Individual – Programmer-guide SmartDataProject

Esteban BARRACHO

15 juillet 2025

Table des matières

1 Preamble

This document provides a comprehensive guide to the Smart Data Project, a secure and modular platform developed as part of an individual engineering initiative. Designed for professional environments, the system streamlines project coordination, time tracking, and administrative workflows. The solution includes features such as intelligent Excel data adaptation using OpenRouter AI, real-time Outlook-style planning, and advanced task encoding. This report outlines the functionalities, architecture, and user interfaces of the platform, developed independently and delivered to S2 Engineering.

The Smart Data Project delivers a modern and intelligent platform designed to optimize operational efficiency through structured task management, time tracking, and real-time collaboration. Built with a modular architecture, it integrates advanced features such as AI-driven Excel adaptation, collaborative agenda planning, and automated alerts for delayed or incomplete entries. Its seamless user experience accommodates both desktop and mobile use, empowering project stakeholders at all levels—from field technicians to financial administrators.

This documentation not only serves as a user manual but also outlines the design rationale, technical foundations, and innovations underlying the software. Particular attention is paid to the system's extensibility, including its integration with Microsoft Outlook, dynamic foreign key handling, and fuzzy search mechanisms using Levenshtein distance.

While the project was commissioned by S2 Engineering, it was developed as an independent engineering initiative with a strong emphasis on adaptability, maintainability, and user empowerment. This guide aims to support both end users and future developers in deploying, understanding, and extending the Smart Data Project within professional or educational contexts.

2 The chapters

2.1 Where would I start if I had to continue developing this software ?

2.1.1 Where are the sources ?

The sources are listed in the **Method** folder, which contains three subfolders : *Analysis*, *Design* and *Implementation*.link to folder.

Analysis This section contains the various meetings and the three cheat sheets that were used to bring the project to a successful conclusion.

Design This section contains the prototype drawings and sketches that we produced ourselves.

Implementation This last section deals with tests carried out on APIs and encryption functionality, as well as AI testing in Python.

2.1.2 Where's the documentation ?

User and Progammer documentation

2.1.3 Who were the developers ?

— Esteban BARRACHO

2.1.4 Where are the specifications ?

specifications

2.1.5 Who are the sponsors ?

The client for the Smart Data Project is Vincent Pirnay. He communicated his needs clearly from the outset and remained actively involved in the definition and evolution of the platform. I contributed my own technical proposals, which were iteratively validated through an Agile approach to ensure alignment with his expectations at every stage. Whenever necessary, he provided constructive feedback that was promptly integrated into the development process, contributing to a collaborative and goal-oriented workflow.

2.1.6 What was the development environment like ?

Languages, editors, testing, continuous integration, etc. The development of the Smart Data Project was carried out in a structured and modern programming environment, ensuring both code quality and team coordination throughout the project lifecycle.

- **Programming language :** Python, chosen for its expressiveness and ecosystem adapted to web APIs and data processing.
- **Development environment :** PyCharm, an IDE that offers powerful debugging tools and seamless integration with version control systems.
- **Testing :** Local tests and runtime assertions were used to validate code logic and ensure application stability. Most components were manually validated through an interactive interface.
- **Version control :** Git was used for source code tracking, branch management and collaborative work through a GitHub private repository.
- **Deployment :** Docker and Docker Compose were used to containerize the application and its MySQL database, ensuring environment consistency across platforms.

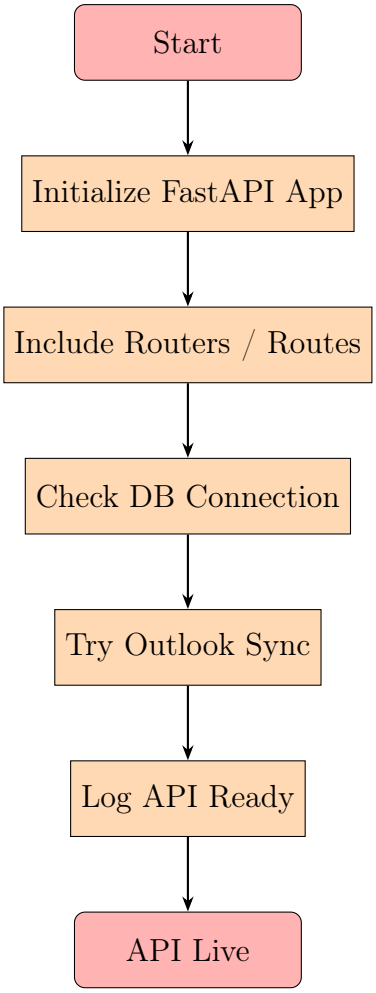
2.1.7 How to compile, how to deploy, how to test, etc.

