Design Project

Group 13

Learning Path Trajectories: Mapping Current Skills to Future Educational Goals

Ayolt ten Have (s2793199)
Ruslan Kasač(s2980843)
Simay Odabaşı (s2822547)
Tara van Haarst (s3190016)
Thijs Frauenfelder (s2739127)

Abstract

University students often struggle with making choices when attempting to navigate through course catalogs to find courses that align with their academic choices and their specific career aspirations. Traditional university databases, such as the Osiris system used in the University of Twente, provide static syllabus information and fail to account for a student's unique starting knowledge or their long term career goals. This report presents the design and implementation process of the "Learning Path Trajectory" platform which is an AI driven web application developed to eliminate the guesswork and create a personalized curriculum planning for their user.

To achieve this, we build up a custom data extraction pipeline to securely retrieve raw course data and develop a more sophisticated pipeline that uses Ministral 3 (14B) Large Language Model (LLM). There are two main pipelines in the system. First pipeline works as a semantic parser and translates unstructured, text heavy syllabus from courses to a structured database as required and gained skills. The second pipeline, on the other hand, evaluates user-inputted skills and career goals and generates current and future user profiles. Finally the system calculates and visualizes a step by step roadmap within a responsive React frontend, explicitly visualizing the courses to follow and highlighting the skills gained and could not gained. Ultimately, this prototype aims to reduce the cognitive load for users by transforming educational planning from a manual struggle to a more actionable, goal oriented trajectory.

Keywords: *Large Language Models (LLMs), Personalized Learning, Semantic Parsing, Curriculum Planning, Skill Extraction.*

Contents

Chapter 1

Introduction

In the landscape of higher education, students often face challenges in navigating complex course offerings to reach their specific professional goals. Traditional course catalogs such as the Osiris system most of the time provide static information but do not account for user's unique starting knowledge. This disconnect can often lead to a choice paralysis, where users struggle to connect the dots between their current skills, the prerequisites of various courses, and the requirements for their dream job. Recognizing this gap, this project focuses on how to use artificial intelligence to transform the static academic data into more dynamic, personalized career roadmaps.

This report presents the design and the implementation process of the "Learning Path Trajectory" platform. Utilizing Large Language Models (LLMs) this system bridges the educational gap between a user's current profile and their desired career goals. Instead of relying on manual searches, the system semantically analyzes unstructured course data to understand specific learning outcomes and prerequisites of the courses. After that, it automatically calculates and presents the most suitable course sequence the user can take to fulfill their specific knowledge gaps.

To address this challenge, two primary goals were set:

- Implement an LLM-driven backend to perform personalized gap analysis by evaluating a user's achieved skills against their target career to extract missing knowledge in between.
- Design and develop an accessible, interactive web interface that visualises the structured learning path trajectory, allowing users to easily track the skills they will achieve by following the courses.

These goals support the development of a digital system that helps users to shift the burden of curriculum planning. By automating the extraction and semantic matching of academic data, the system goal is to eliminate all the guesswork, empowering users to make informed decisions about their education and future careers.

The remainder of this report is structured as follows. The report first focuses on discussing methodology and planning by outlining the project's development approach and team logistics. Chapter 3 details the user, system and non-functional requirements that guided the platform's creation. Chapter 4 presents the System Design by illustrating the architecture, user interactions and data flow through UML diagrams. Chapter 5 provides a risk analysis of the system by identifying potential technical and logistical challenges. Chapter 6 provides the Development process of this system by providing an in-depth implementation of the backend,

database and the evaluation of the frontend from low fidelity to the implementation. Chapter 7 details the unit, integration and the system testing used to validate the platform. Finally, the last chapters cover the Evaluation, Discussion and Conclusion, reflecting on the project's overall outcomes, limitations and potential for future development.

AI Statement

During this project AI is used for research and helping with grammar mistakes. We reviewed the suggestions and information and take full responsibility for the final work. AI is also used for debugging purposes during the implementation phase.

Chapter 2

Methodology and Planning

This section outlines the organization framework used in the development of this project. It details the methods that are employed, the planning process of the project, strategic division of tasks among team members and the logistical protocols used for communication.

2.1 Client

The primary client and the supervisor for this project is Faizan Ahmed, representing the University of Twente. The project is further informed by research conducted at the University of Twente regarding LLM benchmarks.

We gathered the requirements for the system as is written down in chapter 3 and verified them with our client to determine the scope of this project. We also had meetings every week discussing the process and gathering feedback from the client.

2.2 Planning

The project is divided into 10 distinct weeks, moving from planning to deployment and final reflection. The project was divided into the following distinct development phases:

1. Week 1-2: Focused on meeting with clients and gathering project requirements. These criteria were then translated into UML diagrams, establishing the architecture for the frontend and backend.
2. Week 3: Worked on the backend architecture. This included engineering the custom data extraction from the Osiris database and evaluating various Large Language Models.
3. Week 4-5: Centered on developing the initial LLM logic for keyword extraction. On the other hand, the team also focused on designing the user interface and initial prototypes.
4. Week 6-7: Focused heavily on the backend architecture. The team designed the two core AI pipelines: a pipeline for the creation of the database and a pipeline for the profile creation and trajectory path. On the front end, the implementation was finalized.
5. Week 8-9: Focused on bridging the React front end with the LLM driven backend. Unit and integration testing were conducted to ensure accurate results.
6. Week 10: Final testing and debugging were conducted. The team presented the final presentation and report submission.

By establishing a highly structured workflow early on, the team ensured the project remained adaptable to technical hurdles while consistently meeting developmental milestones.

2.3 Task Division

The project is divided into several technical components. As a group, we divided into two person groups and worked on each topic.

- Backend Team: This team was responsible for the LLM implementation and all the backend logic.
- Data Engineer Team: This team focused on the automated scraping from OSIRIS and keyword extraction.
- Front End Team: This team worked on developing the design and the implementation of the web site.

As a group, we were all responsible for the final design report and the reports in the reflection part.

	Backend	Frontend	Data Extraction	Integration	Report
Ayolt	X			X	X
Ruslan	X		X		X
Simay		X	X	X	X
Tara	X				X
Thijs					

2.4 Logistic

2.4.1 Team Communication

To ensure seamless collaboration and efficient project management, the team established different communication strategies. Throughout the project, the main communication channel used was Whatsapp. We created a group in the first week to talk about our plans, team meetings and team progress. More intensive collaborative sessions, including online meetings and link distribution were conducted through Discord.

To maintain alignment and track development milestones, the team held internal meetings at least once a week. Furthermore, every week, we also did a supervised meeting with our

supervisor. We communicate with our supervisor by emailing him or arranging meetings to discuss our progress.

2.4.2 File Sharing and Version Control

All non code documents such as presentations, reports and diagrams were collected in a shared Google Drive workspace. We also shared this folder with our supervisor to get feedback quicker.

For software development, we created a Git Lab environment to share and work on it together. We separated each component into its own branch to avoid any merge conflict.

Chapter 3

Requirements

To ensure the Learning Path Trajectory system effectively addresses the challenges users face when navigating university course catalogs, a comprehensive set of requirements was established before the development process. This chapter focuses on the requirements derived from the project description, and meetings with the project supervisors. The requirements are separated into two main topics such as functional and non-functional requirements. The functional requirements are also separated into two sub requirements such as user requirements and system requirements.

For all the requirements, the MoSCoW method is used, which categorises the requirements into four groups depending on their priority: Must have, Should have, Could have and Won't have.

Must have
These requirements are necessary and must be completed for the successful completion of the project. Without these requirements, the system will not function as intended.

Should have
These requirements are important, but not necessary for the successful completion of the project. If they are not implemented, the overall quality of the project may be reduced, but the system will still function as intended.

Could have
These requirements are nice to have, but not essential. If they are not implemented, the impact on the project is minimal.

Won't have
These requirements are not considered a priority and will therefore not be implemented. They are included to clearly show the scope of the project.

3.1 User requirements

Must have:

- The user must be able to describe their current knowledge profile (starting point of the student) and their desired job (goal of the student) in natural language.
- The user must be able to define a desired profile / job.

- The user must receive a clear explanation regarding the AI choices of the courses that forms a logical educational journey. The system should indicate why it was selected.
- The output must include the official Course IDs or links to the OSIRIS page to allow the student to actually enrol the course.
- The recommended courses must align with their specific domain goals.

Should have:

- The user should be able to ask multiple questions.
- The student should access the tool via a simple web based interface without needing technical AI knowledge.
- The user should be able to ask multiple questions.
- The system should have an user-friendly interface.
- If multiple paths exist, the UI should allow the user to compare them side by side. (e.g. Shortest Path vs Deep Dive)

Could have:

- The user can login and get back to an earlier given trajectory or change an earlier extracted profile.

3.2 System requirements

Must have:

- The pipeline must identify specific keywords for entrance requirements and for desired completion outcomes.
- All extracted course data must be stored in a structured format to provide efficient searching and future expansion.
- All extracted course requirements keywords must be stored in a structured format to provide efficient searching and future expansion.
- The backend must output an ordered list of courses that logically follows from a user's starting profile to their destination goal.
- The system must be tested using specific scenarios.
- The system should be continuously tested during the implementation process to assure a working project in the end.
- The system must be hosted in an environment that guarantees no data of users leaks, satisfying the requirement to handle non-pıublic or sensitive information.
- The system must use an LLM model to achieve a good contextual matching.
- The system must extract from the users input the current knowledge profile and desired knowledge profile
- The system must give multiple trajectory paths in case of multiple path options.
- The model should be lightweight enough to run as a web service while maintaining high accuracy in trajectory development.

Should have:

- The model should be lightweight enough to run as a web service while maintaining high accuracy in trajectory development.
- The system should be designed as expanded so the other departments or institutions can also use it in the future.

Could have:

- The system can remember the user's trajectory or remember the current and desired profile in case the user is logged in.
- The system should implement a source referencing mechanism where every course in a trajectory is linked to a section of the database to prevent fake prerequisites.
- The system can give multiple trajectory paths in case of multiple path options and let the user choose their preferred path.
- The system can give the trajectory with the shortest amount of courses in case of multiple path options.
- The system can collect OSIRIS data and extract prerequisite knowledge and expected knowledge levels for each course.

Won’t have:

- The system will not have a real time connection with OSIRIS. It will rely on the scraped JSON data store.
- The system will not store sensitive student information such as academic records or personal identifiers permanently on the server.
- The system will not be integrated with osiris.
- The system will not over explain the reasoning of why they chose that specific course. It should be short and clear.

3.3 Non-functional requirements

Must have:

- The user must not wait longer than 1 minute before the program outputs a trajectory.
- Users should be able to navigate the menu in the first 2 minutes of using the website for the first time.
- The system must achieve a 100% success rate in JSON schema validation for every output, ensuring there is no malformed data in the user interface.
- The system must provide a justification for the output trajectory created.
- The system should be easily updated and maintained.
- The backend must maintain stable accuracy.

- The system must record all AI reasoning steps.
- The system should be easily used without additional help.
- The system should iteratively learn from earlier reasoning and feedback.
- Every recommended course must include a source reference so students can check the AI's logic in the official university data.
- The system must be clear in what profiles are determined so the user can give feedback in case of mistakes.

Should have:

- The website should display the details of the usage of AI and LLM models.

Chapter 4

System Design

This section focuses on the comprehensive high level design of the platform. Using Unified Modeling Language (UML) diagrams, this chapter visualizes the flow of the data, hierarchy of components and the complex interactions between the user, the frontend and the LLM driven backend.

4.1 System Architecture Diagram

This **System Architecture Diagram** shows the workflow of a system that generates **Recommended Trajectory Path** using **LLM**. The process begins with user input. This data is passed to a Search Engine, which determines the user's required skills. The user has a profile indicating their current skills and knowledge. The system processes the course dataset from Osiris, extracting the required skills and exit skills. The required skills, user profile data, and course data prompt the **LLM** to generate a **Recommended Trajectory Path**. Finally, this path is displayed to the user through the interface.

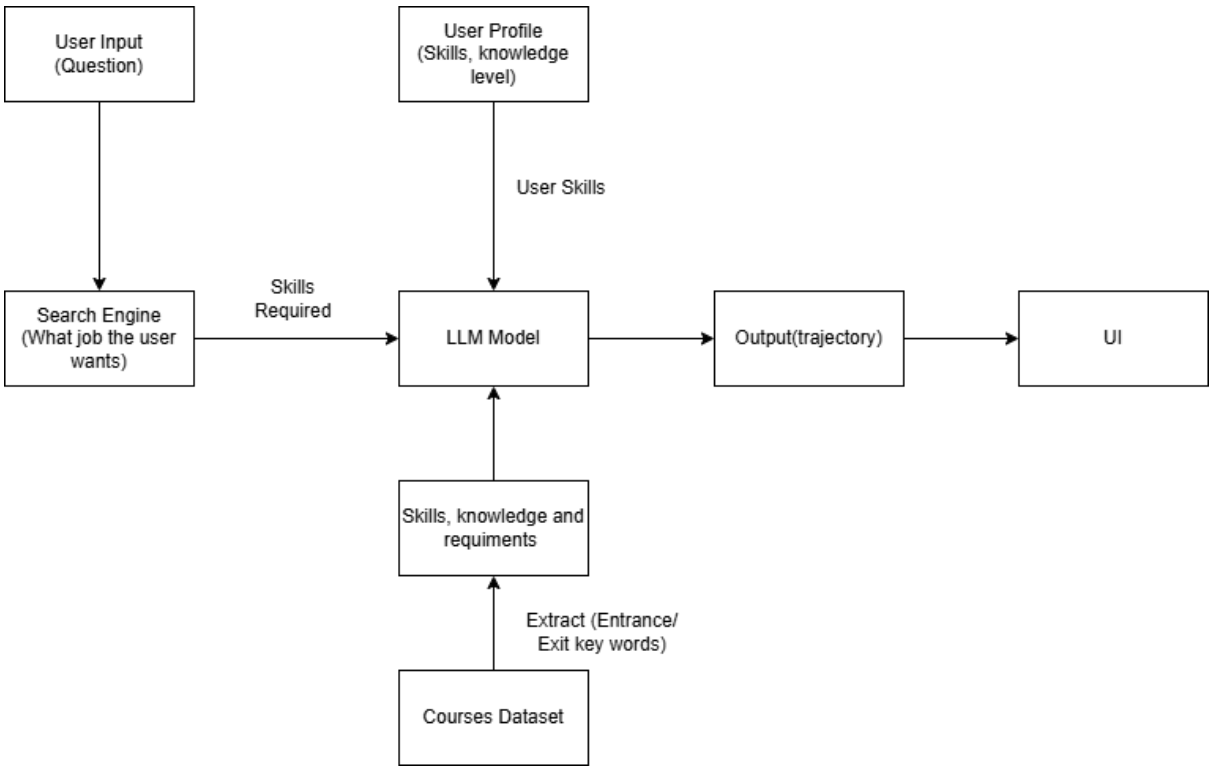

Figure 4.1: Architecture Diagram

4.2 Use-case Diagram

This **Use Case Diagram** shows a system designed to generate recommended **Learning Path Trajectory**. The diagram has one actor: a **User**. The **User** interacts with the system through several use cases: **Create Profile**, **View Personalized Trajectory**, **Login** and **Enter Profile and Goal**. After User input (Enter Profile and Goal), the user can **Generate Learning Trajectory** using **extracted processed Profile** and **extracted Processed Course**. During the generation process the system extracts the user's profile and obtains keywords from course data. Finally, the user can view the Personalized trajectory through the Web Interface.

