|  |
| --- |
|  |
| Pathfinder |
| Personalized Career Path Recommender System for University Students |

|  |
| --- |
| CSIS-4495 - 050  Leshan Chathuranga Kuruppuarachchi – 300363077 Hyunhee Kim - 300372531  11-27-2025 |

Table of Contents

[1 Introduction 3](#_Toc215157711)

[2 Proposed Research Project 4](#_Toc215157712)

[2.1 Methodology 4](#_Toc215157713)

[2.2 Technologies to be used 4](#_Toc215157714)

[2.3 Expected Results 5](#_Toc215157715)

[3 Project Planning and Timeline 6](#_Toc215157716)

[4 Implemented Feature: Web scraper 8](#_Toc215157717)

[4.1 Details of Implementation: Indeed Scraper 8](#_Toc215157718)

[4.2 Details of Implementation: Coursera Scraper 11](#_Toc215157719)

[5 Implemented Feature: Recommendation model 13](#_Toc215157720)

[5.1 Details of Implementation: Job Recommendation Model 13](#_Toc215157721)

[5.2 Details of Implementation: Course Suggestion Engine 15](#_Toc215157722)

[5.3 Details of Implementation: recommendation api 17](#_Toc215157723)

[6 Implemented Feature: personality profiling model 18](#_Toc215157724)

[6.1 Overview 18](#_Toc215157725)

[6.2 Details of Implementation: k-means personality clustering 18](#_Toc215157726)

[6.3 Details of Implementation: randomforest classification 20](#_Toc215157727)

[7 Lessons Learned and Future Work 23](#_Toc215157728)

[7.1 Lessons Learned by Technical Development 23](#_Toc215157729)

[7.2 Lessons Learned from System Integration and Fast API 23](#_Toc215157730)

[7.3 Personal Growth, Teamwork, and Career-Relevant Learning 24](#_Toc215157731)

[7.4 Future Work 24](#_Toc215157732)

[8 Concluding Remarks 25](#_Toc215157733)

[9 References 26](#_Toc215157734)

[10 Appendix 27](#_Toc215157735)

[10.1 Appendix A: Installation Guide 27](#_Toc215157736)

[10.1.1 Backend Installation 27](#_Toc215157737)

[10.1.2 Frontend Installation 28](#_Toc215157738)

[10.2 Appendix B: User Guide 28](#_Toc215157739)

[10.3 Appendix C: Dataset and API Used 30](#_Toc215157740)

[10.4 Appendix D: Hardware, Software, Architecture 30](#_Toc215157741)

[10.4.1 Hardware Requirements 30](#_Toc215157742)

[10.4.2 Software Requirements 30](#_Toc215157743)

[10.4.3 Architecture Overview 31](#_Toc215157744)

[10.5 Appendix E: Code Explanation 31](#_Toc215157745)

[10.5.1 Pathfinder database 31](#_Toc215157746)

[10.5.2 Scheduler 35](#_Toc215157747)

[10.5.3 API request Response 37](#_Toc215157748)

[10.5.4 Loading Machine Learning Models (personality\_model.py) 38](#_Toc215157749)

[10.5.5 Personality Prediction Endpoint (main.py) 38](#_Toc215157750)

# Introduction

In today’s competitive job market, university students face increasing challenges when choosing their future career path. The traditional methods of career finding, such as personality tests, career questionnaires, or career advisers often fail to understand individual skills and interests, because they are too broad and fail to understand individual differences in skills, academic background and personality.

These limitations have led to interest in the application of intelligent systems in the field of career counseling and education technology.

To overcome these limitations, personalized career recommendation system has emerged as a solution. These systems collect user-provided information such as gender, academic background, skills, interests, and analyze the data to generate personalized recommendations. Based on this context, this research addresses two key questions:

1. How can machine learning models improve the accuracy and related to these recommendations compared to traditional methods?

2. How can student-provided information be used effectively to generate personalized career recommendations?

These questions are important because wrong career choices can lead to poor experience for students and labor market needs. By developing a data-driven and personalized recommendation system, this research aims to help students make confident decisions about their future careers.

Recent career-recommendation studies range from classical ML pipelines to modern hybrid ensembles. Classical content-based pipelines [1] preprocess user-provided descriptions, vectorize with TF-IDF, segment users via K-Means, reduce dimensions with PCA, and rank careers using cosine similarity; a full-stack reference implementation integrates these components into a web app (e.g., Django). Hybrid stacking ensembles [2], recent work for competency-based education creates a structured student dataset (5 features, 5,000 records) and combines Deep Neural Networks and Random Forest via stacking to recommend STEM tracks, reporting ~90.06% accuracy and 92.07% precision with 5-fold CV. Mobile applications such as CareerX [3] have demonstrated the integration of NLP and machine learning into mobile applications, with user satisfaction rates of around 90%. Other studies have highlighted Hybrid filtering approaches [4], combining collaborative and content-based methods to generate personalized educational roadmaps. Hierarchical multi-tiered systems [5] have also been developed to address sparsity and cold-start problems, achieving over 99% accuracy across diverse career datasets.

Despite these advances, several gaps remain. Classical pipelines often rely on limited or generic features, while hybrid models tend to be tailored to narrow contexts such as STEM education. Feedback mechanisms are rarely integrated in near real-time, leaving evaluations dependent on static, offline metrics. Furthermore, most systems lack generalizability, as results validated in one population may not transfer well to others without adaptation.

Based on these gaps, this research assumes that a machine learning based recommendation system using effectively diverse student-provided information will significantly enhance personalization and accuracy. The expected benefit is to provide students with reliable, data-driven guidance for career choices, while contributing to the development of intelligent systems in education technology.

# Proposed Research Project

The proposed Pathfinder project is a data-driven experimental design. The main objective is to build a personalized career recommendation system that uses user provided information, such as educational background, gender, skills and interests, to recommend personalized career pathways.

## Methodology

Pathfinder follows a four-step methodology, data collection and preprocessing, model training and recommendation engine design, skill gap analysis and job market trend visualization.

First, training data will be collected from public sources such as Kaggle and through web scraping platforms like LinkedIn and Indeed. Operational data will consist of information provided directly by users via Pathfinder website, including major, GPA, gender, interest, skill, and personality traits. Next, the collected data will be used to train models capable of recommending career paths and personality traits to the suggested career path based on similarity measures such as cosine similarity, KNN, or other machine learning classifiers.

Following recommendation generation, a skill gap analysis will be conducted by comparing the requirements of each recommended career with the user’s current skills. To complement this, job market datasets will be analyzed to highlight trends and demand for recommended careers over the last several years. Finally, the system will be implemented as a functional platform that integrates all these components.

## Technologies to be used

This project will integrate several technologies

• Operating System: The system will run on both Windows and mac environments.

• Programming Language/Frameworks: Python will be the primary language, using machine learning libraries such as scikit-learn, TensorFlow, and PyTorch.

• Database: MySQL or H2 database will be used to store cleaned data for model and visualization data since those databases are open source and easy to config.

• Frontend: React or Vue.js will be used to build a responsive and interactive user interface that allows users to input their profiles and view career recommendations, skill gap analysis, and job market trends.

• Backend: FastAPI or Flask will serve as the backend to manage model inference, skill gap analysis, and job trend visualization.

• Visualization Tools: Libraries such as Chart.js, D3.js, or Recharts will be used to display skill gaps and job market trends on the dashboard.

• Collaborative Development: GitHub will be used for version control, source code management, and team collaboration. Team members will work together using branches, pull requests and reviews during development.

