Aletheia – Local Development Setup (Windows & macOS)

Repo Link: https://github.com/akhil1729/llm

Repo layout (single Git repo named llm/):

```
llm/
  frontend/   # Next.js + Tailwind
  backend/    # FastAPI + PostgreSQL + Alembic
  README.md
```

---

0) What you'll need to run

- **PostgreSQL** (local)
- **Python 3.11** (backend)
- **Node.js 20 LTS** (frontend)
- **Git**
- (Optional) **pgAdmin** (GUI for Postgres)

You'll run:

- Backend (FastAPI) at http://127.0.0.1:8000
- Frontend (Next.js) at http://localhost:3000

A) Windows 10/11 Setup (step-by-step)

A1. Install prerequisites

1. **Git**

- Download: https://git-scm.com/download/win

- During install, accept defaults.

2. **Python 3.11 (64-bit)**

- Download: https://www.python.org/downloads/release/python-3110/
- Check “Add python.exe to PATH”.

3. **Node.js 20 LTS**

- Download: https://nodejs.org/en/download (LTS)

Confirm with:

```
node -v
npm -v
```

-

4. **PostgreSQL 15 or 16 + pgAdmin**

- Download: https://www.postgresql.org/download/windows/
- During install, set:
    - Superuser: postgres
    - Password: (choose one, e.g., postgres)
- pgAdmin is included; it’s helpful for DB creation & browsing.

A2. Clone the repo

Choose a working folder (e.g., C:\dev\).

```
cd C:\dev
git clone <YOUR_REPO_URL> llm
cd llm
```

You should see the frontend/ and backend/ inside C:\dev\llm\.

A3. Create the PostgreSQL database

Option 1 – Using pgAdmin (GUI)

1. Open **pgAdmin** → connect to localhost with the password you set.
2. Right-click **Databases** → **Create** → **Database…**
    - Name: aletheia
    - Owner: postgres (or a new role you create)
3. (Optional, recommended) Create a dedicated DB user:
    - **Login/Group Roles** → **Create** → **Login/Group Role…**
        - Name: aletheia_user
        - Definition → Password: aletheia_pass
        - Privileges: Can login
    - Grant privileges:
        - Right-click **aletheia** → **Grant Wizard** → grant CONNECT/CREATE/USAGE as needed to aletheia_user.

Or run SQL in pgAdmin query tool:

```
 CREATE USER aletheia_user WITH PASSWORD 'aletheia_pass';
GRANT ALL PRIVILEGES ON DATABASE aletheia TO aletheia_user;
```

-

Option 2 – Using psql (CLI)

```
# Open "SQL Shell (psql)" from Start menu or use psql in terminal:
psql -U postgres

-- In psql:
CREATE DATABASE aletheia;
CREATE USER aletheia_user WITH PASSWORD 'aletheia_pass';
GRANT ALL PRIVILEGES ON DATABASE aletheia TO aletheia_user;
\q
```

A4. Backend setup (FastAPI)

Folder: C:\dev\llm\backend

1. Create and activate a virtual environment:

```
cd C:\dev\llm\backend
py -3.11 -m venv .venv
.venv\Scripts\activate
```

2. Install dependencies:

```
pip install --upgrade pip
pip install -r requirements.txt
```

If requirements.txt doesn't exist, typical deps are:

```
fastapi
uvicorn[standard]
python-dotenv
pydantic
SQLAlchemy
psycopg2-binary
alembic
passlib[bcrypt]
python-jose[cryptography]
```

3. Create the backend environment file:

- **File:** C:\dev\llm\backend\.env

```
# === FastAPI/Aletheia backend ===
APP_ENV=local
SECRET_KEY=change_this_to_a_random_long_string

# Allowed CORS origins for local Next.js dev server
ALLOW_ORIGINS=http://localhost:3000

# Postgres connection (pick one):
# If you created a dedicated user:
DATABASE_URL=postgresql+psycopg2://aletheia_user:aletheia_pass@localhost:5432/aletheia
# Or, using the default postgres superuser (not recommended long-term):
#
DATABASE_URL=postgresql+psycopg2://postgres:<YOURPASSWORD>@localhost:5432/alet
heia

# If your app uses an external LLM key (e.g., Gemini/OpenAI), add it here:
# GEMINI_API_KEY=...
# OPENAI_API_KEY=...

# Personality assignment mode
MODEL_ASSIGNMENT_MODE=round_robin
```

```
# Logging Google logo clicks (feature flag, if used)
ENABLE_GOOGLE_CLICK_LOGGING=true
```

4. Run database migrations:

```
# still inside backend venv
alembic upgrade head
```

- If Alembic complains it can't find alembic.ini or env, ensure you're in the backend/ folder and that alembic.ini + alembic/ folder exist.
- If you changed DATABASE_URL, re-run with the right value.

