WayFinder

Parcoursup Analytics Platform

Development Report

Introduction to Python

GROUP 2

Bryan Boislève
Mizaan-Abbas Katchera
Nawfel Bouazza

Master Data Science & Business Analytics

ESSEC Business School / CentraleSupélec

CentraleSupélec

December 2025

Contents

1 Introduction

1.1 Context and Motivation

Every year, more than 900,000 French students face one of the most consequential decisions of their academic lives: choosing where to pursue higher education. Through Parcoursup, the national admission platform, they must navigate a labyrinth of over 20,000 training programs spanning universities, preparatory classes, engineering schools, and specialized institutions. The sheer volume of options, combined with opaque admission criteria and scattered statistical data, leaves many students feeling overwhelmed and uncertain about their choices.

WayFinder emerged from our desire to address this very real problem. We designed it as a proof-of-concept demonstrating how public data could be transformed into a genuinely useful tool for students. Rather than building yet another academic exercise, we wanted to create something that could make a tangible difference in how students approach their Parcoursup decisions. The platform aggregates and analyzes publicly available admission statistics, turning raw numbers into personalized insights that help students understand not just where they might be admitted, but which programs truly align with their profile and aspirations.

1.2 Problem Statement and Objectives

At its core, WayFinder tackles a fundamental information asymmetry. While Parcoursup data is technically public, accessing and interpreting it presents several interconnected challenges, the data is fragmented across multiple annual datasets spanning 2018 to 2024, each with its own quirks in column naming and structure. Even when students manage to find the right statistics, the raw numbers often lack the context needed to be meaningful: what does a 45% admission rate actually imply for a student with a particular baccalauréat type and grade profile? How do specialty choices influence chances at selective programs?

Beyond data access, students need personalized guidance. A generic admission rate tells only part of the story; what matters is how that rate translates to an individual's specific circumstances. Furthermore, comparing multiple programs requires standardized metrics and clear visualizations, which the raw data simply does not provide.

These challenges shaped our objectives as we set out to build an interface that makes searching and exploring formations intuitive, while simultaneously developing algorithms that estimate personalized admission probabilities based on student profiles. We also aimed to provide actionable specialty recommendations, enable meaningful multi-program comparisons, and ultimately leverage machine learning to surface insights that would be impossible to derive manually.

2 Informal Specifications

2.1 Scope and Constraints

2.1.1 Scope

The application encompasses all Parcoursup formations from 2018 through 2024, representing roughly 13,000 programs per year. It serves students holding any of the three French baccalauréat types: Général, Technologique, and Professionnel. Geographic coverage extends across all 24 académies, and the system accounts for the nine specialty subjects introduced with the baccalauréat reform of 2021.

2.1.2 Constraints

Several constraints shaped our technical decisions. The application requires Python 3.12 or higher along with internet connectivity to access the OpenData API, which occasionally throttles requests during peak periods. On the accuracy front, our probability calculations represent informed estimates rather than guarantees; we deliberately cap predictions below 100% to avoid false certainty. The scope remains limited to French domestic formations, excluding international programs and post-bac+1 admissions.

Perhaps the most significant constraint we encountered was the major methodology overhaul that occurred in 2021. The Ministry of Higher Education restructured how Parcoursup data is collected and organized, resulting in changed establishment names, columns that appeared or vanished between years, and fundamentally incompatible schemas. This disruption made seamless cross-year analysis far more challenging than we initially anticipated and required extensive harmonization work.

2.1.3 Assumptions

We assume users have basic familiarity with Parcoursup terminology and understand concepts like "voeux" (wishes), "taux d'accès" (access rate), and "mention" (honors). We also assume that historical trends provide reasonable, if imperfect, indicators of future patterns. Finally, we trust that users provide accurate profile information, as the quality of our recommendations depends directly on the truthfulness of their inputs.

2.2 Main Functionalities

Table 1 summarizes how capabilities evolved across our three versions. The progression illustrates our incremental development strategy: starting with essential features and gradually layering on sophistication.

