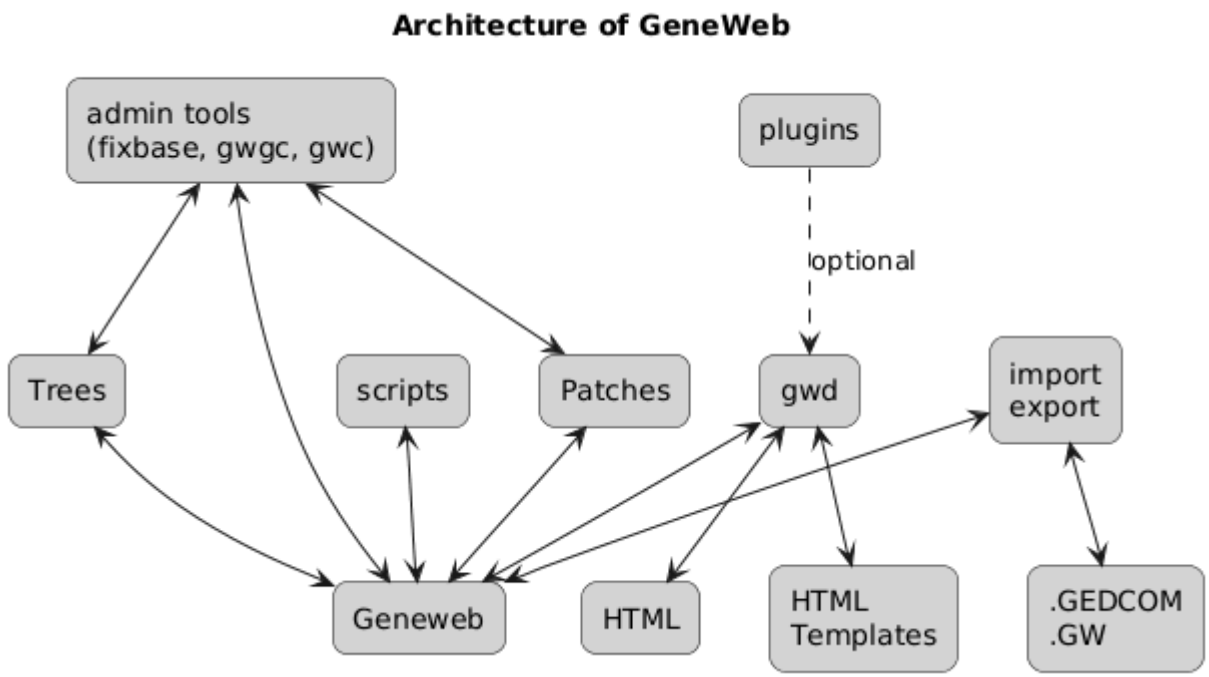

Coding Style – PEP 8

Our project follows the PEP 8 coding style guidelines, the official standard for writing clean and consistent Python code. PEP 8 promotes uniformity across the codebase, helping developers easily read, understand, and maintain each other's work. Since code is read far more often than it is written, having a common structure and format improves collaboration, reduces confusion, and ensures long-term maintainability.

We chose PEP 8 because it enforces clarity and readability through clear rules on indentation, spacing, line length, and naming conventions. Consistent style makes the codebase predictable, which simplifies debugging and speeds up onboarding for new contributors. This also improves version control readability, as diffs focus on logical changes rather than formatting differences.

To maintain compliance with PEP 8, we use automated tools such as flake8, black, and isort. These tools detect style violations, automatically reformat code, and organize imports consistently. They are integrated into our development workflow through pre-commit hooks and continuous integration (CI) checks, ensuring that every commit and pull request adheres to the same standards.

Regarding file and naming conventions, PEP 8 recommends using lowercase letters with underscores for module and package names (e.g., data_loader.py, api_client.py). Class names follow the CapWords convention (e.g., UserProfile), while functions, variables, and method names use snake_case (e.g., get_user_data, process_file). This consistent naming strategy improves readability and makes it immediately clear what type of element each name represents.

Overall, using PEP 8 helps us maintain a clean, professional, and scalable Python codebase. It ensures that every contributor, regardless of experience or background, can write code that integrates seamlessly with the rest of the project.

Justification of Technologies Used

Facilities and Infrastructure

GitHub Repository and GitHub Actions (CI/CD)
The project is hosted on GitHub to provide a centralized, version-controlled environment that facilitates collaboration and transparency among developers. GitHub Actions is used to automate the Continuous Integration and Continuous Deployment (CI/CD) pipeline. This ensures that every code change is automatically built, tested, and validated before being merged into the main branch. By automating testing and deployment, we reduce manual errors, improve release reliability, and maintain a consistent development workflow.

