

```

"""
Telegram Auto-Report Bot
Supports multi-account reporting, token system, and Railway deployment
"""

import asyncio
import logging
import os
from aiogram import Bot, Dispatcher, types, F
from aiogram.filters import Command, CommandStart
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
from aiogram.fsm.storage.memory import MemoryStorage
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton, ReplyKeyboa
from aiogram.utils.keyboard import InlineKeyboardBuilder, ReplyKeyboardBuilder
from database import Database
from account_manager import AccountManager
from report_engine import ReportEngine
from config import Config

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Initialize
config = Config()
db = Database(config.DATABASE_URL)
account_manager = AccountManager(db)
report_engine = ReportEngine(db, account_manager)

bot = Bot(token=config.BOT_TOKEN)
storage = MemoryStorage()
dp = Dispatcher(storage=storage)

# _____
# FSM States
# _____

class AddAccount(StatesGroup):
    waiting_phone = State()
    waiting_code = State()
    waiting_password = State()

class ReportTarget(StatesGroup):

```

```

waiting_target = State()
waiting_report_type = State()
waiting_custom_text = State()
waiting_count = State()

class AdminActions(StatesGroup):
    waiting_user_id = State()
    waiting_token_amount = State()
    waiting_custom_report_text = State()

# _____
# Keyboards
# _____

def main_menu(user_id: int) -> InlineKeyboardMarkup:
    user = db.get_user(user_id)
    is_owner = user and user['role'] == 'owner'
    is_admin = user and user['role'] in ('admin', 'owner')

    builder = InlineKeyboardBuilder()
    builder.button(text="📝 Report", callback_data="menu_report")
    builder.button(text="💰 Buy Tokens", callback_data="menu_buy_tokens")
    builder.button(text="📱 Accounts", callback_data="menu_accounts")
    builder.button(text="📊 My Reports", callback_data="menu_my_reports")
    builder.button(text="ℹ️ Help", callback_data="menu_help")
    builder.button(text="📞 Contact", callback_data="menu_contact")

    if is_admin:
        builder.button(text="👑 Admin Panel", callback_data="menu_admin")
    if is_owner:
        builder.button(text="🔱 Owner Panel", callback_data="menu_owner")
    builder.adjust(2)
    return builder.as_markup()

def report_type_keyboard() -> InlineKeyboardMarkup:
    builder = InlineKeyboardBuilder()
    types_list = [
        ("🚫 Illegal Adult Content", "illegal_adult"),
        ("🚫 Pornography", "pornography"),
        ("⚠️ Child Abuse", "child_abuse"),
        ("⚔️ Violence", "violence"),
        ("💊 Illegal Goods/Services", "illegal_goods"),
        ("🔞 Non-Consensual Imagery", "non_consensual"),
        ("💳 Scam or Fraud", "scam_fraud"),
        ("(SPAM) Spam", "spam"),
        ("🐾 Animal Abuse", "animal_abuse"),
    ]

```

```

        ("🔒 Personal Data", "personal_data"),
        ("⌚ Copyright", "copyright"),
        ("❓ Other", "other"),
        ("✍️ Custom Text", "custom"),
    ]
    for label, data in types_list:
        builder.button(text=label, callback_data=f"rtype_{data}")
builder.adjust(2)
return builder.as_markup()

def report_count_keyboard() -> InlineKeyboardMarkup:
    builder = InlineKeyboardBuilder()
    for count in [1, 5, 10, 25, 50]:
        builder.button(text=f"×{count}", callback_data=f"rcount_{count}")
    builder.button(text="Custom", callback_data="rcount_custom")
    builder.adjust(3)
    return builder.as_markup()

def admin_keyboard() -> InlineKeyboardMarkup:
    builder = InlineKeyboardBuilder()
    builder.button(text="👤 All Users", callback_data="admin_users")
    builder.button(text="➕ Add Tokens", callback_data="admin_add_tokens")
    builder.button(text="📱 All Accounts", callback_data="admin_accounts")
    builder.button(text="📊 Stats", callback_data="admin_stats")
    builder.button(text="✉️ Set Report Text", callback_data="admin_set_report_text")
    builder.button(text="⬅️ Back", callback_data="menu_back")
    builder.adjust(2)
    return builder.as_markup()

def owner_keyboard() -> InlineKeyboardMarkup:
    builder = InlineKeyboardBuilder()
    builder.button(text="👑 Promote to Admin", callback_data="owner_promote")
    builder.button(text="🔨 Demote Admin", callback_data="owner_demote")
    builder.button(text="🚫 Ban User", callback_data="owner_ban")
    builder.button(text="✅ Unban User", callback_data="owner_unban")
    builder.button(text="♾ Unlimited Tokens", callback_data="owner_unlimited")
    builder.button(text="🔊 Broadcast", callback_data="owner_broadcast")
    builder.button(text="⬅️ Back", callback_data="menu_back")
    builder.adjust(2)
    return builder.as_markup()

# -----
# START

