**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**

**(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кафедра |  | О7 |  | Программная инженерия |
|  |  | шифр |  | наименование кафедры, по которой выполняется работа |
| Дисциплина |  | Базы данных | | |
|  |  | наименование дисциплины | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ПРАКТИЧЕСКАЯ РАБОТА | | | | |
|  | 5 |  |
|  | номер задания (при наличии) |  |
| СОЗДАНИЕ ИНТЕРФЕЙСА ДЛЯ РАБОТЫ С БАЗОЙ | | | |
| ДАННЫХ ПРЕДМЕТНОЙ ОБЛАСТИ КРИПТОБИРЖИ | | | |

при наличии указать тему учебно-практической работы и (или) номер варианта

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ОБУЧАЮЩИЙСЯ** | | | | | |
| группы | | | |  | О726Б |
|  |  | Махов Н.М. | | | |
| подпись |  | фамилия и инициалы | | | |
|  | | | | |
| дата сдачи | | | | |
| **ПРОВЕРИЛ** | | | | | |
| Преподаватель | | | | | | |
| ученая степень, ученое звание, должность | | | | | | |
|  |  | Шибаев А.А. | | | | |
| подпись |  | фамилия и инициалы | | | | |
| Оценка / балльная оценка | | |  | | |
|  | | | | |
| дата проверки | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Санкт-Петербург | | | | | | |
|  |  |  | 20 | 25 | г. |  |

**Введение**

В данной работе разработан клиент-серверный интерфейс для взаимодействия с базой данных, представленный в виде мобильного приложения под iOS с использованием фреймворка UIKit. Приложение обеспечивает удобное пользовательское взаимодействие с системой, реализуя функции авторизации, регистрации, покупки криптовалюты, открытия стейкинга, а также создания и удаления кошельков.

Связь с базой данных осуществляется через серверную часть, написанную на языке Python с использованием фреймворка FastAPI, обеспечивающего быстрое и эффективное REST API-взаимодействие. Для хранения и управления данными используется СУБД PostgreSQL, доступ к которой организован через pgAdmin — графическую среду для администрирования баз данных.

Таким образом, реализованная система представляет собой полнофункциональное клиент-серверное приложение, в котором мобильный клиент на iOS взаимодействует с сервером для выполнения операций, связанных с криптовалютными активами и пользовательскими данными.

**Результаты работы**

Иконка приложения представлена на рисунке 1.

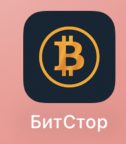


Рисунок 1 – Иконка приложения

Запуск приложения будет осущетсвлятьс на эмуляторе iOS, встроенным в среду разработки XCode. Результат рисунок 2.



Рисунок 2 – Иконка приложения на рабочем столе

При входе пользователю предлагают зарегистрироваться, создадим нового пользователя, результат представлен на рисунк 3.

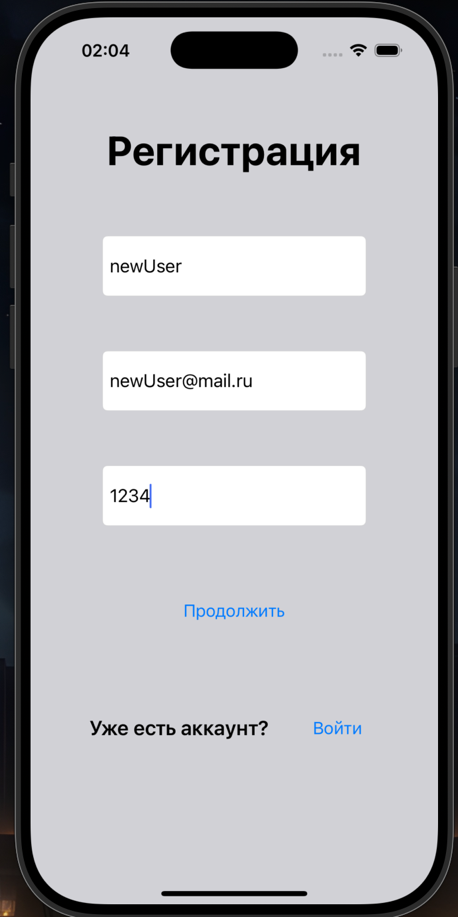


Рисунок 3 – Страница регистрации

Новый пользователь отобразился в базе данных, результат представлен на рисунке 4.

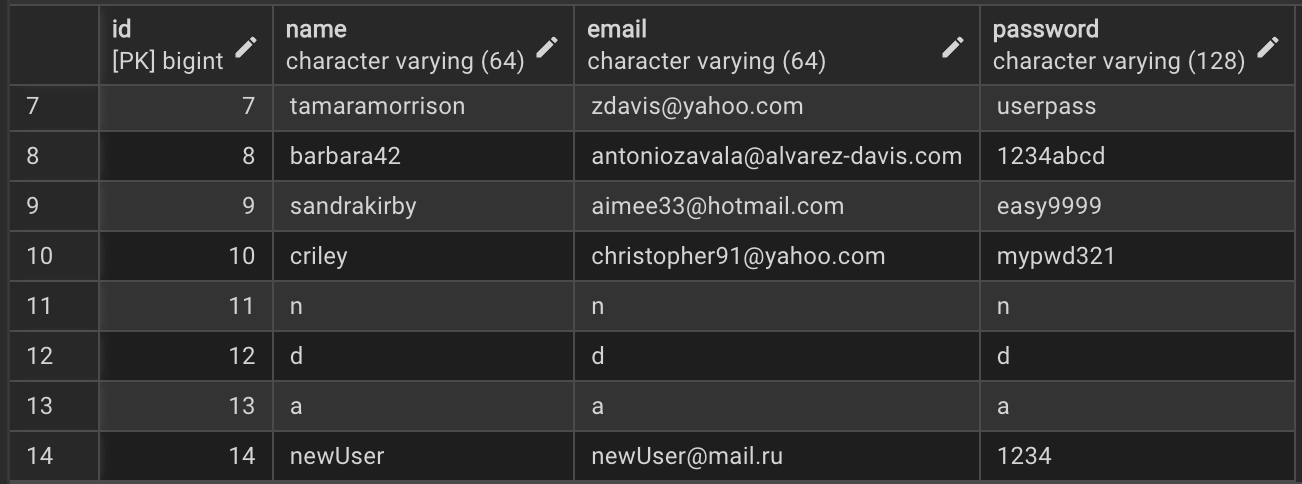


Рисунок 4 –Новый пользовтель в таблицк users

Экран для просмотра информации о пользователе, результат представлен на рисунке 5.

