Product Requirements Document (PRD)

The goal of the PRD is to address four critical risks:

Risk	Perspective
Value	User
Usability	User
Feasibility	Engineering
Viability	Business

Problem Description

Job seeking is a difficult endeavor for entry level candidates. We have identified two ways people get jobs through a referral or through mass applying to open positions via the internet. Candidates need a way to speed up the second process in order to maximize their chances of employment

Liberal Arts students often struggle to find relevant mentorship in their career paths. Our solution is a website that allows students to connect with mentors based on industry and role.

Scope

The platform serves Liberal Arts students by providing access to a database of mentors, focusing on facilitating meaningful connections.

Use Cases

The primary use case is for Liberal Arts students to efficiently find and connect with mentors, enhancing their career prospects and industry understanding.

Purpose and Vision (Background)

To democratize access to career mentorship for Liberal Arts students, aspiring to become the go-to platform for mentorship connections in this demographic.

Stakeholders

Investors: They will need updates quarterly to see if we are meeting benchmarks set in our pitch deck. (Users and Revenue)

Sales Team: Bi-Weekly check in to see how they are recruiting employers for recruiting services/platforms.

Users: They will need updates when bugs are fixed/big platform changes occur. Also when ToS change they will need to be informed

Customers (Employers): This group will be most concerned about pricing and reach of our services. These people do not have a defined reach out schedule except when pricing changes occur but will need to be responded to quickly if they reach out to us.

Preliminary Context

Assumptions

Assumes a demand for a specialized mentorship platform and the ability to develop it using modern web technologies.

Constraints

Development is limited by budget, timelines, and the need for a scalable, secure architecture.

Dependencies

Success relies on robust technology choices: Next.js for the front end, Planetscale as the database, and Clerk for authentication.

Market Assessment and Competition

Positions against generic networking platforms by offering specialized mentorship for Liberal Arts students.

Requirements

Key features include mentor-student profile creation, search functionality, and secure messaging. Priority is given to usability and data security.

Non-Functional Requirements

1. Our product should not take too long to load properly. We want to ensure that our users are able to use our product in a time efficient manner
2. We want our website to have a solid foundation in terms of its security with communication between the client and server being encrypted using SSL/TLS
3. All code behind our website as well as our product will be documented, following coding standards
4. Logging and Monitoring: We want to implement robust logging and monitoring mechanisms for identifying as well as diagnosing issues promptly.
5. This product should be available for free access 24 hours a day, seven days throughout the week, unless we are scheduling a server shutdown for maintenance
6. This product should follow along with relevant data protection regulations

Data Requirements

We need to collect user resumes and be able to store them into a database such as but not limited to MySQL.

Therefore we need to store things such as Names, Address, Phone, Email, Experiences, Projects, References, Skills, and more various information through a database for each individual user.

User Interaction and Design

1. Simplicity/Clarity
   a. Clear Navigation: Simple and intuitive navigation that guides users through the process.
   b. Minimalism: Uncluttered interface with only essential elements, reducing cognitive load on users.
2. Feedback and Guidance:
   a. Real-time feedback: Provide instant feedback on user actions, such as successful resume upload or completion of a questionnaire
   b. Guided Process: Clearly indicate the steps users need to follow, providing tooltips or guidance wherever necessary.

Milestones and Timeline

Planning and Ideas -> Designing and Prototypes -> Incorporate the AI API tools -> Develop core features -> Launch Alpha Prototype (MVP)

Goals and Success Metrics

Goal	Metric	Baseline	Target	Tracking Method
Get a group of mentors to join the site.	Average duration to apply for a job	30 minutes to 2 hours	15 minutes to 30 minutes.	Name page till Job Submitted Page
Liberal Arts students have a much better direction	Baseline survey on the average amount of students using site	Survey before using site	Increase the confidence in plan of how to work the system increase it by 50%.	Every Complete page on a job application
Product-market fit	How would you feel if you could no longer use this product?	Very disappointed < 40%	Very disappointed > 40%	Interview
Number of Users	Self-Explanatory	0	50	Basic Website Analysis

Software Development Process (SDP)

[Principles]
[Process]
[Roles]

Principles

- Backlog: We will utilize Jira to manage our weekly tasks
- Git Branching: We will create new branches for features and bug fixes as to not mess with the main branch
- Tasks Size: We will make tasks that ideally do not exceed one week of work.
- Review and Approval: We will talk with all members of the group before we commit anything to main from our branches.
- Progress Documentation: We will document our progress through information sharing through Discord or Team Messages.