```

```

# _____

@dp.message(CommandStart())
async def cmd_start(message: types.Message):
    user_id = message.from_user.id
    username = message.from_user.username or ""
    full_name = message.from_user.full_name or ""

    # Register user
    db.register_user(user_id, username, full_name)
    user = db.get_user(user_id)

    tokens = "∞" if user['role'] == 'owner' else user['tokens']
    role_emoji = {"owner": "🔱", "admin": "👑", "user": "👤"}.get(user['role'], '👤')

    text = (
        f"👋 <b>Welcome {full_name}!</b>\n\n"
        f"🆔 User ID: <code>{user_id}</code>\n"
        f"💰 Tokens: <b>{tokens}</b>\n"
        f"📊 Reports Made: <b>{user['reports_made']}</b>\n"
        f"{role_emoji} Role: <b>{user['role'].upper()}</b>\n\n"
        f"Select an option below:"
    )
    await message.answer(text, reply_markup=main_menu(user_id), parse_mode="HTML")

```

```

# _____
# MENU CALLBACKS
# _____

@dp.callback_query(F.data == "menu_back")
async def menu_back(cb: types.CallbackQuery):
    user_id = cb.from_user.id
    user = db.get_user(user_id)
    tokens = "∞" if user['role'] == 'owner' else user['tokens']
    role_emoji = {"owner": "🔱", "admin": "👑", "user": "👤"}.get(user['role'], '👤')
    text = (
        f"👋 <b>Welcome back!</b>\n\n"
        f"💰 Tokens: <b>{tokens}</b>\n"
        f"📊 Reports Made: <b>{user['reports_made']}</b>\n"
        f"{role_emoji} Role: <b>{user['role'].upper()}</b>\n\n"
        f"Select an option below:"
    )
    await cb.message.edit_text(text, reply_markup=main_menu(user_id), parse_mode="HTML")
    await cb.answer()

```

```

@dp.callback_query(F.data == "menu_report")
async def menu_report(cb: types.CallbackQuery, state: FSMContext):
    user_id = cb.from_user.id
    user = db.get_user(user_id)

    if user['banned']:
        await cb.answer("❌ You are banned.", show_alert=True)
        return

    accounts = db.get_user_accounts(user_id)
    # Admins/owners can use all active accounts
    if user['role'] in ('admin', 'owner'):
        accounts = db.get_all_active_accounts()

    if not accounts:
        await cb.answer("❌ No accounts available. Add an account first.", show_alert=True)
        return

    await cb.message.edit_text(
        "🎯 <b>Enter Report Target</b>\n\n"
        "Send the <b>username, link, or ID</b> of the channel/group/user to report."
        "<i>Examples:</i>\n"
        "<ul>\n"
        "<li>• @channelname\n"
        "<li>• https://t.me/channelname\n"
        "<li>• t.me/channelname",
        parse_mode="HTML"
    )
    await state.set_state(ReportTarget.waiting_target)
    await cb.answer()
    
```



```

@dp.message(ReportTarget.waiting_target)
async def process_target(message: types.Message, state: FSMContext):
    target = message.text.strip()
    await state.update_data(target=target)
    await message.answer(
        f"🎯 Target: <code>{target}</code>\n\n"
        "📋 <b>Select report type:</b>",
        reply_markup=report_type_keyboard(),
        parse_mode="HTML"
    )
    await state.set_state(ReportTarget.waiting_report_type)
    
```



```

@dp.callback_query(F.data.startswith("rtype_"))
async def process_report_type(cb: types.CallbackQuery, state: FSMContext):
    rtype = cb.data.replace("rtype_", "")
    
```

```

await state.update_data(report_type=rtype)

if rtype == "custom":
    await cb.message.edit_text(
        "✍ <b>Enter your custom report text:</b>\n\n"
        "This text will be included in the report to Telegram.",
        parse_mode="HTML"
    )
    await state.set_state(ReportTarget.waiting_custom_text)
else:
    await cb.message.edit_text(
        "🔢 <b>How many times to report?</b>\n\n"
        "Each report costs <b>1 token</b>.\n"
        "Select or enter a number:",
        reply_markup=report_count_keyboard(),
        parse_mode="HTML"
    )
    await state.set_state(ReportTarget.waiting_count)
await cb.answer()