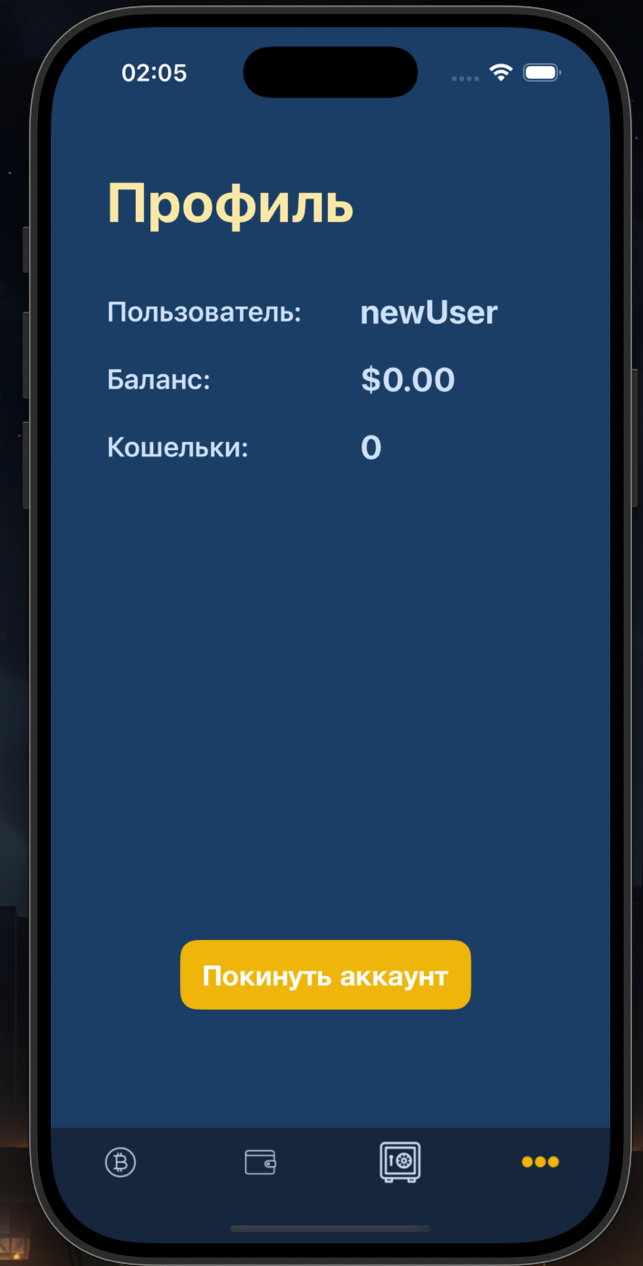


Рисунок 5 – Информация о пользователе

Нажимаем на кнопку покинуть аккаунт, для того чтобы зайти на существующий аккаунт, вводим данные от существующего аккаунта. Результат рисунок 6.

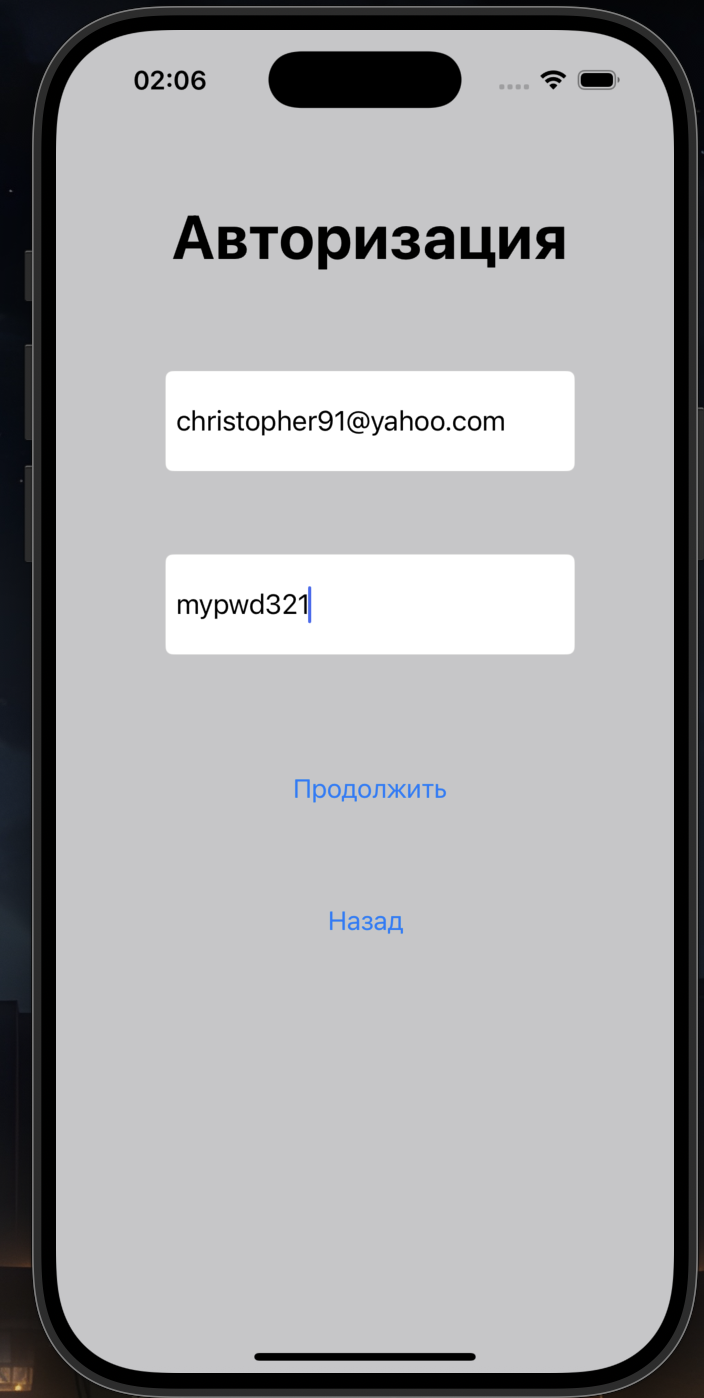


Рисунок 6 – Вход в уже существующий аккаунт

Попадаем на главный экран, где можно приобрести криптовалюту, в нашей базе данных существует 10 криптовалют, их список отображен на экране с ценок. Результат представлен на рисунке 7.

