

```

import sqlite3
import logging
import random
import os
import asyncio
from datetime import datetime, timedelta
from typing import Dict, Optional, Tuple, List

from telegram import (
    Update,
    InlineKeyboardButton,
    InlineKeyboardMarkup,
    ReplyKeyboardMarkup,
    KeyboardButton,
    BotCommand
)
from telegram.ext import (
    Application,
    CommandHandler,
    MessageHandler,
    CallbackQueryHandler,
    ContextTypes,
    filters
)

# ===== КОНФИГУРАЦИЯ =====
BOT_TOKEN = "8244265951:AAFpmG4DRb640YLvURAhlySdpf6VVJgXX4g"
ADMIN_ID = 7973988177
SUPPORT_USERNAME = "@starfizovoi"

# Ссылки
CHANNEL_LINK = "https://t.me/nezeexshop"
PRIVACY_POLICY_LINK = "https://telegra.ph/Politika-konfidecialnosti-12-28"

# Курсы валют
EXCHANGE_RATES = {
    "USDT": 76.0,
    "TON": 115.0
}

# ПРАЙС-ЛИСТ СТРАН
COUNTRIES = {
    "usa": {"name": "🇺🇸 США", "price_rub": 30, "code": "+1"},
    "canada": {"name": "🇨🇦 Канада", "price_rub": 35, "code": "+1"},
    "russia": {"name": "🇷🇺 Россия", "price_rub": 199, "code": "+7"},
    "kazakhstan": {"name": "🇰🇿 Казахстан", "price_rub": 175, "code": "+7"},
    "egypt": {"name": "🇪🇬 Египет", "price_rub": 50, "code": "+20"},
    "south_africa": {"name": "🇿🇦 ЮАР", "price_rub": 100, "code": "+27"},

```

```

"greece": {"name": "🇬🇷 Греция", "price_rub": 175, "code": "+30"},
"netherlands": {"name": "🇳🇱 Нидерланды", "price_rub": 275, "code": "+31"},
"belgium": {"name": "🇧🇪 Бельгия", "price_rub": 1200, "code": "+32"},
"france": {"name": "🇫🇷 Франция", "price_rub": 250, "code": "+33"},
"spain": {"name": "🇪🇸 Испания", "price_rub": 250, "code": "+34"},
"hungary": {"name": "🇭🇺 Венгрия", "price_rub": 250, "code": "+36"},
"italy": {"name": "🇮🇹 Италия", "price_rub": 600, "code": "+39"},
"romania": {"name": "🇷🇴 Румыния", "price_rub": 80, "code": "+40"},
"switzerland": {"name": "🇨🇭 Швейцария", "price_rub": 2000, "code": "+41"},
"austria": {"name": "🇦🇹 Австрия", "price_rub": 1000, "code": "+43"},
"uk": {"name": "🇬🇧 Великобритания", "price_rub": 125, "code": "+44"},
"denmark": {"name": "🇩🇰 Дания", "price_rub": 1150, "code": "+45"},
"sweden": {"name": "🇸🇪 Швеция", "price_rub": 400, "code": "+46"},
"norway": {"name": "🇳🇴 Норвегия", "price_rub": 1150, "code": "+47"},
"poland": {"name": "🇵🇱 Польша", "price_rub": 275, "code": "+48"},
"brazil": {"name": "🇧🇷 Бразилия", "price_rub": 125, "code": "+55"},
"colombia": {"name": "🇨🇴 Колумбия", "price_rub": 75, "code": "+57"},
"indonesia": {"name": "🇮🇩 Индонезия", "price_rub": 50, "code": "+62"},
"vietnam": {"name": "🇻🇳 Вьетнам", "price_rub": 70, "code": "+84"},
"china": {"name": "🇨🇳 Китай", "price_rub": 750, "code": "+86"},
"turkey": {"name": "🇹🇷 Турция", "price_rub": 100, "code": "+90"},
"india": {"name": "🇮🇳 Индия", "price_rub": 40, "code": "+91"},
"pakistan": {"name": "🇵🇰 Пакистан", "price_rub": 70, "code": "+92"},
"afghanistan": {"name": "🇦🇫 Афганистан", "price_rub": 75, "code": "+93"},
"sri_lanka": {"name": "🇱🇰 Шри-Ланка", "price_rub": 100, "code": "+94"},
"myanmar": {"name": "🇲🇲 Мьянма", "price_rub": 35, "code": "+95"},
"iran": {"name": "🇮🇷 Иран", "price_rub": 175, "code": "+98"},
"morocco": {"name": "🇲🇦 Марокко", "price_rub": 75, "code": "+212"},
"ivory_coast": {"name": "🇨🇮 Кот-д'Ивуар", "price_rub": 750, "code": "+225"},
"ghana": {"name": "🇬🇭 Гана", "price_rub": 550, "code": "+233"},
"nigeria": {"name": "🇳🇮 Нигерия", "price_rub": 45, "code": "+234"},
"kenya": {"name": "🇰🇪 Кения", "price_rub": 40, "code": "+254"},
"moldova": {"name": "🇲🇩 Молдова", "price_rub": 175, "code": "+373"},
"armenia": {"name": "🇦🇲 Армения", "price_rub": 400, "code": "+374"},
"belarus": {"name": "🇧🇪Л Беларусь", "price_rub": 170, "code": "+375"},
"ukraine": {"name": "🇺🇦 Украина", "price_rub": 235, "code": "+380"}
}

```

Карта для оплаты

CARD_NUMBER = "5599 0021 2767 5173"

CRYPTO_BOT_LINK = "http://t.me/send?start=IVKF2M5j40O5" # ИСПРАВЛЕНА ССЫЛКА

Аккаунты с отлегой (будет загружаться из БД)

ACCOUNTS_WITH_OTL = {}

Состояния для админ-панели

```

(
    MAIN_MENU,

```

```

STATS_MENU,
BROADCAST_MENU,
PRICE_MENU,
WAITING_BROADCAST,
WAITING_PRICE_CHANGE,
WAITING_PRICE_VALUE,
WAITING_ADMIN_REPLY,
WAITING_PROMO_CREATE,
# Новые состояния для пошагового добавления аккаунта
WAITING_OTL_COUNTRY,
WAITING_OTL_NAME,
WAITING_OTL_CODE,
WAITING_OTL_PRICE,
WAITING_OTL_STOCK,
) = range(14)

# Настройка логирования
logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    level=logging.INFO
)
logger = logging.getLogger(__name__)

# ===== БАЗА ДАННЫХ =====

class Database:
    def __init__(self, db_name="bot_database.db"):
        self.db_name = db_name
        self.init_db()

    def get_connection(self):
        """Получение соединения с БД"""
        return sqlite3.connect(self.db_name)

    def init_db(self):
        """Инициализация базы данных"""
        with self.get_connection() as conn:
            cursor = conn.cursor()

            # Таблица пользователей
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS users (
                    user_id INTEGER PRIMARY KEY,
                    username TEXT,
                    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                    last_prize_claimed TIMESTAMP
                )
            """)

```

Таблица заказов

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS orders (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    order_id TEXT UNIQUE,
    user_id INTEGER,
    country_code TEXT,
    country_name TEXT,
    phone_code TEXT,
    price_rub INTEGER,
    status TEXT DEFAULT 'pending',
    payment_method TEXT,
    payment_screenshot TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    discount_percent INTEGER DEFAULT 0,
    discount_code TEXT,
    account_type TEXT DEFAULT 'fiz',
    FOREIGN KEY (user_id) REFERENCES users (user_id)
)
""")
```

Таблица выданных данных

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS issued_data (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    order_id TEXT,
    data_type TEXT,
    data_text TEXT,
    issued_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (order_id) REFERENCES orders (order_id)
)
""")
```

Таблица для ожидания ответов админа

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS pending_admin_replies (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    order_id TEXT,
    data_type TEXT,
    user_id INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
""")
```

Таблица промокодов

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS promo_codes (
```

```

code TEXT PRIMARY KEY,
discount_percent INTEGER,
created_by INTEGER,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
used_by INTEGER,
used_at TIMESTAMP,
is_active BOOLEAN DEFAULT 1,
max_uses INTEGER DEFAULT 1,
use_count INTEGER DEFAULT 0
)
"""

```

Таблица полученных призов

```

cursor.execute("""
CREATE TABLE IF NOT EXISTS user_prizes (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    prize_type TEXT,
    prize_value TEXT,
    claimed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users (user_id)
)
""")

```

Таблица аккаунтов с отлегой (отдельная таблица для управления)

```

cursor.execute("""
CREATE TABLE IF NOT EXISTS otl_accounts (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    country_code TEXT,
    country_name TEXT,
    otl_name TEXT,
    phone_code TEXT,
    price_rub INTEGER,
    stock INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_active BOOLEAN DEFAULT 1
)
""")

```

```

conn.commit()
logger.info("База данных инициализирована")

```

def add_user(self, user_id: int, username: str):

"""Добавление пользователя в БД"""

with self.get_connection() as conn:

cursor = conn.cursor()

cursor.execute(

"INSERT OR IGNORE INTO users (user_id, username) VALUES (?, ?)",

```

        (user_id, username or "")
    )
    conn.commit()

```

```

def create_order(self, order_id: str, user_id: int, country_code: str, country_name: str,
phone_code: str, price_rub: int, discount_code: str = None, discount_percent: int = 0,
account_type: str = "fiz"):
    """Создание нового заказа"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            """INSERT INTO orders
            (order_id, user_id, country_code, country_name, phone_code, price_rub, status,
discount_code, discount_percent, account_type)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)""",
            (order_id, user_id, country_code, country_name, phone_code, price_rub,
'pending', discount_code, discount_percent, account_type)
        )
        conn.commit()
        return order_id

```

```

def update_order_payment(self, order_id: str, payment_method: str, screenshot_path: str
= None):
    """Обновление информации об оплате заказа"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            """UPDATE orders
            SET payment_method = ?, payment_screenshot = ?, status = 'waiting_approval'
            WHERE order_id = ?""",
            (payment_method, screenshot_path, order_id)
        )
        conn.commit()
        return cursor.rowcount > 0

```

```

def update_order_status(self, order_id: str, status: str):
    """Обновление статуса заказа"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            "UPDATE orders SET status = ? WHERE order_id = ?",
            (status, order_id)
        )
        conn.commit()
        return cursor.rowcount > 0

```

```

def get_order(self, order_id: str) -> Optional[Tuple]:
    """Получение информации о заказе"""

```

```

with self.get_connection() as conn:
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM orders WHERE order_id = ?", (order_id,))
    return cursor.fetchone()

def get_user_orders(self, user_id: int, limit: int = 10) -> list:
    """Получение истории заказов пользователя"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            """SELECT order_id, country_name, price_rub, created_at, status, account_type
            FROM orders
            WHERE user_id = ?
            ORDER BY created_at DESC
            LIMIT ?""",
            (user_id, limit)
        )
        return cursor.fetchall()

def get_completed_user_orders(self, user_id: int, limit: int = 10) -> list:
    """Получение истории успешных покупок пользователя"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            """SELECT order_id, country_name, price_rub, created_at, account_type
            FROM orders
            WHERE user_id = ? AND status = 'completed'
            ORDER BY created_at DESC
            LIMIT ?""",
            (user_id, limit)
        )
        return cursor.fetchall()

def add_issued_data(self, order_id: str, data_type: str, data_text: str):
    """Добавление выданных данных"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO issued_data (order_id, data_type, data_text) VALUES (?, ?, ?)",
            (order_id, data_type, data_text)
        )
        conn.commit()
        return cursor.lastrowid

def get_issued_data(self, order_id: str, data_type: str = None) -> list:
    """Получение выданных данных для заказа"""
    with self.get_connection() as conn:
        cursor = conn.cursor()

