Standard Operating Procedure

Project: CYTOCHALDB

Introduce

Cytochaldb is a web-based application that uses Flask for the server side and React for the client side to present molecular reports. The purpose of this SOP is to provide clear instructions for installing, running, deploying, monitoring, and troubleshooting cytochaldb in development and production environments.

System Overview

- **Backend Framework:** Flask (Python 3.10.7)
- **WSGI Server:** Gunicorn
- **Frontend Framework:** React with TypeScript (built using Vite)
- **Containerization:** Docker and docker-compose
- **Reverse Proxy / Web Server:** Nginx
    - Serving the React frontend
    - Forwarding API calls to the Flask backend
    - Handling health checks
    - Providing basic security headers
    - Enabling gzip compression and access/error logging
- **Data Storage:** File-based JSON data (`backend/static/data.json`)
- **Main Entry Point (Backend):** `backend/app.py`
- **Health Check Endpoint:** `GET /api_cytochal/health`

Project Structure

```
cyto_db_python-main/  # Project root folder for cytochadb
├── backend/           # Flask backend API, data processing, search logic
├── frontend/          # React + TypeScript frontend (user interface)
├── nginx/             # Nginx config for serving frontend + proxying API to backend
├── docker-compose.yml # Defines and runs all services (backend, frontend, nginx) with Docker
└── .env.example       # Example environment configuration; copy to .env and fill in real values
```

Configuration and set up

1.1)create .env in root directory

The root `.env` file stores key settings for backend and frontend (host, port, API base path, data file paths), and you can use `env.example` as a reference for the keys.

1.2 Prepare static directory, image folders, and SDF file

Procedure

1. **From the project root, go to the backend static directory and ensure the image folders exist (create them if missing.**

```
cd backend/static
mkdir "microscopy_images"
mkdir "molecule_structures_image"
```

2. **Your SDF file must be located in the `backend/static` directory and must be named CytoLabs_Database.sdf. If you want to use a different file name or store it in another directory, you must also update the corresponding path in the `.env` file so that the application can still find it.**

3. **The file data.json must be in the backend/static directory. It is the JSON version of the SDF data used by the web app. If you update the SDF file and want new data, delete data.json and run the parsing script backend/cyto_db_shiny_app/parse_sdf.py again.**

Update and generate `data.json`

Procedure

1. **Make sure the SDF file and image folders are already set up as described in the *Configuration and set up* section**
2. **From the project root, go to the parser script directory and Run the parser script:**

```
cd backend/cyto_db_shiny_app
python parse_sdf.py
```

3. **After the script finishes, confirm that a fresh data.json has been created in:`backend/static/data.json`.**

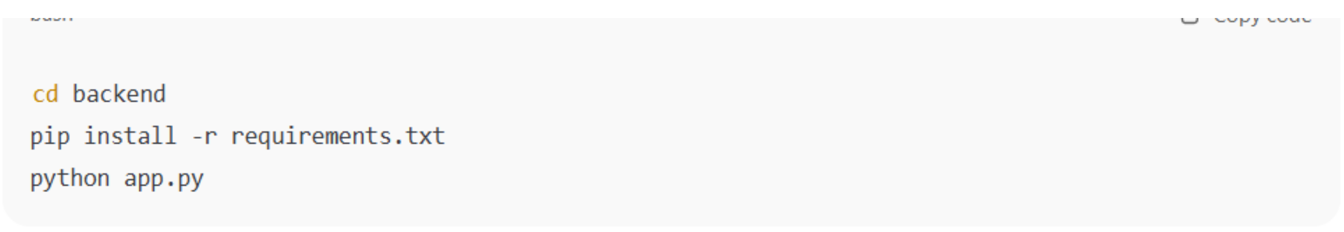
Notes

- **The script reads the image paths dynamically from `.env` (with fallbacks). If you want to customize or move the image folders, update `CYTOCHAL_IMAGE_DIR=static/microscopy_images` and `CYTOCHAL_STRUCTURE_DIR=static/molecule_structures_image` in `.env`.**

Development phase (run locally)

For developing the application it is recommended to run the backend and frontend locally.
This makes debugging and testing easier, and you do not need Nginx in this phase.