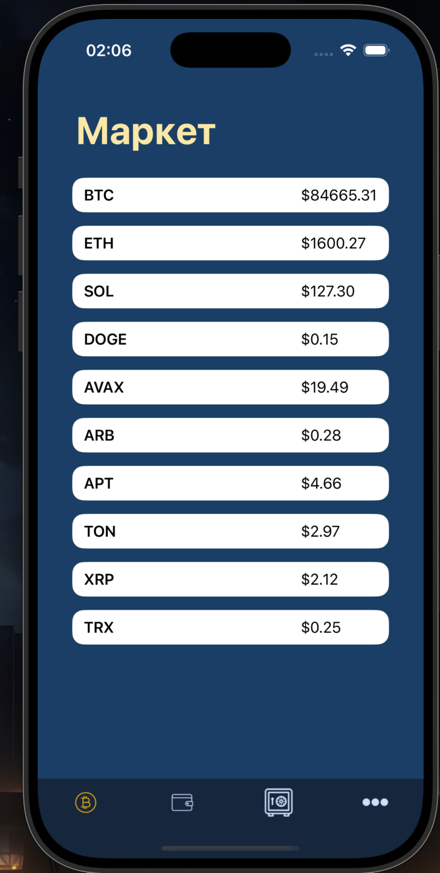


Рисунок 7 – Главная страница.

При нажатии на любую из представленных валют пользователь переходит на экран покупки/продажи. Покупка и продажа реализованы в условной форме: фактическое хранение криптовалют на кошельке не предусмотрено, так как текущая структура базы данных не содержит сведений о типе валюты, привязанной к кошельку.

Условием успешной покупки является наличие достаточного баланса на выбранном кошельке — сумма покупки не должна превышать текущий баланс, а цена определяется как рыночная. При выполнении операции создаётся ордер на покупку или продажу, который немедленно исполняется (используется рыночный тип ордера, лимитные ордера в приложении не реализованы).

После исполнения ордера автоматически формируется транзакция, содержащая всю необходимую информацию об операции, включая id созданного ордера. Результат представлен на рисунке 8.

