**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра технологий программирования**

**Разработка прикладной интеллектуальной системы с использованием ансамбля различных математических моделей машинного обучения**

Курсовой проект

Бусыгина Владимира Дмитриевича

студента 3 курса, 8 группы

специальность

«прикладная математика»

Научный руководитель:

заведующий кафедры ТП,

доктор технических наук,

профессор

Курбацкий Александр Николаевич

Минск, 2021

**АННОТАЦИЯ**

Бусыгин В.Д. Разработка прикладной интеллектуальной системы с использованием ансамбля различных математических моделей машинного обучения: Курсовой проект / Минск: БГУ, 2021. - 33 стр.

В данной работе подробно описывается процесс создания бота на платформе Telegram, для последующей возможности добавления моделей машинного обучения.

**АНАТАЦЫЯ**

Бусыгін В.Д. Распрацоўка прыкладной інтэлектуальнай сістэмы з выкарыстаннем ансамбля розных матэматычных мадэляў машыннага навучання: Курсавы праект / Мінск: БДУ, 2021. - 33 стар.

У дадзенай працы падрабязна апісваецца працэс стварэння бота на платформе Telegram, для наступнай магчымасці дадання мадэляў машыннага навучання.

**ANNOTATION**

Busygin V. D. Development of an applied intelligent system by using an ensemble of different mathematical models of machine learning : Course project / Minsk : BSU, 2021. - 33 p.

This paper describes in detail the process of creating a bot on the Telegram platform, for the subsequent possibility to add machine learning models.

**РЕФЕРАТ**

Курсовой проект, 33 с., 6 рис., 4 источников.

***Ключевые слова*** : БОТ, ДИСПЕТЧЕР, ОБРАБОТЧИК СООБЩЕНИЙ, AIOGRAM, КОНЕЧНЫЙ АВТОМАТ СОСТОЯНИЙ, ORM, SQLITE.

***Объект исследования -*** библиотека для реализации бота aiogram, методы построения архитектуры многопользовательского приложения.

***Цель работы*** - реализовать необходимый функционал бота на платформе Telegram, реализовать получение первичных данных о пользователе, связать полученные данные с базой данных приложения.

***Методы исследования*** - программирование на языке python, библиотека для реализации бота aiogram, библиотека для объектно-реляционной модели peewee.

***Результаты***:

Подробно описан процесс программной реализации бота, разработан удобный пользовательский интерфейс для работы с ботом, механизмы взаимодействия реализуемого приложения с сервером Telegram, использование базы данных и создание объектно-реляционной модели для работы с базой данных, обработка сообщений от пользователей, создание конечного автомата состояний для последовательного опроса пользователя.

**Содержание**

ВВЕДЕНИЕ 5

**ГЛАВА 1 СОЗДАНИЕ И ВЕДЕНИЕ БОТА НА ПЛАТФОРМЕ TELEGRAM** 7

1.1. Ведение проекта разрабатываемого приложения 7

1.2. Платформа размещения приложения 8

1.3. Регистрация бота на сервере Telegram 8

1.4. Сокрытие ключа доступа 9

1.5. Библиотека aiogram 9

1.6. Программная реализация бота 11

**ГЛАВА 2 ОБРАБОТКА СООБЩЕНИЙ ПОЛЬЗОВАТЕЛЕЙ** 14

2.1.1. Пользовательская клавиатура 14

2.1.2. Программная реализация пользовательской клавиатуры 15

2.2. Принцип работы обработчика сообщений 15

2.3. Программная реализация обработчиков сообщений 17

2.4.1. Постановка задачи сбора информации о пользователе 18

2.4.2. Программная реализация конечного автомата 19

2.4.3. Валидация и сбор персональных данных пользователя 20

**ГЛАВА 3 РАБОТА С БАЗОЙ ДАННЫХ** 24

3.1. Выбор базы данных 24

3.2. Программная реализация ORM 24

3.3. Программная реализация работы с базой данных 25

ЗАКЛЮЧЕНИЕ 26

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 28

ПРИЛОЖЕНИЯ 29

Приложение А 29

Приложение Б 30

Приложение В 31

Приложение Г 33

**ВВЕДЕНИЕ**

В современном мире ведущую роль занимают технологии и без них представить современного человека практически невозможно. Стремительное развитие науки порождает новые технологии, которым находят применение в различных сферах жизни человека. От положения науки в стране, зависит ее экономическое, политическое состояние и фактический статус в мировом сообществе. Непосредственное влияние на ее развитие оказывает образование.

Возможность получить соответствующее образование в современном мире есть почти каждого у человека и для этого необходимо окончить государственное или же иное учебное заведение, но помимо данного способа получения знаний не менее важным является самообразование. В этом случае у человека имеется множество ресурсов в интернете для извлечения тех или иных знаний, однако в этом случае учителем или же помощником будет выступать сам человек. И на качество полученных знаний будет влиять уровень самостоятельно подобранного материала, его актуальность и последовательность изученного материала.

Разрабатываемое приложение выступает в качестве персонального помощника в процессе самообразования. Приложение нацелено на решение задач персонализированного подбора информации для пользователя в области программирования. Под персонализированный подбором выступает алгоритм подбора информации в зависимости от уровня знаний и количества опыта пользователя. В качестве данного алгоритма подбора информации будет выступать совокупность различных моделей машинного обучения для работы с различными типами данных.

Само приложение представляет из себя бота-помощника, расположенного в одной из самых популярных систем обмена сообщений Telegram. Выбор такого формата приложения и системы размещения обусловлен доступностью, простотой обслуживания, безопасностью, а также скоростью взаимодействия между приложением и пользователем.

В главе 1 подробно описывается механизм взаимодействия приложения с сервером платформы размещения - Telegram. Также описывается программная реализация и размещение бота на платформе.

В главе 2 подробно описываются способы обработки сообщения и механизм работы обработчика сообщений. Также описывается процесс реализации конечного автомата состояний для последовательного опроса пользователя, получение данных и их хранение в каждом состоянии.

В главе 3 описывается процесс реализации базы данных, её выбор и связывание с приложением с использованием ORM.

