Technology Specification Document: Nonprofit & For-Profit Matching System

Version: 2.1
Date: *(Add Date)*
Author: *(Your Name)*

1. Introduction

1.1 Purpose

This document provides detailed specifications for the Nonprofit & For-Profit Matching System, describing the end-to-end workflow, API requirements, and system processes for engineers to implement the platform.

The system facilitates partnerships by allowing organizations to create profiles, store their data, generate AI embeddings using OpenAI, and retrieve matching partners through **MongoDB Atlas Vector Search, which performs AI-powered similarity matching directly within the database**.

2. System Architecture

The system is composed of three main layers:

- **Frontend Layer**: A web-based interface that allows users to register, manage profiles, and search for partnerships.
- **Backend Layer**: A FastAPI-based RESTful API that handles authentication, data storage, AI processing, and search functionality.

- **Database Layer**: A MongoDB Atlas database that stores organization details and AI embeddings for similarity matching.

The workflow of the system follows a sequential process:

1. **User Registration & Authentication**
    - Users create an account and log in using JWT authentication.
2. **Profile Creation & Data Storage**
    - Users submit organization details, which are validated and stored in MongoDB.
    - An AI embedding is generated from the organization's description and stored alongside other data.
3. **Search Query Processing & AI Matching**
    - When a user searches for a partner, their query is converted into an embedding.
    - The system filters organizations based on **organization type and location**.
    - Cosine similarity matching is handled directly within MongoDB Atlas Vector Search, eliminating the need for external similarity computations in FastAPI.
4. **Results Display & Interaction**
    - Users receive a list of recommended matches with options to refine the search.

3. Workflow Pipeline

The following table outlines the step-by-step data flow across the system, detailing frontend interactions, backend processing, and database operations.

Workflow Step	Frontend (React.js)	Backend (FastAPI, Python)	Database (MongoDB)

1. User Registration	User enters email & password, submits registration form.	Receives request, validates input, hashes password, stores user data.	Stores user credentials and assigns a unique user ID.
2. User Authentication	User logs in, JWT token is stored in local storage.	Verifies credentials, issues JWT token, returns authentication response.	Verifies user credentials.
3. Profile Creation (Input Data)	User fills out organization profile (name, category, location, description).	Receives request, validates inputs, prepares data for storage.	Stores raw profile data (excluding embedding).
4. AI Embedding Generation	-	Calls OpenAI API with the generated **tags** of the organization. Receives an AI-generated vector embedding of the **tags only**.	Embedding vector is stored alongside profile data.
5. Profile Submission Completion	Displays success message, user can view/edit profile.	Confirms profile and embedding are stored successfully.	Organization profile is complete with all details and embedding.

6. User Initiates a Search Query	User enters search text and selects filters (state, category). Clicks Search.	Converts search text into an AI embedding using OpenAI API.	-
7. Pre-Matching Filtering	-	"Sends one query to MongoDB with the search embedding and filters.	"Sends one query to MongoDB with the search embedding and filters. Uses `$vectorSearch` to **filter by category, state, and organization type (`nonprofit` or `for-profit`)** before computing similarity on the tag-based embeddings."
8. AI-Based Similarity Matching	-	No longer computes cosine similarity in Python. Simply **forwards MongoDB's ranked results**.	MongoDB Atlas Vector Search retrieves stored embeddings and directly returns the top 5 most relevant

			matches, pre-ranked based on similarity scores. The search is **limited to organizations matching the selected organization type (`nonprofit` or `for-profit`).**
9. Returning Matched Organizations	-	Sends the results directly back to the frontend.	-
10. Displaying Search Results	Renders matched organizations in a ranked list. User can refine search.	-	-

4. Explanation of Workflow Steps

Step 1-2: User Registration & Authentication

- The user registers an account with an email and password.
- FastAPI hashes the password and stores credentials securely in the database.
- Once registered, the user logs in, and FastAPI generates a JWT token, which is required for accessing protected API endpoints.

Step 3-5: Organization Profile Submission

- The user enters organization details (name, description, location, and category) and submits the form.
- FastAPI validates the input and calls OpenAI's embedding API to generate a vector representation of the organization based on its description.
- The organization profile and embedding are stored together in MongoDB.

Step 6-7: User Searches for a Partner

- The user enters a search query and selects filters (state, category).
- FastAPI converts the search query into an embedding using OpenAI.
- The system filters organizations that match the selected category and state to improve search efficiency.

