PixelAffari — Pacchetto pronto per GitHub (Render)

Questo documento contiene tutti i file pronti da copiare nel repository GitHub per effettuare il deploy su **Render**. Copia ogni file così come è nella root del repo `pixelaffari/`.

File: `main.py`

```
# main.py
import os
import re
import asyncio
import time
import logging
from urllib.parse import urlparse, parse_qs, urlencode, urlunparse

from telethon import TelegramClient, events, Button
from telethon.sessions import StringSession
from telethon.tl.types import MessageMediaWebPage
from aiohttp import web

logging.basicConfig(level=logging.INFO, format='%(asctime)s %(levelname)s: %
(message)s')

# ---------------- CONFIG (da env / Secrets) ----------------
API_ID = int(os.environ['API_ID'])
API_HASH = os.environ['API_HASH']
STRING_SESSION = os.environ['STRING_SESSION']
DEST_CHANNEL_RAW = os.environ.get('DEST_CHANNEL', '@PixelAffari')
AFFILIATE_TAG = os.environ.get('AFFILIATE_TAG', 'tag=pixelofferte-21')
FORWARD_ALL = os.environ.get('FORWARD_ALL', 'true').lower() in ('1', 'true',
'yes')
ANTI_SPAM_DELAY = float(os.environ.get('ANTI_SPAM_DELAY', '5'))
SEEN_TTL = int(os.environ.get('SEEN_TTL', str(24 * 3600)))
PORT = int(os.environ.get('PORT', '8080'))

# Normalizza DEST_CHANNEL (accetta ID numerico o @username)
try:
    DEST_CHANNEL = int(DEST_CHANNEL_RAW)
except Exception:
    DEST_CHANNEL = DEST_CHANNEL_RAW

# ---------------- utilità link ----------------
LINK_RE = re.compile(r'https?://[^\s)\]>\]]+')

def extract_asin_from_amazon_path(path: str):
```

```
    m = re.search(r'/dp/([A-Z0-9]{10})', path)
    if m:
        return m.group(1)
    m = re.search(r'/gp/product/([A-Z0-9]{10})', path)
    if m:
        return m.group(1)
    m = re.search(r'/product/([A-Z0-9]{10})', path)
    if m:
        return m.group(1)
    return None

def build_affiliate_amazon_url(original_url: str, tag: str):
    try:
        p = urlparse(original_url)
        if 'amazon.' not in p.netloc:
            return original_url
        asin = extract_asin_from_amazon_path(p.path)
        if asin:
            new_path = f'/dp/{asin}/'
            q = {'tag': tag}
            new_url = urlunparse((p.scheme, p.netloc, new_path, '',
urlencode(q), ''))
            return new_url
        qs = parse_qs(p.query)
        if 'tag' in qs:
            return original_url
        q = {k: v for k, v in qs.items()}
        q['tag'] = [tag]
        new_query = urlencode({k: v[0] for k, v in q.items()})
        new_url = urlunparse((p.scheme, p.netloc, p.path, p.params,
new_query, p.fragment))
        return new_url
    except Exception:
        return original_url

def transform_links_in_text(text: str):
    if not text:
        return text, []
    links = LINK_RE.findall(text)
    replaced = text
    fixed_links = []
    for l in links:
        if 'amazon.' in l:
            new = build_affiliate_amazon_url(l, AFFILIATE_TAG)
            if new != l:
                replaced = replaced.replace(l, new)
                fixed_links.append((l, new))
    return replaced, fixed_links

# ---------------- load sources ----------------
import os as _os
```

```
def load_sources(fname='canali.txt'):
    if not _os.path.exists(fname):
        logging.warning(f"{fname} non trovato. Crealo e inserisci username
canali, uno per riga.")
        return []
    with open(fname, 'r', encoding='utf-8') as f:
        lines = [ln.strip() for ln in f if ln.strip()]
    return lines

SOURCE_CHANNELS = load_sources()
logging.info(f"Canali sorgente caricati: {SOURCE_CHANNELS}")

# ---------------- dedupe cache ----------------
SEEN = {}  # (chat_id, msg_id) -> timestamp

async def cleanup_seen_task():
    while True:
        now = time.time()
        to_del = [k for k, v in list(SEEN.items()) if now - v > SEEN_TTL]
        for k in to_del:
            del SEEN[k]
        await asyncio.sleep(600)

# ---------------- Telethon client (user session) ----------------
client = TelegramClient(StringSession(STRING_SESSION), API_ID, API_HASH)

from telethon.tl.types import MessageMediaWebPage


def rebuild_buttons(original_buttons):
    if not original_buttons:
        return None
    new = []
    try:
        for row in original_buttons:
            new_row = []
            for b in row:
                text = getattr(b, 'text', None) or (getattr(b, 'button',
None) and getattr(b.button, 'text', None))
                url = getattr(b, 'url', None)
                if url and text:
                    if 'amazon.' in url:
                        url = build_affiliate_amazon_url(url, AFFILIATE_TAG)
                    new_row.append(Button.url(text, url))
                else:
                    new_row.append(b)
            new.append(new_row)
        return new
    except Exception:
        logging.exception("Errore durante rebuild_buttons")
```

```
        return None

# ---------- handler senza decorator (registreremo dinamicamente) ----------
async def handler(event):
    try:
        chat_id = getattr(event.chat, 'id', None) or event.chat_id
        msg_id = event.message.id
        key = (chat_id, msg_id)
        if key in SEEN:
            logging.info("⏩ Ignoro messaggio duplicato")
            return
        SEEN[key] = time.time()

        raw_text = event.message.message or ""
        new_text, fixed_links = transform_links_in_text(raw_text)

        if (not FORWARD_ALL):
            found = any(('amazon.' in l or 'zalando' in l or 'ebay.' in l)
for l in LINK_RE.findall(raw_text or ""))
            if not found:
                logging.info("⚠️ Messaggio filtrato (no link target)")
                return

        buttons = None
        if getattr(event.message, 'buttons', None):
            buttons = rebuild_buttons(event.message.buttons)

        if event.message.media and isinstance(event.message.media,
MessageMediaWebPage):
            webpage = getattr(event.message.media, 'webpage', None)
            extracted_url = None
            try:
                extracted_url = getattr(webpage, 'url', None)
            except Exception:
                extracted_url = None
            out_text = new_text or ""
            if extracted_url and 'amazon.' in extracted_url:
                out_text = out_text.replace(extracted_url,
build_affiliate_amazon_url(extracted_url, AFFILIATE_TAG))
            if buttons:
                await client.send_message(DEST_CHANNEL, out_text or " ",
buttons=buttons)
            else:
                await client.send_message(DEST_CHANNEL, out_text or " ")
            logging.info(f"✅ Inviato WebPage-as-text (msg {msg_id})")

        elif event.message.media:
            caption: str = new_text or ""
            try:
                await client.send_file(DEST_CHANNEL, event.message.media,
caption=caption, buttons=buttons)
```

```
                logging.info(f"✅ Inviato media (msg {msg_id}) da
{getattr(event.chat,'username',chat_id)}")
            except Exception as e:
                logging.warning(f"Forward media fallito ({e}), provo a
inviare solo file senza bottoni.")
                try:
                    await client.send_file(DEST_CHANNEL, event.message.media,
caption=caption)
                    logging.info("✅ Inviato (fallback) media senza bottoni")
                except Exception:
                    logging.exception("Errore invio media fallback")

        else:
            out_text: str = new_text or " "
            try:
                if buttons:
                    await client.send_message(DEST_CHANNEL, out_text,
buttons=buttons)
                else:
                    await client.send_message(DEST_CHANNEL, out_text)
                logging.info(f"✅ Inviato testo (msg {msg_id}) da
{getattr(event.chat,'username',chat_id)}")
            except Exception:
                logging.exception("Errore invio testo")

        for old, new in fixed_links:
            logging.info(f"🔁 Link modificato: {old} -> {new}")

        await asyncio.sleep(ANTI_SPAM_DELAY)

    except Exception:
        logging.exception("Errore nel handler")

# ---------------------- small health server (keepalive)
----------------------
async def handle_ping(request):
    logging.info("Ping ricevuto (UptimeRobot o browser).")
    return web.Response(text="PixelAffari alive")

async def start_health_server():
    app = web.Application()
    app.router.add_get('/', handle_ping)
    app.router.add_get('/health', handle_ping)
    runner = web.AppRunner(app)
    await runner.setup()
    site = web.TCPSite(runner, '0.0.0.0', PORT)
    await site.start()
    logging.info(f"Health server avviato su porta {PORT} (per keepalive).")

# ---------------- main ----------------
async def main():
```

```
    asyncio.create_task(cleanup_seen_task())
    asyncio.create_task(start_health_server())

    await client.start()
    if SOURCE_CHANNELS:
        client.add_event_handler(handler,
events.NewMessage(chats=SOURCE_CHANNELS))
        logging.info("Handler registrato per i canali sorgente.")
    else:
        logging.warning("Nessun canale sorgente caricato; aggiungi usernames
a canali.txt e riavvia per attivare l'ascolto.")

    logging.info("🔕 PixelAffari userbot avviato e in ascolto...")
    await asyncio.Event().wait()

if __name__ == '__main__':
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        logging.info("Arresto richiesto manualmente.")
```