**Глава 1**

**СОЗДАНИЕ И ВЕДЕНИЕ БОТА НА ПЛАТФОРМЕ TELEGRAM**

Разрабатываемое приложение подразумевает под собой работу с множеством пользователей, главная функция которого затрагивает множество типов задач. Для такого приложения важно правильно подобрать форму, платформу размещения, а также составить подходящий каркас для реализации поставленной цели.

**1.1. Ведение проекта разрабатываемого приложения**

Любое разрабатываемое приложение требует аккуратного, правильного и согласованного ведения проекта. Наименование переменных, классов, функций, модулей и других объектов ведется согласно правилам, описанным в конвенции об именах в языке Python. Данный документ не является обязательной частью в разработке, однако позволяет придерживаться понятия чистого кода в процессе разработки. Проектирование приложения осуществлялось с использованием модулей, схожие по своим функциям или связанные по своей логике модули размещались в соответствующей им директории, в необходимых случаях формировались пакеты из соответствующих модулей. Основной методологией программирования в приложении является объектно-ориентированное программирование с применением функциональных подходов.

Разрабатываемое приложение включает в себя несколько функциональных частей : работа с базой данных содержащую в себе таблицы с информацией о пользователях, их прогрессе в обучении и полученных знаниях, обработчики сообщений пользователей, логгирование происходящих процессов, работа с моделями машинного обучения.

**1.2. Платформа размещения приложения**

Приложение должно быть размещено в какой-либо системе для доступа пользователя, выбор системы обмена сообщений Telegram, как платформы в которой функционирует приложение, обусловлен простотой доступности для пользователя, безопасностью и конфиденциальностью.

Главной задачей приложения является оказание помощи обучающемуся в процессе обучения и для удобства взаимодействия между приложением и пользователем формой приложения был выбран бот. Это позволяет работать приложению в автономном режиме и в случае необходимости оказания помощи обучающемуся позволяет выполнить необходимый запрос к приложению через удобный пользовательский интерфейс.

**1.3. Регистрация бота на сервере Telegram**

Перед началом программной реализации приложения необходимо зарегистрировать бота на сервере Telegram. Для этого необходимо обратиться к BotFather - бот Telegram, предназначенный для управления ботами и их различными настройками. Регистрируем бота, а для этого вводим ему уникальное имя и заполняем другие необходимые данные.

Теперь зарегистрированного бота можно найти в общем поиске Telegram по уникальному имени, которое указывалось при регистрации, также после окончания регистрации пользователь Telegram, который регистрировал бота получает access-token необходимых для управления зарегистрированным ботом.

Access-token - уникальный набор символов для доступа к HTTP API,

После всех необходимых действий мы получаем зарегистрированного бота и ключ доступа для управления им и переходим к программной реализации приложения.

**1.4. Сокрытие ключа доступа**

Поскольку ключ доступа является главным рычагом управления ботом, то в таком случае при публичной демонстрации или хранении программного кода, другой человек получает возможность управления зарегистрированным ботом, поскольку значение ключа хранится непосредственно в самом проекте. Для избежания утечки ключа доступа в сеть, существует несколько способов его сокрытия.

В первом случае мы можем игнорировать файл глобальной конфигурации, но в этом случае приходится хранить шаблон конфигурационного файла и не забывать периодически его обновлять.

В другом случае, мы можем хранить скрываемую информацию в переменных окружения, с этой целью можно воспользоваться одной из утилит Python - direnv.

Выбор подхода к хранению глобальной конфигурации остается за администратором приложения, однако стоит придерживаться последнего случая для избежания случайной утечки ключа доступа.

**1.5. Библиотека aiogram**

Для взаимодействия разрабатываемого приложения со сторонним приложением, в данном случае сторонним приложением является Telegram в котором и будет расположено приложение, необходимо использовать API. Telegram использует свой собственный протокол шифрования - называемым MTProto и с помощью MTProto API приложение Telegram связывается с сервером. Надстройкой над MTProto API является Telegram Bot API, который несет в себе конкретную часть общего функционала Telegram, а именно создание и поддержка ботов на платформе Telegram.

Для функционирования бота необходимо чтобы он связывался с сервером Telegram и получал от него обновления. Существует несколько способов получения обновлений от сервера: получение от сервера запросов по определенному URL-адресу и регулярный запрос к серверу. В таком случае возникает необходимость ручного написания http-запросов и для облегчения задачи разработки приложения разумно использовать фреймворк для Telegram Bot API.

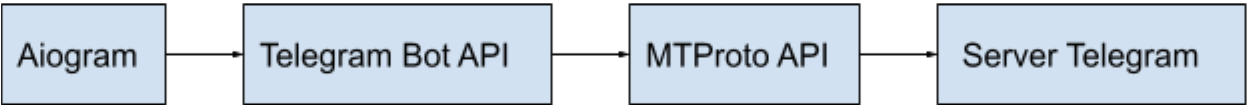


Рисунок 1.1 - Схема взаимодействия aiogram с сервером Telegram

Существует множество библиотек на языке программирования Python для упрощения работы с Telegram Bot API, однако выбранной библиотекой для каркаса разрабатываемого приложения является aiogram. Aiogram имеет документацию лишь на английском языке, кроме документации информации о ней крайне мало, однако выбор ее в качестве основной библиотеки для работы с ботом обусловлен принципом ее работы.

Преимущество данной библиотеки над другими заключается в принципе ее работы, а именно асинхронности. Приложение непосредственно работает с базой данных, вносит изменения, добавляет или удаляет информацию, также занимается обработкой запросов от пользователей, которые в свою очередь требуют времени из-за применения различных алгоритмов и работы моделей машинного обучения над ними. Поскольку приложение включает в себя задачу обучения вместе с пользователем, то помимо обработки запроса и выдачи определенного результата, оно должно внести в базу данных соответствующую информацию об обучении пользователя. Для самого пользователя важен результат работы алгоритма, однако для корректной работы приложения необходимо выполнить определенную работу с базой данных, результат которой для пользователя не важен, но требует время на ее выполнение.