## Expected Results

The expected outcomes of this research are multifaceted. First, the system is anticipated to provide improved accuracy and personalization compared to traditional career counseling methods. Each user will receive the top three to five most suitable career recommendations, offering clear and actionable pathways. Alongside these recommendations, users will also receive a breakdown of the skills they lack, the importance of these skills, and the estimated effort required to bridge the gap.

Additionally, users will gain insights into job market trends, with visualizations of labor market data over the last five years to highlight demand in various careers. The platform will be designed as a user-friendly and interactive tool that encourages students and job seekers to explore potential career paths and refine their skills continuously

Ultimately, this project will contribute to the advancement of educational technology by developing an intelligent system that supports students in making better-informed career decisions while also aligning educational outcomes with labor market needs.

# Project Planning and Timeline

This section presents the planned schedule for completing the personalized Career Path Recommender System. The project is divided into clear phases, each with specific milestones and deliverables, to ensure steady progress and timely completion. Table 1 contains the timeline which follows the official course deadlines for the proposal, progress reports, midterm and final submissions.

Table 1:Proposed project schedule

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | Duration/  Deadline | Milestones | Deliverables |
| Phase 1:  Literature Review & Project Initialization | 2025/09/04 – 2025/09/11 | Collect references on career recommendation systems | Project Proposal |
| Phase 2:  Data Collection & Preprocessing | 2025/09/08 – 2025/09/25 | * Identify data sources * Implement scraping & preprocessing | Cleaned dataset |
| Phase 3:  System Design | 2025/09/12 – 2025/09/25 | * Define system architecture (frontend, backend) * Design recommendation logic | * System architecture diagram * Progress Report 1 |
| Phase 4:  Recommendation Model Development | 2025/09/17 – 2025/10/16 | * Implement recommendation model * Train/test with sample student data | * Working recommendation engine * Progress Report 2 |
| Phase 5:  Frontend & User Interaction | 2025/10/08 – 2025/11/05 | * Build React UI for student input * Connect frontend to backend API | * Functional prototype * Midterm Report * Progress Report 3 |
| Phase 6:  Testing & Refinement | 2025/11/03 – 2025/11/18 | * Test system with multiple profiles * Debug & bug fixing | * Progress Report 4 * Midterm Video Report |
| Phase 7:  Documentation & Final Report | 2025/11/12 – 2025/11/27 | * Prepare final report and presentation | * Progress Report 5 * Final Report & Implementation * Project Defense |

**Responsibilities**

Leshan Chathuranga Kuruppuarachchi

• Overall project management and coordination

• Data scraping design and implementation

• Model design and implementation

• Frontend development

• Testing and bug fixing

Hyunhee Kim

• Data collection and preprocessing

• Model design and implementation

• Backend API development

• Data visualization implementation

• Testing and bug fixing

• Documentation

**Project Timeline Gantt Chart**

**A graph with blue rectangular bars

AI-generated content may be incorrect.**The figure 1 below shows the project timeline Gantt chart

Figure 1: Proposed project Gantt chart

# Implemented Feature: Web scraper

The Automated Web Scraping Module is a core feature of the system designed to continuously collect real-time labor-market data and up-to-date learning resources. Every hour, the system automatically scrapes

* Indeed (Canada) > job postings, job titles, company names, descriptions
* Coursera > relevant courses, skills taught, organization, ratings, url

The purpose of this feature is to ensure the Pathfinder platform always has fresh, accurate, and relevant data for generating job recommendations, skill-gap analysis, and course suggestions for users. This scraping task runs unattended, triggered by an automated scheduler. It cycles through a predefined list of 100+ job titles, ensuring that all occupations receive equal scraping frequency.

## Details of Implementation: Indeed Scraper

The IndeedScraper module is responsible for extracting job postings from the Canadian Indeed website. The scraper targets Page 1 results for a given job title and captures structured information about each job. This ensures that the Pathfinder system has access to real-time job data to support its recommendation and analytics features.

This module is implemented using Python and Selenium WebDriver with Chrome, with the ChromeDriverManager handling automated driver installation. As shown in figure 2, the scraper is configured to run in headless mode with several Chrome options, including custom user-agent, window size, and flags to disable automation detection and GPU rendering.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 2: indeed\_scraper.py part 1

A computer screen shot of a program code

AI-generated content may be incorrect.

Figure 3: indeed\_scraper.py part 2

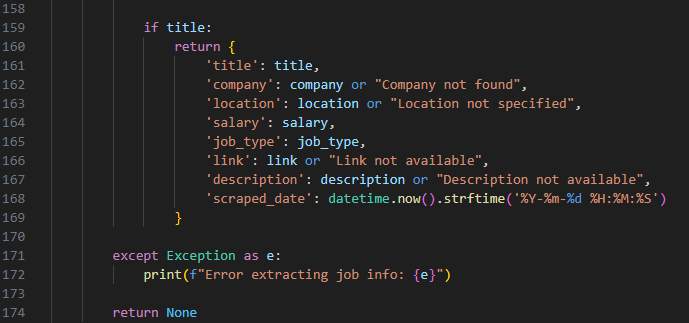
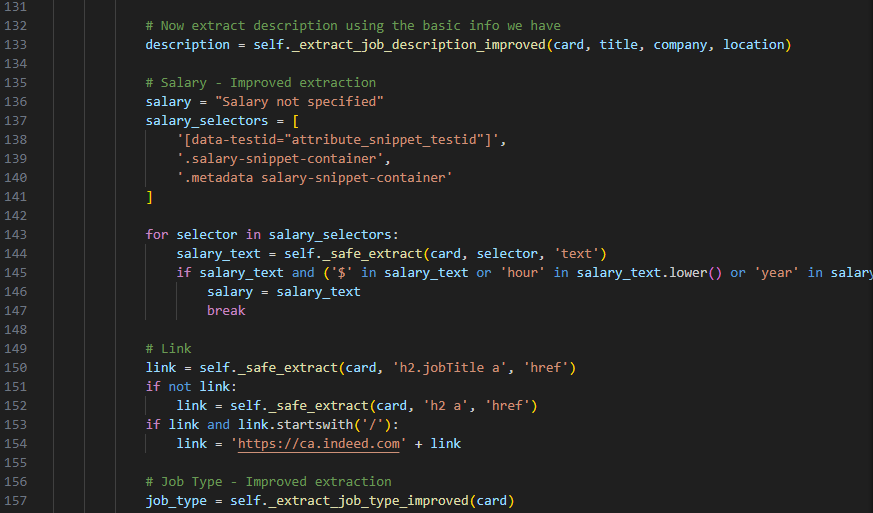
When executed, the scraper navigates to the constructed search URL based on the job title and location. It waits for job cards to load on the page and handles any cookie consent popups automatically to prevent interruptions as shown in figure 3. Each job card is then processed to extract structured information including job title, company name, location, salary, job type, job description, and the link to the posting as shown in figure 4 and 5. To handle the dynamic structure of the Indeed website, multiple CSS selectors and fallback logic are used for each field, ensuring robust extraction even if some elements are missing or the layout changes. The job description is carefully extracted, first attempting structured snippets and then filtering relevant text from the job card, while limiting the length for database storage. Finally, the scraped data is timestamped and saved into the SQLite database, making it available for recommendation engines and analytics modules.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4: indeed\_scraper.py part 3

Figure 5: indeed\_scraper.py part 4



The scraper includes additional logic to avoid detection or being blocked by the website as shown in figure 6. Delays are introduced between card extractions, and the system cycles through a predefined list of job titles to ensure that all occupations receive equal attention. This approach allows continuous, automated scraping of real-time job postings every hour without manual intervention, providing a reliable and up-to-date dataset for the Pathfinder platform.

