**Національний технічний університет України**

**„Київський політехнічний інститут** **імені Ігоря Сікорського”**

**Кафедра автоматизації проектування енергетичних процесів і систем**

***КУРСОВА РОБОТА***

**з дисципліни: «Бази даних»**

**тема: «Student Bot Assistant»**

|  |  |
| --- | --- |
| **Керівник**  **ст. викладач Колумбет В.П** | **Виконала Круть К.О.** |
| **Допущено до захисту** | **Студентка 2 курсу** |
| **«28» грудня 2021р.** | **Групи ТІ-01** |
| **Захищено з оцінкою** | **залікова книжка** |
| **\_\_\_\_\_\_\_\_95\_\_\_\_\_\_\_\_\_\_** | **№ ТІ-0114** |

**Київ – 2021**

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Кафедра автоматизації проектування енергетичних процесів і систем

**КУРСОВА РОБОТА**

з дисципліни «Бази Даних»

(назва дисципліни)

на тему: «Student Bot Assistant»

Студента групи \_\_\_\_\_\_\_\_\_\_\_\_ТІ-01\_\_\_\_\_\_\_\_\_\_\_\_\_

напряму підготовки 6.050103 Програмна інженерія

спеціальності 121“Інженерія програмного забезпечення”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Круть К.О. \_ \_\_\_\_\_\_ \_\_\_\_\_

(прізвище та ініціали)

Керівник ***\_\_\_\_\_\_\_\_\_\_***ст.вик. Дацюк О. А.\_\_\_\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_ Оцінка ECTS \_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ст. вик. Дацюк О. А.\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ ст. вик. Колумбет В. П.\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

Київ – 2021

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ТЕПЛОЕНЕРГЕТИЧНИЙ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра \_\_\_\_\_\_Автоматизації проектування енергетичних процесів і систем\_\_\_\_\_\_\_

Напрям підготовки \_\_\_\_\_\_\_\_\_6.050103 Програмна інженерія\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр і назва)

**ЗАВДАННЯ**

**НА КУРСОВУ РОБОТУ СТУДЕНТУ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Круть Катерині Олександрівні\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1. Тема роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ «Student Bot Assistant» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

керівник курсової роботи – \_\_\_\_\_\_\_\_\_ст. вик. Дацюк Оксана Антонівна\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи \_\_\_\_\_\_\_\_28.12.2021\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3.Вихідні дані до проекту(роботи): мова програмування Python, середовище розробки PyCharm, система керування базами даних – MySQL.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): розробити інформаційну систему «Student Bot Assistant», спроектувати базу даних.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів виконання курсової роботи | Строк виконання етапів роботи | Примітка |
| 1. | Затвердження теми роботи | 22.10.2021 |  |
| 2. | Вивчення та аналіз здачі | 22.10.2021 |  |
| 3. | Проектування структури бази даних | 23.10.2021 |  |
| 4. | Розробка алгоритму реалізації здачі | 03.11.2021 |  |
| 5. | Програмна реалізація здачі | 14.11.2021 |  |
| 6. | Тестування програми | 29.11.2021 |  |
| 7. | Оформлення пояснювальної записки | 19.12.2021 |  |

**Студент** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

**Керівник курсової роботи** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

**АНОТАЦІЯ**

Сьогодні, в час Цифрової революції особливий інтерес викликають програми, які були б здатні допомагати в повсякденному житті, роботи життя легшим та зручнішим. Хвиля  популярності ботів все більше і більше захоплює світ. Програма «Student Bot Assistant» призначена саме щоденної допомоги. Вона дозволяє налаштувати свій розклад, використовувати нотатки та організувати власні фінанси. Програма орієнтована на студентів, проте підходить всім користувачам, адже є простою у використані та гнучкою у налаштуваннях, має багатий функціонал

Код написаний мовою програмування Python та реалізований у середовищах розробки PyCharm за допомогою сервера баз даних MySQL.

Обсяг пояснювальної записки 75 аркушів, 1 додаток.

**ANNOTATION**

Nowadays, in the time of Digital Revolution, programs that would be able to help us in everyday life and make it easier and more comfortable continue to arouse particular interest. The wave of popularity of bots is capturing the world more and more. The program «Student Bot Assistant» is intended for daily assistance. It allow to set up your schedule, use notes and organize your own finances. The program is aimed at students, but is suitable for all users, because it is easy to use and flexible in settings, also because it has rich functionality.

The code is written in the Python programming language and implemented in PyCharm integrated development environment, using the MySQL database server.

The volume of the explanatory note is 75 sheets, 1 appendix.

**ЗМІСТ**

ВСТУП 6

РОЗДІЛ 1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ 9

1.1.Предметна область роботи 9

1.2. Основний функціонал 11

РОЗДІЛ 2. ОПИС БАЗИ ДАНИХ 12

2.1. Концептуальна модель бази даних 12

2.2. Фізична модель бази даних 14

2.3. Структура таблиць бази даних 15

РОЗДІЛ 3. ОПИС ПРОЕКТУ ПРОГРАМНОГО ПРОДУКТУ 18

3.1. Використане програмне забезпечення 18

РОЗДІЛ 4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ 20

4.1. Загальний опис програми 20

4.3. Опис інтерфейсу програми 19

ВИСНОВКИ 24

СПИСКИ ВИКОРИСТАНИХ ДЖЕРЕЛ 29

ДОДАТОК 1 30

# ВСТУП

В XXI столітті, відомому також як цифрове, світ – невпинно швидкозмінний, і ми живемо в його бурхливому темпі з щільним графіком, отримуючи щодня купу інформації та завдань. Життя студента особливо насичене, тому важко все пам’ятати, організувати та за всім слідкувати. Як і тисячі років тому, гроші відіграють надзвичайно важливу роль у житті кожного з нас. Щодня ми взаємодіємо з фінансами, витрачаємо, отримуємо кошти, і саме тому надзвичайно важливою для сучасної людини є фінансова грамотність, а саме уміння контролювати свої кошти, спланувати майбутнє, для досягнення фінансового добробуту. Важливим для кожного студента/школяра, або ж просто зайнятої людини є ведення розкладу на щодень, на тиждень чи навіть рік, ведення нотаток – та пошук необхідної інформації. Тягати купу записників, щоденників та переглядати нескінченну кількість книг на бібліотечних полицях немає сенсу, адже сьогодні, в час цифрової революції, нам в цьому можуть допомогти програми-помічники, особливо зручними є ті, що є гнучкими, доступними та не потребують додаткових завантажень на пристрої.

Саме тому реалізація цієї програми виконана у формі бота-помічника в одному із найбільш популярних у світі мессенджерів – Telegram (станом на 2021 рік Telegram використовує більше ніж 500 млн людей у всьому світі). Оскільки найзручніше писати подібні програми підлаштовані під платформу Telegram найзручніше мовою Python, тому для реалізації цієї програми обрано саме її та фреймворк Aiogram, що дозволяє забезпечити багатофункціональність бота, модуль pymysql забезпечує під’єднання програми до бази даних, що не обхідна для функціонування бота-помічника.

Для реалізації бази даних була обрана система керування реляційними базами даних MySQL корпорації Oracle. Система MySQL дає можливість легко створювати бази даних та працювати з ними.

Мета цієї курсової роботи полягає в отримані нових та збагаченні вже отриманих знань про бази даних, удосконаленні навичок роботи з СУБД. Також застосування цих знань та умінь для розробки концептуальної моделі та структури необхідної для програми бази даних, проектування інформаційної системи, звичайно, також програмна реалізація «Student Bot Assistant», в функціонал якого входить організація та контроль власних фінансів.

Завдання – написати програму, що дозволяла б керувати власним розкладом, створювати нотатки, шукати необхідні книги та записувати їх, управляти фінансами та організовувати їх, а саме додавати витрати/прибуток, переглядати їх, керувати бюджетом та переглядати статистику.

# РОЗДІЛ 1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

## **Предметна область роботи**

Для цієї курсової роботи предметною областю є бот-помічник, а фрагментами цієї ПО є «Нотатки», «Розклад» та «Фінанси». Кожен фрагмент цієї системи має свої об’єкти, функціональну область та процеси, їх перечислено нижче у таблиці 1.1

Таблиця 1.1. Опис функціональної області та її процесів

|  |  |
| --- | --- |
| Функціональна область | Процеси |
| Прибуток | Додавання прибутоку |
| Видалити прибутоку |
| Витрати | Дадавання витрат |
| Видалення витрат |
| Бюджет | Перегляд встановленого ліміту |
| Редагування встановленого ліміту |
| Категорія | Додавання категорії |
| Перегляд всіх категорій |
| Статистика | Перегляд статистику за тиждень |
| Перегляд статистику за місяць |
| Інше | Перегляд витрати за сьогодні |
| Перегляд витрати за тиждень |
| Перегляд витрати за місяць |
| Перегляд прибуток за сьогодні |
| Перегляд прибуток за тиждень |
| Перегляд прибуток за місяць |
| Групи | Додавання групи |
| Парсинг груп з сайту розклад КПІ |
| Редагування даних групи |
| Видалення групи |
| Викладачі | Додавання нового викладача |
| Парсинг викладачів з сайту розклад КПІ |
| Редагування даних викладача |
| Видалення викладача |
| Дисципліни | Додавання дисципліни |
| Редагування дисципліни |
| Видалення дисципліни |
| Аудиторії | Додавання аудиторії |
| Редагування даних аудиторії |
| Видалення аудиторії |
| Розклад | Відображення розкладу на сьогодні |
| Відображення розкладу на наступний навчальний день |
| Відображення наступної пари |
| Відображення розкладу на обидва тижні |
| Пари | Додавання пари |
| Редагування даних пари |
| Видалення пари |
| Тема | Додавання нової теми |
| Редагування теми |
| Видалення теми |
| Назва | Додавання нової назви |
| Редагування назви за темою |
| Видалення назви за темою |
| Текст нотаток | Додавання нової нотатки |
| Редагування редагування існуючої нотатки, за назвою та темою |
| Видалення доданої нотатки, за назвою і темою |
| Пошук доданої нотатки, для конкретного користувача, за ім’ям |
| Пошук доданої нотатки, для конкретного користувача, за темою |
| Вивід всіх, занесених конкретним користувачем, нотаток |

## **1.2. Основний функціонал**

Аналізуючи дані попередньої таблиці визначили основний функціонал створеної програми, доступний користувачеві:

* додавання розкладу
* редагування розкладу
* відображення розкладу на певний період часу
* створення нотаток
* пошук нотатки за певний критерієм
* редагування нотатки
* видалення нотатки
* додавання прибутку/витрат
* видалення прибутку/витрат
* додавання категорій
* перегляд всіх категорій
* перегляд статистики за певний період часу
* перегляд прибутку/витрат за певний період часу

# РОЗДІЛ 2. ОПИС БАЗИ ДАНИХ

**2.1. Концептуальна модель бази даних**

Концептуальна модель — модель предметної області, що складається з переліку взаємопов'язаних понять, що використовуються для опису цієї області, разом з властивостями й характеристиками, класифікацією цих понять, за типами, ситуацій, ознаками в даній області і законів протікання процесів в ній. Концептуальна модель відноситься дод моделей, які формуються після процесу концептуалізації або узагальнення. Концептуальні моделі часто є абстракціями речей в реальному світі [1].

Концептуальна модель бази даних – інформаційна модель найвищого рівня абстракції для ІС «Student Bot Assistant». Всі дані ІС вміщено в шістнадцяти таблицях: «Користувач», «Група», «Теми», «Нотатки», «Пара», «День тижня», «Тиждень», «Час пари», «Дисципліна», «Аудиторія», «Тип пари», «Викладач», «Бюджет», «Категорія», «Витрати», «Прибутки».

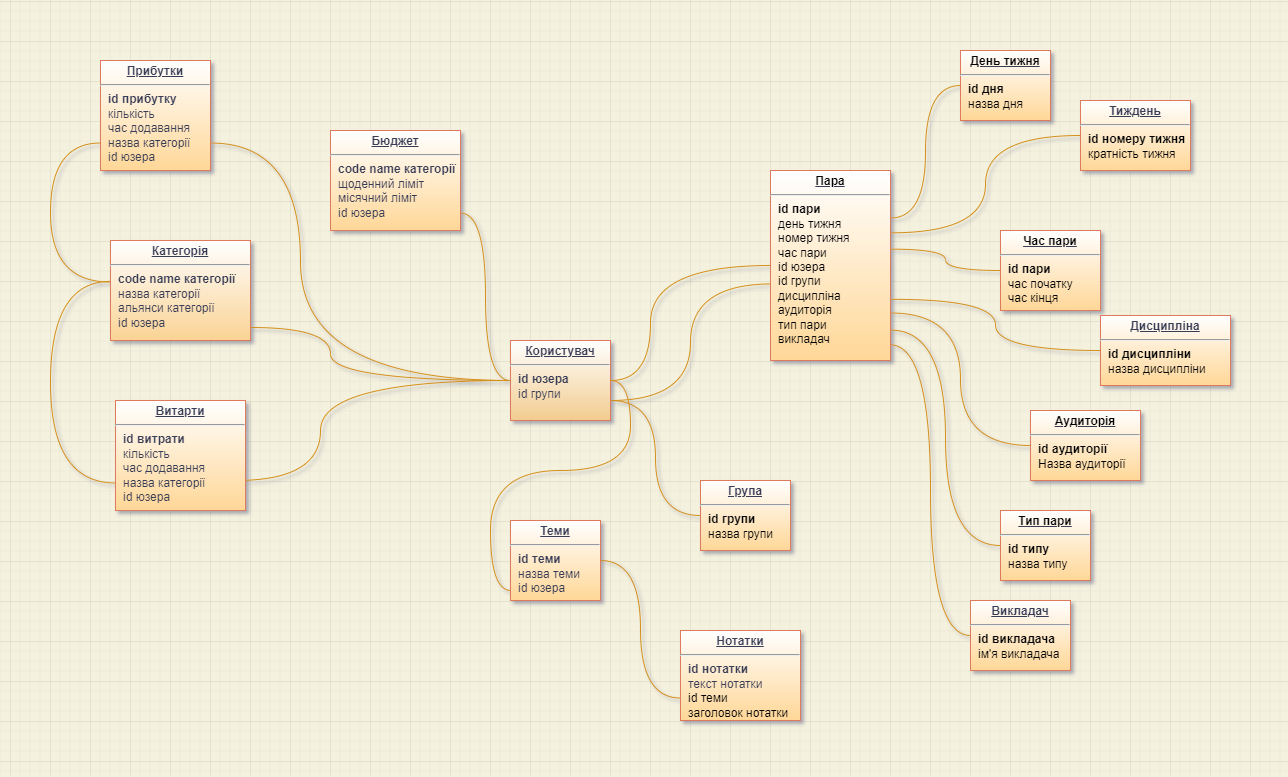


Рис. 2.1.1 – Концептуальна модель бази даних

Як вище було сказано, метою цієї роботи є створення функціональної частини бота-помічника, що відповідає за організацію і контроль фінансів, яка не може функціонувати без бази даних, що зберігала б всі необхідні для користувача та системи дані. Їх вміщено в п’яти таблицях, а саме: «Користувач», «Бюджет», «Категорія», «Витрати» та «Прибутки».

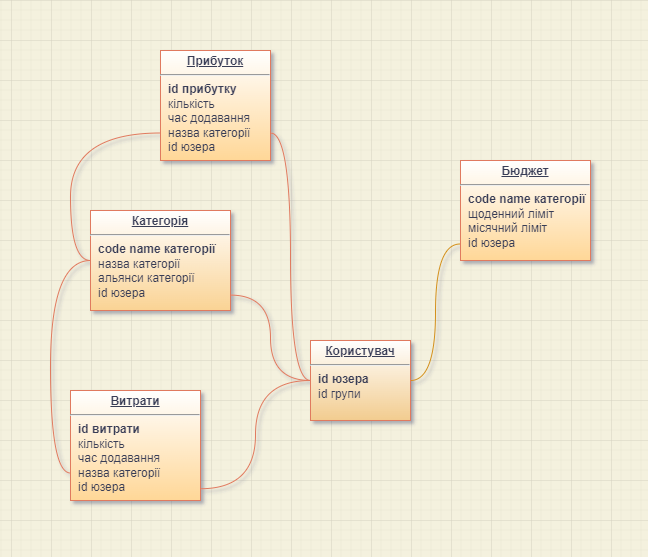


Рис. 2.2.1 Концептуальна модель БД складової «Фінанси» ІС «Student Bot Assistant»

Таблиця «Користувач» є головною таблицею взагалі у всій інформаційній системі «Student Bot Assistant», з нею пов’язані всі інші таблиці. Вона зберігає найважливіші дані – telegram id користувача та id групи.

«Бюджет» пов’язаний лише з головною таблицею, вона вміщує в собі інформацію про щоденний та місячний бюджет користувача, тобто певний ліміт, який він собі  встановлює.

Таблиця «Категорія» також дуже важлива, адже без неї неможливе функціонування таблиць «Витрати» та «Прибутки». Вона зберігає літеральний код категорії, її ім’я та також альянси, тобто допоміжні ключові слова, які забезпечують додавання витрат/прибутку до бази даних так само, як і назва категорії.

Структура таблиць «Витрати» і «Прибутки» майже ідентична: вона відрізняється лише id, проте призначення їх різне. За назвою зрозуміло, що «Витрати» містить інформацію про витрати записані користувачем: кількість грошей в його валюті, точний час додавання запису та код назви категорії. Такі ж дані зберігає і таблиця «Прибутки», проте для прибутків користувача.

**2.2. Фізична модель бази даних**

Фізична модель бази даних – це модель даних, яка визначає яким чином представляються дані і все зберігає всі деталі, необхідні СУБД для створення бази даних. Вона включає всі таблиці, колонки, зв’язки та властивості для фізичної реалізації бази даних [2].

Нижче, на рисунку 2.2.1, представлена фізична модель всієї бази даних «Student Bot Assistant». Вона складається з шістнадцяти таблиць.