При синхронном подходе реализации программного кода каждая операция требует ожидания окончания предыдущей операции, в таком случае пользователь тратит свое время на ожидания операций, которые выполняются друг за другом. Асинхронный подход реализации программного кода позволяет устранить проблему траты времени от ожидания сторонних процессов, вместо ожидания ответа от сервера, пользователь получает возможность продолжать работу с приложением, в то время как его запрос обрабатывается в другом процессе, и после обработки запроса пользователь получает ответ от сервера.

Таким образом выбирая асинхронную библиотеку aiogram для реализации приложения мы получаем оптимизацию затраченного времени пользователя, соответственно более эффективное приложение.

**1.6. Программная реализация бота**

Создаем в проекте файл с глобальной конфигурацией бота - configuration.py, в нем создаем константы, которые будут хранить параметры конфигурации бота: ACCESS\_TOKEN, TIMEZONE и другие. Следующим шагом создается ядро бота, для него создаем файл core.py в который импортируем необходимые модули из библиотеки aiogram и также переменные глобальной конфигурации.

Класс Bot используется для разделения сетевого интерфейса от всех методов API. Объявляется переменная bot, которая является экземпляром класса Bot в конструктор которого которого мы передаем ключ доступа ACCESS\_TOKEN, полученный при регистрации бота.

Класс Dispatcher используется для получения обновлений и их обработки. Объявляется переменная dispatcher, которая является экземпляром класса Dispatcher в конструктор которого которого мы передаем бота, также в нем выбираем в качестве хранилища состояний в памяти класс MemoryStorage.

Далее создаем файл с которого будет начинаться запуск приложения - start\_polling.py. В него импортируются все модули в которых будет происходить обработка ответов на сообщения пользователей, также импортируем ранее созданный dispatcher.

Как было упомянуто ранее, боту необходимо связываться с сервером Telegram и получать обновления от него для нормального функционирования. Поскольку в большинстве случаев ответы от сервера будут неинформативными, то запрашивать обновления каждую секунду не будет эффективным подходом. В таком случае эффективным подходом, которое использует разрабатываемое приложение, будет long-polling, при котором приложение регулярно отправляет запросы на сервер и получает ответ в случаях, когда приходит новое сообщение или по необходимости разрыва соединения.

И далее для запуска приложения в режиме long-polling требуется вызов метода у aiogram.executor.start\_polling(), в параметры которого передается ранее созданный dispatcher, который будет принимать и обрабатывать обновления от сервера Telegram. Теперь при запущенном файле start\_polling.py, ранее созданный бот будет принимать сообщения от пользователей и отвечать на них в предусмотренных случаях.

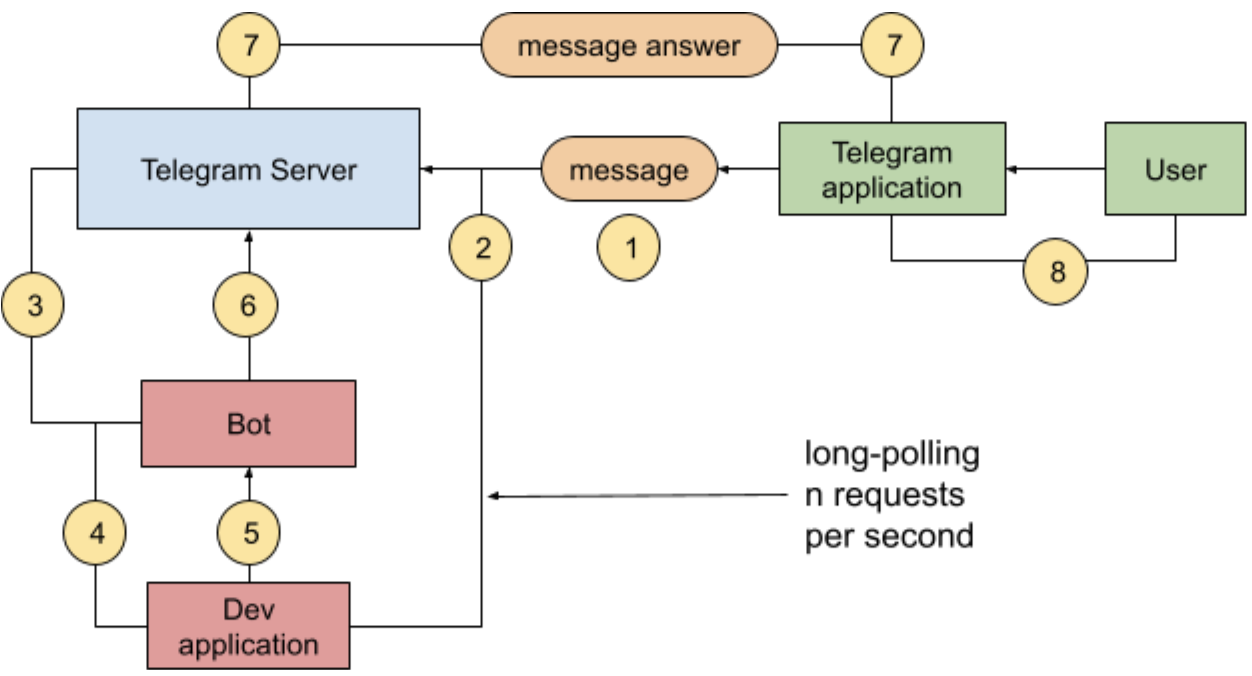


Рисунок 1.2 - Схема обмена сообщениями между пользователям и приложением

**ВЫВОДЫ**

1.Выбрана платформа размещения

2.Выбран формат приложения

3.Изучен механизм взаимодействия приложения с платформой.

4.Изучена и применена библиотека aiogram для реализации бота.

5.Реализован и размещен функционирующий бот на платформе Telegram.

**Глава 2**

**ОБРАБОТКА СООБЩЕНИЙ ПОЛЬЗОВАТЕЛЕЙ**

Важную часть в разрабатываемом приложении являются сообщения пользователя. Именно через них происходит взаимодействие между пользователем и приложением. Задача состоит в том, чтобы разобраться, как извлекать данные из сообщений пользователей, как построить процесс удобного взаимодействия между пользователем и приложением.

