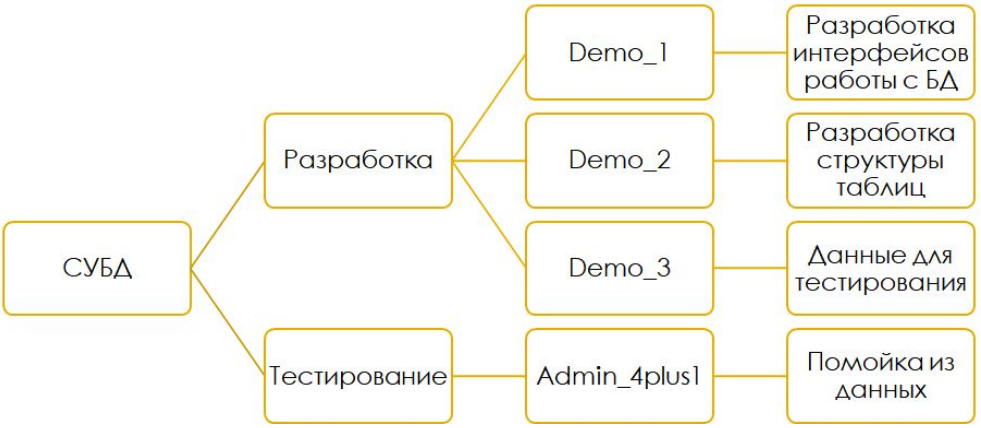
Работа в команде

Для работы в команде использовали кафедральную систему контроля версий GIT. Изначально выделили три основные состояния проекта.

- релиз версия
- альфа версия
- разработка проекта

Самое сложное состояние - это состояние разработки проекта, так как над ним трудятся несколько человек. Изначально было принято решение один участник разработки - одна ветка, но данный подход оказался неудачным на первых днях, так как участники постоянно вносили новые изменения в разные модули, и одни модули зависели от других, что заставляло постоянно вливать чужие ветки в свои, в итоге приняли решение объединить все свои ветки в одну общую и работать в ней. А разные фичи (если на них останется время) ветвить от данной ветки.

Чтобы все участники практики не были привязаны к единому серверу, создали дополнительные таблицы баз данных, помимо основной.



Чтобы предотвратить перезаписывание настроек подключения к разным телеграмм ботам, разным таблицам бд, прокси выделили файлы настроек, которые не вносили под версионный контроль. И в него вынесли все вариативности работы с системой.

Основные проблемы командной разработки:

- участники сразу не могли определиться, чем хотят заниматься, поэтому возникаль внутренние конфдикты интересов, проблема была решена тем, что каждый попробовал все по чуть чуть и определился со своей ролью в проекте

- разный взгляд на 1 проблему - каждый участник смотрит на ту или иную проблему со своей стороны и видит удачным только свое решения. Решали такие проблемы общими обсуждениями вопросов и пытались урегулировать конфликты.

Взаимодействие с БД

Для взаимодействия с БД было решено использовать SQLAlchemy, которая реализует технологию программирования ORM. Благодаря этому, создавать структуры базы данных и взаимодействовать с ними можно прямо на языке Python.

Было решено использовать декларативный метод работы с базой данных, который позволяет еще сильнее упростить работу с БД. При помощи declarative_base() создается родительский класс, от которого наследуются все остальные. Таким образом, для создания таблицы достаточно просто создать класс, наследовать его от declarative_base() и объявить необходимые поля класса, которые станут столбцами таблицы, а также расписать Relationships между таблицами, если это требуется.

Однако, несмотря на кажущуюся простоту в работе с БД, благодаря ORM, столкнулись с множеством проблем. Главное проблемой было постоянное “падение” бота по причине утери соединения с базой данных. Пытаясь разрешить данную проблему, попробовали несколько разных коннекторов, используемых для подключения непосредственно к MySQL, кроме того пробовали изменять запросы в БД, которые могли стать причиной проблемы.

Однако истинной причиной падений оказалась постоянно открытая сессия (соединение) с БД. В течение wait_timeout секунд не поступало никаких запросов в базу и она обрывала соединение с SQLAlchemy, которая в свою очередь сообщала о потере соединения с базой данных. Чтобы решить данную проблему, внедрили промежуточные слои PyTelegramBot. В начале запроса бота открывалась сессия, после обработку запроса она закрывалась, что предотвратило падения.

Кроме этого, в процессе разработки постоянно возникала необходимость что-то менять в структуре таблиц. Так, например, из-за изначально неверного выбора типа данных или значения по умолчанию возникали проблемы с корректностью заполнения столбцов с датой и временем. Также по мере реализации логики взаимодействия пользователей с ботом возникала необходимость добавлять или изменять различные методы классов, при помощи которых осуществлялось взаимодействие с БД.