Table 1: Main Application Functionalities by Version

Functionality	**Description**	**V0**	**V1**	**V2**
Formation Search	Query formations by name/establishment	✓	✓	✓
Statistics Display	View admission rates, wishes, capacity	✓	✓	✓
Formation Comparison	Compare up to 4 formations	✓	✓	✓
User Profile	Configure bac type, grades, specialties		✓	✓
Admission Probability	Personalized probability calculation		✓	✓
Specialty Recommendations	Doublette suggestions by formation type		✓	✓
Web Interface	Streamlit-based GUI		✓	✓
ML Clustering	K-Means formation grouping			✓
Similar Formations	KNN-based recommendations			✓
Historical Trends	Multi-year trend analysis			✓
Predictive Analysis	Linear regression forecasting			✓

2.3 Expected Outputs

When a user searches for formations, they receive a ranked list sorted by popularity, with each result displaying the program name, host institution, wish count, and access rate at a glance. Drilling into any formation reveals comprehensive statistics: not just overall admission metrics, but breakdowns by baccalauréat type and honors level, along with information about scholarship holders and geographic patterns.

The probability calculator goes further, providing a percentage estimate accompanied by contextual information explaining what factors help or hurt the user's chances. For students comparing multiple options, a side-by-side view aggregates key metrics and synthesizes specialty recommendations across all selected programs. In the advanced version, users can discover similar formations they might not have considered, explore how programs have evolved over recent years, and even glimpse projections of where trends might lead.

3 Development Plan

3.1 Stages and Milestones

We structured development around three distinct milestones, each representing a functional product that could stand on its own while laying groundwork for subsequent enhancements. Table 2 outlines what each version delivered.

Table 2: Development Milestones

Version	Milestone	Key Deliverables
V0	MVP (CLI)	Command-line interface with argparse, API integration for formation search, basic statistics extraction, and interactive menu system
V1	Core Features (GUI)	Streamlit web interface, user profile configuration, admission probability calculator, specialty recommendations, and Plotly visualizations
V2	Advanced Analytics	Machine learning clustering with K-Means, similar formation finder using KNN, historical trend analysis, predictive modeling via linear regression, and model persistence with pickle

3.2 Rationale for Implementation Order

Our development sequence was deliberately chosen to balance learning objectives with practical deliverables.

3.2.1 V0: Foundation and Core Logic

We began with a command-line interface precisely because it forced us to concentrate on what mattered most: the underlying logic. Without the distraction of designing a user interface, we could focus entirely on establishing robust API communication patterns, defining the data structures that would carry formation statistics throughout the application, and building a solid test foundation. The **_fetch_api_cached** function emerged during this phase as our solution to API rate limiting, while **extract_formation_stats** became the workhorse for parsing raw API responses into usable dictionaries. By the end of V0, we had a working product that, while spartan in presentation, proved our core concept was viable.

3.2.2 V1: User Experience and Interactivity

With the foundation in place, we turned our attention to making the application accessible to non-technical users. Streamlit provided an ideal framework: powerful enough to create a responsive web interface, yet simple enough that we could iterate quickly. Streamlit was seen as a better option compared to Flask, as we wished to focus mainly on organizing the consequent datasets and, it was also seen as a good alternative to visualize Parcoursup data with a better UI/UX. This phase introduced user profiles, enabling personalized probability calculations that accounted for baccalauréat type, expected honors, chosen specialties, and geographic factors. We integrated Plotly for visualizations, transforming abstract statistics into intuitive charts. The comparison feature, which required careful session state management to track selected formations across interactions, became one of the most technically challenging aspects of this version.

3.2.3 V2: Advanced Analytics and Machine Learning

The final version pushed into machine learning territory. We implemented K-Means clustering to automatically group formations by their statistical profiles, revealing natural categories that might not be obvious from names alone. A KNN-based recommendation system suggests similar formations based on these clusters, helping students discover programs they might otherwise overlook. Historical trend analysis aggregates data across years, while linear regression provides tentative forecasts. Throughout this phase, we wrestled with the tension between model sophistication and practical utility, ultimately favoring interpretable approaches over black-box solutions.

4 Design and Implementation

4.1 Architecture Overview

4.1.1 High-Level Architecture

Figure 1 illustrates how the various modules interact in the V2 application. At the top sits the Streamlit interface, orchestrating user interactions and delegating work to specialized modules below. The search module handles API communication and probability calculations, while the visualization module transforms data into charts. The data loader, introduced in V2, manages machine learning workflows and model persistence.