The Smart Data Project is a modern web application developed with FastAPI and designed to be easily deployed via Docker. It does not require traditional compilation.

- **To run the project locally :**
  1. Ensure that `Docker` and `Docker Compose` are installed on your system.
  2. Open a terminal in the project root directory (containing the `docker-compose.yml` file).
  3. Execute the following command :
     `docker-compose up -build`
- **To access the application :**
  Once launched, the API will be available at `http://localhost:8000`, and the database at port `3306` (MySQL).
- **To run and test specific parts :**
  - Open the `main.py` file in the `code` folder to browse the entry point of the API.
  - Use the Swagger interface at `/docs` to interactively test all endpoints.
  - Authentication is required to access most endpoints (a login form is available at the root path `/`).
- **To reset the database :**
  Execute `docker-compose down -v` followed by `docker-compose up -build` to rebuild the containers and reload the schema from `init.sql`.

2.1.8 The "keys" to architecture !

Flowchart of `main.py` for the Smart Data Project



Global architecture overview

- The `main.py` script acts as the central entry point and orchestrator of the application.
- Route files such as `admin.py`, `dashboard.py`, `planification.py` and others are registered modularly into the FastAPI instance.
- The application is launched with Uvicorn as ASGI server through Docker, with hot reload enabled during development.
- The initialization also attempts to connect to MySQL and triggers a synchronization call to Outlook (if credentials are configured).

2.2 Point of view

2.2.1 Top-down

Responding to specific needs

The tree structure of modules and packages, with a motivation for each one The Smart Data Project adopts a clean and modular architecture to ensure clear separation of concerns, reusability of code, and maintainability over time :

- **Module *main.py*** : Entry point of the FastAPI application, it initializes the app, sets up middleware, and includes the route modules.
- **Package *routes/*** : Contains route modules grouped by functionality such as `admin.py`, `planificatio` `dashboard.py`, etc., improving modularity and clarity.
- **Module *models.py*** : Defines ORM classes that map directly to the database tables using SQLAlchemy.
- **Module *schemas.py*** : Contains Pydantic models used for validation and data serialization/-deserialization in API requests and responses.
- **Module *database.py*** : Manages the SQLAlchemy session and connection configuration.
- **Module *openrouter_ adapter.py*** : Handles intelligent Excel import with automatic structure suggestion using OpenRouter and the Mistral model.
- **Module *outlook_ sync.py*** : Manages Outlook calendar synchronization, OAuth2 login, and event retrieval.

The main components of the architecture The project is architectured around core components that allow for easy data interaction, real-time planning, and administrative monitoring :

- **FastAPI server** : Manages asynchronous API routes, user sessions, file uploads, calendar syncing, and dynamic frontend rendering.
- **Jinja2 Templates** : Used to render HTML pages with dynamic content including agenda, dashboards, planning views, etc.
- **MySQL Database** : Stores all persistent entities, including users, tasks, projects, invoices, planning, and prestations.
- **Docker + Uvicorn** : Provides an isolated and scalable runtime for deployment and testing with hot-reload support.

Documentation on the application's run-time behaviour Several runtime behaviours are handled for robustness, performance, and user feedback :

- **Error handling** : Centralized exception handlers catch validation errors, authentication failures, or database constraints, and render user-friendly error templates.
- **Execution logs** : Key lifecycle events such as database startup, Excel adaptation, and Outlook integration are logged to the console.
- **Performance and optimisation** : The app defers heavy Excel parsing and Outlook synchronization using asynchronous FastAPI endpoints and retry loops for slow-starting services (e.g., MySQL via Docker).

2.3 Comment on the "motivation" behind my choices !

2.3.1 Why things happen

The design and implementation decisions in the Smart Data Project were guided by two main pillars : the functional requirements of the client and an emphasis on user experience, maintainability, and smart automation.

From the outset, the project aimed to deliver a platform capable of managing project data, planning, and task tracking while integrating intelligent import mechanisms (e.g., Excel file adaptation). The client emphasized the need for a user-friendly interface with efficient encoding, agenda views, and automatic alerts.