File: `create_string_session.py`

```
from telethon.sync import TelegramClient
from telethon.sessions import StringSession

print("🔹 Inserisci i dati API di Telegram (https://my.telegram.org)")

api_id = int(input("API_ID: ").strip())
api_hash = input("API_HASH: ").strip()

with TelegramClient(StringSession(), api_id, api_hash) as client:
    print("\n✅ Esegui l'accesso con il tuo numero di telefono")
    print("Accesso effettuato come:", client.get_me().username)
    session_string = client.session.save()
    print("\n🕸️ La tua STRING SESSION è:\n")
    print(session_string)
    input("\nPremi INVIO per chiudere...")
```

File: `requirements.txt`

```
telethon>=1.31.0
aiohttp>=3.8.0
python-dotenv>=0.20.0
```

File: `canali.txt`

```
@pazziperilgaming
@RibassiTech
@ScontiMela
@UltimaOfferta
@ATuttoBonus
```

File: `.gitignore`

```
__pycache__/
*.pyc
.env
venv/
.idea/
.replit
```

File: `README.md` (istruzioni rapide)

```
# PixelAffari — Deploy su Render

## Contenuto del repo
- `main.py` — bot userbot + health endpoint
- `create_string_session.py` — genera la STRING_SESSION (esegui localmente
una sola volta)
- `canali.txt` — lista dei canali sorgente (uno per riga)
- `requirements.txt` — dipendenze

## Preparazione
1. Genera `STRING_SESSION` eseguendo `create_string_session.py` in
```