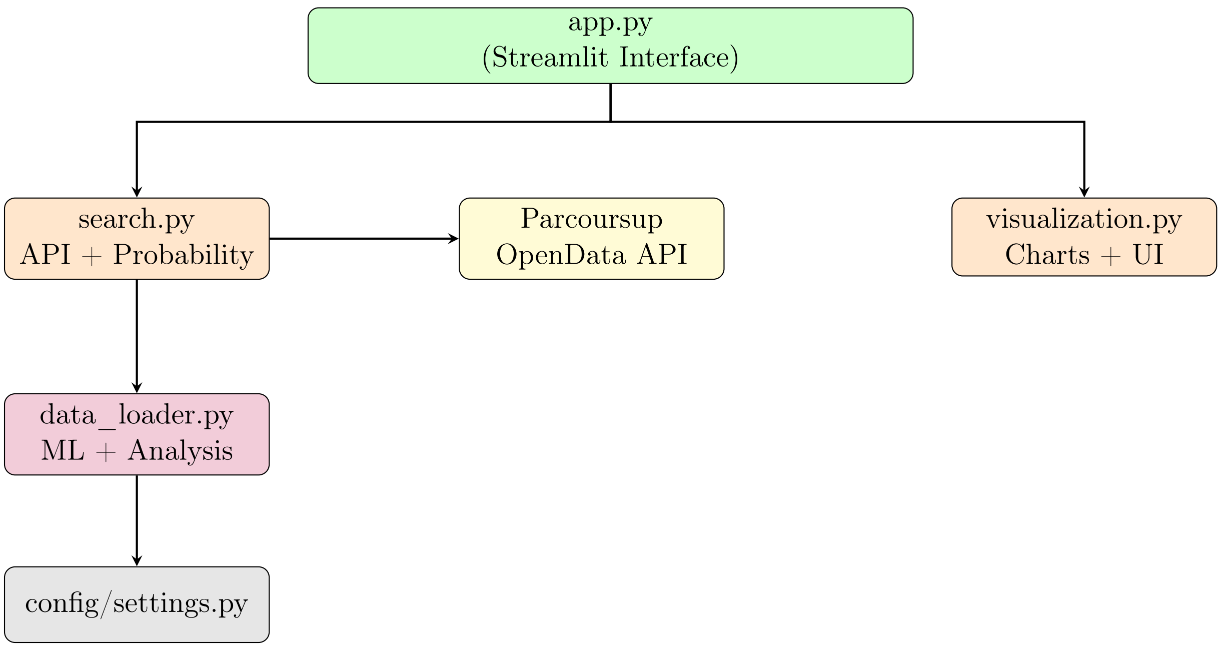

Figure 1: Module Architecture (V2)

4.1.2 Module Responsibilities

Each module in our architecture serves a distinct purpose, as detailed in Table 3. This separation of concerns proved invaluable during development, allowing team members to work on different modules simultaneously without stepping on each other's toes.

Table 3: Module Responsibilities

Module	**Responsibilities**
app.py	Streamlit entry point, page configuration, CSS styling, tab orchestration, session state management
search.py	API queries with caching, formation search, statistics extraction, admission probability calculation, specialty recommendations
visualization.py	Plotly chart styling, trend analysis rendering, clustering visualization, overview dashboard
data_loader.py	Multi-year data download, feature engineering, K-Means clustering, KNN similarity, model persistence
config/settings.py	Centralized configuration including logging parameters, API settings, and file paths

4.2 Key Data Structures

Two data structures form the backbone of information flow throughout the application.

4.2.1 Formation Statistics Dictionary

When the application retrieves information about a formation from the API, the **extract_formation_stats** function transforms the raw response into a standardized dictionary. This structure captures everything we need to know about a program: identifying information like name, establishment, and academy; admission metrics including access rate, wish counts, and capacity; distributions showing what percentage of admitted students held each baccalauréat type and honors level; and supplementary data about scholarship holders and geographic patterns. By centralizing this transformation logic, we ensure that every part of the application works with consistently structured data, regardless of how the underlying API format might vary between years.