```

```

        if data_type:
            cursor.execute(
                "SELECT data_text FROM issued_data WHERE order_id = ? AND data_type = ? ORDER BY issued_at DESC LIMIT 1",
                (order_id, data_type)
            )
        else:
            cursor.execute(
                "SELECT data_type, data_text FROM issued_data WHERE order_id = ? ORDER BY issued_at DESC",
                (order_id,)
            )
        return cursor.fetchall()

```

```

def check_order_ownership(self, order_id: str, user_id: int) -> bool:

```

```

    """Проверка принадлежности заказа пользователю"""

```

```

    with self.get_connection() as conn:

```

```

        cursor = conn.cursor()

```

```

        cursor.execute(

```

```

            "SELECT 1 FROM orders WHERE order_id = ? AND user_id = ?",
            (order_id, user_id)

```

```

        )

```

```

        return cursor.fetchone() is not None

```

```

def get_order_by_id(self, order_id: str):

```

```

    """Получение заказа по ID"""

```

```

    with self.get_connection() as conn:

```

```

        cursor = conn.cursor()

```

```

        cursor.execute(

```

```

            "SELECT * FROM orders WHERE order_id = ?",
            (order_id,)

```

```

        )

```

```

        row = cursor.fetchone()

```

```

        if row:

```

```

            columns = [description[0] for description in cursor.description]

```

```

            return dict(zip(columns, row))

```

```

        return None

```

```

# ===== ПРОМОКОДЫ
=====

```

```

def create_promo_code(self, code: str, discount_percent: int, created_by: int, max_uses:
int = 1):

```

```

    """Создание промокода"""

```

```

    with self.get_connection() as conn:

```

```

        cursor = conn.cursor()

```

```

        cursor.execute(

```

```

            """INSERT OR REPLACE INTO promo_codes

```

```

        (code, discount_percent, created_by, max_uses)
VALUES (?, ?, ?, ?)"""
        (code, discount_percent, created_by, max_uses)
    )
    conn.commit()
    return True

```

```

def get_promo_code(self, code: str):
    """Получение информации о промокоде"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            "SELECT * FROM promo_codes WHERE code = ? AND is_active = 1",
            (code,)
        )
        row = cursor.fetchone()
        if row:
            columns = [description[0] for description in cursor.description]
            return dict(zip(columns, row))
        return None

```

```

def use_promo_code(self, code: str, user_id: int):
    """Использование промокода"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        # Проверяем, не использовал ли уже пользователь этот код
        cursor.execute(
            "SELECT use_count, max_uses FROM promo_codes WHERE code = ?",
            (code,)
        )
        result = cursor.fetchone()
        if not result:
            return False

        use_count, max_uses = result
        if use_count >= max_uses:
            cursor.execute(
                "UPDATE promo_codes SET is_active = 0 WHERE code = ?",
                (code,)
            )
            conn.commit()
            return False

```

```

# Увеличиваем счетчик использования
cursor.execute(
    """UPDATE promo_codes
SET use_count = use_count + 1,
    used_by = ?,

```

```

        used_at = CURRENT_TIMESTAMP
        WHERE code = ?""",
        (user_id, code)
    )

    # Если достигли максимального количества использований
    if use_count + 1 >= max_uses:
        cursor.execute(
            "UPDATE promo_codes SET is_active = 0 WHERE code = ?",
            (code,)
        )

    conn.commit()
    return True

```

```

def get_all_promo_codes(self):
    """Получение всех промокодов"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM promo_codes ORDER BY created_at DESC")
        rows = cursor.fetchall()
        if rows:
            columns = [description[0] for description in cursor.description]
            return [dict(zip(columns, row)) for row in rows]
    return []

```

```

# ===== НОВОГОДНИЕ ПРИЗЫ
=====

```

```

def can_claim_prize(self, user_id: int) -> bool:
    """Проверка, может ли пользователь получить приз"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            "SELECT last_prize_claimed FROM users WHERE user_id = ?""",
            (user_id,)
        )
        result = cursor.fetchone()
        if not result or not result[0]:
            return True

        last_claimed = datetime.strptime(result[0], "%Y-%m-%d %H:%M:%S")
        return (datetime.now() - last_claimed).total_seconds() >= 24 * 3600

def claim_prize(self, user_id: int, prize_type: str, prize_value: str):
    """Запись получения приза"""
    with self.get_connection() as conn:
        cursor = conn.cursor()

```

```

# Обновляем время последнего получения приза
cursor.execute(
    """UPDATE users
    SET last_prize_claimed = CURRENT_TIMESTAMP
    WHERE user_id = ?""",
    (user_id,)
)
# Добавляем запись о призе
cursor.execute(
    """INSERT INTO user_prizes (user_id, prize_type, prize_value)
    VALUES (?, ?, ?)""",
    (user_id, prize_type, prize_value)
)
conn.commit()
return cursor.lastrowid

```

```

def get_user_prizes(self, user_id: int, limit: int = 10) -> list:

```

```

    """Получение истории призов пользователя"""

```

```

    with self.get_connection() as conn:

```

```

        cursor = conn.cursor()

```

```

        cursor.execute(

```

```

            """SELECT prize_type, prize_value, claimed_at

```

```

            FROM user_prizes

```

```

            WHERE user_id = ?

```

```

            ORDER BY claimed_at DESC

```

```

            LIMIT ?""",

```

```

            (user_id, limit)

```

```

        )

```

```

        return cursor.fetchall()

```

```

# ===== АККАУНТЫ С ОТЛЕГОЙ

```

```

=====

```

```

def get_all_otl_accounts(self):

```

```

    """Получение всех аккаунтов с отлегой"""

```

```

    with self.get_connection() as conn:

```

```

        cursor = conn.cursor()

```

```

        cursor.execute("SELECT * FROM otl_accounts WHERE is_active = 1 ORDER BY
country_name")

```

```

        rows = cursor.fetchall()

```

```

        if rows:

```

```

            columns = [description[0] for description in cursor.description]

```

```

            return [dict(zip(columns, row)) for row in rows]

```

```

        return []

```

```

def get_otl_account(self, account_id: int):

```

```

    """Получение аккаунта с отлегой по ID"""

```

```

    with self.get_connection() as conn:

```

```

        cursor = conn.cursor()
        cursor.execute(
            "SELECT * FROM otl_accounts WHERE id = ? AND is_active = 1",
            (account_id,)
        )
        row = cursor.fetchone()
        if row:
            columns = [description[0] for description in cursor.description]
            return dict(zip(columns, row))
        return None

def get_otl_account_by_code(self, country_code: str):
    """Получение аккаунта с отлегой по коду страны"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            "SELECT * FROM otl_accounts WHERE country_code = ? AND is_active = 1",
            (country_code,)
        )
        row = cursor.fetchone()
        if row:
            columns = [description[0] for description in cursor.description]
            return dict(zip(columns, row))
        return None

def update_otl_account_stock(self, account_id: int, new_stock: int):
    """Обновление количества аккаунтов с отлегой"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            "UPDATE otl_accounts SET stock = ? WHERE id = ?",
            (new_stock, account_id)
        )
        conn.commit()
        return cursor.rowcount > 0

def create_otl_account(self, country_code: str, country_name: str, otl_name: str,
phone_code: str, price_rub: int, stock: int):
    """Создание нового аккаунта с отлегой"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            """INSERT INTO otl_accounts
            (country_code, country_name, otl_name, phone_code, price_rub, stock)
            VALUES (?, ?, ?, ?, ?, ?)""",
            (country_code, country_name, otl_name, phone_code, price_rub, stock)
        )
        conn.commit()

```

```
return cursor.lastrowid
```

```
def update_otl_account(self, account_id: int, price_rub: int = None, stock: int = None):
```

```
    """Обновление аккаунта с отлегой"""
```

```
    with self.get_connection() as conn:
```

```
        cursor = conn.cursor()
```

```
        updates = []
```

```
        params = []
```

```
        if price_rub is not None:
```

```
            updates.append("price_rub = ?")
```

```
            params.append(price_rub)
```

```
        if stock is not None:
```

```
            updates.append("stock = ?")
```

```
            params.append(stock)
```

```
        if not updates:
```

```
            return False
```

```
        params.append(account_id)
```

```
        query = f"UPDATE otl_accounts SET {'', '.join(updates)} WHERE id = ?"
```

```
        cursor.execute(query, params)
```

```
        conn.commit()
```

```
        return cursor.rowcount > 0
```

```
def deactivate_otl_account(self, account_id: int):
```

```
    """Деактивация аккаунта с отлегой"""
```

```
    with self.get_connection() as conn:
```

```
        cursor = conn.cursor()
```

```
        cursor.execute(
```

```
            "UPDATE otl_accounts SET is_active = 0 WHERE id = ?",
```

```
            (account_id,)
```

```
        )
```

```
        conn.commit()
```

```
        return cursor.rowcount > 0
```

```
# ===== АДМИН ФУНКЦИИ
```

```
=====
```

```
def get_statistics(self):
```

```
    """Получение статистики"""
```

```
    with self.get_connection() as conn:
```

```
        cursor = conn.cursor()
```

```
        # Общее количество пользователей
```

```
        cursor.execute("SELECT COUNT(*) FROM users")
```

```
        total_users = cursor.fetchone()[0]
```

```

# Количество новых пользователей за последние 24 часа
cursor.execute(
    "SELECT COUNT(*) FROM users WHERE created_at >= datetime('now', '-1 day')"
)
new_users_24h = cursor.fetchone()[0]

# Общее количество заказов
cursor.execute("SELECT COUNT(*) FROM orders")
total_orders = cursor.fetchone()[0]

# Заказы за последние 24 часа
cursor.execute(
    "SELECT COUNT(*) FROM orders WHERE created_at >= datetime('now', '-1
day')"
)
new_orders_24h = cursor.fetchone()[0]

# Выручка за все время
cursor.execute(
    "SELECT SUM(price_rub) FROM orders WHERE status = 'completed'"
)
total_revenue = cursor.fetchone()[0] or 0

# Выручка за последние 24 часа
cursor.execute(
    "SELECT SUM(price_rub) FROM orders WHERE status = 'completed' AND
created_at >= datetime('now', '-1 day')"
)
revenue_24h = cursor.fetchone()[0] or 0

# Статусы заказов
cursor.execute(
    "SELECT status, COUNT(*) FROM orders GROUP BY status"
)
status_stats = cursor.fetchall()

# Количество промокодов
cursor.execute("SELECT COUNT(*) FROM promo_codes")
total_promo_codes = cursor.fetchone()[0]

# Активные промокоды
cursor.execute("SELECT COUNT(*) FROM promo_codes WHERE is_active = 1")
active_promo_codes = cursor.fetchone()[0]

# Полученные призы
cursor.execute("SELECT COUNT(*) FROM user_prizes")
total_prizes = cursor.fetchone()[0]

```

```

# Аккаунты с отлегой
cursor.execute("SELECT COUNT(*) FROM otl_accounts WHERE is_active = 1")
total_otl_accounts = cursor.fetchone()[0]

# Аккаунты с отлегой в наличии
cursor.execute("SELECT COUNT(*) FROM otl_accounts WHERE is_active = 1 AND
stock > 0")
available_otl_accounts = cursor.fetchone()[0]

# Заказы аккаунтов с отлегой
cursor.execute("SELECT COUNT(*) FROM orders WHERE account_type = 'otl'")
otl_orders = cursor.fetchone()[0]

return {
    'total_users': total_users,
    'new_users_24h': new_users_24h,
    'total_orders': total_orders,
    'new_orders_24h': new_orders_24h,
    'total_revenue': total_revenue,
    'revenue_24h': revenue_24h,
    'status_stats': dict(status_stats),
    'total_promo_codes': total_promo_codes,
    'active_promo_codes': active_promo_codes,
    'total_prizes': total_prizes,
    'total_otl_accounts': total_otl_accounts,
    'available_otl_accounts': available_otl_accounts,
    'otl_orders': otl_orders,
}

def get_all_users(self):
    """Получение всех пользователей"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT user_id FROM users")
        return [row[0] for row in cursor.fetchall()]

def update_country_price(self, country_code: str, new_price: int):
    """Обновление цены страны"""
    if country_code in COUNTRIES:
        COUNTRIES[country_code]['price_rub'] = new_price
        return True
    return False

def get_pending_admin_reply(self, order_id: str, data_type: str):
    """Получение ожидающего ответа админа"""
    with self.get_connection() as conn:
        cursor = conn.cursor()