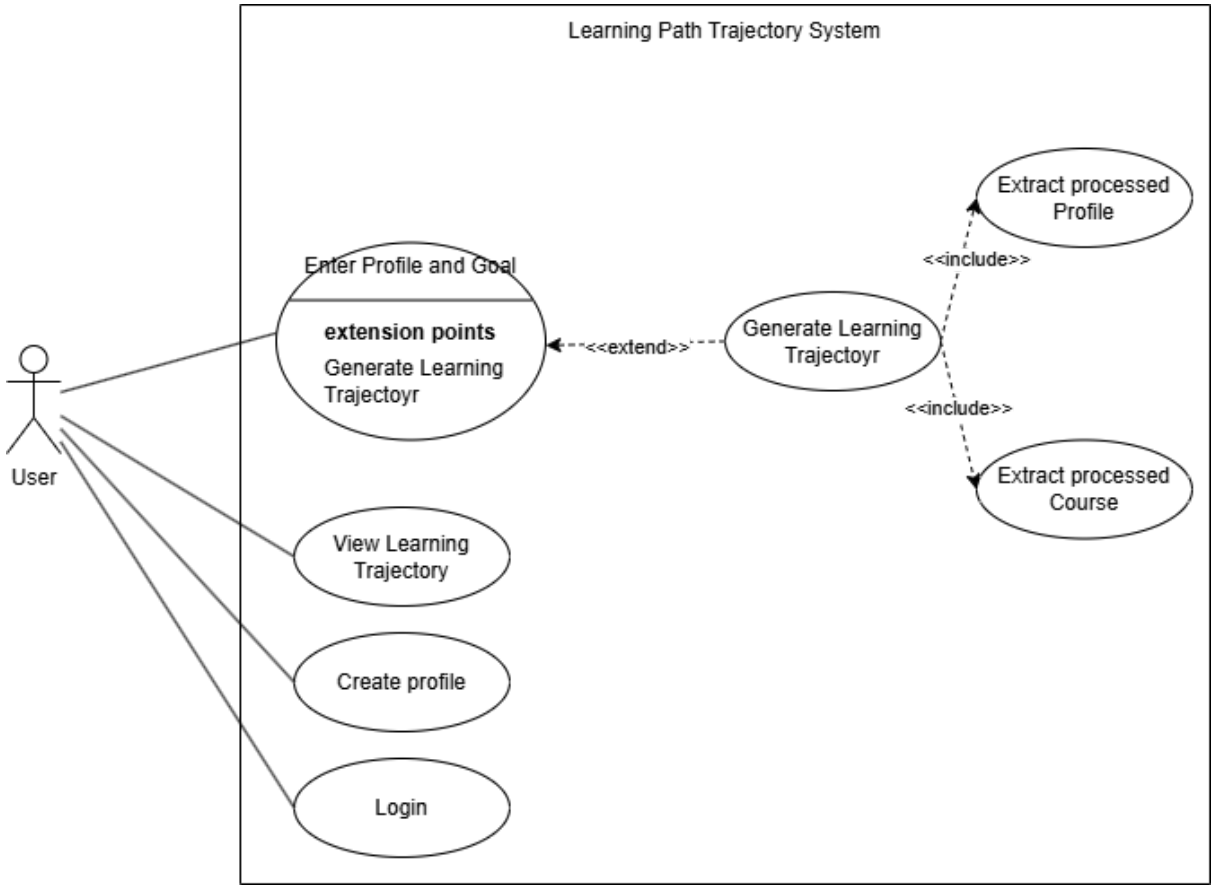

Figure 4.2: Use Case Diagram

4.3 Activity Diagrams

4.3.1 Activity Diagram 1

These **Activity Diagrams** show workflow for generating **Learning Path Trajectory** using **LLM**. The process starts when the system receives input from the user. Then the LLM extracts user profiles and retrieves course data keywords from storage, using this extracted data the LLM computes a **Learning Trajectory Path**. Then the **Generated Path Trajectory** is displayed in the **Web UI**.

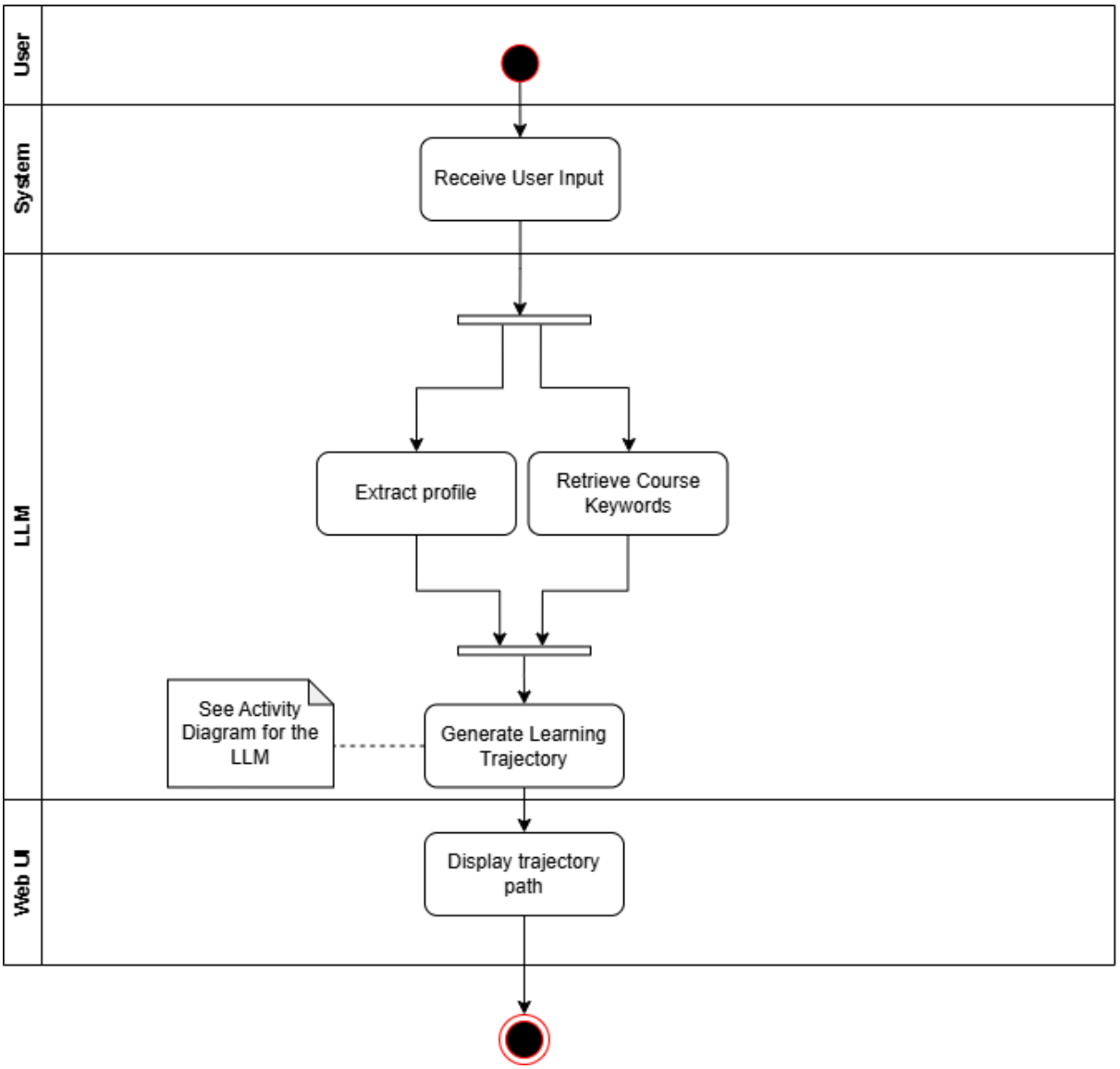

Figure 4.3.1: Activity Diagram 1

4.3.2 Activity Diagram 2

Inside the **LLM Trajectory Generator**, the LLM receives the profile information and keyword data, then checks whether there are keywords left to match. If there are, it matches the keywords with the course requirements, when requirements are met, the course is added to the trajectory. This matching process continues until no more relevant keywords remain. Finally the LLM returns a **Generated Path Trajectory** to the **Web UI.**

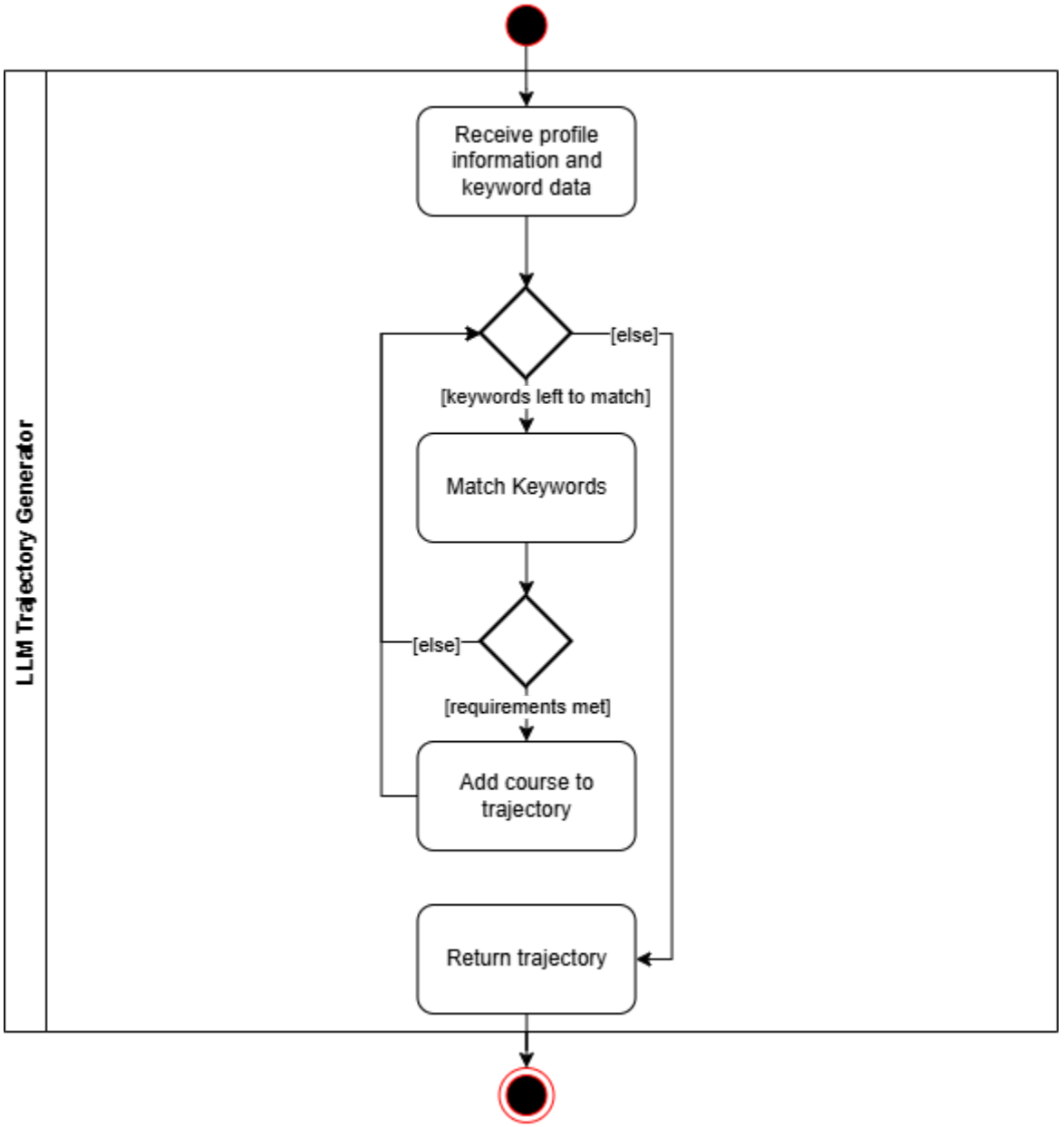

Figure 4.3.2: Activity Diagram 2

4.4 Sequence Diagram

This Sequence Diagram shows how the learning path is generated through interaction between **User**, **Web Interface**, **Local Database and LLM**. First, the user enters their profile and goal in the **Web Interface**. The **Web Interface** then sends the Profile to the **LLM**. The **LLM** begins generating a **Learning Path** by requesting course data from the **Local Storage**. Using user and course data, the **LLM** generates a **Learning Path** and sends it to the **Web Interface**. Finally, **the Web Interface** displays it to the user.