3.1 Run backend locally



```bash
cd backend
pip install -r requirements.txt
python app.py
```

Procedure

1. **Go to the backend folder, optionally install dependencies, and start the backend (main entry point is [`app.py`]).**

2. **The API should now be available on the local backend port (for example http://localhost:5000/cytochal-api/).**

Backend folder structure (for reference)

```
backend/
├── cyto_db_shiny_app/    # Scripts to convert the SDF file to data.json (e.g. parse_sdf.py)
├── search_index/         # Search index for compounds (Whoosh library) for faster search
├── static/               # SDF file, data.json, and image folders
├── app.py                # Main Flask application
├── Backend_Dockerfile    # Dockerfile for backend (used in production)
├── filters.py            # Filter helper functions used by the backend
├── params.py             # Configuration parameters used in the app
├── search_engine.py      # Search logic using data.json / search_index
├── start_app.sh          # Script to run parser + start backend (production)
└── requirements.txt      # Python dependencies
```

3.2 Run frontend locally

The frontend is built with React and TypeScript and requires Node.js (recommended version ≥ 20).

Procedure

1. **From the project root ,check that Node.js and npm are installed and the version is OK:**

```
node --version
npm --version
```

2. **In the root `.env` file, enable development mode for local runs:**

```
CYTOCHAL_IS_DEV=true
```

3. **Go to the frontend folder, install dependencies, and start the dev server:**

```
cd frontend
npm install
npm run dev
```

3. **Access the frontend in your browser (http://localhost:3000/cytochal/).**

Frontend folder structure (for reference):

```
frontend/                 # React + TypeScript frontend (UI)
├── src/                  # Main source code (components, pages, hooks, styles, etc.)
├── Frontend_Dockerfile   # Dockerfile for building the frontend image (production)
├── index.html            # HTML entry file that mounts the React app
├── package.json          # Frontend dependencies and npm scripts (dev, build, preview)
├── package-lock.json     # Locked versions of all Node dependencies (created by npm)
└── vite.config.ts        # Vite configuration (dev server, build options, proxies)
```

Run locally using docker

For production it is recommended to run the application with Docker. This makes the whole project easy to deploy and access in the browser.
In this project there are three services:

- backend – Flask API
- frontend – built React app (static files)
- nginx – reverse proxy and web server

The browser only communicates with nginx. Nginx serves the frontend files and forwards API requests internally to the backend container, so users never talk directly to the backend or frontend containers.

Procedure

1. **Ensure Docker and docker-compose are installed:**

```
cd cyto_db_python-main
docker-compose up -d --build
```

2. **From the project root (`cyto_db_python-main`), build and start all services:**

```
docker --version
docker-compose --version
```

3. **Check that all containers are running ,all services (`backend`, `frontend`, `nginx`) should show status Up.**

```
docker-compose ps
```

4. **Access the application in a browser**
    - On your local machine: http://localhost/cytochal/

Deploy (run on server)

In the previous section (Run locally using Docker), we set up and tested the Docker-based stack on a local machine. For deployment, we use the same Docker setup, but run it on a remote server so other users can access the application in their browsers. The following steps describe how to connect to the server and start the Docker services there.

Procedure

1. **Connect to the server via SSH,replace `<user>` with your server username and `<server-address>` with the server's IP or domain.**

```
bash                                                          Copy code

ssh <user>@<server-address>
```

2. **make sure Docker and docker-compose are installed and the firewall allows ports 80/443**

```
docker --version
docker-compose --version
```

3. **Run the project with Docker on the server ,on the server, install Docker and docker-compose (if not already installed), then go to the project directory and start the stack**

```
# (Only once) install Docker and docker-compose according to your OS docs

cd /path/to/cyto_db_python-main    # go to the project root on the server
docker-compose up -d --build       # build and start all services in the background
docker-compose ps                  # check that backend, frontend, and nginx are all "Up"
```

4. **Access the application in a browser:**

   - On a remote server: **http://<server-address>/cytochal/**

Replace \<server-address\> with the server's IP address or domain name (for example, http://192.168.1.10/cytochal/ or http://myserver.example.com/cytochal/).