**2.1.1. Пользовательская клавиатура**

По умолчанию пользователь не имеет никакой клавиатуры, что ставит его в положение ручного набора команд. Для упрощения работы пользователя с Telegram-ботом необходимо продумать удобную клавиатуру для чата, а для этого на каждом этапе взаимодействия с ботом нужно сопровождать пользователя клавиатурой с командами.

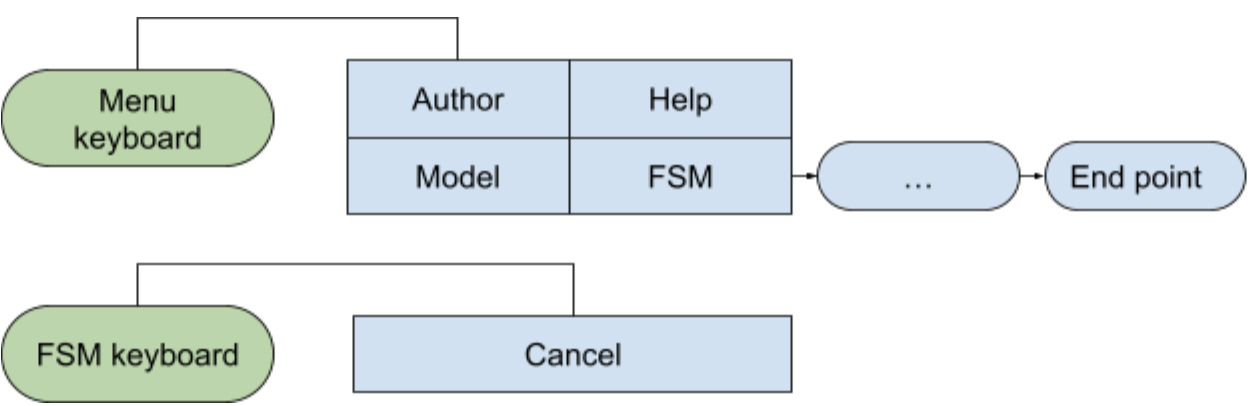


Рисунок 2.1 - Представление клавиатуры для различных состояний

Для многих состояний клавиатура будет оставаться неизменной и чтобы не повторять один и тот же программный код будем придерживаться принципа DRY - Don’t repeat yourself и создадим отдельный класс для работы с пользовательской клавиатурой.

**2.1.2. Программная реализация пользовательской клавиатуры**

Создаем в проекте файл keyboard.py и импортируем в него из aiogram.types KeyboardButton, ReplyKeyboardMarkup, InlineKeyboardMarkup. KeyboardButton представляет собой класс кнопки в конструкторе, которой мы указываем текст означающий нажатие кнопки. ReplyKeyboardMarkup - шаблон сообщений, который передается в качестве параметра при ответе пользователю. Данный тип кнопок позволяет лишь ввести команду пользователю при нажатии на них. InlineKeyboardMarkup - также является шаблоном сообщений, но в отличии от ReplyKeyboardMarkup позволяет крепить клавиатуру к сообщению и программно задавать действие при нажатии на кнопки этой клавиатуры. Также можно задать необходимые размеры кнопок и их расположение на клавиатуре.

Класс клавиатуры содержит в себе приватные поля, которые представляю собой кнопки с заданным текстом. Методы класса представляют собой функции, которые возвращают необходимый шаблон сообщений с набором кнопок внутри. Все методы обозначены декоратором @classmethod, который говорит о том, что методы используют атрибуты своего класса, данные методы могут быть вызваны без создания экземпляра самого класса.

**2.2. Принцип работы обработчика сообщений**

Пользователь все время взаимодействует с приложением через чат бота и имеет возможность отправлять в чат различные по типу и составу данные: ссылки на ресурсы, фотографии, текст, видео. Пользователь отправляет сообщение, оно приходит на сервер Telegram и после отправки запроса обновлений приложения на сервер Telegram сообщение приходит приложению где его требуется обработать. Диспетчер получает сообщение и решает какой из обработчиков сообщений будет работать с ним.

Обработчик сообщений - функция, которой диспетчер поручает работу с поступившим сообщением.

Чтобы диспетчер нашел необходимый обработчик сообщений, нужно указать обработчику сообщений с какими типами сообщений он работает. Можно выделить несколько типов сообщений: команда выполнения боту, сообщение, не являющееся командой, сообщение от пользователя находящегося в определенном состоянии и комбинации этих вариаций. Таким образом в зависимости от комбинации данных критериев, диспетчер определяет какой из обработчиков сообщений примет поступившее сообщение.

Сообщение с командой боту обозначается символом “/” перед самой командой, например, /start, /help, /author. В таком случае диспетчер ищет обработчик сообщений для которого указана данная команда. По своей функциональности команды служат для запуска какого-либо процесса с состояниями или же совершения однократного действия. По умолчанию при начале работы с ботом начальной командой является /start и разрабатываемом приложении на данную команду бот реагирует также, как и на команду /help, которая выводит список всех команд для пользователя. Команды /author, /about выводят информацию о разработчике и проекте соответственно.

В случае, когда пользователь отправляет сообщение, которое не является командой, диспетчер ищет обработчик сообщений в котором указано, что представляет из себя полученное сообщение.

И последний случай, когда диспетчер получает сообщение от пользователя находящегося в определенном состоянии и определяет, что в таком состоянии необходимо обработать сообщение определенным образом. Эти состояния можно задавать и сбрасывать пользователю, комбинировать в различные цепочки обработок сообщений.

Таким образом диспетчер проходит по всем обработчикам сообщений и ищет первого обработчика, для которого указано, что он работает с заданным типом сообщения.

В случае, когда несколько обработчиков могут принять одно и то же сообщение, отработает лишь первый идущий, причем обработчики идущие в программном коде за ним не примут сообщения. Такой принцип работы обработчиков задает их определенную программную реализацию.

В случае, когда сообщению не было найдено соответствующего обработчика в разрабатываемом приложении предусмотрена реплика пользователю. Если же в приложении не предусмотрен такой случай, то бот не выдаст никакого сообщения.