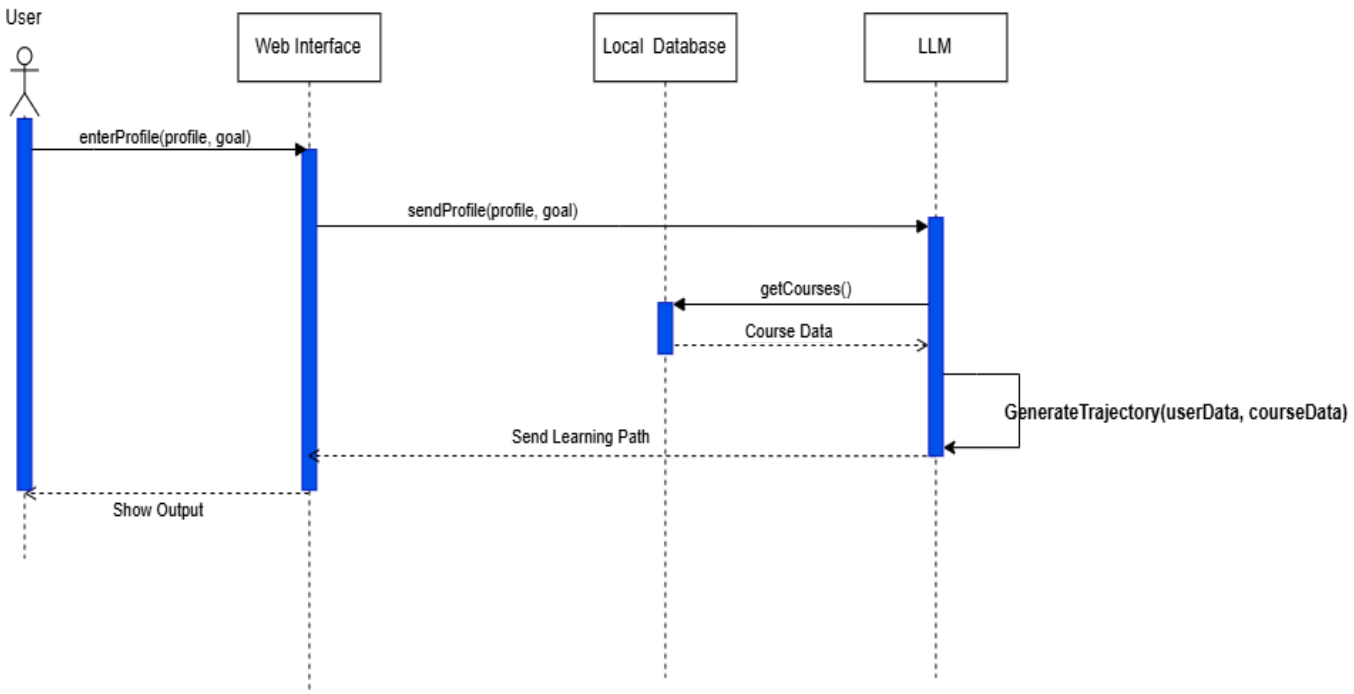

Figure 4.4: Sequence Diagram

4.5 Class Diagram

This class diagram represents a recommendation system that uses the LLM **Ministral 3 14B** to generate a **Learning path trajectory** for users. **The User** has login info such as a name and a password. **UserDescription** is a part of **User** and contains **UserDescriptionId**, **CurrentSkillsRaw** and **DesiredGoalRaw**. **The User** can have many **UserDescription**. **ProcessedCourse** and **ProcessedDescription** were generated by the **LLM** using **Course** and **UserDescription**, respectively. The **LLM** analyses both **ProcessedCourse** and **ProcessedDescription** to compute the **Trajectory**. The **Trajectory** contains TrajectoryId and Courses. The user can have multiple **Trajectories**.

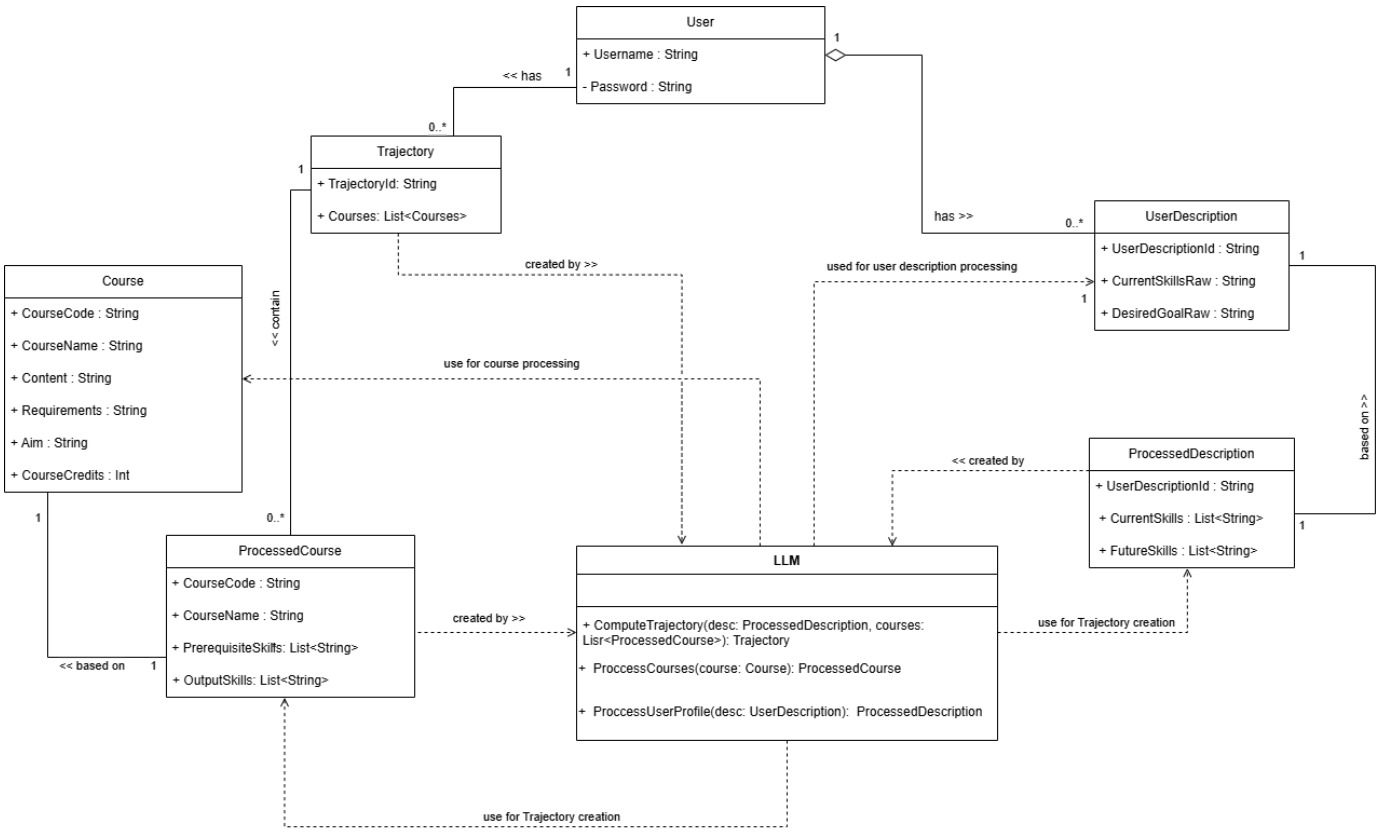

Figure 4.5: Class Diagram

Chapter 5

Risk Analysis

This section provides a detailed Risk Analysis for the platform by explaining and discussing the risks associated with implementing the Learning Path Trajectory system. Alongside these possible risks, this chapter also outlines the mitigation strategies implemented by the team to safeguard the development process and ensure the final product's reliability.

5.1 AI and Language Model Risks

- Irrelevant Recommendations
  - **Risk:** The system may recommend learning paths that are not aligned with the user's current skills or desired goals. Since the system itself relies heavily on the Ministral 3 model. If the LLM extracts inaccurate skill tokens from the courses, it can generate misaligned path trajectories.
  - **Mitigation:** Implementing a dual pipeline architecture that uses a cosine similarity normalizer. This maps all LLM generated skills to a verified, standardized vocabulary, reducing the risk of mismatched semantics and hallucinations.

- Irrelevant Keyword Extraction
  - **Risk:** Errors in extracting keywords from course data or user profiles can lead to incorrect matching and poor recommendations.
  - **Mitigation:** Testing and refining data extraction algorithms, as well as manual verification of sampling results.

- Lack of Explainability
  - **Risk:** The LLM system may generate vague or unconvincing reasoning, reducing users' trust in recommendations.
  - **Mitigation:** The system provides explanations for each recommended course, including how it relates to the user's current skills and desired goals.

- Ethical Concerns
  - **Risk:** The system may unintentionally direct users toward suboptimal or biased career paths, influencing real-life decisions.
  - **Mitigation:** The platform is positioned as a support tool and not a definitive academic advisor. The website is built to be absolutely transparent, explicitly displaying the skills achieved on every course so that the user can understand exactly why a course was recommended.

- Biased Output
  - **Risk:** The Ministrall 14B model selected may be biased, reducing the fairness and diversity of the recommendations.
  - **Mitigation:** Customize prompts to help you make balanced recommendations. Adding an option to users to choose their own LLM model to use.

- Over-dependent from LLM
  - **Risk:** The system heavily depends on the LLM for generating trajectories. If the model fails or produces low-quality in User Profile, Course Dataset or System output, the entire system performance is affected.
  - **Mitigation:** Design the System to handle temporary failures correctly and minimize inconvenience to users.

5.2 Data and Database Risk

- Poor Quality of Course Data
  - **Risk:** If the data collected from Osiris is outdated, incomplete or inconsistent, the system will produce incorrect trajectories.
  - **Mitigation:** Regularly updating and validating Osiris data. Engineering a robust data sanitization pipeline that clean and validate the data before it enters the system's database.

- Dynamic Changes in Course Structure
  - **Risk:** The Osiris course database may change over time (e.g., changes in prerequisites or content) which may invalidate previously created learning paths.
  - **Mitigation:** Regularly update and re-collect data from Osiris.

- Data Privacy Constraints
  - **Risk:** User profiles may contain sensitive information, which leads to risks associated with data storage and processing.
  - **Mitigation:** Avoid storing sensitive data whenever possible and minimize the collection of personal information. Creation of a secure, encrypted transmission between the frontend and the LLM backend.

5.3 System Performance and Integration Risks

- Integration Failures Between Components
  - **Risk:** Miscommunication and problems between the frontend, backend, and data processing pipeline.

- ➢ **Mitigation:** Implementation of error handling and logging. Rigorous unit and integration testing to validate data flow before merging the backend and front end together.

- ❖ System Performance Issue
  - ➢ **Risk:** Complex LLM can result in slow response times, especially under high user load.
  - ➢ **Mitigation:** Limit unnecessary computations. Implement clear loading states in the frontend to manage user expectations during the delays.

- ❖ User Interface Usability
  - ➢ **Risk:** A poorly designed interface can confuse users even if the backend is working correctly.
  - ➢ **Mitigation:** Prototypes and usability testing to prioritise a minimalist, accessible, natural language interface.

Chapter 6

Development

This chapter provides the details regarding the core technical process of the Learning Path Trajectory platform by documenting the transitions from architectural designs to a fully functional website. This chapter is divided into three chapters: the backend integration of the LLM model, the structuring of the relational database and the evaluation process of the frontend interface.

6.1 Backend

The backend was implemented with two main pipelines. A pipeline for the creation of the database information and a pipeline for the profile creation and trajectory path. Both pipelines are using the LLM model. The model used during the project and testing was Ministral 3 (14b).

The model Ministral 3 was chosen based on the following results and findings [1]. The model scored best out of the smaller size models which made it suitable for local deployment and testing. The model has one of the lowest hallucination rate scores which was an important criteria for the skills extraction. The model is also free to use and scored the best on the GovtBench performance for small models.

6.1.1 Collected data and dataset pipeline

The foundation of the Learning Path Trajectory platform relies on accurate and up to date data to give the user an accurate path. In this project, in the beginning we used The University of Twente catalog of courses from Osiris. However, Osiris was not designed for open data access and it was protected by enterprise grade security and dynamic authentication tokens. To overcome this, we engineered a custom, multi stage data pipeline to extract, clean and semantically structure course data. To gather this data we did the following:

- Authenticated Scraping:
  Firstly we bypassed the Osiris's enterprise firewalls by replicating exact browser payloads and injecting live session tokens into a Python script. This allowed us to safely extract the hidden internal IDs for courses. We specifically focused on Geoinformation courses and collected 110 courses. And for the testing part of the project we extracted more courses from OSIRIS by collecting EEMCS courses. Furthermore, to validate our AI pipeline and not just focus on one format of courses, we integrated a standardised, open source dataset of general online courses sourced from Kaggle [2].

Sanitization:
After collecting the courses, we passed the raw JSON responses through a custom Regular Expression filter to strip out all unstructured HTML and CSS tags to get a clean, machine readable text in the end.

To standardize the data retrieved from Osiris and Kaggle, each course is first stored in a unified raw input format before processing. This ensures consistency and prepares the data for LLM-based enrichment.
Each raw course entry contains the following attributes:

- course_code: The unique university identifier (e.g., 202500043)
- internal_id: The unique id assigned to each course
- title: The official course name used for display and user recognition.
- credits: Number used to calculate the total academic load of a suggested path in that university.
- content: Provides context for the LLM to understand topics and sometimes contains skills students will possess upon completion.
- aim: Defines the skills the students will possess upon completion.
- requirements: Defines the prerequisites needed to know before taking that course.

AI Semantic Parsing:
The next step in the process is the AI semantic parsing step. This step receives the cleaned course data and transforms it into structured data using the Ministral 3 LLM model. The parsing process is split into two stages: prerequisite extraction and skills extraction.

The first step is prerequisite extraction. In this step the system first checks whether a course contains explicit requirements using a predefined rule-based check. If no requirements are detected (e.g., phrases like “no prerequisites”), the LLM call is skipped and an empty prerequisite structure is returned. Otherwise, a structured prompt is sent to the LLM to extract prerequisite skills. These skills are grouped into lists, where all skills within a group must be satisfied (AND logic), while multiple groups represent alternative valid entry paths (OR logic). The extracted skills are concise and normalized so they can be matched later in the pipeline.

The second step is skills extraction. In this step, a second prompt is used that analyzes both the course content and aim fields. The LLM extracts concise, normalized skill keywords that represent the knowledge a student acquires after completing the course.

To ensure that the system is working correctly all the time, all LLM calls are executed with a retry mechanism and strict JSON validation. This ensures that failed or malformed responses are automatically re-requested until a valid JSON structure is returned or the retry limit is reached.

After extraction, a post processor refines the results. This step refines the LLM output by normalizing skill text, removing filler phrases, resolving duplicates and splitting “or” conditions into explicit alternatives. Additionally, for each course, its course title and course code is added as a skill gained.