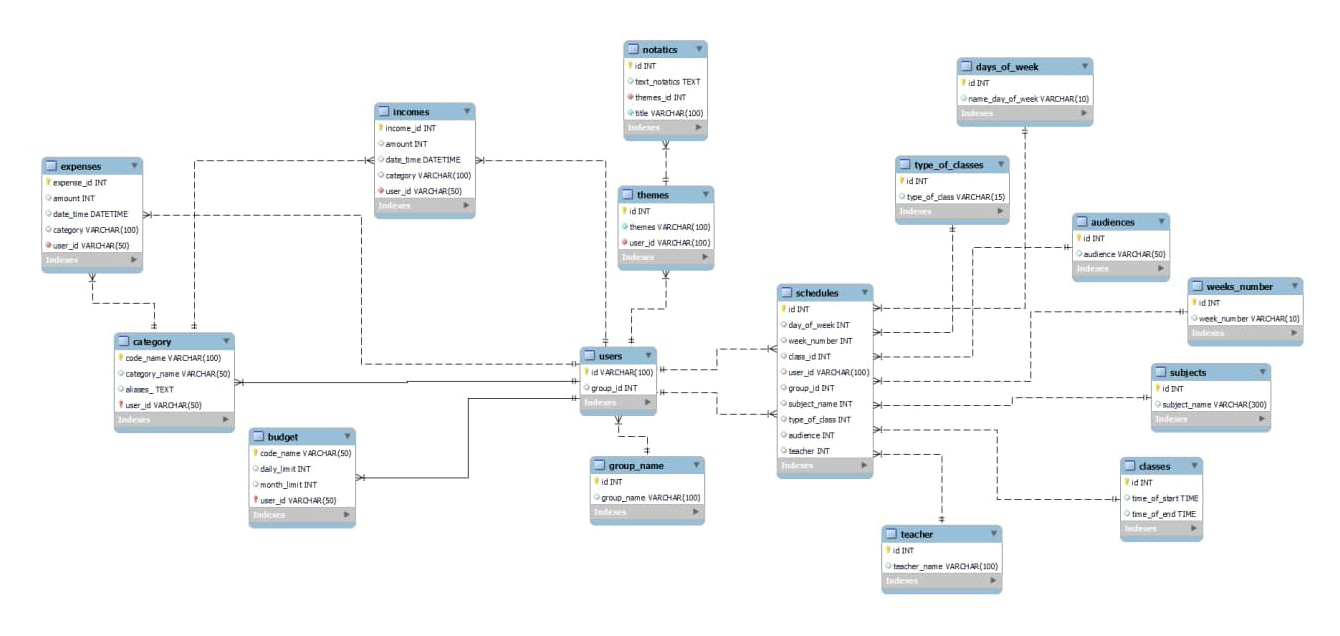


Рис. 2.2.1 – Фізична модель бази даних «Student Bot Assistant»

Частина «Фінанси» складається з чотирьох таблиць, її фізична модель представлена на рисунку 2.2.3.

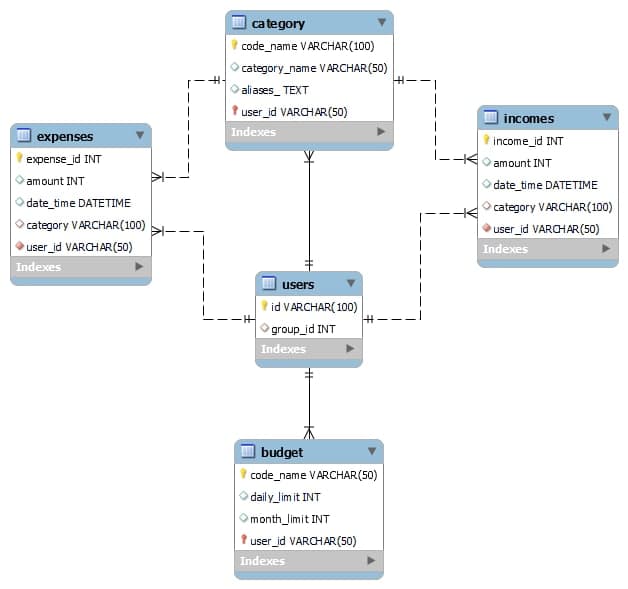


Рис. 2.2.3. Фізична модель фінансової частини БД «Student Bot Assistant»

**2.3. Структура таблиць бази даних**

Головною таблицею є – «users», що зберігає дані про юзера. Вона складається з трьох полів: *id*типу VARCHAR (обмеження до 100 символів), *group\_id* типу INT та *username* VARCHAR (обмеження – 50 символів). Поле *id* – PRIMARY KEY, *username* має властивість – UNIQUE KEY, також *group\_id* виступає як FOREIGN KEY та посилається на поле *id* в таблиці *group\_name.*

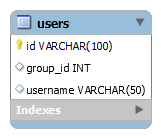


Рис. 2.3.1 Таблиця «users»

Таблиця «budget» містить дані про встановлений користувачем ліміт на щодень та місяць. Складається вона з полів: *code\_name* VARCHAR (обмеження – 50 символів та значенням за замовчуванням – 'general'), *daily\_limit* типу INT (DEFAULT 0), *month\_limit* типу INT (DEFAULT 0), *user\_id* VARCAHR (обмеження – 50 символів), що не може набувати значення NULL та має властивість FOREIGN KEY, тобто посилається на поле *id* таблиці *users.* PRIMARY KEY виступає комбінація полів *code\_name* та *user\_id.*

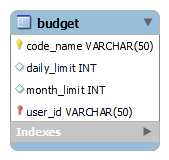


Рис. 2.3.2 Таблиця «budget»

Таблиця «category» містить дані про категорію для витрат або прибутків. Складається з *code\_name* типу VARCHAR (обмеження – 100 символів), *category\_name* – VARCHAR (обмеження – 50 символів), *aliases\_* типу TEXT та поля *user\_id* VARCHAR (обмеження – 50 символів, не може бути NULL). PRIMARY KEY цієї таблиці – це комбінація полів *code\_name* та *user\_id.* А FOREIGN KEY є *user\_id.*

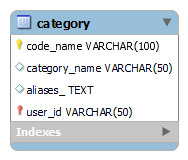


Рис. 2.3.3 Таблиця «category»

Четвертою таблицею є «expenses», що містить п’ять полів. Головне поле *expense\_id* типу INT (має властивості PRIMARY KEY, NOT NULL та AUTO\_INCREMENT); *amount*, що є INT; *date\_time* типу DATETIME, *category* – VARCHAR (з обмеженням в 100 символів) та є FOREIGN KEY й посилається на поле в таблиці «users». Властивіть FOREIGN KEY має також поле *category* й посилається на *code\_name* з таблиці«category».

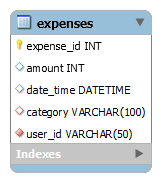


Рис. 2.3.4 Таблиця «expenses»

П'ята таблиця – «incomes», що є дуже подібною до попередньої і також містить п’ять полів. Головне поле *income\_id* типу INT (RIMARY KEY, NOT NULL та AUTO\_INCREMENT); *amount*, що є INT; *date\_time* типу DATETIME, *category* – VARCHAR (з обмеженням в 100 символів) та є FOREIGN KEY й посилається на поле в таблиці «users». Властивіть FOREIGN KEY має також поле *category* й посилається на *code\_name* з таблиці«category».

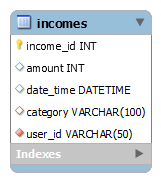
**

Рис. 2.3.5 Таблиця «expenses»

# РОЗДІЛ 3. ОПИС ПРОЕКТУ ПРОГРАМНОГО ПРОДУКТУ

## **3.1. Використане програмне забезпечення**

Для розробки бази даних був використаний інструмент MySQL Workbench, призначений для візуального проектування баз даних та інтегроване середовище для розробки програмного забезпечення мовою програмування Python – PyCharm компанії JetBrains.

MySQL Workbench – це комплексний візуальний інструмент, що використо-вується архітекторами, розробниками та адміністраторами баз даних. MySQL Workbench забезпечує всебічне моделювання даних, засоби розробки та методи управління SQL для конфігурації сервера, крім того засоби управління користувачами, резервного копіювання та ще багато іншого [3].

Інструмент MySQL Workbench є популярним, адже надає візуальні інструменти для створення, виконання та оптимізації запитів SQL, що роблять програму простою та легкозрозумілою у використанні. Редактор SQL забезпечує виділення кольорового синтаксису, автоматичне заповнення, повторне використання фрагментів SQL та історію виконання SQL. Панель Database Connections дозволяє розробникам легко керувати стандартними підключеннями до бази даних, включаючи MySQL Fabric. Браузер об’єктів забезпечує миттєвий доступ до схеми та об’єктів бази даних.

MySQL Workbench надає можливість вибору де працювати – безпосередньо в програмі чи в консолі, що призначена для управління середовищами MySQL є простою у використані і водночас містить все для повноцінної роботи. Крім того, надається можливість використання візуальних засобів для налаштування серверів, управління користувачами, резервного копіювання та відновлення, перевірки даних аудиту та перегляду стану бази даних, що є надзвичайно корисним для розробників та адміністраторів БД [4].

Середовище розробки PyCharm призначене для програмування мовою Python, та визнана найкращим серед сьогодні існуючих. PyCharm надає багато можливостей для зручної роботи: зміна користувацького інтерфейсу, різні теми та підсвітка синтаксису, налаштування користувацьких комбінацій клавіш. PyCharm робить розробку максимально продуктивною завдяки функція автодоповнення, аналізу коду, підствіткою помилок та швидким виправленням. Доступна навігація, яка дозволяє швидко переходити до будь-якого класу, файлу, вікну чи навіть символу в один клік, для швидкого переміщення проектом. PyCharm містить також вбудовані інструменти для розробки: відладчик, інструмент запуску тестів, повноцінний термінал, підтримує наукові бібліотеки Pandas, Numpy, Matplotlib та метроди віддаленої розробки Git, Docker і Vagran та web розробку.

Крім того, включає інструменти для роботи з базами даних: має доступ до Oracle, SQL Server, PostgreSQL, MySQL й інших баз даних. PyCharm навіт допомагає редагувати SQL-код, виконувати запити, преглядати схеми та змінювати схеми [5].

# РОЗДІЛ 4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

**4.1. Загальний опис програми**

Програма являє собою інтерфейс для допомоги студенту в різних аспектах діяльності: налаштування розкладу, ведення нотаток та контроль фінансів. Функціонал фінансової частини програмного продукту розміщений в модулях *Bot\_Assistant.py, Finances.py, Statistic.py, markup.py, exceptions.py та databases.py,* він дозволяє організовувати та контролювати власні кошти: встановлювати ліміт та редагувати його, додавати та переглядати категорії, додавати витрати, прибуток й видаляти ці записи з бази даних. Крім того доступна можливість перегляду статистики за певний період часу.

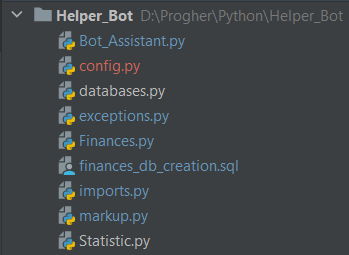


Рис. 4.1.1 – Модулі програмного продукту

Модуль *Bot\_Assistant.py* – головний модуль програми, в якому розміщені всі хендлери (анг. *handlers*)та класи станівнеобхідні для обробки команд, встановлення станів – функціонування бота. Нижче перечислені класи та хендлери забезпечують роботу фінансової частини програми.

Класи:

* class HandlerIncomes;
* class HandlerExpenses;
* class HandlerStatistic;
* class HandlerCategory;
* class HandlerBudget;
* class HandlerOtherFinances;

Хендлери для роботи з фінансами:

* finance\_menu;
* budget\_menu;
* statistic\_menu;
* other\_finance\_menu;
* add\_expense\_handler;
* adding\_expense\_message\_handler;
* add\_incomes\_handler;
* adding\_incomes\_message\_handler;
* edit\_budget\_handler;
* editing\_budget\_message\_handler;
* add\_category\_handler;
* creating\_finance\_category;
* del\_expense\_handler;
* del\_income\_handler;
* categories\_viewing\_handler;
* today\_expenses\_handler;
* week\_expenses\_handler;
* month\_expenses\_handler;
* today\_incomes\_handler;
* week\_incomes\_handler;
* month\_incomes\_handler;
* this\_week\_statistic\_handler;
* this\_month\_statistic\_handler;
* back\_to\_finance\_menu;
* back\_to\_other\_finance\_menu.

Модуль *markup.py* – модуль для створення кнопок бота.

* inline\_button\_finance;
* inline\_button\_add\_finance\_category;
* inline\_button\_add\_expense;
* inline\_button\_add\_incomes;
* inline\_button\_budget;
* inline\_button\_finance\_statistic;
* inline\_button\_finance\_other;
* inline\_button\_back\_to\_finance;
* inline\_button\_statistic\_by\_week;
* inline\_button\_statistic\_by\_month;
* inline\_button\_edit\_budget;
* inline\_button\_see\_today\_expenses;
* inline\_button\_see\_week\_expenses;
* inline\_button\_see\_month\_expenses;
* inline\_button\_see\_today\_incomes;
* inline\_button\_see\_week\_incomes;
* inline\_button\_see\_month\_incomes;
* inline\_button\_back\_to\_other\_finance;
* inline\_button\_see\_categories;
* inline\_keyboard\_budget\_menu;
* inline\_keyboard\_statistic\_menu;
* inline\_keyboard\_finance\_menu;
* inline\_keyboard\_other\_menu.

Модуль *Finances.py* – модуль, в якому розміщені всі класи та методи, що забезпечують логіку роботи фінансової частини бота. Функції забезпечують парсинг повідомлень, додавання, видалення фінансових записів, перегляд даних користувача, тобто відбувається взаємодія з базою даних: вивід даних, додавання, видалення та редагування.

Класи:

* class Categories;
* class CreateCategory;
* class CategoryMessage;
* class Category;
* class Message;
* class BudgetMessage;
* class IncomeExpense;
* class UserData.

Методи:

* \_parse\_user\_data;
* parse\_category;
* \_parse\_message;
* \_parse\_budget\_message;
* check\_user\_exists;
* add\_user;
* add\_expense;
* edit\_budget;
* add\_incomes;
* create\_category\_finance;
* see\_categories;
* delete\_income;
* \_get\_now\_formatted;
* \_get\_now\_datetime;
* set\_default\_budget;
* get\_budget\_month\_limit;
* get\_budget\_daily\_limit;
* today\_expenses;
* this\_week\_expenses;
* this\_month\_expenses;
* today\_incomes;
* this\_week\_incomes;
* this\_month\_incomes.

Роботу в модулі з БД, представлено на прикладі функції *this\_month\_expenses*, що забезпечує «витягування» всіх витрат за поточний місяць.

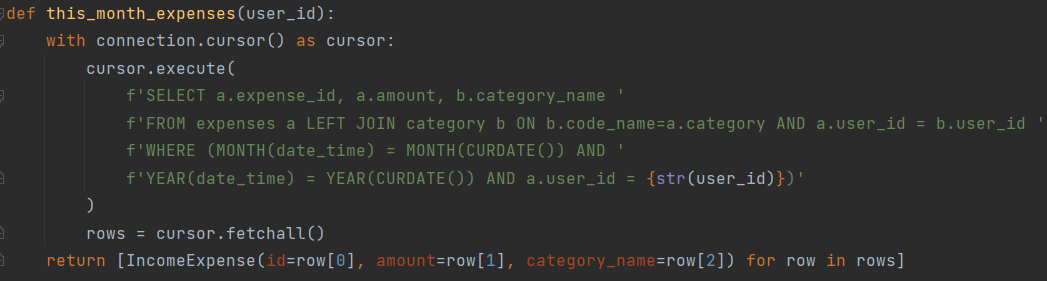


Рис. 4.1.2 – Приклад SELECT в модулі Finances.py

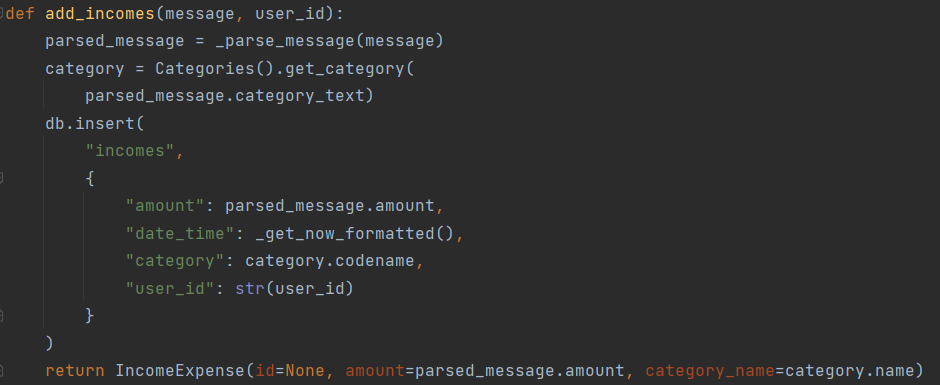


Рис. 4.1.3 – Приклад INSERT в модулі Finances.py

У модулі *Finances.py* insert, delete, fetchall здійснюється через перезавантаженні методи бібліотеки pymysql в модулі *databases.py*

Модуль *Statistic.py* забезпечує аналіз та виведення його результатів стосовно витрат та прибутків користувача за певний період часу. Результат подається у вигляді графіку з кривими витрат та прибутку, зображення прикріплюються сумою *expenses  & incomes* та чистий дохід (анг. *pure profit*).

Методи:

* \_get\_formatted;
* merging\_list;
* delete\_stats\_image;
* get\_week\_expenses\_for\_stats;
* get\_week\_incomes\_for\_stats;
* get\_month\_expenses\_for\_stats;
* get\_month\_incomes\_for\_stats;
* week\_data\_for\_stats;
* month\_data\_for\_stats;
* stats\_for\_current\_week;
* stats\_for\_current\_month;
* create\_diagram\_for\_stats;
* calculating\_results;
* resulting\_for\_the\_current\_week;
* resulting\_for\_the\_current\_month.

Робота в цьому модулі з базою даних подібна до роботи в *Finances.py,* проте призначення функцій та значення, що вони повертають відрізняється.

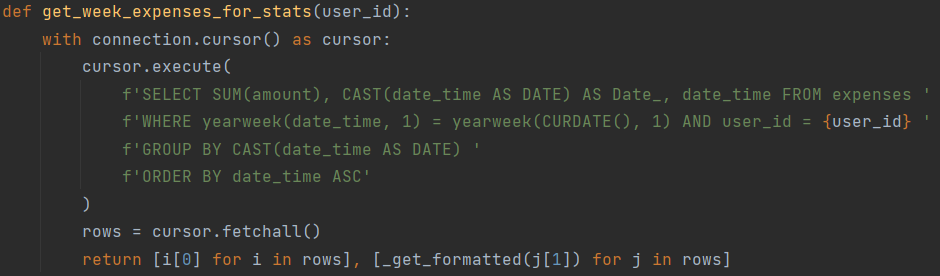


Рис. 4.1.2 – Приклад SELECT в модулі Finances.py

В модулі *databeses.py* виконано перезавантаження стандартних функцій бібліотеки *pymysql,* призначених для роботи з базами даних, а саме з спроектованими в MySQL. Хоч функції стандартні, проте тут вони перезавантажені так, щоб з їх допомогою можна було просто виконувати подібні запити без повторів, тобто ці функції є універсальними і працюють для будь-яких таблиць з бази даних з різними комбінація. Крім того, вони надають можливість виконання команди не лише для одного рядка значень, а для будь якої кількості.

