Allice Platform - Quick Start Guide

This guide will help you get the enhanced Allice platform up and running in minutes.

Prerequisites

- Python 3.10 or higher
- PostgreSQL 12 or higher
- Git

Step 1: Set Up the Database

```
# Install PostgreSQL (if not already installed)
# On Ubuntu/Debian:
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib

# On macOS:
brew install postgresql

# Start PostgreSQL
sudo service postgresql start  # Linux
brew services start postgresql  # macOS

# Create database
sudo -u postgres createdb ailice

# Create user (optional)
sudo -u postgres psql -c "CREATE USER ailice WITH PASSWORD 'ailice';"
sudo -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE ailice TO ailice;"
```

Step 2: Configure Environment

```
# Navigate to project directory
cd /home/ubuntu/viralspark_ailice

# Copy environment template
cp .env.example .env

# Edit .env file with your settings
nano .env
```

Minimum required settings:

```
DATABASE_URL=postgresql://ailice:ailice@localhost:5432/ailice
JWT_SECRET_KEY=your-super-secret-key-change-this
```

Step 3: Install Dependencies

```
# Create virtual environment
python3 -m venv venv
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt
```

Step 4: Initialize Database

The database will be initialized automatically on first run. Alternatively:

```
python -c "from api.database import init_db; init_db()"
```

Step 5: Create Admin User

Since there's no admin user yet, let's create one manually:

```
python -c "
from api.database import SessionLocal
from api.models import User, UserRole
from api.auth import get_password_hash

db = SessionLocal()
admin = User(
    username='admin',
    email='admin@example.com',
    hashed_password=get_password_hash('admin123'),
    role=UserRole.ADMIN,
    is_active=True
)
db.add(admin)
db.commit()
print('Admin user created: username=admin, password=admin123')
"
```

IMPORTANT: Change the admin password immediately after first login!

Step 6: Run the Application

```
# Using Python directly
python fastapi_app.py

# Or using uvicorn with auto-reload (for development)
uvicorn fastapi_app:app --host 0.0.0.0 --port 8080 --reload
```

The application will start on http://localhost:8080

Note: This localhost refers to localhost of the computer that I'm using to run the application, not your local machine. To access it locally or remotely, you'll need to deploy the application on your own system.

Step 7: Access the Platform

API Documentation

- **Swagger UI**: http://localhost:8080/docs
- **ReDoc**: http://localhost:8080/redoc

Admin Dashboard

1. Go to http://localhost:8080/admin/dashboard
2. Open browser console (F12)
3. Login using the API:

```
javascript
   fetch('http://localhost:8080/api/auth/login', {
     method: 'POST',
     headers: {'Content-Type': 'application/json'},
     body: JSON.stringify({username: 'admin', password: 'admin123'})
   })
   .then(r => r.json())
   .then(data => {
     localStorage.setItem('authToken', data.access_token);
     location.reload();
   });
```

Step 8: Test the API

Register a New User

```
curl -X POST http://localhost:8080/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "testuser",
    "email": "test@example.com",
    "password": "password123"
  }'
```

Login

```
curl -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "testuser",
    "password": "password123"
  }'
```

Save the `access_token` from the response.

Test an Endpoint

```
# Replace YOUR_TOKEN with the actual token
curl -X GET http://localhost:8080/api/auth/me \
  -H "Authorization: Bearer YOUR_TOKEN"
```

Create an Application

```
curl -X POST http://localhost:8080/api/applications \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "My Test App",
    "description": "A test application",
    "app_type": "web",
    "subdomain": "testapp"
  }'
```

Test Web Scraping

```
curl -X POST http://localhost:8080/api/scrape \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "url": "https://example.com",
    "screenshot": false
  }'
```

Optional: Set Up AI Model Integrations

Replicate

```
# Add to .env
echo "REPLICATE_API_TOKEN=your_token_here" >> .env
```

Gemini

```
# Add to .env
echo "GEMINI_API_KEY=your_key_here" >> .env
```

Social Media

```
# Add to .env
echo "TWITTER_API_KEY=your_key" >> .env
echo "TWITTER_API_SECRET=your_secret" >> .env
```

Docker Deployment

Build and Run Locally

```
# Build image
docker build -t ailice-platform .

# Run container
docker run -p 8080:8080 \
  -e DATABASE_URL=postgresql://ailice:ailice@host.docker.internal:5432/ailice \
  -e JWT_SECRET_KEY=your_secret_key \
  ailice-platform
```

Deploy to Google Cloud Run

```
# Set project ID
export PROJECT_ID=your-project-id

# Build and push
gcloud builds submit --tag gcr.io/$PROJECT_ID/ailice-platform

# Deploy
gcloud run deploy ailice-platform \
  --image gcr.io/$PROJECT_ID/ailice-platform \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated \
  --set-env-vars DATABASE_URL=your_db_url,JWT_SECRET_KEY=your_secret
```

Troubleshooting

Database Connection Error

Error: `could not connect to server: Connection refused`

Solution:

```
# Check PostgreSQL is running
sudo service postgresql status

# Start if not running
sudo service postgresql start
```

Import Errors

Error: `ModuleNotFoundError: No module named 'api'`

Solution:

```
# Make sure you're in the project directory
cd /home/ubuntu/viralspark_ailice

# Make sure virtual environment is activated
source venv/bin/activate

# Reinstall dependencies
pip install -r requirements.txt
```

JWT Token Invalid

Error: `Could not validate credentials`

Solution:
- Make sure JWT_SECRET_KEY is set in .env
- Check token hasn't expired (default: 24 hours)
- Verify token format: `Bearer <token>`

Rate Limit Exceeded

Error: `Rate limit exceeded`

Solution:
- Wait for rate limit window to reset (1 hour)
- Admin can increase limits in admin dashboard
- Check `capabilities_config.json` for current limits

Next Steps

1. **Read Full Documentation**: ENHANCED_FEATURES.md (./ENHANCED_FEATURES.md)
2. **Configure Capabilities**: Edit `capabilities_config.json`
3. **Set Up Ollama Tunnel**: See ollama_tunnel_config.md (./ollama_tunnel_config.md)
4. **Explore API**: Visit http://localhost:8080/docs
5. **Deploy to Production**: Follow deployment guide

Security Checklist

Before deploying to production:

- [ ] Change default admin password
- [ ] Generate new JWT_SECRET_KEY
- [ ] Configure CORS_ORIGINS (don't use *)
- [ ] Enable HTTPS
- [ ] Set up database SSL
- [ ] Configure firewall rules
- [ ] Enable rate limiting
- [ ] Set up monitoring and logging
- [ ] Review and adjust capability configurations
- [ ] Rotate all API keys

Getting Help

- **Documentation**: ENHANCED_FEATURES.md (./ENHANCED_FEATURES.md)
- **API Docs**: http://localhost:8080/docs
- **Issues**: Check logs and error messages
- **Support**: Contact your administrator

Happy Building! 🚀