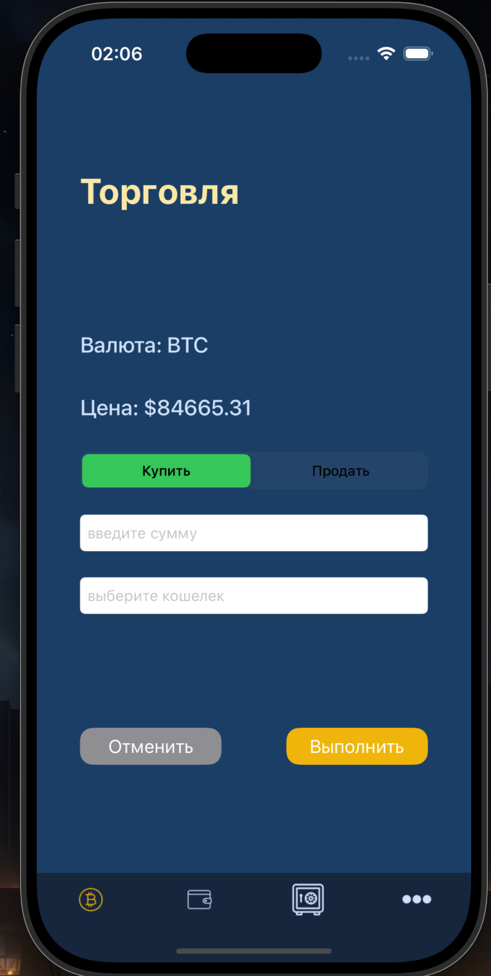


Рисунок 8 – Страница покупки

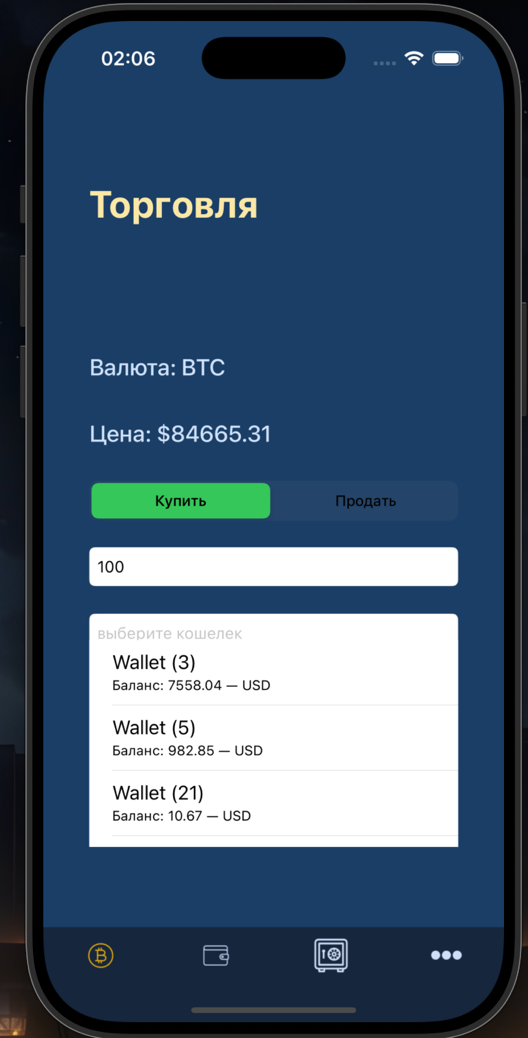


Рисунок 9 – Выбор кошелька для покупки

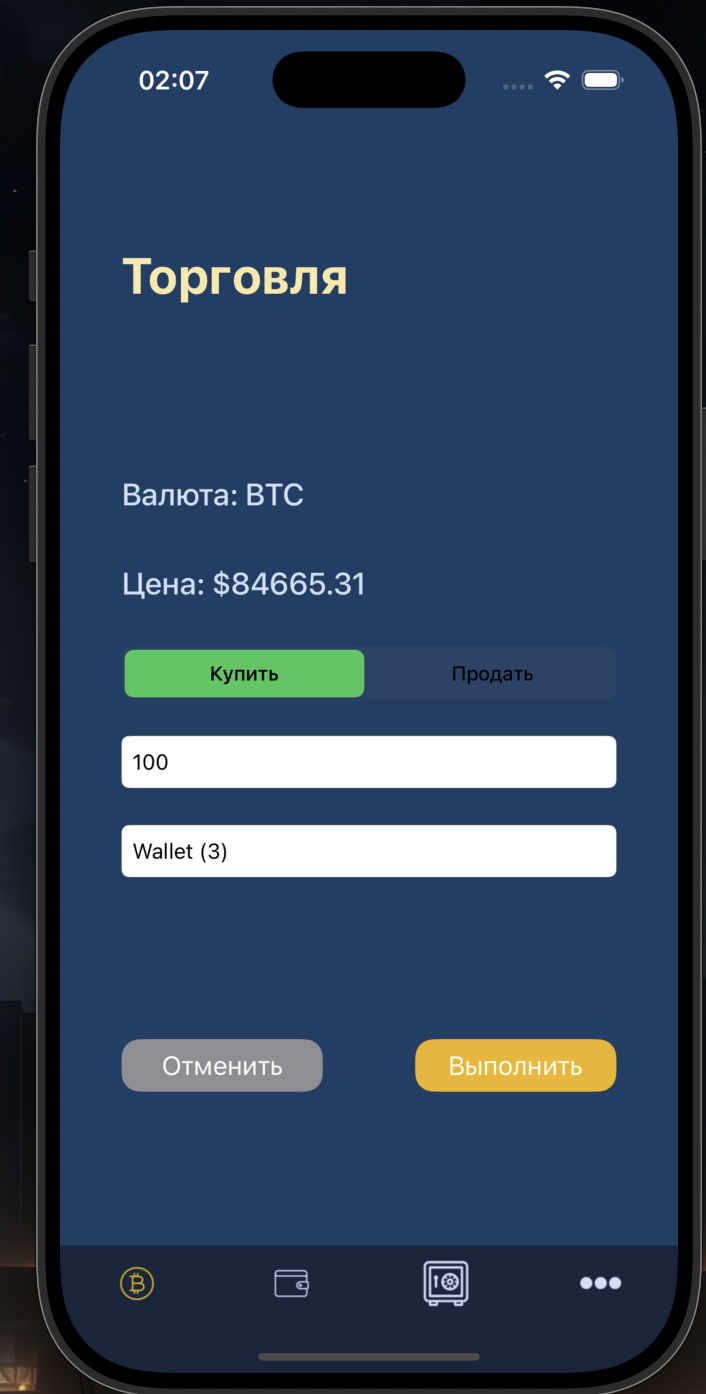


Рисунок 10 – Заполненная страница ордера

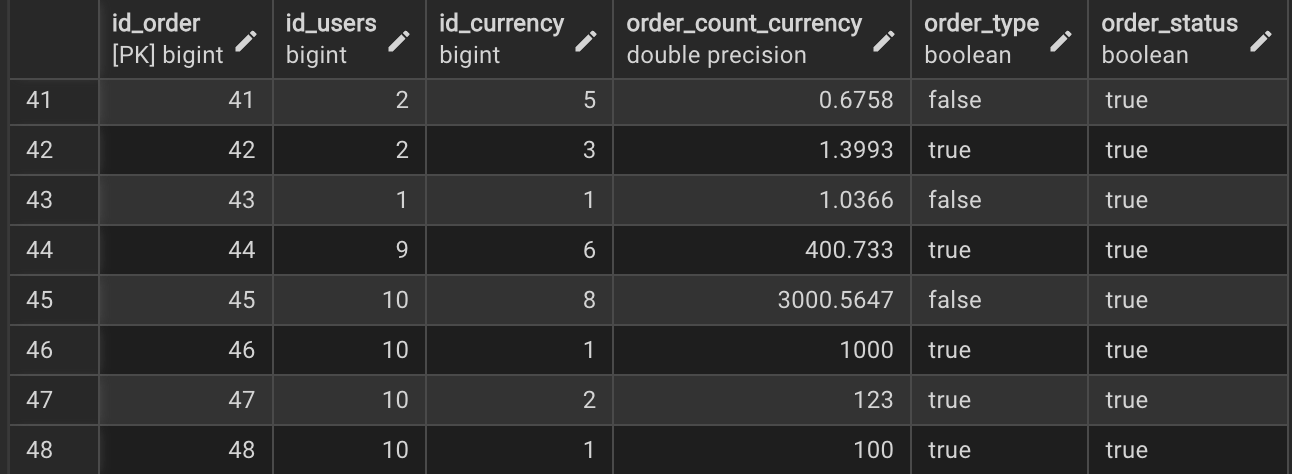


Рисунок 11 – Ордер был выполнен, id кошелька 3

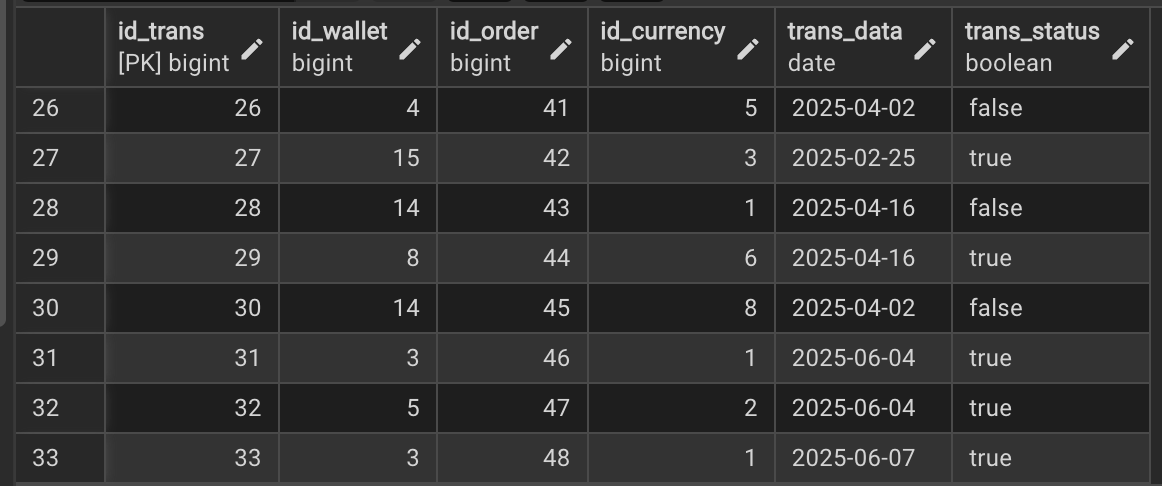


Рисунок 12 – Транзация была выполнена, кошелек 3, id ордера 48

У выбранного пользователя несколько кошельков, интерфейс дает возможность просматривать баланс, а также вылюту, выбранную для хранения. Результат рисунок 13.