**2.3. Программная реализация обработчиков сообщений**

Создаем в проекте директорию handlers, в которой будут храниться файлы с непосредственно реализацией обработчиков сообщений. Данные файлы импортируются в ядро бота - core.py. В файлы обработчиков сообщений импортируем необходимые модули и классы, а именно нам потребуется класс кастомной клавиатуры, диспетчер из ядра бота - core.py, который и принимает от сервера Telegram сообщения. Также в проекте присутствует файл commands\_information.py, который хранит в себе переменные с подробным описанием каждой команды боту, данный файл также импортируем в файлы обработчиков сообщений. Также файлы с обработчиками сообщений должны содержать в себе класс состояний пользователя, который будет описан позже.

Функции, занимающиеся обработкой сообщений необходимо обозначить декоратором:

@dispatcher.message\_handler().

Декоратор - это функция, позволяющая обернуть другую функцию без изменения ее программного кода, с целью расширения ее функциональности.

В самом декораторе мы указываем по какому критерию происходит выбор функции для обработки поступившего сообщения. Поскольку используемая библиотека для программной реализации бота aiogram является асинхронной, то при объявлении функции занимающейся обработкой сообщения необходимо обозначать ключевым словом async, которое указывает приложению, что данную функцию требуется выполнить в асинхронном режиме. Также при работе с библиотечными объектами требуется указывать перед ними ключевое слово await.

Конечный автомат состояний - Finite-state machine, абстрактная модель содержащая конечное число состояний, при котором лишь одно состояние может быть активным.

**2.4.1. Постановка задачи сбора информации о пользователе**

Необходимо последовательно собрать первичную информацию о пользователе и провести проверку введенных данных на каждом шаге чтобы подобрать параметры для персонального обучения. Решение данной задачи с помощью привычных команд становится невозможным, поскольку ставится вопрос какой из обработчиков сообщений использовать для работы с пришедшим сообщением. Для эффективного решения поставленной задачи существует машина состояний. Мы можем явно указать состояние и если пользователь находится в определенном состоянии, то диспетчер будет искать обработчик сообщений подходящий для этого состояния и в таком случае мы построим конечный автомат состояний для опроса пользователя.

**2.4.2. Программная реализация конечного автомата**

Создаем файл state.py и имортируем из aiogram.dispatcher.filters.state классы StatesGroup и State в него. В данном файле будет реализован класс, хранящий состояния пользователя, также при реализации данного класса необходимо унаследовать его от StatesGroup. Атрибуты класса представляют собой каждое состояние пользователя и являются объектами класса State.

Далее создаем обработчики сообщений для каждого состояния и указываем в декораторе принадлежность обработчика к данному состоянию:

@dispatcher.message\_handler(state=wait\_for\_name)

При отправки боту команды опросить пользователя, устанавливается начальное состояние и далее по цепочке в каждом состоянии устанавливается состояние, следующее за ним, таким образом осуществляется перемещение между вопросами и по окончанию процесса текущее состояние сбрасывается и пользователь перемещается в начальное меню приложения.

Установка состояния: States.wait\_for\_name.set()

Сброс состояния: States.wait\_for\_name.finish()

При этом чтобы получить данные пользователя в нужном формате необходимо проводить валидацию введенных данных на каждом шаге. Если пользователь вводит неверные данные, то он соответственно не перемещается на следующих шаг, а остается в текущем состоянии. Однако, чтобы пользователь не застрял на каком-то шаге, его сопровождают текстовые сообщения о необходимом формате данных, а также дается возможность преждевременного завершения опроса и возврата в начальное меню, при котором ранее введенные данные сбрасываются.

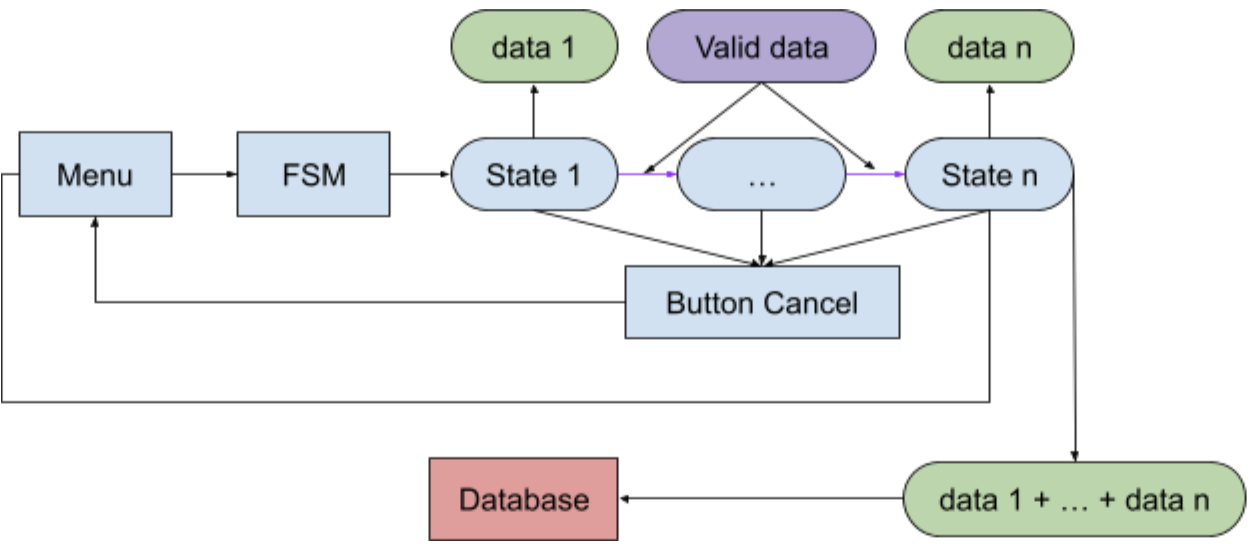


Рисунок 2.2 - Схема конечного автомата

**2.4.3. Валидация и сбор персональных данных пользователя**

На каждом этапе ввода пользователем данных, происходит валидация для получения необходимых параметров. Извлечение текста из сообщения пользователя происходит посредством обращения к поля класса сообщения:

message : aiogram.types.Message

user\_text : str = message.text

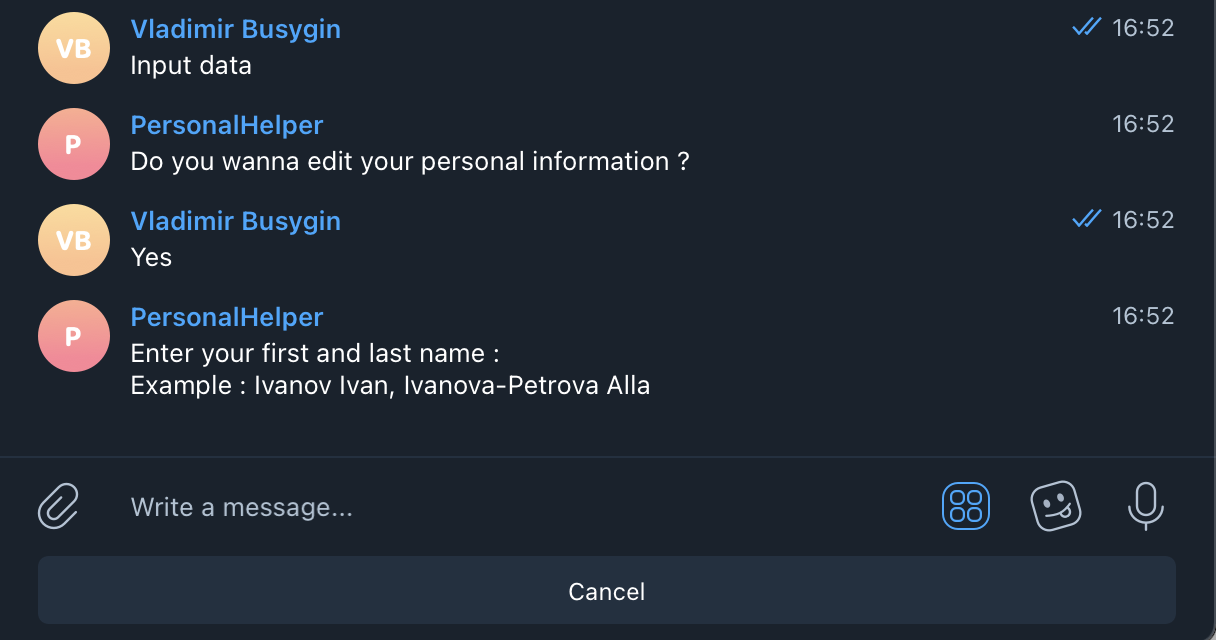
Помимо извлечения данных из сообщения необходимо также сохранять эти данные на каждом шаге, для этого в аргументе обработчика сообщения указываем дополнительный параметр, который должен являться объектом класса aiogram.dispatcher.FSMContext через которого мы получаем данные от бэкэнда конечного автомата. И на каждом шаге обновляем его данные :

state : FSMContext, message : aiogram.types.Message

await state.update\_data(user\_text=message.text)

После окончания опроса извлекаем данные методом state.get\_data() и сохраняем их в базу данных соответствующему пользователю.

