# MLOps Final Project

The purpose of this project is to combine all the knowledge acquired in the course, and build a web app that serves a machine learning model, with all the associated CI/CD processes, versioning, and testing. Finally, the web app should be deployed to a cloud platform.

**You are to work in groups of 3**

## Project Goal

Design, build, and deploy a production-grade machine learning system that includes the full MLOps lifecycle:

- data versioning
- model training and evaluation
- model registry and promotion
- CI/CD with guard gates
- reproducible deployments

The focus is execution quality, not the business idea nor the model's accuracy. You are free to choose whichever business idea to implement.

## Technical Overview

The project is based on a web app that serves a machine learning model. The technology stack is up to you, but it's recommended to use Python for the backend that runs the ML model, and a NodeJS framework for the frontend, like ReactJS or NextJS. You can have more than one backend if you see the need to, and include a database if it is required by your app. A good online service for a DBMS is https://supabase.com/ or https://neon.com/.

## Git Branching Model (Strict Requirement)

You must use the following branching strategy:

- `feature/*` — all development happens here
- `dev` — integration branch
- `staging` — pre-production validation
- `main` — production

## CI/CD Requirements

### Required Pipelines

1. PR → dev
    - run unit tests
    - run integration tests
    - build Docker images (no push)

2. dev → staging
    - full test suite
    - deploy code to staging environment
    - deploy *candidate model* from registry (MLFlow)
3. staging → main
    - requires passing model promotion gates
    - deploy the code/models previously deployed on the staging env to production

---

## 12-Factor App

The different environments need unique environment variables, which are to be generated from github secrets in their respective workflows.

## Testing Requirements (Mandatory)

You must implement:

- 3 unit tests
- 2 integration tests
- 1 end-to-end test

Tests must be meaningful, trivial or duplicated tests will not count, and tests must run automatically in CI.

## Data Versioning (DVC)

You must:

- track raw training data with DVC
- store data remotely (S3, google drive, etc.)
- reference data versions explicitly in training runs

Every training run must be traceable to:

- a DVC data version
- a Git commit hash

## Model Versioning & Registry (MLflow + DagsHub)

You must use:

- MLflow experiments
- MLflow Model Registry

Each model version must log:

- metrics
- parameters
- data version (from DVC)

- code version (Git commit)

The registry is the single source of truth for deployments.

---

## Model Promotion Pipeline (Core Requirement)

Your project must implement a promotion pipeline (on pushing to staging).

### Required Flow

1. Train candidate model
2. Register model version in MLflow
3. Deploy automatically to staging
4. Run automated quality gates
5. If gates pass:
    - promote model to Production stage in registry

### Quality Gates (at least one required)

Examples:

- accuracy threshold
- latency limit
- schema compatibility
- smoke tests

If a gate fails:

- model stays in *Staging*
- production must not change

## Cloud Deployment

- Application must be publicly accessible
- Any platform is acceptable. Example platforms (these offer free tiers):
    - Render https://render.com
    - Railway https://railway.com
    - Scalingo https://scalingo.com/
    - Koyeb https://www.koyeb.com/
- Production must serve predictions from the Production registry stage only

---

## Deliverables

1. GitHub repository
2. Public production URL
3. README containing:
    - architecture diagram
    - CI/CD explanation

    - model promotion explanation
    - reproducibility instructions
4. Our in-class presentation