A computer screen shot of code

AI-generated content may be incorrect.

Figure 6: indeed\_scraper.py part 5

## Details of Implementation: Coursera Scraper

The CourseraScraper module extracts online course information relevant to the job title being scraped from Indeed. This allows the Pathfinder platform to recommend learning resources and skills development opportunities aligned with the labor market demands.

Upon execution, the scraper navigates to the Coursera search results page for the given job title and waits for course cards to appear as shown in figure 7. Each course card is processed to extract structured data such as course title, providing organization or university partner, skills taught, course rating, number of students enrolled, and the course URL. The scraper uses multiple fallback selectors to handle variations in page structure and ensure reliable extraction. Text data is cleaned and normalized to remove unnecessary content, producing a consistent dataset suitable for database storage.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 7: coursera\_scraper.py

The scraped course data is timestamped and stored in the SQLite database, aligning with the job data from Indeed. This integration allows the Pathfinder platform to provide relevant learning resources alongside job recommendations. By running hourly, the scraper continuously updates the dataset with new courses, supporting real-time insights into available training resources. The CourseraScraper is designed for automation, resilience to page layout changes, and scalability, enabling Pathfinder to maintain a comprehensive and current mapping of jobs to associate skills and courses.

# Implemented Feature: Recommendation model

The Pathfinder system includes an intelligent recommendation engine designed to assist users in identifying the most suitable job roles and skill-building courses based on users’ education and skill information. This feature is composed of two tightly integrated components:

## Details of Implementation: Job Recommendation Model

The Job Recommendation Model is responsible for predicting the most suitable job title based on textual job descriptions. The implementation begins by retrieving job postings stored in the local SQLite database using the PathfinderDatabase class. From the dataset, job descriptions are extracted as feature text, while the job title serves as the target label as shown in figure 8.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 8: job\_model.py part 1

To convert the raw text into machine-interpretable numerical vectors, a TF-IDF Vectorizer is applied, configured with a vocabulary size of 10,000 features and an n-gram range of (1, 200). This enables the model to capture both single words and longer phrases commonly found in job postings. After vectorization, job titles are transformed into encoded integer labels using a LabelEncoder, preparing the data for training.

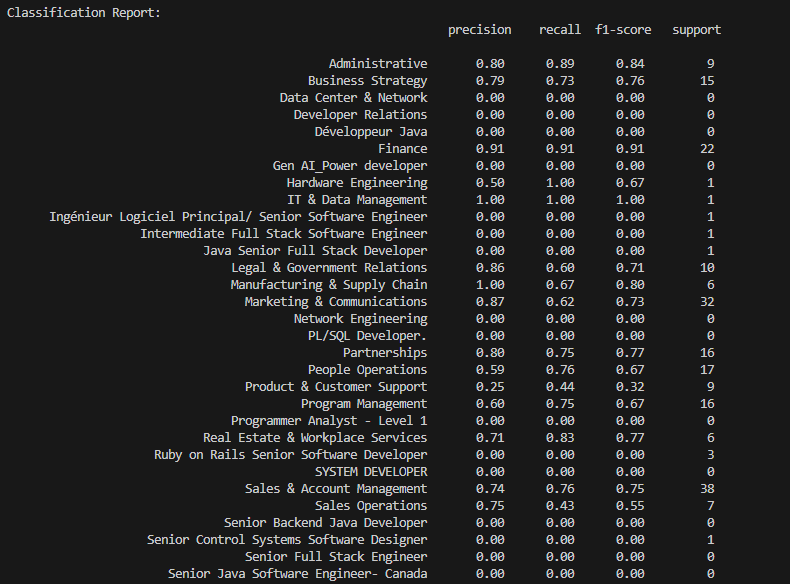
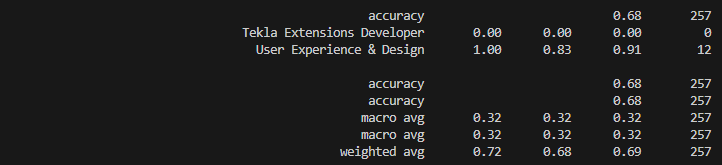


Figure 9: job\_model.py classification report

To improve the model’s ability to generalize across job categories with unequal representation, RandomOverSampler (ROS) is applied to the training data. This oversampling strategy balances the distribution of job titles by replicating minority-class examples, addressing the common issue of imbalanced job datasets. The classification algorithm used is Logistic Regression with class\_weight='balanced' and an increased max\_iter threshold to ensure convergence with high-dimensional TF-IDF vectors. The pipeline then performs prediction on the test set, calculates accuracy, and generates a detailed classification report as shown in figure 9. Finally, the trained model (saved\_model.pkl), the TF-IDF vectorizer (tfidf.pkl), and the label encoder (label\_encoder.pkl) persisted to disk as shown in figure 10. A background scheduler retrains this model every 30 days, ensuring that predictions remain accurate as job descriptions change over time.

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 10: job\_model.py part 2

## Details of Implementation: Course Suggestion Engine

The Course Suggestion Engine identifies relevant online courses a user should take to acquire missing job-related skills. The system begins by retrieving two datasets from the database: job postings and course listings. Each course contains a comma-separated list of skills, which are preprocessed into Python sets for fast intersection operations. A global vocabulary of skills is constructed from all course listings, allowing the system to detect those skills within job descriptions using a case-insensitive exact-word regex match as shown in figure 11.

A computer screen shot of text

AI-generated content may be incorrect.

Figure 11: recommend.py part 1

To generate recommendations, the engine performs skill-gap analysis. Given a target job and a user’s existing skills, the system extracts the required skills from the job description using the previously constructed global skill list as shown in figure 12. The missing skills are computed by subtracting the user’s skill set from the job-required skills. For each course, the engine computes a coverage score based on the proportion of missing skills addressed by the course. Courses are then ranked using a multi-criteria scoring system prioritizing (1) coverage score and (2) course rating. The top five ranked courses are returned along with the list of missing skills, providing a concise and actionable learning path for the user.

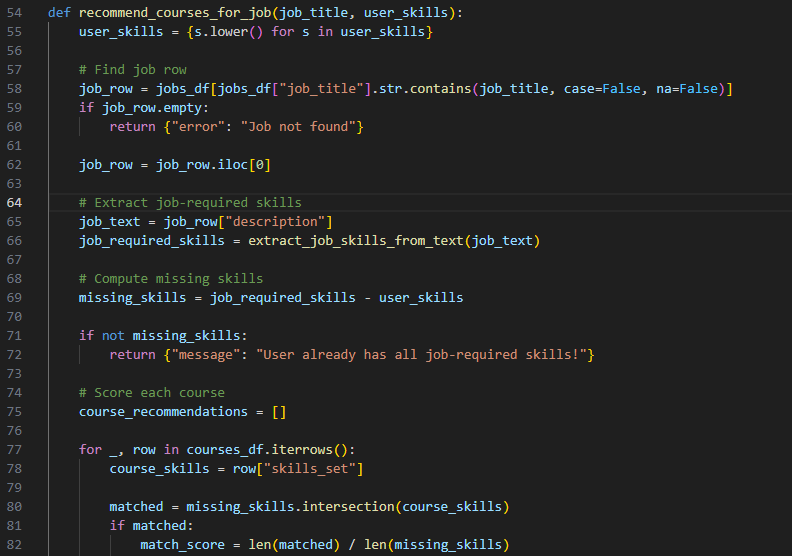


Figure 12:recommend.py part 2

## Details of Implementation: recommendation api

The recommendation API endpoint evaluates a user’s profile to determine the most suitable job categories and corresponding courses. As shown in figure 13 the system begins by receiving a validated UserProfile object from the client, containing the user’s education, GPA, skills, and interests. These inputs are passed through a preprocessing pipeline that converts categorical attributes into encoded representations and transforms text-based fields into numerical vectors using the same TF-IDF and scaling models used during training. The processed vector is then fed into the machine-learning classifier, which outputs probability scores for all job categories. The top five categories are selected by sorting the prediction probabilities and mapping the highest-scoring indices back to human-readable labels through the fitted label encoder.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 13:main.py