**2.5. Демонстрация работы приложения**

Рисунок 2.3 - Переход в цепочку состояний автомата

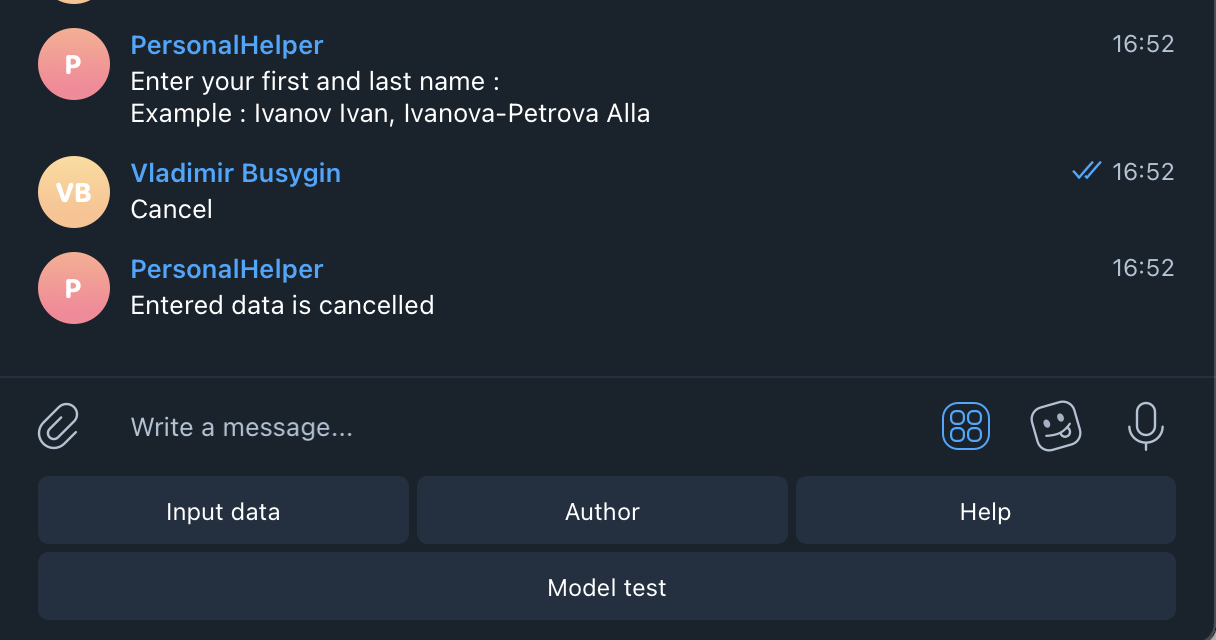


Рисунок 2.4 - Сброс состояния на этапе ввода данных

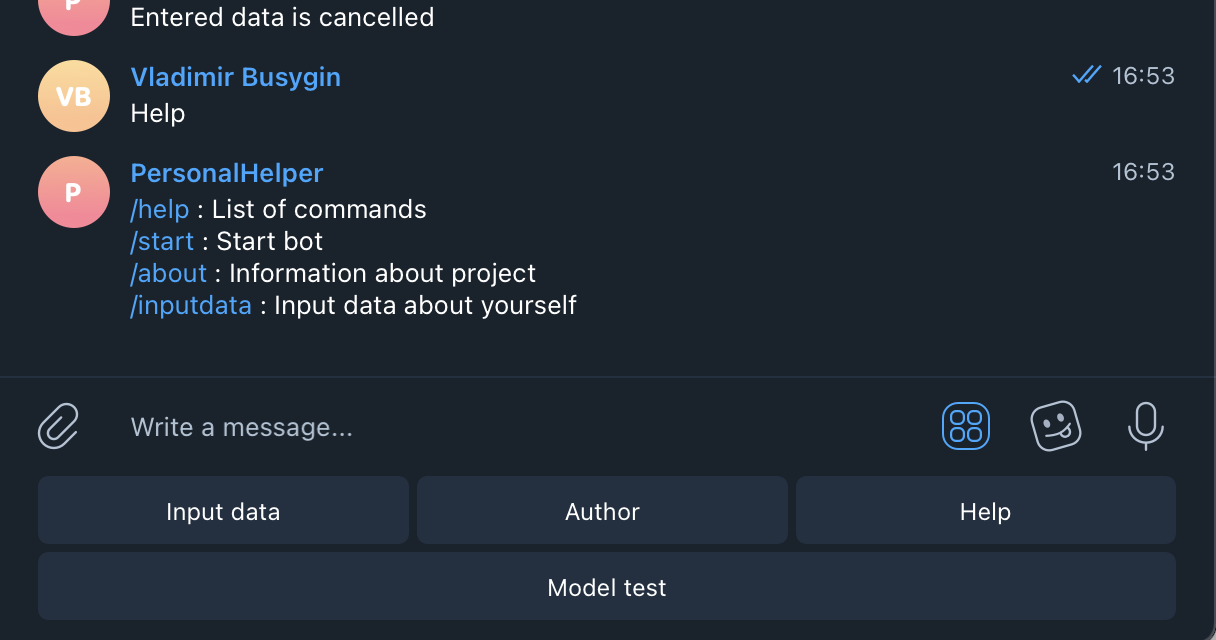


Рисунок 2.5 - Отправка команды /help при нажатии на клавиатуру

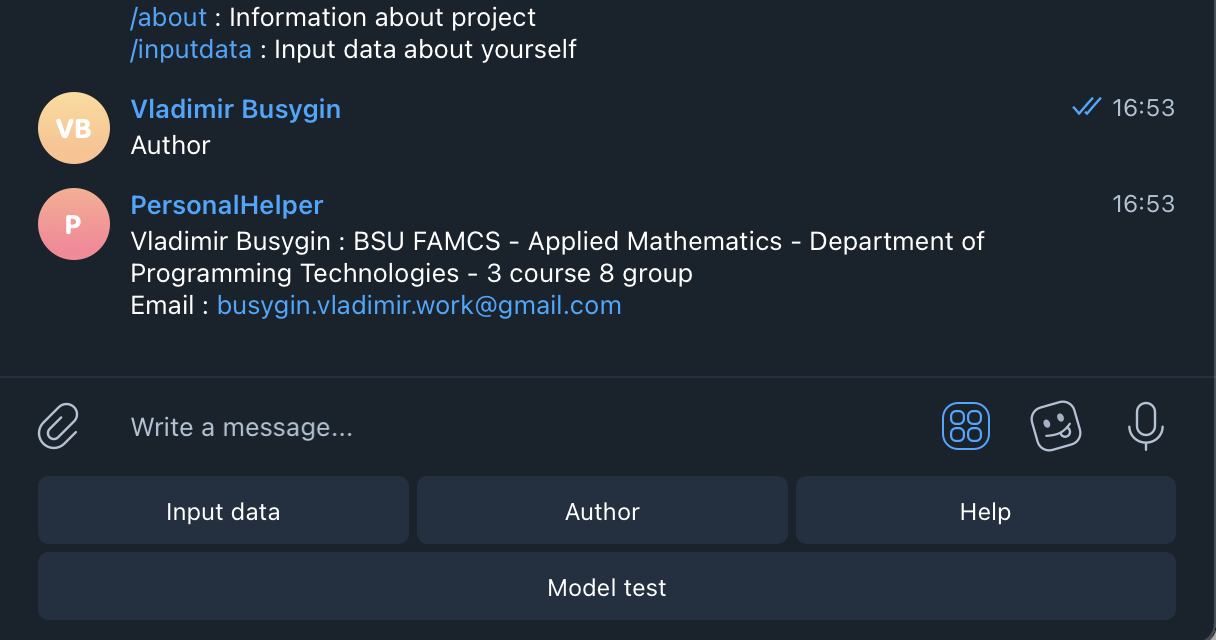


Рисунок 2.6 - Отправка команды /author при нажатии на клавиатуру

**ВЫВОДЫ**

1.Разработан и реализован удобный пользовательский интерфейс.

2.Изучен и реализован механизм обработки сообщений пользователя.

3.Изучен и реализован механизм конечного автомата состояний.

4.Изучен и реализован механизм сохранения и извлечения данных из сообщений.

**Глава 3**

**РАБОТА С БАЗОЙ ДАННЫХ**

**3.1. Выбор базы данных**

Первоначальной задачей ставится выбор базы данных для хранение первичной информации о пользователях и ведение их учета. Для решения данной задачи в приложении была использована база данных SQLite.

SQLite - компактная, встраиваемая система управления базами данных, поддерживает практически полный набор команд SQL.

Хранение первичной информации о пользователях и их учет ведется в единой таблице, уникальным идентификатором является id пользователя, который мы извлекаем в процессе заполнения данных пользователем.

**3.2. Программная реализация ORM**

Чтобы обращаться к базе данных, добавлять, изменять, удалять данные в таблицах необходимо писать вручную запросы к ней. Для упрощения данного способа обращения в приложении была использована ORM - peewee.

ORM - объектно реляционная модель, технология программирования, которая обеспечивает работу с таблицами в рамках классов.

Для реализации данной технологии создаем директорию database, которая будет хранить в себе файлы для работы с различными базами данных. Создаем в директории файл sqlite\_database\_models.py и импортируем из peewee класс Model.

В этом файле будут реализовываться классы различных моделей, работающих с SQLite.