The final structured output for each course is:

- course_code: The unique university identifier (e.g., 202500043)
- internal_id: The unique id given to each course
- title: The official course name used for display and user recognition.
- credits: Number used to calculate the total academic load of a suggested path in that university.
- Prerequisite_skills: A list of skills groups representing required entry knowledge.
- Skills_gained: A list of skills acquired after completing the course, including course name + code

6.1.2 Profile and trajectory pipeline

The profile and trajectory pipeline is developed in a structured order. As we described in the design phase the flow of the application starts with user input and ends with the trajectory output. For each part in the pipeline we made a separate class and implemented them as such. Each class is a main component step in the pipeline for the profile creation and trajectory creation.

The first class is the profile pipeline, which processes the user's input. This class sends this input along with the prompt for profile creation to the LLM and receives a json response back. This response contains two lists: the skills for the current profile and the skills of the future profile. Also here, the system includes retry logic and safe JSON parsing to handle incomplete LLM responses.

Since the LLM can create similar or almost similar skills the next step in the process is giving both these lists to the skills normalizer class. This class is implemented using a sentence transformer to encode and embed the skills. These embedded skills are then compared for cosine similarity to check if the skills are similar to a previous seen skill. In the case that it exceeds the threshold the skills are mapped together and otherwise the skills will be added as new to the list. This will be done for all skills in the current profile and for all skills in the future profile. With this similar skills will be mapped together and duplication errors will be prevented.

The next step in the process is the trajectory builder class. This receives the current profile, the future profile and the courses dataset as described in 6.1.1. The main function here is the loop for trajectory building which adds the courses to the trajectory one by one based on certain criteria. First the prerequisites have to be met otherwise the course can be ignored for that iteration. The prerequisites are evaluated per group, where all skills in a group must be satisfied, and satisfying any one group is sufficient. Skill-based prerequisites are checked

using cosine similarity between embeddings, while course-based prerequisites require an exact match with the user's completed courses.

After that there has to be checked if the course teaches a missing skill in the future profile to see if it is applicable for the trajectory. This is done by comparing the embeddings of the skills gained in the course with the embeddings of the missing skills. If the similarity exceeds a predefined threshold, the course is considered relevant.

When that is the case the course gets added to the trajectory and the skills learned are added to the user's current skill set. At the same time, the list of missing skills is recomputed by comparing updated user skills with the target skills. The course also keeps track of which missing skills it helped cover.

This process is repeated iteratively, where in each iteration at most one course is added, until there are no more missing skills or no further progress can be made.

Once this is finished there have been multiple lists created for the output. The user's current skills, the skills not covered by the courses, the trajectory and the relevant skills learned from the courses are all displayed in the application.

6.2 Front-End

The front end of the system was designed in three phases. We will go through each of these phases, giving descriptions and design choices.

- Prototype of the Wireframe
- Figma Design
- Implementation

6.2.1 Lo-fi Prototype

This section focuses on the low fidelity wireframes for the web platform which is serving as the blueprint for how students will interact with the AI trajectory. In this part of the design, the primary goal was to create a linear, goal oriented roadmap by prioritising minimalist design and natural interactions.

Due to the nature of university students as a user base, the design rationale needed to focus heavily on clarity, motivation and the reduction of cognitive load. That's why, it was important that the structure and styling of the website prioritise approachability and minimalism. Furthermore, as students and most of the users can be overwhelmed by complex academic jargon, the user experience needs to feel familiar and goal oriented, ensuring that instructions are clearly given and the steps are visually obvious.

By keeping this in mind, we made mock ups of this user interface:

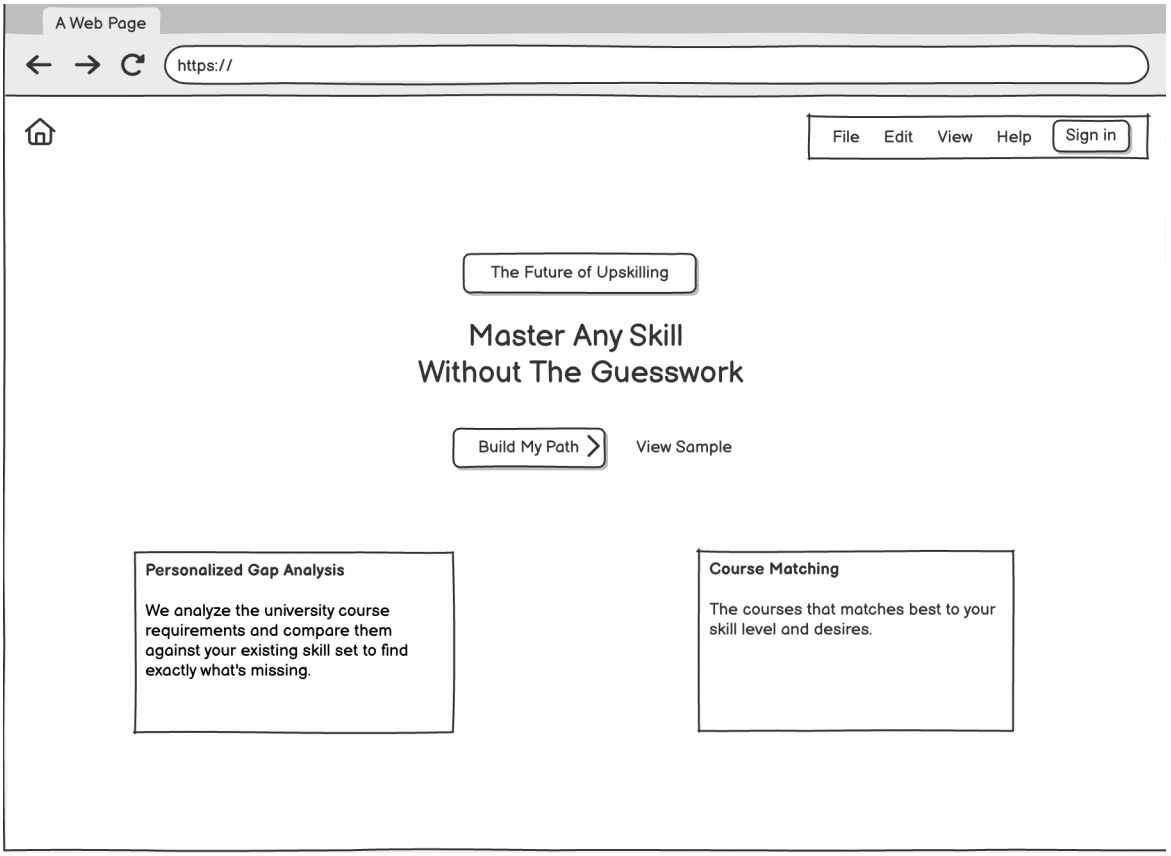

Figure 6.2.1: Lo-fi Main Page

The main page of the website was designed to give an explanation about the website. The landing page immediately communicates and showcases the platform's value proposition. There are two informational blocks below to set user expectations and explain how the platform will work.

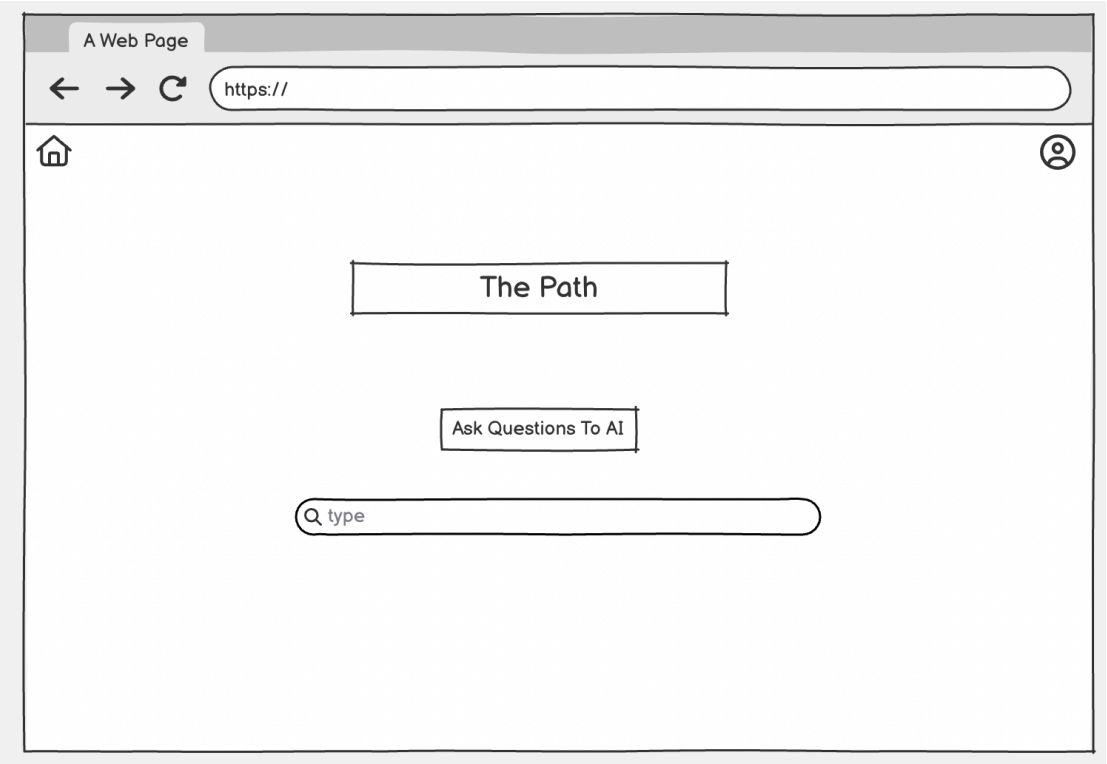

Figure 6.2.2: Lo-fi Inquiry Page

The Inquiry Page is focused on a minimalist AI search bar to give a natural interaction. Other than overwhelming academic filters (like faculty codes or credits) this page encourages users to type their goals freely, such as “I know Python and I want to become a Data Scientist”. This page serves as a data gathering stage for the LLM model, allowing AI to go through the user's interest and their current background before it scans the database to find relevant matches.

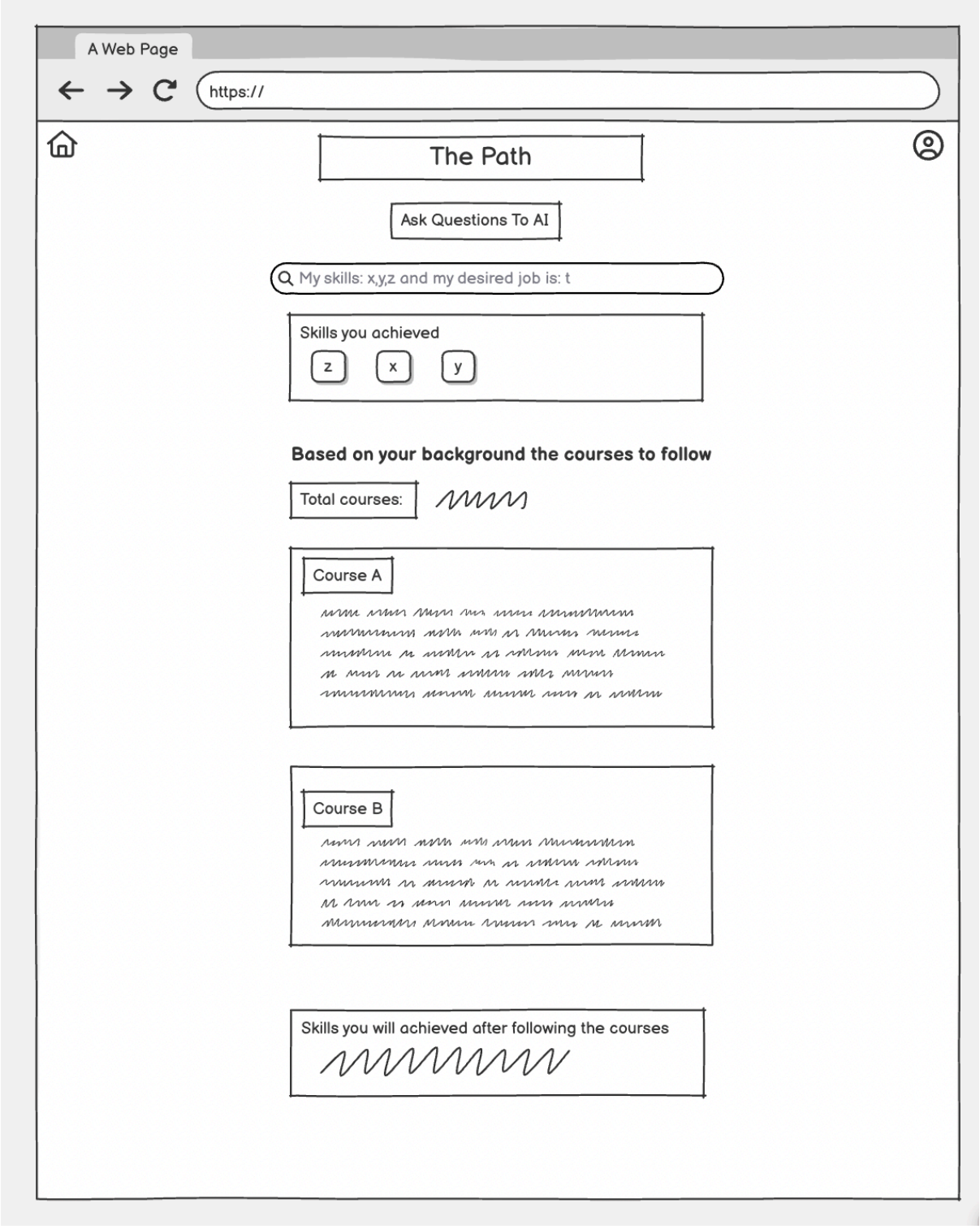

Figure 6.2.3: Lo-fi Path Page

The Path Page is the result interface for the user. It is designed to display the logical steps for choosing the required courses for users to reach their goal. The page first displays the user's baseline which is the skills they already achieved. And then lists possible courses as a list in a vertical sequence. This layout specifically shows how the "Exit Knowledge" of one course serves as the "Requirement" for the next. At the bottom part, the Skills You Will Achieve

section provides the user about what they will achieve after following this path, effectively summarizing the new skills they will possess upon completion of the courses.