```

```

        cursor.execute(
            "SELECT user_id FROM pending_admin_replies WHERE order_id = ? AND
data_type = ?",
            (order_id, data_type)
        )
        result = cursor.fetchone()
        return result[0] if result else None

def add_pending_admin_reply(self, order_id: str, data_type: str, user_id: int):
    """Добавление ожидающего ответа админа"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO pending_admin_replies (order_id, data_type, user_id) VALUES (?,
?, ?)",
            (order_id, data_type, user_id)
        )
        conn.commit()

def remove_pending_admin_reply(self, order_id: str, data_type: str):
    """Удаление ожидающего ответа админа"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            "DELETE FROM pending_admin_replies WHERE order_id = ? AND data_type =
?",
            (order_id, data_type)
        )
        conn.commit()

# Инициализация базы данных
db = Database()

# Загрузка аккаунтов с отлегой из БД
def load_otl_accounts_from_db():
    """Загрузка аккаунтов с отлегой из базы данных"""
    global ACCOUNTS_WITH_OTL
    ACCOUNTS_WITH_OTL = {}
    accounts = db.get_all_otl_accounts()
    for account in accounts:
        # Создаем уникальный ключ для аккаунта
        key = f"otl_{account['id']}"
        ACCOUNTS_WITH_OTL[key] = {
            'id': account['id'],
            'name': f"{account['country_name']} с отлегой",
            'otl': account['otl_name'],
            'price_rub': account['price_rub'],
            'stock': account['stock'],

```

```

        'code': account['phone_code'],
        'country_name': account['country_name'],
        'country_code': account['country_code']
    }

# Загружаем аккаунты при старте
load_otl_accounts_from_db()

# ===== ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ =====

def generate_order_id() -> str:
    """Генерация уникального ID заказа"""
    timestamp = datetime.now().strftime("%Y%m%d")
    random_part = random.randint(10000, 99999)
    return f"ORD-{random_part}"

def format_price(price_rub: int) -> str:
    """Форматирование цены в разных валютах"""
    usdt_price = price_rub / EXCHANGE_RATES["USDT"]
    ton_price = price_rub / EXCHANGE_RATES["TON"]
    return f"~{usdt_price:.3f} USDT / ~{ton_price:.3f} TON"

def create_main_keyboard():
    """Создание главной клавиатуры для пользователей"""
    keyboard = [
        [KeyboardButton("🛒 Купить аккаунт"), KeyboardButton("👤 Профиль")],
        [KeyboardButton("🎁 Новогодние призы"), KeyboardButton("📄 Промокод")],
        [KeyboardButton("ℹ️ О нас"), KeyboardButton("🆘 Поддержка")]
    ]
    return ReplyKeyboardMarkup(keyboard, resize_keyboard=True)

def create_admin_keyboard():
    """Создание клавиатуры для админа"""
    keyboard = [
        [KeyboardButton("/admin")],
        [KeyboardButton("🛒 Купить аккаунт"), KeyboardButton("👤 Профиль")],
        [KeyboardButton("🎁 Новогодние призы"), KeyboardButton("📄 Промокод")],
        [KeyboardButton("ℹ️ О нас"), KeyboardButton("🆘 Поддержка")]
    ]
    return ReplyKeyboardMarkup(keyboard, resize_keyboard=True)

def create_countries_keyboard(page: int = 0, account_type: str = "fiz"):
    """Создание клавиатуры с пагинацией стран"""
    if account_type == "fiz":
        countries_list = list(COUNTRIES.items())
        items_per_page = 6
    elif account_type == "otl":

```

```

countries_list = list(ACCOUNTS_WITH_OTL.items())
items_per_page = 4
else:
    countries_list = []
    items_per_page = 6

total_pages = max(1, (len(countries_list) + items_per_page - 1) // items_per_page)
page = min(page, total_pages - 1)

start_idx = page * items_per_page
end_idx = start_idx + items_per_page
page_countries = countries_list[start_idx:end_idx]

keyboard = []

for code, info in page_countries:
    if account_type == "otl":
        stock_status = "✅" if info['stock'] > 0 else "❌"
        button_text = f"{info['country_name']} с отлегой - {info['price_rub']}₽ {stock_status}"
    else:
        button_text = f"{info['name']} - {info['price_rub']}₽"
    button = InlineKeyboardButton(button_text, callback_data=f"country_{code}")
    keyboard.append([button])

# Кнопки навигации
navigation_buttons = []
if page > 0:
    navigation_buttons.append(InlineKeyboardButton("◀ Назад",
callback_data=f"page_{page-1}_{account_type}"))

if page < total_pages - 1:
    navigation_buttons.append(InlineKeyboardButton("Вперед ▶",
callback_data=f"page_{page+1}_{account_type}"))

if navigation_buttons:
    keyboard.append(navigation_buttons)

# Кнопка "Назад" к выбору типа аккаунта
keyboard.append([InlineKeyboardButton("⬅ Назад к выбору типа",
callback_data="back_to_types")])

return InlineKeyboardMarkup(keyboard)

def create_account_types_keyboard():
    """Создание клавиатуры выбора типа аккаунта"""
    keyboard = [
        [InlineKeyboardButton("🏠 ФИЗ аккаунты", callback_data="type_fiz")],
        [InlineKeyboardButton("👤 Аккаунты с отлегой", callback_data="type_otl")],
    ]

```

```

]
return InlineKeyboardMarkup(keyboard)

def create_admin_panel_keyboard():
    """Создание клавиатуры админ-панели"""
    keyboard = [
        [InlineKeyboardButton("📊 Статистика", callback_data="admin_stats"),
         InlineKeyboardButton("✉️ Рассылка", callback_data="admin_broadcast")],
        [InlineKeyboardButton("💰 Изменить цены", callback_data="admin_prices"),
         InlineKeyboardButton("🎟 Промокоды", callback_data="admin_promos")],
        [InlineKeyboardButton("👉 Управление отлегой", callback_data="admin_otl")],
        [InlineKeyboardButton("⬅ Назад", callback_data="admin_back")]
    ]
    return InlineKeyboardMarkup(keyboard)

def create_stats_keyboard():
    """Создание клавиатуры статистики"""
    keyboard = [
        [InlineKeyboardButton("🔄 Обновить", callback_data="admin_stats_refresh")],
        [InlineKeyboardButton("⬅ Назад", callback_data="admin_back_to_main")]
    ]
    return InlineKeyboardMarkup(keyboard)

def create_broadcast_keyboard():
    """Создание клавиатуры рассылки"""
    keyboard = [
        [InlineKeyboardButton("❌ Отмена", callback_data="admin_cancel_broadcast")],
        [InlineKeyboardButton("⬅ Назад", callback_data="admin_back_to_main")]
    ]
    return InlineKeyboardMarkup(keyboard)

def create_price_change_keyboard():
    """Создание клавиатуры изменения цен"""
    keyboard = []
    # Первые 10 стран
    countries_list = list(COUNTRIES.items())[:10]
    for code, info in countries_list:
        button = InlineKeyboardButton(
            f"{info['name']} - {info['price_rub']}₽",
            callback_data=f"admin_change_price_{code}"
        )
        keyboard.append([button])

    keyboard.append([
        InlineKeyboardButton("⬅ Назад", callback_data="admin_back_to_main")
    ])

    return InlineKeyboardMarkup(keyboard)

```

```
def create_promo_admin_keyboard():
    """Создание клавиатуры управления промокодами"""
    keyboard = [
        [InlineKeyboardButton("➕ Создать промокод",
callback_data="admin_create_promo")],
        [InlineKeyboardButton("📋 Список промокодов", callback_data="admin_list_promos")],
        [InlineKeyboardButton("⬅ Назад", callback_data="admin_back_to_main")]
    ]
    return InlineKeyboardMarkup(keyboard)
```

```
def create_otl_admin_keyboard():
    """Создание клавиатуры управления аккаунтами с отлегой"""
    keyboard = []
    accounts = db.get_all_otl_accounts()
    for account in accounts:
        stock_status = "✅" if account['stock'] > 0 else "❌"
        button_text = f"{account['country_name']} - {account['price_rub']}₽ {stock_status}"
        button = InlineKeyboardButton(button_text,
callback_data=f"admin_otl_edit_{account['id']}")
        keyboard.append([button])
```

```
        keyboard.append([InlineKeyboardButton("➕ Добавить аккаунт",
callback_data="admin_otl_add")])
        keyboard.append([InlineKeyboardButton("⬅ Назад",
callback_data="admin_back_to_main")])
```

```
    return InlineKeyboardMarkup(keyboard)
```

```
def create_otl_country_keyboard():
    """Создание клавиатуры выбора страны для отлегой"""
    keyboard = []
    # Группируем страны по 2 в ряд
    countries_list = list(COUNTRIES.items())
    for i in range(0, len(countries_list), 2):
        row = []
        for j in range(2):
            if i + j < len(countries_list):
                code, info = countries_list[i + j]
                button_text = info['name']
                row.append(InlineKeyboardButton(button_text,
callback_data=f"otl_country_{code}"))
        if row:
            keyboard.append(row)
```

```
        keyboard.append([InlineKeyboardButton("⬅ Отмена", callback_data="admin_otl")])
```

```
    return InlineKeyboardMarkup(keyboard)
```

```
def create_prize_keyboard(can_claim: bool = True):
    """Создание клавиатуры для новогодних призов"""
    keyboard = []
    if can_claim:
        keyboard.append([InlineKeyboardButton("🎁 Забрать приз",
        callback_data="claim_prize"))]
        keyboard.append([InlineKeyboardButton("📜 История призов",
        callback_data="prize_history"))]
        keyboard.append([InlineKeyboardButton("⬅️ Назад", callback_data="back_to_main")])
    return InlineKeyboardMarkup(keyboard)
```

```
def create_about_keyboard():
    """Создание клавиатуры для раздела 'О нас'"""
    keyboard = [
        [InlineKeyboardButton("🗣️ Наш канал", url=CHANNEL_LINK)],
        [InlineKeyboardButton("🔒 Политика конфиденциальности",
        url=PRIVACY_POLICY_LINK)],
        [InlineKeyboardButton("⬅️ Назад", callback_data="back_to_main")]
    ]
    return InlineKeyboardMarkup(keyboard)
```