Создаем в файле sqlite\_database\_models.py класс BaseModel, который наследуется от класса Model и в который мы прописываем метаданные для таблицы, а конкретнее путь к базе данных. Далее модели для работы с таблицами наследуем от класса BaseModel и объявляем атрибуты классов моделей в соответствии с их типами в таблицах.

**3.3. Программная реализация работы с базой данных**

В директории database создаем файл sqlite\_database\_handler.py в который импортируем необходимые модули для работы с ORM peewee, а также ранее созданные классы моделей из файла sqlite\_database\_models.py.

В данном файле будет реализован класс для работы с базой данных SQLite, который мы назовем SQLiteDatabase, методы которого объявляются как @classmethod, таким образом мы сможем пользоваться методами без объявления экземпляра класса. Методы данного класса представляют собой соответствующие запросы к базе данных.

Таким образом теперь мы можем записывать данные о пользователях после прохождения команды заполнения данных и извлекать их из базы данных для дальнейшей работы с ними.

**ВЫВОДЫ**

1.Выбрана база данных для хранения первичных данных о пользователях.

2.Изучена и реализована объектно-реляционная модель для базы данных SQLite.

3.Реализованы методы записи, чтения и редактирования данных в базе данных SQLite.

**ЗАКЛЮЧЕНИЕ**

Современные технологии являются неотъемлемой частью нашей жизни. С развитием науки появляются все больше технологий, и планка требований знаний к образованному человеку, чтобы быть наравне с волной прогресса, становится все выше. Помимо стандартных путей образования, человеку необходимо совершенствоваться самостоятельно. С развитием технологий, человек получает все больше возможностей самостоятельно обучаться чему-либо и спрос на образовательные приложения возрастает.

Сейчас почти каждый человек имеет современный девайс с выходом в интернет и почти каждый из них использует для общения какие-либо соцсети. Удобный формат приложения и его расположение в совокупности с растущей потребностью в самообразовании, позволит повысить качество образования и облегчит задачу обучения при использовании персонального бота-помощника.

В ходе выполненной работы были рассмотрены и решены следующие задачи:

1.Выбор платформы для размещения приложения, регистрация и размещение бота на платформе.

2.Детально изучен механизм взаимодействия приложения с платформой и пользователем.

3.Изучена библиотека aiogram для реализации бота на платформе Telegram.

4.Реализованы и изучены механизмы работы с сообщениями пользователя.

5.Разработан и реализован удобный интерфейс взаимодействия пользователя с приложением.

6.Реализован и изучен принцип работы конечного автомата состояний.

7.Реализован сбор первичной информации о пользователях с валидацией введенных данных пользователем.

8.Реализованы методы для чтения, записи, редактирования данных о пользователях при работе с базой данных SQLite.

9.Реализован и изучен принцип работы объектно-реляционной модели.

10.Реализовано приложение с частью требуемого функционала.

11.Для реализации приложения было использована среда разработки Pycharm.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1.Асинхронное программирование в Python [электронный ресурс] : https://habr.com/ru/company/ruvds/blog/475246/

2.peewee documentation [электронный ресурс] : http://docs.peewee-orm.com/en/latest/

3.aiogram documentation [электронный ресурс] : https://docs.aiogram.dev/en/latest/

4.sqlite3 documentation [электронный ресурс] : https://docs.python.org/3/library/sqlite3.html

**ПРИЛОЖЕНИЯ**

*Приложение A*

**from** core **import** dispatcher

**from** aiogram **import** executor

**import** importlib

importlib.import\_module(**'handlers.user\_data\_handler'**)

importlib.import\_module(**'handlers.message\_handler'**)

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

executor.start\_polling(dispatcher)

*Приложение Б*

**from** aiogram.contrib.fsm\_storage.memory **import** MemoryStorage

**from** aiogram **import** Bot, Dispatcher

**import** config

bot = Bot(token=config.ACCESS\_TOKEN)

dispatcher = Dispatcher(bot, storage=MemoryStorage())

*Приложение В*

**from** aiogram **import** types

**from** core **import** dispatcher

**from** state **import** States

**from** keyboard\_handler **import** Keyboard

**import** commands\_information **as** info

@dispatcher.message\_handler(commands=**'start'**)

**async def** command\_start(message: types.Message) -> **None**:

**await** message.answer(text=info.START\_MESSAGE, reply\_markup=Keyboard.get\_start\_keyboard())

@dispatcher.message\_handler(commands=**'inputdata'**)

@dispatcher.message\_handler(**lambda** message: message.text == **'Input data'**)

**async def** command\_input\_data(message: types.Message) -> **None**:

**await** message.answer(text=**'Do you wanna edit your personal information ?'**,

reply\_markup=Keyboard.get\_yes\_or\_no\_keyboard())

**await** States.wait\_for\_edit\_personal\_information.set()

@dispatcher.message\_handler(commands=**'help'**)

@dispatcher.message\_handler(**lambda** message: message.text == **'Help'**)

**async def** command\_help(message: types.Message) -> **None**:

**await** message.answer(text=info.HELP)

@dispatcher.message\_handler(commands=**'author'**)

@dispatcher.message\_handler(**lambda** message: message.text == **'Author'**)

**async def** command\_author(message: types.Message) -> **None**:

**await** message.answer(text=info.AUTHOR)

@dispatcher.message\_handler(state=**'\*'**)

**async def** command\_test(message: types.Message) -> **None**:

**await** message.answer(text=**'This not command...'**, reply\_markup=Keyboard.get\_start\_keyboard())

*Приложение Г*

START\_MESSAGE = **'Welcome, if you want to see more information about this bot enter /help'**

ABOUT = **'\nPersonalHelper - telegram bot to solve the applied problem of content selection '** \

**'depending on the level of the user.'**

AUTHOR = **'\nVladimir Busygin : BSU FAMCS - Applied Mathematics - Department of Programming Technologies - '** \

**'3 course 8 group \nEmail : busygin.vladimir.work@gmail.com'**

HELP = **'\n/help : List of commands'** \

**'\n/start : Start bot'** \

**'\n/about : Information about project'** \

**'\n/inputdata : Input data about yourself'**