Методи:

* insert
* fetchall\_
* fetchone\_for\_budget
* update\_

**4.2. Опис інтерфейсу програми**

Після запуску головної команди бота «*/start*» користувачу стає доступним головне меню «помічника». Бот виводить повідомлення-привітання та кнопки *Library, Schedule, Notes* та *Finance.*

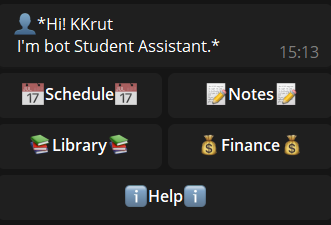


Рис. 4.2.1 – Головне меню програми «Student Bot Assistant»

Після натискання кнопки Finance стає доступним меню для менеджменту коштів. Надається можливість додати прибуток/витрати/категорію, перейти в меню статистики, бюджету та іншого

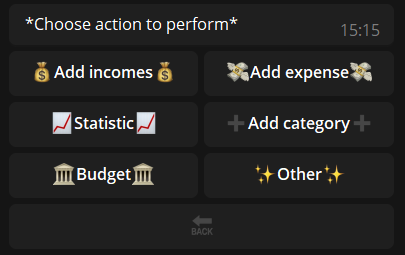


Рис. 4.2.2 – Вигляд меню «Finances»

Після натискання кнопки «Add category» користувач може додати категорію витрат/доходів.

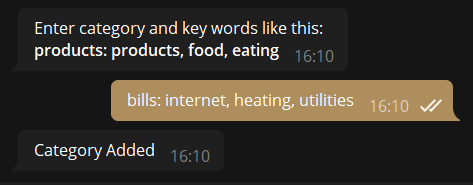


Рис. 4.2.3 – Додавання категорії. Результат «Add category»

Після натискання кнопки «Add expenses» надається можливість додати витрати. Додавання відбувається за назвою категорії, або ж за «альянсами», тобто допоміжними ключовими словами: додавання в категорію «transport» можливо за всіма введеними словами: transport, taxi, metro, bus. Кнопка «Add incomes» працює аналогічно, різниця лише в тому, в яку таблицю додаються дані – expenses чи incomes

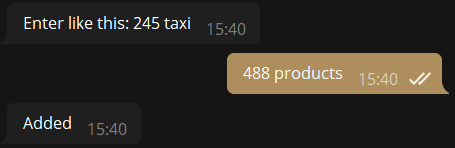


Рис. 4.2.4 – Приклад роботи «Add expenses»

Кнопка «Budget» надає можливість пергелянути свій ліміт, що за замовчуванням встановлюється *Daily: 0* та *Month: 0,* відредагувати, тобто встановити інший ліміт, та повернутись назад до фінансового меню.

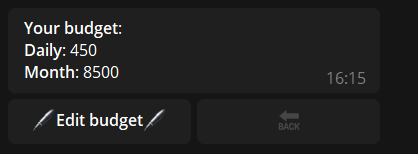


Рис. 4.2.5 – «Budget» меню

Натискання на кнопку «Statistic» переводить юзера в інше меню, де можна вибрати за який період переглянути статистичні дані: поточний тиждень чи місяць.

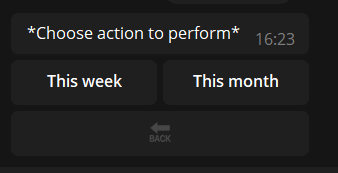


Рис. 4.2.6 – «Statistic» меню

Графіки будуються на основі даних взятих з бази даних, щодо витрат та прибутку користувача, на цих же даних відбувається обчислення.



Рис. 4.2.6 – Приклад роботи статистики («This month»)

Меню «Other» дає можливість переглянути користувачеві свої категорії, витрати чи прибутки за поточний день, тиждень чи місяць, або ж повернутись до фінансового меню.

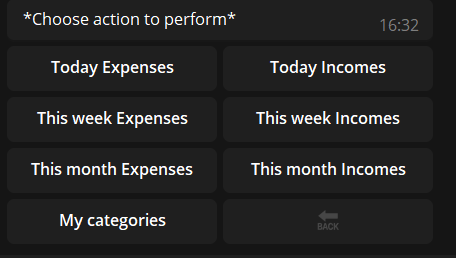


Рис. 4.2.7 – Меню «Other»

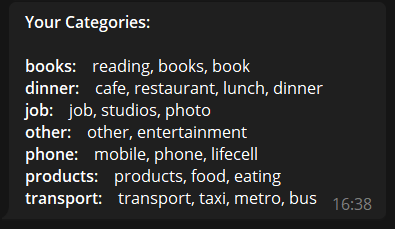


Рис. 4.2.8 – Перегляд власних категорій. Кнопка «My categories»

При перегляді прибутку або витрат надається можливість їх видалити зі своєї бази даних, видалення відбувається дуже просто – в один клік.

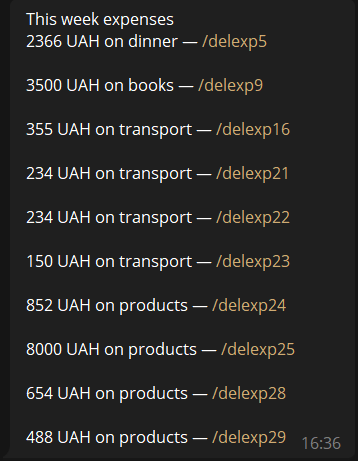


Рис. 4.2.9 – Приклад роботи кнопки «This week expenses»

Натиснувши на команду навпроти витрати чи прибутку можна легко її видалити зі свого списку.

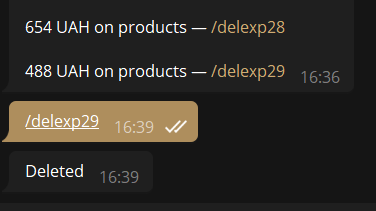


Рис. 4.2.10 – Приклад видалення

**ВИСНОВКИ**

Під час виконання цієї курсової роботи вдалося набути нові та вдосконалити вже здобуті знання і навички з створення та взаємодії з базами даних за допомогою мови структурованих запитів SQL на основі додатку MySQL. Крім того, здобуто досвід роботи з Framework Aiogram для створення Telegram ботів, здобуто досвід роботи з бібліотекою pymysql.

Метою роботи було створення програмного продукту «Student Bot Assistant», що допомагав би в організації власного розкладу, веденні нотаток та організації коштів, що за допомогою створеного функціоналу давав би можливість додавати, редагувати та видаляти інформацію користувача з бази даних.

Сьогодні в нашому світі, що невпинно швидко рухається й змінюється дуже важливо вміти правильно організувати свій час, розклад, вміти слідкувати за власним добробутом, тому такий, однозначно, буде корисним для будь-кого, простий, інтуїтивно зрозумілий інтерфейс та можливість налаштування – зручним у використані, а доступність в одному із найпопулярніших мессенджерів світу робить такий додаток доступним для кожного.

У ході виконання курсової роботи було реалізовано програму для допомоги у навчанні та в повсякденному житті.

# СПИСКИ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вікіпедія – вільна енциклопедія [Електронний ресурс] https://inlnk.ru/agg1oj
2. Вікіпедія – вільна енциклопедія [Електронний ресурс]  https://inlnk.ru/BppY
3. Mysql Documentation [Електронний ресурс] <https://dev.mysql.com/doc/>
4. Mysql [Електронний ресурс] https://www.mysql.com/
5. https://www.jetbrains.com/ru-ru/pycharm/features/

# ДОДАТОК 1

Код програми

НТУУ «КПІ ім. Ігоря Сікорського» ТЕФ АПЕПС ТІ-01

Листів 3

Київ – 2021

**Модуль *markup.py***

from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton  
  
"""INLINE KEYBOARD BUTTONS"""  
  
inline\_button\_notes = InlineKeyboardButton('📝Notes📝', callback\_data='📝Notes📝')  
inline\_button\_library = InlineKeyboardButton('📚Library📚', callback\_data='📚Library📚')  
inline\_button\_schedule = InlineKeyboardButton('📅Schedule📅', callback\_data='📅Schedule📅')  
inline\_button\_finance = InlineKeyboardButton('💰Finance💰', callback\_data='💰Finance💰')  
inline\_button\_back = InlineKeyboardButton('🔙', callback\_data='🔙')  
inline\_keyboard\_menu = InlineKeyboardMarkup(row\_width=2).add(inline\_button\_schedule, inline\_button\_notes, inline\_button\_library, inline\_button\_finance)  
  
inline\_button\_check\_notes = InlineKeyboardButton("🔎Search note🔎", callback\_data="🔎Search note🔎")  
inline\_button\_add\_note = InlineKeyboardButton('➕Add note➕', callback\_data='➕Add note➕')  
inline\_button\_delete\_note = InlineKeyboardButton("🔥Delete note🔥", callback\_data="🔥Delete note🔥")  
inline\_button\_edit\_note = InlineKeyboardButton("🖋️Edit note🖋️", callback\_data="🖋️Edit note🖋️")  
inline\_keyboard\_note\_menu = InlineKeyboardMarkup(row\_width=2).add(inline\_button\_check\_notes, inline\_button\_add\_note, inline\_button\_edit\_note, inline\_button\_delete\_note, inline\_button\_back)  
  
inline\_button\_search\_by\_name = InlineKeyboardButton("🔎Search by name🔎", callback\_data="🔎Search by name🔎")  
inline\_button\_search\_by\_theme = InlineKeyboardButton("🔎Search by theme🔎", callback\_data="🔎Search by theme🔎")  
inline\_button\_show\_all = InlineKeyboardButton("🔎Show all notes🔎", callback\_data="🔎Show all notes🔎")  
inline\_keyboard\_search\_menu = InlineKeyboardMarkup(row\_width=2).add(inline\_button\_search\_by\_name,  
 inline\_button\_search\_by\_theme,  
 inline\_button\_show\_all, inline\_button\_back)  
  
inline\_button\_schedule\_settings = InlineKeyboardButton("🔧Settings🔧", callback\_data="🔧Settings🔧")  
inline\_button\_schedule\_currentday = InlineKeyboardButton("⌚Today schedule⌚",  
 callback\_data="⌚Today schedule⌚")  
inline\_button\_schedule\_nextday = InlineKeyboardButton("📅Next day schedule📅", callback\_data="📅Next day schedule📅")  
inline\_button\_schedule\_next = InlineKeyboardButton("⏭️Next pair⏭️", callback\_data="⏭️Next pair⏭️")  
inline\_button\_schedule\_currentweek = InlineKeyboardButton("🟡This week schedule🟡", callback\_data="🟡This week schedule🟡")  
inline\_button\_schedule\_nextweek = InlineKeyboardButton("⚫Next week schedule⚫", callback\_data="⚫Next week schedule⚫")  
inline\_keyboard\_schedule\_menu = InlineKeyboardMarkup(row\_width=2).add(inline\_button\_schedule\_settings, inline\_button\_schedule\_currentday, inline\_button\_schedule\_nextday, inline\_button\_schedule\_next, inline\_button\_schedule\_currentweek, inline\_button\_schedule\_nextweek, inline\_button\_back)  
  
inline\_button\_add\_group = InlineKeyboardButton("➕Add group➕", callback\_data="➕Add group➕")  
inline\_button\_delete\_group = InlineKeyboardButton("➖Delete group➖", callback\_data="➖Delete group➖")  
# inline\_button\_on\_notification = InlineKeyboardButton("📧On notification📧", callback\_data="📧On notification📧")  
# inline\_button\_off\_notification = InlineKeyboardButton("📴Off notification📴", callback\_data="📴Off notification📴")  
inline\_button\_add\_pair = InlineKeyboardButton("➕Add schedule➕", callback\_data="➕Add schedule➕")  
inline\_button\_add\_monday = InlineKeyboardButton("Monday", callback\_data="Monday")  
inline\_button\_add\_tuesday = InlineKeyboardButton("Tuesday", callback\_data="Tuesday")  
inline\_button\_add\_wednesday = InlineKeyboardButton("Wednesday", callback\_data="Wednesday")  
inline\_button\_add\_thursday = InlineKeyboardButton("Thursday", callback\_data="Thursday")  
inline\_button\_add\_friday = InlineKeyboardButton("Friday", callback\_data="Friday")  
inline\_button\_add\_saturday = InlineKeyboardButton("Saturday", callback\_data="Saturday")  
  
inline\_keyboard\_schedule\_settings = InlineKeyboardMarkup(row\_width=2).add(inline\_button\_add\_group, inline\_button\_delete\_group, inline\_button\_add\_pair, inline\_button\_back)  
  
inline\_keyboard\_day\_of\_week = InlineKeyboardMarkup(row\_width=2).add(inline\_button\_add\_monday, inline\_button\_add\_tuesday, inline\_button\_add\_wednesday, inline\_button\_add\_thursday, inline\_button\_add\_friday, inline\_button\_add\_saturday, inline\_button\_back)  
  
inline\_button\_first = InlineKeyboardButton("First", callback\_data="First")  
inline\_button\_second = InlineKeyboardButton("Second", callback\_data="Second")  
inline\_button\_third = InlineKeyboardButton("Third", callback\_data="Third")  
inline\_button\_fourth = InlineKeyboardButton("Fourth", callback\_data="Fourth")  
inline\_button\_fifth = InlineKeyboardButton("Fifth", callback\_data="Fifth")  
inline\_button\_sixth = InlineKeyboardButton("Sixth", callback\_data="Sixth")  
  
inline\_keyboard\_pair = InlineKeyboardMarkup(row\_width=2).add(inline\_button\_first, inline\_button\_second, inline\_button\_third, inline\_button\_fourth, inline\_button\_fifth, inline\_button\_sixth, inline\_button\_back)  
  
inline\_button\_week = InlineKeyboardButton("Week", callback\_data="Week")  
inline\_button\_subject\_name = InlineKeyboardButton("Subject name", callback\_data="Subject name")  
inline\_button\_type\_of\_class = InlineKeyboardButton("Type of class", callback\_data="Type of class")  
inline\_button\_audience = InlineKeyboardButton("Audience", callback\_data="Audience")  
inline\_button\_teacher = InlineKeyboardButton("Teacher", callback\_data="Teacher")  
inline\_button\_ready = InlineKeyboardButton("✅Ready✅", callback\_data="✅Ready✅")  
  
inline\_keyboard\_add\_pair = InlineKeyboardMarkup(row\_width=2).add(inline\_button\_week, inline\_button\_subject\_name, inline\_button\_type\_of\_class, inline\_button\_audience, inline\_button\_teacher, inline\_button\_ready)  
  
inline\_button\_even = InlineKeyboardButton("EVEN", callback\_data="EVEN")  
inline\_button\_odd = InlineKeyboardButton("ODD", callback\_data="ODD")  
inline\_keyboard\_week\_menu = InlineKeyboardMarkup(row\_width=2).add(inline\_button\_even, inline\_button\_odd)  
  
inline\_button\_lecture = InlineKeyboardButton("Lecture", callback\_data="Lecture")  
inline\_button\_lab = InlineKeyboardButton("Lab", callback\_data="Lab")  
inline\_button\_practice = InlineKeyboardButton("Practice", callback\_data="Practice")  
inline\_keyboard\_type\_of\_lesson\_menu = InlineKeyboardMarkup(row\_width=2).add(inline\_button\_lecture, inline\_button\_lab, inline\_button\_practice, inline\_button\_back)  
  
inline\_button\_add\_finance\_category = InlineKeyboardButton('➕Add category➕', callback\_data='➕Add category➕')  
inline\_button\_add\_expense = InlineKeyboardButton('💸Add expense💸', callback\_data='💸Add expense💸')  
inline\_button\_add\_incomes = InlineKeyboardButton('💰Add incomes💰', callback\_data='💰Add incomes💰')  
inline\_button\_budget = InlineKeyboardButton('🏛️Budget🏛️', callback\_data='🏛️Budget🏛️')  
inline\_button\_finance\_statistic = InlineKeyboardButton('📈Statistic📈', callback\_data='📈Statistic📈')  
inline\_button\_finance\_other = InlineKeyboardButton('✨Other✨', callback\_data='OTHER\_FINANCE\_MENU')  
inline\_button\_back\_to\_finance = InlineKeyboardButton('🔙', callback\_data='BACK\_TO\_FINANCE')  
  
inline\_keyboard\_finance\_menu = InlineKeyboardMarkup(row\_width=2).add(  
 inline\_button\_add\_incomes, inline\_button\_add\_expense, inline\_button\_finance\_statistic,  
 inline\_button\_add\_finance\_category, inline\_button\_budget, inline\_button\_finance\_other,  
 inline\_button\_back  
)  
  
inline\_button\_statistic\_by\_week = InlineKeyboardButton('This week', callback\_data='WEEK\_STATISTIC')  
inline\_button\_statistic\_by\_month = InlineKeyboardButton('This month', callback\_data='MONTH\_STATISTIC')  
  
inline\_keyboard\_statistic\_menu = InlineKeyboardMarkup(row\_width=2).add(  
 inline\_button\_statistic\_by\_week, inline\_button\_statistic\_by\_month,  
 inline\_button\_back\_to\_finance  
)  
  
inline\_button\_edit\_budget = InlineKeyboardButton('🖊️Edit budget🖊️', callback\_data='🖊️Edit budget🖊️')  
  
inline\_keyboard\_budget\_menu = InlineKeyboardMarkup(row\_width=2).add(  
 inline\_button\_edit\_budget, inline\_button\_back\_to\_finance  
)  
  
inline\_button\_see\_today\_expenses = InlineKeyboardButton('Today Expenses', callback\_data='TODAY\_EXPENSES')  
inline\_button\_see\_week\_expenses = InlineKeyboardButton('This week Expenses', callback\_data='WEEK\_EXPENSES')  
inline\_button\_see\_month\_expenses = InlineKeyboardButton('This month Expenses', callback\_data='MONTH\_EXPENSES')  
  
inline\_button\_see\_today\_incomes = InlineKeyboardButton('Today Incomes', callback\_data='TODAY\_INCOMES')  
inline\_button\_see\_week\_incomes = InlineKeyboardButton('This week Incomes', callback\_data='WEEK\_INCOMES')  
inline\_button\_see\_month\_incomes = InlineKeyboardButton('This month Incomes', callback\_data='MONTH\_INCOMES')  
  
  
inline\_button\_back\_to\_other\_finance = InlineKeyboardButton('🔙', callback\_data='BACK\_TO\_OTHER\_FINANCE')  
inline\_button\_see\_categories = InlineKeyboardButton('My categories', callback\_data='SEE\_CATEGORIES')  
  
  
inline\_keyboard\_other\_menu = InlineKeyboardMarkup(row\_width=2).add(  
 inline\_button\_see\_today\_expenses, inline\_button\_see\_today\_incomes,  
 inline\_button\_see\_week\_expenses, inline\_button\_see\_week\_incomes,  
 inline\_button\_see\_month\_expenses, inline\_button\_see\_month\_incomes,  
 inline\_button\_see\_categories, inline\_button\_back\_to\_finance  
)

