Software Requirements Specification (SRS)

FounderMatching: A Full-Stack Web Application for Forming Startups

Version: 1.0.0 approved

Prepared by:

Truong Dang Gia Huy

Luu Nguyen Chi Duc

Cao Lam Huy

Nguyen Nhat Nam

VINUNIVERSITY - Entrepreneurship Lab

January, 2025

Table of Contents

I. Abstract

This project report discusses the development of an innovative, tailor-made job-matching platform for the entrepreneurial ecosystem at VinUniversity and its potential in creating impact across Vietnam. Combining modern design and algorithmic features inspired by the likes of Tinder, LinkedIn, and YCombinator, this solution addresses the critical gaps of inefficient networking, manual sorting processes, and a lack of centralized platforms. The team developed a structured process for Agile development, using cutting-edge technologies and tools—in this case, Next.js for the frontend and Tailwind CSS with Django REST Framework in the backend. Given the timeline, this project delivered a working prototype with scalable potential and outlined the future enhancements needed to ensure robust matching and networking.

II. Introduction

Founder Matching Project is a platform designed for tackling the pain point of team formation in startups. For the past few years, it is undoubted that the number of startup projects is much larger than the number of individuals willing to find a startup. There are many reasons for this, mainly come from the limitation of existing communication platforms, where the connection bridge supposed to be. The reality also shows that traffic to the Entrepreneurship Website of E-lab is very low. Then, our team was inspired by the novel idea from them, creating a Founder Matching platform. By utilizing the catchy UI of Tinder and robust matching algorithms from LinkedIn, this can provide a user-friendly interface for people to highlight their startups so that potential candidates can be connected and become co-founders. Also, it is a place for individuals to showcase to present their skillful profiles and connect with their suitable startups. So, we are working on this project to:

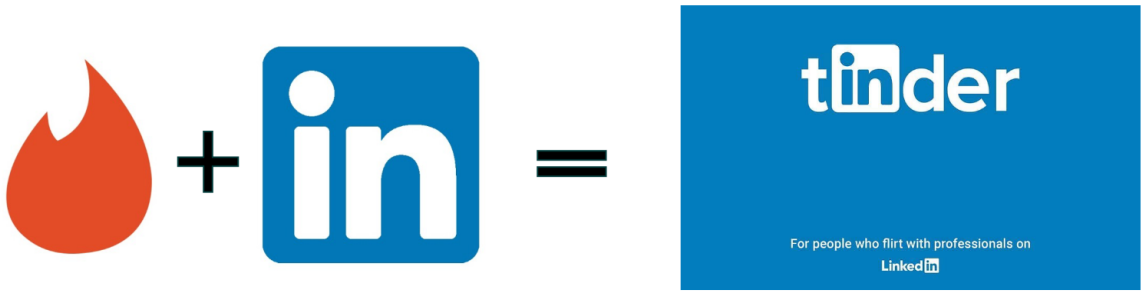
- Resolve limitations of traditional means of communication for gathering potential startup cofounders together.
- Encourage connecting and networking within the Entrepreneur community, starting from university first
- Automate the process of manually recruiting and seeking for jobs.

VinUniversity are strongly developing a thriving entrepreneurial ecosystem that can extende its influence beyond campus to the broader entrepreneurial landscape in Vietnam. However, the challenge of effectively matching startups with suitable team members is a critical gap that still persists within this dynamic environment. Current networking processes are often inefficient, lacking a centralized platform to streamline connections between aspiring entrepreneurs and potential collaborators. Moreover, it is also a time-consuming experience where manual sorting processes frequently overlook promising matches, possibly leading to overlook collaborative opportunities within the ecosystem. Therefore, addressing this issue is essential to broadening the potential of VinUniversity’s entrepreneurial community and its contributions to Vietnam’s innovation economy.

The proposed solution draws inspiration from market leaders like Tinder, LinkedIn, and YCombinator, integrating friendly user interfaces, dynamic profile features, and advanced algorithms for matching and recommendations. The site assures an engaging professional experience through swiping mechanics, profile customization, and motivational features such as badges and notifications which is believed to attract many users from different careers. Additionally, it incorporates PageRank and keyword filtering matching algorithms to increase the accuracy of the matchmaking process between a startup and a potential team member.

Although certain advanced features are not implemented due to time constraints, this report will focus on the crucial elements of the platform, its development process, and the possibilities of enhancement in the future.

- Solution Overview:



- Contribution of each team members:
  - **Truong Dang Gia Huy:** Design and setup database architecture, implement backend and database.
  - **Luu Nguyen Chi Duc**: Develop and implement UI components such as buttons, input fields, and construct mock databases.
  - **Cao Lam Huy:** Implement UI for Sign up - Log in and authentication feature.
  - **Nguyen Dai Nghia:** Project management and collaboration with backend developers to ensure seamless frontend-backend communication.

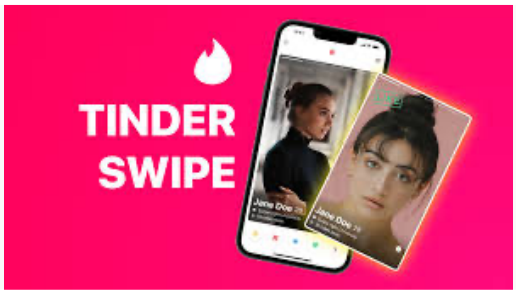
III. Literature Review

Development of job-portal web platforms needs deep insight into user interaction and experience. By integrating modern design principles and functionalities inspired by successful applications like Tinder, LinkedIn, and YCombinator, we strive to provide innovation in this platform. This literature review explores the referenced attributes, their application, and possible algorithms for enhancement of user engagement and efficiency of matching.

1. **Tinder swiping mechanics**

Tinder pioneered the art of users interacting through a very intuitive swipe mechanism. In addition, such a feature simplifies decision-making by creating binary actions to swipe right to accept or swipe left to decline. Our job portal replicates this swipe on the discover page for rapid and effective browsing of job opportunities. Key takeaways from the UX/UI design used in Tinder are as follows:

- **Simplicity of Design**: Clean, minimalistic interfaces that focus user attention on the primary action.
- **Gamification Elements**: Swiping introduces an engaging, game-like interaction.
- **Responsive Design**: Ensures fluid interactions across devices, enhancing accessibility.

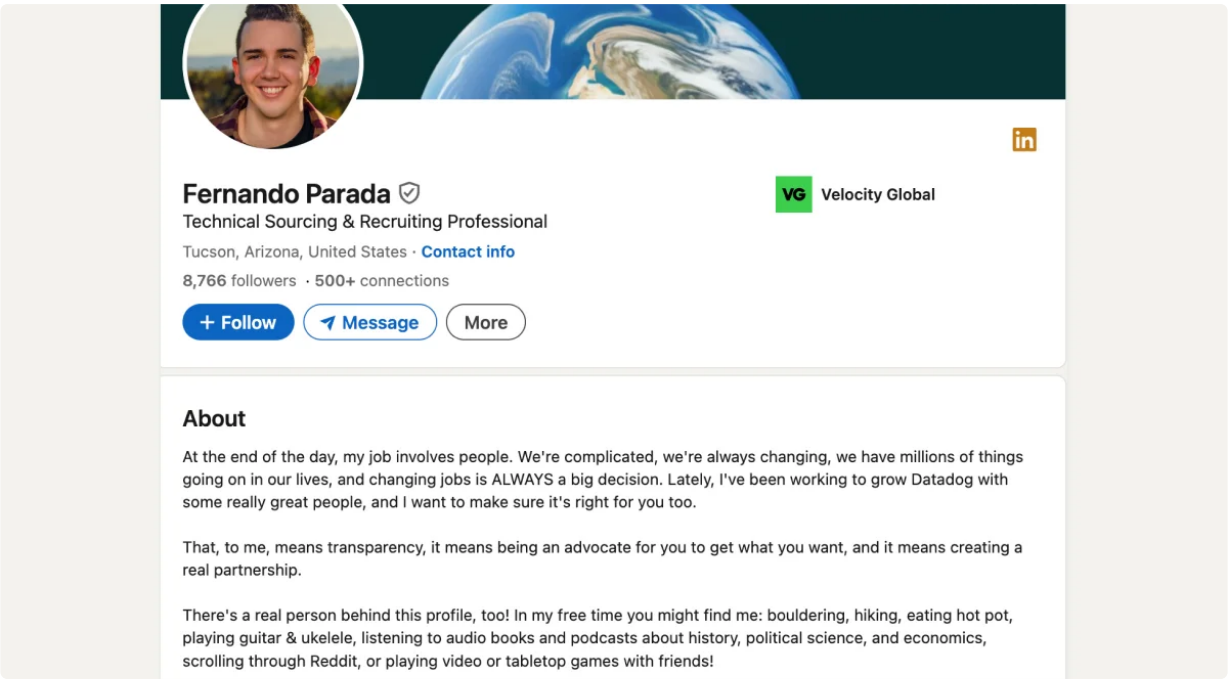


By adopting these principles, we aim to make job discovery intuitive and enjoyable while maintaining professional integrity.