--- ОСНОВНЫЕ ОБРАБОТЧИКИ ---

```
async def start_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обработчик команды /start"""
    try:
        user = update.effective_user
        logger.info(f"Пользователь {user.id} запустил бота")

        db.add_user(user.id, user.username)

        welcome_text = (
            "🎄 Добро пожаловать в бота для покупки аккаунтов!\n\n"
            "🎁 Доступные функции:\n"
            "🛒 Купить аккаунт - выбор страны и оплата\n"
            "👤 Профиль - ваши данные и история покупок\n"
            "🎁 Новогодние призы - получайте подарки каждый день!\n"
            "🎫 Промокод - введите промокод для скидки\n"
            "📄 О нас - информация о нас и правила\n"
            "🆘 Поддержка - связь с администратором\n\n"
            "🎉 С Новым 2026 Годом! 🎉"
        )

        if user.id == ADMIN_ID:
            await update.message.reply_text(
                welcome_text,
                reply_markup=create_admin_keyboard()
```

```

    )
    else:
        await update.message.reply_text(
            welcome_text,
            reply_markup=create_main_keyboard()
        )
except Exception as e:
    logger.error(f"Ошибка в start_command: {e}")
    await update.message.reply_text("Произошла ошибка. Попробуйте еще раз.")

async def admin_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обработчик команды /admin"""
    user = update.effective_user

    if user.id != ADMIN_ID:
        await update.message.reply_text("У вас нет доступа к админ-панели.")
        return

    admin_text = (
        "⚙️ Админ-панель\n\n"
        "Выберите действие:"
    )
    await update.message.reply_text(
        admin_text,
        reply_markup=create_admin_panel_keyboard()
    )
    return MAIN_MENU

async def handle_text_message(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обработчик текстовых сообщений (основной)"""
    try:
        user = update.effective_user
        text = update.message.text

        logger.info(f"Получено сообщение от {user.id}: {text}")

        # Проверяем состояние для админа
        if user.id == ADMIN_ID and 'admin_state' in context.user_data:
            state = context.user_data['admin_state']
            if state == WAITING_BROADCAST:
                await process_broadcast(update, context, text)
                return
            elif state == WAITING_PRICE_VALUE:
                await process_price_change(update, context, text)
                return
            elif state == WAITING_ADMIN_REPLY:
                await process_admin_reply(update, context, text)
                return
    
```

```

elif state == WAITING_PROMO_CREATE:
    await process_promo_create(update, context, text)
    return
elif state == WAITING_OTL_COUNTRY:
    await process_otl_country(update, context, text)
    return
elif state == WAITING_OTL_NAME:
    await process_otl_name(update, context, text)
    return
elif state == WAITING_OTL_CODE:
    await process_otl_code(update, context, text)
    return
elif state == WAITING_OTL_PRICE:
    await process_otl_price(update, context, text)
    return
elif state == WAITING_OTL_STOCK:
    await process_otl_stock(update, context, text)
    return

# Проверка промокода
if text == "👉 Промокод":
    await update.message.reply_text(
        "👉 Введите промокод для получения скидки:",
        reply_markup=ReplyKeyboardMarkup([[KeyboardButton("← Отмена")]]),
        resize_keyboard=True)
    )
    context.user_data['waiting_promo'] = True
    return

if 'waiting_promo' in context.user_data and context.user_data['waiting_promo']:
    if text == "← Отмена":
        keyboard = create_admin_keyboard() if user.id == ADMIN_ID else
create_main_keyboard()
        await update.message.reply_text(
            "Действие отменено.",
            reply_markup=keyboard
        )
        context.user_data.pop('waiting_promo', None)
        return

promo_info = db.get_promo_code(text.upper())
if promo_info:
    context.user_data['current_promo'] = {
        'code': text.upper(),
        'discount': promo_info['discount_percent']
    }
    await update.message.reply_text(
        f"✅ Промокод активирован!\n"

```

```

        f"📄 Код: {text.upper()}\n"
        f"💰 Скидка: {promo_info['discount_percent']}%\n\n"
        f"Скидка будет применена к следующей покупке.",
        reply_markup=create_admin_keyboard() if user.id == ADMIN_ID else
create_main_keyboard()
    )
    else:
        await update.message.reply_text(
            "❌ Неверный или неактивный промокод.",
            reply_markup=create_admin_keyboard() if user.id == ADMIN_ID else
create_main_keyboard()
        )
        context.user_data.pop('waiting_promo', None)
        return

# Обычная обработка для всех пользователей
if text == "🛒 Купить аккаунт":
    await show_account_types(update, context)
elif text == "👤 Профиль":
    await show_profile(update, context)
elif text == "🎁 Новогодние призы":
    await show_new_year_prizes(update, context)
elif text == "ℹ️ О нас":
    await show_about(update, context)
elif text == "🆘 Поддержка":
    await show_support(update, context)
elif text == "/admin" and user.id == ADMIN_ID:
    await admin_command(update, context)
else:
    # Проверяем, не ждем ли мы скриншот
    if 'waiting_screenshot_for' in context.user_data:
        await update.message.reply_text("Пожалуйста, отправьте скриншот (фото или
документ).")
    else:
        keyboard = create_admin_keyboard() if user.id == ADMIN_ID else
create_main_keyboard()
        await update.message.reply_text(
            "Используйте меню ниже:",
            reply_markup=keyboard
        )

except Exception as e:
    logger.error(f"Ошибка в handle_text_message: {e}")
    await update.message.reply_text("Произошла ошибка. Попробуйте еще раз.")

async def show_support(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Показать информацию о поддержке"""
    support_text = f"🆘 По всем вопросам обращайтесь: {SUPPORT_USERNAME}"

```

```
await update.message.reply_text(support_text)
```

```
async def show_about(update: Update, context: ContextTypes.DEFAULT_TYPE):
```

```
    """Показать информацию 'О нас'"""
```

```
    about_text = (
```

```
        "📄 О нас\n\n"
```

```
        "Мы предоставляем качественные аккаунты для различных целей.\n\n"
```

```
        "👉 Подпишитесь на наш канал, чтобы быть в курсе новостей и акций!\n"
```

```
        "🔒 Ознакомьтесь с нашей политикой конфиденциальности.\n\n"
```

```
        "Выберите нужный раздел ниже:"
```

```
)
```

```
await update.message.reply_text(
```

```
    about_text,
```

```
    reply_markup=create_about_keyboard()
```

```
)
```

```
async def show_profile(update: Update, context: ContextTypes.DEFAULT_TYPE):
```

```
    """Показать профиль пользователя"""
```

```
    try:
```

```
        user = update.effective_user
```

```
        orders = db.get_completed_user_orders(user.id)
```

```
        prizes = db.get_user_prizes(user.id, 5)
```

```
        profile_text = f"👤 Ваш профиль:\n"
```

```
        profile_text += f"└ ID: `{user.id}`\n"
```

```
        profile_text += f"└ Юзернейм: @{user.username if user.username else
```

```
'Отсутствует'}\n"
```

```
        profile_text += f"└ История покупок:\n"
```

```
    if orders:
```

```
        for order in orders:
```

```
            order_id, country_name, price_rub, created_at, account_type = order
```

```
            try:
```

```
                date_str = datetime.strptime(created_at, "%Y-%m-%d %H:%M:%S").strftime("%Y-%m-%d")
```

```
            except:
```

```
                date_str = created_at[:10]
```

```
            order_short_id = order_id.split('-')[1] if len(order_id.split('-')) > 1 else order_id
```

```
            account_type_icon = "👉" if account_type == "otl" else "📱"
```

```
            profile_text += f"└ • {date_str} | {account_type_icon} {country_name}
```

```
(#{order_short_id})\n"
```

```
    else:
```

```
        profile_text += "└ Пока нет покупок\n"
```

```
    profile_text += f"└ Полученные призы:\n"
```

```
    if prizes:
```

```

for prize in prizes:
    prize_type, prize_value, claimed_at = prize
    date_str = claimed_at[:10]
    if prize_type == "promo":
        profile_text += f"| • {date_str} | 🎟 Промокод {prize_value}%\n"
    elif prize_type == "account":
        profile_text += f"| • {date_str} | 🎁 Бесплатный аккаунт\n"
else:
    profile_text += "| Пока нет призов\n"

await update.message.reply_text(profile_text, parse_mode='Markdown')
except Exception as e:
    logger.error(f"Ошибка в show_profile: {e}")
    await update.message.reply_text("Ошибка при загрузке профиля.")

async def show_new_year_prizes(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    """Показать новогодние призы"""
    user = update.effective_user
    can_claim = db.can_claim_prize(user.id)

    prize_text = (
        "🎄 Новогодние Призы 2026! 🎄\n\n"
        "🎁 Каждый день вы можете получить один приз:\n"
        "• 🎟 Промокод на скидку (1-30%)\n"
        "• 🎁 Бесплатный ФИЗ аккаунт (шанс 1%)\n\n"
    )

    if can_claim:
        prize_text += "🎯 Вы можете забрать приз сейчас!"
    else:
        last_prize = db.get_user_prizes(user.id, 1)
        if last_prize:
            last_time = datetime.strptime(last_prize[0][2], "%Y-%m-%d %H:%M:%S")
            next_time = last_time + timedelta(hours=24)
            time_left = next_time - datetime.now()
            hours_left = max(0, int(time_left.total_seconds() // 3600))
            minutes_left = max(0, int((time_left.total_seconds() % 3600) // 60))
            prize_text += f"⌚ Следующий приз можно забрать через {hours_left}ч {minutes_left}м"
        else:
            prize_text += "🎯 Вы можете забрать приз сейчас!"

    await update.message.reply_text(
        prize_text,
        reply_markup=create_prize_keyboard(can_claim)
    )

```

```

async def show_account_types(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Показать типы аккаунтов"""
    keyboard = create_account_types_keyboard()

    await update.message.reply_text(
        "Выберите тип аккаунта:",
        reply_markup=keyboard
    )

```

```

async def handle_country_page(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обработка пагинации стран"""
    query = update.callback_query
    await query.answer()

```

```

try:
    data_parts = query.data.split("_")
    page = int(data_parts[1])
    account_type = data_parts[2] if len(data_parts) > 2 else "fiz"

    reply_markup = create_countries_keyboard(page, account_type)

    if account_type == "otl":
        text = "👤 Аккаунты с отлегой:\n✅ - в наличии\n❌ - нет в наличии"
    else:
        text = "Выберите страну:"

    await query.message.edit_text(
        text,
        reply_markup=reply_markup
    )
except Exception as e:
    logger.error(f"Ошибка в handle_country_page: {e}")
    await query.message.edit_text("Произошла ошибка. Попробуйте еще раз.")

```

```

async def show_countries(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Показать список стран для выбора"""
    query = update.callback_query
    await query.answer()

    try:
        reply_markup = create_countries_keyboard(0, "fiz")
        await query.message.edit_text(
            "Выберите страну:",
            reply_markup=reply_markup
        )
    except Exception as e:
        logger.error(f"Ошибка в show_countries: {e}")
        await query.message.edit_text("Произошла ошибка. Попробуйте еще раз.")

```

```

async def show_otl_countries(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Показать список аккаунтов с отлегой"""
    query = update.callback_query
    await query.answer()

    try:
        reply_markup = create_countries_keyboard(0, "otl")
        await query.message.edit_text(
            "👉 Аккаунты с отлегой:\n✅ - в наличии\n❌ - нет в наличии",
            reply_markup=reply_markup
        )
    except Exception as e:
        logger.error(f"Ошибка в show_otl_countries: {e}")
        await query.message.edit_text("Произошла ошибка. Попробуйте еще раз.")

async def show_order_details(update: Update, context: ContextTypes.DEFAULT_TYPE,
country_code: str):
    """Показать детали заказа"""
    query = update.callback_query
    await query.answer()

    try:
        # Определяем тип аккаунта
        account_type = "fiz"
        if country_code.startswith("otl_"):
            account_type = "otl"
            if country_code not in ACCOUNTS_WITH_OTL:
                await query.message.edit_text("❌ Ошибка: аккаунт не найден.")
                return
            account_info = ACCOUNTS_WITH_OTL[country_code]

            # Проверяем наличие
            if account_info['stock'] <= 0:
                await query.message.edit_text("❌ Извините, этот аккаунт временно отсутствует в наличии.")
                return

            country_name = account_info['country_name'] + " с отлегой"
            phone_code = account_info['code']
            price_rub = account_info['price_rub']

        else:
            if country_code not in COUNTRIES:
                await query.message.edit_text("❌ Ошибка: страна не найдена.")
                return
            country_info = COUNTRIES[country_code]
            country_name = country_info['name']

```

```

    phone_code = country_info['code']
    price_rub = country_info['price_rub']

order_id = generate_order_id()

# Применяем промокод если есть
discount_percent = 0
discount_code = None
final_price = price_rub

if 'current_promo' in context.user_data:
    promo = context.user_data['current_promo']
    discount_percent = promo['discount']
    discount_code = promo['code']
    final_price = int(price_rub * (100 - discount_percent) / 100)

# Используем промокод
db.use_promo_code(discount_code, query.from_user.id)
context.user_data.pop('current_promo', None)

# Сохраняем order_id в контексте
context.user_data['current_order'] = {
    'order_id': order_id,
    'country_code': country_code,
    'country_name': country_name,
    'phone_code': phone_code,
    'price_rub': final_price,
    'original_price': price_rub,
    'discount_percent': discount_percent,
    'account_type': account_type
}

# Создаем заказ в БД
db.create_order(
    order_id,
    query.from_user.id,
    country_code,
    country_name,
    phone_code,
    final_price,
    discount_code,
    discount_percent,
    account_type
)

price_info = format_price(final_price)

order_text = (

