

# DOCUMENTATION TECHNIQUE

<b>I - Architecture du système</b>	<b>2</b>
<b>II - Structure des répertoires</b>	<b>2</b>
<b>III - Composants principaux</b>	<b>4</b>
<b>IV - Choix des technologies</b>	<b>4</b>
Backend	4
Frontend	4
API externes	4
<b>V - Intégration API</b>	<b>4</b>
<b>VI - Interprétations retenues du sujet</b>	<b>4</b>
Concept général	4
Rôles utilisateurs	4
Processus de clustering	5
<b>VII - Fonctionnement de l'algorithme</b>	<b>5</b>
<b>VIII - Diagramme de classe</b>	<b>5</b>

Projet par :

- Wael AKIL
- Carlos OKINDA
- Jade DELEBECQUE

## I - Architecture du système

Le système est conçu selon une architecture MVC (Modèle-Vue-Contrôleur) modulaire utilisant le framework Flask, organisée comme suit:

## II - Structure des répertoires

```
ClusteringEtudiant/
├── .env                    # Variables d'environnement (API keys, configuration)
├── .gitignore              # Fichiers ignorés par git
├── README.md               # Documentation du projet
├── requirements.txt         # Dépendances Python
├── run.py                  # Point d'entrée de l'application
├── app/                    # Package principal de l'application
│   ├── __init__.py         # Factory de l'application Flask
│   ├── config.py           # Configuration (BDD, clés API, etc.)
│   └── extensions.py       # Extensions Flask (SQLAlchemy)
│   ├── blueprints/         # Organisation modulaire des routes par rôle
│   │   ├── admin/         # Fonctionnalités administrateur
│   │   │   ├── __init__.py
│   │   │   └── routes.py  # Routes pour gérer users, etc.
│   │   ├── auth/          # Authentification
│   │   │   ├── __init__.py
│   │   │   └── routes.py  # Login, logout, etc.
│   │   ├── student/       # Fonctionnalités étudiant
│   │   │   ├── __init__.py
│   │   │   └── routes.py  # Dashboard, vote, etc.
│   │   └── teacher/       # Fonctionnalités enseignant
│   │       ├── __init__.py
│   │       └── routes.py  # Gérer élections, groupes
│   ├── dao/               # Data Access Objects - Accès à la BDD
│   │   ├── __init__.py    # Agrège tous les DAOs
│   │   ├── admin_dao.py   # Opérations BDD pour Admin
│   │   ├── election_dao.py # Opérations BDD pour Election
│   │   ├── group_dao.py   # Opérations BDD pour Group
│   │   ├── student_dao.py # Opérations BDD pour Student
│   │   ├── teacher_dao.py # Opérations BDD pour Teacher
│   │   └── vote_dao.py    # Opérations BDD pour Vote
│   └── models/            # Modèles SQLAlchemy (tables BDD)
│       ├── __init__.py    # Import des modèles
│       ├── admin.py       # Modèle Admin
│       ├── election.py     # Modèle Election + table association
│       ├── group.py       # Modèles Group et GroupMember
│       ├── student.py     # Modèle Student
│       ├── teacher.py     # Modèle Teacher
│       └── vote.py        # Modèle StudentVote
```

```
└─ services/                # Logique métier
  └─ __init__.py
  └─ admin_service.py        # Gestion des admins
  └─ clustering_service.py   # Algorithme de groupement
  └─ election_service.py     # Gestion des élections
  └─ group_service.py        # Gestion des groupes
  └─ openai_service.py       # Génération noms via OpenAI
  └─ student_service.py      # Gestion des étudiants
  └─ teacher_service.py      # Gestion des enseignants
  └─ vote_service.py         # Gestion des votes

└─ static/                  # Fichiers statiques
  └─ style.css               # Style principal de l'application

└─ templates/               # Templates HTML (Jinja2)
  └─ base.html               # Template de base (hérité)
  └─ login.html              # Page de connexion
  └─ 404.html                # Page erreur 404

  └─ admin/                  # Templates administrateur
    └─ dashboard.html
    └─ teachers_list.html
    └─ students_list.html
    └─ user_form.html

  └─ student/                # Templates étudiant
    └─ complete_profile.html
    └─ dashboard.html
    └─ group_results.html
    └─ vote_form.html

  └─ teacher/                # Templates enseignant
    └─ complete_profile.html
    └─ dashboard.html
    └─ election_form.html
    └─ manage_election.html

└─ tests/                   # Tests unitaires
  └─ __init__.py
  └─ test_clustering.py      # Tests de l'algorithme
  └─ test_models.py          # Tests des modèles
  └─ test_routes.py          # Tests des routes HTTP
  └─ test_services.py        # Tests des services
```

### III - Composants principaux

**Factory de l'application** : Le fichier `__init__.py` contient la fonction `create_app()` qui initialise et configure l'application Flask.

**Blueprints** : L'application est divisée en modules (blueprints) par rôle utilisateur pour isoler les fonctionnalités.

**Models** : Définition des entités de base de données avec SQLAlchemy.

**Services** : Encapsulation de la logique métier pour chaque entité.

**DAO** : Couche d'abstraction pour l'accès aux données.

### IV - Choix des technologies

#### Backend

- Python 3.8+: Langage de programmation principal
- Flask: Framework web léger et flexible
- SQLAlchemy: ORM (Object-Relational Mapping) pour l'abstraction de la base de données
- Blueprint: Pour la modularité de l'application
- SQLite: Base de données par défaut (facilement remplaçable)

#### Frontend

- HTML/CSS: Interface utilisateur simple et responsive
- Jinja2: Moteur de templates intégré à Flask

#### API externes

- OpenAI API: Génération automatique de noms de groupes basés sur les initiales des membres

### V - Intégration API

Uses OpenAI API for generating group names :

- **Function** : `generate_group_name_from_initials(first_names)`
- Requires `OPENAI_API_KEY` environment variable
- Uses OpenAI gpt-4o-mini model for creative naming

### VI - Interprétations retenues du sujet

#### Concept général

L'application permet de regrouper des étudiants selon leurs préférences mutuelles à travers un système de vote et un algorithme de clustering.

#### Rôles utilisateurs

- **Administrateur**: Gestion des utilisateurs (création, suppression des enseignants et étudiants, attribution de rôle)
- **Enseignant**: Création et gestion des élections, paramétrage des élections, mise en oeuvre de la création de groupe, visualisation des résultats

- **Étudiant:** Participation aux élections par vote, visualisation des groupes formés

### Processus de clustering

L'enseignant crée une élection avec des paramètres (taille des groupes, les étudiants concernés)

Les étudiants votent en classant leurs préférences avec un système de 100 points à répartir.

Une fois l'élection terminée, l'algorithme forme des groupes optimaux

Les noms des groupes sont générés automatiquement via l'API OpenAI

Les résultats sont accessibles aux enseignants et étudiants

## VII - Fonctionnement de l'algorithme

L'algorithme de clustering est au cœur du système et fonctionne selon ces principes:

**Collecte des votes :** Les votes des étudiants sont recueillis sous forme de classements

**Matrice de préférences :** Construction d'une matrice représentant les affinités entre étudiants

Application de l'algorithme de clustering:

- Utilisation d'une approche basée sur les graphes pour regrouper les étudiants
- Prise en compte des contraintes de taille de groupe définies par l'enseignant

Optimisation pour maximiser les affinités intra-groupe

Formation des groupes: Assignment finale des étudiants aux groupes

Génération des noms: Utilisation de l'API OpenAI pour générer des noms créatifs basés sur les initiales des membres

## VIII - Diagramme de classe