```

```

@dp.message(ReportTarget.waiting_custom_text)
async def process_custom_text(message: types.Message, state: FSMContext):
    await state.update_data(custom_text=message.text.strip())
    await message.answer(
        "🔢 <b>How many times to report?</b>\n\n"
        "Each report costs <b>1 token</b>.",
        reply_markup=report_count_keyboard(),
        parse_mode="HTML"
    )
    await state.set_state(ReportTarget.waiting_count)

```

```

@dp.callback_query(F.data.startswith("rcount_"), ReportTarget.waiting_count)
async def process_report_count(cb: types.CallbackQuery, state: FSMContext):
    user_id = cb.from_user.id
    user = db.get_user(user_id)
    count_str = cb.data.replace("rcount_", "")

    if count_str == "custom":
        await cb.message.edit_text("Enter the number of reports:")
        await cb.answer()
        return

    count = int(count_str)
    data = await state.get_data()

```

```

# Token check
is_unlimited = user['role'] == 'owner' or user.get('unlimited_tokens')
if not is_unlimited and user['tokens'] < count:
    await cb.answer(
        f"❌ Not enough tokens! You have {user['tokens']}, need {count}.",
        show_alert=True
    )
    return

# Get accounts
if user['role'] in ('admin', 'owner'):
    accounts = db.get_all_active_accounts()
else:
    accounts = db.get_user_accounts(user_id)

if not accounts:
    await cb.answer("❌ No accounts available!", show_alert=True)
    return

target = data['target']
rtype = data['report_type']
custom_text = data.get('custom_text', '')

await cb.message.edit_text(
    f"⚡ <b>Starting reports...</b>\n\n"
    f"🎯 Target: <code>{target}</code>\n"
    f"📅 Type: <b>{rtype}</b>\n"
    f"🔢 Count: <b>{count}</b>\n"
    f"📱 Accounts: <b>{len(accounts)}</b>\n\n"
    f"⏳ Processing...",
    parse_mode="HTML"
)

# Run reports
result = await report_engine.run_reports(
    user_id=user_id,
    target=target,
    report_type=rtype,
    custom_text=custom_text,
    count=count,
    accounts=accounts
)

# Deduct tokens
if not is_unlimited:
    db.deduct_tokens(user_id, result['success'])

```

```

await cb.message.edit_text(
    f"✅ <b>Report Complete!</b>\n\n"
    f"🎯 Target: <code>{target}</code>\n"
    f"✅ Successful: <b>{result['success']}</b>\n"
    f"❌ Failed: <b>{result['failed']}</b>\n"
    f"💰 Tokens used: <b>{result['success']}</b>\n\n"
    f"Telegram moderators will review the reports.",
    reply_markup=InlineKeyboardMarkup(inline_keyboard=[[
        InlineKeyboardButton(text="⬅ Back to Menu", callback_data="menu_back"
    ]]),
    parse_mode="HTML"
)
await state.clear()
await cb.answer()