6.2.2 Hi-fi Prototype

This section focuses on the high-fidelity prototype which is the final stage of the design process. This prototype was built using Figma and the functional requirements from the lo-fi stage are merged with a visual display. The design choice of a soft, warm color was aiming to reduce the academic anxiety and present an upskilling process. Mostly the layout and colourings used in Figma were taken to the final implementation of the website.

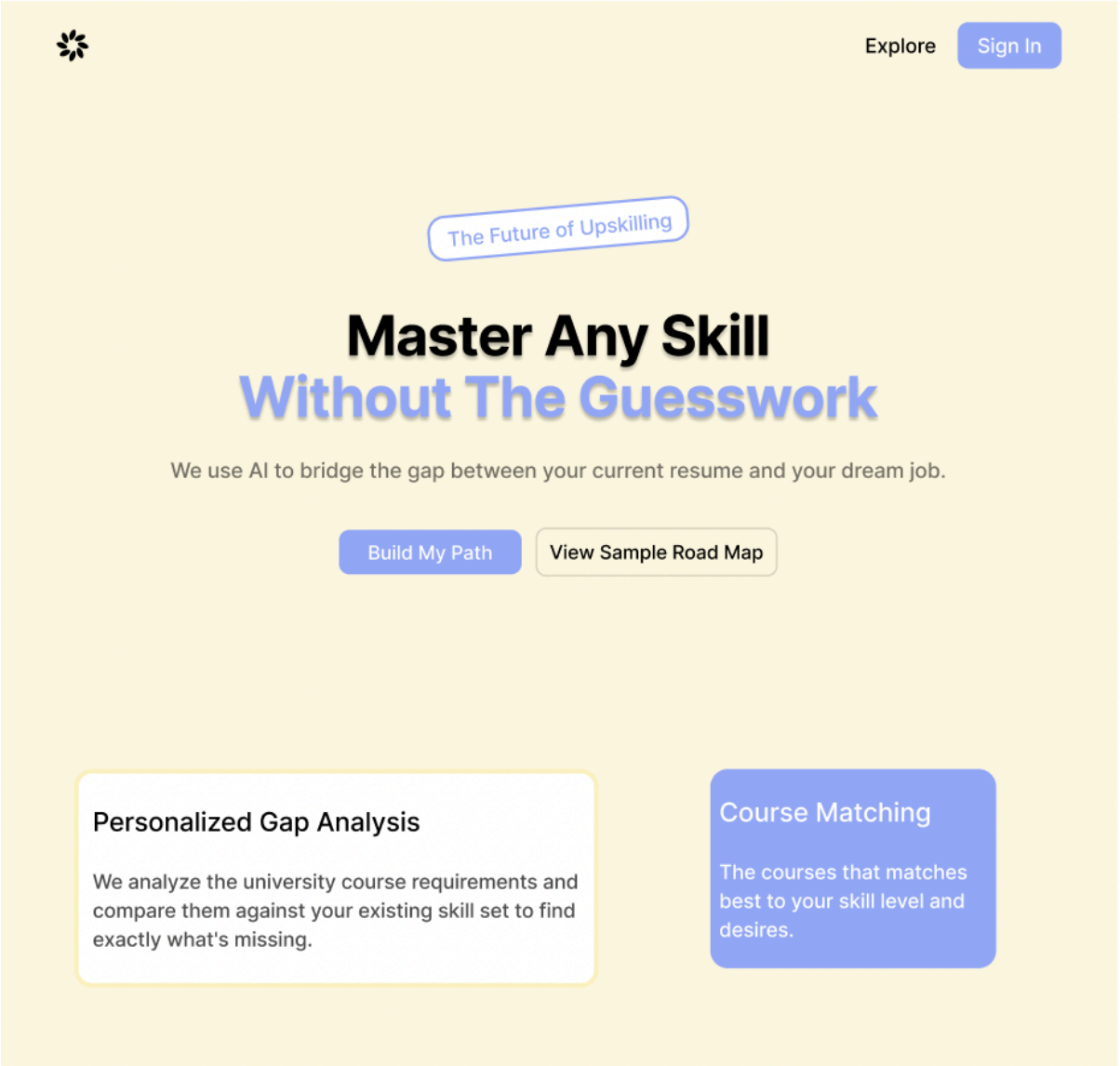

Figure 6.2.4: Hi-fi Main Page

The Home Screen serves as the primary page, emphasizing a clean value proposition. There are two action buttons offering users two entry points: one for immediate personalized action and the other one is exploring the previous entries to understand the website more. The

descriptive cards at the bottom are now visually distinct to reassure the user that the AI logic is structured.

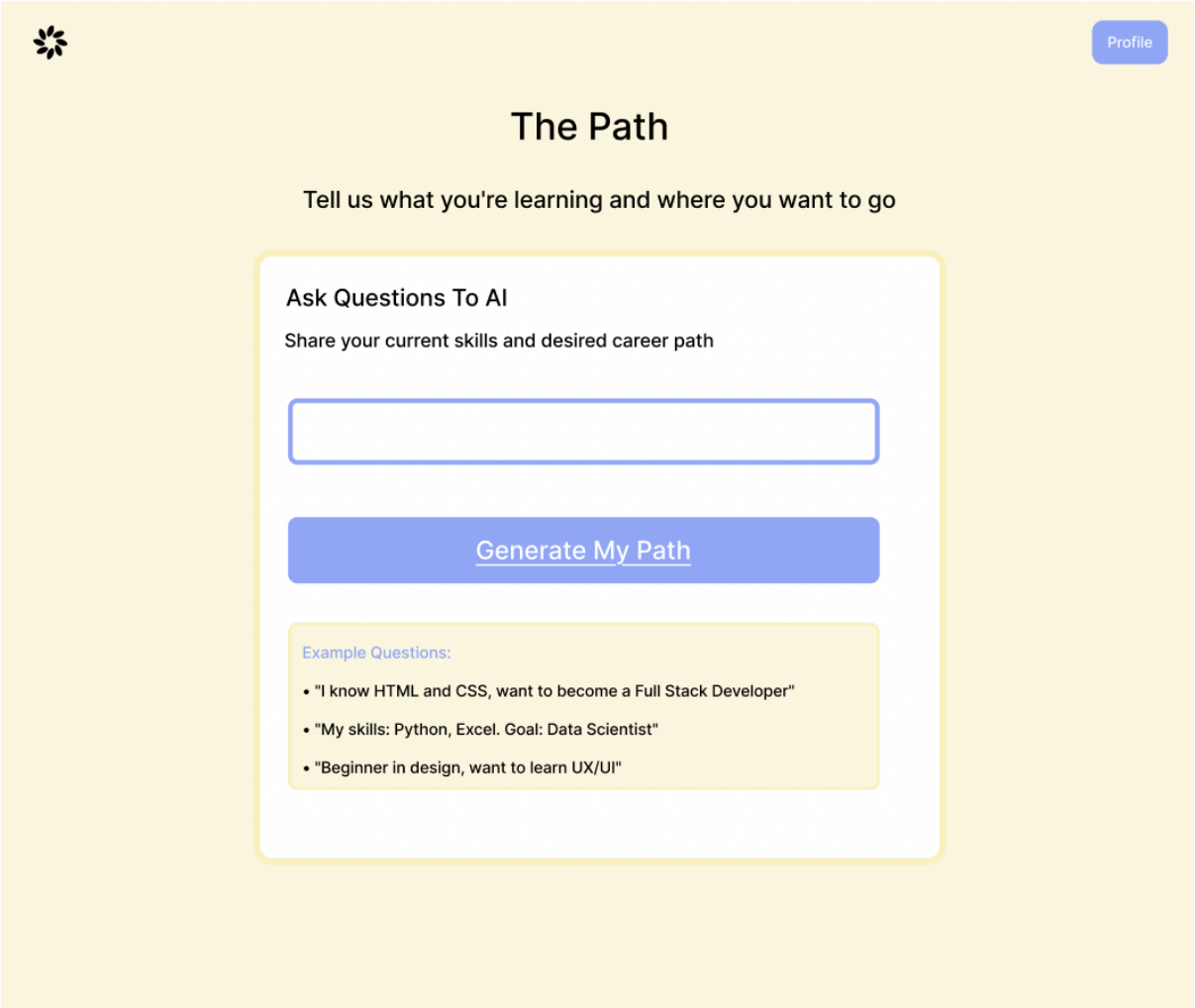

Figure 6.2.5: Hi-fi Inquiry Page

The Inquiry Screen was designed to act as a main portal for the LLM interaction. A new critical design addition here was the adding a block beneath the main input field for Example Questions. These examples give users an understanding about what they can ask the AI. By providing these templates, we are ensuring the AI receives the specific “Current Skills” and “Desired Job” context it needs to generate the accurate trajectory.

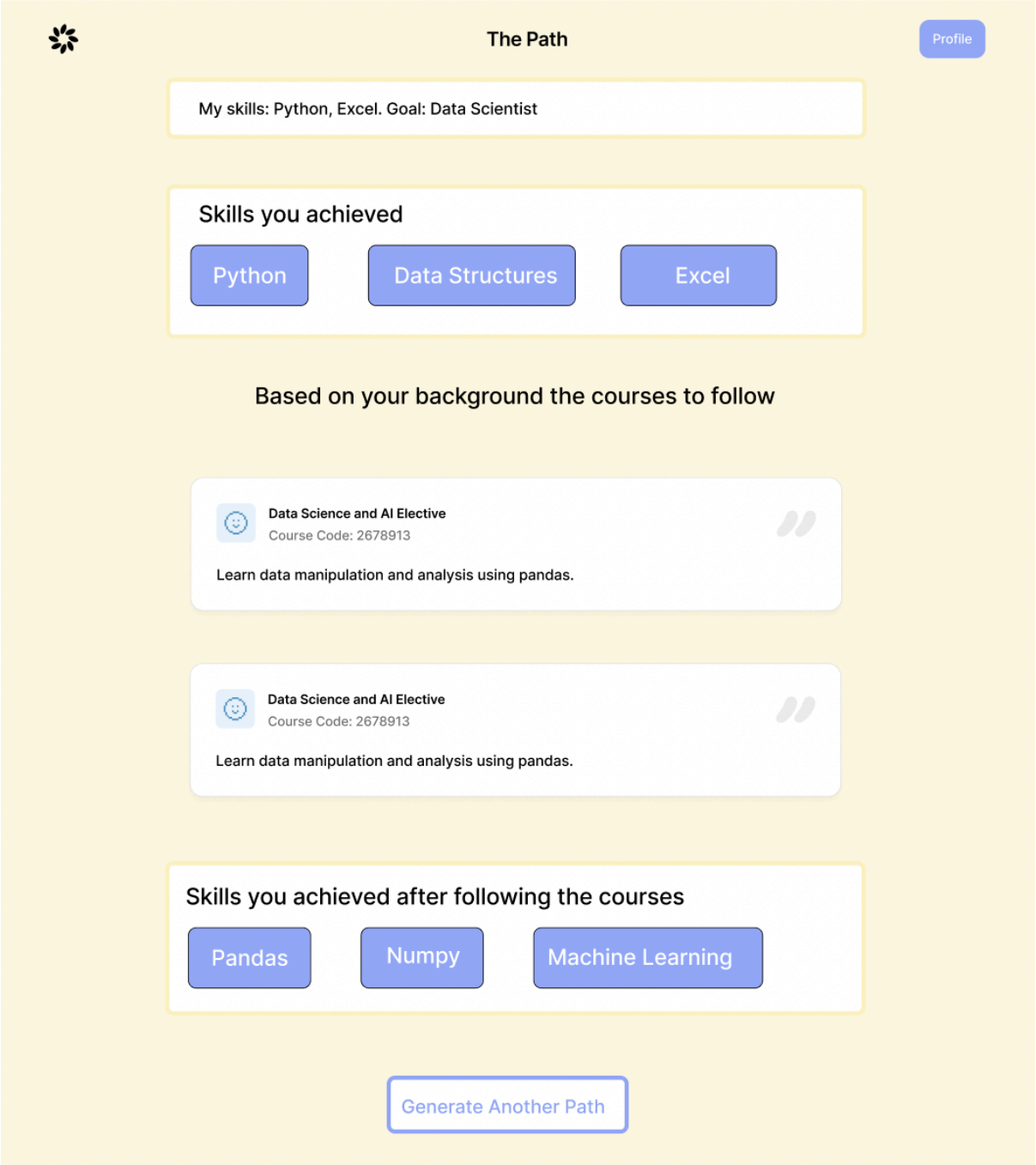

Figure 6.2.6: Hi-fi Path Page

The Result Screen is the page that successfully displays the output of the database and LLM parsing pipeline.

- Skills Tags: Current and future skills are presented as tags, making their skills as tangible assets they already collected.
- Course Cards: The courses to follow are presented in a clean and distinct cards style featuring all the information they need about that course.
- The Transformation Logic: The design separates the skills they achieved and they will achieve after they followed the courses to give a clear understanding about their end

skills. This helps users to understand exactly how courses will bridge the gap to their dream career.

6.2.3 React Implementation

The final version of the Learning Path Trajectory platform was developed using React styled with Tailwind CSS. We selected React for our implementation because of several technical reasons:

- The modular nature of React gives us a chance to build standardised UI components such as the use of “Action Cards” and “Skill Tags” while maintaining visual consistency across the entire website.
- React’s ecosystem allowed for an easy and straightforward integration with our backend logic. This facilitated the direct flow of the data from the scraped database and the LLM into the user’s view.
- React’s efficient state management was also crucial for managing dynamic interaction between user input and AI generated responses. Using React ensured a seamless transition while “The Path” page was being calculated and generated.

