🚀 Quick Start: Python Implementation

Current Project Structure

```
omnivox-api/
├── omnivox-crawler/          # ☑ KEEP - Working TypeScript crawler (reference)
├── archive/                  # ℹ ARCHIVED - Old implementations
│   ├── middleware/           # REST API (not needed)
│   ├── graphql-api/          # GraphQL API (not needed)
│   ├── website/              # Frontend (not needed)
│   └── api-bindings/         # TypeScript bindings (not needed)
├── README.md                 # Updated project documentation
└── PYTHON_PACKAGE_ROADMAP.md # Detailed roadmap for Python package
```

🎯 Your Mission: Create Python pip Package

Why Pure Python? ⭐

For your use case (pip package for other apps):

- ☑ No Node.js dependency for end users
- ☑ Native Python integration
- ☑ Simple `pip install omnivox-api`
- ☑ Better for Python ecosystem

Alternative (TypeScript Bridge):

- ⚠ Requires Node.js installed
- ⚠ Complex deployment
- ⚠ Subprocess overhead
- ☑ Faster initial development

📋 Step-by-Step Implementation Guide

1️⃣ Create Python Package Structure

```
cd "C:\Users\felix\OneDrive - Dawson College\Projects\omnivox-api"

# Create Python package directory
mkdir python-package
cd python-package

# Create virtual environment
python -m venv venv
venv\Scripts\activate
```

```
# Install dependencies
pip install requests beautifulsoup4 pydantic python-dotenv
pip install pytest black mypy  # Dev tools
```

2 Create Package Structure

```
# Create package structure
mkdir omnivox
cd omnivox
type nul > __init__.py
type nul > client.py
type nul > auth.py
type nul > exceptions.py
type nul > utils.py

mkdir lea
cd lea
type nul > __init__.py
type nul > manager.py
type nul > models.py
cd ..

mkdir mio
cd mio
type nul > __init__.py
type nul > manager.py
type nul > models.py
cd ..

cd ..
mkdir tests
cd tests
type nul > test_auth.py
type nul > test_lea.py
type nul > test_mio.py
```

🗺️ Porting Reference Map

TypeScript → Python File Mapping

TypeScript File	Python File	Purpose
`modules/Login.ts`	`omnivox/auth.py`	Authentication
`managers/LeaManager.ts`	`omnivox/lea/manager.py`	LEA operations
`managers/MioManager.ts`	`omnivox/mio/manager.py`	MIO operations

TypeScript File	Python File	Purpose
`modules/lea/Lea.ts`	`omnivox/lea/manager.py`	Parse class list
`modules/Mio.ts`	`omnivox/mio/manager.py`	Parse messages
`types/LeaClass.ts`	`omnivox/lea/models.py`	Data models
`utils/HTMLDecoder.ts`	`omnivox/utils.py`	HTML utilities

🔑 Key Porting Tasks

Task 1: Authentication (Day 1)

Reference: `omnivox-crawler/src/modules/Login.ts`

What to port:

1. Create `requests.Session()` (replaces axios with cookie jar)
2. GET login page
3. Extract hidden 'k' token from HTML
4. POST credentials
5. Check for success indicator in response

Python equivalent:

```
session = requests.Session()  # Automatically handles cookies
response = session.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
```

Task 2: LEA Manager (Days 2-3)

Reference: `omnivox-crawler/src/managers/LeaManager.ts`

What to port:

1. `getLeaCookie()` - Get authentication cookie
2. `getAllClasses()` - Parse class list from HTML
3. `getClass()` - Find specific class
4. `getClassDocumentSummary()` - Get document counts
5. `getClassDocuments()` - Get document list

Key HTML selectors to port:

- `.card-panel` - Class cards
- `.card-panel-title` - Course code/title
- `.card-panel-desc` - Section, schedule, teacher
- `.note-principale` - Grades

Task 3: MIO Manager (Days 4-5)

Reference: `omnivox-crawler/src/managers/MioManager.ts`

What to port:

1. `getMioCookies()` - Get MIO authentication
2. `loadMioPreview()` - Get message list
3. `loadMioById()` - Get full message
4. `getUserList()` - Search users
5. `sendMio()` - Send message

🧪 Testing Strategy

Create `.env` file for testing

```
# In python-package/.env
OMNIVOX_USERNAME=your_student_id
OMNIVOX_PASSWORD=your_password
```

Test script

```
# test_manual.py
from omnivox import OmnivoxClient
import os
from dotenv import load_dotenv

load_dotenv()

client = OmnivoxClient(
    os.getenv("OMNIVOX_USERNAME"),
    os.getenv("OMNIVOX_PASSWORD")
)

print("✅ Login successful!")
classes = client.lea.get_all_classes()
print(f"📚 Found {len(classes)} classes")

for cls in classes:
    print(f"  - {cls.code}: {cls.title}")
```

📦 Package Publishing (Final Step)

Create `setup.py`

```
from setuptools import setup, find_packages
```

```
setup(
    name="omnivox-api",
    version="0.1.0",
    packages=find_packages(),
    install_requires=[
        "requests>=2.31.0",
        "beautifulsoup4>=4.12.0",
        "pydantic>=2.0.0",
        "python-dotenv>=1.0.0",
    ],
    python_requires=">=3.8",
)
```

Build and publish

```
# Build
python setup.py sdist bdist_wheel

# Publish to PyPI
pip install twine
twine upload dist/*
```

🎯 Success Criteria

When done, users should be able to:

```
pip install omnivox-api
```

```
from omnivox import OmnivoxClient

# One-line authentication
client = OmnivoxClient("student_id", "password")

# Simple API calls
classes = client.lea.get_all_classes()
messages = client.mio.get_messages()

# Full type hints and IDE autocomplete
```

🆘 Need Help?

Resources:

1. **TypeScript reference code:** `omnivox-crawler/src/`
2. **Roadmap:** `PYTHON_PACKAGE_ROADMAP.md`
3. **HTML parsing:** Use BeautifulSoup (same as node-html-parser)
4. **HTTP requests:** Use requests.Session() (same as axios with cookies)

Common gotchas:

- HTML parsing: Omnivox HTML is messy, test selectors carefully
- Cookies: Use `requests.Session()` to auto-handle cookies
- Hidden fields: Many forms have hidden tokens (like 'k' in login)
- Error handling: Omnivox returns 200 OK even on errors

📊 Progress Tracker

- ☐ Day 1: Authentication working
- ☐ Day 2-3: LEA Manager (get classes, documents)
- ☐ Day 4-5: MIO Manager (messages, send)
- ☐ Day 6: Data models and utilities
- ☐ Day 7-8: Testing and bug fixes
- ☐ Day 9: Documentation and examples
- ☐ Day 10: Package and publish to PyPI

Start with authentication - once that works, everything else follows the same pattern! 🚀