Python Package Roadmap

Goal

Create `omnivox-api` as a pip-installable package that Python developers can use to interact with Omnivox programmatically.

Phase 1: Planning & Architecture

Decision: Pure Python vs. TypeScript Bridge

Option A: Pure Python Rewrite ⭐ RECOMMENDED

Structure:

```
omnivox-api/
├── omnivox/
│   ├── __init__.py
│   ├── client.py           # Main OmnivoxClient class
│   ├── auth.py             # Authentication logic
│   ├── lea/
│   │   ├── __init__.py
│   │   ├── manager.py      # LeaManager (classes, documents, grades)
│   │   └── models.py       # Data classes (LeaClass, Document, etc.)
│   ├── mio/
│   │   ├── __init__.py
│   │   ├── manager.py      # MioManager (messages, users)
│   │   └── models.py       # Data classes (Message, User, etc.)
│   ├── exceptions.py       # Custom exceptions
│   └── utils.py            # HTML parsing, helpers
├── tests/
│   ├── test_auth.py
│   ├── test_lea.py
│   └── test_mio.py
├── setup.py
├── requirements.txt
└── README.md
```

Dependencies:

```
requests>=2.31.0          # HTTP requests with session support
beautifulsoup4>=4.12.0    # HTML parsing
python-dotenv>=1.0.0      # Environment variables
pydantic>=2.0.0           # Data validation
```

Advantages:

- ☑ No Node.js dependency
- ☑ Native Python typing with type hints
- ☑ Standard pip installation
- ☑ Easier to maintain and test
- ☑ Better IDE support for Python developers

Disadvantages:

- ⏱ Requires porting ~1000 lines of TypeScript
- 🩹 Need to re-test all scraping logic

Option B: Python + TypeScript Bridge

Structure:

```
omnivox-api/
├── omnivox/
│   ├── __init__.py
│   ├── client.py           # Python wrapper
│   ├── bridge.py           # subprocess calls to Node.js
│   ├── _nodejs/            # Bundled omnivox-crawler
│   │   ├── node_modules/
│   │   ├── build/
│   │   └── package.json
│   └── models.py           # Python data models
├── setup.py
└── README.md
```

Advantages:

- ☑ Reuse existing tested TypeScript code
- ☑ Faster initial development (1-2 days)
- ☑ Less risk of bugs

Disadvantages:

- ❌ Users must have Node.js installed
- ❌ Larger package size (~50MB with node_modules)
- ❌ More complex deployment
- ❌ Subprocess overhead for each call
- ❌ Harder to debug cross-language issues

🚀 Phase 2: Implementation (Recommended: Pure Python)

Step 1: Setup Package Structure

```
mkdir omnivox
cd omnivox
python -m venv venv
source venv/bin/activate  # or venv\Scripts\activate on Windows
pip install requests beautifulsoup4 pydantic python-dotenv
```

Step 2: Port Authentication

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

```python
# omnivox/auth.py
import requests
from bs4 import BeautifulSoup
from typing import Optional

class OmnivoxAuth:
    BASE_URL = "https://dawsoncollege.omnivox.ca"
    LOGIN_URL = f"{BASE_URL}/intr/Module/Identification/Login/Login.aspx"

    def __init__(self):
        self.session = requests.Session()
        self.session.headers.update({
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36'
        })

    def login(self, username: str, password: str) -> bool:
        # Get login page to extract 'k' token
        response = self.session.get(self.LOGIN_URL)
        soup = BeautifulSoup(response.text, 'html.parser')

        # Extract hidden 'k' token
        k_value = self._extract_k_token(response.text)

        # Submit login
        login_data = {
            'NoDa': username,
            'PasswordEtu': password,
            'k': k_value
        }

        response = self.session.post(self.LOGIN_URL, data=login_data)
        return 'headerNavbarLink' in response.text

    def _extract_k_token(self, html: str) -> str:
        # Extract k value from HTML
        start = html.find('value="6') + len('value="')
        return html[start:start+18]
```