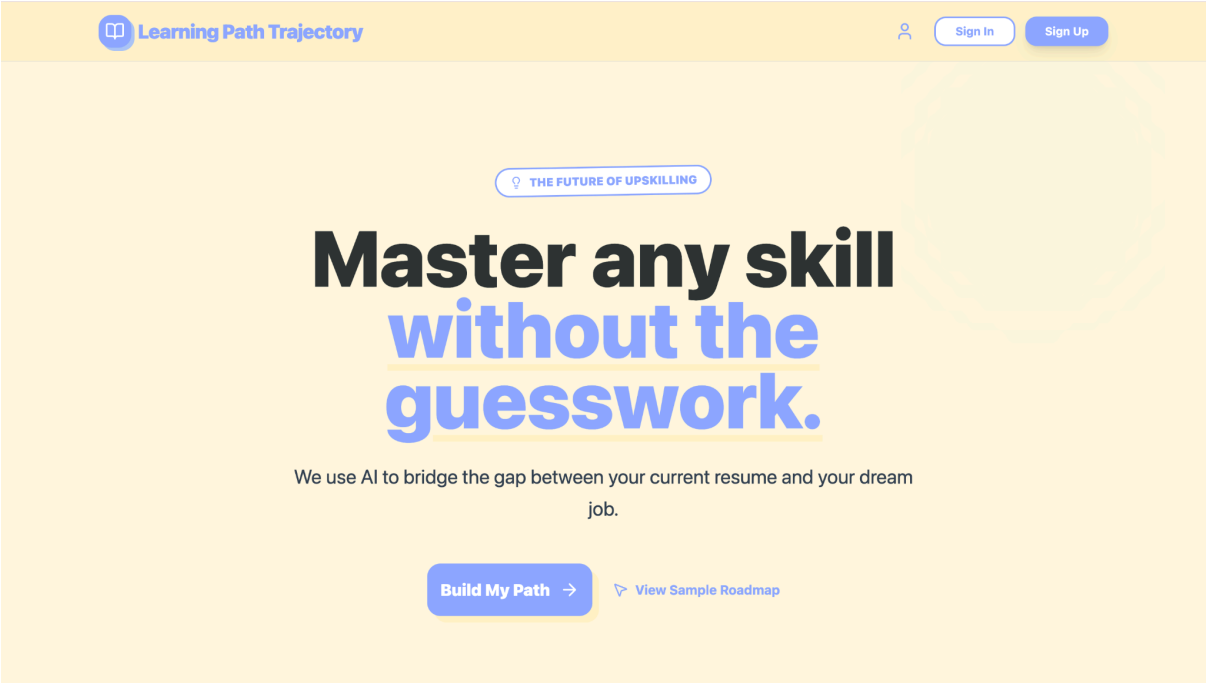

Figure 6.2.7: Main Page

The final version of the main page, mostly keeps the design choices of the hi-fi prototype. The main focus was to design a strong visual hierarchy to guide the user toward the core functionality. One of the most significant improvements from the hi-fi prototype to final implementation was the changes in the Navigation Architecture. In the hi-fi prototype the navigation consisted of simple text based and basic buttons. In the final implementation, we introduced a persistent header. On the left we added the name of the website and on the right we replaced text links with cleaner user icons. In other parts of the main page, we kept the

same design since it was already simple and good looking, however, we improved the design by changing the fonts and adding relative icons.

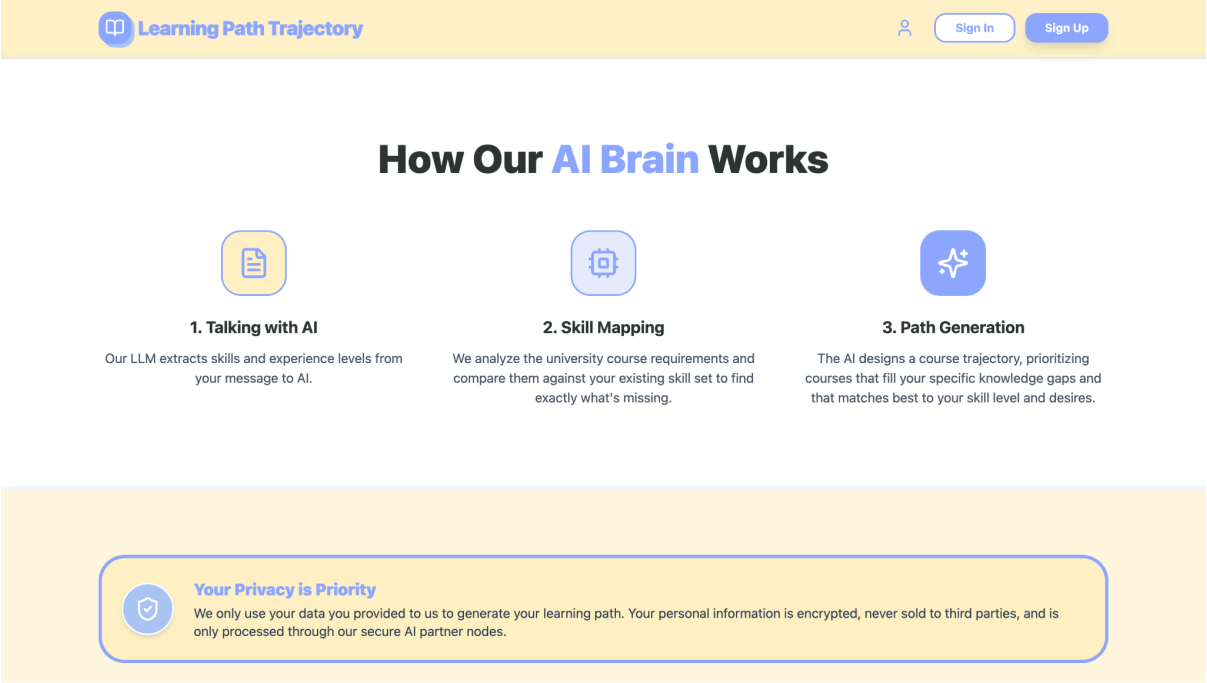

Figure 6.2.9: Main Page 2

Additionally, we added more information regarding the website and its function. We demonstrated the highly technical backend into a simple 3 step visual narrative. The other addition was to, adding information regarding the data privacy and the use of AI in the system. This was one of the topics discussed in Risk Analysis (Chapter 5) and adding this helped us to reassure users that their information is encrypted and handled securely.

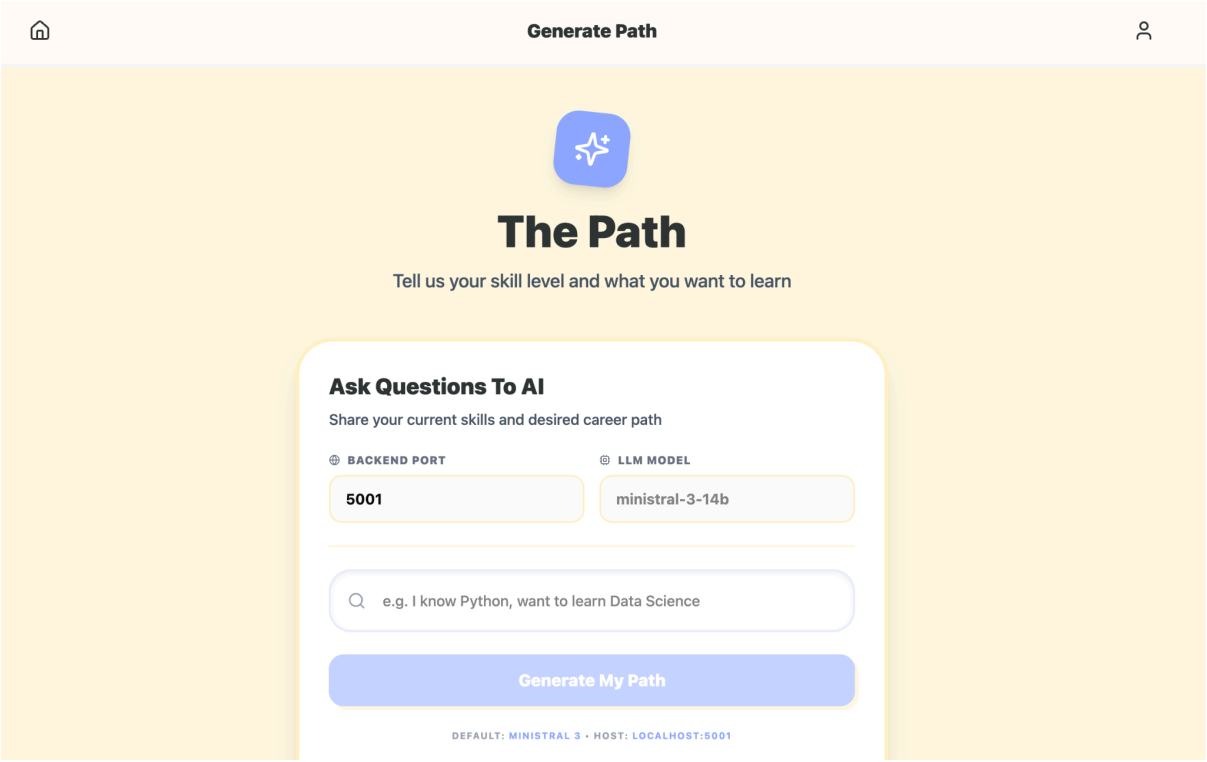

Figure 6.2.9: Inquiry Page

The Inquiry Page was also kept the same design with the hi-fi prototype. The main focus in this page was to create a distraction free page that prioritises the user's career goal. That's why we chose a prominent, single text area for users to type their goals freely, rather than overwhelming them with dozens of dropdown filters.

The Example Questions box also kept in the final design since it provided instructional scaffolding. This functional design aimed to help users provide the specific, structured information that the backend requires to generate a mathematical sound learning trajectory. During the integration with the backend we realized sometimes LLM can take quite a long time to load the path page, that's why we decided to add a loading. After the user clicks on the "Generate My Path" button, the button starts to show the percentage of how much the LLM model has already been configured.

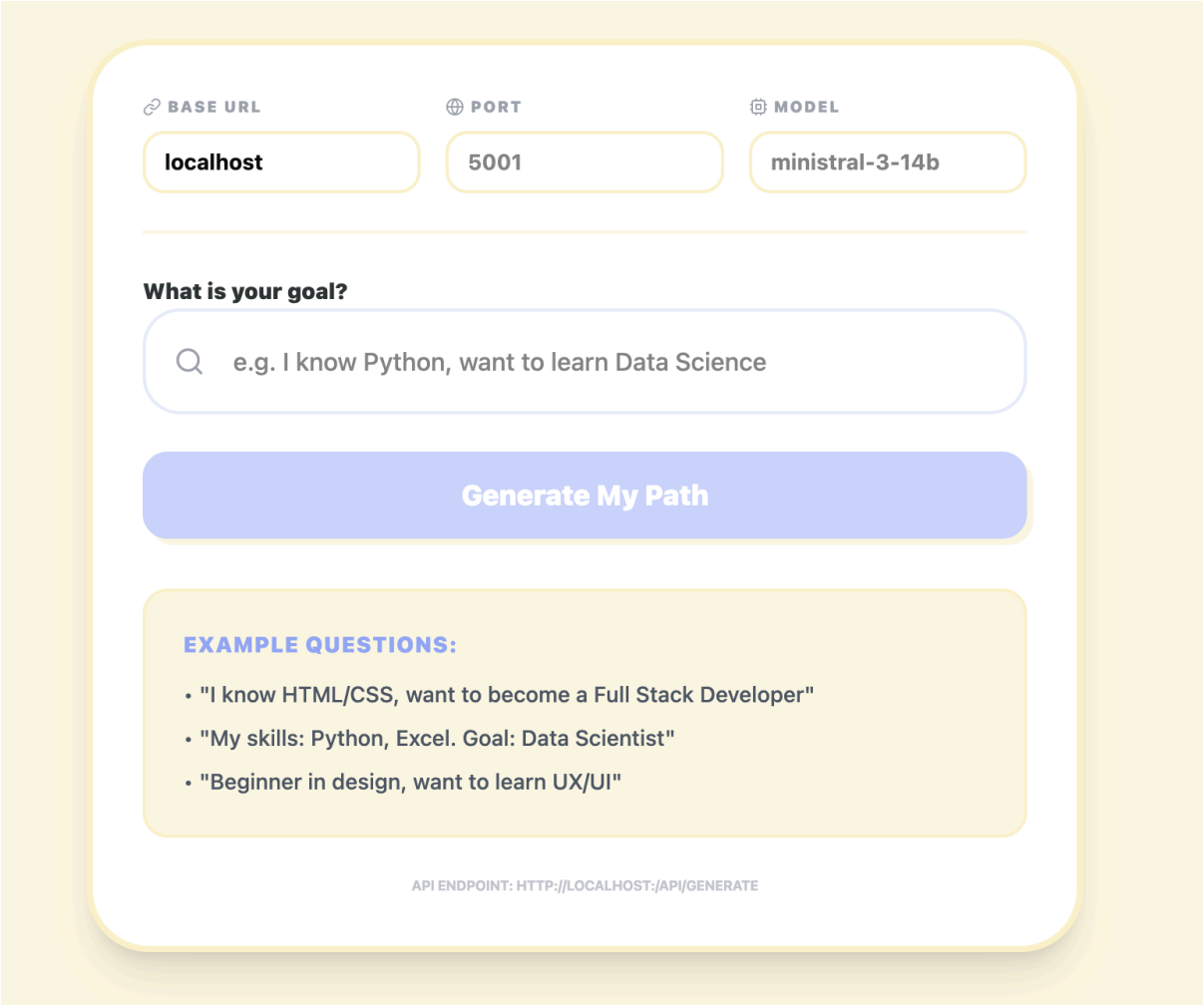

Figure 6.2.10: Inquiry Page 2

The other addition in this page was adding an option to choose your own LLM model as requested by the client. As shown in the picture, users are provided with input fields to manually define the Base Url, Port and Model. This gave an option to users to easily swap out the underlying AI engine without needing to make changes in the backend code. By routing the prompt through different language models, the user can observe how different LLM architectures handle semantic parsing and how their result would change.

However, to ensure this option to not hinder the core user experience, we made these configuration fields optional. If the user chooses not to specify these parameters, the system automatically defaults to the predetermined LLM model which is Ministral 3 14B.