**Модуль *Bot\_Assistant.py***

from aiogram.dispatcher import FSMContext  
from aiogram.types import InlineKeyboardMarkup  
import Finances  
import Statistic  
import bd  
import exceptions  
from imports import \*  
import config  
import logging  
from datetime import datetime, timedelta as tmd  
from config import connection  
from aiogram import Bot, Dispatcher, executor, types  
  
from aiogram.contrib.fsm\_storage.memory import MemoryStorage  
from aiogram.dispatcher.filters.state import State, StatesGroup  
import asyncio  
import markup  
  
logging.basicConfig(level=logging.INFO)  
bot = Bot(token=config.TOKEN)  
dp = Dispatcher(bot, storage=MemoryStorage())  
  
  
class GroupAdd(StatesGroup):  
 AddGroup = State()  
 SelectGroup = State()  
  
  
class NoteSearch(StatesGroup):  
 SearchNote = State()  
  
  
class NoteSearchByTitle(StatesGroup):  
 SearchByTitle = State()  
  
  
class NoteEdit(StatesGroup):  
 DeleteNote = State()  
 DeleteTopic = State()  
  
  
class EditNote(StatesGroup):  
 SearchThemes = State()  
 SearchTitle = State()  
 EnterText = State()  
  
  
class Note(StatesGroup):  
 AddTheme = State()  
 AddName = State()  
 AddText = State()  
  
  
class AddPair(StatesGroup):  
 AddSubject = State()  
  
  
class AddAudiences(StatesGroup):  
 AddAudience = State()  
  
  
class AddTeachers(StatesGroup):  
 AddTeacher = State()  
  
  
class HandlerIncomes(StatesGroup):  
 AddIncomesState = State()  
  
  
class HandlerExpenses(StatesGroup):  
 AddExpensesState = State()  
  
  
class HandlerStatistic(StatesGroup):  
 StatisticState = State()  
  
  
class HandlerCategory(StatesGroup):  
 CategoriesState = State()  
  
  
class HandlerBudget(StatesGroup):  
 BudgetState = State()  
  
  
class HandlerOtherFinances(StatesGroup):  
 OtherState = State()  
  
  
list\_of\_themes = []  
selected\_day = 0  
selected\_pair = 0  
selected\_week = 0  
selected\_type\_of\_class = 0  
selected\_subject = 0  
selected\_audience = 0  
selected\_teacher = 0  
selected\_audience\_id = 0  
selected\_subject\_id = 0  
selected\_teacher\_id = 0  
update\_week = False  
  
  
@dp.message\_handler(commands=['start'])  
async def send\_welcome\_message(message: types.Message):  
 await message.delete()  
 if not Finances.check\_user\_exists(str(message.from\_user.id)):  
 Finances.add\_user(message['from'])  
 Finances.set\_default\_budget(str(message.from\_user.id))  
 await bot.send\_message(  
 message.from\_user.id, f"👤\*Hi! {message.from\_user.first\_name if message.from\_user.first\_name else ''} "  
 f"{message.from\_user.last\_name if message.from\_user.last\_name else ''}\n "  
 f"I'm bot Student Assistant.\*", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_menu  
 )  
  
  
@dp.callback\_query\_handler(text="📝Notes📝")  
async def note\_menu(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "<b>Choose action to perform</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_note\_menu)  
  
  
@dp.callback\_query\_handler(text="➕Add note➕")  
async def add\_note(call: types.CallbackQuery):  
 await call.message.delete()  
 selected\_theme\_id = 0  
 selected\_note\_id = 0  
  
 with connection.cursor() as cursor:  
  
 select\_id = f"""SELECT themes FROM themes WHERE user\_id={call.from\_user.id}"""  
 cursor.execute(select\_id)  
 result = cursor.fetchall()  
 global list\_of\_themes  
 list\_of\_themes = []  
 if result:  
 for themes in result:  
 for theme in themes:  
 list\_of\_themes.append(theme)  
 await bot.send\_message(call.from\_user.id, "<b>Available themes:</b>\n" + '\n'.join(list\_of\_themes),  
 parse\_mode="HTML")  
 await bot.send\_message(call.from\_user.id, "<b>📝Enter THEME of note📝</b>", parse\_mode="HTML")  
 await Note.AddTheme.set()  
  
 @dp.message\_handler(state=Note.AddTheme)  
 async def add\_theme(message: types.Message):  
 nonlocal selected\_theme\_id  
 global list\_of\_themes  
  
 if message.text in list\_of\_themes:  
 with connection.cursor() as curs:  
 search\_theme = f"SELECT id FROM themes WHERE user\_id = {message.from\_user.id} " \  
 f"AND themes = '{message.text}'"  
 curs.execute(search\_theme)  
 selected\_theme\_id = curs.fetchone()[0]  
 else:  
 with connection.cursor() as curs:  
 adding\_theme = f"INSERT INTO themes(user\_id, themes) VALUES({message.from\_user.id}, '{message.text}');"  
 curs.execute(adding\_theme)  
 connection.commit()  
 search\_theme = f"SELECT id FROM themes WHERE user\_id = {message.from\_user.id} " \  
 f"AND themes = '{message.text}'"  
 curs.execute(search\_theme)  
 selected\_theme\_id = curs.fetchone()[0]  
 await bot.send\_message(message.from\_user.id, "<b>📝Enter NAME of note📝</b>", parse\_mode="HTML")  
 await Note.AddName.set()  
  
 @dp.message\_handler(state=Note.AddName)  
 async def add\_name(message: types.Message):  
 nonlocal selected\_theme\_id  
 nonlocal selected\_note\_id  
 with connection.cursor() as cur:  
  
 this\_title = f"""SELECT title FROM notatics WHERE themes\_id={selected\_theme\_id}"""  
 cur.execute(this\_title)  
 results = cur.fetchall()  
 list\_of\_title = []  
 if results:  
 for titles in results:  
 for title in titles:  
 list\_of\_title.append(title)  
 if message.text in list\_of\_title:  
 await bot.send\_message(message.from\_user.id, "This name is already used on this topic, try another")  
 else:  
  
 with connection.cursor() as curs:  
 adding\_name = f"""INSERT INTO notatics(themes\_id, title)   
 VALUES({selected\_theme\_id}, "{message.text}")"""  
 curs.execute(adding\_name)  
 connection.commit()  
 search\_note = f"""SELECT id FROM notatics WHERE themes\_id = {selected\_theme\_id} """ \  
 f"""AND title = "{message.text}" """  
 curs.execute(search\_note)  
 selected\_note\_id = curs.fetchone()[0]  
 await Note.AddText.set()  
 await bot.send\_message(message.from\_user.id, "<b>📝Enter TEXT of note📝</b>", parse\_mode="HTML")  
  
 @dp.message\_handler(state=Note.AddText)  
 async def add\_text(message: types.Message):  
 nonlocal selected\_note\_id  
 with connection.cursor() as curs:  
 adding\_name = f"""UPDATE notatics SET text\_notatics = "{message.text}" WHERE id = {selected\_note\_id};"""  
 curs.execute(adding\_name)  
 connection.commit()  
 await Note.next()  
 mes = await bot.send\_message(message.from\_user.id, "<b>✅Note added✅</b>", parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await mes.edit\_text(f"👤<b>Hi! {call.from\_user.first\_name if call.from\_user.first\_name else ''} "  
 f"{call.from\_user.last\_name if call.from\_user.last\_name else ''}\n I'm "  
 f"bot Student Assistant.</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_menu)  
  
  
@dp.callback\_query\_handler(text="🔥Delete note🔥")  
async def send\_themes(call: types.CallbackQuery):  
 await call.message.delete()  
 topic\_names\_list = []  
 theme\_id = 0  
 with connection.cursor() as cursor:  
 select\_id = f"""SELECT themes FROM themes WHERE user\_id={call.from\_user.id}"""  
 cursor.execute(select\_id)  
 result = cursor.fetchall()  
 list\_themes = []  
 if result:  
 for themes in result:  
 for theme in themes:  
 list\_themes.append(theme)  
 await bot.send\_message(call.from\_user.id, "<b>Available themes:</b>\n" + '\n'.join(list\_themes),  
 parse\_mode="HTML")  
 if len(list\_themes):  
 await bot.send\_message(call.from\_user.id, "<b>📝Enter THEME of note to delete it📝</b>", parse\_mode="HTML")  
 await NoteEdit.DeleteNote.set()  
 else:  
 note = await bot.send\_message(call.from\_user.id, "<b>❌THERE IS NO NOTES TO DELETE❌</b>", parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await note.edit\_text(f"👤<b>Hi! {call.from\_user.first\_name if call.from\_user.first\_name else ''} "  
 f"{call.from\_user.last\_name if call.from\_user.last\_name else ''}\n I'm "  
 f"bot Student Assistant.</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_menu)  
  
 @dp.message\_handler(state=NoteEdit.DeleteNote)  
 async def delete\_note(message: types.Message):  
 nonlocal list\_themes  
  
 if message.text in list\_themes:  
 with connection.cursor() as curs:  
 nonlocal topic\_names\_list  
 nonlocal theme\_id  
 select\_theme\_id = f"""SELECT id FROM themes WHERE themes="{message.text}" AND user\_id={message.from\_user.id};"""  
 curs.execute(select\_theme\_id)  
 theme\_id = curs.fetchone()[0]  
 select\_topics = f"""SELECT title FROM notatics WHERE themes\_id= {theme\_id}"""  
 curs.execute(select\_topics)  
 topics = curs.fetchall()  
 for topic in topics:  
 topic\_names\_list.append(topic[0])  
 await bot.send\_message(call.from\_user.id,  
 "<b>Available notes on this theme:</b>\n" + '\n'.join(topic\_names\_list),  
 parse\_mode="HTML")  
 await bot.send\_message(call.from\_user.id, "<b>📝Enter TITLE of note to delete it📝</b>",  
 parse\_mode="HTML")  
 await NoteEdit.DeleteTopic.set()  
  
 @dp.message\_handler(state=NoteEdit.DeleteTopic)  
 async def delete\_topic(message: types.Message):  
 if message.text in topic\_names\_list:  
 with connection.cursor() as curs:  
 note\_delete = f"""DELETE   
 FROM notatics   
 WHERE themes\_id={theme\_id}   
 AND title = '{message.text}';"""  
 curs.execute(note\_delete)  
 connection.commit()  
 mes = await bot.send\_message(message.from\_user.id, f"<b>✅Note was successfully deleted✅</b>",  
 parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await mes.edit\_text(f"👤<b>Hi! {message.from\_user.first\_name if message.from\_user.first\_name else ''} "  
 f"{message.from\_user.last\_name if message.from\_user.last\_name else ''}\n I'm "  
 f"bot Student Assistant.</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_menu)  
 await NoteEdit.next()  
 topic\_names\_list.remove(message.text)  
 if not topic\_names\_list:  
 with connection.cursor() as curs:  
 themes\_delete = f"""DELETE FROM themes WHERE id={theme\_id};"""  
 curs.execute(themes\_delete)  
 connection.commit()  
 else:  
 await bot.send\_message(message.from\_user.id, "<b>❌THERE IS NO NAMES LIKE THIS❌</b>\nTRY AGAIN.",  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="🔎Search note🔎")  
async def search\_note\_menu(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "<b>Choose action to perform</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_search\_menu)  
  
 @dp.callback\_query\_handler(text="🔎Show all notes🔎")  
 async def show\_all\_notes(c: types.CallbackQuery):  
 tuple\_list = await find\_tuple\_list(c)  
 output\_list = []  
 for i in tuple\_list:  
 notes\_text = f"Title: {i[0]}\nText: {i[1]}"  
 output\_list.append(notes\_text)  
 await bot.send\_message(c.from\_user.id, "<b>Notes:</b>\n" + "\n\n".join(output\_list), parse\_mode="HTML")  
  
 @dp.callback\_query\_handler(text="🔎Search by theme🔎")  
 async def search\_by\_theme(c: types.CallbackQuery):  
 list\_themes = []  
 tuple\_list = await find\_tuple\_list(c)  
 with connection.cursor() as cursor:  
 select\_id = f"""SELECT themes FROM themes WHERE user\_id={c.from\_user.id}"""  
 cursor.execute(select\_id)  
 result = cursor.fetchall()  
 if result:  
 for themes in result:  
 for theme in themes:  
 list\_themes.append(theme)  
 await bot.send\_message(c.from\_user.id, "<b>Available themes:</b>\n" + '\n'.join(list\_themes),  
 parse\_mode="HTML")  
 await bot.send\_message(c.from\_user.id, "<b>📝Enter THEME of note📝</b>", parse\_mode="HTML")  
 await NoteSearch.SearchNote.set()  
 else:  
 note = await bot.send\_message(c.from\_user.id, "<b>❌THERE IS NO THEMES❌</b>", parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await note.edit\_text(f"👤<b>Hi! {c.from\_user.first\_name if c.from\_user.first\_name else ''} "  
 f"{c.from\_user.last\_name if c.from\_user.last\_name else ''}\n I'm "  
 f"bot Student Assistant.</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_menu)  
  
 @dp.message\_handler(state=NoteSearch.SearchNote)  
 async def search\_notes\_for\_themes(message: types.Message):  
 nonlocal tuple\_list  
 output\_list = []  
 with connection.cursor() as curs:  
 curs.execute(f"""SELECT id FROM themes WHERE themes = '{message.text}'""")  
 this\_theme\_id = curs.fetchone()[0]  
 for i in tuple\_list:  
 if this\_theme\_id == i[2]:  
 notes\_text = f"Title: {i[0]}\nText: {i[1]}"  
 output\_list.append(notes\_text)  
 await bot.send\_message(message.from\_user.id, "<b>Notes:</b>\n" + "\n\n".join(output\_list),  
 parse\_mode="HTML")  
 await NoteSearch.next()  
  
 @dp.callback\_query\_handler(text="🔎Search by name🔎")  
 async def search\_by\_name(call\_this: types.CallbackQuery):  
 tuple\_list = await find\_tuple\_list(call\_this)  
 this\_title = []  
 iteration\_list = []  
 with connection.cursor() as curs:  
  
 for i in tuple\_list:  
 curs.execute(f"""SELECT title FROM notatics WHERE themes\_id = {i[2]}""")  
 temp = curs.fetchall()[iteration\_list.count(i[2])][0]  
 if temp not in this\_title:  
 this\_title.append(temp)  
 iteration\_list.append(i[2])  
 if this\_title:  
 await bot.send\_message(call\_this.from\_user.id, "<b>Titles:</b>\n" + "\n".join(this\_title),  
 parse\_mode="HTML")  
 await bot.send\_message(call\_this.from\_user.id, "<b>📝Enter TITLE of note📝</b>", parse\_mode="HTML")  
 await NoteSearchByTitle.SearchByTitle.set()  
 else:  
 note = await bot.send\_message(call\_this.from\_user.id, "<b>❌THERE IS NO TITLE❌</b>", parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await note.edit\_text(f"👤<b>Hi! {call\_this.from\_user.first\_name if call\_this.from\_user.first\_name else ''} "  
 f"{call\_this.from\_user.last\_name if call\_this.from\_user.last\_name else ''}\n I'm "  
 f"bot Student Assistant.</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_menu)  
  
 @dp.message\_handler(state=NoteSearchByTitle.SearchByTitle)  
 async def search\_notes\_for\_title(message: types.Message):  
 output\_list = []  
  
 for j in tuple\_list:  
 if j[0] == message.text:  
 notes\_text = f"Title: {j[0]}\nText: {j[1]}"  
 output\_list.append(notes\_text)  
  
 await bot.send\_message(message.from\_user.id, "<b>Notes:</b>\n" + "\n\n".join(output\_list),  
 parse\_mode="HTML")  
 await NoteSearchByTitle.next()  
  
  
async def find\_tuple\_list(c):  
 await c.message.delete()  
 id\_themes\_list = []  
 result\_list = []  
 with connection.cursor() as cursor:  
 cursor.execute(f"""SELECT id FROM themes WHERE user\_id = {c.from\_user.id}""")  
 result = cursor.fetchall()  
 for id\_themes in result:  
 id\_themes\_list.append(id\_themes[0])  
  
 with connection.cursor() as cursor:  
 for id\_themes in id\_themes\_list:  
 cursor.execute(f"""SELECT title, text\_notatics, themes\_id FROM notatics WHERE themes\_id = {id\_themes}""")  
 result = cursor.fetchall()  
 result\_list.append(result)  
 tuple\_list = []  
 for i in result\_list:  
 for j in i:  
 tuple\_list.append(j)  
 return tuple\_list  
  
  
@dp.callback\_query\_handler(text="🖋️Edit note🖋️")  
async def edit\_note(call: types.CallbackQuery):  
 await call.message.delete()  
 topic\_names\_list = []  
 theme\_id = 0  
 title\_this = 0  
 with connection.cursor() as cursor:  
 select\_id = f"""SELECT themes FROM themes WHERE user\_id={call.from\_user.id}"""  
 cursor.execute(select\_id)  
 result = cursor.fetchall()  
 list\_themes = []  
 if result:  
 for themes in result:  
 for theme in themes:  
 list\_themes.append(theme)  
 await bot.send\_message(call.from\_user.id, "<b>Available themes:</b>\n" + '\n'.join(list\_themes),  
 parse\_mode="HTML")  
 if len(list\_themes):  
 await bot.send\_message(call.from\_user.id, "<b>📝Enter THEME of note to delete it📝</b>", parse\_mode="HTML")  
 await EditNote.SearchThemes.set()  
 else:  
 note = await bot.send\_message(call.from\_user.id, "<b>❌YOU HAVE NO THEMES❌</b>", parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await note.edit\_text(f"👤<b>Hi! {call.from\_user.first\_name if call.from\_user.first\_name else ''} "  
 f"{call.from\_user.last\_name if call.from\_user.last\_name else ''}\n I'm "  
 f"bot Student Assistant.</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_menu)  
  
 @dp.message\_handler(state=EditNote.SearchThemes)  
 async def delete\_note(message: types.Message):  
 nonlocal list\_themes  
  
 if message.text in list\_themes:  
 with connection.cursor() as curs:  
 nonlocal topic\_names\_list  
 nonlocal theme\_id  
 select\_theme\_id = f"""SELECT id FROM themes WHERE themes="{message.text}" AND user\_id={message.from\_user.id};"""  
 curs.execute(select\_theme\_id)  
 theme\_id = curs.fetchone()[0]  
 select\_topics = f"""SELECT title FROM notatics WHERE themes\_id= {theme\_id}"""  
 curs.execute(select\_topics)  
 topics = curs.fetchall()  
 for topic in topics:  
 topic\_names\_list.append(topic[0])  
 await bot.send\_message(call.from\_user.id,  
 "<b>Available notes on this theme:</b>\n" + '\n'.join(topic\_names\_list),  
 parse\_mode="HTML")  
 await bot.send\_message(call.from\_user.id, "<b>📝Enter TITLE of note to delete it📝</b>",  
 parse\_mode="HTML")  
 await EditNote.SearchTitle.set()  
  
 @dp.message\_handler(state=EditNote.SearchTitle)  
 async def delete\_topic(message: types.Message):  
 nonlocal title\_this  
 if message.text in topic\_names\_list:  
 title\_this = message.text  
 await bot.send\_message(message.from\_user.id, "<b>Enter new note text</b>", parse\_mode="HTML")  
 await EditNote.EnterText.set()  
 else:  
 await bot.send\_message(message.from\_user.id, "<b>❌THERE IS NO NAMES LIKE THIS❌</b>\nTRY AGAIN.",  
 parse\_mode="HTML")  
  