2. **Influence of LinkedIn**

The LinkedIn profile page has become the standard for professional networking sites, offering all the features to showcase the career achievements of a user. Our portal includes the following elements:

- **Profile Completeness Metrics**: This encourages users to add experiences, certifications, and skills.
- **Dynamic Sections**: This allows customization and prioritization of content, such as pinning key achievements.
- **Visual Appeal**: Symmetrical use of visuals and text for readability and professionalism.



These features promote user authenticity and make profiles more attractive to recruiters, thereby enhancing the overall user experience.

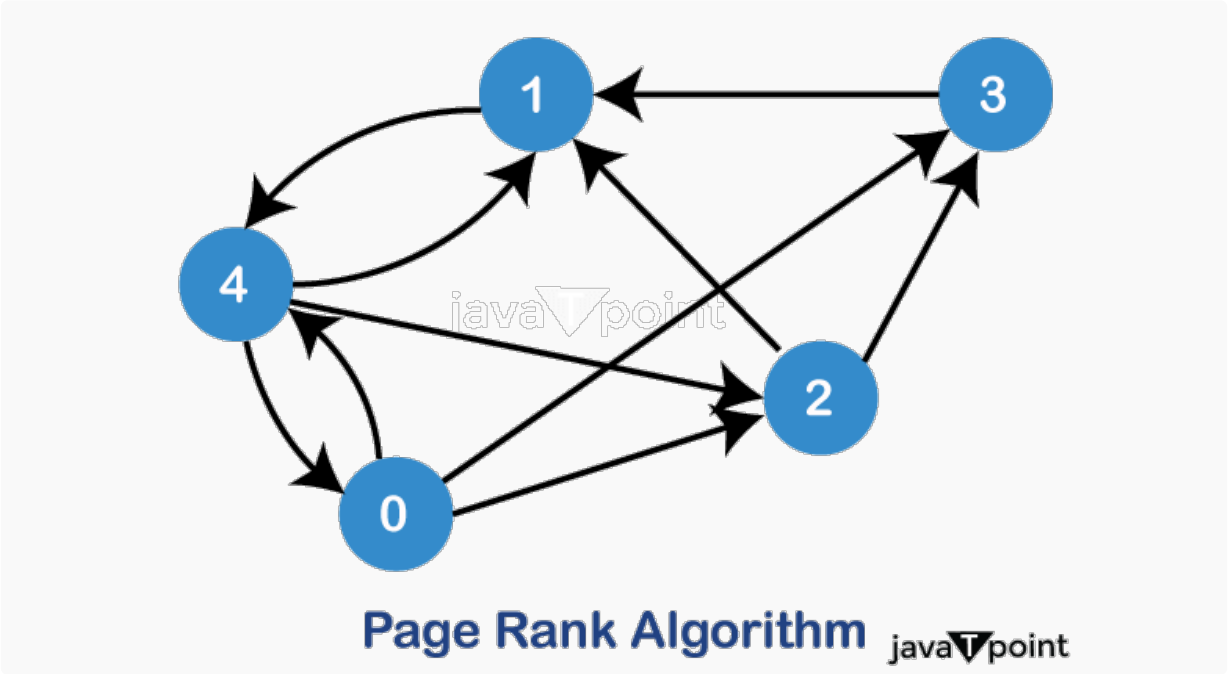
3. **Learning from YCombinator**

YCombinator's platform is very good at creating connections and motivating users. Our design borrows a number of its strategies:

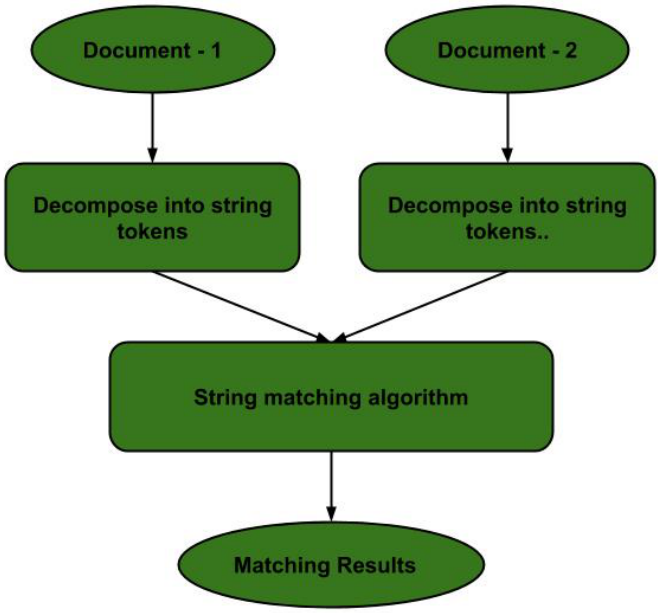
- **Discover Features**: Encourage the user to browse and revisit profiles or job postings through well-organized categories and recommendation prompts.
- **Motivational Design**: The use of badges, notifications, and success stories will motivate the user to take action.
- **Networking Opportunities**: Provide opportunities for connecting to like-minded professionals through suggestions of groups or events.

In the end, we embed these features to create a very interactive environment that keeps the user always active on the platform.

4. **Matching Algorithm**
    - Page Rank algorithm : The PageRank algorithm, originally used by Google for ranking web pages, can be adapted to job portals for ranking and matching users and jobs. Each user or job listing is treated as a node in a graph, with connections based on relevance, endorsements, or shared attributes. The algorithm assigns a score to each node, representing its relevance and popularity within the network. (https://www.geeksforgeeks.org/page-rank-algorithm-implementation/)



- Keyword Matching algorithm: Keyword matching relies on the matching of the job description and the profile of the individual on basic key terms that involve skills, qualifications, or even job titles. It uses techniques like string matching and keyword extraction to ensure relevance.
  - **Application**: A candidate listing "Python programming" as a skill will be matched with job descriptions that explicitly require "Python".
  - **Advantages**: Simple implementation and real-time matching capabilities.



(https://www.geeksforgeeks.org/keyword-searching-algorithms-for-search-engines/)

Due to the time and workload constraint, there are still some features and algorithm we would like to implement in our website but we are not able to. In the future, our team will work harder to fully complete.