5. Start the backend API:

```
uvicorn main:app --reload --host 127.0.0.1 --port 8000
```

- Visit http://127.0.0.1:8000/docs to confirm it's running.

A5. Frontend setup (Next.js)

Folder: C:\dev\llm\frontend

1. Install packages:

```
cd C:\dev\llm\frontend
npm install
```

2. Create the frontend environment file:

- **File:** C:\dev\llm\frontend\.env.local

```
# === Next.js frontend ===
# Point to your local backend:
NEXT_PUBLIC_API_BASE_URL=http://127.0.0.1:8000

# Your app likely just links out:
NEXT_PUBLIC_GOOGLE_URL=https://www.google.com
```

```
# If you display which personality is assigned (optional flag)
NEXT_PUBLIC_SHOW_PERSONALITY=true
```

3. Run the dev server:

```
npm run dev
```

- Open http://localhost:3000.

  If you see CORS errors in the browser console, confirm ALLOW_ORIGINS=http://localhost:3000 is set in backend/.env and that the backend was restarted after editing.

A6. (Optional) Create a local admin/test user

If your backend exposes a registration route, call it via Swagger UI:

- Go to http://127.0.0.1:8000/docs
- Find POST /auth/register (or your app's equivalent)
- Create a test user (e.g., test@example.com / 12345678)

If you seeded an initial user via migration or script, note it here. Otherwise, standard app flows (signup → demographics → etc.) should work.

B) macOS Setup (step-by-step)

B1. Install prerequisites

1. **Homebrew** (recommended)

Open Terminal and install:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

-
- Add brew to your PATH if prompted by the installer output.

2. **Git**

```
brew install git
```

3. **Python 3.11**

```
brew install python@3.11
python3.11 --version
```

4. **Node.js 20 LTS**

```
brew install node@20
node -v
npm -v
```

5. **PostgreSQL 15 or 16**

```
brew install postgresql@16
brew services start postgresql@16
```

- (Optional) **pgAdmin** on macOS can be installed via dmg from the official website: https://www.pgadmin.org/download/

B2. Clone the repo

Pick a dev folder (e.g., ~/dev):

```
mkdir -p ~/dev && cd ~/dev
git clone <YOUR_REPO_URL> llm
cd llm
```

You should have frontend/ and backend/ inside ~/dev/llm/.

B3. Create the PostgreSQL database

Using psql:

```
createdb aletheia
psql -d postgres -c "CREATE USER aletheia_user WITH PASSWORD 'aletheia_pass';"
psql -d postgres -c "GRANT ALL PRIVILEGES ON DATABASE aletheia TO aletheia_user;"
```

(If you installed Postgres via Homebrew, psql should already be on your PATH. If not, run brew link postgresql@16 --force.)

B4. Backend setup (FastAPI)

Folder: ~/dev/llm/backend

1. Create and activate a virtual environment:

```
cd ~/dev/llm/backend
python3.11 -m venv .venv
source .venv/bin/activate
```

2. Install dependencies:

```
pip install --upgrade pip
pip install -r requirements.txt
```

3. Create the backend environment file:

- **File:** ~/dev/llm/backend/.env

```
APP_ENV=local
SECRET_KEY=change_this_to_a_random_long_string
ALLOW_ORIGINS=http://localhost:3000

DATABASE_URL=postgresql+psycopg2://aletheia_user:aletheia_pass@localhost:5432/aletheia

# GEMINI_API_KEY=...
# OPENAI_API_KEY=...

MODEL_ASSIGNMENT_MODE=round_robin
ENABLE_GOOGLE_CLICK_LOGGING=true
```

4. Run Alembic migrations:

```
alembic upgrade head
```

5. Start the backend API:

```
uvicorn main:app --reload --host 127.0.0.1 --port 8000
```

Open http://127.0.0.1:8000/docs to verify.

B5. Frontend setup (Next.js)

Folder: ~/dev/llm/frontend

1. Install packages:

```
cd ~/dev/llm/frontend
npm install
```

2. Create the frontend environment file:

- **File:** ~/dev/llm/frontend/.env.local

```
NEXT_PUBLIC_API_BASE_URL=http://127.0.0.1:8000
NEXT_PUBLIC_GOOGLE_URL=https://www.google.com
NEXT_PUBLIC_SHOW_PERSONALITY=true
```

3. Run the dev server:

```
npm run dev
```

Open http://localhost:3000.