 @dp.message\_handler(state=EditNote.EnterText)  
 async def edit\_text(message: types.Message):  
 with connection.cursor() as curs:  
 curs.execute(f"""UPDATE notatics   
 SET text\_notatics = '{message.text}'   
 WHERE themes\_id={theme\_id}   
 AND title = '{title\_this}';""")  
 connection.commit()  
 this\_note = await bot.send\_message(message.from\_user.id, "<b>✅NOTE SUCCESSFULLY EDITED✅</b>",  
 parse\_mode="HTML")  
  
 await asyncio.sleep(3)  
 await EditNote.next()  
 await this\_note.edit\_text(  
 f"👤<b>Hi! {message.from\_user.first\_name if message.from\_user.first\_name else ''} "  
 f"{message.from\_user.last\_name if message.from\_user.last\_name else ''}\n I'm "  
 f"bot Student Assistant.</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_menu)  
  
  
@dp.callback\_query\_handler(text="📅Schedule📅")  
async def schedule\_menu(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "<b>Choose action to perform</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_schedule\_menu)  
  
  
@dp.callback\_query\_handler(text="🔧Settings🔧")  
async def schedule\_settings(call: types.CallbackQuery):  
 await call.message.delete()  
 with connection.cursor() as cursor:  
 cursor.execute(f"""SELECT group\_id FROM users WHERE id = '{call.from\_user.id}';""")  
 group\_id = cursor.fetchone()[0]  
 if group\_id:  
 cursor.execute(f"""SELECT group\_name FROM group\_name WHERE id = {group\_id}""")  
 group\_name = cursor.fetchone()[0]  
 else:  
 group\_name = "❌"  
 await bot.send\_message(call.from\_user.id, f"<b>Current settings</b>:\nGroup: {group\_name}", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_schedule\_settings)  
  
  
@dp.callback\_query\_handler(text="➕Add group➕")  
async def add\_group(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "<b>Enter your group</b>", parse\_mode="HTML")  
 founded\_groups = []  
 await GroupAdd.AddGroup.set()  
  
 @dp.message\_handler(state=GroupAdd.AddGroup)  
 async def find\_group(message: types.Message):  
 nonlocal founded\_groups  
 groups\_name = []  
 groups\_correct\_name = []  
 with connection.cursor() as cursor:  
 cursor.execute("SELECT group\_name FROM group\_name")  
 groups\_name.append(cursor.fetchall())  
 for group in groups\_name[0]:  
 groups\_correct\_name.append(group[0])  
 for group\_name in filter(re.compile(message.text.lower()).match, groups\_correct\_name):  
 founded\_groups.append(group\_name)  
 if not founded\_groups or not message.text[:1].isalpha():  
 await bot.send\_message(message.from\_user.id, "<b>Invalid name (at least two letters), try again</b>",  
 parse\_mode="HTML")  
 else:  
 await bot.send\_message(message.from\_user.id,  
 "<b>Founded group:\n</b>" + '\n'.join(founded\_groups),  
 parse\_mode="HTML")  
 await bot.send\_message(message.from\_user.id, "<b>Choose group from this list, and write it</b>",  
 parse\_mode="HTML")  
 await GroupAdd.SelectGroup.set()  
  
 @dp.message\_handler(state=GroupAdd.SelectGroup)  
 async def select\_group(message: types.Message):  
 nonlocal founded\_groups  
 if message.text.lower() in founded\_groups:  
 with connection.cursor() as curs:  
 curs.execute(f"""SELECT id FROM group\_name WHERE group\_name = "{message.text.lower()}";""")  
 group\_id = curs.fetchone()[0]  
 curs.execute("SET FOREIGN\_KEY\_CHECKS=OFF;")  
 curs.execute(f""" UPDATE users SET group\_id = {group\_id} WHERE id="{message.from\_user.id}";""")  
 connection.commit()  
 note = await bot.send\_message(message.from\_user.id, "<b>✅GROUP SUCCESSFULLY ADDED✅</b>",  
 parse\_mode="HTML")  
 await GroupAdd.next()  
 await asyncio.sleep(3)  
 await note.edit\_text(f"👤<b>Hi! {message.from\_user.first\_name if message.from\_user.first\_name else ''} "  
 f"{message.from\_user.last\_name if message.from\_user.last\_name else ''}\n I'm "  
 f"bot Student Assistant.</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_menu)  
 else:  
 await bot.send\_message(message.from\_user.id,  
 "<b>❌Invalid name, there is no such name in groups!❌\nTry again</b>",  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="➖Delete group➖")  
async def delete\_group(call: types.CallbackQuery):  
 await call.message.delete()  
 with connection.cursor() as cursor:  
 cursor.execute("SET FOREIGN\_KEY\_CHECKS=OFF;")  
 cursor.execute(  
 f"""UPDATE users SET group\_id = NULL WHERE id = '{call.from\_user.id}';""")  
 connection.commit()  
 note = await bot.send\_message(call.from\_user.id, "<b>✅GROUP SUCCESSFULLY DELETED✅</b>",  
 parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await note.edit\_text(f"👤<b>Hi! {call.from\_user.first\_name if call.from\_user.first\_name else ''} "  
 f"{call.from\_user.last\_name if call.from\_user.last\_name else ''}\n I'm "  
 f"bot Student Assistant.</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_menu)  
  
  
@dp.callback\_query\_handler(text="➕Add schedule➕")  
async def add\_lesson\_menu(call: types.CallbackQuery):  
 with connection.cursor() as curs:  
 curs.execute(f"""SELECT group\_id FROM users WHERE id = '{call.from\_user.id}';""")  
 result = curs.fetchone()[0]  
 await call.message.delete()  
 if result:  
 await bot.send\_message(call.from\_user.id, "<b>Choose day of weekday</b>",  
 reply\_markup=markup.inline\_keyboard\_day\_of\_week, parse\_mode="HTML")  
 else:  
 note = await bot.send\_message(call.from\_user.id, "<b>❌You can not use schedule before you choose group❌</b>",  
 parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await note.edit\_text(f"<b>Current settings</b>:\nGroup: ❌", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_schedule\_settings)  
  
  
@dp.callback\_query\_handler(text="Monday")  
async def add\_monday(call: types.CallbackQuery):  
 global selected\_day  
 selected\_day = 1  
 await add\_lessons\_day\_of\_week(call)  
  
  
@dp.callback\_query\_handler(text="Tuesday")  
async def add\_monday(call: types.CallbackQuery):  
 global selected\_day  
 selected\_day = 2  
 await add\_lessons\_day\_of\_week(call)  
  
  
@dp.callback\_query\_handler(text="Wednesday")  
async def add\_monday(call: types.CallbackQuery):  
 global selected\_day  
 selected\_day = 3  
 await add\_lessons\_day\_of\_week(call)  
  
  
@dp.callback\_query\_handler(text="Thursday")  
async def add\_monday(call: types.CallbackQuery):  
 global selected\_day  
 selected\_day = 4  
 await add\_lessons\_day\_of\_week(call)  
  
  
@dp.callback\_query\_handler(text="Friday")  
async def add\_monday(call: types.CallbackQuery):  
 global selected\_day  
 selected\_day = 5  
 await add\_lessons\_day\_of\_week(call)  
  
  
@dp.callback\_query\_handler(text="Saturday")  
async def add\_monday(call: types.CallbackQuery):  
 global selected\_day  
 selected\_day = 6  
 await add\_lessons\_day\_of\_week(call)  
  
  
async def add\_lessons\_day\_of\_week(call):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "<b>Add lesson</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_pair)  
  
  
@dp.callback\_query\_handler(text="First")  
async def first(call: types.CallbackQuery):  
 await call.message.delete()  
 global selected\_pair  
 selected\_pair = 1  
 await bot.send\_message(call.from\_user.id, "<b>ADD FIRST PAIR INFO</b>",  
 reply\_markup=markup.inline\_keyboard\_add\_pair,  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="Second")  
async def second(call: types.CallbackQuery):  
 await call.message.delete()  
 global selected\_pair  
 selected\_pair = 2  
 await bot.send\_message(call.from\_user.id, "<b>ADD SECOND PAIR INFO</b>",  
 reply\_markup=markup.inline\_keyboard\_add\_pair,  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="Third")  
async def third(call: types.CallbackQuery):  
 await call.message.delete()  
 global selected\_pair  
 selected\_pair = 3  
 await bot.send\_message(call.from\_user.id, "<b>ADD THIRD PAIR INFO</b>",  
 reply\_markup=markup.inline\_keyboard\_add\_pair,  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="Fourth")  
async def fourth(call: types.CallbackQuery):  
 await call.message.delete()  
 global selected\_pair  
 selected\_pair = 4  
 await bot.send\_message(call.from\_user.id, "<b>ADD FOURTH PAIR INFO</b>",  
 reply\_markup=markup.inline\_keyboard\_add\_pair,  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="Fifth")  
async def fifth(call: types.CallbackQuery):  
 await call.message.delete()  
 global selected\_pair  
 selected\_pair = 5  
 await bot.send\_message(call.from\_user.id, "<b>ADD FIFTH PAIR INFO</b>",  
 reply\_markup=markup.inline\_keyboard\_add\_pair,  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="Sixth")  
async def sixth(call: types.CallbackQuery):  
 await call.message.delete()  
 global selected\_pair  
 selected\_pair = 6  
 await bot.send\_message(call.from\_user.id, "<b>ADD SIXTH PAIR INFO</b>",  
 reply\_markup=markup.inline\_keyboard\_add\_pair,  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="Week")  
async def week(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, f"<b>SELECT WEEK</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_week\_menu)  
  
  
@dp.callback\_query\_handler(text="EVEN")  
async def even(call: types.CallbackQuery):  
 await choose\_week(call, 1, "EVEN")  
  
  
@dp.callback\_query\_handler(text="ODD")  
async def odd(call: types.CallbackQuery):  
 await choose\_week(call, 2, "ODD")  
  
  
async def choose\_week(call, select\_week, string\_week):  
 await call.message.delete()  
 global selected\_week  
 global update\_week  
  
 with connection.cursor() as curs:  
 curs.execute(  
 f"""SELECT week\_number FROM schedules   
 WHERE user\_id='{call.from\_user.id}' AND day\_of\_week = {selected\_day} AND class\_id = {selected\_pair};""")  
 information = curs.fetchall()  
 weeks = [x[0] for x in information]  
 if select\_week in weeks:  
 update\_week = True  
 selected\_week = select\_week  
 note = await bot.send\_message(call.from\_user.id, f"<b>✅YOU SELECT {string\_week} WEEK✅</b>", parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await note.edit\_text("<b>ADD PAIR INFO</b>",  
 reply\_markup=markup.inline\_keyboard\_add\_pair,  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="Type of class")  
async def type\_of\_class(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, f"<b>SELECT TYPE OF LESSON</b>", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_type\_of\_lesson\_menu)  
  
  
@dp.callback\_query\_handler(text="Lecture")  
async def lecture(call: types.CallbackQuery):  
 await select\_type(call, 2, "LECTURE")  
  
  
@dp.callback\_query\_handler(text="Lab")  
async def lab(call: types.CallbackQuery):  
 await select\_type(call, 3, "LAB")  
  
  
@dp.callback\_query\_handler(text="Practice")  
async def practice(call: types.CallbackQuery):  
 await select\_type(call, 1, "PRACTICE")  
  
  
async def select\_type(call, select\_type\_class, string\_type\_of\_class):  
 await call.message.delete()  
 global selected\_type\_of\_class  
 selected\_type\_of\_class = select\_type\_class  
 note = await bot.send\_message(call.from\_user.id, f"<b>✅YOU SELECT {string\_type\_of\_class}✅</b>",  
 parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await note.edit\_text("<b>ADD PAIR INFO</b>",  
 reply\_markup=markup.inline\_keyboard\_add\_pair,  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="Subject name")  
async def subject\_name(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "<b>🖊Enter subject🖊</b>", parse\_mode="HTML")  
 await AddPair.AddSubject.set()  
  
 @dp.message\_handler(state=AddPair.AddSubject)  
 async def add\_subject(message: types.Message):  
 global selected\_subject  
 selected\_subject = message.text  
 await AddPair.next()  
 note = await bot.send\_message(call.from\_user.id, f"<b>✅YOU SELECT SUBJECT✅</b>",  
 parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await note.edit\_text("<b>ADD PAIR INFO</b>",  
 reply\_markup=markup.inline\_keyboard\_add\_pair,  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="Audience")  
async def audience(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "<b>🖊Enter audience🖊</b>", parse\_mode="HTML")  
 await AddAudiences.AddAudience.set()  
  
 @dp.message\_handler(state=AddAudiences.AddAudience)  
 async def add\_audience(message: types.Message):  
 global selected\_audience  
 selected\_audience = message.text  
 await AddAudiences.next()  
  
 note = await bot.send\_message(call.from\_user.id, f"<b>✅YOU SELECT AUDIENCE✅</b>",  
 parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await note.edit\_text("<b>ADD PAIR INFO</b>",  
 reply\_markup=markup.inline\_keyboard\_add\_pair,  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="Teacher")  
async def teacher(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "<b>🖊Enter teacher🖊</b>", parse\_mode="HTML")  
 await AddTeachers.AddTeacher.set()  
  
 @dp.message\_handler(state=AddTeachers.AddTeacher)  
 async def add\_audience(message: types.Message):  
 global selected\_teacher  
 selected\_teacher = message.text  
 await AddTeachers.next()  
  
 note = await bot.send\_message(call.from\_user.id, f"<b>✅YOU SELECT TEACHER✅</b>",  
 parse\_mode="HTML")  
 await asyncio.sleep(3)  
 await note.edit\_text("<b>ADD PAIR INFO</b>",  
 reply\_markup=markup.inline\_keyboard\_add\_pair,  
 parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="✅Ready✅")  
async def ready(call: types.CallbackQuery):  
 await call.message.delete()  
 global selected\_day, selected\_pair, selected\_type\_of\_class, selected\_week, selected\_audience, selected\_subject, \  
 selected\_teacher, selected\_audience\_id, selected\_subject\_id, selected\_teacher\_id  
 if selected\_subject:  
 selected\_subject\_id = await push\_info('subjects', 'subject\_name', selected\_subject)  
 if selected\_audience:  
 selected\_audience\_id = await push\_info('audiences', 'audience', selected\_audience)  
 if selected\_audience:  
 selected\_teacher\_id = await push\_info('teacher', 'teacher\_name', selected\_teacher)  
 with connection.cursor() as cursor:  
 cursor.execute(f"""SELECT group\_id FROM users WHERE id = {call.from\_user.id}""")  
 group\_id = cursor.fetchone()  
 note1 = 0  
 note2 = 0  
 note3 = 0  
 note4 = 0  
 note5 = 0  
  
 if not selected\_type\_of\_class:  
 note1 = await bot.send\_message(call.from\_user.id,  
 "<b>You must enter all fields. Please, select type of class. </b>",  
 parse\_mode="HTML")  
  
 if not selected\_week:  
 note2 = await bot.send\_message(call.from\_user.id,  
 "<b>You must enter all fields. Please, select week, odd or even.</b>",  
 parse\_mode="HTML")  
  
 if not selected\_audience\_id:  
 note3 = await bot.send\_message(call.from\_user.id,  
 "<b>You must enter all fields. Please, write down audience.</b>",  
 parse\_mode="HTML")  
  
 if not selected\_subject\_id:  
 note4 = await bot.send\_message(call.from\_user.id,  
 "<b>You must enter all fields. Please, write down subject.</b>",  
 parse\_mode="HTML")  
  
 if not selected\_teacher\_id:  
 note5 = await bot.send\_message(call.from\_user.id,  
 "<b>You must enter all fields. Please, write down teacher.</b>",  
 parse\_mode="HTML")  
  
 if not selected\_audience\_id or not selected\_week or not selected\_audience\_id or not selected\_subject\_id \  
 or not selected\_teacher\_id:  
 await asyncio.sleep(3)  
 if not isinstance(note1, int):  
 await note1.delete()  
 if not isinstance(note2, int):  
 await note2.delete()  
 if not isinstance(note3, int):  
 await note3.delete()  
 if not isinstance(note4, int):  
 await note4.delete()  
 if not isinstance(note5, int):  
 await note5.delete()  
  
 await bot.send\_message(call.from\_user.id, "<b>ADD PAIR INFO</b>", reply\_markup=markup.inline\_keyboard\_add\_pair,  
 parse\_mode="HTML")  
 else:  
 await add\_schedules(call, selected\_day, selected\_pair, selected\_type\_of\_class, selected\_week,  
 selected\_audience\_id, selected\_subject\_id, call.from\_user.id, group\_id[0],  
 selected\_teacher\_id)  
  
 selected\_subject = 0  
 selected\_audience = 0  
 selected\_day = 0  
 selected\_pair = 0  
 selected\_type\_of\_class = 0  
 selected\_week = 0  
 selected\_teacher = 0  
  
 await bot.send\_message(call.from\_user.id, "<b>Choose day of weekday</b>",  
 reply\_markup=markup.inline\_keyboard\_day\_of\_week, parse\_mode="HTML")  
  
  
async def push\_info(table, line, selected):  
 with connection.cursor() as cursor:  
 cursor.execute(f"""SELECT id FROM {table} WHERE {line}='{selected}'""")  
 result = cursor.fetchone()  
 if result:  
 return result[0]  
 else:  
 with connection.cursor() as cur:  
 cur.execute(f"""INSERT INTO {table}({line}) VALUES("{selected}");""")  
 connection.commit()  
 with connection.cursor() as c:  
 c.execute(f"""SELECT id FROM {table} WHERE {line} = '{selected}';""")  
 return c.fetchone()[0]  
  
  
async def add\_schedules(call, day, pair, type\_class, weeks, audiences, subject, user\_id, group\_id, teacher\_id):  
 global update\_week  
 if update\_week:  
 with connection.cursor() as cur:  
 cur.execute(  
 f"""UPDATE schedules SET group\_id = {group\_id}, subject\_name = {subject}, type\_of\_class = {type\_class},   
 audience = {audiences}, teacher = {teacher\_id} WHERE user\_id = '{call.from\_user.id}'   
 AND day\_of\_week = {selected\_day} AND class\_id = {selected\_pair} AND week\_number = {selected\_week}""")  
 connection.commit()  
 update\_week = False  
 else:  
 with connection.cursor() as curs:  
 curs.execute(  
 f"""INSERT INTO schedules(  
 day\_of\_week, class\_id, type\_of\_class, week\_number ,audience, subject\_name, user\_id, group\_id, teacher)   
 VALUES({day}, {pair}, {type\_class}, {weeks}, {audiences}, {subject},   
 '{user\_id}', {group\_id}, {teacher\_id});"""  
 )  
 connection.commit()  
  
