ONAYLF Fair Management System Documentation

Author: Kavon Hooshiar

Overview

The ONAYLF (Oklahoma Native American Youth Language Fair) Management System is a comprehensive web application designed to facilitate the organization and management of educational fairs focusing on indigenous language and cultural education. The system handles student registration, submission management, and fair administration through a user-friendly web interface.

Core Features

1. Users

Account Creation and Management

- **Registration Process**
  - Users create accounts with email and password
  - No email verification required before account activation
  - Initial profile setup required on first login
  - Program/School affiliation must be specified

User Profiles

- **Required Information**
  - Name
  - Email (verified)
  - Program/School affiliation
- **Additional Information**
  - Contact information
- **Program/School (aka Organization) Selection**
  - Semi-controlled vocabulary system
  - Users presented with existing Program/School list
  - "Other" option allows custom Program/School entry
  - Program/Schools stored as text field on user

Profile Review System

- **Mandatory Reviews**
  - New accounts should be confirmed by moderators, although use is not limited before confirmation
  - Users are redirected to their profile page on first login every calendar year and asked to review

Account Recovery

- **Password Reset**
  - Self-service password reset via email
  - Admin-assisted recovery when needed

Access Levels

- **Role-Based Permissions**
  - Administrators: Full system access
  - Moderators: Review and approval capabilities
  - Users: create submissions including students and instructors

Program/School (aka Organization) Management

- **Dynamic Program/School System**
  - Existing Program/Schools displayed in dropdown
  - Custom entry option with "Other" selection

Profile Updates

- **Change Management**
  - Users can update profile information
  - Program/School changes only affect current fair and future fairs

2. Submission System

Dynamic Configuration

- **Fair-Specific Settings**
  - Categories, tribes, languages, and accessories configurable per fair
  - Changes don't affect historical submissions
  - Moderators can modify options during fair setup

Categories

- **Category Properties**
  - Material/Non-Material designation
  - Maximum student limit

Submission Form Fields

- **Required Fields**
  - Title
  - Category
  - Grade Range
  - Language(s)
  - Instructors

    - Student Participants

- **Conditional Fields**
    - Accessories Count
    - Additional Language Details (when "Other" selected)

- **Automatically calculated fields**
    - Program/School
    - Grade range
    - Submission Type (Individual/Group/Both)

Dynamic Form Behavior

- Form adapts based on category selection

Submission Features

- **Status Management**
    - Pending review status
    - In progress/Submitted/Approved states

- **Tracking and History**
    - Models have updated dates as well as modified by

- **Moderator submission management**
    - Moderators can view a user's profile to add submissions on their behalf

3. Instructor and Student Management

Instructors and users are associated with a user as well as a specific fair. A user only needs to add an instructor or student once and they show up as available to add on any of that users submission's for that fair.

- **Instructor Profiles**
    - First name and last name only

- **Student Profiles**
    - Basic information (name, grade, location)
    - Tribal affiliations
    - T-shirt size tracking (which only displays as needed for non-material submissions)

- **Student Administration**
    - Moderators can view a list of students for the current fair
    - Remove students
    - View which submissions a student is in

    - Editing students can be done in the submission edit view

- **Instructor Administration**
    - Editing instructors can be done in the submission view

4. Fair Management

Administrative Features

- Fair creation and configuration
- Registration period control
- Category, languge, tribe, and accessory type management

5. Reporting System

Statistical Tracking

The system provides statistics for:

- Program participation
- Student engagement
- Submission categories
- Language representation
- Grade level distribution

Generated Reports

The system can produce various reports including:

- Submission cards
- Registration cover sheets
- Participation certificates
- Statistical summaries

Export Capabilities

- JSON data for fairs

5. Access Control

Limiting operations

- Restricted access based on user roles
- Limited actions for fairs not current
- Limited actions for current fair registration not open

View Mode System

- Can view non-current fairs

- Shows data for a specific fair that is not the current fair in the home page, student list page, and fair info page

Technical Implementation

Technology Stack

- **Backend Framework:** Django 5.2 LTS (supported until April 2028)
- **Frontend Framework:** Bootstrap 5
- **Database:** PostgreSQL 14+
- **JavaScript:** Interactive features and real-time updates

Key Components

- User authentication system
- RESTful API endpoints
- Real-time search and filtering
- Modal-based interactions
- Responsive design implementation

Installation and Setup

Prerequisites

- **Python:** 3.10 or higher (for local development)
- **PostgreSQL:** 14 or higher
- **Docker and Docker Compose:** For production deployment
- **Nginx Proxy Manager** - A separate Docker container that handles SSL certificates, domain routing, and reverse proxy. This runs outside of the ONAYLF repository and must be set up before deploying ONAYLF.
- Domain name configured (for production use with SSL)
- SSL certificate (automatically obtained through Nginx Proxy Manager via Let's Encrypt)