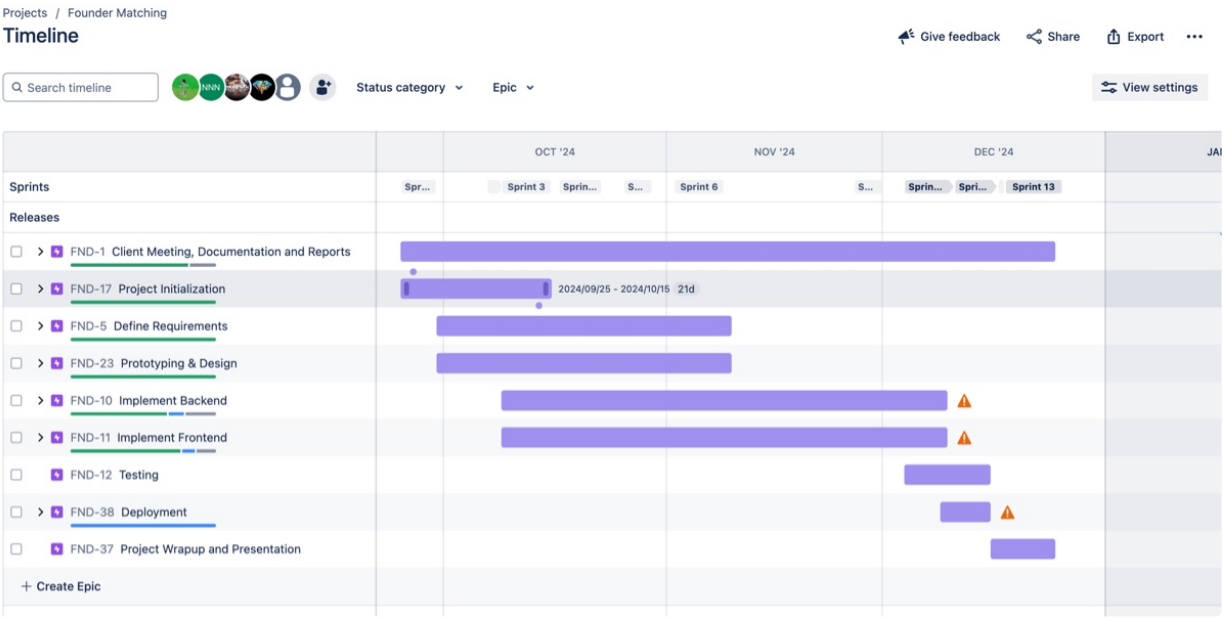
IV. Methodology

1. Approach and Project Management

Our team software project use **Agile methodology** for iterative progress, active collaboration and adaptability. By doing this way, we can make sure that our team focus on delivering value incrementally, thus allowing for regular feedback and adjustments throughout the project's life cycle.

You can see from our timeline that our project is structured around multiple sprints. Here is a breakdown of the project management using agile principles:

- **Sprint Organization and Release Planning**: The timeline below highlights the division of the project into clearly defined stages aligned with the Agile framework, including:
  - **Project Initialization (Sprint 1)**: This phase involved initial client meetings, documentation, and goal-setting, ensuring alignment between stakeholders and us.
  - **Requirement Definition (Sprints 2-3)**: Detailed requirements were gathered and analyzed to conduct subsequent design and implementation.
  - **Prototyping and Design (Sprints 4-5)**: We worked iteratively to design and validate prototypes days by days and client feedback early to ensure the solution met expectations.
  - **Implement Frontend and Backend (Sprint 6-10):** Developed core functions and corresponding UI.
  - **Testing (Sprint 11):** Conduct unit testing and system testing.
- **Incremental Process:** Deliver small pieces that represent functional pieces of the product. Break core tasks that needed development-Backend and frontend implementation are segmented into a form to simultaneously make progress across many features in parallel fashion.
- **Collaboration and Communication**: Agile ceremonies, such as daily stand-ups, sprint reviews, and retrospectives, have been fundamental in maintaining transparency and alignment within the team. Besides that, client engagement was integrated at each phase, fostering collaboration and shared ownership of the project's outcomes.



2. Tools

To ensure the efficient execution of tasks and maintain high-quality deliverables, our team learned to use these tools:

• **Figma**: Used for designing and refining prototypes, enabling effective collaboration on user interface and experience design.

• **GitHub**: Facilitated version control and collaborative coding. The project's repository is available at GitHub - FounderMatching.

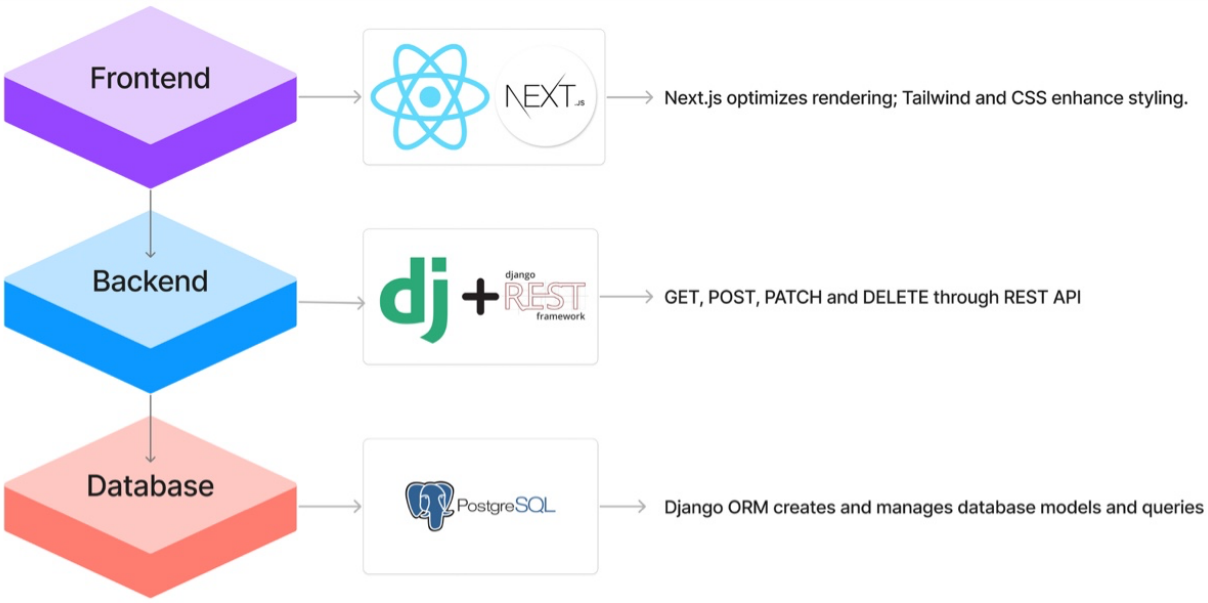
• **dbdiagram.io**: Utilized for creating and visualizing the database schema, ensuring a well-structured and scalable database design.

• **Swagger**: Leveraged for API design and documentation, ensuring consistency and clarity in backend service development.

• **Postman**: Used for API testing and validation, enabling the team to identify and resolve issues early in the development process.

3. Technologies

- Our techstack includes:



Frontend

- **Next.js:** Enabled server-side rendering (SSR) and static site generation (SSG), ensuring fast and scalable web applications with optimized performance.
- **Tailwind CSS:** Employed for its utility-first approach, allowing rapid and consistent styling.
- **Custom CSS:** Used alongside Tailwind for implementing complex designs and animations that were beyond Tailwind's direct capabilities.
- **Clerk:** Integrated for secure user authentication, supporting third-party login providers like Google, LinkedIn and Facebook.
- **Framer Motion:** Utilized for dynamic animations, such as smooth card transitions and hover effects.

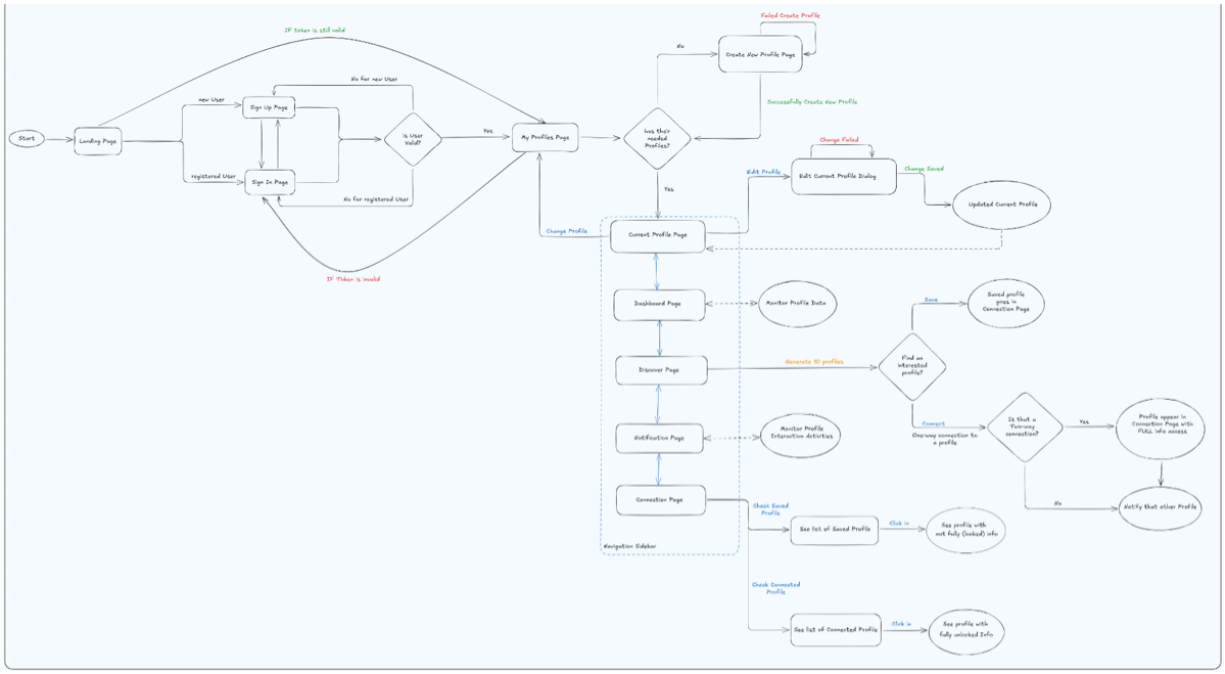
Backend

- **Django REST Framework:** Used for building RESTful APIs, serializers for data transformation and multiple parser support
- **PostgreSQL:** Primary database system for string data, a perfect combination with Django
- **Redis:** For caching and session management
- **JWT Authentication:** Custom implementation for secure user authentication and authorization