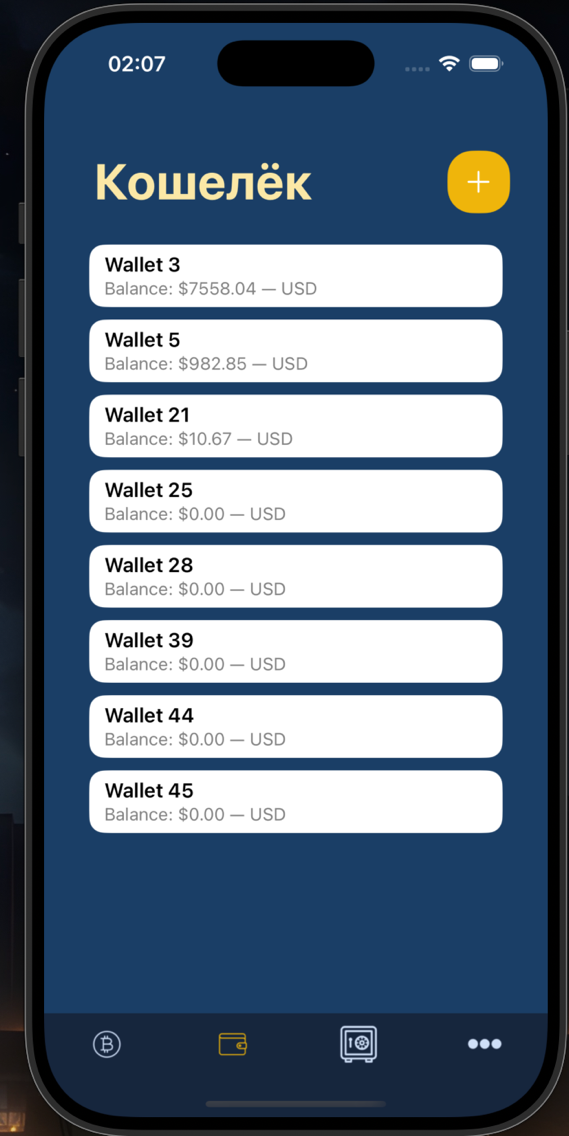


Рисунок 13 – Список кошельков пользователя

По нажатию на кнопку «+» можно создать новый кошелек, все необходимая информация будет добавлена в базу данных. Результат представлен на рисунке 14 и рисунке 15.

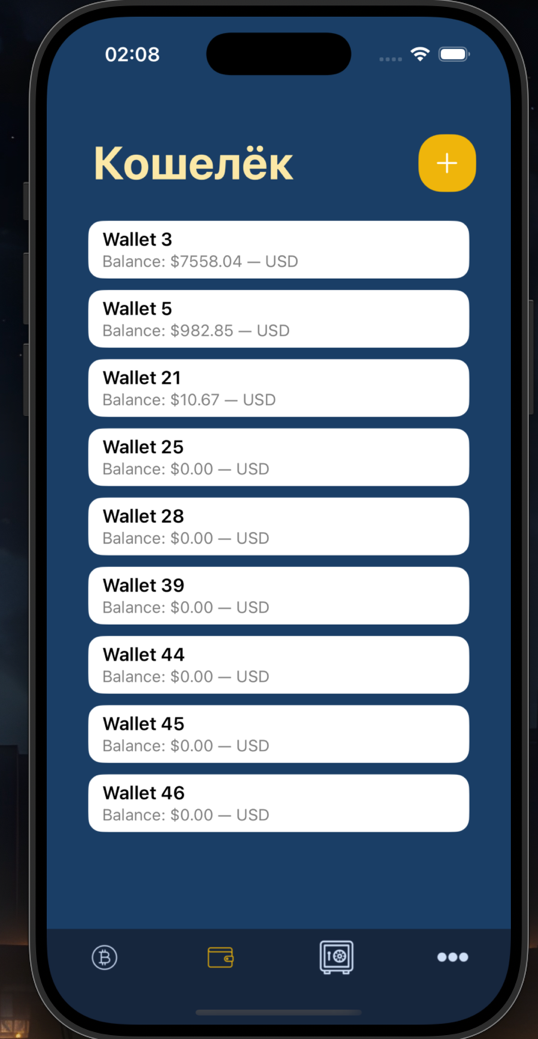


Рисунок 14 – Создание нового кошелька

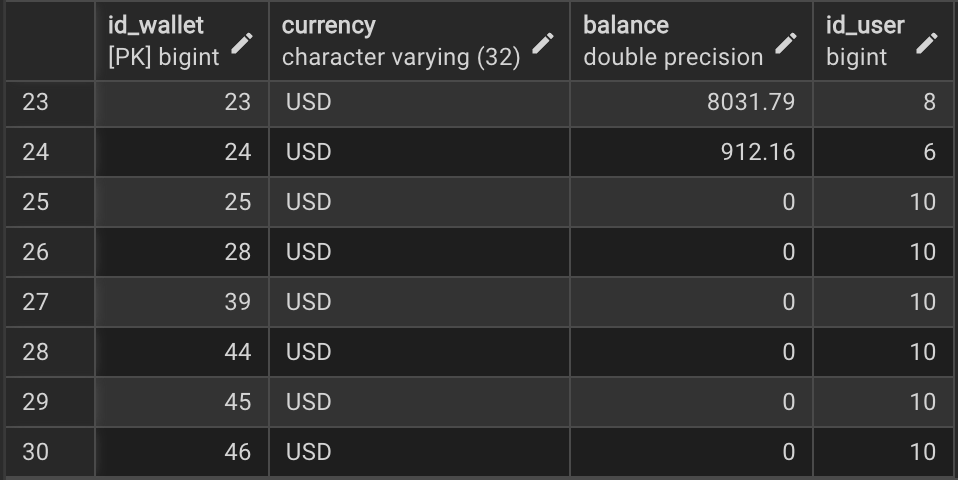


Рисунок 15 – Добавленный кошелек в базе данных

Если баланс кошелька 0, то его можно удалить. Результат представлен на рисунке 16.