Once the top job roles are identified, the API proceeds with skill-gap detection and course recommendation generation. For each predicted job category, the system invokes the recommend\_courses\_for\_job module, which compares the required skills for that role with the user’s existing skill set. Missing skills are identified and used to query a curated set of course listings to determine the most relevant training options. Each recommendation entry includes the job role, model-generated confidence score, list of missing skills, and a set of matched online courses. The endpoint compiles these results into a structured JSON response, enabling the frontend to present personalized, actionable career guidance to the user.

# Implemented Feature: personality profiling model

## Overview

The personality profiling feature is one of the core components of our Career Path Recommendation System. This feature analyzes a user’s personality traits through a customized OCEAN-based questionnaire and uses machine learning models to determine the user’s personality type. The initial project proposal aimed to build a personality-driven recommendation pipeline by combining user characteristics, job requirements, and skill gaps. Although the final implementation did not fully achieve a personality-to-job matching system, we successfully developed a complete pipeline that evaluates user personality traits, classifies the user into one of five personality clusters, and generates a structured personality report.

During the proposal stage, we planned to adopt the original Big Five assessment format. However, after conducting early testing and receiving feedback during the midterm presentation, we revised the interaction design to use 25 selected questions with a 0–5 integer scale, which more closely reflects standard personality assessments. We also refined our personality descriptions to provide clearer explanations so users can better understand the results.

This feature serves as a foundational element for future enhancements, where personality types could be aligned with job categories, required competencies, and course suggestions in a full recommendation pipeline.

## Details of Implementation: k-means personality clustering

The first stage of the personality feature involved clustering personality scores using K-Means, an unsupervised machine learning algorithm. We collected and analyzed the dataset to identify the optimal number of clusters using the Elbow Method and silhouette evaluation, and after multiple experiments, we selected K = 5, which produced clearly separable and interpretable clusters as shown in figure 14.

라인, 텍스트, 그래프, 도표이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

Figure 14: K-Means elbow plot

During data preprocessing, we scaled all 25 personality question responses using the StandardScaler to ensure consistent variance across features, converted all inputs into an integer-based 0–5 format to match the structure of the personality test, and applied reverse-scoring to negatively keyed items so that all questions aligned directionally. Additionally, normalization procedures were applied to maintain equal weight across the five OCEAN dimensions.

스크린샷, 텍스트, 다채로움이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

Figure 15: K-Means Cluster visualization

Figure 15 shows the 3D PCA visualization of the five K-Means personality clusters.

Once preprocessing was complete, the K-Means model successfully grouped users into five distinct personality categories. Rather than presenting users with raw numeric cluster labels, we created meaningful and readable category names, such as the Balanced Collaborator and the Reserved Analyzer. For each of these clusters, we wrote extended personality descriptions to help users better understand their behavioral tendencies, strengths, and areas for improvement.

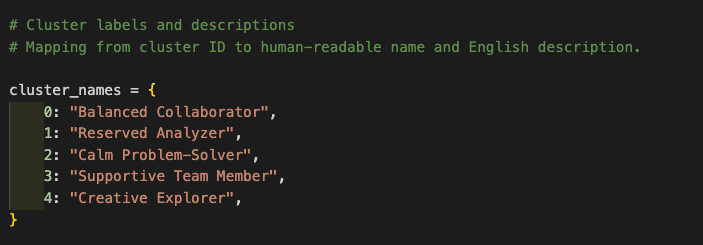
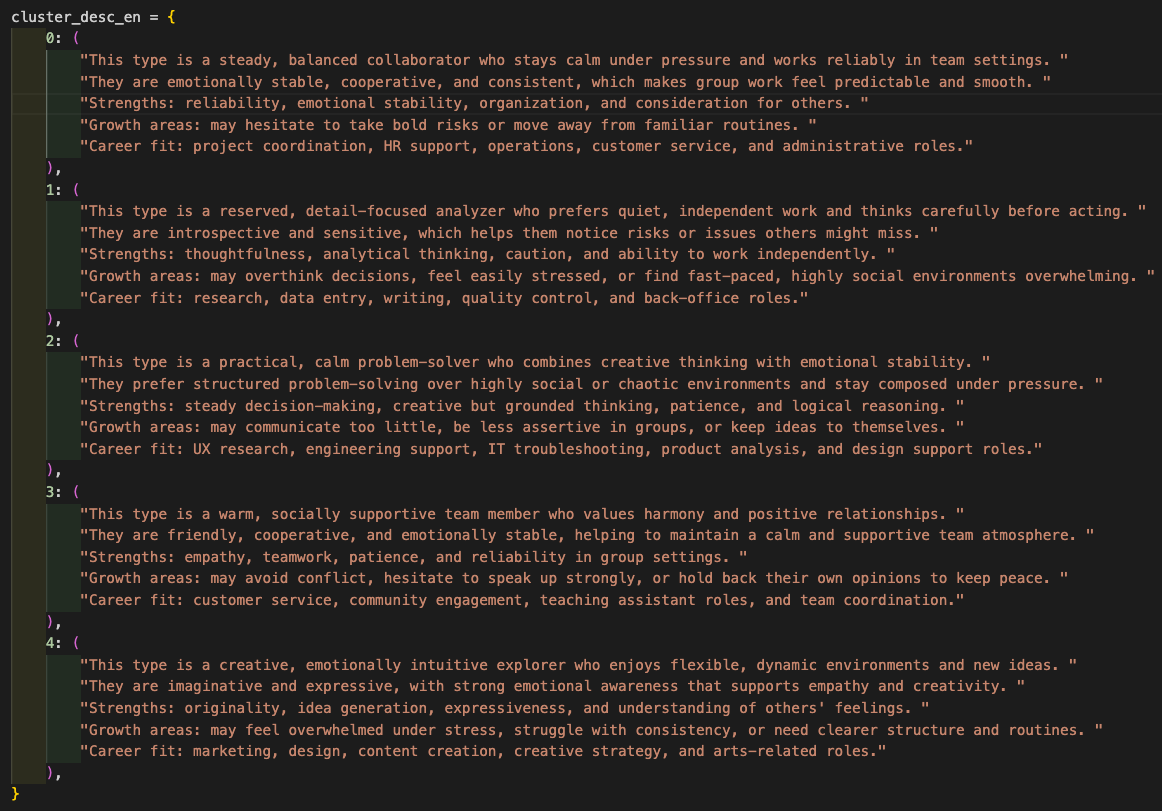


Figure 16: Example Cluster Description Screenshot

As shown in figure 16 these descriptions include strengths, tendencies, communication style, and potential areas for growth.

## Details of Implementation: randomforest classification

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

Figure 17: personality\_model.py Screenshot

Figure 17 shows how the Random Forest model and scaler are loaded.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

Figure 18: API endpoint Definition in Swagger UI

Figure 18 shows the /personality POST endpoint documented in Swagger UI. It outlines the endpoint description, expected JSON input format, and example request body. This confirms that the FastAPI backend exposes a well-defined interface for receiving OCEAN personality scores.