```

```

# _____
# ACCOUNTS
# _____

@dp.callback_query(F.data == "menu_accounts")
async def menu_accounts(cb: types.CallbackQuery):
    user_id = cb.from_user.id
    user = db.get_user(user_id)
    accounts = db.get_user_accounts(user_id)

    builder = InlineKeyboardBuilder()
    builder.button(text="➕ Add Account", callback_data="acc_add")
    for acc in accounts:
        status = "✅" if acc['active'] else "❌"
        builder.button(
            text=f"{status} {acc['phone']}",
            callback_data=f"acc_view_{acc['id']}"
        )
    builder.button(text="⬅ Back", callback_data="menu_back")
    builder.adjust(1)

    await cb.message.edit_text(
        f"📱 <b>Your Accounts</b> ({len(accounts)} linked)\n\n"
        "Accounts are used to submit reports simultaneously.",
        reply_markup=builder.as_markup(),
        parse_mode="HTML"
    )
    await cb.answer()

@dp.callback_query(F.data == "acc_add")

```

```

async def acc_add(cb: types.CallbackQuery, state: FSMContext):
    await cb.message.edit_text(
        "📱 <b>Add Telegram Account</b>\n\n"
        "Enter your phone number in international format:\n"
        "<i>Example: +1234567890</i>",
        parse_mode="HTML"
    )
    await state.set_state(AddAccount.waiting_phone)
    await cb.answer()

@dp.message(AddAccount.waiting_phone)
async def process_phone(message: types.Message, state: FSMContext):
    phone = message.text.strip()
    user_id = message.from_user.id

    await message.answer("⏳ Sending verification code...")

    result = await account_manager.send_code(phone, user_id)
    if result['success']:
        await state.update_data(phone=phone, phone_code_hash=result['phone_code_h'])
        await message.answer(
            "✅ Code sent!\n\n"
            "Enter the verification code you received:\n"
            "<i>Format: 1-2-3-4-5 (with dashes)</i>",
            parse_mode="HTML"
        )
        await state.set_state(AddAccount.waiting_code)
    else:
        await message.answer(f"❌ Failed: {result['error']}")
        await state.clear()

@dp.message(AddAccount.waiting_code)
async def process_code(message: types.Message, state: FSMContext):
    code = message.text.strip().replace("-", "").replace(" ", "")
    data = await state.get_data()
    user_id = message.from_user.id

    await message.answer("⏳ Verifying...")

    result = await account_manager.verify_code(
        data['phone'], code, data['phone_code_hash'], user_id
    )

    if result['success']:
        await message.answer(

```

```

    "✓ <b>Account added successfully!</b>\n\n"
    f"📱 Phone: {data['phone']}\n"
    "This account will now be used for reporting.",
    parse_mode="HTML",
    reply_markup=InlineKeyboardMarkup(inline_keyboard=[[
        InlineKeyboardButton(text="⬅ Back to Menu", callback_data="menu_"
    ]])
)
await state.clear()
elif result.get('need_password'):
    await message.answer("🔒 2FA enabled. Enter your password:")
    await state.set_state(AddAccount.waiting_password)
else:
    await message.answer(f"✗ Failed: {result['error']}")  

    await state.clear()

@dp.message(AddAccount.waiting_password)
async def process_password(message: types.Message, state: FSMContext):
    password = message.text.strip()
    data = await state.get_data()
    user_id = message.from_user.id

    result = await account_manager.verify_password(data['phone'], password, user_

if result['success']:
    await message.answer(
        "✓ <b>Account added with 2FA!</b>",
        parse_mode="HTML",
        reply_markup=InlineKeyboardMarkup(inline_keyboard=[[
            InlineKeyboardButton(text="⬅ Back to Menu", callback_data="menu_"
        ]])
)
else:
    await message.answer(f"✗ Failed: {result['error']}")

await state.clear()

@dp.callback_query(F.data.startswith("acc_view_"))
async def acc_view(cb: types.CallbackQuery):
    acc_id = int(cb.data.replace("acc_view_", ""))
    acc = db.get_account_by_id(acc_id)
    if not acc:
        await cb.answer("Account not found", show_alert=True)
    return