4.2.2 Probability Result Dictionary

The output of our probability calculation, returned by **calculate_admission_probability**, goes beyond a simple percentage. It includes a confidence indicator reflecting how much historical data supports our estimate, along with categorized feedback: blocking factors that severely impact chances, warnings about moderate concerns, and positive factors working in the student's favor. A separate advice section offers concrete suggestions in French, while a details sub-dictionary exposes the individual factor values used in the calculation. This transparency helps users understand not just their estimated probability, but why we arrived at that number.

4.3 Key Algorithms

4.3.1 Admission Probability Calculation

At the heart of WayFinder lies our probability estimation algorithm. Rather than training a machine learning model on outcome data we don't have access to, we developed a heuristic approach that combines the formation's base access rate with multiplicative factors reflecting how well the student's profile matches historical admission patterns:

$$P = \text{taux\_acces} \times f_{bac} \times f_{mention} \times f_{spe}^{k} \times f_{boursier} \times f_{academie} \times f_{moyenne} \tag{1}$$

Each factor in Equation 1 adjusts the baseline probability based on a specific aspect of the student's profile. The baccalauréat type factor ranges from 0.3 to 1.0, reflecting the substantial advantage that Bac Général holders enjoy at many selective formations. The honors factor spans 0.4 to 1.3, rewarding students whose expected mention aligns with or exceeds the formation's typical admit profile. Specialty matching proves particularly important for competitive programs; we raise the specialty factor to an exponent for formations with access rates below 30%, amplifying its impact where it matters most. Geographic and scholarship factors account for quota-based admission patterns, while the grade factor provides a final adjustment based on overall academic performance.

4.3.2 Formation Type Detection

To provide relevant specialty recommendations, we must first identify what type of program the student is considering. The **get_formation_type** function examines the formation's name and track description, applying pattern matching to classify it into one of nine categories: scientific CPGE, business CPGE, medical PASS, computer science, engineering, law, economics, humanities, or a general fallback. The matching follows a priority order to handle ambiguous cases correctly; for instance, "CPGE MPSI" contains keywords that could match both scientific CPGE and general engineering, but by checking CPGE patterns first, we ensure proper classification.

4.3.3 K-Means Clustering

Our clustering algorithm groups formations based on three carefully chosen features: access rate (capturing selectivity), tension ratio (measuring popularity relative to capacity), and the percentage of admits with top honors (indicating academic caliber). We normalize these features using scikit-learn's RobustScaler, which handles the outliers common in Parcoursup data where a handful of elite programs have extreme values. The optimal number of clusters emerges from silhouette score analysis, testing configurations from 5 to 20 clusters and selecting whichever maximizes cohesion within clusters while maintaining separation between them. PCA then projects the results into two dimensions for visualization in the overview dashboard.

4.3.4 K-Nearest Neighbors for Similar Formations

Rather than training a single global similarity model, we maintain separate KNN models for each formation type. This design reflects the insight that meaningful similarity depends on context: a computer science program is most usefully compared to other computer science programs, not to law schools that happen to have similar admission statistics. When a user requests similar formations, we identify the query program's type, retrieve the corresponding KNN model, and return the nearest neighbors in the scaled feature space. Similarity scores are computed as inverse distance, providing an intuitive 0-to-1 scale.

4.4 Class Design

The data loading and machine learning functionality is encapsulated in two classes within the data_loader module.

ParcoursupDownloader handles the complexities of API communication. Because the primary endpoint occasionally fails, the class maintains a list of fallback URLs and automatically discovers which one is responding. Its methods cover testing connectivity, downloading data for specific years, and falling back to alternative API endpoints when the primary approach fails.

ParcoursupAnalyzer serves as the machine learning engine. It manages the full historical DataFrame alongside the latest year's snapshot, maintains trained model objects for scaling, clustering, and PCA, and stores per-type KNN models for similarity search. Its methods span the entire ML pipeline from data download through feature engineering, model training, and persistence via pickle serialization.

4.5 Modular Organization

Table 4 presents the directory structure of the V2 application. This organization keeps concerns cleanly separated while making dependencies explicit, as illustrated in Figure 1.