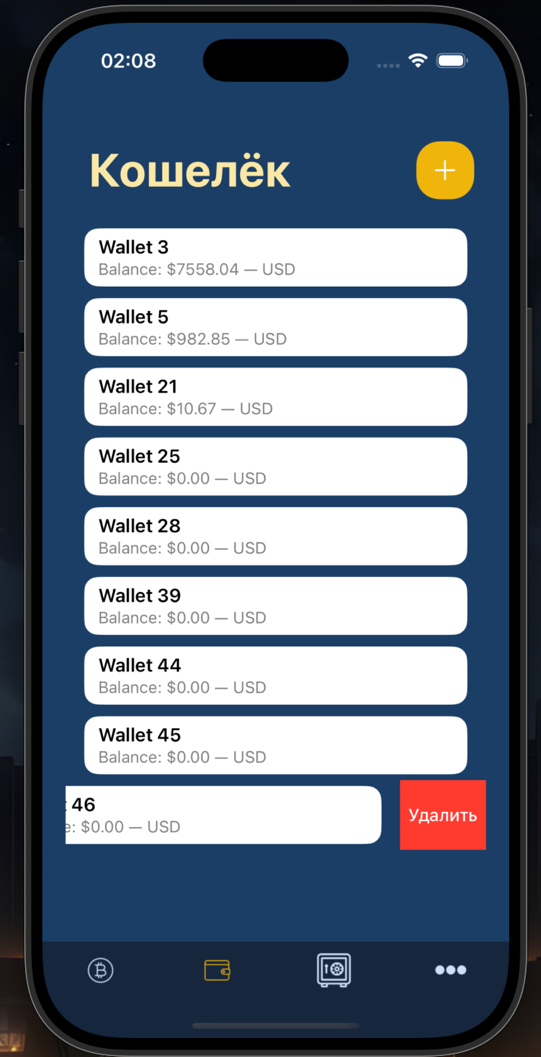


Рисунок 16 – Удаление кошелька

Кошелек 46 был удален из базы данных. Результат представлен на рисунке 17.

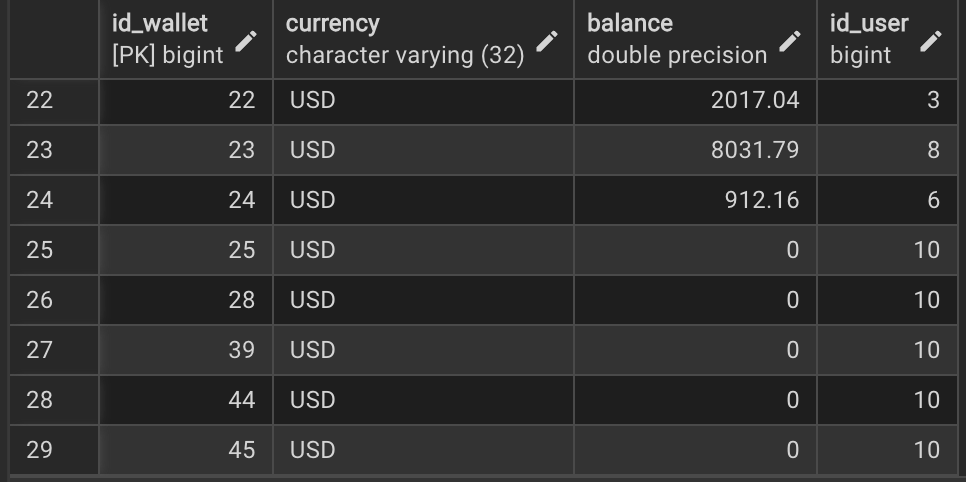


Рисунок 17 – Удаление кошелька из базы данных

Список активных стейкингов пользователя. Результат представлен на рисунке 18.



Рисунок 18 – Экран активных стейкингов пользователя

Начнем новый стейкинг, ставка фиксированная 20%, стейкинг не поддерживает выбор валюты. Результат представлен на рисунке 19.

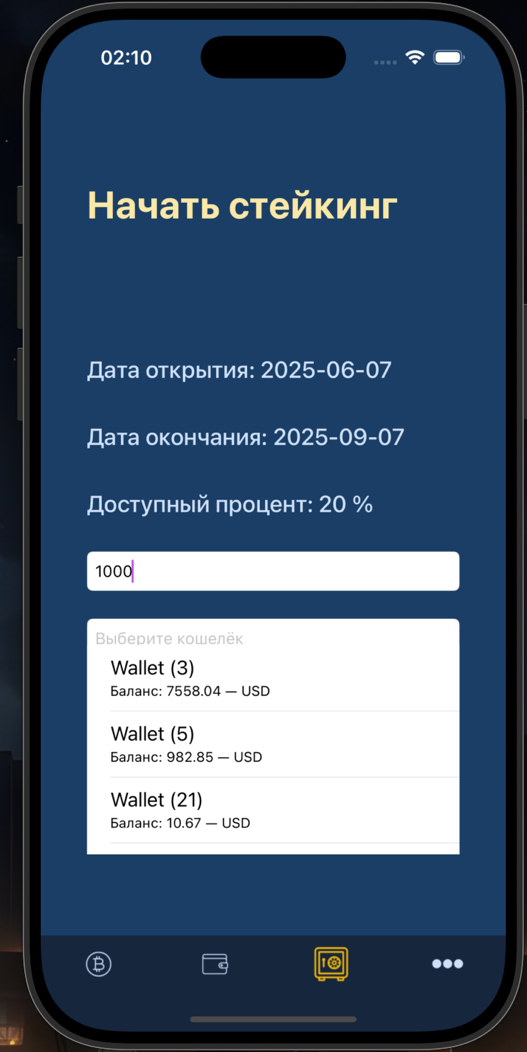


Рисунок 19 – Начать новый стейкинг, выбор кошелька



Рисунок 20 – Начать новый стейкинг

Созданный стейкинг был внесен в базу данных, привязан к кошельку. Результат представлен на рисунке 21.