Developer Machines for Local Testing
Each developer has a local environment configured to mirror the Docker-based testing environment. This allows developers to quickly run and debug tests before pushing code changes to the shared repository. Local testing reduces feedback loops and ensures that code meets baseline quality standards before CI/CD validation.

Tools

Flask (Front-End Web Framework)
Flask est utilisé principalement pour la partie front-end de l'application. Grâce à sa simplicité et à sa flexibilité, il permet de gérer facilement le rendu des pages web, la configuration des routes côté client et l'intégration de templates HTML, CSS et JavaScript. Son architecture légère facilite le développement d'interfaces web dynamiques tout en gardant un contrôle précis sur la structure du projet. Flask offre également une bonne compatibilité avec des outils comme Jinja2 pour le templating et permet un prototypage rapide de l'interface utilisateur.

FastAPI (Backend Web Framework)

FastAPI est utilisé pour le développement du backend de l'application, notamment pour la création et l'exposition des API REST. Conçu pour la performance et la rapidité, FastAPI s'appuie sur Python type hints et le modèle asynchrone pour offrir une exécution efficace et une validation automatique des données. Il facilite la documentation automatique des endpoints grâce à OpenAPI et Swagger UI, tout en permettant une intégration fluide avec les bases de données et les systèmes d'authentification. Cette approche assure une communication robuste entre le front-end et le backend, ainsi qu'une maintenance simplifiée du code serveur.

pytest (Unit and Integration Testing)

We use pytest as the main testing framework due to its simplicity, flexibility, and extensive plugin ecosystem. It supports both unit and integration testing, offering readable syntax and detailed output that simplifies debugging. Pytest integrates seamlessly with CI pipelines and coverage tools, ensuring that test execution and reporting are efficient and automated.

Selenium (UI and Accessibility Testing)

Selenium is employed for testing the application's user interface and verifying accessibility compliance. It allows automated interaction with web elements, ensuring that user workflows function as expected across browsers. By incorporating Selenium tests into our CI pipeline, we continuously validate front-end behavior and user experience with every code update.

SonarQube (Static Code Analysis and Security Checks)

SonarQube is integrated to perform static code analysis, detect code smells, and identify potential security vulnerabilities. It enforces coding standards, improves maintainability, and helps the team proactively address issues before they reach production. Regular SonarQube scans contribute to cleaner, safer, and more robust code.

Allure / GitHub Actions Reports (Test Reporting)

For transparent and accessible test reporting, we use Allure or GitHub's built-in test report summaries. These tools generate visual, detailed reports that document test execution results, failure trends, and coverage metrics. This allows both developers and project managers to track quality evolution over time and identify problem areas quickly.

Code Organisation

The GeneWeb codebase is organized in a modular and layered way, ensuring a clear separation between the core logic, supporting libraries, and additional tools. At the heart of the repository lies the core engine, located in the lib directory. This is where most of the application's logic is implemented, including the fundamental components for data processing, genealogical computations, and communication between different modules. The code in this section is primarily written in OCaml, which reflects GeneWeb's focus on performance, type safety, and functional programming principles.

Around the core library, several supporting directories provide complementary functionality. The bin directory contains the compiled executables and scripts used to launch the main server processes and command-line utilities. The rpc directory defines the modules responsible for remote procedure calls and interaction between the web interface and the backend logic. This structure helps maintain a clean boundary between computational logic and user interaction layers, which is essential for stability and scalability.

The repository also includes a plugins directory that contains optional extensions and add-ons. This modular approach allows developers to add or modify functionalities without altering the core engine. Each plugin can define its own logic, dependencies, and configuration, promoting flexibility and long-term maintainability. Testing code is centralized in the test directory, which includes unit, integration, and regression tests to ensure that each module behaves as expected. This systematic organization of test resources simplifies automation and continuous integration processes.

In addition, the repository integrates infrastructure-related code, such as the docker directory used for environment configuration and deployment, and the benchmark directory, which contains performance testing and profiling tools. These components are kept separate from the main logic to avoid coupling deployment and runtime environments with the core software.

Overall, the GeneWeb repository follows a clean, modular architecture where each part of the system—core logic, extensions, infrastructure, and testing—is isolated yet integrated through well-defined interfaces. This structure enhances maintainability, makes navigation intuitive for developers, and facilitates collaborative work across different areas of the project.