Process

- Define clear objectives for the next two weeks
- Prioritize features, bug fixes, and technical debts
- Columns: Backlog - Development - Testing - Peer Review - Deployment
- Tasks move through each phase for a clear visual representation of our teams progress
- Asynchronous updates: Our team will have annual updates regarding what team members did plans for our next steps, as well as any blockers we are facing.
- Automated tests run on every code push
- Ensures code quality, reduces manual deployment efforts
- Gather information about workdays job applications from various companies
- Create a simple as well as user-friendly external interface
- Write code for the autofill process

- Develop functions incorporating AI engines such as ChatGPT’s API for unfamiliar questions on the workdays application
- Test different resumes and scenarios to ensure a working project

Roles

- Gather information about user and mentor requirements and how to
- Create an external interface that is both simple to use and easy to interact with.
- Write the code for the autofill process.
- Test different resumes and scenarios to provide a working project.

Tooling

Version Control	GitHub
Project Management	Jira and Github
Documentation	Github
Database	Planetscale (MySQL)
AI LLM	ChatGPT API
IDE	Visual Studio Code
Graphic Design	Figma
Collaboration	Discord and Messages

Definition of Done (DoD)

Our definition is that first the pull request is reviewed by another member of the team. Unit tests must be put into the code. After that is done and the code passes the basic automated tests then its reviewed by another team member in the form of a pull request. Finally the team will at the end of the sprint go over tasks complete and make sure they fit as part of the larger ecosystem. Once the pull request is submitted that means the task is probationally done once its reviewed at end of sprint is when its actually done.

Release Cycle

Our release cycle will follow that code is kept in the dev branch and pull requests are reviewed before being merged.
It gets deployed to production at the end of every major milestone. We will follow the MAJOR.minor.patch format

Minor versions for new features Patch version for bug fixes
Major version for breaking API changes

Environments

Environment	Infrastructure	Deployment	What is it for?	Monitoring
Production	Still Unknown to this time	Release	Present a platform for Liberal Arts students to connect with mentors.	N/A
Staging (Test)	Jest	PR	New unreleased features and integration tests	All Group Members
Dev	Chromium based Web Browser	Commit	Development and unit tests	N/A

Software Design and Architecture

Introduction

The architectural vision for Origin Point Career Strategy's mentorship platform centers on creating a secure, scalable, and user-friendly environment that connects Liberal Arts students with industry professionals for guidance and mentorship. Emphasizing security, the platform employs Clerk for robust authentication processes, ensuring user data is protected at all levels. Scalability is achieved through a microservices architecture and the use of Planetscale, allowing for efficient data management and the capability to grow seamlessly with the user base. The user experience is prioritized with the adoption of Next.js for a dynamic and responsive interface, ensuring ease of access and interaction for users across devices. This architecture supports the platform's mission to facilitate meaningful career advancements for Liberal Arts students through mentorship, embodying a balance between technical excellence and practical usability.

