```python
import pandas as pd
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException, StaleElementReferenceException
import time
import random
from dotenv import load_dotenv
import os
import logging


# Load environment variables from .env
load_dotenv()

# CONFIG
URL_1 = os.getenv("URL_1")
URL_2 = os.getenv("URL_2")
STATE_1 = os.getenv("STATE_1")
STATE_2 = os.getenv("STATE_2")
OUTPUT_FILE_1 = os.getenv("OUTPUT_FILE_1")
OUTPUT_FILE_2 = os.getenv("OUTPUT_FILE_2")
BACKUP_FILE_1 = os.getenv("BACKUP_FILE_1")
BACKUP_FILE_2 = os.getenv("BACKUP_FILE_2")
SAVE_EVERY = 100
RESTART_BROWSER_EVERY = 500


# Selenium options
def create_driver():
    options = webdriver.ChromeOptions()
    # options.add_argument('--headless')
    options.add_argument('--headless=new')
    options.add_argument('--disable-gpu')
    options.add_argument('--window-size=1920,1080')
    return webdriver.Chrome(options=options)


# Create logger for each state
def setup_logger(state: str):
    logger = logging.getLogger(state)
    logger.setLevel(logging.INFO)

    # Avoid re-adding handlers if already set
    if not logger.handlers:
        file_handler = logging.FileHandler(f"{state.lower()}_log_info.log", mode="a", encoding="utf-8")
        stream_handler = logging.StreamHandler()

        formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s', datefmt="%Y-%m-%d %H:%M:
%S")
        file_handler.setFormatter(formatter)
        stream_handler.setFormatter(formatter)

        logger.addHandler(file_handler)
        logger.addHandler(stream_handler)

    return logger


def agents_from_state(url, state, output_file, backup_file, start_page, finish_page):

    logger = setup_logger(state)
```

```python
logger.info(f"{state}: Start scrapping: Pages: {start_page}-{finish_page}")

driver = create_driver()
wait = WebDriverWait(driver, 15)

# Determine starting Agent_index
if os.path.exists(output_file):
    df_existing = pd.read_csv(output_file, sep="*")
    if "Agent_index" in df_existing.columns:
        index = df_existing["Agent_index"].str.extract(r'(\d+)').astype(int).max()[0]
    else:
        index = 0
else:
    index = 0

# Final data list
all_data = []

for page in range(start_page, finish_page + 1):
    logger.info(f"{state}: Scraping page {page}")
    agent_cards = []

    try:
        driver.get(url.format(page=page))
        time.sleep(random.uniform(2, 3))
        wait.until(EC.presence_of_element_located((By.CLASS_NAME, "qa-flh-results-list")))

        retries = 3
        while retries > 0:
            try:
                result = driver.find_element(By.CLASS_NAME, "qa-flh-results-list")
                agent_cards = result.find_elements(By.XPATH, "./li")
                break
            except StaleElementReferenceException:
                retries -= 1
                logger.warning(f"{state}: Page {page} Agent {index}: Retrying due to stale element...")
                time.sleep(1)

    except Exception as e:
        logger.error(f"{state}: Failed on page {page}: {e}")
        continue

    if not agent_cards:
        logger.error(f"{state}: Page {page} contains 0 agent cards.")