Столкнулись с проблемой, когда потребовалось писать метод, возвращающий тикеты, не обработанные менеджером. Сложность заключалась в том, что не удалось целиком разобраться в принципе работы метода join непосредственно самой ORM, так как в Интернете не нашлось наглядных примеров использования. Решение - из сообщения данные о тикете можно вытащить с помощью "Relationships" и не использовать join. Это решение породило новую проблему: пришлось организовывать обработку в цикле, что существенно отразилось на времени работы метода. Однако на практике остроту проблемы мы ощутили слишком поздно, поэтому оптимизация алгоритма, скорее всего, не войдет в начальный релиз.

Взаимодействие с Ботом

Разработку начали со знакомства с интерфейсом библиотеки pyTelegramBotApi и ее возможностями. Сначала научились регистрировать пользователя в таблице базы данных и определять зарегистрированного пользователя по его номеру чата телеграм. Во время реализации более сложных команд возникали проблемы "однопроходности" бота. Решили её с помощью register_next_step_handler. Далее начали разработку согласно техническому заданию.

Чтобы упростить взаимодействие пользователя с ботом решили заменить использование inline-команд на клавиатуры с кнопками-действиями. Во время разработки удобного пользовательского интерфейса возникли некоторые трудности. Inline команды после сообщения не заменяли обычную клавиатуру и таким образом клиент мог нажать кнопки повторно, что приводило к выходу за рамки штатного режима. Поэтому мы поменяли тип клавиатуры с Inline на Reply(текстовая клавиатура), таким образом клавиатура становится одноразовая и часть действий клиента мы предопределили за него.

Самой главной была проблема выбора "рабочего" тикета. По ТЗ предполагалось, что клиент будет выбирать один тикет из активных с помощью команды /ticket_id, но это порождало много ошибок, необходимость хранить логи команд и восстанавливать последовательность событий. Также пока по такой концепции открыт один тикет, другие не могут быть выбраны, и менеджеры не могли бы в них писать. Им приходилось бы ждать, когда освободится клиент, "караулить" его, а это неэффективно: поэтому мы отказались от понятия рабочего тикета и команды /ticket_id(того, что было вложено в неё в начале, по крайней мере) и сделали все Тикеты рабочими. Таким образом, у менеджеров будет больше возможностей в плане ответа, а клиент прочитает ответ тогда, когда ему будет удобно (ему придёт уведомление)

В процессе написания бота возникало много неясных моментов, один из которых - предобработчик событий middleware_handler, он использовался не по назначению, через него передавали переменные для переиспользования в других функциях, что порождало множество багов: значения не передавались далее, а перекрывались новыми от других пользователей, использующих бота - решили передачей аргументов через вышеупомянутый register_next_step_handler (подробнее прочитав документацию, оказалось, что существует возможность передачи через данный обработчик аргументов).

Организация тестирования

Для начала дополнили тестовые данные, которые изначально были рассчитаны на только взаимодействие с ботом посредством команд. Написали модуль тестирования, который эмулировал клиент телеграм и от имени введенного аккаунта передавал сообщения боту. Модуль использует библиотек Telethone.

Тестовые данные представили в виде JSON файла, чтобы могли менять тестовые случаи, не меняя логику тестирования. В логику бота добавили дополнительные команды,которые только доступны в режиме отладки. Их назначение -принудительно переключать роль пользователя.

Развертывание продукта

Наш продукт - бот - изначально размещался на локальном сервере (на наших компьютерах). Каждый разработчик запускал его у себя, используя прокси - сервер посредник, так как сервис Telegram заблокирован в России. Это неудобно и непрактично - для использования бота нужно 24/7 поддерживать активное соединение. Было решено развернуть бота на внешнем сервере за территорией РФ. Мы решили использовать heroku. Причина проста - есть возможность сделать это бесплатно, достаточно ввести несколько команд и скачать оболочку взаимодействия Heroku CLI, и бот размещен на сервере. Также CLI позволяет отслеживать “логи” продукта, что помогает отслеживать непредвиденные ошибки. Все это упрощает процесс написания кода и его отладки.

Также для работы бота необходим сервер управления базами данных. С ним не возникло никаких проблем, так как он уже был развернут одним из участников для своего личного проекта. Сервер расположен на территории РФ, что поможет исполнять наше законодательство в рамках федерального закона о Персональных данных. Добавили необходимое количество таблиц на развернутый сервер и начали работу с ними.

Заключение

В заключение проанализируем проделанную работу. Первый этап разработки - проектирование оказался весь успешным, многие проблемы были продуманы заранее. Команда работала на протяжении всего времени практики, вносила коррективы в работу и вполне удачно подошла к дедлайну. Удалось реализовать планируемый функционал.

В планах упростить интерфейс взаимодействия пользователя и бота, добавить больше интерактивных элементов и довести проект до программного продукта.

Практика позволила попробовать командную разработку, обдумывание логики и решение конфликтов интересов при создании алгоритма взаимодействия бота - несколько разработчиков, каждый из которых порождает свою идею, и, следовательно, свое видение проекта - практика закончилась, а опыт остался с нами.

Литература

1. https://core.telegram.org/bots/api
2. Object Relational Tutorial
3. eternnoir/pyTelegramBotAPI: Python Telegram bot api.
4. https://habr.com/ru/
5. https://stackoverflow.com/