Key motivations include :

— **Use of FastAPI** : Chosen for its modern, high-performance asynchronous capabilities and ease of integration with Pydantic and SQLAlchemy.
— **Excel import with AI suggestion (OpenRouter)** : Proposed as an enhancement to simplify database population and ensure structural conformity of uploaded data, inspired by my own interest in reducing repetitive manual work.
— **Modular architecture** : Suggested to facilitate future maintainability and allow independent improvements on different components (e.g., tasks, planning, finance).
— **Dynamic dropdowns and ID generation** : Implemented to improve relational consistency and speed up data entry without requiring deep knowledge of database schema.
— **Mobile-compatible interfaces & dark theme** : Inspired by real-world usage scenarios where collaborators work from the field or from their phone. UI/UX clarity was a personal priority.

Additional features like delayed alerting for prestation encoding or Outlook calendar sync arose from proactive suggestions and discussions with the client, seeking to deliver a robust, real-time planning and coordination environment.

2.4 Software libraries or components

2.4.1 Your users are developers

Developers interacting with the Smart Data Project can leverage a variety of powerful and modular libraries chosen for their maturity, community support, and integration capabilities.

— **FastAPI** : Used for building asynchronous and type-safe REST APIs with automatic documentation (OpenAPI), improving developer productivity and API usability.
— **SQLAlchemy** : Enables ORM-based interactions with the MySQL database, simplifying schema definition, data queries, and relationships between models.
— **Pandas** : Used extensively for processing Excel files during intelligent import operations and for manipulating tabular data.
— **OpenRouter API** : Provides access to free and intelligent models (e.g., Mistral) for structure validation and correction suggestions during file imports.
— **Jinja2** : Template engine integrated with FastAPI to render HTML interfaces with dynamic content (e.g., dashboard, agenda, task management).
— **PyMySQL** : Low-level MySQL client used for SQLAlchemy engine connectivity.
— **Uvicorn** : ASGI server that runs the FastAPI application in production and development environments.
— **Matplotlib / Graphviz** (optionally) : Considered for visualisation or export modules involving Gantt views or dependency graphs.

This modular stack ensures the software remains accessible, extensible, and compatible with modern DevOps workflows, including Docker-based deployment and CI/CD integrations.

2.4.2 Examples of code :

Increasing complexity To support developers wishing to explore or extend the functionalities of the project, this section presents code snippets of increasing complexity that demonstrate key components of the Smart Data Project.

Basic example : Calling OpenRouter AI for a simple text-based task.

```
import requests

OPENROUTER_API_KEY = "sk-or-v1-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

headers = {
    "Authorization": f"Bearer {OPENROUTER_API_KEY}",
    "HTTP-Referer": "http://localhost",
    "Content-Type": "application/json"
}

payload = {
    "model": "mistralai/mistral-small-3.2-24b-instruct:free",
    "messages": [
        {"role": "user", "content": "List all capitals of EU countries in JSON format."}
    ]
}

response = requests.post("https://openrouter.ai/api/v1/chat/completions", headers=headers, json=payload)

if response.status_code == 200:
    print(response.json()["choices"][0]["message"]["content"])
else:
    print("Error:", response.status_code)
```

Intermediate example : Reordering columns and aligning types to SQL definitions.

```
def reorder_columns(df, table):
    inspector = inspect(engine)
    sql_cols = [col["name"] for col in inspector.get_columns(table)]
    ordered_cols = [col for col in sql_cols if col in df.columns]
    return df[ordered_cols]
```

Advanced example : Full adaptation of an Excel file to SQL schema with IA suggestion.

```
def adapt_excel(file_path, table):
    df = pd.read_excel(file_path)
    df.columns = df.columns.str.strip()
    suggestion = suggest_structure_from_ia(df, table)
    df = reorder_columns(df, table)
    df = fix_types(df, table)
    return df.fillna("").to_dict(orient="records"), suggestion
```

2.5 Des API

APIs play a central role in orchestrating the communication between the client interface and the backend logic of the Smart Data Project. They provide structured endpoints through which users and developers can interact with the system to query, update, and visualize data, or to trigger intelligent adaptation mechanisms such as Excel import with AI validation.

2.5.1 Prediction API

The prediction API exposes a route allowing clients to submit structured data and obtain predictive results. This is particularly useful in contexts where linear regression models (e.g., LARS) are deployed to forecast numerical outcomes based on various features. The interface is JSON-based and optimized for integration with third-party systems or frontend dashboards.