스크린샷, 텍스트, 폰트, 다채로움이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

Figure 19: Random Forest Pipeline Diagram

Figure 19 illustrates how the Random Forest classifier is trained and deployed. OCEAN scores are combined with cluster labels generated by K-Means to train the model. After evaluation, the trained model and scaler are saved and loaded by the FastAPI backend. When the frontend sends new OCEAN scores to the /personality endpoint, the backend scales the input, predicts the cluster, and returns the cluster ID, name, description, and probability distribution as a JSON response.

To provide real-time predictions on the website, we trained a Random Forest classifier using the cluster labels generated from the K-Means model. This transformed the unsupervised clustering into a deployable supervised prediction model. The model, along with the scaler, was saved as .pkl files for integration.

We implemented the backend using FastAPI, creating an endpoint /predict that:

1. Receives the 25 trait responses from the frontend
2. Scales the input using the saved StandardScaler
3. Predicts the user's personality cluster
4. Returns the cluster label and the full personality description in JSON format

During the midterm presentation, the frontend-backend connection was partially functional, but the final prediction output did not display properly. After refining the API logic, CORS settings, and input validation, the final implementation successfully returns the user’s personality type on the website.

텍스트, 소프트웨어, 스크린샷, 멀티미디어 소프트웨어이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

Figure 20: Swagger UI “Try Out” Example Output

Figure 20 shows the actual output returned by the Random Forest classifier. The response includes the predicted personality cluster, human-readable personality description, and probability distribution across all five clusters.

This completed the functional pipeline from questionnaire -> ML model -> API -> real-time website result.

# Lessons Learned and Future Work

## Lessons Learned by Technical Development

Throughout this project, we significantly expanded our understanding of data acquisition, modeling, and system integration. We learned how to search for appropriate datasets, use web scraping, and evaluate dataset quality before modeling. Initially, we attempted to merge job and course datasets from Kaggle to build a combined recommendation system. However, during preprocessing, we encountered substantial data loss, which forced us to rethink our data strategy. This experience taught us that merging datasets is not always necessary and that modeling can often be done more effectively by treating datasets independently.

Working with large and real-time datasets also highlighted the importance of managing dataset size and sampling. Training machine learning models on large files resulted in long processing times, so we learned how to extract balanced and representative samples for efficient modeling.

We also learned how to save machine learning models using pickle, load them through FastAPI, and deploy them as real-time components of a web application. Before this project, most of our data analytics coursework focused on modeling inside Jupyter notebooks. This project taught us how to transform models into functional, usable components integrated into a real website.

## Lessons Learned from System Integration and Fast API

This project was our first experience implementing both the frontend and backend components of a data-driven application, and FastAPI served as our primary tool for connecting the models to the frontend. Through this implementation process, we learned how to structure API endpoints effectively so that the system could accept and process user input in a consistent manner. We also learned how to validate and handle incoming JSON data to ensure that the model receives the correct format and values, as well as how to configure CORS policies to enable seamless communication between the frontend and backend without triggering access errors. In addition, we became familiar with testing API endpoints through Swagger’s built-in interface, which allowed us to quickly experiment with input values and check whether the prediction pipeline was functioning correctly. Finally, we gained experience returning prediction results in clear and structured formats that could be easily interpreted by the frontend.

These combined skills helped us understand the real-world workflow of deploying machine learning models rather than simply training them in isolation. Concepts that previously felt disconnected, such as frontend frameworks, backend logic, and machine learning modeling, came together cohesively through FastAPI, giving us a much deeper and more practical understanding of how data science and software engineering interact in real applications.

## Personal Growth, Teamwork, and Career-Relevant Learning

This project also strengthened our teamwork and communication skills. We collaborated across different roles, frontend, backend, modeling, and helped each other fill knowledge gaps. This experience will be valuable in future careers where cross-functional collaboration is essential.

We also gained confidence in applying what we learned in Douglas College’s CSIS courses. Concepts that once felt abstract became meaningful when we implemented them into a real system. This project broadened our understanding of how data analysis, software engineering, and machine learning intersect in modern applications.

## Future Work

There are several enhancements we hope to implement in the future to expand the capabilities of the system. One of our primary goals is to develop a personality-to-job compatibility scoring mechanism that can evaluate how well a user’s traits align with specific roles. We also intend to expand the job and course datasets so that the recommendation results become more comprehensive and reliable. Another area for improvement is feature selection, where reducing the number of questions while maintaining predictive accuracy would make the assessment more efficient and user-friendly. Future development could also involve deploying the system on cloud platforms such as AWS or Azure to improve scalability and reliability, and enhancing the UI/UX for clearer and more engaging result visualization. With additional data, refinement, and technical development, this system has the potential to evolve into a fully operational platform that meaningfully supports students and new graduates in making informed and confident career decisions.

# Concluding Remarks

Our project began with a shared desire to help students and new graduates who struggle with finding the right career path. As we approached graduation ourselves, we felt the same uncertainty and pressure, which inspired us to create a system that uses data to support people during this important stage of life.

Although the current version is limited by dataset size and scope, it demonstrates that data can reveal meaningful insights about personal strengths and areas for improvement. We believe that everyone has a career path that suits their personality and abilities, even if it is not immediately obvious. It is our hope that future versions of this project will serve as a valuable resource for job seekers and help them save time, gain confidence, and discover new opportunities.

This project was also a meaningful academic experience. We are grateful for the opportunity to apply the knowledge gained from our courses into a real project. We would like to express our sincere thanks to Professor Bambang Sarif for his guidance, feedback, and encouragement throughout the project, as well as to the instructors at Douglas College who taught the CSIS courses that supported our learning.

Hyunhee:  
I especially thank our team leader, Leshan, for guiding the team, supporting me, and helping turn our ideas into a functional system. This project was a collaborative accomplishment, and we appreciate everyone who contributed to its success.

Leshan:  
I would like to express my sincere appreciation for Hyunhee’s dedication throughout this project. Her commitment, hard work, and willingness to go the extra mile made a significant impact on our progress. Thank you for consistently stepping up and contributing with such positivity and determination.

# References

|  |  |
| --- | --- |
| [1] | N Deepak, M Gowtham, J Mithilesh and A Naresh Kumar, "Personalized Career Recommendation System Based on Customer Segmentation," Journal of Emerging Technologies and Innovative Research, 2024. |
| [2] | F. Kainyu, M. Mwadulo and S. Munialo , "A Hybrid Recommender Model for Career Pathway Selection in Competency-based Education," International Journal of Computer Applications, 2025. |
| [3] | Y. Gupt, "CareerX: AI-Powered Career Path Recommender System for College Students," ResearchGate, 2025. |
| [4] | H. Kumar, U. P. Pandey and P. Kumar, "AI-Based Career Path Recommendation System," [Online]. Available: https://amity.edu/UserFiles/aijem/293Harsh1.0%20(AJCS).pdf. |
| [5] | M. QAMHIEH, H. SAMMANEH and M. N. DEMAIDI, "PCRS: Personalized Career-Path Recommender," IEEEAccess, 2020. |

# Appendix

## Appendix A: Installation Guide

The Pathfinder application consists of:

* Backend – FastAPI (Python), with machine learning models, schedulers, and Selenium-based web scrapers.
* Frontend – React (JavaScript), built using Vite.

This installation guide explains how to set up and run both components locally.

