```
import json
import os
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import (
    ApplicationBuilder,
    CommandHandler,
    MessageHandler,
    CallbackQueryHandler,
    ContextTypes,import json
import os
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import (
    ApplicationBuilder,
    CommandHandler,
    MessageHandler,
    CallbackQueryHandler,
    ContextTypes,
    filters,
)

import os
from flask import Flask
from threading import Thread

# Fake web server for Render free plan
app = Flask(__name__)

@app.route('/')
def home():
    return "Bot is running!"

def run_web():
    port = int(os.environ.get("PORT", 10000))
    app.run(host="0.0.0.0", port=port)

# Run web server in background
Thread(target=run_web).start()
BOT_TOKEN = "8521310200:AAH6TtdzDkYfFt_m3GVWj5TyCBfm8cXM_WI"
ADMIN_ID = 7065784096

PROOF_LINK = "https://t.me/saveemoney"

WELCOME_IMG =
"AgACAgUAAxkBAAMRaXjYHNrw0TN5RYuHDGOcnFccP4oAAnQPaxvGQ8lXdb2ih-UCUH
8BAAMCAAN5AAM4BA"
```

```
QR_FILE_ID =
"AgACAgUAAxkBAAID6GmJaKtmDn3WdZsBG-yI-0H7iEw5AAJMDmsbW8gwVCj2Qrn0I47
MAQADAgADeQADOgQ"

DATA_FILE = "products.json"
USERS_FILE = "users.json"

ORDERS = {}
ORDER_COUNTER = 1


# ---------------- DEFAULT PRODUCTS (tera purana wala) ----------------

DEFAULT_PRODUCTS = {
    "youtube": {
        "name": "YouTube Premium",
        "price": "₹19",
        "validity": "1 Month",
        "img":
"AgACAgUAAxkBAAMCaXjVj8Afi--8XD7SSHkd58u_CnMAAmkPaxvGQ8lXdHxX3xW9qgIB
AAMCAAN5AAM4BA",
    },
    "gemini": {
        "name": "Gemini Pro",
        "price": "₹30",
        "validity": "1 Month",
        "img":
"AgACAgUAAxkBAAMPaXjWvGJ_ZVzj5htqIOsUvLPxqZ4AAnEPaxvGQ8lXXDp0WaskxwwB
AAMCAAN5AAM4BA",
    },
    "chatgpt": {
        "name": "ChatGPT Plus",
        "price": "₹99",
        "validity": "1 Year",
        "img":
"AgACAgUAAxkBAAMDaXjVk76FxLyxaf7iZz6KG59DvLEAAmoPaxvGQ8lXpFtB6TzuFCgBA
AMCAAN5AAM4BA",
    },
}


# ---------------- USERS ----------------

def load_users():
    if not os.path.exists(USERS_FILE):
        return []
    with open(USERS_FILE, "r") as f:
        return json.load(f)
```

```
def save_users(u):
    with open(USERS_FILE, "w") as f:
        json.dump(u, f)


# ---------------- PRODUCTS ----------------

def load_products():
    if not os.path.exists(DATA_FILE):
        with open(DATA_FILE, "w") as f:
            json.dump(DEFAULT_PRODUCTS, f, indent=2)
        return DEFAULT_PRODUCTS.copy()

    with open(DATA_FILE, "r") as f:
        return json.load(f)

def save_products(p):
    with open(DATA_FILE, "w") as f:
        json.dump(p, f, indent=2)


# ---------------- START ----------------

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

    users = load_users()
    uid = update.effective_user.id
    if uid not in users:
        users.append(uid)
        save_users(users)

    PRODUCTS = load_products()

    kb = []

    for key, p in PRODUCTS.items():
        kb.append([
            InlineKeyboardButton(p["name"], callback_data=f"product_{key}")
        ])

    kb.append([InlineKeyboardButton("📸 Proofs", url=PROOF_LINK)])
    kb.append([InlineKeyboardButton("💳 Pay", callback_data="pay")])
    kb.append([InlineKeyboardButton("ℹ️ Information", callback_data="info")])

    await update.message.reply_photo(
        photo=WELCOME_IMG,
        caption=(
            "👋 *Welcome to Danish Digital Saving Store*\n\n"
            "💯 Trusted subscriptions\n"
            "⚡ Instant activation\n"
            "♻️ Replacement available\n\n"
            "👇 Choose your product"
        ),
        reply_markup=InlineKeyboardMarkup(kb),
        parse_mode="Markdown",
    )


# ---------------- /pay ----------------

async def pay_cmd(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_photo(
        photo=QR_FILE_ID,
        caption="📲 Scan & Pay\nPayment ke baad screenshot bhej do."
    )


# ---------------- /fileid ----------------

async def fileid(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != ADMIN_ID:
        return

    if not update.message.reply_to_message:
        await update.message.reply_text("Photo par reply karke /fileid likho")
        return

    msg = update.message.reply_to_message

    if msg.photo:
        await update.message.reply_text(msg.photo[-1].file_id)
    else:
        await update.message.reply_text("Photo par reply karo")


# ---------------- /broadcast ----------------

async def broadcast(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != ADMIN_ID:
        return

    if not context.args:
        await update.message.reply_text("Use:\n/broadcast message")
        return
    text = " ".join(context.args)

    users = load_users()

    sent = 0
    for uid in users:
        try:
            await context.bot.send_message(uid, text)
            sent += 1
        except:
            pass

    await update.message.reply_text(f"✅ Sent to {sent} users")


# ---------------- /proof ----------------

async def proof_cmd(update: Update, context: ContextTypes.DEFAULT_TYPE):
    kb = [[InlineKeyboardButton("📸 Open Proofs Group", url=PROOF_LINK)]]

    await update.message.reply_text(
        "✅ Yahan aap hamare real proofs dekh sakte ho:",
        reply_markup=InlineKeyboardMarkup(kb)
    )


# ---------------- PRODUCT PAGE ----------------

async def product_page(update: Update, context: ContextTypes.DEFAULT_TYPE):
    PRODUCTS = load_products()

    q = update.callback_query
    await q.answer()

    key = q.data.replace("product_", "")

    if key not in PRODUCTS:
        return

    p = PRODUCTS[key]

    kb = [[
        InlineKeyboardButton("🛒 Place Order", callback_data=f"order_{key}"),
        InlineKeyboardButton("❌ Cancel", callback_data="cancel")
    ]]

    await q.message.reply_photo(
        photo=p["img"],
        caption=(
            f"🎁 *{p['name']}*\n\n"
            f"💰 Price: {p['price']}\n"
            f"⏳ Validity: {p['validity']}\n"
            "⚡ Instant activation\n\n"
            "📩 Click place order to continue"
        ),
        reply_markup=InlineKeyboardMarkup(kb),
        parse_mode="Markdown"
    )


# ---------------- PLACE ORDER ----------------

async def place_order(update: Update, context: ContextTypes.DEFAULT_TYPE):
    global ORDER_COUNTER

    PRODUCTS = load_products()

    q = update.callback_query
    await q.answer()

    key = q.data.replace("order_", "")

    if key not in PRODUCTS:
        return

    p = PRODUCTS[key]
    user = q.from_user

    order_id = ORDER_COUNTER
    ORDER_COUNTER += 1

    ORDERS[order_id] = {
        "user_id": user.id,
        "name": user.full_name,
        "username": user.username,
        "product": p["name"],
        "price": p["price"],
        "status": "Pending"
    }

    await q.message.reply_text(
        "✅ Order placed successfully!\n\nAdmin will contact you shortly."
    )

    kb = [[
        InlineKeyboardButton("💳 Send Pay QR", callback_data=f"payuser_{order_id}"),
        InlineKeyboardButton("✅ Order Done", callback_data=f"done_{order_id}"),
        InlineKeyboardButton("❌ Cancel Order", callback_data=f"cancelorder_{order_id}")
    ]]

    text = (
        f"🛒 *New Order*\n\n"
        f"🆔 Order ID: {order_id}\n"
        f"👤 Name: {user.full_name}\n"
        f"👤 Username: @{user.username if user.username else 'N/A'}\n"
        f"📦 Product: {p['name']}\n"
        f"💰 Price: {p['price']}"
    )

    sent = await context.bot.send_message(
        chat_id=ADMIN_ID,
        text=text,
        reply_markup=InlineKeyboardMarkup(kb),
        parse_mode="Markdown"
    )

    context.bot_data[sent.message_id] = user.id


# ---------------- USER → ADMIN ----------------

async def user_message(update: Update, context: ContextTypes.DEFAULT_TYPE):
    msg = update.message
    fwd = await msg.forward(chat_id=ADMIN_ID)
    context.bot_data[fwd.message_id] = msg.from_user.id


# ---------------- ADMIN → USER ----------------

async def admin_reply(update: Update, context: ContextTypes.DEFAULT_TYPE):
    msg = update.message

    if not msg.reply_to_message:
        return

    user_id = context.bot_data.get(msg.reply_to_message.message_id)
    if not user_id:
        return

    await context.bot.send_message(
        chat_id=user_id,
        text=msg.text
    )
```

```
# ---------------- /orders ----------------

async def orders_list(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != ADMIN_ID:
        return

    if not ORDERS:
        await update.message.reply_text("No orders yet.")
        return

    for oid, o in ORDERS.items():
        txt = (
            f"🆔 Order ID: {oid}\n"
            f"👤 {o['name']} (@{o['username'] if o['username'] else 'N/A'})\n"
            f"📦 {o['product']}\n"
            f"💰 {o['price']}\n"
            f"📌 Status: {o['status']}"
        )

        kb = [[
            InlineKeyboardButton("💳 Send Pay QR", callback_data=f"payuser_{oid}"),
            InlineKeyboardButton("✅ Order Done", callback_data=f"done_{oid}"),
            InlineKeyboardButton("❌ Cancel Order", callback_data=f"cancelorder_{oid}")