        continue

    for card in agent_cards:
        index += 1

        # agent name
        try:
            name = card.find_element(By.CLASS_NAME, "qa-flh-resource-name").text
        except Exception as e:
            name = "Name Error"
            logger.error(f"{state}: Scraping page {page} Agent: agent_{index} Error getting name: {e}")

        # year_of_service on a marketplace
        try:
            service_div = card.find_element(By.CLASS_NAME, "ds-u-font-size--md")
            full_text = service_div.text.strip()
            years_of_service = full_text.split("\n")[0]
        except NoSuchElementException:
            years_of_service = ""
            # logger.warning(f"{state}: Scraping page {page} Agent: agent_{index} Year_of_service not found")

        # badges
        try:
            badges = [badge.text for badge in card.find_elements(By.CLASS_NAME, "ds-c-badge")]
        except NoSuchElementException:
            badges = ""
            # logger.warning(f"{state}: Scraping page {page} Agent: agent_{index} Bages not found")

        # phone
        try:
            phone = card.find_element(By.CLASS_NAME, "qa-flh-resource-phone").text
        except NoSuchElementException:
            phone = ""
            logger.warning(f"{state}: Scraping page {page} Agent: agent_{index} Phone not found")

        # email
        try:
            email = card.find_element(By.XPATH, './/a[contains(@href, "mailto:")]').text
        except NoSuchElementException:
            email = ""
            # logger.warning(f"{state}: Scraping page {page} Agent: agent_{index} Email not found")

        # website
        try:
            website = card.find_element(By.XPATH, './/a[contains(@href, "http")]').text
        except NoSuchElementException:
            website = ""
            # logger.warning(f"{state}: Scraping page {page} Agent: agent_{index} Website not found")

        # languages
        try:
            lang_row = card.find_element(
                By.XPATH, './/div[contains(@class, "ds-l-row") and .//span[text()="Languages spoken"]]')
            language_div = lang_row.find_elements(By.XPATH, './div')
            languages_spoken = language_div[1].text.strip() if len(language_div) > 1 else ""
        except NoSuchElementException:
            languages_spoken = ""
            # logger.warning(f"{state}: Scraping page {page} Agent: agent_{index} Languages not found")

        row = {
                "Page": page,
                "Agent_index": f"agent_{index}",
                "Name": name,
                "Years of Service": years_of_service,
                "Badges": ", ".join(badges),
                "Phone": phone,
                "Email": email,
                "Website": website,
                "Languages": languages_spoken,
                "State": state
        }

        all_data.append(row)

    # Save backup every N rows
    if len(all_data) % SAVE_EVERY == 0:
        part_data = all_data[len(all_data) - SAVE_EVERY:len(all_data) + 1]
        pd.DataFrame(part_data).to_csv(backup_file, index=False, sep="*", mode="a",
                                       header=not os.path.exists(backup_file))
        logger.info(f"{state}: Saved {len(all_data[len(all_data) - SAVE_EVERY:len(all_data) + 1])} agents "
                    f"into backup file {backup_file}")
            # Restart browser every M
            if len(all_data) % RESTART_BROWSER_EVERY == 0:

                logger.info("Restarting browser to clear memory...")

                driver.quit()
                driver = create_driver()
                wait = WebDriverWait(driver, 15)

            # Be polite
            time.sleep(random.uniform(0.8, 1.5))

        # Final save
        pd.DataFrame(all_data).to_csv(output_file, index=False, sep="*", mode="a", header=not os.path.exists(output_file))

        logger.info(f"Done! {state}: Pages: {start_page}-{finish_page} Agents: {len(all_data)} "
                    f"Data saved into {output_file}\n\n")

        driver.quit()

    # State_1
    # agents_from_state(URL_1, STATE_1, OUTPUT_FILE_1, BACKUP_FILE_1, 1, 500)
    # agents_from_state(URL_1, STATE_1, OUTPUT_FILE_1, BACKUP_FILE_1, 501, 1000)
    # agents_from_state(URL_1, STATE_1, OUTPUT_FILE_1, BACKUP_FILE_1, 1001, 1500)
    # agents_from_state(URL_1, STATE_1, OUTPUT_FILE_1, BACKUP_FILE_1, 1501, 2000)
    # agents_from_state(URL_1, STATE_1, OUTPUT_FILE_1, BACKUP_FILE_1, 2001, 2500)
    # agents_from_state(URL_1, STATE_1, OUTPUT_FILE_1, BACKUP_FILE_1, 2501, 3000)
    # agents_from_state(URL_1, STATE_1, OUTPUT_FILE_1, BACKUP_FILE_1, 3001, 3500)
    # agents_from_state(URL_1, STATE_1, OUTPUT_FILE_1, BACKUP_FILE_1, 3501, 4000)
    # agents_from_state(URL_1, STATE_1, OUTPUT_FILE_1, BACKUP_FILE_1, 4001, 4500)
    # agents_from_state(URL_1, STATE_1, OUTPUT_FILE_1, BACKUP_FILE_1, 4501, 5000)
    # agents_from_state(URL_1, STATE_1, OUTPUT_FILE_1, BACKUP_FILE_1, 5001, 5100)

    # State_2
    # agents_from_state(URL_2, STATE_2, OUTPUT_FILE_2, BACKUP_FILE_2, 1, 500)
    # agents_from_state(URL_2, STATE_2, OUTPUT_FILE_2, BACKUP_FILE_2, 501, 1000)
```