Table 4: Project Directory Structure (V2)

Path	Description
app.py	Streamlit entry point
search.py	API integration and probability calculation
visualization.py	Chart rendering and UI components
data_loader.py	ML pipeline and data management
download_data.py	CLI script for data downloading
config/	Configuration package with settings.py
tests/	pytest test suite
.streamlit/config.toml	Streamlit theme configuration
data/	Downloaded data directory
models/	Trained model files
requirements.txt	Python dependencies

5 User Guide

5.1 Installation and Execution

5.1.1 Prerequisites

Running WayFinder requires Python 3.12 or higher, the pip package manager, and an internet connection for API access.

5.1.2 Installation Steps

After cloning the repository, navigate to the project directory. We recommend creating a virtual environment using **python -m venv venv** and activating it with **source venv/bin/activate** on Linux or Mac, or **venv\Scripts\activate** on Windows.

For V0, the command-line version, navigate to the v0 directory and run:

pip install -r requirements.txt && pytest tests/ -v –disable-warnings && python main.py

For V1, the basic Streamlit version, navigate to the v1 directory and run:

pip install -r requirements.txt && pytest tests/ -v –disable-warnings && streamlit run app.py

For V2, the version with machine learning features, navigate to the v2 directory. While not strictly required, we strongly advise downloading the data first for optimal performance:

pip install -r requirements.txt && pytest tests/ -v –disable-warnings && python download_data.py && streamlit run app.py

Each command installs dependencies, verifies the installation through tests, and launches the application.

5.1.3 Dependencies

The application relies on several Python packages: streamlit (1.28.0+) for the web interface, pandas (2.0.0+) and numpy (1.24.0+) for data manipulation, plotly (5.15.0+) for visualizations, requests (2.31.0+) for API communication, and scikit-learn (1.3.0+) for machine learning in V2.

5.2 CLI Instructions (V0)

The command-line interface supports both interactive and direct modes. Running **python main.py** without arguments launches an interactive menu guiding users through available options. For scripted usage, **python main.py search "query"** performs a direct search, with optional flags like **–year** to specify the data year, **–limit** to cap results, and **-v** for verbose output. The **–help** flag displays complete documentation.

5.3 GUI Instructions (V1/V2)

5.3.1 Sidebar Configuration

The sidebar serves as the user's profile configuration panel. Here, students specify their baccalauréat type, expected honors, and for Bac Général holders, their two terminal specialties. Additional fields capture scholarship status, expected overall average, and home academy. These parameters feed directly into the probability calculation, so accuracy matters.

5.3.2 Tab Navigation

The main interface organizes functionality across tabs. "Rechercher" provides formation search with instant results as users type. "Statistiques" reveals detailed metrics for whatever formation is currently selected. "Mes Chances" displays the personalized probability calculation with explanatory feedback. "Comparatif" enables side-by-side comparison of up to four formations accumulated from searches.

In V2, three additional tabs appear. "Formations similaires" suggests programs resembling the current selection. "Tendances" charts how key metrics have evolved across available years. "Vue d'ensemble" presents the global clustering visualization and summary statistics.

5.4 Example Inputs and Outputs

Consider a student searching for "informatique" who receives 50 results sorted by popularity. Selecting "Licence Informatique" at Sorbonne Université and configuring a profile as Bac Général with expected "Très Bien" honors, specialties in Mathématiques and NSI, based in the Paris academy with a 16.5 average, the probability tab might display 72% with high confidence. The feedback would highlight the ideal specialty combination as a positive factor while noting popular alternative specialty pairs observed among similar formations.

6 Evaluation and Discussion

6.1 Level of Completion

Table 5 summarizes what we delivered across all three versions. Core features work as intended, with predictive analysis marked as partial due to the inherent limitations of simple linear regression on complex admission dynamics.

