Technical Architecture - “Clustering de Camarades”

Overview

We were inspired by the hexagonal model for the construction of our architecture.

Components

Core (Center)

- **auth_service**: Authentication and role management
- **storage_service**: Data persistence orchestration
- **algo_service**: Clustering functions
- **main**: Entry point and coordination
- **domain** (Class) :
    - group
    - owner
    - student
    - teacher
    - user
    - vote

External packages

- **auth**: Password generation, keys, hashing
- **ihm**: User interface and interactions
- **storage**: JSON storage (votes + users)
- **algo**: Group formation based on preferences

Communication

- **ihm** → Core services (auth, storage, algo)
- **auth** → auth_service only
- **storage** → storage_service only
- **algo** → Multiple Core services

Benefits

- **Decoupling**: One-way dependencies toward Core
- **Maintainability**: Clear separation of concerns
- **Testability**: Independent components

Technological Choices

Python Choice

Project advantages:

- **Simplicity and fast development**: Clear syntax allowing focus on business logic rather than language complexity
- **Rich ecosystem**: Many libraries for algorithms (NumPy, SciPy) facilitating clustering implementation
- **Easy maintenance**: Readable and maintainable code, essential for academic project with documentation
- **Rapid prototyping**: Ideal for the 2-day exam constraint
- **Versatility**: Easily handles UI, algorithm and data persistence in the same environment

Local Storage Choice (JSON)

Context advantages:

- **Simple implementation**: No complex database configuration needed
- **Autonomy**: Functional application without external dependencies
- **Easy debugging**: Data directly readable and modifiable
- **Portability**: Application easily transferable between machines
- **Scope adaptation**: Sufficient for academic prototype with limited number of users
- **Time constraints respect**: Minimal installation and configuration

Academic Project Relevance

These choices are particularly suitable because they allow to:

- Deliver a functional application within given deadlines
- Focus efforts on clustering algorithm
- Facilitate testing
- Ensure simple maintenance

Project Interpretations

Data Storage

Choice: Local database using JSON format

Justification:

- **Usage context**: Solution suitable for a teacher working locally to create groups
- **Security by design**: Local storage eliminates transmission and data exposure risks
- **Operational simplicity**: No server infrastructure required, immediate deployment

Authentication System

Implementation: Automatically generated key system

How it works:

- Generation of authentication keys similar to GitHub tokens
- Security through cryptographic keys and hashed passwords

Role Architecture

Implemented hierarchy:

Owner

- Creates Teacher and Student accounts
- Global administration rights

Teacher

- Creates Student accounts
- Manages groups

Student

- Enters preferences through voting interface
- Views group formation results

Password Management

Process implemented:

1. **Initial generation**: Password automatically generated when creating Student and Teacher accounts
2. **First login**: Invitation to customize the generated password, Implementation of password change on first login (depending on development availability)

User Interface

Design:

- **Simplicity**: Clear and clean interface with integrated login system
- **Voting system**: Dropdown menu allowing students to manage their preferences
- **Navigation**: Interface adapted to user roles

Clustering Algorithm

Selected approach: Greedy algorithm

Balancing strategy:

- Formation of groups of size n and n-1 to maximize balance
- Avoid creating groups of size 1 (student isolation)
- Optimize affinities while maintaining homogeneous sizes

Balancing example: 25 students, desired groups of 4:

- Without balancing: [4,4,4,4,4,4,1] → 1 isolated student
- With balancing: [4,4,4,4,3,3,3] → homogeneous groups

Clustering Algorithm

Overview

The clustering algorithm uses a **greedy approach** to form balanced groups based on user preferences. Users rate each other with numeric scores and the algorithm maximizes mutual satisfaction within groups.

Algorithmic Simplicity and Reasonable Performance :

The greedy approach offers **reduced complexity** compared to exhaustive methods or certain optimization algorithms.

It avoids costly iterations typical of algorithms like **k-means** or **genetic algorithms**, while still producing **fast and acceptable results** for academic or operational use.

Algorithm Flow

1. Data Preparation

- **Input**: Each user affect weight for other users
- **Process**: Build an affinity matrix where `matrix[i][j] = score` given by user i to user j
- **Output**: Square matrix with all preference relationships

2. Group Size Calculation

- **Goal**: Distribute users into balanced groups avoiding single-person groups
- **Logic**: Calculate optimal group sizes that sum to total users
- **Example**: 10 users with max size 4 → groups of [4, 3, 3]

3. Group Formation (Greedy Strategy)

For each group to create:

Step 1: Select Group Leader

- Calculate each ungrouped user's total connection score with all other ungrouped users
- Choose the user with highest total score as group leader

Step 2: Select Partners

- For the leader, calculate mutual affinity scores with all remaining ungrouped users
- Select top `(group_size - 1)` users with highest mutual scores
- Form the group: [leader + selected partners]

Step 3: Update

- Remove all grouped users from ungrouped set
- Repeat until all users are grouped

4. Score Calculation

Group Score: Sum of all mutual preferences within the group

- For each pair of users in the group: add `score(user1→user2) + score(user2→user1)`
- Avoid counting same pair twice

Total Score: Sum of all individual group scores

Key Features

- **Balanced Groups**: No single-person groups, sizes differ by maximum 1
- **Mutual Preferences**: Considers both directions of user preferences
- **Greedy Optimization**: Prioritizes users with strongest overall connections
- **Scalable**: Works with any number of users and group sizes

Installation Instructions

Required Modules

Install the following Python modules:

- `numpy` - For numerical operations and matrix calculations
- `tkinter` - For graphical user interface
- `unittest` - For testing functionality (built-in)
- `datetime` - For date and time handling (built-in)
- `os` - For operating system interactions (built-in)
- `json` - For JSON data handling (built-in)

Installation Commands

Install numpy (all operating systems)

```
pip install numpy
```

Install tkinter (operating system specific)

You need to install a Python version that includes tkinter support or install tkinter separately for your system.

Running the Application

From the project root directory, use one of the following commands:

```
python src/main.py
```

or

```
python -m src.main
```

Project Structure

Make sure you are in the root directory of the project before running the application commands.