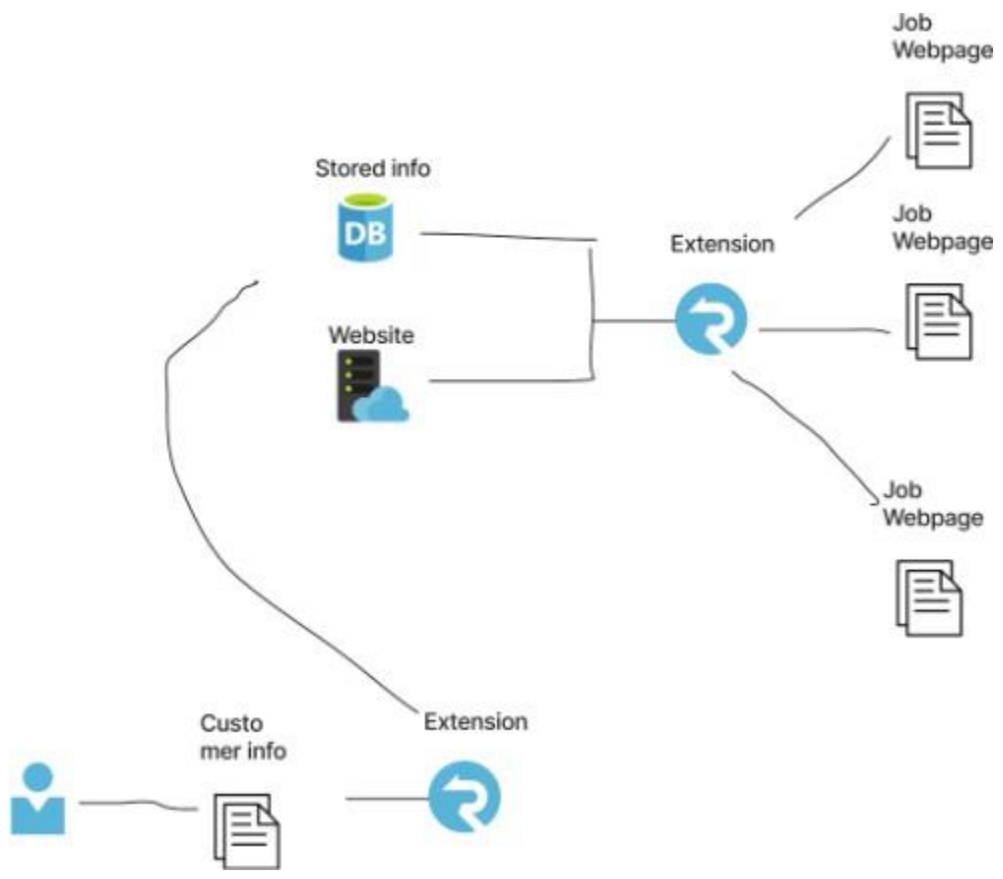
Architectural Goals and Principles

The key goal of this architecture is to have a product that works with users ambitions and stays responsive. Our website should load info into the extension and work well. Ease of use is also one of our goals. There is competition in our field and we want to make our product as easy to use as we can so that way myself and others can use this tool.

Goals

Scalability: The system is architected to handle growth in user numbers and data volume seamlessly, ensuring the platform remains responsive as demand increases.
Security: Security is paramount, with Clerk handling authentication to protect user data and ensure privacy.
Ease of Integration: Designed for easy integration with third-party services and APIs, enhancing the platform's capabilities without compromising performance or security.
Performance: The architecture ensures that the platform operates efficiently, minimizing load times and optimizing the user experience.
Reliability and Availability: The system aims for high availability and reliability, ensuring users have consistent access to the platform's features.

System Overview



Architectural Patterns

We are going to use a microservices model. For different functionalities of our product because it would specifically allow us to use small parts in building our entire project. Another thing would be that if one bit goes down. Our entire project just doesn't collapse.

○ ***Decoupling****: Microservices allow for the decoupling of services, ensuring that changes in one service do not impact others. This is crucial for the browser extension as it will need to integrate with ChatGPT's API and potentially other services in the future.*

○ ***Scalability****: As mentioned in the project goals, scalability is a priority. Microservices can be scaled independently, allowing for efficient resource utilization.*

○ ***Flexibility****: Different microservices can be developed using different technologies best suited for their specific tasks.*

○ ***Maintenance and Updates**: With microservices, individual components can be updated without affecting the entire system, ensuring continuous delivery and integration.*

Component Descriptions

● ***User Interface**: Deals with user entering info to database part of the product deals with getting info to the database.*
● ***Database**: Stores and manages data using a relational database system.*

Data Management

1. User
    a. Represents individual users of the database
    b. Attributes:
        i. Username: A unique identifier for the user
        ii. Email: The email address associated with the user
        iii. Preferences: A JSON structure storing user-specific preferences for the extension, such as autofill settings this will be quite extensive
        iv. Answers to some questions we deem to be the same across companies also a 500 to 700 word about me section that ChatGPT will use to extrapolate responses at times.
2. Form
    a. Represents the online forms in which users interact with
    b. Attributes:
        i. formID: Unique identifier for each form
        ii. websiteURL: The URL of the website where the form is located
        iii. Fields: A JSON structure detailing the various fields in the form, such as text boxes, dropdowns, etc.
        iv. Timestamp: The date and time when the autofill action was performed
3. We have an API that integrates with OpenAI API
4. Our CRUD operations from the website are minimal and communicate with extension via an API

