# Yad2Bot Scraper: Technical Documentation

**Version:** 1.0 **Author:** Manus AI

## 1. Introduction

This document provides a detailed technical overview of the Yad2Bot Scraper, a sophisticated Telegram bot designed for real estate professionals. The bot automates the process of scraping property listings from Yad2, extracting valuable data, and integrating it directly into a dedicated CRM system. It is built for a single-admin user and leverages advanced scraping techniques to ensure data accuracy and reliability.

## 2. System Architecture

The bot is built on a robust Python backend, utilizing a modular architecture to separate concerns and enhance maintainability.

### 2.1. Core Technologies

| Component | Technology/Library | Purpose |
|---|---|---|
| **Bot Framework** | `python-telegram-bot` | Handles all interactions with the Telegram Bot API. |
| **Scraping Engine** | `requests` + `BeautifulSoup4` | Core libraries for making HTTP requests and parsing HTML. |
| **Scheduling** | `APScheduler` | Manages automated daily scraping tasks. |
| **Database** | `SQLite` | Stores user schedules and other persistent data. |

### 2.2. Scraping Service: ZenRows Integration

To overcome Yad2’ s anti-scraping mechanisms (like CAPTCHAs and IP blocking), the bot integrates with **ZenRows**. All HTTP requests to Yad2 are routed through the ZenRows API, which uses a large pool of residential proxies and sophisticated browser fingerprinting to ensure a high success rate for data retrieval.

**Code Snippet (Conceptual):**

```
import requests

ZENROWS_API_KEY = 'YOUR_ZENROWS_API_KEY'
ZENROWS_URL = 'https://api.zenrows.com/v1/'

def get_yad2_page(url):
    params = {
        'url': url,
        'apikey': ZENROWS_API_KEY,
        'js_render': 'true', # Enable JavaScript rendering
    }
    response = requests.get(ZENROWS_URL, params=params)
    return response.text
```

### 2.3. Key Dependencies

The project relies on several key Python libraries, managed via `requirements.txt`:

- **`python-telegram-bot`**: For the bot’ s core functionality.
- **`requests`**: For making HTTP requests to ZenRows.
- **`beautifulsoup4`**: For parsing the HTML content of scraped pages.
- **`apscheduler`**: For scheduling recurring scraping jobs.
- **`pandas`**: For creating and managing the final CSV data files.

## 3. Core Mechanisms

The bot employs several advanced mechanisms to ensure efficient and reliable operation.

## 3.1. Dual-Monitor System

The scraping process is divided into two distinct phases, each monitored by its own progress tracker (`progress_monitor_fixed.py`). This provides the admin with real-time, granular feedback on the bot’s activity.

- **Monitor 1: Listing Scraping**
  - **Responsibility:** Scans the Yad2 search result pages to find new listings.
  - **Feedback:** Shows progress based on pages scanned and listings found (e.g., “📄 דף 5⁄10” ).
- **Monitor 2: Phone Number Extraction**
  - **Responsibility:** Visits each individual listing page to extract the phone number and other details.
  - **Feedback:** Shows progress based on the number of listings processed (e.g., “📊 התקדמות: 75% (15⁄20)” ).

This separation ensures that even if phone extraction fails for one listing, the overall scraping process is not halted.

## 3.2. Duplicate Prevention

To avoid sending the same listing multiple times, the bot maintains a local database (or a simple file-based log) of listing IDs that have already been processed. Before processing a new listing, it checks against this log. If the ID already exists, the listing is marked as a duplicate and skipped, saving processing time and ensuring clean data.

## 3.3. Output Generation

Upon completion of a scraping task, the bot generates two primary outputs:

1. **CSV File:** A comprehensive comma-separated values file containing all extracted data (price, address, description, phone number, etc.). This file is sent directly to the admin via Telegram.
2. **WhatsApp Links:** For each listing with a valid phone number, the bot can generate a direct `wa.me` link, allowing the admin to initiate a WhatsApp

conversation with a single click.

### 3.4. CRM Integration: Automatic Data Transfer

This is one of the bot’ s most powerful features. All successfully scraped and processed listings are **automatically pushed** to the dedicated **`yad2bot` CRM** built as a Mini App within Telegram.

**How it Works:**

1. After the phone extraction phase, the `scraper_manager` compiles the final data for each valid listing.
2. It then makes a secure, authenticated API call to the backend service of the `yad2bot` Mini App.
3. The data (including name, phone, city, property type, etc.) is sent as a JSON payload.
4. The Mini App’ s backend receives the data and creates a new lead/contact in its database.

**How to Use the CRM Data:**

- Open the `yad2bot` Mini App in Telegram.
- The main dashboard or “Leads” section will be populated with the new listings.
- The admin can then manage these leads, change their status (e.g., “New,” “Contacted,” “Closed”), add notes, and track their entire lifecycle directly within the CRM interface.

## 4. Code Structure

The project is organized into several key files:

| File | Description |
|---|---|
| `scraper_service_bot_main.py` | The main entry point of the application. Initializes the bot, scheduler, and handlers. |
| `bot_handlers.py` | Contains all the Telegram command and callback query handlers (e.g., `/start`, button clicks). |
| `bot_menus.py` | Defines the structure and layout of all the inline keyboards (menus and buttons). |
| `scraper_manager_final.py` | Manages the entire scraping lifecycle, from initiating the process to calling the progress monitors and handling results. |
| `progress_monitor_fixed.py` | The dual-monitor system for providing real-time progress updates to the user. |
| `scheduler.py` | Contains the `BotScheduler` class, which manages creating, saving, and loading scheduled jobs. |
| `database.db` | The SQLite database file. |

---

This documentation provides a high-level understanding of the bot’ s internal workings. For more specific implementation details, refer to the source code and inline comments within the source code.