User Flow

Below is an example use case of User Management:

The user flow for the platform was meticulously designed to ensure a seamless, intuitive, and logical navigation experience for users. The flow emphasizes efficient task completion, user engagement, and clarity at each interaction point. This is an explanation of the key components of the design



1. Start and Landing Page

- The user begins at the **Landing Page**, the gateway to the app.
- Based on their status:
  - **New Users** are directed to the **Sign-Up Page** to create an account.
  - **Registered Users** are directed to the **Sign-In Page** to log in.
- If the user's session token is invalid or expired, they are prompted to reauthenticate.

2. Authentication

- **Sign-Up Page**:
  - Designed for new users to register by providing essential details.
  - Supports third-party authentication (Google, Facebook) for simplified account creation.
  - Includes real-time validation and error feedback to guide users during the registration process.
- **Sign-In Page**:
  - Verifies the credentials of registered users through token-based authentication.
  - Redirects successfully authenticated users to their **My Profiles Page**.
  - Provides error messages for invalid credentials or token issues.

3. My Profiles Page

- Acts as the user's central hub for managing profiles.
- Depending on the user's status:
  - **New Users**: Redirected to the **Create New Profile Page**.
  - **Returning Users**: Can view or edit their existing profile using the **Edit Profile Dialog**.

4. Profile Management

- **Create New Profile Page**:
  - Allows users to input and save profile details.
  - Validates user input and provides immediate feedback for errors.
  - Ensures a successful profile creation or highlights areas to fix for failed attempts.
- **Edit Profile Dialog**:
  - Enables users to modify and update their existing profile details dynamically.
  - Incorporates validation to ensure changes are applied accurately.

5. Main App Navigation

Once authenticated and their profile is set, users access the main app via the **Navigation Sidebar**, which connects them to:

  - **Dashboard Page**:
    - Displays key metrics such as profile views, interactions, and activity logs.
    - Allows users to monitor their performance and engagement on the platform.
  - **Discover Page**:

        - Features a Tinder-like swiping mechanism to explore profiles dynamically generated based on preferences.
        - Users can save profiles of interest or send connection requests.
    - **Notifications Page**:
        - Provides updates on profile activities and connection requests.
        - Keeps users informed about critical actions and interactions.
    - **Connection Page**:
        - Lists saved and connected profiles.
        - Saved profiles show partial information, while connected profiles unlock full details.
        - Users can view profiles, manage their connections, or initiate new interactions.

6. Connection Management

- Users interact with profiles on the **Discover Page**, deciding to:
    - **Save Profiles**: Stored for future interaction with limited information access.
    - **Connect**: Initiates a connection request.
- **Mutual Connections**:
    - If both parties connect, the profile moves to the connected list with full details accessible.
- **One-Way Connections**:
    - Notifications are sent to the other profile, encouraging further interaction.

7. Error Handling and Validation

- Throughout the flow, robust error handling ensures smooth navigation:
    - Invalid tokens prompt users to reauthenticate on the Sign-In Page.
    - Failed profile creation or updates provide immediate feedback for corrections.
    - Connection errors notify users and suggest alternative actions.

8. Endpoints and Desired Actions

The user flow leads to meaningful endpoints, ensuring users achieve their goals:

- Successfully creating or updating profiles.
- Monitoring profile performance and activity via the dashboard.
- Exploring and saving profiles of interest.
- Building meaningful connections with other users.
- Staying informed through notifications and managing relationships on the platform.

Testing and Feedback

- Postman and OpenAPI: Used for testing API endpoints to validate secure and reliable data transfer.

- User Testing: Early feedback identified issues with button placements and layouts, leading to iterative refinements.
- Responsiveness: Tailwind Debug Screens were employed to ensure a consistent experience across devices.
- Issue Management: Issues identified during testing were logged on GitHub or directly assigned to team members for resolution, ensuring accountability and efficient problem-solving.

V. Implementation

1. Frontend

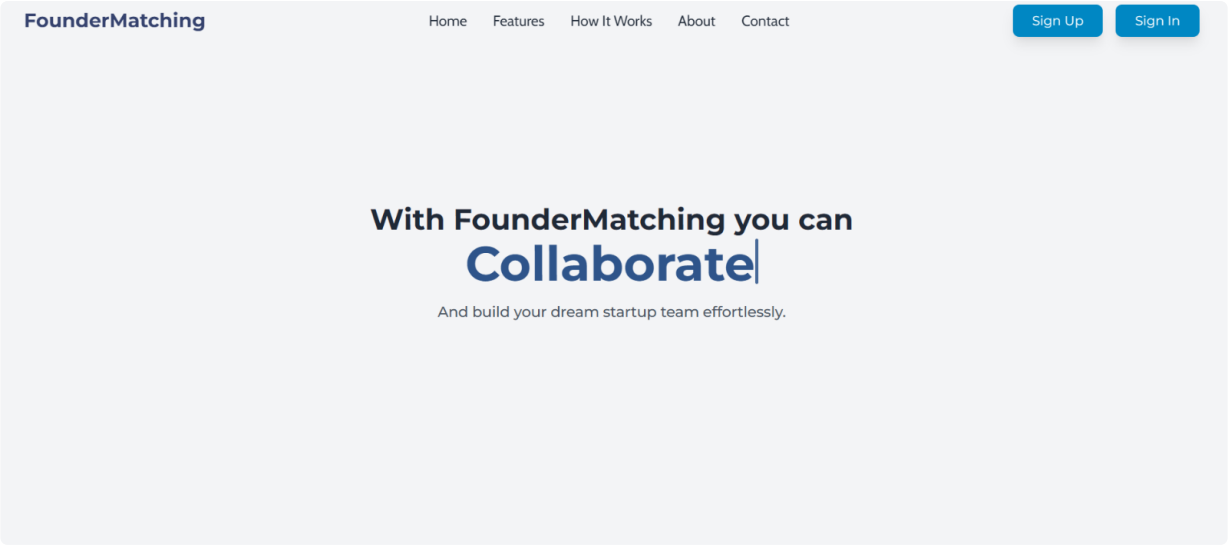
The frontend development methodology was centered around leveraging modern web technologies, ensuring a robust, user-friendly, and scalable platform. The primary focus areas included user experience design, performance optimization, and security.

Component-Based Architecture: Organized into directories to ensure scalability and maintainability

- **src/app:** Main pages, including key functional views such as login, signup, and dashboard, with additional pages added as needed for platform scalability and user interaction.
- **src/ui:** Core reusable components like buttons, forms, and inputs.
- **src/components/layout:** Custom layout components such as headers and sidebars.
- **src/lib:** Implements and performs CRUD requests to the backend endpoints, serving as the bridge for data operations.
- **src/middleware:** Protects routes from unauthorized users by validating authentication tokens and redirecting unauthenticated requests.