  
@dp.callback\_query\_handler(text="⌚Today schedule⌚")  
async def today\_schedule(call: types.CallbackQuery):  
 await call.message.delete()  
 today = datetime.today().isoweekday()  
 today\_week = await even\_or\_odd()  
 group\_id\_number = 0  
 today\_schedules = []  
 await day\_schedule(call, today, today\_week, group\_id\_number, today\_schedules)  
  
  
async def even\_or\_odd():  
 now = datetime.now()  
 sep = datetime(now.year if now.month >= 9 else now.year - 1, 9, 1)  
 return 2 if not ((((now - tmd(days=now.weekday())) -  
 (sep - tmd(days=sep.weekday()))).days // 7) % 2) else 1  
  
  
async def day\_schedule(call, day, weeks, group\_id, today\_schedules):  
 with connection.cursor() as curs:  
 curs.execute(f"""SELECT group\_id FROM users WHERE id = '{call.from\_user.id}';""")  
 group\_id = curs.fetchone()[0]  
 if day == 7:  
 await bot.send\_message(call.from\_user.id, f"<b>Неділя\nВихідний\nLesson: ❌</b>")  
 else:  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f"""SELECT class\_id, type\_of\_class,  
 audience, subject\_name, teacher FROM schedules  
 WHERE day\_of\_week = {day} AND user\_id = "{call.from\_user.id}" AND group\_id = {group\_id}   
 AND week\_number = {weeks};""")  
 today\_info = cursor.fetchall()  
 with connection.cursor() as cur:  
 cur.execute(f"""SELECT name\_day\_of\_week FROM days\_of\_week WHERE id = {day};""")  
 day = cur.fetchone()[0]  
 today\_schedules.append(day)  
 for today\_information in today\_info:  
 cur.execute(f"""SELECT id, time\_of\_start, time\_of\_end FROM classes WHERE id = {today\_information[0]}""")  
 time\_of\_lesson = cur.fetchall()  
 cur.execute(f"""SELECT subject\_name FROM subjects WHERE id = {today\_information[3]}""")  
 subject\_now = cur.fetchone()[0]  
 cur.execute(f"""SELECT type\_of\_class FROM type\_of\_classes WHERE id = {today\_information[1]}""")  
 type\_of\_class\_now = cur.fetchone()[0]  
 cur.execute(f"""SELECT teacher\_name FROM teacher WHERE id = {today\_information[4]}""")  
 teacher\_now = cur.fetchone()[0]  
 cur.execute(f"""SELECT audience FROM audiences WHERE id = {today\_information[2]}""")

audience\_now = cur.fetchone()[0]  
 for pair\_time in time\_of\_lesson:  
 today\_schedules.append(f"Lesson: {pair\_time[0]}")  
 today\_schedules.append("Time of start: " + str(pair\_time[1]))  
 today\_schedules.append("Time of end: " + str(pair\_time[2]))  
 today\_schedules.append(f"Subject: {subject\_now}")  
 today\_schedules.append(f"Type of lesson: {type\_of\_class\_now}")  
 today\_schedules.append(f"Teacher: {teacher\_now}")  
 today\_schedules.append(f"Audience: {audience\_now}\n")  
 await bot.send\_message(call.from\_user.id, "<b>" + '\n'.join(today\_schedules) + "</b>", parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="📅Next day schedule📅")  
async def back(call: types.CallbackQuery):  
 await call.message.delete()  
 next\_day = datetime.today().isoweekday() + 1  
 next\_week = await even\_or\_odd()  
 if next\_day == 8:  
 next\_day = 1  
 if next\_week == 1:  
 next\_week = 2  
 else:  
 next\_week = 1  
 group\_id\_number = 0  
 next\_day\_schedules = []  
 await day\_schedule(call, next\_day, next\_week, group\_id\_number, next\_day\_schedules)  
  
  
@dp.callback\_query\_handler(text="⏭️Next pair⏭️")  
async def next\_pair(call: types.CallbackQuery):  
 await call.message.delete()  
 next\_pair\_day = datetime.today().isoweekday()  
 next\_pair\_weeks = await even\_or\_odd()  
 next\_pair\_this\_time = datetime.now().time()  
 next\_pair\_group\_id = 0  
 today\_schedules = []  
 with connection.cursor() as curs:  
 curs.execute(f"""SELECT group\_id FROM users WHERE id = '{call.from\_user.id}';""")  
 next\_pair\_group\_id = curs.fetchone()[0]  
  
 today\_schedules = await next\_pair\_schedule(call, next\_pair\_day, next\_pair\_weeks, next\_pair\_group\_id,  
 next\_pair\_this\_time)  
 while not today\_schedules:  
 next\_pair\_day += 1  
 next\_pair\_this\_time = datetime.strptime("00:00:00", "%H:%M:%S").time()  
 if next\_pair\_day == 8:  
 next\_pair\_day = 1  
 if next\_pair\_weeks == 1:  
 next\_pair\_weeks = 2  
 else:  
 next\_pair\_weeks = 1  
 today\_schedules = await next\_pair\_schedule(call, next\_pair\_day, next\_pair\_weeks, next\_pair\_group\_id,  
 next\_pair\_this\_time)  
 await bot.send\_message(call.from\_user.id, "<b>" + '\n'.join(today\_schedules) + "</b>", parse\_mode="HTML")  
  
  
async def next\_pair\_schedule(call, day, weeks, group\_id, this\_time):  
 today\_schedules = []  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f"""SELECT class\_id, type\_of\_class,  
 audience, subject\_name, teacher FROM schedules  
 WHERE day\_of\_week = {day} AND user\_id = "{call.from\_user.id}" AND group\_id = {group\_id}   
 AND week\_number = {weeks};""")  
 today\_info = cursor.fetchall()  
 with connection.cursor() as cur:  
 cur.execute(f"""SELECT name\_day\_of\_week FROM days\_of\_week WHERE id = {day};""")  
 day = cur.fetchone()[0]  
 for today\_information in today\_info:  
 cur.execute(f"""SELECT id, time\_of\_start, time\_of\_end FROM classes WHERE id = {today\_information[0]}""")  
 time\_of\_lesson = cur.fetchall()  
 cur.execute(f"""SELECT subject\_name FROM subjects WHERE id = {today\_information[3]}""")  
 subject\_now = cur.fetchone()[0]  
 cur.execute(f"""SELECT type\_of\_class FROM type\_of\_classes WHERE id = {today\_information[1]}""")  
 type\_of\_class\_now = cur.fetchone()[0]  
 cur.execute(f"""SELECT teacher\_name FROM teacher WHERE id = {today\_information[4]}""")  
 teacher\_now = cur.fetchone()[0]  
 cur.execute(f"""SELECT audience FROM audiences WHERE id = {today\_information[2]}""")  
 audience\_now = cur.fetchone()[0]  
 for pair\_time in time\_of\_lesson:  
 lesson\_time = str(pair\_time[1])  
 if this\_time < datetime.strptime(lesson\_time, "%H:%M:%S").time():  
 today\_schedules.append(day)  
 today\_schedules.append(f"Lesson: {pair\_time[0]}")  
 today\_schedules.append("Time of start: " + str(pair\_time[1]))  
 today\_schedules.append("Time of end: " + str(pair\_time[2]))  
 today\_schedules.append(f"Subject: {subject\_now}")  
 today\_schedules.append(f"Type of lesson: {type\_of\_class\_now}")  
 today\_schedules.append(f"Teacher: {teacher\_now}")  
 today\_schedules.append(f"Audience: {audience\_now}\n")  
 return today\_schedules  
  
  
@dp.callback\_query\_handler(text="🟡This week schedule🟡")  
async def this\_week\_schedule(call: types.CallbackQuery):  
 await call.message.delete()  
 await week\_schedule(call, await even\_or\_odd())  
  
  
@dp.callback\_query\_handler(text="⚫Next week schedule⚫")  
async def this\_week\_schedule(call: types.CallbackQuery):  
 await call.message.delete()  
 current\_week = await even\_or\_odd()  
 if current\_week == 1:  
 current\_week = 2  
 else:  
 current\_week = 1  
 await week\_schedule(call, current\_week)  
  
  
async def week\_schedule(call, this\_week\_week):  
 today\_schedules = []  
 with connection.cursor() as curs:  
 curs.execute(f"""SELECT group\_id FROM users WHERE id = '{call.from\_user.id}';""")  
 group\_id = curs.fetchone()[0]  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f"""SELECT day\_of\_week, class\_id, type\_of\_class,  
 audience, subject\_name, teacher FROM schedules  
 WHERE user\_id = "{call.from\_user.id}" AND group\_id = {group\_id} AND week\_number = {this\_week\_week};""")  
 this\_week\_info = cursor.fetchall()  
 with connection.cursor() as cur:  
  
 for today\_information in this\_week\_info:  
 cur.execute(f"""SELECT name\_day\_of\_week FROM days\_of\_week WHERE id = {today\_information[0]};""")  
 day = cur.fetchone()[0]  
 cur.execute(f"""SELECT id, time\_of\_start, time\_of\_end FROM classes WHERE id = {today\_information[1]}""")  
 time\_of\_lesson = cur.fetchall()  
 cur.execute(f"""SELECT subject\_name FROM subjects WHERE id = {today\_information[4]}""")  
 subject\_now = cur.fetchone()[0]  
 cur.execute(f"""SELECT type\_of\_class FROM type\_of\_classes WHERE id = {today\_information[2]}""")  
 type\_of\_class\_now = cur.fetchone()[0]  
 cur.execute(f"""SELECT teacher\_name FROM teacher WHERE id = {today\_information[5]}""")  
 teacher\_now = cur.fetchone()[0]  
 cur.execute(f"""SELECT audience FROM audiences WHERE id = {today\_information[3]}""")  
 audience\_now = cur.fetchone()[0]  
 if day not in today\_schedules:  
 today\_schedules.append(day)  
 for pair\_time in time\_of\_lesson:  
 today\_schedules.append(f"Lesson: {pair\_time[0]}")  
 today\_schedules.append("Time of start: " + str(pair\_time[1]))  
 today\_schedules.append("Time of end: " + str(pair\_time[2]))  
 today\_schedules.append(f"Subject: {subject\_now}")  
 today\_schedules.append(f"Type of lesson: {type\_of\_class\_now}")  
 today\_schedules.append(f"Teacher: {teacher\_now}")  
 today\_schedules.append(f"Audience: {audience\_now}\n")  
 await bot.send\_message(call.from\_user.id, "<b>" + '\n'.join(today\_schedules) + "</b>", parse\_mode="HTML")  
  
  
@dp.callback\_query\_handler(text="🔙")  
async def back(call: types.CallbackQuery):  
 *""" back to main menu """* await call.message.delete()  
 await bot.send\_message(call.from\_user.id,  
 f"👤<b>Hi! {call.from\_user.first\_name if call.from\_user.first\_name else ''} "  
 f"{call.from\_user.last\_name if call.from\_user.last\_name else ''}\n I'm "  
 f"bot Student Assistant.</b>", parse\_mode="HTML", reply\_markup=markup.inline\_keyboard\_menu)  
  
  
""" Finance handlers """  
  
  
@dp.callback\_query\_handler(text="BACK\_TO\_FINANCE")  
async def back(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(  
 call.from\_user.id, "\*Choose action to perform\*", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_finance\_menu  
 )  
  
  
""" Finance handlers """  
  
  
@dp.callback\_query\_handler(text='💰Finance💰')  
async def note\_menu(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "\*Choose action to perform\*", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_finance\_menu)  
  
  
@dp.callback\_query\_handler(text='🏛️Budget🏛️')  
async def note\_menu(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(  
 call.from\_user.id, f'<b>Your budget</b>:\n<b>Daily</b>: {Finances.get\_budget\_daily\_limit(call.from\_user.id)}\n'  
 f'<b>Month</b>: {Finances.get\_budget\_month\_limit(call.from\_user.id)}', parse\_mode='HTML',  
 reply\_markup=markup.inline\_keyboard\_budget\_menu  
 )  
  
  
@dp.callback\_query\_handler(text='📈Statistic📈')  
async def note\_menu(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "\*Choose action to perform\*", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_statistic\_menu)  
  
  
@dp.callback\_query\_handler(text='OTHER\_FINANCE\_MENU')  
async def note\_menu(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "\*Choose action to perform\*", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_other\_menu)  
  
  
@dp.callback\_query\_handler(text='💸Add expense💸')  
async def add\_expense\_(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "Enter like this: 245 taxi")  
 await HandlerExpenses.AddExpensesState.set()  
  
 @dp.message\_handler(state=HandlerExpenses.AddExpensesState)  
 async def adding\_expense(message: types.Message):  
 try:  
 Finances.add\_expense(message['text'], message.from\_user.id)  
 except exceptions.AddExpenseError as exp:  
 await message.answer(str(exp))  
 await bot.send\_message(message.from\_user.id, 'ADDED', parse\_mode='HTML')  
 await HandlerExpenses.next()  
  
  
@dp.callback\_query\_handler(text='💰Add incomes💰')  
async def add\_incomes(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.from\_user.id, "Enter like this: 245 job")  
 await HandlerIncomes.AddIncomesState.set()  
  
 @dp.message\_handler(state=HandlerIncomes.AddIncomesState)  
 async def adding\_incomes(message: types.Message):  
 try:  
 Finances.add\_incomes(message['text'], message.from\_user.id)  
 except exceptions.AddIncomeError(str(message)) as exp:  
 await message.answer(str(exp))  
 await bot.send\_message(message.from\_user.id, 'ADDED', parse\_mode='HTML')  
 await HandlerIncomes.next()  
  
  
@dp.callback\_query\_handler(text='🖊️Edit budget🖊️')  
async def edit\_budget(call: types.CallbackQuery):  
 await call.message.delete()  
 await HandlerBudget.BudgetState.set()  
 await bot.send\_message(call.from\_user.id, "daily <i>number</i> month <i>number</i>", parse\_mode="HTML")  
  
 @dp.message\_handler(state=HandlerBudget.BudgetState)  
 async def editing\_budget(message: types.Message):  
 try:  
 Finances.edit\_budget(message['text'], message.from\_user.id)  
 except exceptions.ChangeBudgetError(str(message)) as exp:  
 await message.answer(str(exp))  
 await HandlerBudget.next()  
  
  
@dp.callback\_query\_handler(text='➕Add category➕')  
async def add\_category(call: types.CallbackQuery):  
 await call.message.delete()  
 await HandlerCategory.CategoriesState.set()  
 await bot.send\_message(  
 call.from\_user.id, "Enter category and key words like this:\n<b>products: products, food, eating</b>",  
 parse\_mode="HTML"  
 )  
  
 @dp.message\_handler(state=HandlerCategory.CategoriesState)  
 async def creating\_finance\_category(message: types.Message):  
 try:  
 Finances.create\_category\_finance(message['text'], message.from\_user.id)  
 except exceptions.AddCategoryError as exp:  
 await message.answer(str(exp))  
 return  
 await bot.send\_message(message.from\_user.id, 'Edited', parse\_mode='HTML')  
 await HandlerCategory.next()  
  
  
@dp.message\_handler(lambda message: message.text.startswith('/delexp'))  
async def del\_expense(message: types.Message):  
 try:  
 Finances.delete\_expense(int(message.text[7:]), message.from\_user.id)  
 except exceptions.DeleteError(str(message)) as exp:  
 await bot.send\_message(message.from\_user.id, f'{exp}', parse\_mode='HTML')  
 await bot.send\_message(message.from\_user.id, 'Deleted')  
  
  
@dp.message\_handler(lambda message: message.text.startswith('/delinc'))  
async def del\_expense(message: types.Message):  
 try:  
 Finances.delete\_expense(int(message.text[7:]), message.from\_user.id)  
 except exceptions.DeleteError(str(message)) as exp:  
 await bot.send\_message(message.from\_user.id, f'{exp}', parse\_mode='HTML')  
 await bot.send\_message(message.from\_user.id, 'Deleted')  
  
  
@dp.callback\_query\_handler(text='SEE\_CATEGORIES')  
async def categories\_viewing\_handler(call: types.CallbackQuery):  
 await call.message.delete()  
 categories\_data = Finances.see\_categories(call.from\_user.id)  
 if not categories\_data:  
 await bot.send\_message(call.from\_user.id, 'You haven`t any category yet', parse\_mode='HTML')  
 return  
 categories\_ = [f'<b>{category.name\_}:</b> {category.category\_text}' for category in categories\_data]  
 await bot.send\_message(call.from\_user.id, '<b>Your Categories:</b>\n\n' + '\n'.join(categories\_), parse\_mode='HTML')  
  
  
@dp.callback\_query\_handler(text='TODAY\_EXPENSES')  
async def today\_expenses\_handler(call: types.CallbackQuery):  
 await call.message.delete()  
 today\_expenses\_ = Finances.today\_expenses(call.from\_user.id)  
 if not today\_expenses\_:  
 await bot.send\_message(call.from\_user.id, 'Today expenses were not added', parse\_mode='HTML')  
 return  
 today\_expenses\_rows = [  
 f'{expense.amount} UAH on {expense.category\_name} — /delexp{expense.id}'  
 for expense in today\_expenses\_  
 ]  
 await bot.send\_message(call.from\_user.id, 'Today expenses\n' + '\n\n'.join(today\_expenses\_rows), parse\_mode='HTML')  
  
  
@dp.callback\_query\_handler(text='WEEK\_EXPENSES')  
async def week\_expenses\_handler(call: types.CallbackQuery):  
 await call.message.delete()  
 this\_week\_expenses\_ = Finances.this\_week\_expenses(call.from\_user.id)  
 if not this\_week\_expenses\_:  
 await bot.send\_message(call.from\_user.id, 'This week expenses were not added', parse\_mode='HTML')  
 return  
 this\_week\_expenses\_rows = [  
 f'{expense.amount} UAH on {expense.category\_name} — /delexp{expense.id}' for expense in this\_week\_expenses\_  
 ]  
 await bot.send\_message(  
 call.from\_user.id, 'This week expenses\n' + '\n\n'.join(this\_week\_expenses\_rows), parse\_mode='HTML'  
 )  
  
  
@dp.callback\_query\_handler(text='MONTH\_EXPENSES')  
async def month\_expenses\_handler(call: types.CallbackQuery):  
 await call.message.delete()  
 this\_month\_expenses\_ = Finances.this\_month\_expenses(call.from\_user.id)  
 if not this\_month\_expenses\_:  
 await bot.send\_message(call.from\_user.id, 'This month expenses were not added', parse\_mode='HTML')  
 return  
 this\_month\_expenses\_rows = [  
 f'{expense.amount} UAH on {expense.category\_name} — /delexp{expense.id}'  
 for expense in this\_month\_expenses\_  
 ]  
 await bot.send\_message(  
 call.from\_user.id, 'This month expenses\n' + '\n\n'.join(this\_month\_expenses\_rows), parse\_mode='HTML'  
 )  
  
  
@dp.callback\_query\_handler(text='TODAY\_INCOMES')  
async def today\_incomes\_handler(call: types.CallbackQuery):  
 await call.message.delete()  
 today\_expenses\_ = Finances.today\_incomes(call.from\_user.id)  
 if not today\_expenses\_:  
 await bot.send\_message(call.from\_user.id, 'Today incomes were not added', parse\_mode='HTML')  
 return  
 today\_expenses\_rows = [  
 f'{expense.amount} UAH on {expense.category\_name} — /delinc{expense.id}'  
 for expense in today\_expenses\_  
 ]  
 await bot.send\_message(  
 call.from\_user.id, 'Today incomes\n' + '\n\n'.join(today\_expenses\_rows), parse\_mode='HTML'  
 )  
  