        ]]

        await update.message.reply_text(txt, reply_markup=InlineKeyboardMarkup(kb))


# ---------------- ADMIN PRODUCT CONTROL ----------------

async def add_product(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != ADMIN_ID:
        return

    data = " ".join(context.args)

    try:
        key, name, price, validity, img = [x.strip() for x in data.split("|")]
    except:
        await update.message.reply_text(
            "Format:\n/addproduct key | Name | Price | Validity | image_file_id"
        )
        return

    p = load_products()

    p[key] = {
        "name": name,
        "price": price,
        "validity": validity,
        "img": img
    }

    save_products(p)
    await update.message.reply_text("✅ Product added")


async def del_product(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != ADMIN_ID:
        return

    if not context.args:
        return

    key = context.args[0]

    p = load_products()

    if key in p:
        del p[key]
        save_products(p)
        await update.message.reply_text("✅ Product deleted")
    else:
        await update.message.reply_text("❌ Product not found")


async def edit_product(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != ADMIN_ID:
        return

    data = " ".join(context.args)

    try:
        key, name, price, validity = [x.strip() for x in data.split("|")]
    except:
        await update.message.reply_text(
            "Format:\n/editproduct key | Name | Price | Validity"
        )
        return

    p = load_products()

    if key not in p:
        await update.message.reply_text("❌ Product not found")
        return
    p[key]["name"] = name
    p[key]["price"] = price
    p[key]["validity"] = validity

    save_products(p)
    await update.message.reply_text("✅ Product updated")


# ---------------- BUTTON HANDLER ----------------

async def buttons(update: Update, context: ContextTypes.DEFAULT_TYPE):
    q = update.callback_query
    data = q.data
    await q.answer()

    if data.startswith("product_"):
        await product_page(update, context)

    elif data.startswith("order_"):
        await place_order(update, context)

    elif data.startswith("payuser_"):
        oid = int(data.replace("payuser_", ""))

        if oid in ORDERS:
            uid = ORDERS[oid]["user_id"]
            await context.bot.send_photo(
                chat_id=uid,
                photo=QR_FILE_ID,
                caption="📲 Scan & Pay\nPayment ke baad screenshot bhej do."
            )

            await q.answer("QR sent to user")

    elif data.startswith("done_"):
        oid = int(data.replace("done_", ""))
        if oid in ORDERS:
            ORDERS[oid]["status"] = "Completed"
            await q.edit_message_text(f"✅ Order {oid} marked as DONE")

    elif data.startswith("cancelorder_"):
        oid = int(data.replace("cancelorder_", ""))
        if oid in ORDERS:
            ORDERS[oid]["status"] = "Cancelled"
            await q.edit_message_text(f"❌ Order {oid} cancelled")

    elif data == "cancel":
        await q.message.reply_text("❌ Cancelled.")

    elif data == "info":
        await q.message.reply_text(
            "💯 Trusted Store\n"
            "⚡ Instant activation\n"
            "♻️ Replacement guaranteed\n\n"
            "Pehle subscription lo, phir payment karo."
        )

    elif data == "pay":
        await q.message.reply_photo(
            photo=QR_FILE_ID,
            caption="📲 Scan & Pay\nPayment ke baad screenshot bhej do."
        )


# ---------------- MAIN ----------------

def main():
    app = ApplicationBuilder().token(BOT_TOKEN).build()

    app.add_handler(CommandHandler("start", start))
    app.add_handler(CommandHandler("orders", orders_list))
    app.add_handler(CommandHandler("proof", proof_cmd))
    app.add_handler(CommandHandler("pay", pay_cmd))
    app.add_handler(CommandHandler("fileid", fileid))
    app.add_handler(CommandHandler("broadcast", broadcast))

    app.add_handler(CommandHandler("addproduct", add_product))
    app.add_handler(CommandHandler("delproduct", del_product))
    app.add_handler(CommandHandler("editproduct", edit_product))

    app.add_handler(CallbackQueryHandler(buttons))

    app.add_handler(MessageHandler(filters.ALL & ~filters.User(ADMIN_ID),
user_message))
    app.add_handler(MessageHandler(filters.ALL & filters.User(ADMIN_ID), admin_reply))

    print("✅ BOT RUNNING ...")
    app.run_polling()


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