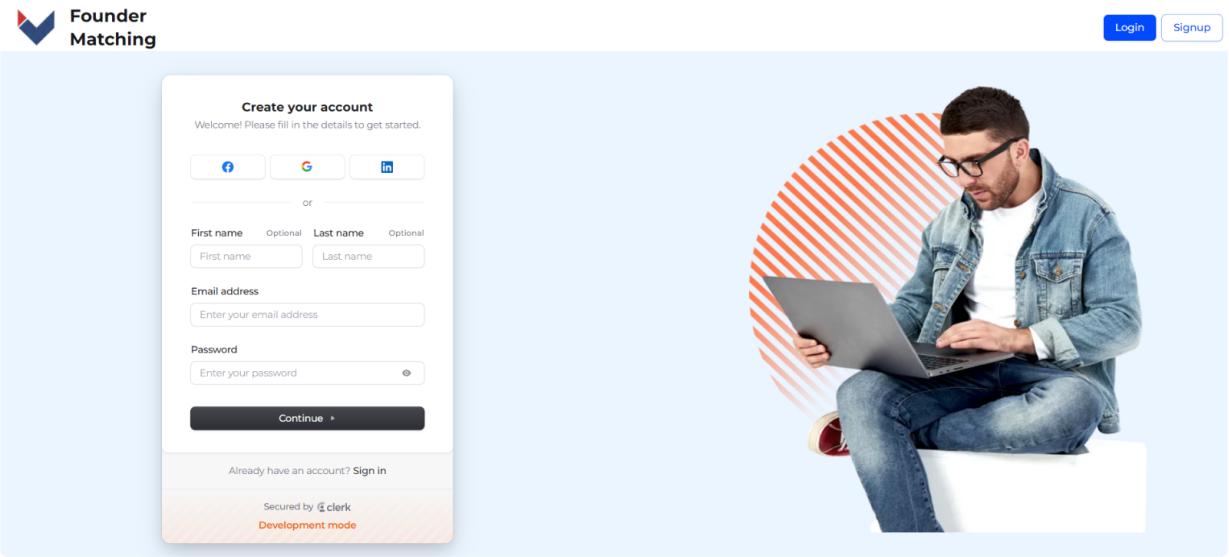
Frontend implementation followed a systematic approach, focusing on delivering functional and visually engaging components. A detailed user flow was created to map out the app's structure and ensure logical progression for users. This flow guided the development of each functional page, ensuring a seamless user experience. The key functional pages include:

Landing Page



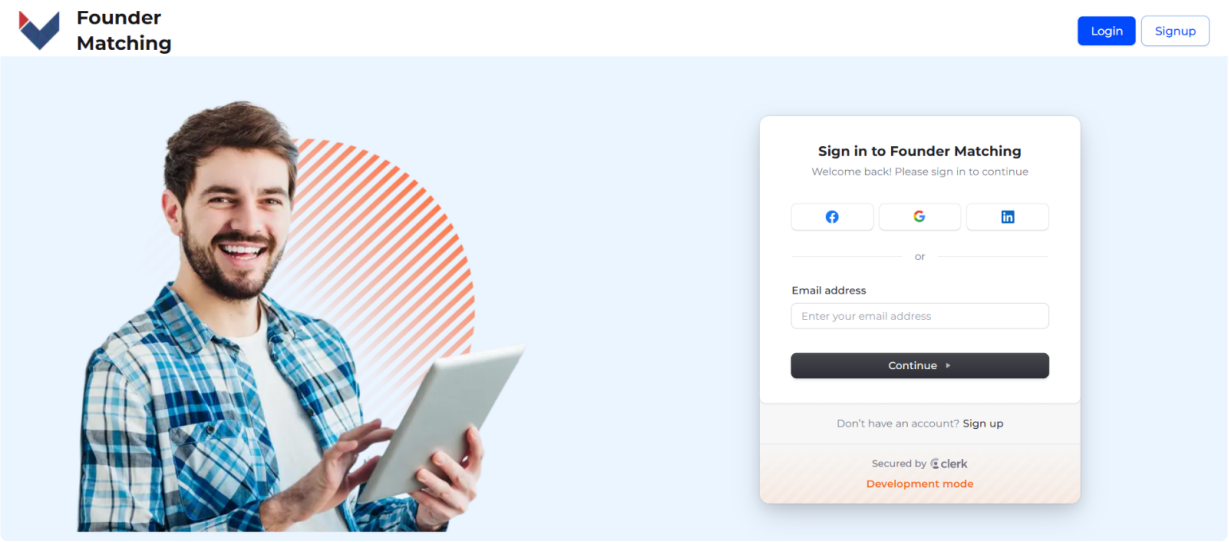
- The entry point of the app where users decide to either sign in or sign up.
- Provides links to authentication flows based on user type (new or returning).

Sign-Up Page



- Designed for new users to register an account.
- Includes fields for essential user information and integrates third-party authentication through Google and Facebook.
- Validates input and provides feedback for errors in real-time.

Sign-In Page



- Allows registered users to log in to the platform securely.
- Implements token-based authentication and redirects users to their profile or dashboard upon successful login.
- Handles invalid or expired tokens with appropriate error messages.

My Profiles Page

- Displays the user's profile management options.
- For new users, this page provides an option to create a profile. For existing users, it allows viewing or editing their current profiles.

Create New Profile Page

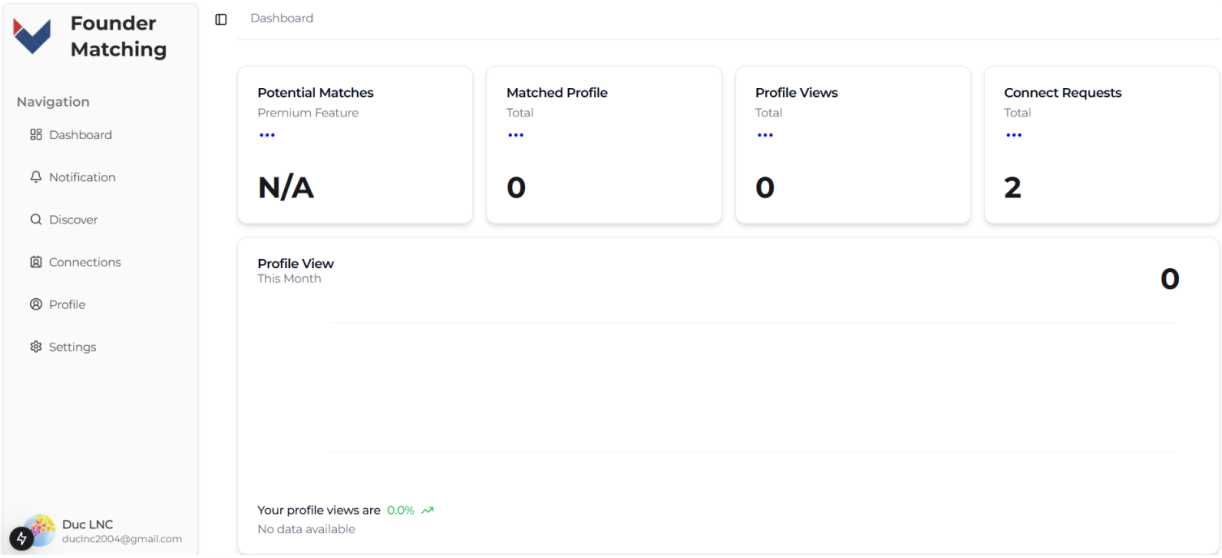
- Enables users to input details for a new profile.
- Validates inputs and ensures the profile is created successfully or provides error messages if creation fails.

Profile Page

- Page where users can view their profile's information
- A pop-up dialog that allows users to modify their existing profile information.

- Changes are saved dynamically and validated before updates are finalized.

Dashboard Page

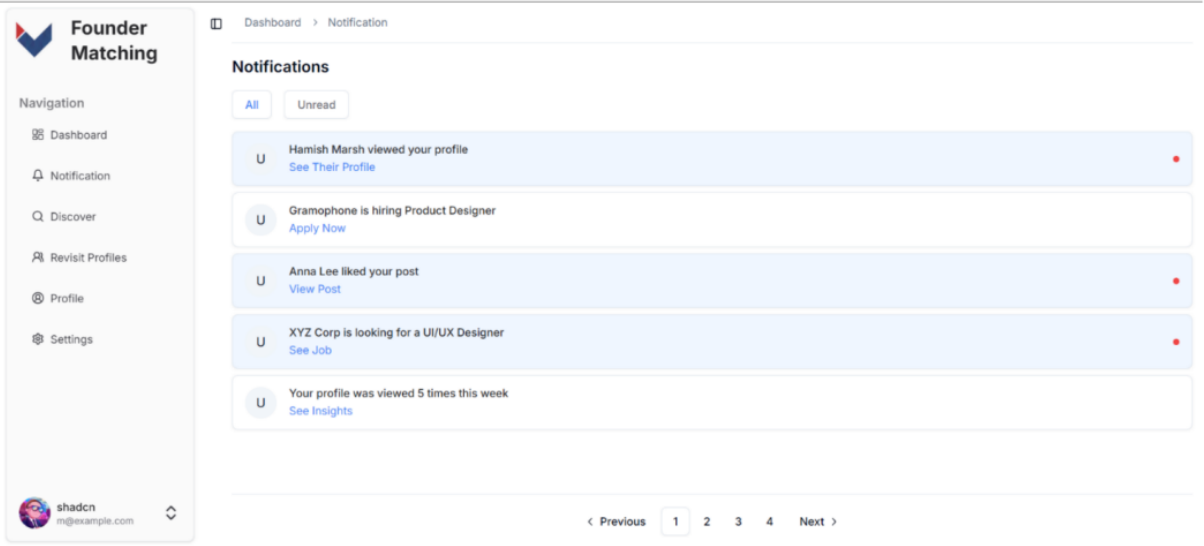


- Centralized hub for users to monitor their profile data, including interactions, views, and saved profiles.
- Includes dynamic components to display metrics and activity logs.