```

```

        f" 📄 Детали заказа:\n"
        f" | Страна: {country_name}\n"
        f" | Код страны: {phone_code}\n"
    )

    if account_type == "otl":
        order_text += f" | Отлегой: {account_info['otl']}\n"
        order_text += f" | Наличие: {account_info['stock']} шт.\n"

    order_text += (
        f" | Цена: {final_price}₽\n"
    )

    if discount_percent > 0:
        order_text += f" | Скидка: {discount_percent}%\n"
        order_text += f" | Изначальная цена: {price_rub}₽\n"

    order_text += (
        f" | Цена в USDT/TON: {price_info}\n"
        f" | Номер заказа: `{order_id}`\n\n"
        "Выберите способ оплаты:"
    )

    keyboard = [
        InlineKeyboardButton("💳 Карта", callback_data="pay_card"),
        InlineKeyboardButton("₿ Крипбот", callback_data="pay_crypto")]

    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.message.edit_text(order_text, reply_markup=reply_markup,
    parse_mode='Markdown')

except Exception as e:
    logger.error(f"Ошибка в show_order_details: {e}")
    await query.message.edit_text("Произошла ошибка при создании заказа.")

async def show_payment_card(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Показать реквизиты карты для оплаты"""
    query = update.callback_query
    await query.answer()

    try:
        order_info = context.user_data.get('current_order', {})
        order_id = order_info.get('order_id', 'N/A')
        price_rub = order_info.get('price_rub', 0)

        payment_text = (
            f"💳 Оплата на карту:\n\n"

```

```

        f"Номер: `{CARD_NUMBER}`\n"
        f"Сумма к оплате: *{price_rub}₽* (точно)\n"
        f"Комментарий к переводу: `{order_id}`\n\n"
        f"⚠ Обязательно укажите комментарий, иначе платеж не будет зачислен!\n\n"
        f"После оплаты нажмите кнопку ниже:"
    )

    keyboard = [
        [InlineKeyboardButton("✅ Я оплатил(а)", callback_data=f"paid_{order_id}")]
    ]

    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.message.edit_text(payment_text, reply_markup=reply_markup,
    parse_mode='Markdown')

    # Сохраняем метод оплаты
    context.user_data['payment_method'] = 'card'

except Exception as e:
    logger.error(f"Ошибка в show_payment_card: {e}")
    await query.message.edit_text("Произошла ошибка.")

async def show_payment_crypto(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Показать реквизиты для крипто-оплаты"""
    query = update.callback_query
    await query.answer()

    try:
        order_info = context.user_data.get('current_order', {})
        order_id = order_info.get('order_id', 'N/A')
        price_rub = order_info.get('price_rub', 0)
        price_info = format_price(price_rub)

        payment_text = (
            f"💰 Оплата через криптобота:\n\n"
            f"Перейдите по ссылке для оплаты: {CRYPTO_BOT_LINK}\n"
            f"Сумма: *{price_rub}₽* ({price_info})\n"
            f"Укажите номер заказа: `{order_id}`\n\n"
            f"⚠ Обязательно укажите номер заказа, иначе платеж не будет зачислен!\n\n"
            f"После оплаты нажмите кнопку ниже:"
        )

        keyboard = [
            [InlineKeyboardButton("✅ Я оплатил(а)", callback_data=f"paid_{order_id}")]
        ]

        reply_markup = InlineKeyboardMarkup(keyboard)

```

```

    await query.message.edit_text(payment_text, reply_markup=reply_markup,
    parse_mode='Markdown')

    # Сохраняем метод оплаты
    context.user_data['payment_method'] = 'crypto'

except Exception as e:
    logger.error(f"Ошибка в show_payment_crypto: {e}")
    await query.message.edit_text("Произошла ошибка.")

async def request_screenshot(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Запрос скриншота об оплате"""
    query = update.callback_query
    await query.answer()

    try:
        callback_data = query.data
        order_id = callback_data.replace("paid_", "")

        # Сохраняем order_id в контексте
        context.user_data['waiting_screenshot_for'] = order_id

        await query.message.edit_text(
            "📷 Пожалуйста, отправьте скриншот чека об оплате (фото или документ).\n\n"
            "⚠️ Убедитесь, что на скриншоте видно:\n"
            "• Сумма оплаты\n"
            "• Номер заказа (комментарий)\n"
            "• Дата и время оплаты"
        )

    except Exception as e:
        logger.error(f"Ошибка в request_screenshot: {e}")
        await query.message.edit_text("Произошла ошибка.")

async def handle_screenshot(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обработка скриншота об оплате"""
    try:
        user = update.effective_user
        order_id = context.user_data.get('waiting_screenshot_for')

        if not order_id:
            await update.message.reply_text("Пожалуйста, начните процесс покупки сначала.")
            return

        # Получаем информацию о заказе
        order_info = db.get_order(order_id)
        if not order_info:

```

```

        await update.message.reply_text("Ошибка: заказ не найден.")
        context.user_data.pop('waiting_screenshot_for', None)
        return


# Получаем файл
file = None
file_ext = "jpg"

if update.message.photo:
    file = await update.message.photo[-1].get_file()
    file_ext = "jpg"
elif update.message.document:
    file = await update.message.document.get_file()
    file_ext = update.message.document.file_name.split('.')[-1] if
update.message.document.file_name and '.' in update.message.document.file_name else
"bin"
else:
    await update.message.reply_text("Пожалуйста, отправьте фото или документ.")
    return

# Сохраняем информацию о файле
os.makedirs("screenshots", exist_ok=True)
file_path =
f"screenshots/{order_id}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.{file_ext}"
await file.download_to_drive(file_path)

# Обновляем заказ в БД
payment_method = context.user_data.get('payment_method', 'unknown')
db.update_order_payment(order_id, payment_method, file_path)

# Получаем информацию о заказе
order = db.get_order_by_id(order_id)
if not order:
    await update.message.reply_text("Ошибка: заказ не найден в БД.")
    context.user_data.pop('waiting_screenshot_for', None)
    return

# Отправляем уведомление админу
admin_text = (
    f"

```

```

keyboard = [
    InlineKeyboardButton("✅ Одобрить",
callback_data=f"approve_{order_id}_{user.id}"),
    InlineKeyboardButton("❌ Отклонить", callback_data=f"reject_{order_id}_{user.id}")
]

reply_markup = InlineKeyboardMarkup(keyboard)

# Отправляем сообщение админу с скриншотом
try:
    with open(file_path, 'rb') as photo:
        await context.bot.send_photo(
            chat_id=ADMIN_ID,
            photo=photo,
            caption=admin_text,
            reply_markup=reply_markup
        )
except Exception as e:
    logger.error(f"Ошибка отправки фото админу: {e}")
    # Если не получилось отправить фото, отправляем только текст
    await context.bot.send_message(
        chat_id=ADMIN_ID,
        text=admin_text + f"\n\n📸 Скриншот сохранен: {file_path}",
        reply_markup=reply_markup
    )

await update.message.reply_text(
    "✅ Скриншот получен и отправлен на проверку администратору.\n"
    "Ожидайте подтверждения оплаты."
)

# Очищаем контекст
context.user_data.pop('waiting_screenshot_for', None)
context.user_data.pop('current_order', None)
context.user_data.pop('payment_method', None)

except Exception as e:
    logger.error(f"Ошибка в handle_screenshot: {e}")
    await update.message.reply_text("Произошла ошибка при обработке скриншота.")

async def handle_admin_approval(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    """Обработка одобрения заказа админом"""
    query = update.callback_query
    await query.answer()

try:

```

```

callback_data = query.data
_, order_id, user_id = callback_data.split("_")
user_id = int(user_id)

# Обновляем статус заказа
db.update_order_status(order_id, "completed")

# Уменьшаем количество аккаунтов с отлегой если это такой заказ
order = db.get_order_by_id(order_id)
if order and order['account_type'] == 'otl':
    # Находим аккаунт с отлегой по коду страны
    account = db.get_otl_account_by_code(order['country_code'])
    if account:
        new_stock = max(0, account['stock'] - 1)
        db.update_otl_account_stock(account['id'], new_stock)
    # Обновляем локальный кэш
    load_otl_accounts_from_db()

# Уведомляем админа об успешном одобрении
admin_notification = f"✅ Вы одобрили заказ #{order_id}"
await context.bot.send_message(chat_id=ADMIN_ID, text=admin_notification)

# Отправляем уведомление пользователю с ВТОРОЙ кнопкой
keyboard = [[
    InlineKeyboardButton("📄 Получить номер", callback_data=f"get_num_{order_id}")
]]

reply_markup = InlineKeyboardMarkup(keyboard)

try:
    await context.bot.send_message(
        chat_id=user_id,
        text=f"✅ Ваш платеж по заказу #{order_id} подтвержден! Аккаунт готов к выдаче.\n\n"
        f"Нажмите '📄 Получить номер' чтобы получить данные аккаунта.",
        reply_markup=reply_markup
    )
    await query.message.edit_text(f"✅ Заказ #{order_id} одобрен. Пользователь уведомлен.")
except Exception as e:
    logger.error(f"Ошибка отправки сообщения пользователю: {e}")
    await query.message.edit_text(f"✅ Заказ #{order_id} одобрен, но не удалось уведомить пользователя.")
except Exception as e:
    logger.error(f"Ошибка в handle_admin_approval: {e}")
    await query.message.edit_text("Произошла ошибка при одобрении заказа.")

```

```

async def handle_admin_rejection(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    """Обработка отклонения заказа админом"""
    query = update.callback_query
    await query.answer()

    try:
        callback_data = query.data
        _, order_id, user_id = callback_data.split("_")
        user_id = int(user_id)

        # Обновляем статус заказа
        db.update_order_status(order_id, "rejected")

        # Уведомляем админа об отклонении
        admin_notification = f"❌ Вы отклонили заказ #{order_id}"
        await context.bot.send_message(chat_id=ADMIN_ID, text=admin_notification)

        # Отправляем уведомление пользователю
        try:
            await context.bot.send_message(
                chat_id=user_id,
                text=f"❌ Ваш платеж по заказу #{order_id} отклонен администратором. "
                    f"Свяжитесь с {SUPPORT_USERNAME} для выяснения причин."
            )
            await query.message.edit_text(f"❌ Заказ #{order_id} отклонен. Пользователь
уведомлен.")
        except Exception as e:
            logger.error(f"Ошибка отправки сообщения пользователю: {e}")
            await query.message.edit_text(f"❌ Заказ #{order_id} отклонен, но не удалось
уведомить пользователя.")
        except Exception as e:
            logger.error(f"Ошибка в handle_admin_rejection: {e}")
            await query.message.edit_text("Произошла ошибка при отклонении заказа.")

    async def handle_data_request(update: Update, context: ContextTypes.DEFAULT_TYPE):
        """Обработка запроса данных пользователем"""
        query = update.callback_query
        await query.answer()

        try:
            callback_data = query.data
            data_type = "phone" if "get_num" in callback_data else "code"
            order_id = callback_data.split("_")[-1]

            # Проверяем принадлежность заказа
            if not db.check_order_ownership(order_id, query.from_user.id):
                await query.answer("❌ Этот заказ не принадлежит вам!")