Interface Definitions

1. **User Management**:
    - **GET /tasks**: returns a list of attributes about user
    - **POST /updates:** database
    - **POST /deletes**: from database
    - **GET/ Gets:** metadata about the users
2. **Form Management**:
    - **GET /forms/{formID}**: Retrieves the details of a specific form based on the provided formID.
    - **POST /forms**: Adds a new form to the system, capturing its fields and associated details.
    - **PUT /forms/{formID}**: Updates the details or fields of a specific form.
    - **DELETE /forms/{formID}**: Removes a specific form from the system.
3. **Autofill History**:
    - **GET /history/{userID}**: Returns the autofill history for a specific user.
    - **POST /history**: Logs a new autofill action performed by a user on a form.
    - **DELETE /history/{historyID}**: Deletes a specific autofill action record.
4. **ChatGPT Integration**:
    - **GET /ChatGPT/suggestions**: Retrieves intelligent suggestions from ChatGPT for a specific form field.
    - **POST /ChatGPT/feedback**: Sends user feedback or corrections to ChatGPT to improve future suggestions.

Sample API Call Descriptions:

1. **GET /users/{userID}**:
    - **Purpose**: To fetch the details and preferences of a specific user.
    - **Parameters**:
        - **userID**: The unique identifier of the user.
    - **Returns**: User details including username, email, and preferences.
2. **POST /forms**:
    - **Purpose**: To add a new form to the system.
    - **Body**: Contains details of the form such as websiteURL and fields.
    - **Returns**: Confirmation of the form addition along with the formID.
3. **GET /ChatGPT/suggestions**:
    - **Purpose**: To get AI-based suggestions for a form field.
    - **Parameters**:
        - **fieldType**: The type of form field (e.g., name, phone number).
    - **Returns**: Intelligent suggestions based on the fieldType.

Considerations

Security

Data Interception:
○ Risk: Unauthorized entities might intercept and access sensitive user data during transmission. The database needs to be protected/encrypted.

Mitigation:
SSL/TLS: Ensure that all data transmitted between the browser extension, backend servers, and ChatGPT's API is encrypted using SSL/TLS protocols. This ensures that data in transit is secure and unreadable to potential eavesdroppers.

Performance

We need to make sure our app keeps the speed when using the API. Or failing that we need to figure out how to make the delay acceptable to the user.

1. **Scalability:**
    a. **Requirement:** As the user base grows, the system should be able to handle an increasing number of requests without compromising performance.
        i. **Horizontal Scaling**: Deploy the backend services in containers, allowing for easy scaling by adding more instances as the demand grows. Container orchestration tools like Kubernetes can be used to manage and scale these containers efficiently.
        ii. **Microservices Architecture**: By designing the system using a microservices architecture, individual components can be scaled independently based on their specific demands.

Deployment Strategy

We will have a CI/CD strategy for rolling out content. This strategy will allow our app to be updated in real time and keep.

Testing Strategy

The microservices will be tested individually and we will write automated tests.

Unit Tests: We create tests for each individual item so like if we need to fill out the users First Name, Last Name, and Email using only the AI. Then we will have unit tests to make sure those outputs are what we want.

Integration test: Make sure that our AI and Database work.

Glossary

● ***API** (Application Programming Interface): A set of rules and protocols that allows different software entities to communicate with each other.*

● ***CRUD** (Create, Read, Update, Delete): Basic functions of persistent storage, often used to describe the basic operations of databases.*

● ***CI/CD** (Continuous Integration/Continuous Deployment): A software engineering practice where code changes are automatically tested and deployed to production.*

● ***E2E** (End-to-End): Testing the flow of an application as a whole to ensure the entire process of inputs and outputs works smoothly.*

● ***UI** (User Interface): The space where interactions between humans and machines occur, in this context, the visual layout of the extension.*

● ***UX** (User Experience): Refers to the overall experience a user has when interacting with a product or service.*