### Backend Installation

**Prerequisites**

Before installing the backend, ensure the following software is installed:

* Python 3.9+
* Pip Latest version
* Google Chrome Latest
* SQLite (optional) Latest

**Create and Activate a Virtual Environment**

Navigate to backend folder execute following commands to create and activate virtual environment.

Windows:  
python -m venv venv  
venv\Scripts\activate

Mac/Linux:  
python3 -m venv venv  
source venv/bin/activate

**Install Backend Dependencies**

Execute following command to install dependencies

pip install -r requirements.txt

**Running the FastAPI Server**

Execute following command to start the backend server

uvicorn main:app –reload

### Frontend Installation

**Prerequisites**

Before installing the frontend, ensure the following software is installed:

* Node.js 18+
* npm 8+

**Install Frontend**

Navigate to the client folder and execute following command

npm install

**Run Development Server**

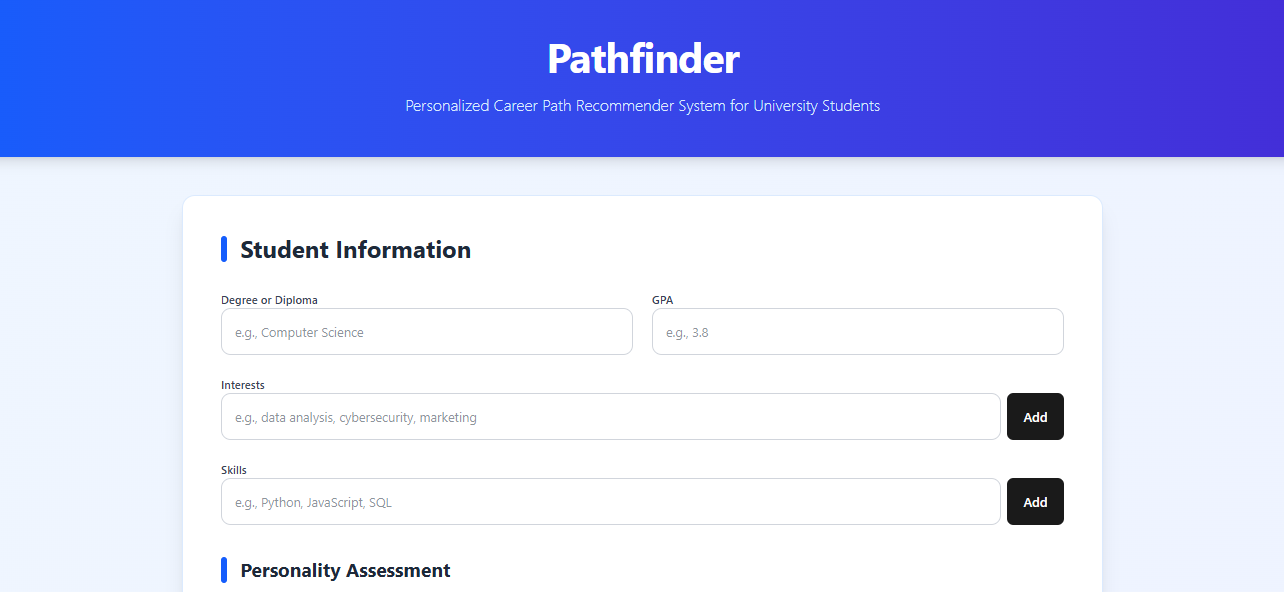
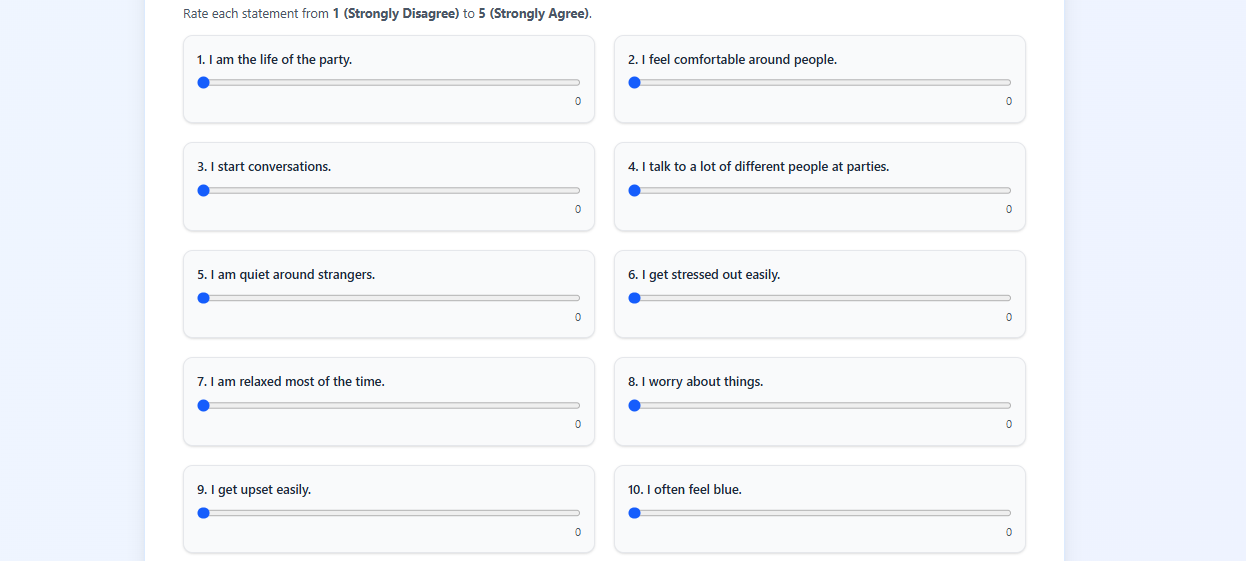
Execute following command to start the frontend application

npm run dev

## Appendix B: User Guide

Users need to fill all the fields in the user interface including Student information and Personality assessment as shown in figure 21.