```

```

return

user = query.from_user

# Проверяем статус заказа
order = db.get_order_by_id(order_id)
if not order or order['status'] != 'completed':
    await query.answer("❌ Заказ не подтвержден или не существует!")
    return

# Проверяем, не были ли уже выданы данные
issued_data = db.get_issued_data(order_id, data_type)
if issued_data:
    # Данные уже выданы - отправляем их
    data_text = issued_data[0][0]
    await query.message.edit_text(
        f"📱 Данные для заказа #{order_id}:\n\n"
        f"☎️ Номер телефона' if data_type == 'phone' else '🔑 Код': `{data_text}`\n\n"
        f"💾 Сохраните эти данные в надежном месте!",
        parse_mode="Markdown"
    )

# Если это был номер, добавляем кнопку для получения кода
if data_type == "phone":
    # Проверяем, есть ли уже код
    issued_code = db.get_issued_data(order_id, "code")
    if not issued_code:
        keyboard = [[
            InlineKeyboardButton("🔑 Получить код",
callback_data=f"get_code_{order_id}")
        ]]
        reply_markup = InlineKeyboardMarkup(keyboard)
        await query.message.reply_text(
            "🔑 Теперь вы можете получить код для аккаунта:",
            reply_markup=reply_markup
        )
    return

# Если данных нет, создаем запрос админу
# Сохраняем запрос в БД
db.add_pending_admin_reply(order_id, data_type, user.id)

# Отправляем запрос админу
admin_text = (
    f"📱 Запрос на получение данных\n"
    f"└ Покупатель: @{user.username if user.username else 'без username'}\n"
    f"└ Заказ: #{order_id}\n"

```

```

        f" | Тип: {👉 Аккаунт с отлегой' if order['account_type'] == 'otl' else '📱 ФИЗ
аккаунт'}\n"
        f" | Страна: {order['country_name']}\n"
        f" | Запрошено: {'номер телефона' if data_type == 'phone' else 'код'}"
    )

    keyboard = [[
        InlineKeyboardButton("💬 Ответить",
callback_data=f"admin_reply_{order_id}_{data_type}")
    ]]

    reply_markup = InlineKeyboardMarkup(keyboard)

    await context.bot.send_message(
        chat_id=ADMIN_ID,
        text=admin_text + f"\n\n🕒 Нажмите кнопку 'Ответить' ниже, чтобы отправить
данные.",
        reply_markup=reply_markup
    )
    await query.message.edit_text(
        "🕒 Запрос отправлен администратору. Ожидайте..."
    )
except Exception as e:
    logger.error(f"Ошибка в handle_data_request: {e}")
    await query.answer("Произошла ошибка. Попробуйте еще раз.")

# ===== НОВОГОДНИЕ ПРИЗЫ
=====

async def handle_prize_claim(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обработка получения приза"""
    query = update.callback_query
    await query.answer()

    try:
        user = query.from_user

        # Проверяем, может ли пользователь получить приз
        if not db.can_claim_prize(user.id):
            await query.answer("🕒 Вы уже получали приз сегодня. Приходите через 24
часа!")
            return

        # Генерируем приз
        prize_type = random.choices(
            ["promo", "account"],
            weights=[99, 1] # 99% шанс на промокод, 1% на аккаунт
        )[0]

```

```

if prize_type == "promo":
    # Генерируем промокод
    discount = random.randint(1, 30)
    promo_code = f"NY2026{random.randint(1000, 9999)}"

    # Создаем промокод в БД
    db.create_promo_code(promo_code, discount, ADMIN_ID, 1)

    # Записываем получение приза
    db.claim_prize(user.id, "promo", f"{discount}% ({promo_code})")

    prize_text = (
        f"🎉 Поздравляем! Вы получили:\n\n"
        f"📜 Промокод: `{promo_code}`\n"
        f"💰 Скидка: {discount}%\n\n"
        f"💡 Используйте промокод при следующей покупке!"
    )

else: # Бесплатный аккаунт
    # Выбираем Индонезию как бесплатный аккаунт
    country_code = "indonesia"
    country_info = COUNTRIES[country_code]

    # Создаем заказ для бесплатного аккаунта
    order_id = generate_order_id()
    db.create_order(
        order_id,
        user.id,
        country_code,
        country_info['name'],
        country_info['code'],
        0, # Бесплатно
        "FREE_GIFT",
        100, # 100% скидка
        "fiz"
    )

    # Автоматически подтверждаем заказ
    db.update_order_status(order_id, "completed")

    # Записываем получение приза
    db.claim_prize(user.id, "account", f"Индонезия #{order_id}")

    prize_text = (
        f"🎉 БАУ! Вам выпал ДЖЕКПОТ!\n\n"
        f"📜 Бесплатный ФИЗ аккаунт:\n"
        f"📍 Страна: {country_info['name']}\n"

```

```

        f"└ Номер заказа: `{order_id}`\n"
        f"└ Цена: БЕСПЛАТНО!\n\n"
        f"Нажмите кнопку ниже, чтобы получить данные аккаунта:"
    )

    keyboard = [
        InlineKeyboardButton("📄 Получить номер",
callback_data=f"get_num_{order_id}")
    ]

    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.message.edit_text(prize_text, reply_markup=reply_markup,
parse_mode='Markdown')
    return

# Обновляем клавиатуру
keyboard = create_prize_keyboard(False)
await query.message.edit_text(prize_text, reply_markup=keyboard,
parse_mode='Markdown')

except Exception as e:
    logger.error(f"Ошибка в handle_prize_claim: {e}")
    await query.message.edit_text("Произошла ошибка при получении приза.")

async def show_prize_history(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Показать историю призов"""
    query = update.callback_query
    await query.answer()

    try:
        user = query.from_user
        prizes = db.get_user_prizes(user.id, 10)

        history_text = "📖 История ваших призов:\n\n"

        if prizes:
            for prize in prizes:
                prize_type, prize_value, claimed_at = prize
                date_str = claimed_at[:10]

                if prize_type == "promo":
                    history_text += f"• {date_str} | 📄 Промокод {prize_value}\n"
                elif prize_type == "account":
                    history_text += f"• {date_str} | 🎁 Бесплатный аккаунт {prize_value}\n"
            else:
                history_text += "🇷🇺 У вас еще нет призов\n"

        # Проверяем, может ли пользователь получить приз сейчас

```

```

can_claim = db.can_claim_prize(user.id)

keyboard = create_prize_keyboard(can_claim)
await query.message.edit_text(history_text, reply_markup=keyboard)

except Exception as e:
    logger.error(f"Ошибка в show_prize_history: {e}")
    await query.message.edit_text("Произошла ошибка при загрузке истории.")

# ===== АДМИН ПАНЕЛЬ
=====

async def admin_callback_handler(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    """Обработчик callback админ-панели"""
    query = update.callback_query
    await query.answer()

    user = query.from_user
    if user.id != ADMIN_ID:
        await query.message.edit_text("❌ У вас нет доступа к админ-панели.")
        return

    data = query.data

    try:
        if data == "admin_stats":
            await show_admin_stats(update, context)
        elif data == "admin_stats_refresh":
            await show_admin_stats(update, context, refresh=True)
        elif data == "admin_broadcast":
            await start_broadcast(update, context)
        elif data == "admin_prices":
            await show_price_menu(update, context)
        elif data == "admin_promos":
            await show_promo_admin_menu(update, context)
        elif data == "admin_otl":
            await show_otl_admin_menu(update, context)
        elif data.startswith("admin_change_price_"):
            country_code = data.replace("admin_change_price_", "")
            await start_price_change(update, context, country_code)
        elif data == "admin_create_promo":
            await start_promo_create(update, context)
        elif data == "admin_list_promos":
            await show_promo_list(update, context)
        elif data.startswith("admin_otl_edit_"):
            account_id = int(data.replace("admin_otl_edit_", ""))
            await edit_otl_account(update, context, account_id)

```

```

elif data == "admin_otl_add":
    await start_otl_add_country(update, context)
elif data.startswith("otl_country_"):
    country_code = data.replace("otl_country_", "")
    await process_otl_country_selection(update, context, country_code)
elif data == "admin_cancel_broadcast":
    await cancel_broadcast(update, context)
elif data == "admin_back_to_main":
    await admin_command(update, context)
elif data == "admin_back":
    await query.message.delete()
    await admin_command(update, context)
elif data.startswith("admin_reply_"):
    await handle_admin_reply_request(query, context)

except Exception as e:
    logger.error(f"Ошибка в admin_callback_handler: {e}")
    await query.message.edit_text("Произошла ошибка. Попробуйте еще раз.")

```

```

async def show_admin_stats(update: Update, context: ContextTypes.DEFAULT_TYPE,
refresh=False):

```

```

    """Показать статистику админа"""

```

```

    query = update.callback_query if hasattr(update, 'callback_query') else None

```

```

    stats = db.get_statistics()

```

```

    stats_text = (

```

```

        "📊 Статистика бота\n\n"
        f"👤 Пользователи\n"
        f"├ Всего: {stats['total_users']}\n"
        f"└ Новые (24ч): {stats['new_users_24h']}\n\n"
        f"🛒 Заказы\n"
        f"├ Всего: {stats['total_orders']}\n"
        f"└ Новые (24ч): {stats['new_orders_24h']}\n\n"
        f"💰 Финансы\n"
        f"├ Общая выручка: {stats['total_revenue']}₽\n"
        f"└ Выручка (24ч): {stats['revenue_24h']}₽\n\n"
        f"🎟 Промокоды\n"
        f"├ Всего: {stats['total_promo_codes']}\n"
        f"└ Активных: {stats['active_promo_codes']}\n\n"
        f"🏠 Аккаунты с отлегой\n"
        f"├ Всего: {stats['total_otl_accounts']}\n"
        f"├ В наличии: {stats['available_otl_accounts']}\n"
        f"└ Заказов: {stats['otl_orders']}\n\n"
        f"🎁 Призы\n"
        f"└ Выдано: {stats['total_prizes']}\n\n"
        f"📊 Статусы заказов\n"

```

```

    )

```

```

for status, count in stats['status_stats'].items():
    status_icon = "✅" if status == "completed" else "⌚" if status == "waiting_approval"
else "❌"
    stats_text += f"{status_icon} {status}: {count}\n"

if query:
    if refresh:
        await query.message.edit_text(stats_text, reply_markup=create_stats_keyboard())
    else:
        await query.message.reply_text(stats_text, reply_markup=create_stats_keyboard())
else:
    await update.message.reply_text(stats_text, reply_markup=create_stats_keyboard())

```

```

async def start_broadcast(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Начать процесс рассылки"""
    query = update.callback_query

```

```

    context.user_data['admin_state'] = WAITING_BROADCAST

```

```

    await query.message.edit_text(
        "🔔 Рассылка сообщения\n\n"
        "Отправьте сообщение, которое хотите разослать всем пользователям.\n"
        "Вы можете использовать разметку Markdown.\n\n"
        "❌ Для отмены нажмите кнопку ниже.",
        reply_markup=create_broadcast_keyboard(),
        parse_mode='Markdown'
    )

```

```

async def process_broadcast(update: Update, context: ContextTypes.DEFAULT_TYPE,
message_text: str):
    """Обработать рассылку"""
    user = update.effective_user

```

```

    if user.id != ADMIN_ID:
        return

```

```

    await update.message.reply_text("🔔 Начинаю рассылку...")