  
@dp.callback\_query\_handler(text='WEEK\_INCOMES')  
async def week\_incomes\_handler(call: types.CallbackQuery):  
 await call.message.delete()  
 this\_week\_expenses\_ = Finances.this\_week\_incomes(call.from\_user.id)  
 if not this\_week\_expenses\_:  
 await bot.send\_message(call.from\_user.id, 'This week incomes were not added', parse\_mode='HTML')  
 return  
 this\_week\_expenses\_rows = [  
 f'{expense.amount} UAH on {expense.category\_name} — /delinc{expense.id}'  
 for expense in this\_week\_expenses\_  
 ]  
 await bot.send\_message(  
 call.from\_user.id, 'This week incomes\n' + "\n\n".join(this\_week\_expenses\_rows), parse\_mode='HTML'  
 )  
  
  
@dp.callback\_query\_handler(text='MONTH\_INCOMES')  
async def month\_incomes\_handler(call: types.CallbackQuery):  
 await call.message.delete()  
 this\_month\_expenses\_ = Finances.this\_month\_incomes(call.from\_user.id)  
 if not this\_month\_expenses\_:  
 await bot.send\_message(call.from\_user.id, 'This month incomes were not added', parse\_mode='HTML')  
 return  
 this\_month\_expenses\_rows = [  
 f'{expense.amount} UAH on {expense.category\_name} — /delinc{expense.id}'  
 for expense in this\_month\_expenses\_  
 ]  
 await bot.send\_message(  
 call.from\_user.id, 'This month incomes\n' + '\n\n'.join(this\_month\_expenses\_rows), parse\_mode='HTML'  
 )  
  
  
@dp.callback\_query\_handler(text='WEEK\_STATISTIC')  
async def this\_week\_statistic\_handler(call: types.CallbackQuery):  
 await call.message.delete()  
 file\_name\_ = Statistic.stats\_for\_current\_week(call.from\_user.id)  
 result\_ = Statistic.resulting\_for\_the\_current\_week(call.from\_user.id)  
 await bot.send\_photo(  
 call.from\_user.id, open(f'{file\_name\_}.png', 'rb'),  
 caption=f'<b>Total expenses:</b> {result\_[0]}\n<b>Total incomes:</b> {result\_[1] - result\_[2]}\n'  
 f'<b>Pure profit:</b> {result\_[2]}\n',  
 # f'<b>Of Budget:</b> {Finances.get\_budget\_month\_limit(call.from\_user.id)}'  
 parse\_mode='HTML'  
 )  
 await asyncio.sleep(10)  
 Statistic.delete\_stats\_image(file\_name\_)  
  
  
@dp.callback\_query\_handler(text='MONTH\_STATISTIC')  
async def this\_month\_statistic\_handler(call: types.CallbackQuery):  
 await call.message.delete()  
 file\_name\_ = Statistic.stats\_for\_current\_month(call.from\_user.id)  
 result\_ = Statistic.resulting\_for\_the\_current\_month(call.from\_user.id)  
 await bot.send\_photo(  
 call.from\_user.id, open(f'{file\_name\_}.png', 'rb'),  
 caption=f'<b>Total expenses:</b> {result\_[0]}\n<b>Total incomes:</b> '  
 f'{result\_[1]}\n<b>Pure profit: </b>{result\_[2]}\n',  
 # f'<b>Of Budget:</b> {Finances.get\_budget\_month\_limit(call.from\_user.id) - result\_[2]}',  
 parse\_mode='HTML'  
 )  
 await asyncio.sleep(10)  
 Statistic.delete\_stats\_image(file\_name\_)  
  
  
@dp.callback\_query\_handler(text='BACK\_TO\_OTHER\_FINANCE')  
async def back(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(  
 call.from\_user.id, "\*Choose action to perform\*", parse\_mode="HTML",  
 reply\_markup=markup.inline\_keyboard\_other\_menu  
 )  
  
lib = types.InlineKeyboardMarkup(row\_width=1)  
add = types.InlineKeyboardButton(text="Add book to list", callback\_data="add")  
stats = types.InlineKeyboardButton(text="View your top", callback\_data="stats")  
full = types.InlineKeyboardButton(text="View full list", callback\_data="full")  
lib.add(add, stats, full, markup.inline\_button\_back)  
class FSMBook(StatesGroup):  
 write\_search = State()  
 write\_delete = State()  
  
  
@dp.message\_handler(content\_types=['text', 'document', 'audio', 'photo', 'sticker', 'video', 'voice', 'unknown'],  
 state=FSMBook.write\_search)  
async def search(message: types.Message, state: FSMContext):  
 if message.text:  
 info = bd.get\_info(message.text)  
 await state.finish()  
 if info:  
 if bd.add\_book(info, message.from\_user.id):  
 await bot.send\_message(message.chat.id, "Книга-" + info[0]['Название'] + '\nЗа авторством-' + info[0][  
 'Автор'] + '\nОт издания-' + info[0]['Издательство'] + '\nБыла добавлена в ваш список')  
 else:  
 await bot.send\_message(message.chat.id, "Это произведение уже есть у вас ,или поиск выдал не книгу")  
 else:  
 await bot.send\_message(message.chat.id, "Я не смог найти книгу по даному запросу")  
 else:  
 await bot.send\_message(message.chat.id, "Введите пожалуйста текст")  
 await message.delete()  
  
  
@dp.message\_handler(content\_types=['text', 'document', 'audio', 'photo', 'sticker', 'video', 'voice', 'unknown'],  
 state=FSMBook.write\_delete)  
async def deletess(message: types.Message, state: FSMContext):  
 if message.text and not any((numb not in '1234567890') for numb in message.text):  
 await state.finish()  
 if bd.delete\_book(int(message.text), message.from\_user.id):  
 await bot.send\_message(message.chat.id, "Книга удалена")  
 else:  
 await bot.send\_message(message.chat.id, "Я не смог найти книгу по даному запросу")  
 else:  
 await bot.send\_message(message.chat.id, "Введите пожалуйста код")  
 await message.delete()  
  
  
@dp.callback\_query\_handler(text="📚Library📚")  
async def library(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.message.chat.id, "Выбирайте,что именно вы хотите сделать с книжным списком желаний:",  
 reply\_markup=lib)  
  
  
@dp.callback\_query\_handler(text="add")  
async def add\_(call: types.CallbackQuery):  
 await call.message.delete()  
 await bot.send\_message(call.message.chat.id, "Напишите запрос,по которому я буду искать книгу:")  
 await FSMBook.write\_search.set()  
  
  
@dp.callback\_query\_handler(text="full")  
async def full\_(call: types.CallbackQuery):  
 await call.message.delete()  
 delete = types.InlineKeyboardButton(text="Удалить книгу со списка", callback\_data="delete")  
 await bot.send\_message(call.message.chat.id, "Список ваших интересов:\n" + bd.get\_list(  
 call.from\_user.id) + '\nВведи код книги ,если хочешь убрать её',  
 reply\_markup=InlineKeyboardMarkup(row\_width=1).add(delete))  
  
  
@dp.callback\_query\_handler(text="delete")  
async def delete\_(call: types.CallbackQuery):  
  
 await bot.send\_message(call.message.chat.id, "Введи код книги для удаления:")  
 await FSMBook.write\_delete.set()  
  
  
@dp.callback\_query\_handler(text="stats")  
async def stats\_(call: types.CallbackQuery):  
 await call.message.delete()  
 data = bd.get\_max\_info(call.from\_user.id)  
 await bot.send\_message(call.message.chat.id, "Информация:\n" + '<><><><><><><><>\n'.join(  
 '\n'.join(str(item) for item in group) for group in data.items()))  
  
  
executor.start\_polling(dp, skip\_updates=True)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 executor.start\_polling(dp, skip\_updates=True)

**Модуль *bd.py***

import requests, pymysql  
from bs4 import BeautifulSoup  
from config import connection  
  
  
def get\_info(book\_info):  
 url = []  
 data = {}  
 r = requests.get('https://www.yakaboo.ua/search/?multi=0&cat=&q=' + book\_info)  
 html\_main = BeautifulSoup(r.text, features="html.parser")  
 quote\_main = html\_main.find('ul',  
 class\_="products-grid thumbnails thumbnails\_horizontal thumbnails\_middle data-list")  
 if quote\_main:  
 r = requests.get(quote\_main.find\_all('li')[0].find('a').get('href'))  
 html\_main = BeautifulSoup(r.text, features="html.parser")  
 quote\_book = html\_main.find('table', id="product-attribute-specs-table")  
 if not quote\_book:  
 return  
 keys = ['Автор', 'Издательство', 'Язык', 'Год издания', 'Количество страниц']  
 key = ''  
 data['Название'] = html\_main.find('div', class\_="big-description block translate").find('span').text  
 data['Код'] = int(html\_main.find('div', class\_="product-sku").find('span', itemprop="sku").text)  
 ind = False  
 for q in quote\_book.find\_all('td'):  
 if (ind):  
 if q.find('a'):  
 url.append(q.find('a').get('href'))  
 data[key] = q.text.removeprefix(' ')  
 ind = False  
 if (q.text in keys):  
 key = q.text  
 ind = True  
 for check in keys:  
 if check not in data:  
 data[check] = 'Нет информации'  
 return [data, url]  
  
  
def add\_book(information, telegram\_id):  
 data = information[0]  
 url = information[1]  
 if data['Издательство'] == 'Нет информации':  
 return False  
 with connection.cursor() as cursor:  
 if len(url) == 1:  
 url.append(url[0])  
 else:  
 html\_author = BeautifulSoup(requests.get(url[0]).text, features="html.parser")  
 psevdo = html\_author.find('div', class\_="page-title category-title").find('h1').text.removesuffix(  
 ' — книги и биография').removeprefix(' ')  
 create\_query = f"""SELECT psevdonim FROM Author WHERE psevdonim='{psevdo}';"""  
 cursor.execute(create\_query)  
 if not cursor.fetchall():  
 create\_query = f"""INSERT INTO Author (psevdonim) VALUES('{psevdo}');"""  
 cursor.execute(create\_query)  
 quote\_author = html\_author.find('div', class\_="product-shop span6")  
 if quote\_author:  
 if len(quote\_author.find\_all('td')) > 2:  
 create\_query = f"""UPDATE Author SET date\_of\_birth='{quote\_author.find\_all('td')[3].text}'   
 WHERE psevdonim='{psevdo}';"""  
 cursor.execute(create\_query)  
 create\_query = f"""UPDATE Author SET author\_description='{quote\_author.find('p').text}'   
 WHERE psevdonim='{psevdo}';"""  
 cursor.execute(create\_query)  
 html\_publisher = BeautifulSoup(requests.get(url[1]).text, features="html.parser")  
 publisher\_name = html\_publisher.find('div', class\_="page-title category-title").find('h1').text.removeprefix(  
 ' Издательство книг ')  
 create\_query = f"""SELECT publisher\_name FROM Publisher WHERE publisher\_name='{publisher\_name}';"""  
 cursor.execute(create\_query)  
 if not cursor.fetchall():  
 create\_query = f"""INSERT INTO Publisher (publisher\_name) VALUES('{publisher\_name}');"""  
 cursor.execute(create\_query)  
 quote\_publisher = html\_publisher.find('div', class\_="product-shop span9")  
 if quote\_publisher:  
 create\_query = f"""UPDATE Publisher SET publisher\_description='{quote\_publisher.find('p').text}'   
 WHERE publisher\_name='{publisher\_name}';"""  
 cursor.execute(create\_query)  
 connection.commit()  
 create\_query = f"""SELECT book\_id FROM Books WHERE book\_id={data['Код']};"""  
 cursor.execute(create\_query)  
 if not cursor.fetchall():  
 create\_query = \  
 f"""INSERT INTO Books (book\_id,title,author,publisher,book\_language,year\_of\_publishing,amount\_of\_pages)   
 VALUES({data['Код']},'{data['Название']}','{data['Автор']}','{data['Издательство']}','{data['Язык']}',  
{data['Год издания']},'{data['Количество страниц']}');"""  
 cursor.execute(create\_query)  
 connection.commit()  
 create\_query = f"""SELECT user\_id FROM LibraryUser WHERE telegram\_id={telegram\_id} AND book={data['Код']};"""  
 cursor.execute(create\_query)  
 if cursor.fetchall():  
 return False  
 create\_query = f"""INSERT INTO LibraryUser (book,telegram\_id) VALUES({data['Код']},{telegram\_id});"""  
 cursor.execute(create\_query)  
 connection.commit()  
 return True  
  
  
def delete\_book(book\_id, telegram\_id):  
 with connection.cursor() as cursor:  
 create\_query = f"""SELECT book FROM LibraryUser WHERE book={book\_id} AND telegram\_id={telegram\_id};"""  
 cursor.execute(create\_query)  
 if not cursor.fetchall():  
 return False  
 create\_query = f"""DELETE FROM LibraryUser WHERE book={book\_id} AND telegram\_id={telegram\_id};"""  
 cursor.execute(create\_query)  
 connection.commit()  
 return True  
  
  
def get\_max\_info(telegram\_id):  
 data = {}  
 keys = [['Псевдоним', 'Дата рождения', 'Описание', 'Количество'], ['Название', 'Описание', 'Количество']]  
 with connection.cursor() as cursor:  
 create\_query = f"""SELECT psevdonim,date\_of\_birth,author\_description,COUNT(telegram\_id) AS amount   
 FROM LibraryUser LEFT OUTER JOIN (SELECT book\_id,psevdonim,date\_of\_birth,author\_description   
 FROM Books LEFT OUTER JOIN Author ON psevdonim=author) AS Psevdo ON book\_id=book   
 WHERE telegram\_id={telegram\_id} GROUP BY psevdonim ORDER BY amount DESC LIMIT 3;"""  
 cursor.execute(create\_query)  
 rows = cursor.fetchall()  
 if not rows:  
 data['Топ 3 ваших автора'] = 'Твой список пуст'  
 else:  
 string = ''  
 for row in rows:  
 string += '--' \* 29 + '\nАвтор:\n'  
 author\_info = '\n'.join(str(item) for item in row).split('\n')  
 for i in range(len(author\_info)):  
 string += keys[0][i] + ': ' + (  
 '-' if author\_info[i] == 'None' or author\_info[i] == 'Нет информации' else author\_info[i]) + '\n'  
 data['Топ 3 ваших автора'] = string  
  
 create\_query = f"""SELECT publisher\_name,publisher\_description,COUNT(telegram\_id) AS amount   
 FROM LibraryUser LEFT OUTER JOIN (SELECT book\_id,publisher\_name,publisher\_description   
 FROM Books LEFT OUTER JOIN Publisher ON publisher\_name=publisher) AS Publ ON book\_id=book   
 WHERE telegram\_id={telegram\_id} GROUP BY publisher\_name ORDER BY amount DESC LIMIT 3;"""  
 cursor.execute(create\_query)  
 rows = cursor.fetchall()  
 if not rows:  
 data['Топ 3 ваших издательcтва'] = 'Твой список пуст'  
 else:  
 string = ''  
 for row in rows:  
 string += '--' \* 29 + '\nИздательство:\n'  
 publisher\_info = '\n'.join(str(item) for item in row).split('\n')  
 for i in range(len(publisher\_info)):  
 string += keys[1][i] + ': ' + (  
 '--' if publisher\_info[i] == 'None' or publisher\_info[i] == 'Нет информации' else  
 publisher\_info[i]) + '\n'  
 data['Топ 3 ваших издательтва'] = string  
 return data  
  
  
def get\_list(telegram\_id):  
 string = ''  
 keys = ['Код', 'Название', 'Автор', 'Издательство', 'Язык', 'Год издания', 'Количество страниц']  
 with connection.cursor() as cursor:  
 create\_query = f"""SELECT book FROM LibraryUser WHERE telegram\_id={telegram\_id};"""  
 cursor.execute(create\_query)  
 if not cursor.fetchall():  
 # connection.close()  
 return 'Твой список пуст'  
 create\_query = f"""SELECT book\_id,title,author,publisher,book\_language,year\_of\_publishing,amount\_of\_pages   
 FROM LibraryUser LEFT OUTER JOIN Books ON book\_id=book WHERE telegram\_id={telegram\_id};"""  
 cursor.execute(create\_query)  
 rows = cursor.fetchall()  
 for row in rows:  
 string += '--' \* 29 + '\nКнига:\n'  
 book\_info = '\n'.join(str(item) for item in row).split('\n')  
 for i in range(len(keys)):  
 string += keys[i] + ': ' + book\_info[i] + '\n'  
 return string

**Модуль *Finances.py***

from datetime import datetime  
from typing import NamedTuple, List, Optional, re  
from config import connection  
  
from imports import \*  
from exceptions import \*  
import databases as db  
  
  
class Categories:  
 def \_\_init\_\_(self):  
 self.\_categories = self.\_load\_categories()  
  
 def \_load\_categories(self):  
 return self.\_fill\_aliases(db.fetchall\_("category", "code\_name category\_name aliases\_".split()))  
  
 def \_fill\_aliases(self, categories):  
 categories\_result = []  
 for index, category in enumerate(categories):  
 aliases = category["aliases\_"].split(",")  
 aliases = list(filter(None, map(str.strip, aliases)))  
 aliases.append(category["code\_name"])  
 aliases.append(category["category\_name"])  
 categories\_result.append(  
 Category(codename=category['code\_name'], name=category['category\_name'], aliases=aliases)  
 )  
 return categories\_result  
  