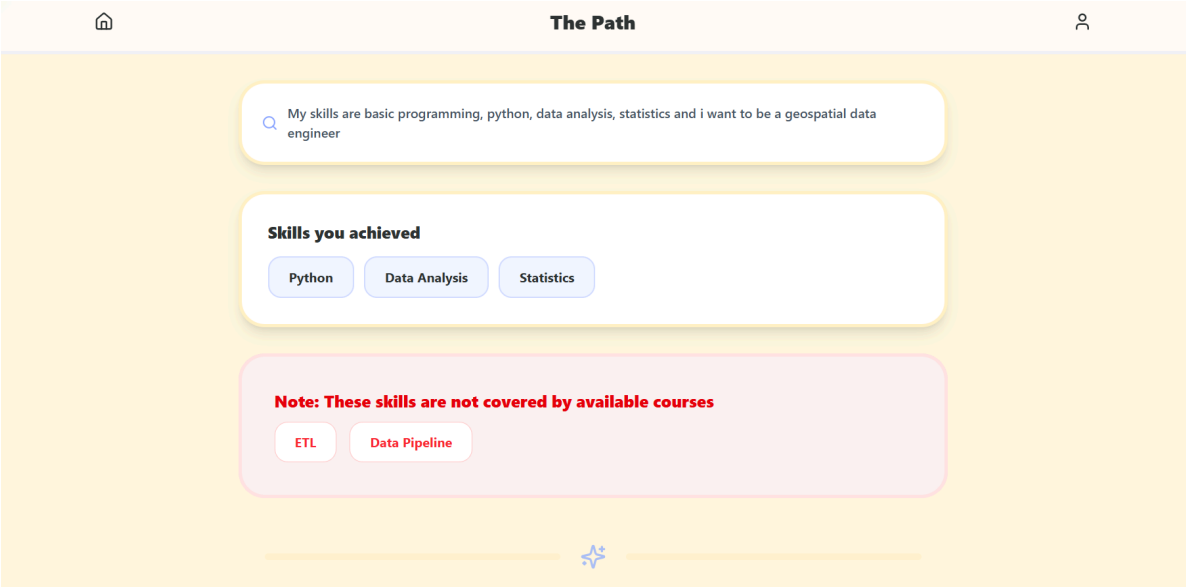

Figure 6.2.11: Path Page

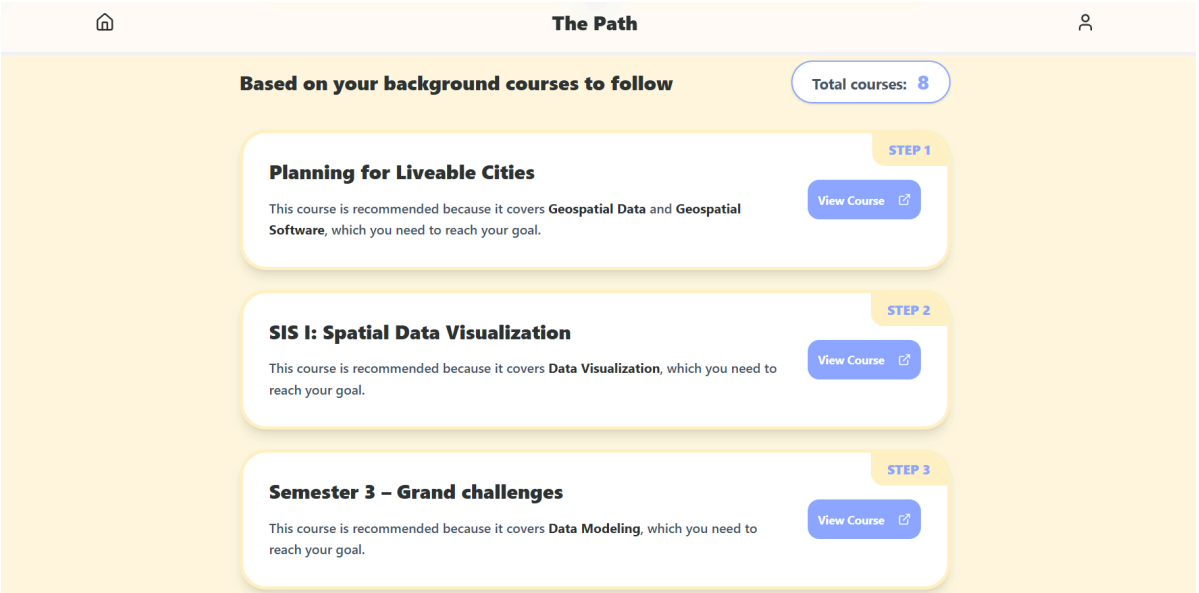

Figure 6.2.12: Path Page Courses

The Path Page followed the same design pattern as the hi-fi implementation. We presented the generated trajectory as a linear, top to bottom sequence to visually reinforce the concept of creating a pathway. Each course details provided with card based information to separate chunks of information.

We kept the same color and design options while changing the font type. The important addition in this page was to add a “Not Covered Skills” part. We realized that the backend was already collecting this information and it would be nice to present this information to the user too since none of the courses covers all the necessary skills they might be needing.

Additionally, we added an option to download the results in a json file as requested from the client. After the path page provides the details accordingly with the button at the end of the page, the user is allowed to download their results.

The main goal of this side of the system was to facilitate the collection of accurate context from the user regarding their current skills and their desired job to feed the LLM model, and to subsequently communicate the calculated course trajectory back to them. It was vital to facilitate this data exchange efficiently while not overwhelming the user with too much information such as displaying the raw, unstructured paragraphs scraped from the course catalog. By rendering only the extracted skills and brief AI summaries, this design ensured that the User Requirements were met by providing a cleaning recognizable environment for planning the curriculum.

Overall, this end product ensures that students and different users can engage with the platform intuitively and confidently, transforming a stressfully academic planning process into an encouraging and personalized journey.

Chapter 7

Testing

This section details the testing process and the methods employed to validate the system. By breaking down to three testing processes for each component of the project. To guarantee the reliability, accuracy and usability of the platform, testing focused on creating a strong prompt to achieve the smallest possible differences between different system launches.

7.1 Unit Testing

To verify the correct functioning of individual system components, unit testing was carried out.

- **Course Dataset extraction:** The dataset was processed using LLM, and each processed course record was manually checked for correct structure and content formatting. During this check, inconsistencies in the extracted data were identified, and the prompt for the model was rewritten to avoid poor-quality data and tested again.
- **User Input Processing**: Several different input scenarios were examined to ensure proper extraction of relevant keywords from the users' query input. For instance, user input "I am a Python programmer and I wish to become a Data Scientist" would result in keywords like Python, data science, and machine learning being extracted. Furthermore, several edge cases involving short input, long descriptions, and unclear queries were also considered.
- **LLM Calls:** LLM calls were used to verify that prompts were sent correctly and that valid, error-free responses were returned. Tests also confirmed that the system properly handled incomplete or delayed responses, ensuring stable and reliable interaction with the language model.

7.2 Integration Testing

- Integration testing focused on verifying connections between system components. This included checking the connection between the web interface and backend, as well as the interactions between backend services and the LLM. These tests ensured that user input was correctly transmitted across all layers without data loss and that responses between components were consistent. Error handling was also assessed to confirm that the system could gracefully handle failures while maintaining data integrity.

7.3 System Testing

- System testing assessed complete end-to-end workflows, examining the entire user journey from initial data entry to final course recommendations. We tested the accuracy of generated recommendations, the clarity of explanations and the correct display of results in the user interface. During testing, several issues were identified, such as incorrect keyword extraction. These were addressed by refining prompts and improving preprocessing logic.

7.4 User Testing

- The system was tested by several users (team members) to assess system usability and interface clarity. Testing focused on the ease of use of the interface, the clarity of instructions, and the usefulness of the generated recommendations. Based on the feedback received, several improvements were made, such as adding clearer explanations for each course, improving user input instructions, and enhancing the overall readability of the interface.

7.5 Performance Testing

- One of the main concerns is performance, as the need to run a local LLM model requires significant computational resources. We had to wait a very long time for a response from the LLM.
- After performance testing, some work was done to improve response time, preprocessing and skill normalization techniques were applied. These optimizations reduced latency and improved overall system responsiveness.
- These optimizations improved performance, although deploying the system on more powerful hardware would further reduce response times.

Chapter 8

General Discussion

This section assesses the system's overall success in meeting the initial project objectives and user requirements. By analyzing the accuracy of the generation of course trajectories, the practical usability of the frontend interface, this chapter measures the platform's real world effectiveness. This section also provides the project's outcomes, reflecting on the development journey and outlining the potential enhancements. This final discussion addresses the current limitations of the current system and highlights the lesson learned during the process.

8.1 Fulfilled Requirements

8.1.1 User Requirements

The user must be able to describe their current knowledge profile (starting point of the student) and their desired job (goal of the student) in natural language.	X
The user must be able to define a desired profile / job.	X
The user must receive a clear explanation regarding the AI choices of the courses that forms a logical educational journey. The system should indicate why it was selected.	X
The output must include the official Course IDs or links to the OSIRIS page to allow the student to actually enrol the course.	X
The recommended courses must align with their specific domain goals.	X
The user should be able to ask multiple questions.	X
The student should access the tool via a simple web based interface without needing technical AI knowledge.	X
If multiple paths exist, the UI should allow the user to compare them side by side. (e.g. Shortest Path vs Deep Dive)	
The system should have an user-friendly interface.	X
The user can login and get back to an earlier given trajectory or change an earlier extracted profile.	

8.1.2 System Requirements

The pipeline must identify specific keywords for entrance requirements and for desired completion outcomes.	X
All extracted course data must be stored in a structured format to provide efficient searching and future expansion.	X
All extracted course requirements keywords must be stored in a structured format to provide efficient searching and future expansion.	X
The backend must output an ordered list of courses that logically follows from a user's starting profile to their destination goal.	X
The system must be tested using specific scenarios.	X
The system should be continuously tested during the implementation process to assure a working project in the end.	X
The system must be hosted in an environment that guarantees no data of users leaks, satisfying the requirement to handle non-pıublic or sensitive information.	X
The system must use an LLM model to achieve a good contextual matching.	X
The system must extract from the users input the current knowledge profile and desired knowledge profile	X
The system must give multiple trajectory paths in case of multiple path options.	
The model should be lightweight enough to run as a web service while maintaining high accuracy in trajectory development.	X
The system should be designed as expanded so the other departments or institutions can also use it in the future.	X
The system can remember the user's trajectory or remember the current and desired profile in case the user is logged in.	
The system should implement a source referencing mechanism where every course in a trajectory is linked to a section of the database to prevent fake prerequisites.	X
The system can give multiple trajectory paths in case of multiple path options and let the user choose their preferred path.	

The system can collect OSIRIS data and extract prerequisite knowledge and expected knowledge levels for each course.	X
The system will not have a real time connection with OSIRIS. It will rely on the scraped JSON data store.	X
The system will not store sensitive student information such as academic records or personal identifiers permanently on the server.	X
The system will not be integrated with osiris.	X
The system will not over explain the reasoning of why they chose that specific course. It should be short and clear.	X

8.1.1 Non-Functional Requirements

The user must not wait longer than 1 minute before the program outputs a trajectory.	
Users should be able to navigate the menu in the first 2 minutes of using the website for the first time.	X
The system must achieve a 100% success rate in JSON schema validation for every output, ensuring there is no malformed data in the user interface.	X
The system must provide a justification for the output trajectory created.	X
The system should be easily updated and maintained.	X
The backend must maintain stable accuracy.	X
The system must record all AI reasoning steps.	
The system should be easily used without additional help.	X
The system should iteratively learn from earlier reasoning and feedback.	X
Every recommended course must include a source reference so students can check the AI's logic in the official university data.	X
The system must be clear in what profiles are determined so the user can give feedback in case of mistakes.	X
The website should display the details of the usage of AI and LLM models.	X

8.2 Limitations

While the prototype successfully demonstrates the feasibility of an AI driven trajectory, there are several technical and structural limitations exist within the current implementation version:

- LLM Token Inconsistency: The system fully relies on the Ministral 3 (14B) model to extract “Skill Tokens” from unstructured input from the user. Because LLMs are probabilistic, the model may occasionally extract synonymous words but distinct tokens for the same concept (e.g, extracting “Python Programming” for one course and just showing “Python” for another one). This semantic variance can sometimes create missing links in the logic of the system causing the AI to miss a valid course connection.
- Real Time Latency: Generating the path for the user requires complex semantic matching and processing of LLM. In a live environment, this can sometimes cause delays for the user. Adding the loading state in the frontend fortunately, prevents users from thinking the application has frozen.
- Different Results: The accuracy and the relevance of the generated trajectory constrained by the database used. The current system uses different datasets such as EEMCS courses dataset or Geoinformation courses. Because of that, it only gives results in the current dataset topic and if a user inputs career goals that falls entirely outside of this scope, the system lacks the foundational data to recommend necessary courses.

8.3 Reflections

Beyond the technical implementations, this project taught us the critical importance of structured team communication and time management. Building a full stack AI application required coordination between the backend data engineers to provide a functional project. Regular internal communication and version control were essential to prevent any integration bottlenecks that might happen when connecting with the front end. That’s why we did weekly meetings as a group to meet and discuss the project and follow our plans. Apart from the weekly meetings, we divided the workload weekly and worked on our predetermined tasks. We believe that we followed our initial planning and finished the project on time. Finally, regular meetings with our project supervisors provided invaluable strategic direction and helped us to ensure the final prototype remained tightly aligned with the end user requirements.

8.4 Future Work

To transit this prototype into a fully deployable, university wide platform, several architectural and feature enhancements are recommended to do:

- Automated Data Pipelines: The current version of the website works as a pipeline for presenting your own course dataset and creating a path trajectory according to your desires. However, adding an automated scraping script that collects the course details automatically inside the backend itself by only presenting a website link or a course file would ensure the user can use any database they want.
- Persistent User Accounts: The current version of the React frontend already includes the necessary front end designs for a possible account system. (Appendix A) In the future, including a fully functional backend authentication system would allow users to save their trajectories, mark courses as completed and automatically update their current skills over multiple semesters.
- University Integration: Ideally, the platform could be integrated directly into the university's official website or into the Osiris itself, providing students with a seamless, institutionally backed planning tool they can use.
- Multiple Path Trajectory: The current version of the implementation is providing only one path the user can follow and they are allowed to ask for generating another path. The system can be improved to provide all the possible paths or additional paths the user can take at the same time. This can help users to see more of the options and choose what they want the most.

Chapter 9

Conclusion

The main focus of this project was to bring a new approach to students' experience regarding the choosing of courses to follow from static, complex university course catalogs. By recognizing that the traditional systems only outline what a course teaches but fail to provide a more personalized way to reach that skill path, this project aims to bridge that gap between a student’s current skills set and their ultimate career plans.