Table 5: Feature Completion Status

Feature	**V0**	**V1**	**V2**	**Notes**
Formation search	✓	✓	✓	Fully functional
Statistics display	✓	✓	✓	All metrics included
CLI interface	✓			argparse-based
Web interface		✓	✓	Streamlit framework
User profile		✓	✓	Six parameters
Probability calc		✓	✓	Multiplicative model
Specialty recs		✓	✓	Eight formation types
Comparison	✓	✓	✓	Up to four formations
ML clustering			✓	K-Means with PCA
Similar formations			✓	KNN by type
Historical trends			✓	Multi-year analysis
Predictions			∼	Basic linear regression
Tests	✓	✓	✓	pytest coverage

6.2 Known Limitations

6.2.1 Abandoned Data Source: Student Pathway Tracking

Our original vision was even more ambitious. We discovered a dataset on data.gouv.fr called "Parcours d'études dans l'enseignement supérieur" that tracks individual student trajectories across French higher education since 2012. Imagine being able to show a prospective student where previous admits from their target program transferred, dropped out, or graduated: this would have allowed us to understand student profiles since 2012, which choices they are making, and why. Unfortunately, this data proved unusable; the documentation explicitly states that establishment identifiers have been deliberately obfuscated for privacy protection. Despite weeks of effort attempting to decode these UAI codes using official nomenclature databases and Ministry of Education tools, we never cracked the mapping. Without linking trajectories to recognizable institution names, the data remains effectively encrypted. We ultimately abandoned this approach entirely and focused on what we could actually deliver with Parcoursup statistics.

6.2.2 The 2021 Methodology Disruption

The 2021 reform created a fracture in Parcoursup data that we could not fully bridge. When the Ministry restructured how information is collected and organized, establishment names changed, columns appeared or vanished, and schemas diverged fundamentally from prior years. Attempting to force compatibility between 2018-2020 and 2021-2024 data proved futile at the individual formation level.

Our workaround involved a dual-track approach. For formation-level analysis where precision matters, we restricted ourselves to post-2021 data, ensuring consistent schemas and reliable comparisons. For broader trend analysis where exact program matching is less critical, we developed separate processing pipelines for each era, then aggregated results at the formation-group level (law, medicine, engineering, etc.) where category-level patterns remain meaningful despite underlying schema changes. Users can observe long-term sectoral trends, even if specific program-level history stops at 2021.

6.2.3 External Dependencies

Our reliance on the OpenData API introduces unavoidable fragility. During high-traffic periods, the API throttles requests, and occasional outages occur without warning. We mitigate this through aggressive caching, fallback URLs, and retry logic with exponential backoff, but users may still experience slowdowns during peak times. Additionally, column names shift between years, requiring ongoing maintenance of our harmonization mappings whenever new data releases reveal new inconsistencies. However, in the V2 when data are downloaded the dependency becomes less problematic as we do not need the API each time.

6.2.4 Model Limitations

Without access to actual admission outcome data linking student profiles to decisions, we cannot train a proper predictive model. Instead, we infer factor weights from aggregate statistics and domain knowledge. The resulting estimates are informed guesses rather than predictions grounded in labeled training data.

Linear regression for trend forecasting assumes patterns continue smoothly, which reality often contradicts, a program gaining sudden popularity from a curriculum revamp or losing appeal from reputational damage will be poorly served by extrapolation. We include confidence intervals to signal uncertainty, but users should treat projections as tentative hypotheses rather than reliable forecasts.

Our specialty recommendations draw from a manually curated dictionary rather than learned patterns. While we based these rankings on official guidance and domain expertise, they may lag behind evolving formation requirements or miss emerging specialty combinations that the data would reveal.

6.2.5 Data Quality

Not every formation reports complete statistics. When fields are missing, we default to zero rather than crashing, but this can understate metrics for programs with incomplete data. A future enhancement might flag such cases explicitly in the UI so users know when they're working with potentially incomplete information.

6.3 Main Difficulties and Solutions

Table 6 summarizes the major obstacles we encountered and how we overcame them.