Step 8-9: AI Similarity Matching

- MongoDB Atlas Vector Search retrieves stored embeddings and **computes similarity using built-in vector search capabilities**. It **directly returns the top 5 most relevant matches** based on similarity scores, eliminating the need for FastAPI to compute similarity.
- Organizations are sorted by similarity score, and the top 5 results are selected.

Step 10: Displaying Search Results

- The frontend renders the ranked list of matches, including organization names, descriptions, websites, and similarity scores.
- Users can refine their search filters and repeat the process.

5. Database Architecture

1. Users Collection → Stores user credentials and authentication data.
2. Organizations Collection → Stores organization details and AI embeddings for matching.

Both collections will be stored in MongoDB (Local Instance or MongoDB Atlas).

5.1 Users Collection

Purpose

This collection stores user authentication data, allowing users to register, log in, and manage their accounts.

Schema

```
{
    "_id": ObjectId("65f2a3e4c56a2f3d2a8d9f23"),
    "email": "user@example.com",
    "password_hash": "$2b$12$exampleHashedPassword",
    "full_name": "John Doe",
    "created_at": ISODate("2024-02-22T12:00:00Z"),
    "last_login": ISODate("2024-02-22T15:30:00Z"),
    "role": "user"  // Possible values: "user", "admin"
}
```

Indexes & Changes

- Index on `email` → Speeds up authentication lookup.
- Ensure `password_hash` is securely stored → Use bcrypt hashing.
- Track `last_login` for monitoring user activity.

Indexing Strategy:

```
db.users.create_index([("email", 1)], unique=True)
```

5.2 Organizations Collection

Purpose

This collection stores organization profiles, including descriptions, categories, locations, and AI-generated embeddings for similarity matching.

Schema

```
{
    "_id": ObjectId("65f5b6d7c56a2f3d2a8d9f45"),
    "user_id": ObjectId("65f2a3e4c56a2f3d2a8d9f23"),
    "name": "Walmart Inc.",
    "category": "Retail",
    "organization_type": "for-profit",  // New field to differentiate
nonprofits from for-profits
    "description": "...",
    "state": "Arkansas",
    "city": "Bentonville",
    "website": "https://corporate.walmart.com",
    "phone": "1-479-273-4000",
    "tags": ["retail", "supermarket", "e-commerce", "consumer goods",
"supply chain"],
    "embedding_tags": Binary("<BSON Vector Format>"),
    "created_at": ISODate("2024-02-22T12:00:00Z")
}
```

Indexes & Changes

✅ Index on `category` and `state` → Enables fast filtering.
✅ **Store embeddings as BSON vectors instead of lists** → Required for MongoDB Atlas Vector Search.

✅ **Create a `$vectorSearch` index** to enable similarity queries.
✅ Reference `user_id` to link organizations to owners.

Indexing Strategy:

```
db.organizations.create_index([("organization_type", 1),
("state", 1), ("category", 1)])
```

5.3. Summary of Changes Needed

Collection	Required Changes
`users`	Add index on email for fast authentication, store passwords securely using bcrypt, track `last_login`.
`organizations`	Add indexes on `"organization_type"` field (values: `"nonprofit"` or `"for-profit"`) to support filtering., category, state, and embeddings, Ensure embeddings are stored in **BSON Binary format (`Binary.from_vector(embedding, "float32")`)**, which is required for MongoDB Atlas Vector Search." link `user_id` for ownership tracking.

.

6. Required API Endpoints & Requests

6.1 Authentication APIs

Method	Endpoint	Purpose
POST	/auth/signup	Registers a new user and stores credentials.
POST	/auth/login	Authenticates a user, issues JWT token.
GET	/auth/profile	Retrieves authenticated user profile.

Organization Management APIs

Method	Endpoint	Purpose
POST	/organization/create	Submits a new organization profile.
GET	/organization/{id}	Retrieves an organization's details.
PUT	/organization/{id}/update	Updates organization profile data.

Search & Matching APIs

Method	Endpoint	Purpose
GET	GET /match?query=text&state=state&category=category&type=nonprofit|for-profit	matching only against embedded tags (`embedding_tags`).

6.2

7. Resources & Dependencies

The system relies on specific tools, dependencies, and configurations to ensure smooth functionality. Below is a breakdown of the required components, including necessary **API keys, URLs, libraries, and applications** that need to be installed or configured before implementation.