Environment Setup

1. **Create .env file in project root**

.env

```
SECRET_KEY="key"
DEBUG="False"
ALLOWED_HOSTS="127.0.0.1,localhost,django,onaylf.samnoblemuseum.ou.edu"
CSRF_TRUSTED_ORIGINS="https://onaylf.samnoblemuseum.ou.edu"
POSTGRES_DB="onaylfdjango"
POSTGRES_USER="postgres"
POSTGRES_PASSWORD="password"
DBHOST="onaylf_db"
DBPORT="port"
DJANGO_SETTINGS_MODULE=onaylf.settings
EMAIL_HOST="smtp-relay.brevo.com"
```

```
EMAIL_HOST_USER="id@smtp-brevo.com"
EMAIL_HOST_PASSWORD='password'
DEFAULT_FROM_EMAIL = 'onaylf.noreply@gmail.com'
ADMINS=[('name', 'email@gmail.com')]
DJANGO_LOG_LEVEL="INFO"
WORDS="word1,word2,etc"
```

Deployment Steps

1. **Clone the Repository**

bash

```
git clone [repository_url]
cd [project_directory]
```

The .env file should be in project_directory

2. **Optional: Restore Database from Backup**

- Place your PostgreSQL dump file in the `/backup` directory
- Name the file `backup.sql` or update the reference in `init-db.sh`
- The database will automatically initialize with this data on first run

3. **Start the Containers**

```
docker-compose up -d --build
```

4. **If you skipped database restore, initialize fresh database**

```
docker exec -it onaylf_django bash
python manage.py createsuperuser
```

for superuser, use email: admin@nal.ou.edu or else change it in build_initial_db.py

```
python manage.py build_initial_db
```

5. **Iteratively update containers, as needed**

```
git pull
docker-compose down
```

```
docker-compose up -d --build
```

6. **For easier debugging on initial deploy**

```
docker-compose down
docker-compose build
docker-compose up
```

This will start three containers:

- `onaylf_django`: Web application (port 8100)
- `onaylf_postgres`: Database
- `onaylf_nginx`: Nginx server (port 8181)

7. **Set Up Nginx Proxy Manager (Required for Production)**

Nginx Proxy Manager is a separate Docker container that runs outside of this repository. It handles SSL certificates, domain routing, and acts as a reverse proxy for the ONAYLF application.

If Nginx Proxy Manager is not already installed:

a. Create a separate directory for Nginx Proxy Manager (outside the ONAYLF project):

```
mkdir -p ~/nginxproxymanager
cd ~/nginxproxymanager
```

b. Create a `docker-compose.yml` file with the following content:

```
services:
  app:
    image: 'jc21/nginx-proxy-manager:latest'
    restart: unless-stopped
    ports:
      - '80:80'     # HTTP
      - '81:81'     # Admin interface
      - '443:443'   # HTTPS
    volumes:
      - ./data:/data
      - ./letsencrypt:/etc/letsencrypt

networks:
  default:
    external: true
    name: nginxproxymanager_default
```

c. Start Nginx Proxy Manager:

```
docker-compose up -d --build
```

d. Access the admin interface:

- URL: `http://your-server-ip:81` or `http://localhost:81`
- Default credentials: `admin@example.com` / `changeme`
- **Important:** Change these credentials immediately on first login

8. **Configure Nginx Proxy Manager for ONAYLF**

a. Log in to the Nginx Proxy Manager admin panel at `http://your-server-ip:81`

b. Navigate to "Proxy Hosts" and click "Add Proxy Host"

c. Configure the proxy host settings:

Details Tab:

- Domain Names: `yourdomain.com` (e.g., `onaylf.example.com`)
- Scheme: `http`
- Forward Hostname/IP: `onaylf_nginx` (this is the container name)
- Forward Port: `8181`
- Enable these options:
  - ✓ Cache Assets
  - ✓ Block Common Exploits
  - ✓ Websockets Support

SSL Tab:

- SSL Certificate: "Request a new SSL Certificate"
- Enable these options:
  - ✓ Force SSL
  - ✓ HTTP/2 Support
- Email Address: Your email (for Let's Encrypt certificate notifications)
- ✓ I Agree to the Let's Encrypt Terms of Service

d. Click "Save" to apply the configuration

e. The application will now be accessible at `https://yourdomain.com`

Security Best Practices:

- After initial setup, block external access to port 81 using your firewall
- Only open port 81 temporarily when making changes to Nginx Proxy Manager
- Keep ports 80 (HTTP) and 443 (HTTPS) open for web traffic
- Example firewall rules (using UFW):

  ```
  sudo ufw allow 80/tcp
  sudo ufw allow 443/tcp
  ```

```
sudo ufw deny 81/tcp  # Block external access to admin panel
```

9. **Network Configuration**

The ONAYLF application uses the `nginxproxymanager_default` Docker network to communicate with Nginx Proxy Manager. This network is automatically created when you start Nginx Proxy Manager with the docker-compose configuration shown above.