C) Working across Frontend & Backend

- Keep both servers running:
    - **Backend**: uvicorn main:app --reload --host 127.0.0.1 --port 8000

    - **Frontend**: npm run dev

- Frontend talks to the backend via NEXT_PUBLIC_API_BASE_URL.
- **CORS**: If you change the frontend port or URL, update ALLOW_ORIGINS in backend/.env accordingly and restart the backend.

D) Database Migrations (Alembic)

Creating a new migration (after editing SQLAlchemy models):

```
 # from backend/ (venv active)
alembic revision -m "describe your change" --autogenerate
alembic upgrade head
```

-

Downgrade (if needed):

```
 alembic downgrade -1
```

-

E) Common Environment Variables (Backend)

Add these to backend/.env if your project uses them:

```
# Auth & security
SECRET_KEY=change_me

# CORS (comma-separated list allowed):
ALLOW_ORIGINS=http://localhost:3000,http://127.0.0.1:3000

# DB
DATABASE_URL=postgresql+psycopg2://USER:PASS@HOST:5432/DBNAME

# Personality assignment
MODEL_ASSIGNMENT_MODE=round_robin   # fixed per user (maps to your utils.py logic)

# Logging features
ENABLE_GOOGLE_CLICK_LOGGING=true

# External keys (fill only if used)
# GEMINI_API_KEY=...
# OPENAI_API_KEY=...
```

F) Seeding / Test Data (Optional)

If you keep a seed script (example backend/scripts/seed.py), run:

```
cd backend
source .venv/bin/activate   # macOS/Linux
# .venv\Scripts\activate    # Windows PowerShell
python scripts/seed.py
```

Document what it inserts (e.g., demo users, example tasks).

G) Verifying the Flow

1. Start backend: http://127.0.0.1:8000/docs should load
2. Start frontend: http://localhost:3000
3. Go through: **Signup →login→ Demographics → Background → Tutorial → Task 1/2/3 → Post-task survey → Debrief**
4. Confirm DB writes:
    - Users
    - ModelAssignment (personality fixed per user)
    - Chats (per message logs)
    - Google logo click logs (if the /google-click endpoint exists)
    - Final answers

H) Troubleshooting

- **CORS errors**: Ensure ALLOW_ORIGINS includes the exact frontend origin (http://localhost:3000). Restart backend.

psycopg2 install issues (Windows): Install the precompiled wheel or ensure Visual C++ Build Tools are present. Try:

```
pip install psycopg2-binary
```

-
- **alembic can't find config**: Run commands from backend/ where alembic.ini lives.
- **DB auth errors**: Re-check DATABASE_URL credentials and that the DB/user exists.
- **Port conflicts**: Change ports:
    - Backend: uvicorn main:app --reload --port 8001
    - Frontend: PORT=3001 npm run dev (or set "dev": "next dev -p 3001" in package.json) and update NEXT_PUBLIC_API_BASE_URL.

I) Exact Files to Create/Edit

backend/.env

```
APP_ENV=local
SECRET_KEY=change_this
ALLOW_ORIGINS=http://localhost:3000
DATABASE_URL=postgresql+psycopg2://aletheia_user:aletheia_pass@localhost:5432/aletheia
MODEL_ASSIGNMENT_MODE=round_robin
ENABLE_GOOGLE_CLICK_LOGGING=true
```

frontend/.env.local

```
NEXT_PUBLIC_API_BASE_URL=http://127.0.0.1:8000
NEXT_PUBLIC_GOOGLE_URL=https://www.google.com
NEXT_PUBLIC_SHOW_PERSONALITY=true
```

Run commands (Windows)
Backend:

```
cd C:\dev\llm\backend
py -3.11 -m venv .venv
.venv\Scripts\activate
pip install -r requirements.txt
alembic upgrade head
uvicorn main:app --reload --host 127.0.0.1 --port 8000
```

Frontend:

```
cd C:\dev\llm\frontend
npm install
npm run dev
```

-

Run commands (macOS)

Backend:

```
cd ~/dev/llm/backend
python3.11 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
alembic upgrade head
uvicorn main:app --reload --host 127.0.0.1 --port 8000
```

Frontend:

```
cd ~/dev/llm/frontend
npm install
npm run dev
```

FOR REAL-TIME DATA COLLECTION:

1. Open pgadmin4 PORTAL
2. Go to Query Tool workspace where you can see Server Connectivity options
3. Use the below credentials to connect to the SQL server and extract data using queries

   Server Name: Any name

   Host name/address: dpg-d1vpq92li9vc73fpeq7g-a.virginia-postgres.render.com

   Port: 5432

   Database: llm_db_gp54

   User: llm_db_gp54_user

   Password: DQ2Cp32trJrbc475ZQT3xGzGiUhka8FA

   Connection Parameters:

   SSL mode: Set it to require

4. As we have tables named : chats, consent_log,demographics, final_answers, google_clicks, model_assignment, surveys, users, alembic_version(not required) we can run basic SQL queries for viewing the tables e.g(select * from users)
5. For Getting the derived tables user_summary and task_summary data, I made few SQL queries to combine above tables , run this queries in the editor

user_summary:

```
DROP VIEW IF EXISTS user_summary CASCADE;

CREATE VIEW user_summary AS
WITH task_rollup AS (
  SELECT
    ts.user_id,
    SUM(ts.total_dialogue_turns)          AS total_dialogue_turns_3tasks,
    SUM(ts.total_time_seconds)            AS total_task_duration_seconds,
    SUM(ts.total_google_clicks)           AS total_google_clicks_3tasks,
    SUM(ts.total_hallucinated_responses)    AS total_hallucinated_responses_3tasks
  FROM task_summary ts
  GROUP BY ts.user_id
)
SELECT
  u.id                              AS user_id,
  u.llm_version                         AS personality_index,
```

```
  d.age,
  d.identity1,
  d.identity2,
  d.education,
  d.college_major,
  d.chatbot_usage,


  s.trust_answers,
  s.verify_needed,
  s.comfort_communication,
  s.reuse_chatbot,
  s.comments,


  NULL::DECIMAL(5,2)                    AS mean_answer_correctness,

  COALESCE(tr.total_dialogue_turns_3tasks, 0)        AS total_dialogue_turns_3tasks,
  COALESCE(tr.total_task_duration_seconds, 0)        AS total_task_duration_seconds,
  COALESCE(tr.total_google_clicks_3tasks, 0)         AS total_google_clicks_3tasks,
  COALESCE(tr.total_hallucinated_responses_3tasks, 0) AS
total_hallucinated_responses_3tasks,

  NULL::INT                             AS total_user_verification_queries_3tasks

FROM users u
LEFT JOIN demographics d
  ON d.user_id = u.id
LEFT JOIN surveys s
  ON s.user_id = u.id
LEFT JOIN task_rollup tr
  ON tr.user_id = u.id;
```

task_summary:

```
DROP VIEW IF EXISTS task_summary CASCADE;

CREATE VIEW task_summary AS
WITH chat_agg AS (
  SELECT
    c.user_id,
```

```
    c.task_number                       AS task_id,
    COUNT(*)                            AS total_dialogue_turns,
    SUM(CASE WHEN c.was_hallucinated THEN 1 ELSE 0 END) AS
total_hallucinated_responses,
    MIN(c.timestamp)                    AS first_chat_ts,
    MAX(c.timestamp)                    AS last_chat_ts,
    MIN(c.personality_index)            AS personality_index_from_chat
  FROM chats c
  GROUP BY c.user_id, c.task_number
),
click_agg AS (
  SELECT
    gc.user_id,
    gc.task_number                      AS task_id,
    COUNT(*)                            AS total_google_clicks,
    MAX(gc.timestamp)                   AS last_click_ts
  FROM google_clicks gc
  GROUP BY gc.user_id, gc.task_number
),
final_ts AS (
  SELECT
    fa.user_id,
    fa.task_number                      AS task_id,
    MAX(fa.timestamp)                   AS final_answer_ts
  FROM final_answers fa
  GROUP BY fa.user_id, fa.task_number
),
merged AS (
  SELECT
    u.id                                AS user_id,
    ca.task_id,
    COALESCE(u.llm_version, ca.personality_index_from_chat) AS personality_index,

    ca.total_dialogue_turns,
    ca.total_hallucinated_responses,
    COALESCE(clk.total_google_clicks, 0)    AS total_google_clicks,

    ca.first_chat_ts,

    GREATEST(
      ca.last_chat_ts::timestamptz,
      COALESCE(clk.last_click_ts::timestamptz, 'epoch'::timestamptz),
      COALESCE(ft.final_answer_ts::timestamptz, 'epoch'::timestamptz)
    )                                   AS last_event_ts
```

```
  FROM chat_agg ca
  JOIN users u
    ON u.id = ca.user_id
  LEFT JOIN click_agg clk
    ON clk.user_id = ca.user_id AND clk.task_id = ca.task_id
  LEFT JOIN final_ts ft
    ON ft.user_id = ca.user_id AND ft.task_id = ca.task_id
)
SELECT
  user_id,
  task_id,
  personality_index,

  total_dialogue_turns,
  EXTRACT(EPOCH FROM (last_event_ts - first_chat_ts::timestamptz))::BIGINT AS
total_time_seconds,

  total_hallucinated_responses,
  total_google_clicks,

  NULL::VARCHAR AS answer_correctness,
  NULL::INT     AS aletheia_query_verification_count
FROM merged;
```