VLE Dashboard (app.py + config.py)

This README explains how the Streamlit app (`app.py`) works, how the database connection is configured in `config.py`, and documents each data-access function.

Quick Start

- Install deps: `pip install -r visualisasi/requirements.txt`
- Run app (Windows PowerShell):

```
streamlit run visualisasi/app.py
```

Database Connection (Supabase)

- `config.py` uses a single PostgreSQL DSN string compatible with Supabase Session Pooler.
- Recommended DSN format (replace placeholders): `postgresql://<USER>:<PASSWORD>@aws-1-<region>.pooler.supabase.com:5432/<DBNAME>?sslmode=require`
- For Supabase default values:
  - User: `postgres.<PROJECT_REF>` (when using Session Pooler) or `postgres`
  - DB: `postgres`
  - Host: `aws-1-<region>.pooler.supabase.com`
  - Always include `?sslmode=require`
- Keep credentials out of code in production:
  - Streamlit Cloud: put DSN in `st.secrets["db"]["url"]`
  - Local dev: set `DATABASE_URL` env var

App Structure (app.py)

- Global filters in sidebar: Semester and Instructor. These filters affect all relevant charts.
- Pages:
  - Overview: totals (students, classes, modules, instructors), distributions by region/gender, students per module/semester/instructor, age histogram, engagement KPIs.
  - Classes: pick a presentation, see enrolled students, assessment score distribution, final score histogram, final result pie, VLE activity timeline, assessment weight donut, completion stacked bar, score vs weight scatter, and a per-student assessment status grid.
  - Students: profile card (age, gender, region, education, DOB), enrollments table, VLE activity timeline + engagement metrics, assessment score and weight bars.
  - Analytics: clicks vs final score scatter + correlation, boxplots by gender/region, enrollment vs withdrawn trend per semester, assessment weight distribution per class.
  - Instructors: instructor classes table, KPIs (class count, average score, pass-rate), comparison bar (avg score + pass rate per class).

VLE Dashboard Data Access (config.py)

This section summarizes each data function in `config.py`, the tables it reads, and the SQL JOINs used. It helps understand where fields come from and how entities relate.

Connection

- Uses `psycopg2` to connect to PostgreSQL.
- Helper `get_df(query, params)` executes SQL and returns a pandas DataFrame.

Functions and JOINs

fetch_students()

- Source: `user_account`
- Filter: `role = 'student'`
- Columns: `student_id (user_id)`, `name`, `gender`, `region`, `highest_education`, `date_of_birth`
- JOINs: None

fetch_instructors()

- Source: `user_account`
- Filter: `role = 'instructor'`
- Columns: `instructor_id (user_id)`, `name`, `department`
- JOINs: None

fetch_presentations()

- Source: `presentation` p
- JOINs:
  - `JOIN course_module m ON p.module_id = m.module_id`
  - `JOIN user_account i ON p.instructor_id = i.user_id`
- Columns: `presentation_id`, `semester`, `year`, `module_code`, `module_name`, `instructor_name`, `instructor_id`

fetch_enrollment(presentation_id)

- Source: `enrollment` e
- JOINs:
  - `JOIN user_account s ON e.student_id = s.user_id`
- Filter: `e.presentation_id = %s AND s.role = 'student'`
- Columns: `enrollment_id`, `presentation_id`, `student_id`, `name`, `studied_credits`, `final_result`

fetch_assessments(presentation_id)

- Source: `assessment` a
- Filter: `a.presentation_id = %s`
- Columns: `assessment_id`, `assessment_name`, `weight`

- JOINs: None

fetch_student_scores(enrollment_id)

- Source: `student_assessment` sa
- JOINs:
  - `JOIN assessment a ON sa.assessment_id = a.assessment_id`
- Filter: `sa.enrollment_id = %s`
- Columns: `student_assessment_id`, `assessment_id`, `assessment_name`, `score`, `weight`

fetch_vle_activity(enrollment_id)

- Source: `student_vle_activity` sva
- JOINs:
  - `JOIN vle_item v ON sva.vle_id = v.vle_id`
- Filter: `sva.enrollment_id = %s`
- Columns: `vle_id`, `vle_type`, `title`, `activity_date`, `clicks`

fetch_enrollments_all()

- Source: `enrollment` e
- Columns: `enrollment_id`, `presentation_id`, `student_id`, `final_result`, `studied_credits`
- JOINs: None

fetch_final_scores_all(presentation_id=None)

- Purpose: Weighted final score per enrollment: `SUM(score * weight) / SUM(weight)`
- Sources:
  - `enrollment` e
  - `student_assessment` sa
  - `assessment` a
- JOINs:
  - `JOIN student_assessment sa ON sa.enrollment_id = e.enrollment_id`
  - `JOIN assessment a ON a.assessment_id = sa.assessment_id`
- Optional filter: `e.presentation_id = %s`
- Columns: `enrollment_id`, `presentation_id`, `student_id`, `final_score`

fetch_final_results_distribution(presentation_id)

- Source: `enrollment`
- Filter: `presentation_id = %s`
- Aggregation: `COUNT(*) GROUP BY final_result`
- JOINs: None

fetch_total_clicks_all(presentation_id=None)

- Purpose: Total clicks per enrollment
- Sources:

    - `student_vle_activity` sva
    - `enrollment` e
- JOINs:
    - `RIGHT JOIN enrollment e ON e.enrollment_id = sva.enrollment_id`
- Optional filter: `e.presentation_id = %s`
- Aggregation: `SUM(sva.clicks)` grouped by `enrollment_id, presentation_id`
- Columns: `enrollment_id`, `presentation_id`, `total_clicks`

fetch_vle_avg_timeline_by_presentation(presentation_id)

- Purpose: Average clicks per date for a presentation
- Sources:
    - `student_vle_activity` sva
    - `enrollment` e
- JOINs:
    - `JOIN enrollment e ON e.enrollment_id = sva.enrollment_id`
- Filter: `e.presentation_id = %s`
- Aggregation: `AVG(sva.clicks)` grouped by `activity_date`
- Columns: `activity_date`, `avg_clicks`

fetch_assessment_scores_by_enrollment(enrollment_id)

- Purpose: Scores for all assessments belonging to the enrollment's presentation
- Sources:
    - `assessment` a
    - `student_assessment` sa (LEFT join)
- JOINs:
    - `LEFT JOIN student_assessment sa ON sa.assessment_id = a.assessment_id AND sa.enrollment_id = %s`
- Filter: `a.presentation_id = (SELECT presentation_id FROM enrollment WHERE enrollment_id = %s)`
- Columns: `assessment_id`, `assessment_name`, `weight`, `score`

fetch_students_by_module_counts()

- Purpose: Student count per module
- Sources:
    - `enrollment` e
    - `presentation` p
    - `course_module` m
- JOINs:
    - `JOIN presentation p ON p.presentation_id = e.presentation_id`
    - `JOIN course_module m ON m.module_id = p.module_id`
- Aggregation: `COUNT(*) GROUP BY m.module_code`
- Columns: `module_code`, `student_count`

Notes

- All functions return DataFrames.
- Filters using parameters (`%s`) are passed via `get_df(query, params=...)`.
- `RIGHT JOIN` in `fetch_total_clicks_all` ensures enrollments with zero activity are included.