If you need to create the network manually for any reason:

```
docker network create nginxproxymanager_default
```

To verify the network exists:

```
docker network ls | grep nginxproxymanager
```

10. **Access and Verify the Application**

- **Production URL:** `https://yourdomain.com` (through Nginx Proxy Manager)
- **Direct Access (for testing):** `http://localhost:8181` (bypasses Nginx Proxy Manager)
- **Admin Interface:** `https://yourdomain.com/admin`

Verify that:

- SSL certificate is active (lock icon in browser)
- HTTP requests redirect to HTTPS
- Static files load correctly
- Admin interface is accessible

Container Architecture

The ONAYLF application runs in a multi-container Docker environment:

ONAYLF Containers (in this repository):

- **onaylf_django**: Django application server running Gunicorn (port 8100, internal only)
- **onaylf_postgres**: PostgreSQL database (internal only)
- **onaylf_nginx**: Nginx reverse proxy (exposes port 8181)

External Container (separate repository):

- **Nginx Proxy Manager**: Handles SSL, domain routing, and public-facing reverse proxy (ports 80, 443, and admin port 81)

Network Configuration:

- All containers communicate via the `nginxproxymanager_default` Docker network
- This allows Nginx Proxy Manager to forward requests to `onaylf_nginx:8181`

- Static files are served through a shared Docker volume between Django and Nginx containers
- PostgreSQL data persists in a named Docker volume
- Automatic database initialization on first run via `init-db.sh`
- Health checks ensure proper startup sequence

Traffic Flow:

1. User → `https://yourdomain.com` (port 443)
2. Nginx Proxy Manager (external container) → `onaylf_nginx:8181`
3. `onaylf_nginx` → `onaylf_django:8100` (for dynamic content)
4. `onaylf_nginx` → serves static files directly from shared volume

Health Checks

- Database: Checks PostgreSQL readiness
- Web application: Verifies HTTP response
- Automatic restart on failure

Backup and Restore

Creating a Database Backup

To output a dump of the database in the current directory:

```
docker exec -i onaylf_postgres /bin/bash -c "PGPASSWORD=password pg_dump --username postgres onaylfdjango" > backup.sql
```

Restoring a Database Backup

To restore a backup, you will destroy the current deployment and redeploy with the new backup file.

In the project folder, take down the deployment

```
docker-compose down
```

Make sure none of the containers are running

```
docker ps
```

Destroy the containers and volumes (note this will affect any downed containers in the same namespace)

```
docker system prune -a --volumes
```

Place the new backup file in the /backup folder, and make sure it's named backup.sql. Then deploy again.

```
docker-compose up -d --build
```

Troubleshooting

Nginx Proxy Manager Connection Issues

Problem: Can't access the application through the domain, but `http://localhost:8181` works.

Solutions:

1. Verify Nginx Proxy Manager is running:

   ```
   docker ps | grep nginx-proxy-manager
   ```

2. Check that both containers are on the same network:

   ```
   docker network inspect nginxproxymanager_default
   ```

   You should see both `nginx-proxy-manager` and `onaylf_nginx` in the containers list.

3. Verify the proxy host configuration in Nginx Proxy Manager:

   - Forward Hostname/IP should be `onaylf_nginx` (not localhost or IP address)
   - Forward Port should be `8181`
   - Scheme should be `http`

4. Check Nginx Proxy Manager logs:

   ```
   docker logs nginx-proxy-manager
   ```

SSL Certificate Issues

Problem: SSL certificate not working or browser shows security warning.

Solutions:

1. Verify your domain's DNS points to your server's IP address
2. Ensure ports 80 and 443 are open in your firewall
3. In Nginx Proxy Manager, delete and recreate the SSL certificate
4. Check Let's Encrypt rate limits (5 certificates per domain per week)

Database Connection Issues

Problem: Application shows database connection errors.

Solutions:

1. Verify PostgreSQL container is running:

   ```
   docker ps | grep onaylf_postgres
   ```

2. Check database logs:

   ```
   docker logs onaylf_postgres
   ```

3. Verify `.env` file has correct database credentials:

   - `DBHOST=onaylf_db` (must be the container name for Docker)
   - `POSTGRES_DB`, `POSTGRES_USER`, `POSTGRES_PASSWORD` must match

Static Files Not Loading

Problem: CSS/JavaScript/images not loading, site looks unstyled.

Solutions:

1. Check if static files were collected:

   ```
   docker exec -it onaylf_django ls -la /app/static
   ```

2. Restart the containers to trigger static file collection:

   ```
   docker-compose restart
   ```

3. Manually collect static files:

   ```
   docker exec -it onaylf_django python manage.py collectstatic --no-
   input
   ```

Container Won't Start

Problem: Containers fail to start or repeatedly restart.

Solutions:

1. Check container logs:

```
docker-compose logs -f
```

2. Verify the `nginxproxymanager_default` network exists:

```
docker network ls | grep nginxproxymanager
```

If not, create it:

```
docker network create nginxproxymanager_default
```

3. Remove and rebuild containers:

```
docker-compose down
docker-compose up -d --build
```

Port Conflicts

Problem: Cannot start containers due to port already in use.

Solutions:

1. Check what's using the port:

```
sudo lsof -i :8181  # For onaylf_nginx
sudo lsof -i :80    # For Nginx Proxy Manager
sudo lsof -i :443   # For Nginx Proxy Manager
```

2. Stop conflicting services or modify `docker-compose.yml` to use different ports

---

This documentation is a living document and will be updated as the system evolves.