AI-Driven Web Automation for SaaS User Management (Vipul251/ai_saas_automation)

Video: Video Demonstration: https://drive.google.com/file/d/1uwng3vmG4VAqBhls3hpOWEh4EI46bf4L/view?usp=sharing

1. Understanding the Problem

Why do some SaaS apps lack APIs for user data and provisioning?

- **Legacy Architecture**: Older SaaS platforms were not built with API-first approaches.
- **Security Concerns**: Companies often restrict API access to avoid exposing sensitive endpoints.
- **Monetization Strategy**: Some vendors offer API access only in premium tiers.
- **Low Demand**: Not all customers require programmatic access, so APIs are not prioritized.

Challenges in Automating SaaS Admin Portals

- **Dynamic UIs**: Frequent frontend changes make selectors brittle.
- **Authentication Complexity**: MFA, CAPTCHAs, and session timeouts hinder automation.
- **Lack of Standardization**: Each portal has its own layout and naming conventions.
- **Pagination and Lazy Loading**: User lists are often split across pages or loaded dynamically.

---

2. Research on Available Technologies

AI-Driven Automation Tools

- **LangChain + OpenAI**: For parsing HTML, deciding actions, generating selectors.
- **Auto-GPT / AgentGPT**: Autonomous agents for multi-step navigation.
- **OpenAI Function Calling**: Define toolsets for AI to decide and call.

Headless Browsers

- **Playwright**: Fast, modern, supports multiple browsers, ideal for automation.
- **Selenium**: Older but still widely supported; flexible for most UI tasks.

RPA Tools

- **Robocorp**: Python-based open-source RPA.
- **UiPath**: Powerful but paid enterprise solution.

Authentication Handling

- **Session Storage**: Save cookies or JWTs for session re-use.
- **2FA Bypass**: Manual setup, OTP forwarders.
- **CAPTCHA**: Use services like 2Captcha or AI vision models (OCR/YOLO).

---

3. Proposed Solution

Workflow for Scraping User Data

1. **Login**: Use Playwright for login automation.
2. **Session Handling**: Store cookies/session data in JSON.
3. **HTML Extraction**: Use Playwright or BeautifulSoup.

4. **Selector Inference**: Use OpenAI to identify labels/selectors dynamically.
5. **Data Structuring**: Convert DOM data to structured JSON (name, email, last login).

UI Change Adaptability

- Use **GPT-based selector agents** to dynamically infer selectors.
- Implement **fallback logic** using multiple matching strategies.

Workflow for Provisioning/Deprovisioning

1. **Navigate to User Management**
2. **Identify Buttons** (e.g., "Add User", "Remove") using LLMs.
3. **Fill Forms**: Trigger actions using Playwright.
4. **Confirm Actions**: Wait for visual confirmation (e.g., "User Created")

UI Flow Handling

- Use **OpenAI GPT-4o** for dynamic reasoning based on DOM structure.
- Maintain **template files** per app as base configuration.

Checks for Execution

- **Screenshot before/after action**
- **Log error/success messages**
- **Retry logic** for transient failures

---

4. Scalability & Automation

Handling Different UIs

- Define a **base RPA framework**
- Create a config file per SaaS app with:
    - Login URLs
    - Field mappings
    - Expected HTML structures

AI for UI Resilience

- GPT-powered selector reasoning enables operation even when UI updates.
- Use **semantic matching** rather than strict IDs/classes.

---

5. Testing & Proof of Concept (PoC)

PoC Target: Dropbox

- Used Playwright to log into a test Dropbox account
- Scraped user list from team settings page
- Selector agent used GPT-4o to find dynamic selectors

Challenges:

- CAPTCHA appeared intermittently
- CSS class names were obfuscated
- Required multiple retries due to dynamic content

Solutions:

- Used session re-use to bypass login frequently

- Added logic to re-query selectors using LLMs

---

6. Prototype / Pseudocode

```
from playwright.sync_api import sync_playwright
from ai_helpers.gpt_selector_agent import get_selector

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("https://dropbox.com/login")

    email_selector = get_selector("Email Field", page.content())
    page.fill(email_selector, "admin@example.com")

    password_selector = get_selector("Password Field", page.content())
    page.fill(password_selector, "securepass")
    page.click("button[type='submit']")

    page.wait_for_load_state("networkidle")
    # Proceed to scrape data
```

Conclusion

This AI-powered automation system combines LLM reasoning with browser RPA to manage SaaS admin portals effectively. It's scalable, adaptable to UI changes, and significantly reduces human effort in SaaS user management.

Optional Add-ons

- CI/CD to automate nightly runs
- Streamlit dashboard to monitor results
- Neo4j/FAISS integration to query scraped users semantically

---

Prepared By: Vipul Bhatt
Contact: contact.vipulbhatt@gmail.com
GitHub: github.com/Vipul251