```

```

status = "✅ Active" if acc['active'] else "❌ Inactive"
builder = InlineKeyboardBuilder()
if acc['active']:
    builder.button(text="❌ Deactivate", callback_data=f"acc_deactivate_{acc_id}")
else:
    builder.button(text="✅ Activate", callback_data=f"acc_activate_{acc_id}")
builder.button(text="🗑 Remove", callback_data=f"acc_remove_{acc_id}")
builder.button(text="⬅ BACK", callback_data="menu_accounts")
builder.adjust(2)

await cb.message.edit_text(
    f"📱 <b>Account Details</b>\n\n"
    f"📞 Phone: {acc['phone']}\n"
    f"📊 Status: {status}\n"
    f"📝 Reports done: {acc['reports_done']}",
    reply_markup=builder.as_markup(),
    parse_mode="HTML"
)
await cb.answer()

```

```

@dp.callback_query(F.data.startswith("acc_remove_"))
async def acc_remove(cb: types.CallbackQuery):
    acc_id = int(cb.data.replace("acc_remove_", ""))
    db.remove_account(acc_id)
    await cb.answer("✅ Account removed", show_alert=True)
    await menu_accounts(cb)

```

```

@dp.callback_query(F.data.startswith("acc_deactivate_"))
async def acc_deactivate(cb: types.CallbackQuery):
    acc_id = int(cb.data.replace("acc_deactivate_", ""))
    db.set_account_active(acc_id, False)
    await cb.answer("❌ Account deactivated")
    await acc_view(cb)

```

```

@dp.callback_query(F.data.startswith("acc_activate_"))
async def acc_activate(cb: types.CallbackQuery):
    acc_id = int(cb.data.replace("acc_activate_", ""))
    db.set_account_active(acc_id, True)
    await cb.answer("✅ Account activated")

```

```

# _____
# MY REPORTS
# _____

```

```

@dp.callback_query(F.data == "menu_my_reports")
async def menu_my_reports(cb: types.CallbackQuery):
    user_id = cb.from_user.id
    reports = db.get_user_reports(user_id, limit=10)

    if not reports:
        text = "📊 <b>My Reports</b>\n\nNo reports yet."
    else:
        text = "📊 <b>My Reports (Last 10)</b>\n\n"
        for r in reports:
            text += f"• <code>{r['target']}</code> - {r['type']} ×{r['count']} ✅"

    await cb.message.edit_text(
        text,
        reply_markup=InlineKeyboardMarkup(inline_keyboard=[
            InlineKeyboardButton(text="⬅ Back", callback_data="menu_back")
        ]),
        parse_mode="HTML"
    )
    await cb.answer()

```

```

# _____
# TOKENS / HELP / CONTACT
# _____

```

```

@dp.callback_query(F.data == "menu_buy_tokens")
async def menu_buy_tokens(cb: types.CallbackQuery):
    token_packages = db.get_token_packages()
    builder = InlineKeyboardBuilder()
    for pkg in token_packages:
        builder.button(text=f"$ {pkg['tokens']} tokens - ${pkg['price']}", callback_data="buy_pkg")
    builder.button(text="⬅ Back", callback_data="menu_back")
    builder.adjust(1)

    await cb.message.edit_text(
        "$ 📰 Buy Tokens\n\nEach report costs 1 token.\n\nSelect a package:",
        reply_markup=builder.as_markup(),
        parse_mode="HTML"
    )
    await cb.answer()

@dp.callback_query(F.data == "menu_help")

```

```

async def menu_help(cb: types.CallbackQuery):
    await cb.message.edit_text(
        "i <b>Help</b>\n\n"
        "<b>How to use:</b>\n"
        "1. Add your Telegram account(s)\n"
        "2. Click Report\n"
        "3. Enter target (username/link)\n"
        "4. Select report type\n"
        "5. Choose how many reports\n"
        "6. All your accounts report simultaneously!\n\n"
        "<b>Token System:</b>\n"
        "• Each report = 1 token\n"
        "• Owners have unlimited tokens\n"
        "• Admins can grant tokens\n\n"
        "<b>Report Types:</b>\n"
        "Pornography, Child Abuse, Violence, Scam, Spam, and more.",
        reply_markup=InlineKeyboardMarkup(inline_keyboard=[[[
            InlineKeyboardButton(text="⬅ Back", callback_data="menu_back")
        ]]]),
        parse_mode="HTML"
    )
    await cb.answer()