```
BOT_TOKEN = "8521310200:AAH6TtdzDkYfFt_m3GVWj5TyCBfm8cXM_WI"
ADMIN_ID = 7065784096

PROOF_LINK = "https://t.me/saveemoney"

WELCOME_IMG =
"AgACAgUAAxkBAAMRaXjYHNrw0TN5RYuHDGOcnFccP4oAAnQPaxvGQ8lXdb2ih-UCUH
8BAAMCAAN5AAM4BA"

QR_FILE_ID =
"AgACAgUAAxkBAAID6GmJaKtmDn3WdZsBG-yI-0H7iEw5AAJMDmsbW8gwVCj2Qrn0I47
MAQADAgADeQADOgQ"

DATA_FILE = "products.json"
USERS_FILE = "users.json"

ORDERS = {}
ORDER_COUNTER = 1


# ---------------- DEFAULT PRODUCTS (tera purana wala) ----------------

DEFAULT_PRODUCTS = {
    "youtube": {
        "name": "YouTube Premium",
        "price": "₹19",
        "validity": "1 Month",
        "img":
"AgACAgUAAxkBAAMCaXjVj8Afi--8XD7SSHkd58u_CnMAAmkPaxvGQ8lXdHxX3xW9qgIB
AAMCAAN5AAM4BA",
    },
    "gemini": {
        "name": "Gemini Pro",
        "price": "₹30",
        "validity": "1 Month",
        "img":
"AgACAgUAAxkBAAMPaXjWvGJ_ZVzj5htqIOsUvLPxqZ4AAnEPaxvGQ8lXXDp0WaskxwwB
AAMCAAN5AAM4BA",
    },
    "chatgpt": {
        "name": "ChatGPT Plus",
        "price": "₹99",
        "validity": "1 Year",
        "img":
"AgACAgUAAxkBAAMDaXjVk76FxLyxaf7iZz6KG59DvLEAAmoPaxvGQ8lXpFtB6TzuFCgBA
AMCAAN5AAM4BA",
    },
}
```