Discover Page

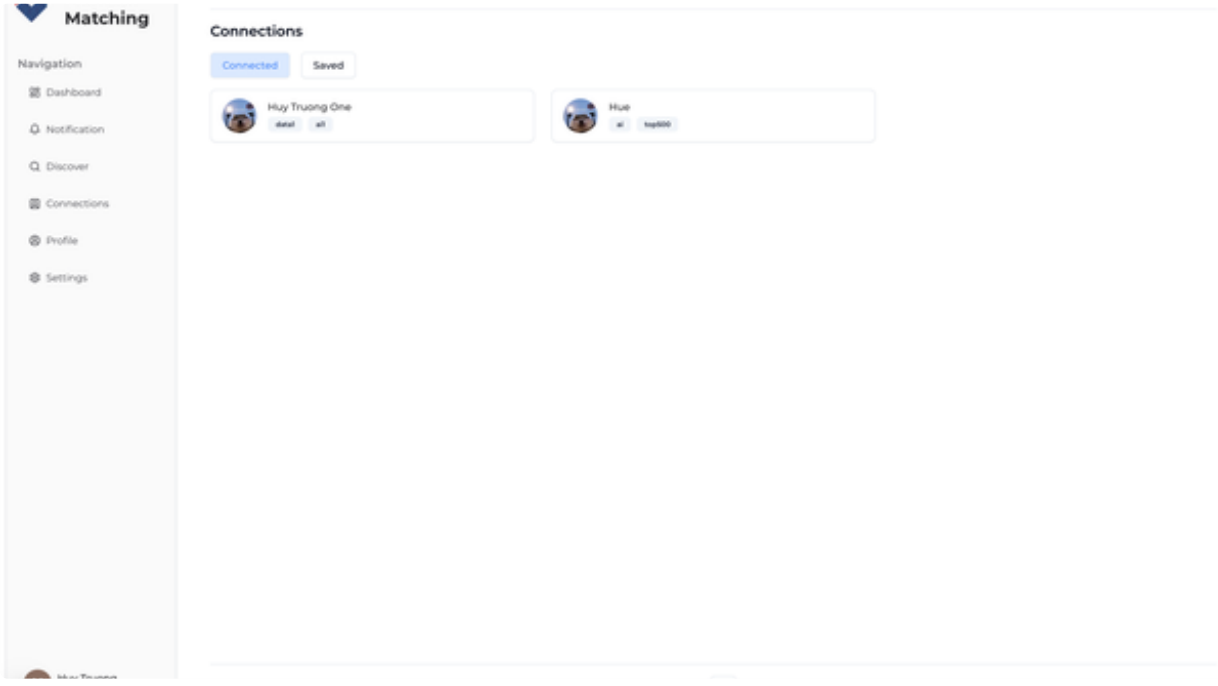
- Features a Tinder-like swiping mechanism for users to explore profiles.
- Dynamically generates a set of 10 profiles for users to interact with.
- Users can save profiles or initiate a connection.

Notifications Page



- Displays alerts for profile interactions, such as new connections or activity updates.
- Allows users to stay informed about important events related to their profiles.

Connection Page



- Shows a list of saved and connected profiles.
- For saved profiles, users can see partial information. For connected profiles, full details are unlocked.
- Handles one-way and mutual connections, notifying users about connection status updates.

2. Backend

- Our systeam is a full-stack web application with separated frontend and backend services. Each app is self-contained with its own models, views, and URLs.

```
backend/
  ├── accounts/     # User authentication
  ├── api/          # Core API endpoints
  ├── dashboard/    # Dashboard features
  ├── discover/     # Discovery functionality
  ├── profiles/     # User profiles
  └── revisit/      # Revisit functionality
```

a. RESTful API architecture

- We implement RESTful principles for client-server communication through Django REST Framework
- API Endpoints:

  Profile Management Endpoints:

  `GET /getUserProfiles`

  - Description: Retrieves all profiles belonging to the authenticated user
  - Flow:
    1. Validates JWT authentication token
    2. Retrieves user account from clerk ID
    3. Fetches all profiles linked to user with associated tags
    4. Returns profiles in preview card format
  - Error Handling:
    - User not found in system
    - No profiles exist for user
    - Authentication token invalid/expired

  `GET /profile/me`

  - Description: Gets detailed information about a specific user profile
  - Flow:
    1. Validates authentication and profile ID
    2. Checks profile ownership
    3. Retrieves full profile data including experiences, certificates, achievements
    4. Returns complete profile with all information
  - Error Handling:
    - Profile ID not provided
    - Profile not found
    - Unauthorized access attempt

  `PATCH /profile/me`

- Description: Updates an existing profile's information
- Flow:
  1. Validates user authentication and profile ownership
  2. Processes avatar if provided (validates size/format)
  3. Updates only provided fields
  4. Returns updated profile data
- Error Handling:
  - Invalid file formats
  - File size exceeds limits
  - Invalid field values
  - Profile not found

`POST /profile/create`

- Description: Creates a new profile for the authenticated user
- Flow:
  1. Validates user authentication
  2. Processes profile information and avatar
  3. Creates profile with privacy settings
  4. Returns new profile data
- Error Handling:
  - Invalid profile data
  - Avatar validation failures
  - Database constraints violations

Connection Management Endpoints:

`POST /connect`

- Description: Initiates or accepts a connection between profiles
- Flow:
  1. Validates both profile IDs
  2. Checks if connection already exists
  3. Creates/updates connection based on profile types (startup/candidate)
  4. Handles matching logic when both parties accept
- Error Handling:
  - Already connected
  - Pending request exists
  - Invalid profile combination
  - Profile not found

`GET /discover`

- Description: Returns potential profiles for connection
- Flow:

1. Validates requesting profile
2. Filters out connected/rejected/own profiles
3. Applies privacy settings to visible fields
4. Returns paginated results

- Error Handling:
  - Invalid pagination parameters
  - No profiles available
  - Unauthorized access

`GET /getConnections`

- Description: Lists all connected profiles
- Flow:
  1. Validates profile ownership
  2. Retrieves matched connections
  3. Applies profile type filtering (startup/candidate)
  4. Returns paginated connection list
- Error Handling:
  - Profile not found
  - Invalid profile type
  - Pagination errors

Analytics Endpoints:

`POST /countView`

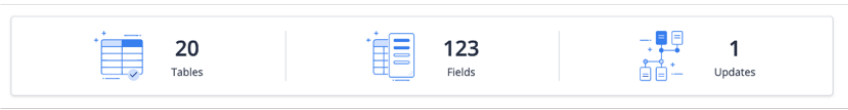
- Description: Records when a profile views another profile
- Flow:
  1. Validates viewer authentication
  2. Verifies both profiles exist
  3. Records view with timestamp
  4. Updates view statistics
- Error Handling:
  - Invalid profile IDs
  - Unauthorized viewing attempt
  - Database recording failures

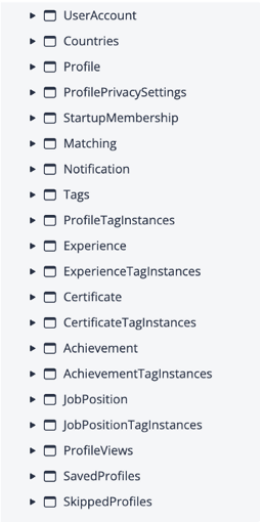
`GET /dashboard`

- Description: Retrieves analytics and statistics for a profile
- Flow:
  1. Validates profile access
  2. Gathers:
     - Total profile views
     - 30-day view history