Step 3: Port LEA Manager

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

```python
# omnivox/lea/manager.py
from typing import List, Optional
from dataclasses import dataclass
from bs4 import BeautifulSoup

@dataclass
class LeaClass:
    code: str
    title: str
    teacher: str
    section: str
    schedule: List[str]
    grade: Optional[str] = None
    average: Optional[float] = None
    median: Optional[float] = None
    distributed_documents: int = 0
    distributed_assignments: int = 0

class LeaManager:
    LEA_URL = "https://www-daw-ovx.omnivox.ca/cvir/doce/Default.aspx"

    def __init__(self, session: requests.Session):
        self.session = session
        self._initialize()

    def _initialize(self):
        # Get LEA cookie
        cookie_url =
"https://dawsoncollege.omnivox.ca/intr/Module/ServicesExterne/Skytech.aspx?
IdServiceSkytech=Skytech_Omnivox&lk=%2festd%2fcvie&IdService=CVIE&C=DAW&E=P&L=ANG"
        self.session.get(cookie_url)

    def get_all_classes(self) -> List[LeaClass]:
        response = self.session.get(self.LEA_URL)
        soup = BeautifulSoup(response.text, 'html.parser')

        classes = []
        for card in soup.select('.card-panel'):
            cls = self._parse_class_card(card)
            classes.append(cls)

        return classes

    def _parse_class_card(self, card) -> LeaClass:
        # Parse HTML card to extract class info
        title_elem = card.select_one('.card-panel-title')
        code_title = title_elem.text.strip()
```

```
        # Split code and title
        code = code_title.split()[0]
        title = ' '.join(code_title.split()[1:])

        # Extract other fields...
        # (Port logic from Lea.ts)

        return LeaClass(
            code=code,
            title=title,
            # ... other fields
        )
```

Step 4: Port MIO Manager

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

Step 5: Create Main Client

```
# omnivox/client.py
from .auth import OmnivoxAuth
from .lea.manager import LeaManager
from .mio.manager import MioManager

class OmnivoxClient:
    def __init__(self, username: str, password: str):
        self.auth = OmnivoxAuth()

        if not self.auth.login(username, password):
            raise Exception("Login failed")

        self.lea = LeaManager(self.auth.session)
        self.mio = MioManager(self.auth.session)
```

Step 6: Package Setup

```
# setup.py
from setuptools import setup, find_packages

setup(
    name="omnivox-api",
    version="0.1.0",
    author="Your Name",
    description="Python API for Dawson College Omnivox",
    long_description=open("README.md").read(),
    long_description_content_type="text/markdown",
    url="https://github.com/yourusername/omnivox-api",
    packages=find_packages(),
    install_requires=[
```

```
        "requests>=2.31.0",
        "beautifulsoup4>=4.12.0",
        "python-dotenv>=1.0.0",
        "pydantic>=2.0.0",
    ],
    python_requires=">=3.8",
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
)
```

Phase 3: Testing

Unit Tests

```python
# tests/test_lea.py
import pytest
from omnivox import OmnivoxClient

def test_get_classes():
    client = OmnivoxClient("test_user", "test_pass")
    classes = client.lea.get_all_classes()
    assert len(classes) > 0
    assert classes[0].code is not None
```

Integration Tests

- Test with real Omnivox credentials (use .env)
- Verify HTML parsing doesn't break
- Check all endpoints

Phase 4: Distribution

PyPI Publishing

```
# Build package
python setup.py sdist bdist_wheel

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

Usage Example

```
pip install omnivox-api
```

```python
from omnivox import OmnivoxClient

# Initialize client
client = OmnivoxClient("2431927", "your_password")

# Get classes
classes = client.lea.get_all_classes()
for cls in classes:
    print(f"{cls.code}: {cls.title} - {cls.grade}")

# Get messages
messages = client.mio.get_messages()
for msg in messages:
    print(f"From {msg.author}: {msg.title}")
```

Phase 5: Future Features

1. **Async Support**

```python
import asyncio
client = await OmnivoxClient.create_async("user", "pass")
classes = await client.lea.get_all_classes()
```

2. **Caching**
   - Cache class data locally
   - Reduce API calls

3. **CLI Tool**

```
omnivox login
omnivox lea classes
omnivox mio messages
```

4. **Type Stubs**
   - Full type hints for IDE autocomplete

Time Estimates

Pure Python Approach

- **Phase 1 (Planning):** 1 day
- **Phase 2 (Implementation):** 5-7 days
  - Auth: 0.5 day
  - LEA Manager: 2 days
  - MIO Manager: 2 days
  - Client + Utils: 1 day
  - Setup/Packaging: 0.5 day
- **Phase 3 (Testing):** 2-3 days
- **Phase 4 (Distribution):** 0.5 day

Total: ~10-12 days

TypeScript Bridge Approach

- **Phase 1:** 1 day
- **Phase 2:** 2-3 days (just wrapper)
- **Phase 3:** 1-2 days
- **Phase 4:** 1 day (bundling Node.js deps)

Total: ~5-7 days

🎯 Recommendation

Go with Pure Python Rewrite for long-term maintainability and user experience, even though it takes longer initially. The TypeScript code is well-structured and serves as an excellent reference for porting.