```
# ---------------- USERS ----------------

def load_users():
    if not os.path.exists(USERS_FILE):
        return []
    with open(USERS_FILE, "r") as f:
        return json.load(f)

def save_users(u):
    with open(USERS_FILE, "w") as f:
        json.dump(u, f)


# ---------------- PRODUCTS ----------------

def load_products():
    if not os.path.exists(DATA_FILE):
        with open(DATA_FILE, "w") as f:
            json.dump(DEFAULT_PRODUCTS, f, indent=2)
        return DEFAULT_PRODUCTS.copy()

    with open(DATA_FILE, "r") as f:
        return json.load(f)

def save_products(p):
    with open(DATA_FILE, "w") as f:
        json.dump(p, f, indent=2)


# ---------------- START ----------------

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

    users = load_users()
    uid = update.effective_user.id
    if uid not in users:
        users.append(uid)
        save_users(users)

    PRODUCTS = load_products()

    kb = []

    for key, p in PRODUCTS.items():
        kb.append([
            InlineKeyboardButton(p["name"], callback_data=f"product_{key}")
        ])

    kb.append([InlineKeyboardButton("📸 Proofs", url=PROOF_LINK)])
    kb.append([InlineKeyboardButton("💳 Pay", callback_data="pay")])
    kb.append([InlineKeyboardButton("ℹ️ Information", callback_data="info")])

    await update.message.reply_photo(
        photo=WELCOME_IMG,
        caption=(
            "👋 *Welcome to Danish Digital Saving Store*\n\n"
            "💯 Trusted subscriptions\n"
            "⚡ Instant activation\n"
            "♻️ Replacement available\n\n"
            "👇 Choose your product"
        ),
        reply_markup=InlineKeyboardMarkup(kb),
        parse_mode="Markdown",
    )


# ---------------- /pay ----------------

async def pay_cmd(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_photo(
        photo=QR_FILE_ID,
        caption="📲 Scan & Pay\nPayment ke baad screenshot bhej do."
    )


# ---------------- /fileid ----------------

async def fileid(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != ADMIN_ID:
        return

    if not update.message.reply_to_message:
        await update.message.reply_text("Photo par reply karke /fileid likho")
        return

    msg = update.message.reply_to_message

    if msg.photo:
        await update.message.reply_text(msg.photo[-1].file_id)
    else:
        await update.message.reply_text("Photo par reply karo")


# ---------------- /broadcast ----------------
```

```
async def broadcast(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != ADMIN_ID:
        return

    if not context.args:
        await update.message.reply_text("Use:\n/broadcast message")
        return

    text = " ".join(context.args)

    users = load_users()

    sent = 0
    for uid in users:
        try:
            await context.bot.send_message(uid, text)
            sent += 1
        except:
            pass

    await update.message.reply_text(f"✅ Sent to {sent} users")


# ---------------- /proof ----------------

async def proof_cmd(update: Update, context: ContextTypes.DEFAULT_TYPE):
    kb = [[InlineKeyboardButton("📸 Open Proofs Group", url=PROOF_LINK)]]

    await update.message.reply_text(
        "✅ Yahan aap hamare real proofs dekh sakte ho:",
        reply_markup=InlineKeyboardMarkup(kb)
    )