2.5.2 Visualisation API

This endpoint enables real-time access to rendered graphical results (e.g., decision trees, project timelines, or task statistics). By abstracting the rendering logic through an API, developers can dynamically fetch visual assets and integrate them seamlessly into any user interface (mobile, desktop, or web).

2.5.3 Excel Adaptation API

One of the most distinctive APIs of the platform enables users to upload Excel files that are automatically validated, cleaned, and structured to fit the SQL schema. Powered by OpenRouter AI, this endpoint not only performs format and type conversions but also returns suggestions on potential column mismatches, missing fields, or schema inconsistencies—enhancing robustness and user autonomy.

2.5.4 Authentication and User Session API

Authentication is managed through session cookies to ensure secure identification of users across the platform. The API validates login credentials and establishes a session, enabling subsequent secure access to protected resources (e.g., agenda, administration panel, or personal prestations).

2.5.5 Administrative API

Administrators can interact with backend tables via dedicated endpoints (CRUD operations) exposed under secured routes. These APIs serve for managing invoices, clients, tasks, or project allocations, streamlining backend workflows while ensuring traceability through SQLAlchemy transaction control.

2.6 Embedded code

2.6.1 Deployment therefore deserves special attention

The deployment of the Smart Data Project requires careful attention due to its architectural composition and interaction with external services such as SQL databases and OpenRouter AI. Although developed with simplicity and modularity in mind, several critical points must be respected to ensure stable deployment :

- **Installation of dependencies** : The project relies on a set of Python libraries, including Flask, SQLAlchemy, pandas, and requests. These must be installed using a package manager such as `pip`, using the provided `requirements.txt` file.
- **Server configuration** : The FastAPI server must be launched within a properly set up environment. The use of environment variables (.env) is required to connect to the SQL backend, configure ports, and access secure API keys.
- **Containerization (optional)** : For robust deployment across environments, Docker support is provided. A `Dockerfile` and `docker-compose.yml` define the full application stack (backend + database).
- **Log management** : To monitor operations (especially Excel imports and AI suggestions), logging is enabled at runtime. The standard Python `logging` module may be extended using external services such as Loggly or Grafana.
- **Testing and validation** : Unit testing of routes and Excel import mechanisms is encouraged before deployment. A test suite can be expanded with `pytest` or custom validation scripts to verify data flow and structure integrity.

2.6.2 Hardware Matters More...

Although the application runs on standard server environments, the performance of the Smart Data Project is directly influenced by the underlying hardware. Below are some hardware-related deployment recommendations :

- **Server capacity** : Ensure that the host server (virtual or dedicated) offers sufficient CPU cores to handle API calls, database transactions, and asynchronous tasks such as Excel parsing or AI suggestions in parallel.
- **Memory and storage** : A minimum of 4 GB RAM is recommended. SSD storage is strongly advised for faster database operations and Excel file access.
- **AI Model usage** : Since OpenRouter is accessed via API, no local GPU is required. However, latency and reliability depend on the network quality and OpenRouter's availability.
- **Backup and fault tolerance** : For production deployments, data persistence via MySQL volumes should be accompanied by regular snapshots. Docker volumes are mapped for this purpose. Consider a failover mechanism for the database if uptime is critical.

2.7 Limitations and Future Work

2.7.1 Current technical limitations

Although the Smart Data Project offers a robust base for data management, planning, and integration with external tools, certain limitations remain in its current version :

— **Dependency on external APIs** : The use of OpenRouter for AI-powered suggestions introduces a dependency on a third-party service. In case of latency or service interruption, the import feature may become temporarily unavailable.
— **Lack of automated test coverage** : While the architecture supports testing, the project currently lacks a formal test suite with unit, integration, and regression tests.
— **Single-user session model** : The authentication system uses session cookies without token refresh or OAuth expiration checks, which may limit long-term scalability or security in multi-user environments.
— **Data validation at import** : Although AI suggestions highlight structural issues in Excel files, no automatic correction or rollback mechanism is in place if the import fails or corrupts rows.
— **Mobile responsiveness** : Some advanced features like planning views and manual time encoding require refinements for optimal use on smartphones and tablets.