Table 6: Challenges and Solutions

Challenge	**Solution**
Unusable student pathway dataset	Abandoned trajectory data after failing to decrypt UAI codes; pivoted to Parcoursup statistics exclusively
2021 methodology disruption	Dual-track analysis: formation-level restricted to 2021-2024; group-level analysis with separate pre/post pipelines
API instability and timeouts	Fallback URLs, retry logic with exponential backoff, aggressive response caching
Schema changes across years	Comprehensive column mapping dictionary with harmonization function
Streamlit session state complexity	Careful initialization patterns, unique widget keys, explicit state mutation ordering
Large data volumes	Batch downloading with progress tracking, feature selection, model persistence via pickle
Test isolation from Streamlit	Architectural separation of pure functions from UI components

Overall, we had three main problems during this project:

The abandoned trajectory dataset deserves special mention because it consumed significant early effort. The "Parcours d'études" data looked perfect on paper: detailed student pathways, longitudinal tracking, rich outcome information. Only after extensive decoding attempts did we accept that the privacy obfuscation was too robust for us to circumvent. This experience taught us to validate data usability before committing to a data source.

The API instability challenge shaped our entire architecture. When the primary endpoint started failing intermittently during development, we implemented a fallback system that automatically discovers working endpoints. This resilience proved its worth repeatedly when the primary URL experienced extended outages during critical testing phases.

Lastly, session state management in Streamlit turned out to be trickier than anticipated, particularly for the comparison feature. Users expect their selected formations to persist as they navigate between tabs, which requires careful state tracking. We eventually settled on explicit initialization checks at each tab's entry point combined with unique key generation for dynamically created widgets, eliminating the collisions that plagued early versions.

6.4 Possible Extensions

Several directions could enhance WayFinder in future iterations.

On the machine learning front, obtaining actual admission outcome data through research partnerships would enable supervised learning approaches with properly validated accuracy. Gradient boosting could replace linear regression for trend forecasting, capturing nonlinear dynamics that our current model misses. Moreover, natural language processing might analyze formation descriptions to surface semantic similarities invisible in numerical statistics alone.

User-facing improvements could include persistent accounts with saved preferences and favorites, enabling students to track how their target formations evolve over time. Email notifications when key statistics change significantly would transform WayFinder from a one-time consultation tool into an ongoing planning companion. A mobile application would meet students where they spend most of their digital time.

Data expansion could integrate employment outcomes from ministry surveys, helping students evaluate formations based on career trajectories rather than admission statistics alone. Cost-of-living data by city would help students assess the full financial implications of their choices, particularly relevant for those considering formations far from home.

Technical improvements could include migrating to a PostgreSQL backend for faster queries, implementing asynchronous API calls for improved responsiveness, and containerizing with Docker for simplified deployment and consistency across environments.

7 Conclusion

7.1 Key Learnings

This project reinforced several important lessons, both technical and methodological.

On the technical side, we gained deep appreciation for robust API communication patterns. Caching, retry logic, and fallback mechanisms aren't optional niceties; they're essential infrastructure when depending on external services. Working with real-world data hammered home the importance of defensive coding: null checks, type validation, and schema normalization saved us from countless runtime errors. The machine learning components taught us that model sophistication matters less than appropriate problem framing; our simple heuristic approach, while imperfect, provides genuinely useful estimates precisely because it's interpretable and aligned with available data.

Methodologically, the incremental development strategy proved its worth. Having a working product at each milestone meant we always had something to show and something to build upon. Documentation-first development, where we wrote specifications before code, forced clarity in design that paid dividends during implementation. Most importantly, keeping actual students in mind throughout development guided our prioritization: features that helped real users trumped technically interesting but practically useless capabilities.

The collaboration experience demonstrated how clean module boundaries enable parallel work. When responsibilities are clearly delineated, as shown in Figure 1, team members can contribute simultaneously without conflicts. Code review through pull requests caught bugs early and spread knowledge across the team.

7.2 Final Remarks

WayFinder delivers what we set out to build: a functional tool that transforms opaque Parcoursup data into actionable insights for students navigating one of the most important decisions of their academic lives. From a command-line prototype to a full-featured web application with machine learning capabilities, the project demonstrates how public data can be leveraged to address real problems.

Our probability model, while heuristic, provides estimates that help students calibrate their expectations. Our specialty recommendations, while manually curated, reflect genuine domain knowledge about what selective formations value. Our trend analysis, while limited by the 2021 disruption, still surfaces patterns that would be invisible in raw data.

Future work could enhance prediction accuracy through supervised learning, expand coverage to international programs, and deepen integration with outcome data. But even in its current form, WayFinder represents a meaningful step toward democratizing access to information that can genuinely help students make better choices about their futures.