# ---------------- PRODUCT PAGE ----------------

async def product_page(update: Update, context: ContextTypes.DEFAULT_TYPE):
    PRODUCTS = load_products()

    q = update.callback_query
    await q.answer()

    key = q.data.replace("product_", "")

    if key not in PRODUCTS:
        return
    p = PRODUCTS[key]

    kb = [[
        InlineKeyboardButton("🛒 Place Order", callback_data=f"order_{key}"),
        InlineKeyboardButton("❌ Cancel", callback_data="cancel")
    ]]

    await q.message.reply_photo(
        photo=p["img"],
        caption=(
            f"🎁 *{p['name']}*\n\n"
            f"💰 Price: {p['price']}\n"
            f"⏳ Validity: {p['validity']}\n"
            "⚡ Instant activation\n\n"
            "📩 Click place order to continue"
        ),
        reply_markup=InlineKeyboardMarkup(kb),
        parse_mode="Markdown"
    )


# ---------------- PLACE ORDER ----------------

async def place_order(update: Update, context: ContextTypes.DEFAULT_TYPE):
    global ORDER_COUNTER

    PRODUCTS = load_products()

    q = update.callback_query
    await q.answer()

    key = q.data.replace("order_", "")

    if key not in PRODUCTS:
        return

    p = PRODUCTS[key]
    user = q.from_user

    order_id = ORDER_COUNTER
    ORDER_COUNTER += 1

    ORDERS[order_id] = {
        "user_id": user.id,
        "name": user.full_name,
        "username": user.username,
        "product": p["name"],
        "price": p["price"],
        "status": "Pending"
    }

    await q.message.reply_text(
        "✅ Order placed successfully!\n\nAdmin will contact you shortly."
    )

    kb = [[
        InlineKeyboardButton("💳 Send Pay QR", callback_data=f"payuser_{order_id}"),
        InlineKeyboardButton("✅ Order Done", callback_data=f"done_{order_id}"),
        InlineKeyboardButton("❌ Cancel Order", callback_data=f"cancelorder_{order_id}")
    ]]

    text = (
        f"🛒 *New Order*\n\n"
        f"🆔 Order ID: {order_id}\n"
        f"👤 Name: {user.full_name}\n"
        f"👤 Username: @{user.username if user.username else 'N/A'}\n"
        f"📦 Product: {p['name']}\n"
        f"💰 Price: {p['price']}"
    )

    sent = await context.bot.send_message(
        chat_id=ADMIN_ID,
        text=text,
        reply_markup=InlineKeyboardMarkup(kb),
        parse_mode="Markdown"
    )

    context.bot_data[sent.message_id] = user.id


# ---------------- USER → ADMIN ----------------

async def user_message(update: Update, context: ContextTypes.DEFAULT_TYPE):
    msg = update.message
    fwd = await msg.forward(chat_id=ADMIN_ID)
    context.bot_data[fwd.message_id] = msg.from_user.id


# ---------------- ADMIN → USER ----------------

async def admin_reply(update: Update, context: ContextTypes.DEFAULT_TYPE):
    msg = update.message

    if not msg.reply_to_message:
        return
    user_id = context.bot_data.get(msg.reply_to_message.message_id)
    if not user_id:
        return

    await context.bot.send_message(
        chat_id=user_id,
        text=msg.text
    )


# ---------------- /orders ----------------

async def orders_list(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != ADMIN_ID:
        return

    if not ORDERS:
        await update.message.reply_text("No orders yet.")
        return

    for oid, o in ORDERS.items():
        txt = (
            f"🆔 Order ID: {oid}\n"
            f"👤 {o['name']} (@{o['username'] if o['username'] else 'N/A'})\n"
            f"📦 {o['product']}\n"
            f"💰 {o['price']}\n"
            f"📌 Status: {o['status']}"
        )

        kb = [[
            InlineKeyboardButton("💳 Send Pay QR", callback_data=f"payuser_{oid}"),
            InlineKeyboardButton("✅ Order Done", callback_data=f"done_{oid}"),
            InlineKeyboardButton("❌ Cancel Order", callback_data=f"cancelorder_{oid}")
        ]]

        await update.message.reply_text(txt, reply_markup=InlineKeyboardMarkup(kb))


