SaveHub Hybrid Architecture - Implementation Guide

Overview

This guide helps you transform your desktop PyQt5 app into a hybrid system with web dashboard capabilities, maintaining your existing PPSSPP integration while adding browser-based access.

Architecture Summary

File Structure

Your project should have this structure:

```
SaveTranslator/
├── gui/
│   ├── app_gui.py            (Enhanced PyQt5 app)
│   ├── local_server.py       (NEW - HTTP API server)
│   └── web_app.py            (NEW - Flask web dashboard)
├── core/
│   ├── launcher.py           (UPDATED - save state support)
│   ├── config.py             (Existing)
│   ├── psp_sfo_parser.py     (Existing)
│   └── game_map.py           (Existing)
├── controller/
│   └── converter.py          (Existing)
├── game_map.json             (Existing)
└── requirements.txt          (UPDATED)
```

Step-by-Step Implementation

Step 1: Update Requirements

Add these to `requirements.txt`:

```
PyQt5
Flask
flask-cors
```

Install:

```
bash

pip install -r requirements.txt
```

Step 2: Create Local Server (`gui/local_server.py`)

Copy the "Hybrid Local Agent Server" artifact I created. This provides:

- `/api/status` - Check if agent is running
- `/api/games` - Get all games with saves
- `/api/launch` - Launch PPSSPP with game
- `/api/icon/<disc_id>` - Serve game icons

- `/api/refresh` - Refresh game library

Step 3: Update Launcher (`core/launcher.py`)

Replace your existing `launcher.py` with the "Enhanced Launcher with Save State Support" artifact. Key changes:

- Added `save_state` parameter
- PPSSPP command: `--state=<path>` for save state loading
- Added `get_save_states_for_game()` helper

Step 4: Update PyQt5 App (`gui/app_gui.py`)

Replace with the "Enhanced PyQt5 App with Server" artifact. New features:

- Embedded HTTP server starts automatically
- Save state list display
- Auto-refresh every 5 seconds
- Better UI with game icons
- Launch modal for save selection

Step 5: Create Web Dashboard (`gui/web_app.py`)

Copy the "Flask Web App Server" artifact. This serves the web interface at `http://localhost:5000`.

Running the Hybrid System

Method 1: Desktop App Only (Local Use)

```
bash

python gui/app_gui.py
```

This starts:

1. PyQt5 GUI window
2. Local agent API server on port 8765
3. You can use the desktop interface OR...

4. Open browser to `http://localhost:5000` (need to run web_app.py separately)

Method 2: Full Web Experience

Terminal 1 - Start desktop agent:

```bash
python gui/app_gui.py
```

Terminal 2 - Start web dashboard:

```bash
python gui/web_app.py
```

Now you can:

- Use desktop app for local control
- Access web dashboard from any device on your network
- Both interfaces stay in sync

Configuration

1. Set PPSSPP Path

First time setup in desktop app:

1. Click "Set PPSSPP Path"
2. Select `PPSSPPWindows.exe` (or PPSSPPWindows64.exe)
3. Path saved to `~/.savetranslator_config.json`

2. Map Games to ISOs

Edit `game_map.json`:

```json
```

```
{
  "ULUS10565": "D:/Games/Tactics Ogre.iso",
  "ULES00851": "D:/Games/Crisis Core.iso"
}
```

3. PSP Save Location

Default: `~/Documents/PPSSPP/PSP/SAVEDATA`

If your saves are elsewhere, update `PSP_SAVEDATA_DIR` in `local_server.py`:

```python
PSP_SAVEDATA_DIR = "C:/Your/Custom/Path/PSP/SAVEDATA"
PSP_SAVESTATE_DIR = "C:/Your/Custom/Path/PSP/SYSTEM/savestates"
```

Features Implemented

Desktop App

- Browse and select PSP save folders
- View game icons (ICON0.PNG)
- Parse PARAM.SFO for game metadata
- List save states with timestamps
- Launch games with/without save states
- Auto-refresh save states every 5 seconds

Web Dashboard

- Visual game library with artwork
- Status indicator (online/offline)
- Game count display
- Click to launch games
- Save state selection modal
- Responsive design (works on mobile)

API Endpoints

- `GET /api/status` - Check local agent
- `GET /api/games` - List all games
- `POST /api/launch` - Launch with save state
- `GET /api/icon/<disc_id>` - Get game icon

Testing Workflow

1. **Add a PSP save folder**
   - Copy any PSP save to `~/Documents/PPSSPP/PSP/SAVEDATA/ULUS10565/`
   - Must contain PARAM.SFO and optionally ICON0.PNG
2. **Create save states**
   - Play game in PPSSPP
   - Press F2 to create save state
   - Files saved to `~/Documents/PPSSPP/PSP/SYSTEM/savestates/`
3. **Test desktop app**
   - Run `python gui/app_gui.py`
   - Select save folder
   - See save states listed
   - Click launch
4. **Test web dashboard**
   - Keep desktop app running
   - Run `python gui/web_app.py`
   - Open `http://localhost:5000`
   - Should see games and launch them

Troubleshooting

"Local Agent: Offline" in web dashboard

- Make sure `app_gui.py` is running
- Check if port 8765 is blocked by firewall
- Try accessing `http://127.0.0.1:8765/api/status` directly

Games not appearing

- Check PSP save directory path
- Ensure PARAM.SFO exists in save folders
- Click "Refresh Saves" button
- Check console for error messages

Launch fails

- Verify PPSSPP path is set correctly
- Check `game_map.json` has correct ISO paths
- Look at `launch.log` for debug info

Save states not showing

- Confirm saves are in `.ppst` format
- Check filename starts with disc ID
- Verify `PSP_SAVESTATE_DIR` path is correct

Next Steps (Future Enhancements)

Phase 2: Advanced Features

- [ ] File watcher to detect new save states immediately
- [ ] Thumbnail screenshots from save states
- [ ] Multiple emulator support (Dolphin, Citra)
- [ ] Save file upload from web interface

Phase 3: Cloud Integration

- [ ] Cloud storage backup (Google Drive)
- [ ] Cross-device sync

- [ ] User authentication
- [ ] Save sharing with friends

Phase 4: Academic Integration

- [ ] Usage analytics for thesis data
- [ ] User behavior tracking
- [ ] A/B testing different UIs
- [ ] Survey integration

Academic Thesis Connections

This project connects to your thesis topics:

AI Acceptance: How users adopt hybrid desktop/web interfaces

Online Identity: Managing game progress across platforms

Algorithms: Save file parsing, metadata extraction, file monitoring

Consider tracking:

- Desktop vs web usage patterns
- Launch frequency by interface
- Save state usage vs traditional saves
- Cross-platform accessibility impact

Support Files

The artifacts I created contain all the code. Save them as:

1. `local_server.py` → Hybrid Local Agent Server
2. `app_gui.py` → Enhanced PyQt5 App
3. `launcher.py` → Enhanced Launcher
4. `web_app.py` → Flask Web App Server

Questions?

Common modifications:

- **Add more emulators**: Extend `local_server.py` to scan other save formats
- **Custom styling**: Edit CSS in web dashboard HTML
- **Different ports**: Change 5000/8765 in code
- **Authentication**: Add Flask-Login to web_app.py

Good luck with your implementation!