```

```

@dp.callback_query(F.data == "menu_contact")
async def menu_contact(cb: types.CallbackQuery):
    await cb.message.edit_text(
        f"📞 Contact\n\n"
        f"For support contact: @{config.SUPPORT_USERNAME}\n"
        f"Bot Owner: @{config.OWNER_USERNAME}",
        reply_markup=InlineKeyboardMarkup(inline_keyboard=[[[
            InlineKeyboardButton(text="⬅ Back", callback_data="menu_back")
        ]]]),
        parse_mode="HTML"
    )
    await cb.answer()

```

```

# _____
# ADMIN PANEL
# _____

```

```

@dp.callback_query(F.data == "menu_admin")
async def menu_admin(cb: types.CallbackQuery):
    user = db.get_user(cb.from_user.id)
    if not user or user['role'] not in ('admin', 'owner'):
        await cb.answer("❌ Access denied", show_alert=True)

```

```

    return

    stats = db.get_stats()
    await cb.message.edit_text(
        f"👑 <b>Admin Panel</b>\n\n"
        f"👤 Total users: {stats['users']}\n"
        f"📱 Total accounts: {stats['accounts']}\n"
        f"📊 Total reports: {stats['reports']}\n",
        reply_markup=admin_keyboard(),
        parse_mode="HTML"
    )
    await cb.answer()

@dp.callback_query(F.data == "admin_add_tokens")
async def admin_add_tokens_start(cb: types.CallbackQuery, state: FSMContext):
    user = db.get_user(cb.from_user.id)
    if not user or user['role'] not in ('admin', 'owner'):
        await cb.answer("❌ Access denied", show_alert=True)
        return

    await cb.message.edit_text("Enter the user ID to add tokens to:")
    await state.set_state(AdminActions.waiting_user_id)
    await state.update_data(admin_action="add_tokens")
    await cb.answer()

@dp.callback_query(F.data == "admin_stats")
async def admin_stats(cb: types.CallbackQuery):
    user = db.get_user(cb.from_user.id)
    if not user or user['role'] not in ('admin', 'owner'):
        await cb.answer("❌ Access denied", show_alert=True)
        return

    stats = db.get_detailed_stats()
    await cb.message.edit_text(
        f"📊 <b>Detailed Stats</b>\n\n"
        f"👤 Users: {stats['users']}\n"
        f"📱 Accounts: {stats['accounts']}\n"
        f"✅ Active accounts: {stats['active_accounts']}\n"
        f"📊 Total reports: {stats['reports']}\n"
        f"✅ Successful reports: {stats['successful_reports']}\n",
        reply_markup=InlineKeyboardMarkup(inline_keyboard=[
            InlineKeyboardButton(text="⬅️ Back", callback_data="menu_admin")
        ]),
        parse_mode="HTML"
    )

```

```

await cb.answer()

@dp.callback_query(F.data == "admin_set_report_text")
async def admin_set_report_text(cb: types.CallbackQuery, state: FSMContext):
    user = db.get_user(cb.from_user.id)
    if not user or user['role'] not in ('admin', 'owner'):
        await cb.answer("❌ Access denied", show_alert=True)
        return

    await cb.message.edit_text(
        "✉️ <b>Set Custom Report Text</b>\n\n"
        "Enter the custom text that will be prepended to reports.\n"
        "Use {target} as placeholder for the target.\n\n"
        "Current text: " + (db.get_setting('report_text') or "Not set"),
        parse_mode="HTML"
    )
    await state.set_state(AdminActions.waiting_custom_report_text)
    await cb.answer()

@dp.message(AdminActions.waiting_custom_report_text)
async def save_custom_report_text(message: types.Message, state: FSMContext):
    db.set_setting('report_text', message.text)
    await message.answer("✅ Custom report text saved!")
    await state.clear()

@dp.message(AdminActions.waiting_user_id)
async def admin_waiting_user_id(message: types.Message, state: FSMContext):
    data = await state.get_data()
    try:
        target_user_id = int(message.text.strip())
        target_user = db.get_user(target_user_id)
        if not target_user:
            await message.answer("❌ User not found.")
            await state.clear()
            return

        await state.update_data(target_user_id=target_user_id)
        await message.answer(
            f"👤 User: {target_user['full_name']} ({target_user_id})\n"
            f"💰 Current tokens: {target_user['tokens']}\n\n"
            "Enter number of tokens to add:"
        )
        await state.set_state(AdminActions.waiting_token_amount)
    except ValueError:
        await message.answer("❌ Invalid user ID.")