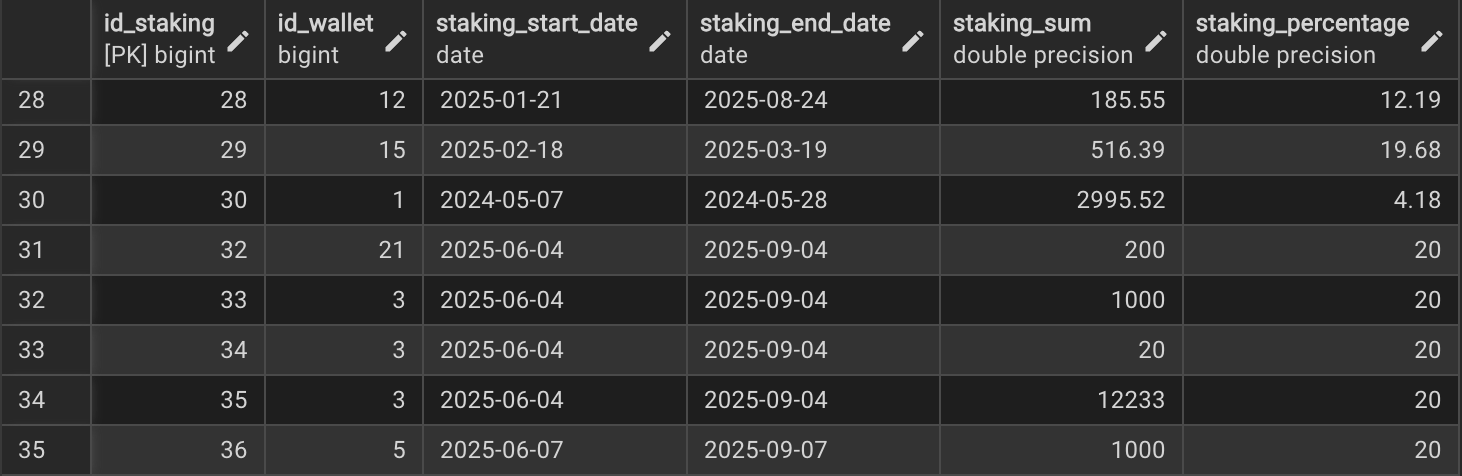


Рисунок 21 – Стейкинг в базе данных

Экран информации о пользователе. Результат представлен на рисунке 22.

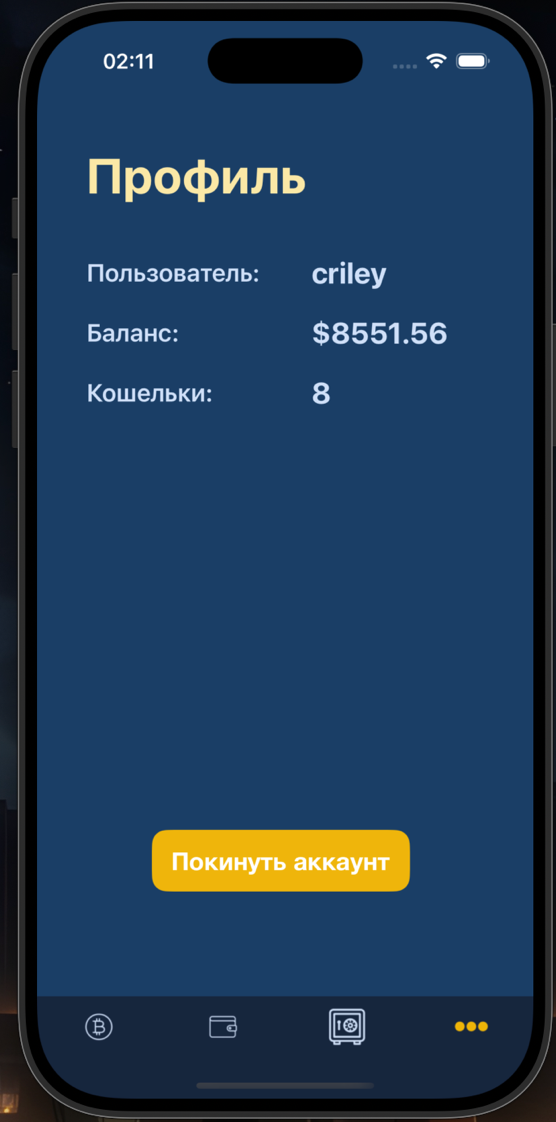


Рисунок 22 – Информация о пользователе

**Заключение**

В ходе работы был реализован полный цикл разработки системы — от построения инфологической и даталогической моделей до создания пользовательского интерфейса, обеспечивающего взаимодействие с удалённой базой данных.

На основе логического проектирования была разработана структура базы данных в PostgreSQL, отражающая основные сущности и связи системы. Далее был создан мобильный интерфейс на платформе iOS с использованием UIKit, через который пользователь может выполнять регистрацию, авторизацию, управлять кошельками, осуществлять покупку и продажу валюты, а также открывать стейкинг.

Приложение обеспечивает прямой доступ к функционалу базы данных, позволяя пользователю работать с данными в удобной и интерактивной форме прямо с экрана смартфона.