Figure 21: pathfinder frondend



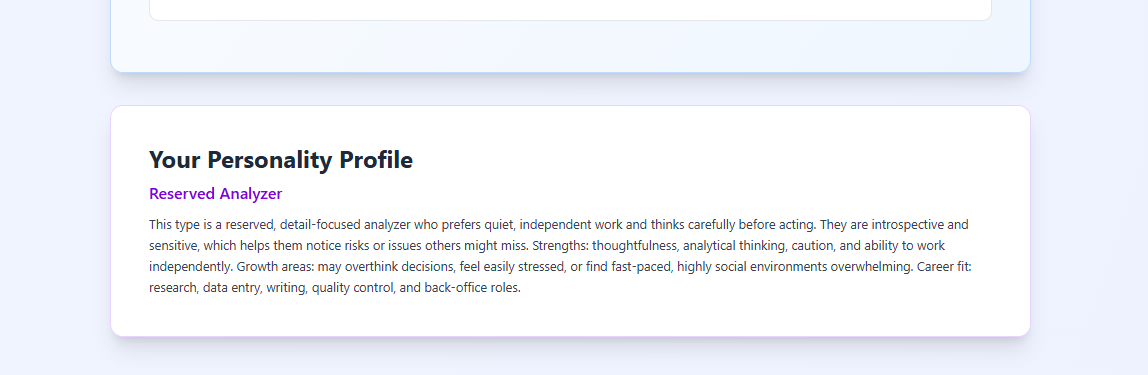
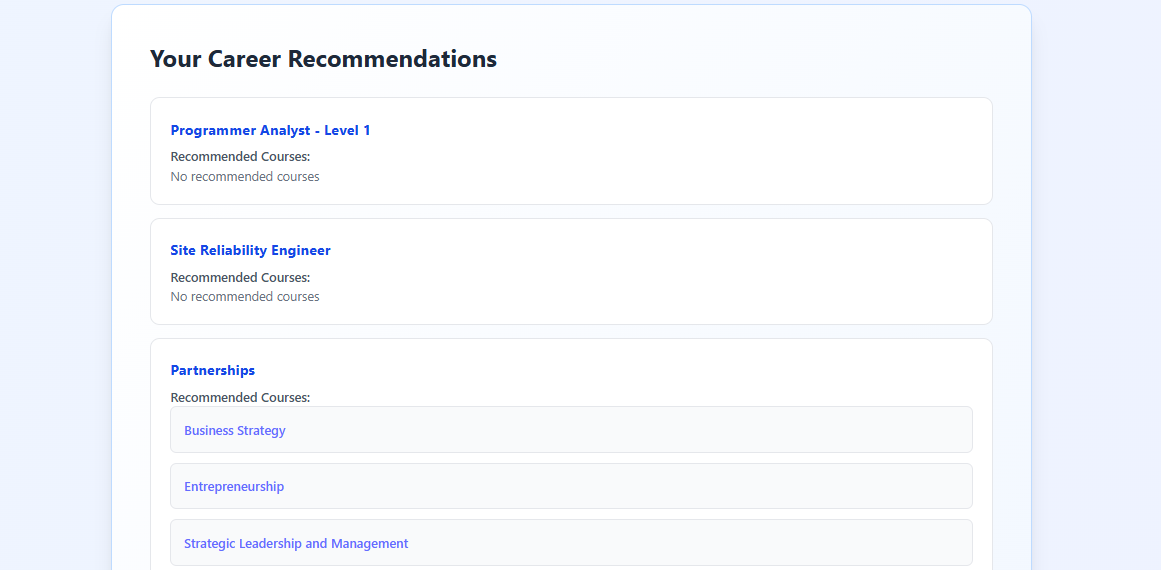
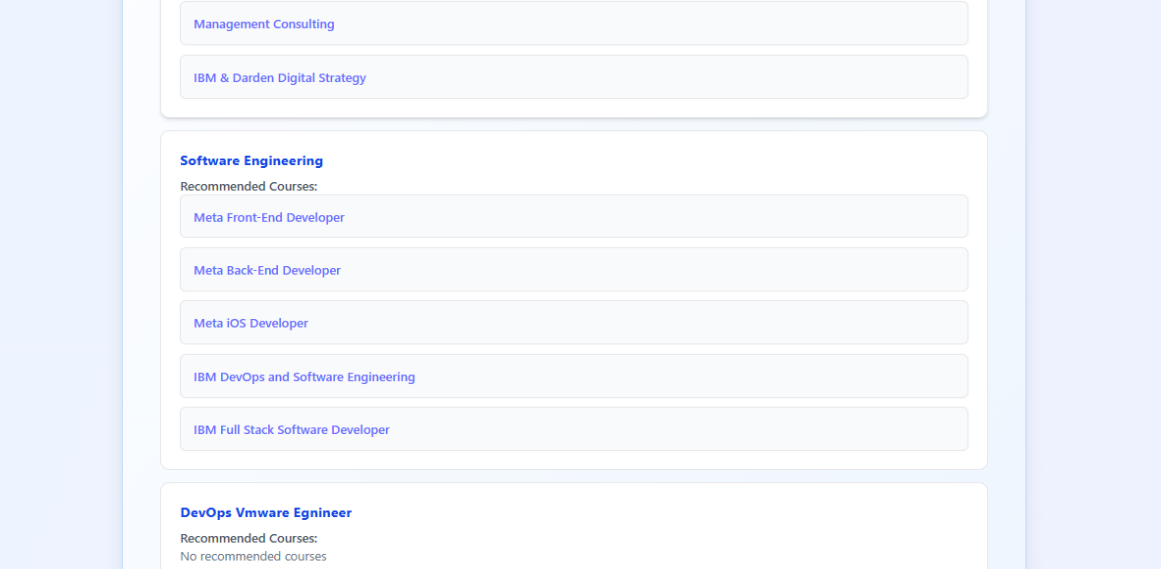
Once fill all the fields user need to click “Get Recommendations” button shown in the figure 22 to get the results as shown in figure 23. User is able the access the recommended courses by clicking on course name

A screenshot of a computer

AI-generated content may be incorrect.

Figure 22:pathfinder frontend button

Figure 23: pathfinder frontend output



## Appendix C: Dataset and API Used

Following datasets were used to initiate the implementation of recommendation models

Coursera course dataset - <https://www.kaggle.com/datasets/azraimohamad/coursera-course-data/data>

LinkedIn job dataset - <https://www.kaggle.com/datasets/asaniczka/1-3m-linkedin-jobs-and-skills-2024/data>

Personality trait dataset - <https://www.kaggle.com/datasets/tunguz/big-five-personality-test/data>

Pathfinder application doesn’t contain any third-party API

## Appendix D: Hardware, Software, Architecture

### Hardware Requirements

The Pathfinder application is designed to run efficiently on standard commodity hardware, as most computation occurs on the server side.

* Processor: Dual-core CPU (Intel i5 or equivalent)
* Memory: Minimum 8 GB RAM to support model loading, web scraping, and local development tools
* Storage: At least 10 GB free space for Python environments, datasets, and generated model artifacts (PKL files)
* Network: Stable broadband connection for web scraping, model updates, and frontend-backend communication

### Software Requirements

The Pathfinder system is built using a modern, full-stack software environment:

**Backend Software (FastAPI)**

* Python 3.x
* FastAPI for REST API development
* Uvicorn ASGI server
* Pandas / NumPy for data manipulation
* Scikit-Learn, Joblib for machine-learning models and prediction
* Selenium + WebDriver Manager for automated web scraping
* Schedule for periodic job model updates
* SQLite for database storage
* Additional utility libraries for preprocessing, TF-IDF vectorization, and recommendation logic

**Frontend Software (React)**

* React with Vite build system
* JavaScript for UI logic
* Axios for calling the backend
* Tailwind for interface styling

### Architecture Overview

The Frontend Layer of the Pathfinder application is built using React.js and serves as the main interaction point for users. It collects key profile information such as education, GPA, skills, and interests through a responsive interface. Once predictions are generated, the frontend displays the results, including job categories, recommended courses and personality trait. All communication between the user interface and the backend takes place through REST API calls, ensuring a smooth and structured data flow.

The Backend Layer, developed using FastAPI, is responsible for handling incoming requests from the frontend and managing the core business logic of the system. When a user submits their profile data, the backend preprocesses the input and converts it into model-ready numerical vectors. It loads the trained machine-learning models, TF-IDF vectorizers, and label encoders to generate predictions. In addition, this layer contains automated Selenium-based scrapers for Indeed and Coursera, and it performs scheduled tasks every 30 days to update and regenerate PKL files that keep the job model fresh and relevant.

The Database Layer provides persistent storage for all essential datasets used by the system. It stores scraped job postings and course listings that support both the prediction and recommendation processes. This organized storage structure enables reliable data retrieval and supports ongoing analysis and system improvements.

The Machine Learning Layer contains all the trained predictive and preprocessing components used by Pathfinder. This includes the main classification model (saved\_model.pkl), skill vocabulary, TF-IDF transformers, label encoders, cluster models, and data scalers. These components work together to generate job predictions and compute confidence scores based on the user’s input. By combining these models with the backend’s processing logic, the system is able to deliver accurate job recommendations along with insights into skill gaps and personalized learning resources.