# ---------------- ADMIN PRODUCT CONTROL ----------------

async def add_product(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != ADMIN_ID:
        return

    data = " ".join(context.args)

    try:
        key, name, price, validity, img = [x.strip() for x in data.split("|")]
    except:
        await update.message.reply_text(
            "Format:\n/addproduct key | Name | Price | Validity | image_file_id"
        )
        return

    p = load_products()

    p[key] = {
        "name": name,
        "price": price,
        "validity": validity,
        "img": img
    }

    save_products(p)
    await update.message.reply_text("✅ Product added")


async def del_product(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != ADMIN_ID:
        return

    if not context.args:
        return

    key = context.args[0]

    p = load_products()

    if key in p:
        del p[key]
        save_products(p)
        await update.message.reply_text("✅ Product deleted")
    else:
        await update.message.reply_text("❌ Product not found")


async def edit_product(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != ADMIN_ID:
        return

    data = " ".join(context.args)

    try:
        key, name, price, validity = [x.strip() for x in data.split("|")]
    except:
        await update.message.reply_text(
            "Format:\n/editproduct key | Name | Price | Validity"
        )
        return

    p = load_products()

    if key not in p:
        await update.message.reply_text("❌ Product not found")
        return

    p[key]["name"] = name
    p[key]["price"] = price
    p[key]["validity"] = validity

    save_products(p)
    await update.message.reply_text("✅ Product updated")


# ---------------- BUTTON HANDLER ----------------

async def buttons(update: Update, context: ContextTypes.DEFAULT_TYPE):
    q = update.callback_query
    data = q.data
    await q.answer()

    if data.startswith("product_"):
        await product_page(update, context)

    elif data.startswith("order_"):
        await place_order(update, context)

    elif data.startswith("payuser_"):
        oid = int(data.replace("payuser_", ""))

        if oid in ORDERS:
            uid = ORDERS[oid]["user_id"]
            await context.bot.send_photo(
                chat_id=uid,
                photo=QR_FILE_ID,
                caption="📲 Scan & Pay\nPayment ke baad screenshot bhej do."
            )

            await q.answer("QR sent to user")

    elif data.startswith("done_"):
        oid = int(data.replace("done_", ""))
        if oid in ORDERS:
            ORDERS[oid]["status"] = "Completed"
            await q.edit_message_text(f"✅ Order {oid} marked as DONE")

    elif data.startswith("cancelorder_"):
        oid = int(data.replace("cancelorder_", ""))
        if oid in ORDERS:
            ORDERS[oid]["status"] = "Cancelled"
            await q.edit_message_text(f"❌ Order {oid} cancelled")

    elif data == "cancel":
        await q.message.reply_text("❌ Cancelled.")

    elif data == "info":
        await q.message.reply_text(
            "💯 Trusted Store\n"
            "⚡ Instant activation\n"
            "♻️ Replacement guaranteed\n\n"
            "Pehle subscription lo, phir payment karo."
        )

    elif data == "pay":
        await q.message.reply_photo(
            photo=QR_FILE_ID,
            caption="📲 Scan & Pay\nPayment ke baad screenshot bhej do."
        )


# ---------------- MAIN ----------------

def main():
    app = ApplicationBuilder().token(BOT_TOKEN).build()

    app.add_handler(CommandHandler("start", start))
    app.add_handler(CommandHandler("orders", orders_list))
    app.add_handler(CommandHandler("proof", proof_cmd))
    app.add_handler(CommandHandler("pay", pay_cmd))
    app.add_handler(CommandHandler("fileid", fileid))
    app.add_handler(CommandHandler("broadcast", broadcast))

    app.add_handler(CommandHandler("addproduct", add_product))
    app.add_handler(CommandHandler("delproduct", del_product))
    app.add_handler(CommandHandler("editproduct", edit_product))

    app.add_handler(CallbackQueryHandler(buttons))

    app.add_handler(MessageHandler(filters.ALL & ~filters.User(ADMIN_ID),
user_message))
    app.add_handler(MessageHandler(filters.ALL & filters.User(ADMIN_ID), admin_reply))
    print("✅ BOT RUNNING ...")
    app.run_polling()


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