```

```

    users = db.get_all_users()
    successful = 0
    failed = 0

```

```

    for user_id in users:
        try:
            await context.bot.send_message(
                chat_id=user_id,
                text=f"🔔 Сообщение от администратора:\n\n{message_text}",

```

```

        parse_mode='Markdown'
    )
    successful += 1
    await asyncio.sleep(0.1) # Чтобы не превысить лимиты Telegram
except Exception as e:
    logger.error(f"Ошибка отправки пользователю {user_id}: {e}")
    failed += 1

# Сбрасываем состояние
context.user_data.pop('admin_state', None)

await update.message.reply_text(
    f"📧 Рассылка завершена\n\n"
    f"📊 Результаты:\n"
    f"✅ Успешно: {successful}\n"
    f"❌ Не удалось: {failed}\n\n"
    f"Вернуться в админ-панель: /admin"
)

async def cancel_broadcast(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Отменить рассылку"""
    query = update.callback_query

    context.user_data.pop('admin_state', None)

    await query.message.edit_text(
        "❌ Рассылка отменена.\n\n"
        "Вернуться в админ-панель: /admin"
    )

async def show_price_menu(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Показать меню изменения цен"""
    query = update.callback_query

    price_text = "💰 Изменение цены\nВыберите страну для изменения цены:"

    await query.message.edit_text(price_text,
    reply_markup=create_price_change_keyboard())

async def start_price_change(update: Update, context: ContextTypes.DEFAULT_TYPE,
country_code: str):
    """Начать изменение цены страны"""
    query = update.callback_query

    if country_code not in COUNTRIES:
        await query.message.edit_text("❌ Страна не найдена.")
    return

```

```
country_info = COUNTRIES[country_code]
```

```
context.user_data['admin_state'] = WAITING_PRICE_VALUE
```

```
context.user_data['price_change_country'] = country_code
```

```
await query.message.edit_text(
```

```
    f"💰 Изменение цены для {country_info['name']}\n\n"
```

```
    f"Текущая цена: {country_info['price_rub']}₽\n\n"
```

```
    f"Введите новую цену в рублях."
```

```
)
```

```
async def process_price_change(update: Update, context: ContextTypes.DEFAULT_TYPE,  
price_text: str):
```

```
    """Обработать изменение цены"""
```

```
    user = update.effective_user
```

```
    if user.id != ADMIN_ID:
```

```
        return
```

```
    try:
```

```
        new_price = int(price_text)
```

```
        if new_price <= 0:
```

```
            await update.message.reply_text("❌ Цена должна быть больше 0.")
```

```
            return
```

```
        country_code = context.user_data.get('price_change_country')
```

```
        if not country_code or country_code not in COUNTRIES:
```

```
            await update.message.reply_text("❌ Ошибка: страна не найдена.")
```

```
            return
```

```
        # Обновляем цену
```

```
        db.update_country_price(country_code, new_price)
```

```
        country_name = COUNTRIES[country_code]['name']
```

```
        # Сбрасываем состояние
```

```
        context.user_data.pop('admin_state', None)
```

```
        context.user_data.pop('price_change_country', None)
```

```
        await update.message.reply_text(
```

```
            f"✅ Цена для {country_name} изменена на {new_price}₽\n\n"
```

```
            f"Вернуться в админ-панель: /admin"
```

```
)
```

```
except ValueError:
```

```
await update.message.reply_text("❌ Пожалуйста, введите целое число (например: 100).")
```

```
async def show_promo_admin_menu(update: Update, context: ContextTypes.DEFAULT_TYPE):
```

```
    """Показать меню управления промокодами"""
```

```
    query = update.callback_query
```

```
    promo_text = "📋 Управление промокодами\n\nВыберите действие:"
```

```
    await query.message.edit_text(promo_text,  
reply_markup=create_promo_admin_keyboard())
```

```
async def start_promo_create(update: Update, context: ContextTypes.DEFAULT_TYPE):
```

```
    """Начать создание промокода"""
```

```
    query = update.callback_query
```

```
    context.user_data['admin_state'] = WAITING_PROMO_CREATE
```

```
    await query.message.edit_text(
```

```
        "📋 Создание промокода\n\n"
```

```
        "Введите данные промокода в формате:\n"
```

```
        "`КОД ПРОЦЕНТ_СКИДКИ МАКСИМАЛЬНОЕ_КОЛИЧЕСТВО`\n\n"
```

```
        "Пример: `SUMMER2025 15 10`\n"
```

```
        "Создаст промокод SUMMER2025 на 15% скидку для 10 человек.\n\n"
```

```
        "Для отмены нажмите /admin"
```

```
    )
```

```
async def process_promo_create(update: Update, context: ContextTypes.DEFAULT_TYPE,  
text: str):
```

```
    """Обработать создание промокода"""
```

```
    user = update.effective_user
```

```
    if user.id != ADMIN_ID:
```

```
        return
```

```
    try:
```

```
        parts = text.split()
```

```
        if len(parts) < 2:
```

```
            await update.message.reply_text(
```

```
                "❌ Неверный формат. Используйте: КОД ПРОЦЕНТ  
[МАКСИМАЛЬНОЕ_КОЛИЧЕСТВО]\n"
```

```
                "Максимальное количество по умолчанию: 1"
```

```
            )
```

```
            return
```

```
        code = parts[0].upper()
```

```
        discount = int(parts[1])
```

```

max_uses = int(parts[2]) if len(parts) > 2 else 1

if discount <= 0 or discount > 100:
    await update.message.reply_text("❌ Процент скидки должен быть от 1 до 100.")
    return

if max_uses <= 0:
    await update.message.reply_text("❌ Максимальное количество использований  
должно быть больше 0.")
    return

# Создаем промокод
db.create_promo_code(code, discount, user.id, max_uses)

# Сбрасываем состояние
context.user_data.pop('admin_state', None)

await update.message.reply_text(
    f"✅ Промокод создан!\n\n"
    f"📄 Код: `{code}`\n"
    f"💰 Скидка: {discount}%\n"
    f"👤 Макс. использований: {max_uses}\n\n"
    f"🔙 Вернуться в админ-панель: /admin"
)

except ValueError:
    await update.message.reply_text("❌ Неверный формат чисел. Используйте целые  
числа.")
except Exception as e:
    logger.error(f"Ошибка создания промокода: {e}")
    await update.message.reply_text(f"❌ Ошибка создания промокода: {e}")

async def show_promo_list(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Показать список промокодов"""
    query = update.callback_query

    promos = db.get_all_promo_codes()

    if not promos:
        await query.message.edit_text(
            "📄 Список промокодов пуст.\n\n"
            "Вернуться в меню промокодов: /admin",
            reply_markup=create_promo_admin_keyboard()
        )
        return

    promo_text = "📄 Список промокодов:\n\n"

```

for promo in promos:

status = "✅ Активен" if promo['is_active'] else "❌ Неактивен"

used_by = f"@{promo['used_by']}" if promo['used_by'] else "Не использован"

promo_text += (

f"📄 `{promo['code']}`\n"

f"└ Скидка: {promo['discount_percent']}%\n"

f"└ Статус: {status}\n"

f"└ Использовано: {promo['use_count']}/{promo['max_uses']}\n"

f"└ Создан: {promo['created_at'][:10]}\n\n"

)

await query.message.edit_text(

promo_text,

reply_markup=create_promo_admin_keyboard(),

parse_mode='Markdown'

)

async def show_otl_admin_menu(update: Update, context: ContextTypes.DEFAULT_TYPE):

"""Показать меню управления аккаунтами с отлегой"""

query = update.callback_query

otl_text = "👉 Управление аккаунтами с отлегой\n\nВыберите аккаунт для редактирования:"

await query.message.edit_text(otl_text, reply_markup=create_otl_admin_keyboard())

===== ПОШАГОВОЕ ДОБАВЛЕНИЕ
АККАУНТА С ОТЛЕГОЙ =====

async def start_otl_add_country(update: Update, context: ContextTypes.DEFAULT_TYPE):

"""Начать добавление аккаунта с отлегой - выбор страны"""

query = update.callback_query

context.user_data['admin_state'] = WAITING_OTL_COUNTRY

context.user_data['otl_account_data'] = {}

await query.message.edit_text(

"👉 Добавление аккаунта с отлегой\n\n"

"Шаг 1: Выберите страну из списка ниже:",

reply_markup=create_otl_country_keyboard()

)

async def process_otl_country_selection(update: Update, context:

ContextTypes.DEFAULT_TYPE, country_code: str):

"""Обработка выбора страны для отлегой"""

query = update.callback_query

await query.answer()

```

if country_code not in COUNTRIES:
    await query.message.edit_text("❌ Ошибка: страна не найдена.")
    return

country_info = COUNTRIES[country_code]

# Сохраняем данные страны
context.user_data['otl_account_data']['country_code'] = country_code
context.user_data['otl_account_data']['country_name'] = country_info['name']
context.user_data['otl_account_data']['phone_code'] = country_info['code']

# Переходим к следующему шагу
context.user_data['admin_state'] = WAITING_OTL_NAME

await query.message.edit_text(
    f"👉 Добавление аккаунта с отлегой\n\n"
    f"Шаг 1: Страна: {country_info['name']} ✅\n"
    f"Шаг 2: Введите название отлегой:\n\n"
    f"Пример: `Отлегой #USA-2025` или `Premium аккаунт США`"
)

async def process_otl_country(update: Update, context: ContextTypes.DEFAULT_TYPE,
text: str):
    """Обработка ввода страны (если через текст)"""
    user = update.effective_user

    if user.id != ADMIN_ID:
        return

    # Проверяем, есть ли страна в списке
    country_found = None
    for code, info in COUNTRIES.items():
        if text.lower() in info['name'].lower():
            country_found = (code, info)
            break

    if not country_found:
        await update.message.reply_text(
            "❌ Страна не найдена. Пожалуйста, введите корректное название страны или используйте кнопки."
        )
        return

    country_code, country_info = country_found

    # Сохраняем данные страны
    context.user_data['otl_account_data']['country_code'] = country_code
    context.user_data['otl_account_data']['country_name'] = country_info['name']

```

```
context.user_data['otl_account_data']['phone_code'] = country_info['code']
```

```
# Переходим к следующему шагу
```

```
context.user_data['admin_state'] = WAITING_OTL_NAME
```

```
await update.message.reply_text(
```

```
    f"👉 Добавление аккаунта с отлегой\n\n"
```

```
    f"Шаг 1: Страна: {country_info['name']} ✅\n"
```

```
    f"Шаг 2: Введите название отлегой:\n\n"
```

```
    f"Пример: `Отлегой #USA-2025` или `Premium аккаунт США`"
```

```
)
```

```
async def process_otl_name(update: Update, context: ContextTypes.DEFAULT_TYPE, text: str):
```

```
    """Обработка ввода названия отлегой"""
```

```
    user = update.effective_user
```

```
    if user.id != ADMIN_ID:
```

```
        return
```

```
    if not text or len(text) < 2:
```

```
        await update.message.reply_text("❌ Название отлегой должно содержать минимум 2 символа.")
```

```
        return
```

```
# Сохраняем название отлегой
```

```
context.user_data['otl_account_data']['otl_name'] = text
```

```
# Переходим к следующему шагу
```

```
context.user_data['admin_state'] = WAITING_OTL_PRICE
```

```
country_name = context.user_data['otl_account_data']['country_name']
```

```
await update.message.reply_text(
```

```
    f"👉 Добавление аккаунта с отлегой\n\n"
```

```
    f"Шаг 1: Страна: {country_name} ✅\n"
```

```
    f"Шаг 2: Отлегой: {text} ✅\n"
```

```
    f"Шаг 3: Введите цену в рублях:\n\n"
```

```
    f"Пример: `150` (для 150 рублей)"
```

```
)
```

```
async def process_otl_price(update: Update, context: ContextTypes.DEFAULT_TYPE, text: str):
```

```
    """Обработка ввода цены"""
```

```
    user = update.effective_user
```

```
    if user.id != ADMIN_ID:
```

```
        return
```

```

try:
    price = int(text)

    if price <= 0:
        await update.message.reply_text("❌ Цена должна быть больше 0.")
        return

    # Сохраняем цену
    context.user_data['otl_account_data']['price_rub'] = price

    # Переходим к следующему шагу
    context.user_data['admin_state'] = WAITING_OTL_STOCK

    country_name = context.user_data['otl_account_data']['country_name']
    otl_name = context.user_data['otl_account_data']['otl_name']

    await update.message.reply_text(
        f"👉 Добавление аккаунта с отлегой\n\n"
        f"Шаг 1: Страна: {country_name} ✅\n"
        f"Шаг 2: Отлегой: {otl_name} ✅\n"
        f"Шаг 3: Цена: {price}₽ ✅\n"
        f"Шаг 4: Введите количество (наличие):\n\n"
        f"Пример: `5` (для 5 аккаунтов в наличии)"
    )

except ValueError:
    await update.message.reply_text("❌ Пожалуйста, введите целое число для цены.")

async def process_otl_stock(update: Update, context: ContextTypes.DEFAULT_TYPE, text: str):
    """Обработка ввода количества"""
    user = update.effective_user

    if user.id != ADMIN_ID:
        return

    try:
        stock = int(text)

        if stock < 0:
            await update.message.reply_text("❌ Количество не может быть отрицательным.")
            return

        # Получаем все данные
        account_data = context.user_data['otl_account_data']