 @property  
 def get\_all\_categories(self):  
 return self.\_categories  
  
 def get\_category(self, category\_name):  
 category\_finded = None  
 others = None  
 for category in self.\_categories:  
 if category.codename == "other\_":  
 others = category  
 for alias in category.aliases:  
 if category\_name in alias:  
 category\_finded = category  
 if not category\_finded:  
 category\_finded = others  
 return category\_finded  
  
  
class CreateCategory(NamedTuple):  
 code\_name\_: str  
 category\_name\_: str  
 aliases\_text\_: str  
  
  
class CategoryMessage(NamedTuple):  
 name\_: str  
 category\_text: str  
  
  
class Category(NamedTuple):  
 codename: str  
 name: str  
 aliases: List[str]  
  
  
class Message(NamedTuple):  
 amount: int  
 category\_text: str  
  
  
class BudgetMessage(NamedTuple):  
 daily\_amount: int  
 month\_amount: int  
  
  
class IncomeExpense(NamedTuple):  
 id: Optional[int]  
 amount: int  
 category\_name: str  
  
  
class UserData(NamedTuple):  
 id: str  
 first\_name: Optional[str]  
 last\_name: Optional[str]  
 username: Optional[str]  
  
  
def \_parse\_user\_data(message):  
 return UserData(  
 id=str(message['id']), first\_name=message['first\_name'], last\_name=message['last\_name'],  
 username=message['username']  
 )  
  
  
def \_parse\_category(message):  
 regexp\_ = re.match(r'^([a-z]+: )([a-z]+, ){1,7}([a-z]+)', str(message))  
 if not regexp\_ or not regexp\_.group(0):  
 raise AddCategoryError(str(message))  
 reg\_str = str(regexp\_.group(0)).split(': ')  
 return CategoryMessage(name\_=reg\_str[0], category\_text=reg\_str[1])  
  
  
def \_parse\_message(message):  
 regexp = re.match(r"([\d ]+) (.\*)", message)  
 if not regexp or not regexp.group(0) \  
 or not regexp.group(1) or not regexp.group(2):  
 raise AddCategoryError(str(message))  
 return Message(amount=int(regexp.group(1).replace(" ", "")), category\_text=regexp.group(2).strip().lower())  
  
  
def \_parse\_budget\_message(message):  
 regexp\_ = re.match(r'(daily) ([1-9]([0-9]){0,10}) (month) ([1-9]([0-9]){0,10})',  
 str(message).lower().replace(" ", " "))  
 # if not (regexp\_.group(1) and regexp\_.group(2)) or (regexp\_.group(4) and regexp\_.group(5)):  
 if not regexp\_.group(0):  
 raise ChangeBudgetError(str(message))  
 return BudgetMessage(daily\_amount=int(regexp\_.group(2)), month\_amount=int(regexp\_.group(5)))  
  
  
def check\_user\_exists(user\_id):  
 with connection.cursor() as cursor:  
 cursor.execute(f"SELECT id FROM users WHERE id = {user\_id}")  
 return bool(cursor.rowcount)  
  
  
def add\_user(message):  
 user\_info = \_parse\_user\_data(message)  
 db.insert(  
 "users",  
 {  
 "id": user\_info.id,  
 "first\_name": user\_info.first\_name,  
 "last\_name": user\_info.last\_name,  
 "username": user\_info.username  
 }  
 )  
  
  
def add\_expense(message, user\_id):  
 parsed\_message = \_parse\_message(message)  
 category = Categories().get\_category(  
 parsed\_message.category\_text)  
 db.insert(  
 "expenses",  
 {  
 "amount": parsed\_message.amount,  
 "date\_time": \_get\_now\_formatted(),  
 "category": category.codename,  
 "user\_id": str(user\_id)  
 }  
 )  
 return IncomeExpense(id=None, amount=parsed\_message.amount, category\_name=category.name)  
  
  
def edit\_budget(message, user\_id):  
 parsed\_message = \_parse\_budget\_message(message)  
 db.update\_(  
 "budget", {  
 'daily\_limit': parsed\_message.daily\_amount,  
 'month\_limit': parsed\_message.month\_amount,  
 'user\_id': str(user\_id)  
 },  
 f"code\_name = 'general' AND user\_id = {user\_id}"  
 )  
  
  
def add\_incomes(message, user\_id):  
 parsed\_message = \_parse\_message(message)  
 category = Categories().get\_category(  
 parsed\_message.category\_text)  
 db.insert(  
 "incomes",  
 {  
 "amount": parsed\_message.amount,  
 "date\_time": \_get\_now\_formatted(),  
 "category": category.codename,  
 "user\_id": str(user\_id)  
 }  
 )  
 return IncomeExpense(id=None, amount=parsed\_message.amount, category\_name=category.name)  
  
  
def create\_category\_finance(message, user\_id):  
 parsed\_data = \_parse\_category(message)  
 db.insert(  
 "category", {  
 'code\_name': parsed\_data.name\_ + '\_',  
 'category\_name': parsed\_data.name\_,  
 'aliases\_': parsed\_data.category\_text,  
 'user\_id': str(user\_id)  
 }  
 )  
 return CreateCategory(code\_name\_=parsed\_data[0] + '\_', category\_name\_=parsed\_data[0], aliases\_text\_=parsed\_data[1])  
  
  
def see\_categories(user\_id):  
 with connection.cursor() as cursor:  
 cursor.execute(f'SELECT category\_name, aliases\_ FROM category WHERE user\_id = {str(user\_id)}')  
 rows = cursor.fetchall()  
 return [CategoryMessage(name\_=row[0], category\_text=row[1]) for row in rows]  
  
  
def delete\_expense(row\_id, user\_id):  
 with connection.cursor() as cursor:  
 row\_id = int(row\_id)  
 cursor.execute(f"DELETE FROM expenses WHERE expense\_id = {row\_id} AND user\_id = {user\_id}")  
 connection.commit()  
  
  
def delete\_income(row\_id, user\_id):  
 with connection.cursor() as cursor:  
 row\_id = int(row\_id)  
 cursor.execute(f"DELETE FROM incomes WHERE income\_id = {row\_id} AND user\_id = {user\_id}")  
 connection.commit()  
  
  
def \_get\_now\_formatted():  
 return \_get\_now\_datetime().strftime('%Y-%m-%d %H:%M:%S')  
  
  
def \_get\_now\_datetime():  
 return datetime.now()  
  
  
def set\_default\_budget(user\_id):  
 with connection.cursor() as cursor:  
 cursor.execute(f'INSERT INTO budget (daily\_limit, month\_limit, user\_id) VALUES (0, 0, {str(user\_id)})')  
 connection.commit()  
  
  
def get\_budget\_month\_limit(user\_id):  
 return db.fetchone\_for\_budget('budget', f"month\_limit".split(), f'user\_id = {user\_id}')  
  
  
def get\_budget\_daily\_limit(user\_id):  
 return db.fetchone\_for\_budget('budget', f"daily\_limit".split(), f'user\_id = {user\_id}')  
  
  
def today\_expenses(user\_id):  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f'SELECT a.expense\_id, a.amount, b.category\_name '  
 f'FROM expenses a LEFT JOIN category b ON b.code\_name=a.category AND a.user\_id = b.user\_id '  
 f'WHERE (CAST(date\_time AS DATE) = CAST(CURDATE() AS DATE) AND a.user\_id = {str(user\_id)})'  
 )  
 rows = cursor.fetchall()  
 return [IncomeExpense(id=row[0], amount=row[1], category\_name=row[2]) for row in rows]  
  
  
def this\_week\_expenses(user\_id):  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f'SELECT a.expense\_id, a.amount, b.category\_name '  
 f'FROM expenses a LEFT JOIN category b ON b.code\_name=a.category AND a.user\_id = b.user\_id '  
 f'WHERE (YEARWEEK(CURDATE()) = YEARWEEK(date\_time) AND a.user\_id = {str(user\_id)})'  
 )  
 rows = cursor.fetchall()  
 return [IncomeExpense(id=row[0], amount=row[1], category\_name=row[2]) for row in rows]  
  
  
def this\_month\_expenses(user\_id):  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f'SELECT a.expense\_id, a.amount, b.category\_name '  
 f'FROM expenses a LEFT JOIN category b ON b.code\_name=a.category AND a.user\_id = b.user\_id '  
 f'WHERE (MONTH(date\_time) = MONTH(CURDATE()) AND '  
 f'YEAR(date\_time) = YEAR(CURDATE()) AND a.user\_id = {str(user\_id)})'  
 )  
 rows = cursor.fetchall()  
 return [IncomeExpense(id=row[0], amount=row[1], category\_name=row[2]) for row in rows]  
  
  
def today\_incomes(user\_id):  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f'SELECT a.income\_id, a.amount, b.category\_name '  
 f'FROM incomes a LEFT JOIN category b ON b.code\_name=a.category AND a.user\_id = b.user\_id '  
 f'WHERE (CAST(date\_time AS DATE) = CAST(CURDATE() AS DATE) AND a.user\_id = {str(user\_id)})'  
 )  
 rows = cursor.fetchall()  
 return [IncomeExpense(id=row[0], amount=row[1], category\_name=row[2]) for row in rows]  
  
  
def this\_week\_incomes(user\_id):  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f'SELECT a.income\_id, a.amount, b.category\_name '  
 f'FROM incomes a LEFT JOIN category b ON b.code\_name=a.category AND a.user\_id = b.user\_id '  
 f'WHERE (YEARWEEK(CURDATE()) = YEARWEEK(date\_time) AND a.user\_id = {str(user\_id)})'  
 )  
 rows = cursor.fetchall()  
 return [IncomeExpense(id=row[0], amount=row[1], category\_name=row[2]) for row in rows]  
  
  
def this\_month\_incomes(user\_id):  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f'SELECT a.income\_id, a.amount, b.category\_name '  
 f'FROM incomes a LEFT JOIN category b ON b.code\_name=a.category AND a.user\_id = b.user\_id '  
 f'WHERE (MONTH(date\_time)=MONTH(CURDATE()) AND '  
 f'YEAR(date\_time)=YEAR(CURDATE()) AND a.user\_id = {str(user\_id)})'  
 )  
 rows = cursor.fetchall()  
 return [IncomeExpense(id=row[0], amount=row[1], category\_name=row[2]) for row in rows]

**Модуль *Statistic.py***

import os  
  
import numpy as np  
from matplotlib import pyplot as plt  
from matplotlib.dates import DayLocator  
from matplotlib.ticker import AutoMinorLocator  
from random import shuffle  
from config import connection  
  
  
def \_get\_formatted(date):  
 return date.strftime('%Y-%m-%d')  
  
  
def merging\_list(arr1, arr2):  
 return [i for i in arr2 if i not in arr1] + arr1  
  
  
def delete\_stats\_image(name\_):  
 path\_ = os.getcwd().replace(f'\\', '/')  
 os.remove(f"{path\_}/{name\_}.png")  
  
  
def get\_week\_expenses\_for\_stats(user\_id):  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f'SELECT SUM(amount), CAST(date\_time AS DATE) AS Date\_, date\_time FROM expenses '  
 f'WHERE yearweek(date\_time, 1) = yearweek(CURDATE(), 1) AND user\_id = {user\_id} '  
 f'GROUP BY CAST(date\_time AS DATE) '  
 f'ORDER BY date\_time ASC'  
 )  
 rows = cursor.fetchall()  
 return [i[0] for i in rows], [\_get\_formatted(j[1]) for j in rows]  
  
  
def get\_week\_incomes\_for\_stats(user\_id):  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f'SELECT SUM(amount), CAST(date\_time AS DATE) AS Date\_, date\_time FROM incomes '  
 f'WHERE yearweek(date\_time, 1) = yearweek(CURDATE(), 1) AND user\_id = {user\_id} '  
 f'GROUP BY CAST(date\_time AS DATE) '  
 f'ORDER BY date\_time ASC'  
 )  
 rows = cursor.fetchall()  
 return [i[0] for i in rows], [\_get\_formatted(j[1]) for j in rows]  
  
  
def get\_month\_expenses\_for\_stats(user\_id):  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f'SELECT SUM(amount), CAST(date\_time AS DATE) AS Date\_, date\_time FROM expenses '  
 f'WHERE MONTH(date\_time) = MONTH(CURDATE()) '  
 f'AND YEAR(date\_time) = YEAR(CURDATE()) AND user\_id = {user\_id} '  
 f'GROUP BY CAST(date\_time AS DATE) '  
 f'ORDER BY date\_time ASC'  
 )  
 rows = cursor.fetchall()  
 return [i[0] for i in rows], [\_get\_formatted(j[1]) for j in rows]  
  
  
def get\_month\_incomes\_for\_stats(user\_id):  
 print('get\_month\_incomes\_for\_stats')  
 with connection.cursor() as cursor:  
 cursor.execute(  
 f'SELECT SUM(amount), CAST(date\_time AS DATE) AS Date\_, date\_time FROM incomes '  
 f'WHERE MONTH(date\_time) = MONTH(CURDATE()) '  
 f'AND YEAR(date\_time) = YEAR(CURDATE()) AND user\_id = {user\_id} '  
 f'GROUP BY CAST(date\_time AS DATE) '  
 f'ORDER BY date\_time ASC'  
 )  
 rows = cursor.fetchall()  
 print(rows)  
 # rows.sort(key=lambda x: x.count, reverse=True)  
 return [i[0] for i in rows], [\_get\_formatted(j[1]) for j in rows]  
  
  
def week\_data\_for\_stats(user\_id):  
 expenses\_ = get\_week\_expenses\_for\_stats(user\_id)  
 incomes\_ = get\_week\_incomes\_for\_stats(user\_id)  
 return expenses\_[0], expenses\_[1], incomes\_[0], incomes\_[1]  
  
  
def month\_data\_for\_stats(user\_id):  
 expenses\_ = get\_month\_expenses\_for\_stats(user\_id)  
 incomes\_ = get\_month\_incomes\_for\_stats(user\_id)  
 return expenses\_[0], expenses\_[1], incomes\_[0], incomes\_[1]  
  
  
def stats\_for\_current\_week(user\_id):  
 current\_week\_data\_ = week\_data\_for\_stats(user\_id)  
 return create\_diagram\_for\_stats(current\_week\_data\_[0], current\_week\_data\_[2],  
 current\_week\_data\_[1], current\_week\_data\_[3], 'Week Statistic')  
  
  
def stats\_for\_current\_month(user\_id):  
 current\_month\_data\_ = month\_data\_for\_stats(user\_id)  
 return create\_diagram\_for\_stats(current\_month\_data\_[0], current\_month\_data\_[2],  
 current\_month\_data\_[1], current\_month\_data\_[3], 'Month Statistic')  
  
  
def create\_diagram\_for\_stats(values\_expenses, values\_incomes, dates\_expenses, dates\_incomes, type\_):  
 print('create\_diagram\_for\_stats')  
 x = np.arange(len(dates\_expenses))  
 y = np.array(values\_expenses)  
 y2 = np.array(values\_incomes)  
 x2 = np.arange(len(dates\_incomes))  
 fig, ax = plt.subplots(figsize=(10, 6))  
 plt.title(type\_)  
 plt.xlabel("Date")  
 plt.ylabel("Amount")  
 ax.xaxis.set\_major\_locator(DayLocator())  
 plt.plot(x, y, 'o-', label='expenses')  
 plt.plot(x2, y2, 'o-', label='incomes')  
 plt.legend(loc="upper left")  
 print(dates\_expenses)  
 print(merging\_list(dates\_expenses, dates\_incomes))  
 ax.set\_xticklabels(merging\_list(dates\_expenses, dates\_incomes), fontsize=10)  
 ax.grid(color='b', alpha=0.5, linestyle='dashed', linewidth=0.5)  
 ax.xaxis.set\_minor\_locator(AutoMinorLocator())  
 fig.autofmt\_xdate()  
 alphabet\_ = list('abcdefghhijklmnopqrsruwxyz')  
 shuffle(alphabet\_)  
 name\_ = ''.join(i for i in alphabet\_[:10])  
 plt.savefig(name\_)  
 return name\_  
  
  
def calculating\_results(values\_expenses, values\_incomes):  
 total\_expenses = sum(values\_expenses)  
 total\_incomes = sum(values\_incomes)  
 pure\_profit = total\_incomes - total\_expenses  
 return total\_expenses, total\_incomes, pure\_profit  
  
  
def resulting\_for\_the\_current\_week(user\_id):  
 current\_week\_data\_ = week\_data\_for\_stats(user\_id)  
 return calculating\_results(current\_week\_data\_[0], current\_week\_data\_[2])  
  
  
def resulting\_for\_the\_current\_month(user\_id):  
 current\_month\_data\_ = month\_data\_for\_stats(user\_id)  
 return calculating\_results(current\_month\_data\_[0], current\_month\_data\_[2])

**Модуль *exceptions.py***

class AddCategoryError(Exception):  
 def \_\_init\_\_(self, category, error\_message='Incorrect category input'):  
 self.\_category = category  
 self.\_error\_message = error\_message  
 super().\_\_init\_\_(self.\_error\_message)  
  
 def \_\_str\_\_(self):  
 return f'{self.\_error\_message}:\n{self.\_category} does not much the format:\n' \  
 f'`products: products, food, eat`'  
  
  
class AddExpenseError(Exception):  
 def \_\_init\_\_(self, expense, error\_message='Incorrect expense input'):  
 self.\_expense = expense  
 self.\_error\_message = error\_message  
 super().\_\_init\_\_(self.\_error\_message)  
  
 def \_\_str\_\_(self):  
 return f'{self.\_error\_message}:\n{self.\_expense} does not much the format: `125 taxi`'  
  
  
class AddIncomeError(Exception):  
 def \_\_init\_\_(self, income, error\_message='Incorrect income input'):  
 self.\_income = income  
 self.\_error\_message = error\_message  
 super().\_\_init\_\_(self.\_error\_message)  
  
 def \_\_str\_\_(self):  
 return f'{self.\_error\_message}:\n{self.\_income} does not much the format: `125 job`'  
  
  
class ChangeBudgetError(Exception):  
 def \_\_init\_\_(self, budget, error\_message='Incorrect budget input'):  
 self.\_budget = budget  
 self.\_error\_message = error\_message  
 super().\_\_init\_\_(self.\_error\_message)  
  
 def \_\_str\_\_(self):  
 return f'{self.\_error\_message}:\n{self.\_budget} does not much the format:\n' \  
 f'`daily 150 month 8500`\n' \  
 f'daily 360\n' \  
 f'month 4520'  
  
  
class DeleteError(Exception):  
 def \_\_init\_\_(self, delete, error\_message='Deleting error'):  
 self.\_delete = delete  
 self.\_error\_message = error\_message  
 super().\_\_init\_\_(self.\_error\_message)  
  
 def \_\_str\_\_(self):  
 return f'{self.\_error\_message}:\n{self.\_delete}\n'

**Модуль *databases.py***

from config import connection  
  
  
def insert(table, column\_values):  
 columns = ', '.join(column\_values.keys())  
 values = [tuple(i for i in column\_values.values())]  
 placeholders = '%s' + ', %s' \* (len(column\_values) - 1)  
 with connection.cursor() as cursor:  
 cursor.executemany(f'INSERT INTO {table} ({columns}) VALUES ({placeholders})', values)  
 connection.commit()  
  
  
def fetchall\_(table, columns):  
 columns\_joined = ", ".join(columns)  
 with connection.cursor() as cursor:  
 cursor.execute(f"SELECT {columns\_joined} FROM {table}")  
 rows = cursor.fetchall()  
 result\_ = []  
 for row in rows:  
 dict\_row = {}  
 for index, column in enumerate(columns):  
 dict\_row[column] = row[index]  
 result\_.append(dict\_row)  
 return result\_  
  
  
def fetchone\_for\_budget(table, columns, condition):  
 columns\_joined = ", ".join(columns)  
 with connection.cursor() as cursor:  
 cursor.execute(f"SELECT {columns\_joined} FROM {table} WHERE {condition}")  
 z = cursor.fetchone()  
 return z[0]  
  
  
def delete(table, row, value):  
 with connection.cursor() as cursor:  
 cursor.execute(f"DELETE FROM {table} WHERE {row}='{value}'")  
 connection.commit()  
  
  
def update\_(table, data, condition):  
 columns = ', '.join([f'{i} = %s' for i in data])  
 values = [[i] for i in data.values()]  
 with connection.cursor() as cursor:  
 cursor.execute(f'UPDATE {table} SET {columns} WHERE {condition}', values)  
 connection.commit()