ПРИЛОЖЕНИЕ 1

Архив с проектом доступен по ссылке:

https://disk.yandex.ru/d/3\_rfLZsklfGE4w

ПРИЛОЖЕНИЕ 2

Код Python.

from fastapi import FastAPI, Request  
from fastapi.middleware.cors import CORSMiddleware  
import psycopg2  
  
app = FastAPI()  
  
app.add\_middleware(  
 CORSMiddleware,  
 allow\_origins=["\*"],  
 allow\_credentials=True,  
 allow\_methods=["\*"],  
 allow\_headers=["\*"],  
)  
  
conn = psycopg2.connect(  
 dbname="crypto\_exchange",  
 user="postgres",  
 password="1",  
 host="localhost",  
 port="5433"  
)  
  
@app.get("/cryptocurrencies")  
def get\_cryptocurrencies():  
 with conn.cursor() as cur:  
 cur.execute("SELECT id\_currency, currency\_name, currency\_price\_usd FROM cryptocurrency")  
 rows = cur.fetchall()  
 return [{"id": r[0], "name": r[1], "price": r[2]} for r in rows]  
  
@app.post("/check\_user")  
async def check\_user(request: Request):  
 data = await request.json()  
 name = data.get("name")  
 with conn.cursor() as cur:  
 cur.execute("SELECT id FROM users WHERE name = %s", (name,))  
 user = cur.fetchone()  
 return {"exists": bool(user)}  
  
@app.post("/register\_user")  
async def register\_user(request: Request):  
 data = await request.json()  
 name = data.get("name")  
 email = data.get("email")  
 password = data.get("password")  
  
 with conn.cursor() as cur:  
 cur.execute(  
 "INSERT INTO users (name, email, password) VALUES (%s, %s, %s) RETURNING id",  
 (name, email, password)  
 )  
 user\_id = cur.fetchone()[0]  
 conn.commit()  
 return {"id": user\_id, "name": name}  
  
@app.post("/authorize\_user")  
async def authorize\_user(request: Request):  
 data = await request.json()  
 email = data.get("email")  
 password = data.get("password")  
 with conn.cursor() as cur:  
 cur.execute("SELECT id, name FROM users WHERE email = %s AND password = %s", (email, password))  
 user = cur.fetchone()  
 if user:  
 return {"authorized": True, "id": user[0], "name": user[1]}  
 else:  
 return {"authorized": False}  
  
@app.get("/staking/{user\_id}")  
def get\_staking(user\_id: int):  
 with conn.cursor() as cur:  
 cur.execute("""  
 SELECT s.id\_staking, s.staking\_sum, s.staking\_percentage, s.staking\_start\_date, s.staking\_end\_date  
 FROM staking s  
 JOIN spot\_wallet w ON s.id\_wallet = w.id\_wallet  
 WHERE w.id\_user = %s  
 """, (user\_id,))  
 rows = cur.fetchall()  
 return [  
 {  
 "id": r[0],  
 "sum": r[1],  
 "percent": r[2],  
 "start": r[3],  
 "end": r[4]  
 }  
 for r in rows  
 ]  
  
@app.post("/add\_wallet")  
async def add\_wallet(request: Request):  
 data = await request.json()  
 currency = data["currency"]  
 balance = data["balance"]  
 user\_id = data["user\_id"]  
  
 with conn.cursor() as cur:  
 cur.execute(  
 "INSERT INTO spot\_wallet (currency, balance, id\_user) VALUES (%s, %s, %s)",  
 (currency, balance, user\_id)  
 )  
 conn.commit()  
 return {"status": "success"}  
  
@app.get("/wallets/{user\_id}")  
def get\_wallets(user\_id: int):  
 with conn.cursor() as cur:  
 cur.execute("""  
 SELECT id\_wallet, currency, balance   
 FROM spot\_wallet   
 WHERE id\_user = %s  
 """, (user\_id,))  
 rows = cur.fetchall()  
 return [  
 {  
 "id\_wallet": r[0],  
 "currency": r[1],  
 "balance": r[2]  
 }  
 for r in rows  
 ]  
  
@app.delete("/delete\_wallet/{wallet\_id}")  
def delete\_wallet(wallet\_id: int):  
 with conn.cursor() as cur:  
 cur.execute("SELECT balance FROM spot\_wallet WHERE id\_wallet = %s", (wallet\_id,))  
 result = cur.fetchone()  
 if result is None:  
 return {"status": "error", "reason": "Wallet not found"}  
 if result[0] != 0.0:  
 return {"status": "error", "reason": "Balance must be zero"}  
  
 cur.execute("DELETE FROM spot\_wallet WHERE id\_wallet = %s", (wallet\_id,))  
 conn.commit()  
 return {"status": "deleted"}  
  
#для создания нового стейкинга  
@app.post("/add\_staking")  
async def add\_staking(request: Request):  
 data = await request.json()  
 with conn.cursor() as cur:  
 cur.execute("""  
 INSERT INTO staking (id\_wallet, staking\_start\_date, staking\_end\_date, staking\_sum, staking\_percentage)  
 VALUES (%s, %s, %s, %s, %s)  
 """, (  
 data["id\_wallet"],  
 data["staking\_start\_date"],  
 data["staking\_end\_date"],  
 data["staking\_sum"],  
 data["staking\_percentage"]  
 ))  
 conn.commit()  
 return {"status": "success"}  
  
#для ордеров и транзакций  
  
from datetime import date  
  
@app.post("/create\_order")  
async def create\_order(request: Request):  
 data = await request.json()  
 with conn.cursor() as cur:  
 cur.execute("""  
 INSERT INTO orders (  
 id\_users, id\_currency, order\_count\_currency,  
 order\_type, order\_status  
 )  
 VALUES (%s, %s, %s, %s, %s)  
 RETURNING id\_order  
 """, (  
 data["id\_users"],  
 data["id\_currency"],  
 data["order\_count\_currency"],  
 data["order\_type"],  
 data["order\_status"]  
 ))  
 order\_id = cur.fetchone()[0]  
 conn.commit()  
 return {"id\_order": order\_id}  
  
@app.post("/create\_transaction")  
async def create\_transaction(request: Request):  
 data = await request.json()  
 with conn.cursor() as cur:  
 cur.execute("""  
 INSERT INTO transactions (  
 id\_wallet, id\_order, id\_currency,  
 trans\_data, trans\_status  
 )  
 VALUES (%s, %s, %s, %s, %s)  
 """, (  
 data["id\_wallet"],  
 data["id\_order"],  
 data["id\_currency"],  
 date.today(),  
 data["trans\_status"]  
 ))  
 conn.commit()  
 return {"status": "success"}