2.7.2 Potential improvements

Based on user feedback and architectural analysis, several enhancements can be considered to evolve the project :

— **Introduction of a test suite** : Add a testing framework using `pytest` with mock databases to secure future code updates and refactoring.
— **Progressive enhancement of AI features** : Enable intelligent mapping of mismatched Excel columns, automatic type inference correction, and column renaming suggestions via IA.
— **Real-time collaboration features** : Integrate WebSocket or long-polling to enable live project editing or planning updates by multiple users concurrently.
— **Admin analytics dashboard** : Develop a dashboard summarizing task progression, invoice status, and user activity using charting libraries (e.g., Plotly, Recharts).
— **Advanced notification system** : Replace current static alerts with an asynchronous notification center (email or in-app) to remind users of overdue entries or approvals.
— **Progressive web app (PWA)** : Make the platform installable on mobile devices with offline support and caching of agenda data.
— **Role-based access control** : Refine permission levels with finer granularity for managers, financial officers, and external stakeholders.

3 Glossary

OpenRouter A universal API gateway allowing access to various large language models (LLMs) including Mistral and GPT-based models, used here to analyze Excel structures and suggest data corrections.

SQLAlchemy A Python SQL toolkit and Object Relational Mapper (ORM) used to interact with the database in a Pythonic way.

FastAPI A modern web framework for building APIs with Python, used for handling routes and backend logic in the Smart Data Project.

Session In web development, a session tracks the interaction between the user and the application during their visit. The session ID is stored in a cookie for authentication.

Docker A tool designed to create, deploy, and run applications in containers, ensuring that the software behaves the same regardless of the environment.

CRUD Create, Read, Update, Delete : Basic operations for managing data in a database.

ORM Object-Relational Mapping : A programming technique for converting data between incompatible systems using object-oriented programming.

RedirectResponse A response type in FastAPI that instructs the browser to redirect the user to a different route.

Levenshtein Distance A string metric for measuring the difference between two sequences, used here to support approximate search functionality across database tables.

Excel Import A feature allowing users to upload Excel files and populate the database, guided by AI suggestions and structure adaptation mechanisms.

Agenda The planning interface of the Smart Data Project, where events and task encodings can be visualized similarly to Outlook.

Prestation A work entry indicating time worked on a project or task, manually or automatically logged by users.

Tâche A task within the system, possibly linked to projects, planification, or prestations, which can be tracked and updated.

DeepSeek A large language model platform previously used in this project and replaced by OpenRouter for Excel adaptation and structure suggestions.

PWA Progressive Web Application : A type of application built using web technologies that behaves like a native app on mobile devices.

4 Bibliographie

4.1 Further reading

Références

[1] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. Third edition, McGraw-Hill, 2003.
[2] Robert C. Martin. *Clean Code : A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.
[3] Brandon Rhodes and Dan Beyer. *The Hitchhiker's Guide to Python : Best Practices for Development*. O'Reilly Media, 2020.
[4] Miguel Grinberg. *Flask Web Development : Developing Web Applications with Python*. Second edition, O'Reilly Media, 2022.

4.2 Structures to compensate for pre-requisites

Références

[1] David Beazley and Brian K. Jones. *Python Cookbook*. Third edition, O'Reilly Media, 2013.
[2] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.
[3] Tim Clark. *SQL for Data Scientists : A Beginner's Guide for Building Datasets for Analysis*. Wiley, 2020.
[4] Ashish Maurya. *FastAPI : The complete guide — from beginner to advanced*. Packt Publishing, 2022.

5 Index

6 Annexes

6.1 Annexe A : Future Plans

One of the most strategic enhancements planned for future releases of the Smart Data Project platform involves deeper integration with external productivity tools—particularly Microsoft Outlook via the Azure API.

The current system already enables users to connect their Outlook calendar and synchronize events to the internal agenda. Future iterations will go further by allowing the system to automatically analyze the contents of incoming emails (via Microsoft Graph API) using integrated AI (OpenRouter and Mistral). The goal is to detect task-related messages—such as client requests, deadlines, or deliverables—and automatically convert them into structured tasks within the platform's database.

This would involve :

— Parsing and preprocessing email bodies using Azure-secured access tokens.
— Detecting keywords, dates, and context using a custom AI prompt on OpenRouter.
— Matching these to existing projects, or suggesting new entries if needed.
— Offering a confirmation step before inserting the task into the internal system.

This smart feature will not only improve planning reactivity and reduce manual entry, but also provide seamless integration between professional communication tools and operational data workflows.

© 2025 Esteban BARRACHO.
This document is distributed under the terms of the Creative Commons CC BY-ND (Attribution-No Derivative Works) licence.

This work is licensed under a Creative Commons "Attribution-NoDerivatives 4.0 International" license.