```

```

# Создаем аккаунт в БД
account_id = db.create_otl_account(
    account_data['country_code'],
    account_data['country_name'],
    account_data['otl_name'],
    account_data['phone_code'],
    account_data['price_rub'],
    stock
)

# Обновляем локальный кэш
load_otl_accounts_from_db()

# Сбрасываем состояние
context.user_data.pop('admin_state', None)
context.user_data.pop('otl_account_data', None)

await update.message.reply_text(
    f"✅ Аккаунт с отлегой успешно добавлен!\n\n"
    f"📋 Информация об аккаунте:\n"
    f"└ Страна: {account_data['country_name']}\n"
    f"└ Отлегой: {account_data['otl_name']}\n"
    f"└ Телефонный код: {account_data['phone_code']}\n"
    f"└ Цена: {account_data['price_rub']}₽\n"
    f"└ Наличие: {stock} шт.\n\n"
    f"Вернуться в админ-панель: /admin"
)

except ValueError:
    await update.message.reply_text("❌ Пожалуйста, введите целое число для количества.")

async def edit_otl_account(update: Update, context: ContextTypes.DEFAULT_TYPE,
account_id: int):
    """"Редактирование аккаунта с отлегой""""
    query = update.callback_query

    account = db.get_otl_account(account_id)
    if not account:
        await query.message.edit_text("❌ Аккаунт не найден.")
        return

    edit_text = (
        f"🔧 Редактирование аккаунта с отлегой\n\n"
        f"📋 Текущая информация:\n"
        f"└ Страна: {account['country_name']}\n"
        f"└ Отлегой: {account['otl_name']}\n"
        f"└ Цена: {account['price_rub']}₽\n"

```

```

f"└─ Наличие: {account['stock']} шт.\n"
f"└─ Телефонный код: {account['phone_code']}\n\n"
f"Введите новые данные в формате:\n"
f"ЦЕНА НАЛИЧИЕ`\n\n"
f"Пример: `250 5`\n"
f"Установит цену 250₽ и наличие 5 шт.\n\n"
f"Для отмены нажмите /admin"
)

```

```

context.user_data['admin_state'] = WAITING_OTL_PRICE
context.user_data['otl_edit_id'] = account_id

```

```

await query.message.edit_text(edit_text)

```

```

# ===== ОБРАБОТКА ОТВЕТОВ АДМИНА
=====

```

```

async def handle_admin_reply_request(query, context: ContextTypes.DEFAULT_TYPE):

```

```

    """Обработка запроса админа на ответ"""

```

```

    data = query.data

```

```

    parts = data.split("_")

```

```

    if len(parts) >= 4:

```

```

        order_id = parts[2]

```

```

        data_type = parts[3]

```

```

    # Сохраняем состояние ожидания ответа

```

```

    context.user_data['admin_state'] = WAITING_ADMIN_REPLY

```

```

    context.user_data['admin_reply_order'] = order_id

```

```

    context.user_data['admin_reply_type'] = data_type

```

```

    await query.message.edit_text(

```

```

        f"💬 Введите {'номер телефона' if data_type == 'phone' else 'код'} для заказа

```

```

#{order_id}:\n\n"

```

```

        f"✍ Отправьте текстовое сообщение с данными."

```

```

    )

```

```

async def process_admin_reply(update: Update, context: ContextTypes.DEFAULT_TYPE,
text: str):

```

```

    """Обработать ответ админа"""

```

```

    user = update.effective_user

```

```

    if user.id != ADMIN_ID:

```

```

        return

```

```

    order_id = context.user_data.get('admin_reply_order')

```

```

    data_type = context.user_data.get('admin_reply_type')

```

```

if not order_id or not data_type:
    await update.message.reply_text("❌ Ошибка: данные запроса не найдены.")
    return

# Ищем пользователя в БД
user_id = db.get_pending_admin_reply(order_id, data_type)

if not user_id:
    await update.message.reply_text("❌ Ошибка: запрос не найден или устарел.")
    # Сбрасываем состояние
    context.user_data.pop('admin_state', None)
    context.user_data.pop('admin_reply_order', None)
    context.user_data.pop('admin_reply_type', None)
    return

try:
    # Отправляем данные пользователю
    await context.bot.send_message(
        chat_id=user_id,
        text=f"📱 Данные для заказа #{order_id}:\n\n"
            f"{'📞 Номер телефона' if data_type == 'phone' else '🔑 Код': `{{text}}`\n\n"
            f"💾 Сохраните эти данные в надежном месте!",
        parse_mode='Markdown'
    )

    # Сохраняем в БД
    db.add_issued_data(order_id, data_type, text)

    # Удаляем ожидающий запрос
    db.remove_pending_admin_reply(order_id, data_type)

    # Уведомляем админа
    await update.message.reply_text(
        f"✅ Данные отправлены пользователю для заказа #{order_id}"
    )

    # Если это был номер, уведомляем админа, что нужно отправить код
    if data_type == "phone":
        await update.message.reply_text(
            f"💡 Теперь пользователь может запросить код для этого заказа."
        )

    # Сбрасываем состояние
    context.user_data.pop('admin_state', None)
    context.user_data.pop('admin_reply_order', None)
    context.user_data.pop('admin_reply_type', None)

except Exception as e:

```

```

logger.error(f"Ошибка отправки данных пользователю: {e}")
await update.message.reply_text(
    f"❌ Ошибка отправки данных: {e}"
)

```

--- ОБЩИЙ CALLBACK ОБРАБОТЧИК ---

```

async def callback_handler(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обработчик callback запросов"""
    query = update.callback_query
    data = query.data

    try:
        # Админские callback
        if data.startswith("admin_"):
            await admin_callback_handler(update, context)
            return

        # Новогодние призы
        if data == "claim_prize":
            await handle_prize_claim(update, context)
            return
        elif data == "prize_history":
            await show_prize_history(update, context)
            return
        elif data == "back_to_main":
            await query.message.edit_text(
                "Главное меню:",
                reply_markup=create_admin_keyboard() if query.from_user.id == ADMIN_ID else
                create_main_keyboard()
            )
            return

        # Обычные callback
        if data == "type_fiz":
            await show_countries(update, context)
        elif data == "type_otl":
            await show_otl_countries(update, context)
        elif data == "back_to_types":
            await query.message.edit_text(
                "Выберите тип аккаунта:",
                reply_markup=create_account_types_keyboard()
            )
        elif data.startswith("page_"):
            await handle_country_page(update, context)
        elif data.startswith("country_"):
            country_code = data.replace("country_", "")
            await show_order_details(update, context, country_code)

```

```

elif data.startswith("otl_country_"):
    country_code = data.replace("otl_country_", "")
    await process_otl_country_selection(update, context, country_code)
elif data == "pay_card":
    await show_payment_card(update, context)
elif data == "pay_crypto":
    await show_payment_crypto(update, context)
elif data.startswith("paid_"):
    await request_screenshot(update, context)
elif data.startswith("approve_"):
    await handle_admin_approval(update, context)
elif data.startswith("reject_"):
    await handle_admin_rejection(update, context)
elif data.startswith("get_num_") or data.startswith("get_code_"):
    await handle_data_request(update, context)
else:
    await query.answer("Неизвестная команда")
except Exception as e:
    logger.error(f"Ошибка в callback_handler: {e}")
try:
    await query.answer("Произошла ошибка. Пожалуйста, попробуйте снова.")
except:
    pass

```

--- ОСНОВНАЯ ФУНКЦИЯ ---

```

async def set_bot_commands(application: Application):
    """Установка команд бота"""
    commands = [
        BotCommand("start", "Запустить бота"),
        BotCommand("admin", "Админ-панель (только для админа)")
    ]
    await application.bot.set_my_commands(commands)

def main():
    """Основная функция запуска бота"""
    # Создаем папку для скриншотов
    os.makedirs("screenshots", exist_ok=True)

    # Создаем приложение
    application = Application.builder().token(BOT_TOKEN).build()

    # Устанавливаем команды бота
    application.post_init = set_bot_commands

    # Регистрируем обработчики в правильном порядке
    application.add_handler(CommandHandler("start", start_command))
    application.add_handler(CommandHandler("admin", admin_command))

```

```
application.add_handler(CallbackQueryHandler(callback_handler))
```

```
# Обработчик медиафайлов (скриншотов)
```

```
application.add_handler(MessageHandler(  
    filters.PHOTO | filters.Document.ALL,  
    handle_screenshot  
))
```

```
# Общий обработчик текстовых сообщений для всех пользователей
```

```
application.add_handler(MessageHandler(  
    filters.TEXT & ~filters.COMMAND,  
    handle_text_message  
))
```

```
# Запуск бота
```

```
print("=" * 50)  
print("🤖 БОТ ЗАПУЩЕН")  
print("=" * 50)  
print(f"👑 Администратор: {ADMIN_ID}")  
print(f"🆘 Поддержка: {SUPPORT_USERNAME}")  
print(f"📺 Наш канал: {CHANNEL_LINK}")  
print(f"🔒 Политика конфиденциальности: {PRIVACY_POLICY_LINK}")  
print(f"💰 Ссылка для оплаты криптоботом: {CRYPTO_BOT_LINK}")  
print("=" * 50)  
print(f"🌍 Доступно стран: {len(COUNTRIES)}")
```

```
# Загружаем аккаунты с отлегой
```

```
load_otl_accounts_from_db()  
print(f"👉 Аккаунтов с отлегой: {len(ACCOUNTS_WITH_OTL)}")
```

```
print("=" * 50)
```

```
print("📊 Статистика цен:")
```

```
min_country = min(COUNTRIES.items(), key=lambda x: x[1]['price_rub'])
```

```
max_country = max(COUNTRIES.items(), key=lambda x: x[1]['price_rub'])
```

```
print(f"💰 Самая дешевая: {min_country[1]['name']} - {min_country[1]['price_rub']}₽")
```

```
print(f"💎 Самая дорогая: {max_country[1]['name']} - {max_country[1]['price_rub']}₽")
```

```
print("=" * 50)
```

```
print(f"👉 Аккаунты с отлегой в наличии:")
```

```
accounts = db.get_all_otl_accounts()
```

```
if accounts:
```

```
    for account in accounts:
```

```
        stock_status = "✅" if account['stock'] > 0 else "❌"
```

```
        print(f"    {account['country_name']} - {account['price_rub']}₽ {stock_status}")
```

```
    ({account['stock']} шт.)")
```

```
else:
```

```
    print("    Нет аккаунтов с отлегой")
```

```
print("=" * 50)
```

```
print("🎄 Новогодние призы включены!")
```

```
print("📜 Система промокодов активирована!")
print("👉 Пошаговое добавление аккаунтов с отлегой настроено!")
print("ℹ️ Раздел 'О нас' с ссылками добавлен!")
print("💵 Исправлена ссылка для оплаты криптоботом!")
print("=" * 50)
print("✅ Бот готов к работе!")
print("✅ Команда /start должна работать")
print("✅ Команда /admin для админа")
print("✅ Главное меню доступно")
print("✅ База данных создана")
print("=" * 50)
```

```
try:
    application.run_polling(allowed_updates=Update.ALL_TYPES,
drop_pending_updates=True)
except Exception as e:
    logger.error(f"Ошибка запуска бота: {e}")
    print(f"❌ Ошибка запуска: {e}")
```

```
if __name__ == "__main__":
    main()
```