        - Connection request count
        - Matched profile count
        - Recent viewer profiles
    3. Compiles dashboard data
- Error Handling:
    - Profile access denied
    - Data aggregation failures
    - Invalid date ranges

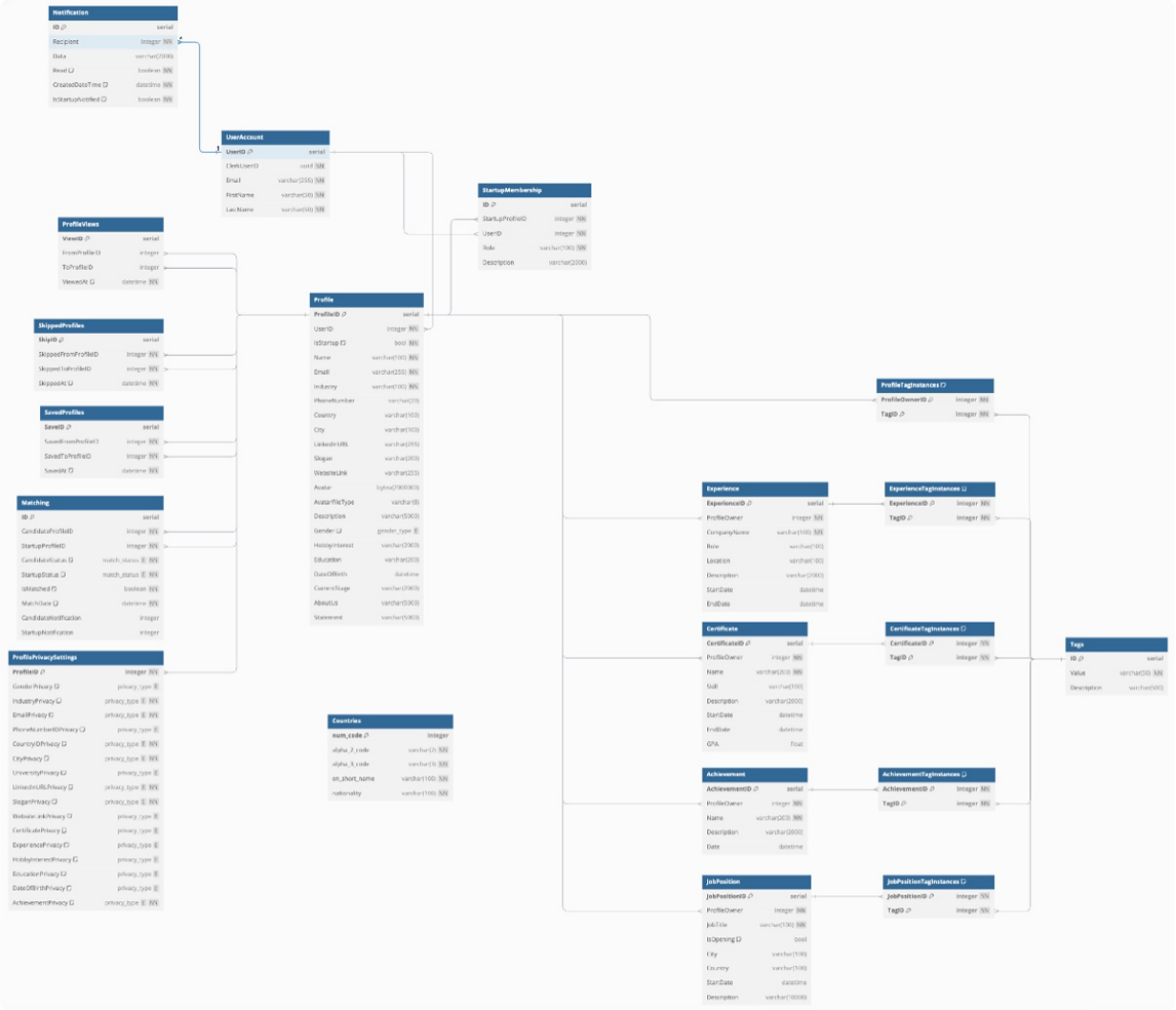
b. Database





- Core Models:
    - UserAccount
        - Primary user information (Clerk integration)
        - Basic profile data (email, name)
    - Profile
        - Comprehensive user profile information
        - Supports both individual and startup profiles
        - Fields include:
            - Basic info (name, email, industry)
            - Location data (country, city)
            - Professional info (LinkedIn, website)
            - Personal details (gender, DOB)
            - Startup-specific fields (current stage, about us)
    - Experience
        - Professional experience records
        - Company details, role, duration

        - Location and description
    - Certificate
        - Educational and professional certifications
        - Skills and qualifications
        - Duration and GPA if applicable
    - Achievement
        - Personal and professional achievements
        - Dates and descriptions
    - JobPosition
        - Job openings and positions
        - Location, requirements
        - Status tracking (open/closed)
    - Connection
        - Network connections between profiles
        - Status tracking (pending/accepted/rejected)
        - Timestamp information
- Supporting Models:
    - Tags: Skill and interest categorization
    - ProfilePrivacySettings: Granular privacy controls
    - Countries: Reference data for locations



c. Security & Authentication

- JWT (PyJWT) for token-based authentication
- Django's built-in security middleware
- CORS headers for cross-origin resource sharing
- Redis is used in the authentication middleware (accounts/middlewares.py) for caching JWT Web Key Sets (JWKS)

3. Example Implementation

Here is an example implementation and data flow of **profile creation** feature

- **Frontend Layer**

```
// 1. Item Response Structure
interface ProfileData {
    name: string;
    email: string;
    industry: string;
    phoneNumber?: string;
    country?: string;
    city?: string;
    linkedInURL?: string;
    websiteLink?: string;
    avatar?: File;
    // ... other fields
}

// 2. Form Validation
const formSchema = z.object({
    name: z.string().min(2),
    email: z.string().email(),
    industry: z.string(),
    // ... other validations
});

// 3. Form Submission
async function onSubmit(profileData: ProfileData) {
    try {
        const response = await createUserProfile(profileData);
        // Handle success
    } catch (error) {
        // Handle error
    }
}
```

- **Backend Layer**
  - Authentication Middleware

```
class CreateProfileView(APIView):
    authentication_classes = [JWTAuthenticationMiddleware]
    parser_classes = (MultiPartParser, FormParser, JSONParser)
```

  - Authentication Check

```
user_account = UserAccount.objects.get(clerkUserID=request.user.username)
user_id = user_account.userID
```

  - Data Parsing

```
profile_data = json.loads(request.data.get('ProfileInfo', '{}'))
profile_data['userID'] = user_id
```

- File Handling

```
def validate_avatar_file(self, file):
    # Size validation
    if file.size > 2097152:  # 2MB limit
        raise ValidationError("Avatar file size must be less than 2MB")

    # Type validation
    if file_type not in ['jpeg', 'jpg', 'png', 'gif']:
        raise ValidationError("Invalid file type")

    # Convert to base64
    return f'{base64_data}'
```

- Data Validation & Saving

```
@transaction.atomic
def post(self, request):
    # Validate data
    serializer = ProfileSerializer(data=profile_data)
    if serializer.is_valid():
        # Save to database
        profile = serializer.save()
        return Response(
            serializer.data,
            status=status.HTTP_201_CREATED
        )
```

- **Database Layer**

```
class Profile(models.Model):
    profileID = models.AutoField(primary_key=True)
    userID = models.ForeignKey(UserAccount)
    name = models.CharField(
        max_length=100,
        validators=[RegexValidator(
            r'^[A-Za-z]+((\s)?((\'|\-|\.)?([A-Za-z])+))*$'
        )]
    )
    email = models.CharField(
        max_length=255,
        validators=[RegexValidator(
            r'^[^@\s]+@[^@\s]+\.[^@\s]+$'
        )]
    )
    # ... other fields
```

- **Response**
  - Success Response: transfer this data to the backend

```
{
    "profileID": "123",
    "name": "John Doe",
    "email": "john@example.com",
    "industry": "Technology",
    "avatar": "...",
    // ... other fields
}
```

- Error Response

```
{
    "error": "Validation error",
    "details": {
        "email": ["Invalid email format"],
        "avatar": ["File size too large"]
    }
}
```

4. Challenges During Execution

Axios Configuration

- Challenge: Managing authenticated requests with FormData and blob handling.
- Solution: Enhanced Axios interceptors to automate token addition and improved error management.