## Appendix E: Code Explanation

### Pathfinder database

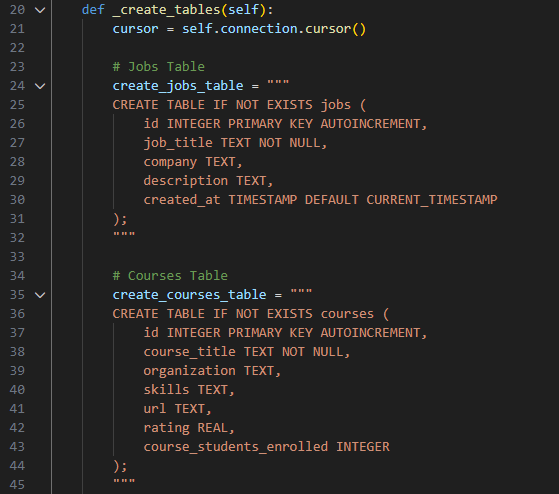
As shown in figure 24 to 29 the PathfinderDatabase class is designed to manage all database operations for the application by using SQLite as the underlying storage engine. When initialized, the class sets the database path and provides a connect() method that establishes a connection to the SQLite database file while also creating the necessary tables through the private \_create\_tables() method. Two tables—jobs and courses—are defined to store job postings and online course information, respectively. The class includes dedicated methods such as save\_jobs() and save\_courses() that accept lists of job or course dictionaries and insert each entry into the corresponding table using parameterized SQL queries to ensure safe and consistent data insertion. For machine learning workflows, the class provides fetch\_jobs() and fetch\_courses() methods, which load stored records into Pandas DataFrames, allowing the data to be used directly for preprocessing, training models, or generating recommendations. Overall, this class abstracts all low-level database operations and provides a clean, modular, and reusable interface for storing, retrieving, and backing up data in the Pathfinder system.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 24: database.py - class

Figure 25: database.py – create\_table function



A screen shot of a computer program

AI-generated content may be incorrect.

Figure 26: database.py – save\_jobs function

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 27: database.py – save\_courses function

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 28: database.py – fetch\_jobs function

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 29: database.py – fetch\_courses function

### Scheduler

Figure 30 to 33 shows implementation of the automated continuous data collection and model maintenance through scheduled scraping and periodic retraining. It begins by defining a list of common job titles across various industries and cycles through them using Python’s cycle() function to ensure that every scraping run targets a different job title. The run\_hourly\_scraper() function executes once every hour and manages the full scraping workflow: it first connects to the SQLite database through the PathfinderDatabase class and initializes an IndeedScraper instance to collect job postings for the next job title in the cycle. Once scraped, the jobs are stored in the database, after which the scraper is shut down to free resources. The function then initializes the CourseraScraper to gather relevant courses for the same job title and similarly stores them in the database. In parallel, the system also supports automated machine learning model updates through the run\_monthly\_training() function, which retrains the job prediction model every 30 days. It updates the core assets required for inference, including the trained model, TF-IDF vectorizer, and label encoder by loading newly saved .pkl files. The start\_scheduler() function configures the automation using the schedule library, running the hourly scraping job and the monthly training job at specified intervals, while also triggering both once during system startup. Finally, FastAPI’s startup\_event launches the scheduler in a background thread to ensure continuous execution without blocking the API. Overall, this code provides a fully automated pipeline that continuously updates job and course datasets while periodically improving the machine learning model to ensure fresh and accurate predictions.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 30: main.py - job list

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 31: main.py - hourly job and course scraper

A computer screen with many colorful text

AI-generated content may be incorrect.

Figure 32: main.py - monthly model builder

A black screen with white text

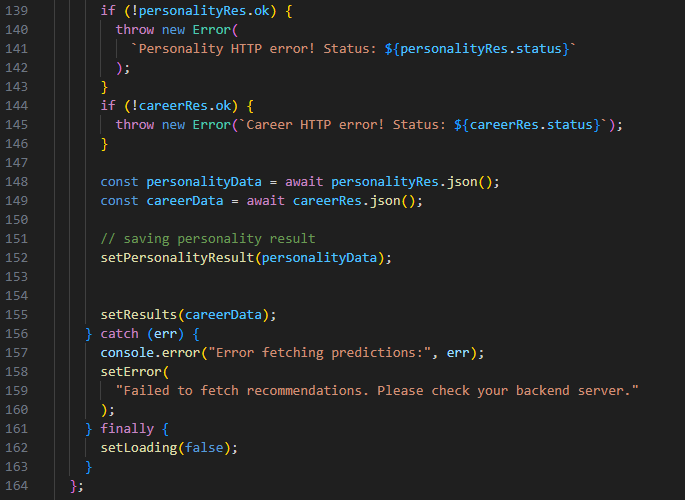
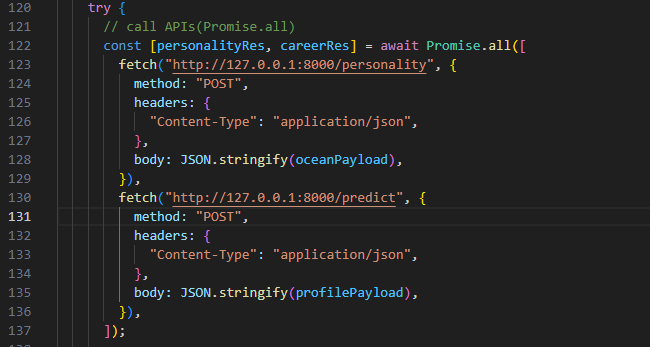
AI-generated content may be incorrect.

Figure 33: main.py - schedule initiator

### API request Response

This section of the frontend handles parallel API requests for personality analysis and career prediction using Promise.all to improve efficiency and reduce waiting time for the user. Inside a try block, the function sends two asynchronous POST requests simultaneously: one to the /personality endpoint with the oceanPayload containing personality test inputs, and another to the /predict endpoint with the profilePayload used for job prediction. Once both requests complete, their responses are checked to ensure they returned successful HTTP status codes; if either response is not OK, a descriptive error is thrown. The JSON data from both endpoints is then extracted and stored in the component state, with setPersonalityResult() saving the personality analysis and setResults() storing career recommendations. If any part of the operation fails—such as a network issue or backend error—the catch block logs the error and displays a user-friendly message through the setError() state. Finally, the finally block ensures that the loading indicator is turned off by calling setLoading(false) regardless of success or failure. This structure provides a robust, efficient, and user-responsive way to fetch and display multiple prediction results concurrently.

Figure 34: pathfinder.jsx API call



### Loading Machine Learning Models (personality\_model.py)

The personality\_model.py file is responsible for loading the trained RandomForest classifier and the StandardScaler. As shown in figure 35 when the application starts, the file attempts to load the pickled model (rf\_cluster\_k5.pkl) and scaler (ocean\_scaler.pkl). It also defines cluster names and descriptions that map the numeric cluster output to human-readable categories. This file provides a helper function that formats the prediction result before returning it to the API.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

Figure 35: personality\_model.py loading model files

### Personality Prediction Endpoint (main.py)

As shown in figure 36 the /personality endpoint receives five OCEAN scores from the frontend as JSON. Once the data is validated through Pydantic (OceanInput), the scores are scaled using the loaded StandardScaler, and the RandomForest model predicts the personality cluster. The endpoint returns the cluster ID, cluster label, description, and probability distribution of all five clusters.

This endpoint is used by the frontend personality UI to show users their personality type.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

Figure 36: personality API endpoint