```

```

    await state.clear()

@dp.message(AdminActions.waiting_token_amount)
async def admin_add_tokens_amount(message: types.Message, state: FSMContext):
    admin_id = message.from_user.id
    admin = db.get_user(admin_id)
    if not admin or admin['role'] not in ('admin', 'owner'):
        await state.clear()
        return

    data = await state.get_data()
    try:
        amount = int(message.text.strip())
        target_id = data['target_user_id']
        db.add_tokens(target_id, amount)
        await message.answer(f"✅ Added {amount} tokens to user {target_id}!")
    except ValueError:
        await message.answer("❌ Invalid amount.")
    await state.clear()

```

```

# _____
# OWNER PANEL
# _____

```

```

@dp.callback_query(F.data == "menu_owner")
async def menu_owner(cb: types.CallbackQuery):
    user = db.get_user(cb.from_user.id)
    if not user or user['role'] != 'owner':
        await cb.answer("❌ Access denied", show_alert=True)
        return

    await cb.message.edit_text(
        "🔱 <b>Owner Panel</b>\n\nFull control over the bot:",
        reply_markup=owner_keyboard(),
        parse_mode="HTML"
    )
    await cb.answer()

```

```

@dp.callback_query(F.data == "owner_unlimited")
async def owner_unlimited(cb: types.CallbackQuery, state: FSMContext):
    user = db.get_user(cb.from_user.id)
    if not user or user['role'] != 'owner':
        await cb.answer("❌ Access denied", show_alert=True)
        return

```

```
await cb.message.edit_text("Enter user ID to give unlimited tokens:")
await state.set_state(AdminActions.waiting_user_id)
await state.update_data(admin_action="unlimited_tokens")
await cb.answer()
```

```
@dp.callback_query(F.data == "owner_promote")
async def owner_promote(cb: types.CallbackQuery, state: FSMContext):
    user = db.get_user(cb.from_user.id)
    if not user or user['role'] != 'owner':
        await cb.answer("❌ Access denied", show_alert=True)
        return
    await cb.message.edit_text("Enter user ID to promote to Admin:")
    await state.set_state(AdminActions.waiting_user_id)
    await state.update_data(admin_action="promote")
    await cb.answer()
```

```
# _____
# ADMIN USERS LIST
# _____
```

```
@dp.callback_query(F.data == "admin_users")
async def admin_users(cb: types.CallbackQuery):
    user = db.get_user(cb.from_user.id)
    if not user or user['role'] not in ('admin', 'owner'):
        await cb.answer("❌ Access denied", show_alert=True)
        return

    users = db.get_all_users(limit=20)
    text = "👤 <b>All Users (Latest 20)</b>\n\n"
    for u in users:
        role_e = {"owner": "🔱", "admin": "👑", "user": "👤"}.get(u['role'], "👤")
        ban = "🚫" if u['banned'] else ""
        text += f"{role_e} <code>{u['user_id']}</code> @{u['username']} - {u['tok
```

await cb.message.edit_text(
 text,
 reply_markup=InlineKeyboardMarkup(inline_keyboard=[
 InlineKeyboardButton(text="⬅ Back", callback_data="menu_admin")
]),
 parse_mode="HTML"
)
await cb.answer()

```
# _____
```

```
# MAIN
# _____

async def main():
    await db.init()
    await dp.start_polling(bot)

if __name__ == "__main__":
    asyncio.run(main())
```