Hangman Web Application - Architecture Document (Flask, C4)

This document describes a C4-based architecture for a Python Flask web application implementation of the Hangman game. It is intended for teaching, testing and small-scale deployment.

# 1 Purpose and Scope

Provide a browser-based Hangman game backed by a Flask server. The architecture supports unit testing of game logic, CI/CD with GitHub Actions, and optional simple persistence (SQLite or JSON).

# 2 High-level C4 Context

Primary external actor: Player (Web Browser). The Player interacts with the Flask web app which orchestrates game logic and optionally persists game state. CI/CD and tests run separately in GitHub Actions.

# 3 Containers (Level 2)

- Browser (UI): HTML/CSS templates rendered by Flask, optional client-side JS for UX.

- Flask Web App: Routing, request handling, templating (Jinja2), session handling.

- Game Logic package (`hangman/`): Pure Python package with `game.py`, `words.py`, `utils.py` (high-testability).

- Persistence: SQLite or JSON file for word lists and optional scores.

- CI/CD: GitHub Actions running linters, pytest and coverage.

# 4 Components (Level 3)

- `app.py` / `routes.py`: Flask app and routes (index, guess, new\_game, admin).

- `templates/`: Jinja2 templates (index.html, game.html).

- `static/`: CSS and small JS where needed.

- `hangman/game.py`: Game class with operations: start\_game, make\_guess, is\_won, is\_lost, remaining\_attempts.

- `hangman/words.py`: Word loading, sanitisation and filtering.

- `tests/`: pytest unit tests and Flask integration tests (test\_client).

# 5 Data & Control Flow

Typical guess flow:  
1. Browser POSTs guess to `/guess`.  
2. Flask route validates input and calls `Game` in `hangman/game.py`.  
3. `Game` updates in-memory state (letters guessed, attempts) and returns result.  
4. Flask renders updated template with current game state (or returns JSON).

# 6 Testing Strategy

Focus on unit tests for `hangman/` package (fast, isolated) and integration tests for Flask routes. Use pytest and pytest-cov. Enforce coverage threshold in CI. Keep tests deterministic and fast; use fixtures to create predictable Game instances.

# 7 CI/CD Integration

Use GitHub Actions to run on PRs/pushes: checkout, setup-python, install requirements, run flake8, run pytest with coverage, upload coverage results. Optionally fail on coverage < threshold.

# 8 Deployment & Operational Notes

For production-like deployment: use gunicorn behind nginx; optionally Dockerise the app. Keep secret configuration out of repo (use GitHub Secrets).

# 9 Files in this pack

- `architecture\_doc.md` (Markdown)  
- `hangman\_architecture\_diagram.png`  
- `hangman\_architecture\_diagram.puml` (PlantUML source with inline comments)  
- This Word doc.