With the successful design and the implementation of the Learning Path Trajectory platform, we know that this gap can now be effectively closed using modern artificial intelligence. By integrating the Ministral 3 (14B) Large Language Model, the system now successfully performs true semantic parsing and translates dense syllabus paragraphs into discrete skills tokens.

The first pipeline successfully transformed the raw, scraped data collected from courses into a structured knowledge graph by parsing dense syllabus paragraphs into discrete skill tokens to be used. The second pipeline solved the inherent unpredictability of LLMs by introducing a semantic normalizer. This was applied by using a cosine similarity. With this the system ensured that user inputted goals and generated profiles were accurate and mapped to the list of skills. This helped us to demonstrate the highly accurate, step by step trajectory that identifies which skills are fulfilled clearly and which remaining skills are missing from these courses.

Furthermore, the backend logic is implemented into a responsive React frontend. By allowing students and any users to input their goals as they wanted, the platform transformed the struggle of guessing and figuring out which courses to take and academic planning process into a more encouraging, visual and personalized user journey.

Even though the current implementation has some limitations such as dataset scope or real time processing latency, the foundational architecture is remarkably robust. In the end, this project shows that Large Language Models can successfully transform static academic databases and a frustrated process into dynamic, actionable career road maps, empowering learners to master any skill without the guesswork.

References

[1] Mulder, T. (2026). *GovtBench: A large language model benchmark and evaluation framework for the Dutch public sector* [Master's thesis, University of Twente].
[2] Atef, K. (2024). *Online Courses Dataset* [Data set]. Kaggle. https://www.kaggle.com/datasets/khaledatef1/online-courses/data
[3] Musazade, N., Mezei, J., & Zhang, M. (2026, March 3). *UniSkill: a dataset for matching university curricula to professional competencies*. arXiv.org. https://arxiv.org/abs/2603.03134

[4] Tamascelli, M., Bunch, O., Fowler, B., Taeb, M., & Cohen, A. (2025). Academic Advising Chatbot Powered with AI Agent. *ACM Other Conferences*, 195–202. https://doi.org/10.1145/3696673.3723065

Appendix A: Additional Front End Designs

This appendix focuses on the front end implementation for the user authentication, specifically the Login and Sign Up pages for the Learning Path Trajectory platform.

Due to the time constraints of the 10 week development process, we prioritised the core AI pipelines and the backend infrastructure required for this page such as password hashing, tokenized session management and database were not implemented during this process. However, the frontend was implemented to lay the immediate groundwork for the "User Accounts" detailed in Section 8.4 (Future Work). These interfaces were fully designed and developed using React and Tailwind CSS same as the other parts of the website. The design maintains the platform's main goal of creating minimalist, accessible design.

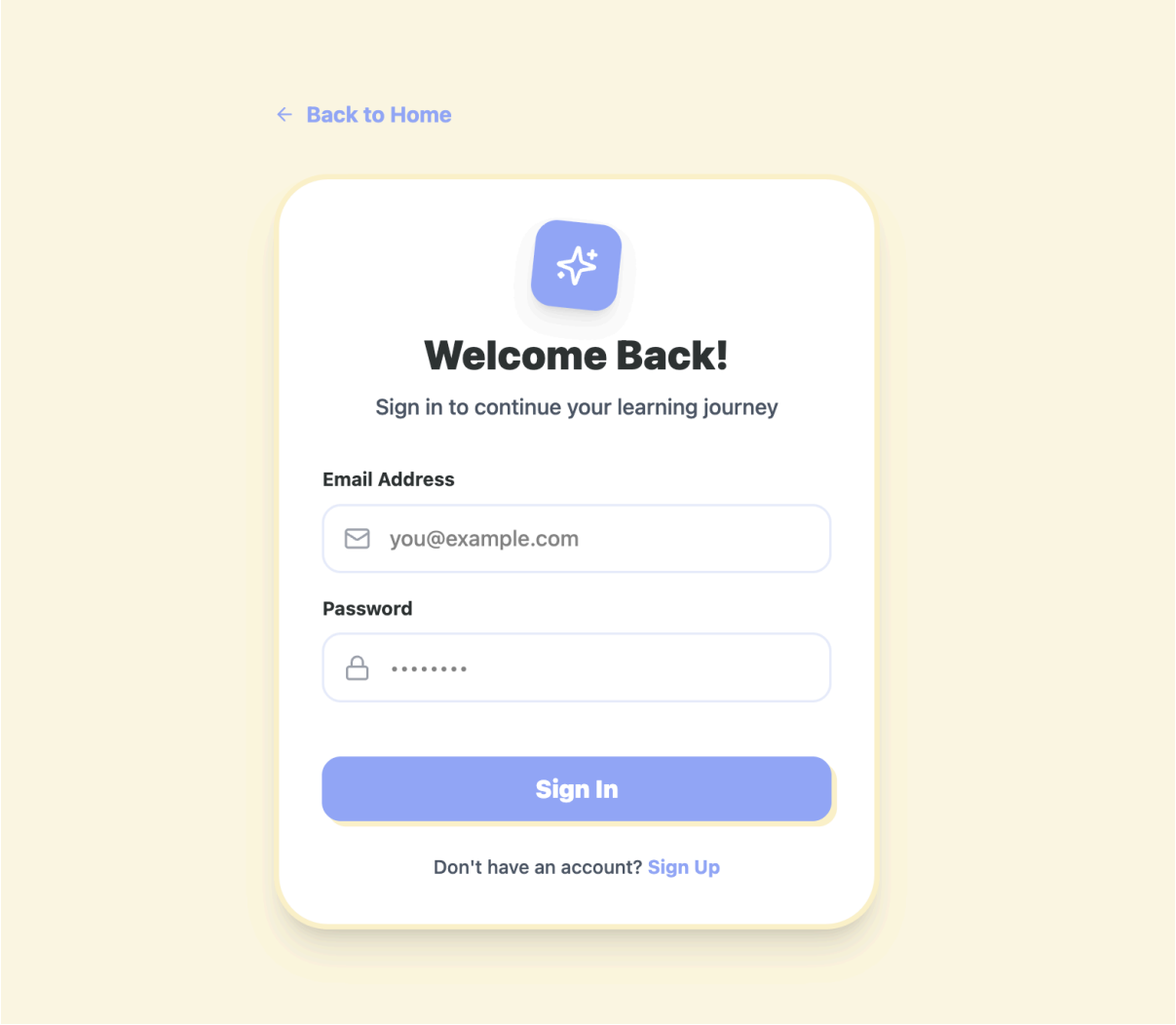

Figure Appendix A. 1: Sign In Page

The Login page is designed for users to securely access their saved trajectories and update their current skills.

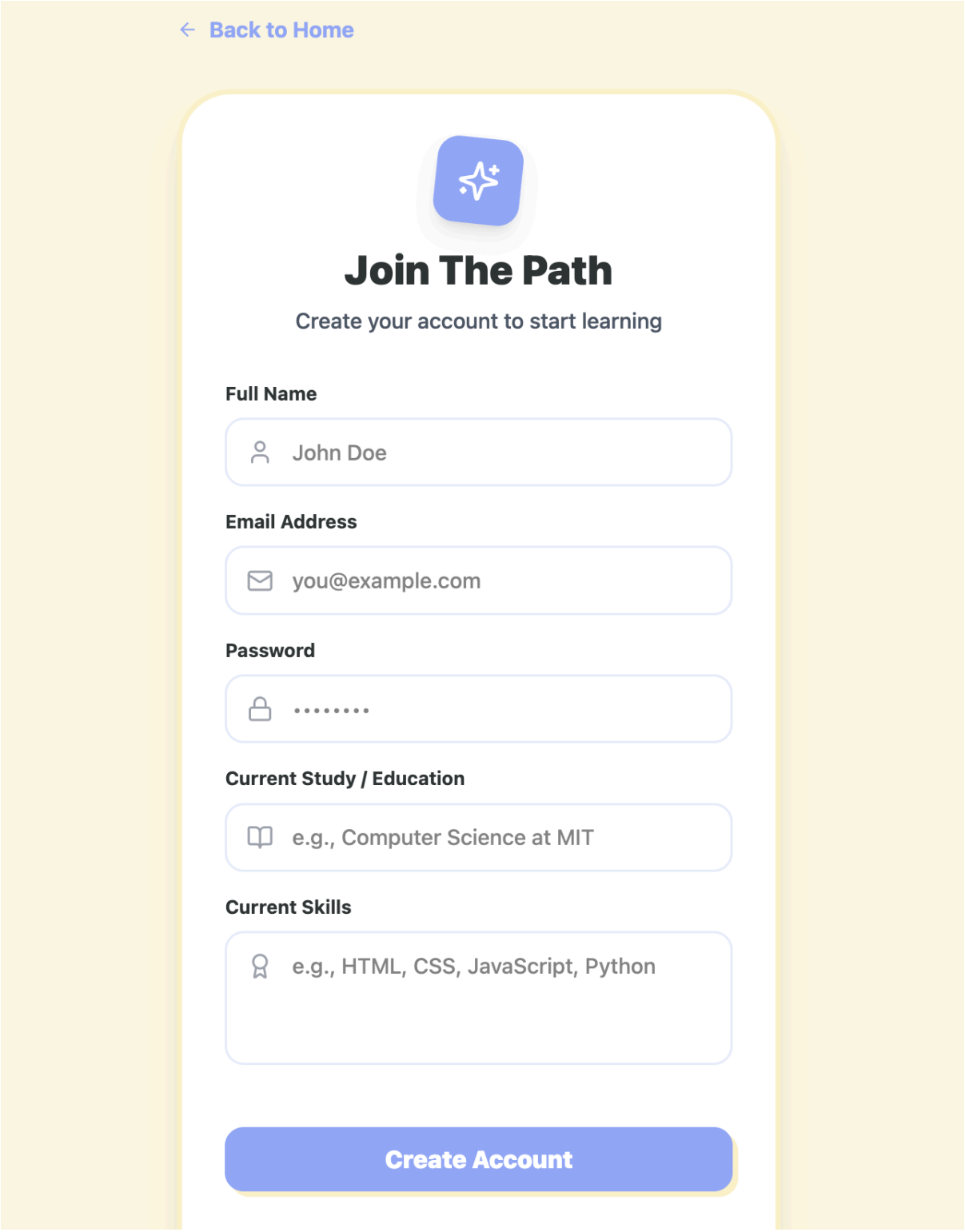

Figure Appendix A. 2: Sign Up Page

The Sign Up page is designed to onboard new users easily without overwhelming them with unnecessary data.

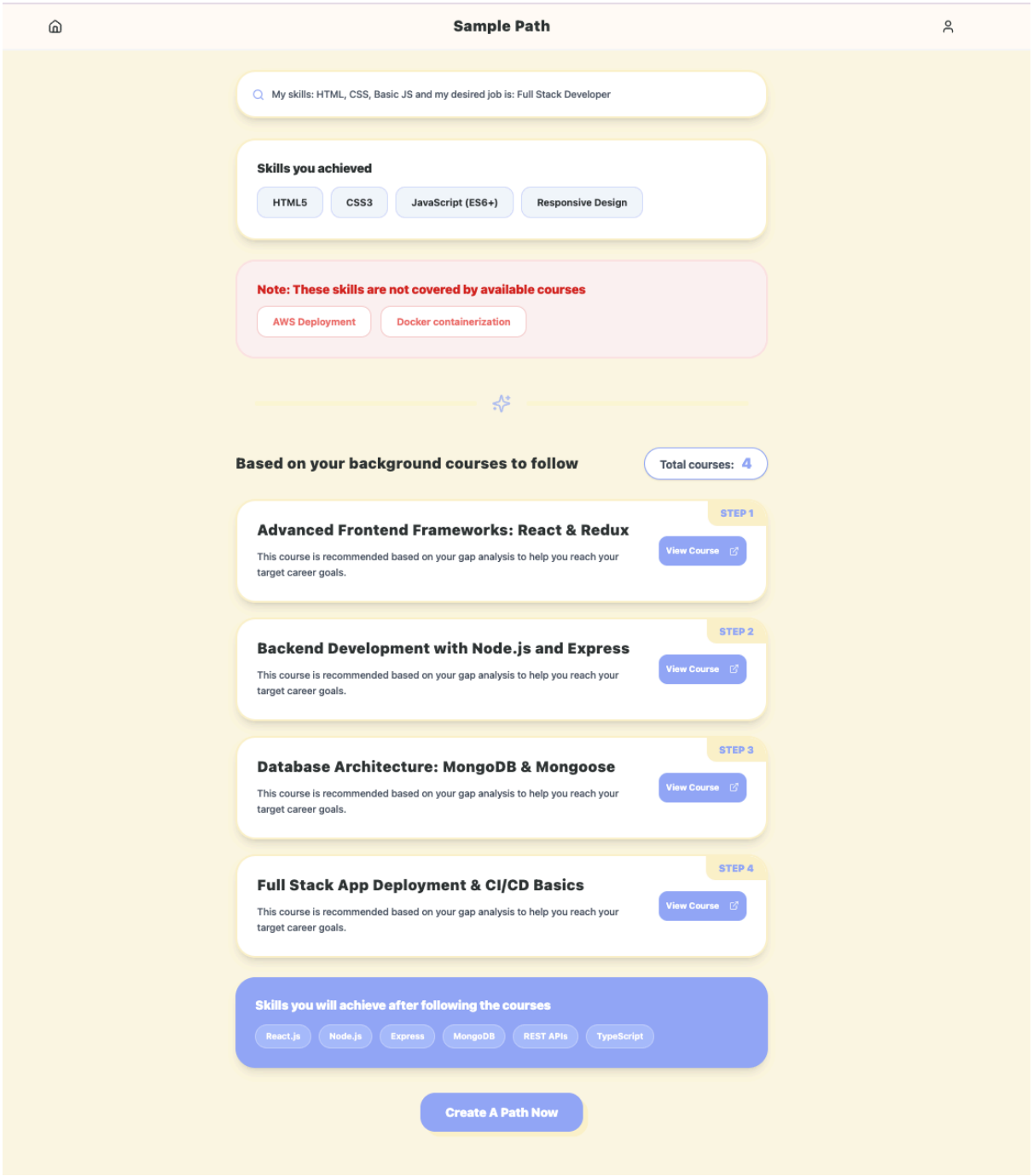

Figure Appendix A. 3: Sample Road Page

Additionally, we also added a sample path page to give an idea to users how their path results would look like. The page mimicked the path page and provided the exact same results with a mock data.