State Management

- Challenge: Maintaining consistent state across dynamic API responses and managing loading/error states.
- Solution: Utilized React Context API with efficient state-updating methods to ensure synchronization.

Dynamic Routing

- Challenge: Configuring Next.js dynamic [slug] routes and handling complex query parameters.
- Solution: Implemented clear URL structures and parameter parsers to ensure smooth navigation and reliable routing.

UI/UX Challenges

- Challenge: Designing responsive layouts, skeleton loaders, and fallbacks for missing data.
- Solution: Leveraged Tailwind CSS for responsiveness and designed custom skeleton components for data loading states.

File Handling

- Challenge: Dynamically handling Base64 avatars and file attachments.

- Solution: Integrated file preprocessing methods to manage encoding/decoding efficiently.

Animations

- Challenge: Implementing performance-optimized 3D flip cards with Framer Motion.
- Solution: Optimized animation timelines and incorporated lightweight libraries to prevent performance degradation.

Component Design

- Challenge: Ensuring reusability and efficient state management in dynamic components.
- Solution: Standardized component templates and modularized shared logic.

Debugging

- Challenge: Inspecting raw HTTP responses and resolving parameter-related issues.
- Solution: Integrated advanced logging tools to debug API requests effectively.

Scalability

- Challenge: Organizing project folders and decoupling logic for long-term maintainability.
- Solution: Adopted a feature-based folder structure and separated concerns within the codebase.

Validation and Forms

- Challenge: Validating FormData with files and dynamic field requirements.
- Solution: Created custom validation schemas to handle complex form validations.

Usability

- Challenge: Designing smooth workflows for user interactions such as profile edits and connections.
- Solution: Iteratively tested and refined user flows to ensure a seamless experience.

Database Optimization

- Challenge: Since this is a complete system and there are many data tables and fields to be stored, it is very difficult for our team to organize and fit them into the database at once.
- Solution: We had to drawed and optimize ER diagram iteratively and make adjustment on the corresponding backend and frontend parts.

API Integration

- Challenge: Synchronizing frontend and backend systems required more effort than expected, especially in ensuring seamless data flow and real-time updates.
- Solution: We prepared detailed API documentation to streamline the integration process.

VI. Results

The development successfully achieved its objectives, delivering a polished, functional platform across both frontend and backend.

Functional Achievements

- **Authentication**:
  - Integrated secure user authentication with third-party providers like Google, Facebook, and LinkedIn via Clerk.
  - Backend handled token-based authentication and session storage securely.
- **Dashboard**:
  - Frontend displayed dynamic metrics such as profile views, matched startups, and search appearances using reusable components.
  - Real-time updates enabled through backend APIs.
- **Profile Management**:
  - Users could create, edit, and manage profiles dynamically.
  - Backend maintained profile data securely in a structured PostgreSQL database.
- **Discover Feature**:
  - Tinder-like swiping mechanism enabled users to explore profiles interactively.
  - Backend implemented algorithms to generate personalized profile recommendations.
- **Notifications**:
  - Users received real-time notifications for interactions, such as connection requests and updates.
  - Notifications were managed efficiently through backend event triggers and APIs.
- **Connection Management**:
  - Users could view, save, and connect with other profiles.
  - Backend ensured mutual connections were established and managed properly.

User Feedback and Refinements

- **Positive Feedback**:
  - Test users highlighted the intuitive interface, engaging design, and smooth navigation.
  - Backend reliability was commended for quick responses and seamless data synchronization.
  - The client expressed great enthusiasm for the project and scheduled a demo for after the Tet holiday.
- **Issues Addressed**:
  - Manage Errors gracefully across the application
  - Resolved UI/UX and page routing concerns on the frontend.
  - Fixed API endpoint inconsistencies and improved response times on the backend.

Performance Metrics

- **Frontend Responsiveness**:

    - Consistent performance across devices was achieved.
    - Pages loaded within an average of under 3 seconds, ensuring a smooth user experience.
- **Backend Efficiency**:
    - Optimized API calls reduced average response time to under 1 second.
    - Database queries were refined to handle large datasets without performance degradation.

Challenges Overcome

- **Design Constraints**:
    - Simplified complex Figma designs into functional components without losing the intended aesthetics.
    - Adjusted backend logic to accommodate evolving frontend requirements.
- **Midterm and Final Period**:
    - Extended timelines and team "hackathons" helped recover lost progress for both frontend and backend, successfully regained momentum and delivered the planned features.
- **Integration Hurdles**:
    - Synchronized frontend and backend systems effectively, ensuring seamless communication between APIs and UI components.

Impact

- The platform’s user-friendly and visually engaging frontend, combined with a robust and scalable backend, laid a solid foundation for fostering entrepreneurial connections.
- Initial user feedback validated the platform’s potential for enhancing engagement and collaboration within the startup ecosystem.
- The backend’s secure and efficient data handling ensured a reliable user experience, even under high loads.

VII. Discussion

1. Analysis

- **Delivered Outcomes**
    - **User-Centric Design**: Delivered a user-friendly interface designed for seamless navigation and interaction, inspired by popular UI patterns.
    - **Core Functionalities**: Successfully implemented key features, namely creating profile, connection, and dashboards.
    - **RESTful API**: Deployed all APIs that align with API docs, ensuring efficiency
    - **Data-Driven Insights**: Integrated analytics functionalities, allowing users to track profile views, connections, and engagement metrics.
    - **Database Optimization**: Designed a scalable database schema with detailed relationships and privacy settings for diverse user requirements.
- **User Interface:**

        - **Our UI vs. Tinder UI**: Our interface is inspried by Tinder's intuitive swipe-and-match mechanism. But while Tinder focuses on simplicity for casual connections, our design incorporates additional layers of information and decision making to professional contexts
    - **Feature Set**
        - **Our Features vs. LinkedIn Features**: While LinkedIn excels in professional networking with broad social and content-sharing functionalities, our platform encourage people seek for connections, particularly between startup founders and candidates. The matching algorithm and detailed analytics are unique selling points that make our system different from LinkedIn's generalized approach.

2. Limitations

- **Recommendation System:** Researching and implementing matching algorithm is hard for us, but it is one of the core components of the project so we will need put more effort on it. If we do not have a proper matching strategy, it can lead to unsatisfied experience for users.
- **Design Difference**: Regarding the UI prototype design, even though we have made significant efforts to implement the Figma prototypes, there are still several aspects requiring refinement based on client pop up feedback. These revisions have caused postponements in other stages. We addressed this by proposing alternative designs that balance usability and goals while staying aligned with project timeline.
- **Frontend Scalability**: The current design, while effective, requires substantial adjustments to support larger datasets and more complex user interactions. Future development phases should prioritize enhancing scalability to meet growing user demands.
- **Mobile Optimization**: Although the frontend is responsive, specific mobile-first optimizations could further improve usability on smaller devices. Addressing these gaps would ensure a seamless experience across all device types.
- **Time Limitation:** Due the time constraint and some minor pending bugs, this web application is not ready for deploying yet
- **Documentation:** Comprehensive API documentation and detailed user guides are needed to simplify onboarding for both developers and end-users. This will enhance the platform's maintainability and accessibility.

VIII. Conclusion

The development of the job-matching platform is an integral step in making an improvement to the entrepreneurial ecosystem of VinUniversity and beyond. User-centric designs by the likes of Tinder and LinkedIn converge with novel advanced matching algorithms such as PageRank and keyword-based techniques within a single place to create that particular engaging, yet efficient, networking experience.

While some features have been left out due to the time constraints, the base laid by this project is strong and scalable. Subsequent iterations will polish these elements to ensure full functionality and better user satisfaction, meeting not only the immediate needs of startups and professionals but also creating a model that can scale for similar ecosystems, fostering innovation and collaboration in the fast-growing entrepreneurial landscape of Vietnam.

IX. Project Links

- Github link for main project: https://github.com/HARDeConstruction/FounderMatching
- Github link for API document: https://github.com/HARDeConstruction/api-fm
- Figma design link: https://www.figma.com/design/EoJY3yIgQXV0hvIbvVIdJa/Founder-Matching-App?node-id=0-1&p=f&t=GoMwhYJEvuBT3uCV-0
- Frontend deployment link: https://founder-matching.vercel.app/