Component	Technology Used	Required Dependencies & Configuration

Frontend	React.js	- Node.js (Latest LTS version) - React Router for navigation - Axios for API requests - dotenv for environment variable management - Must install with npm install
Backend	FastAPI (Python)	- Python 3.9+ - FastAPI framework - Uvicorn for ASGI server - Pydantic for data validation - CORS Middleware for cross-origin requests - Install dependencies via pip install fastapi uvicorn pydantic
Database	MongoDB (Local Instance)	- MongoDB Community Server or Atlas - pymongo for database connection - Connection string required: MONGODB_URI="mongodb://localhost:27017/your_db" (for local) or MONGODB_URI="your_mongodb_atlas_url" (for cloud)
Authentication	JWT (JSON Web Token)	- PyJWT for encoding/decoding JWT - Secret key stored in .env: JWT_SECRET_KEY="your_secret_key"

		- Token expiration settings required
AI Embeddings	OpenAI API	- API key required: OPENAI_API_KEY="your_openai_api_key" - OpenAI Python SDK (pip install openai) - Endpoint used: https://api.openai.com/v1/embeddings - Model: text-embedding-ada-002
Search Algorithm	Search	-MongoDB Atlas Search API handles similarity matching natively. -Requires an Atlas cluster with a $vectorSearch index.
Package Manager	pip (Python)	- Required for managing backend dependencies - Install all dependencies via pip install -r requirements.txt
Environment Variables	.env file	- .env file must be created to store sensitive keys - Required variables: MONGODB_URI, OPENAI_API_KEY, JWT_SECRET_KEY, DEPLOYMENT_ENV=development

8. Files Needed for GitHub Upload

To properly upload the project to GitHub, the repository should contain **all essential files** for both frontend and backend while ensuring sensitive information like API keys are **excluded**. Below is a list of the necessary files:

8.1 Project Structure

```
/project-root
├── backend/
│   ├── main.py
│   ├── requirements.txt
│   ├── .env
│   ├── config.py
│   ├── routes/
│   │   ├── auth.py
│   │   ├── organization.py
│   │   ├── search.py
│   ├── models/
│   │   ├── user.py
│   │   ├── organization.py
│   ├── services/
│   │   ├── embedding.py
│   ├── utils/
│   │   ├── security.py
│   ├── .gitignore
│   ├── README.md
│
├── frontend/
│   ├── package.json
│   ├── .env
│   ├── src/
│   │   ├── App.js
│   │   ├── components/
│   │   │   ├── Signup.js
```

```
│   │   │   ├── Login.js
│   │   │   ├── OrganizationForm.js
│   │   │   ├── Search.js
│   ├── .gitignore
│   ├── README.md
│
├── .gitignore
├── README.md
```

8.2 Backend Files

Required Files for Backend

File Name	Purpose
main.py	Entry point for FastAPI application.
requirements.txt	List of dependencies (FastAPI, OpenAI, pymongo, etc.).
.env	Stores sensitive API keys (MONGODB_URI, OPENAI_API_KEY, JWT_SECRET_KEY).
config.py	Configuration settings (e.g., loading .env variables).
routes/auth.py	Handles user authentication (signup, login, JWT token management).
routes/organization.py	Handles organization profile creation and retrieval.

routes/search.py	Implements AI-powered search and matching.
models/user.py	Defines the database schema for users.
models/organization.py	Defines the database schema for organizations.
services/embedding.py	Calls OpenAI API to generate embeddings.
utils/security.py	Handles JWT authentication, password hashing.
.gitignore	Prevents uploading sensitive files like .env.
README.md	Documentation for backend setup and usage.

.gitignore for Backend

Create a .gitignore file in the backend directory to **prevent sensitive files** from being uploaded.

```
# Ignore environment variables
.env

# Ignore Python cache files
__pycache__/
*.pyc
*.pyo

# Ignore virtual environments
venv/
```

8.3 Frontend Files

Required Files for Frontend

File Name	Purpose
package.json	Contains project dependencies and scripts.
.env	Stores API URLs (REACT_APP_BACKEND_URL).
src/App.js	Main React application file.
src/components/Signup.js	Handles user registration.
src/components/Login.js	Handles user authentication.
src/components/OrganizationForm.js	Allows users to submit organization details.
src/components/Search.js	Implements the search and matching feature.
.gitignore	Prevents unnecessary files from being uploaded.
README.md	Documentation for frontend setup and usage.

.gitignore for Frontend

Create a .gitignore file in the frontend directory to **prevent sensitive files** from being uploaded.

```
# Ignore environment variables
.env

# Ignore node_modules
node_modules/
npm-debug.log

# Ignore build files
/build
```

8.4 Environment Variables (.env File for Backend & Frontend)

To prevent exposing sensitive credentials, **store API keys and database connection strings in environment files**.

Backend (backend/.env)

```
# 🚀 Application Settings
DEPLOYMENT_ENV=development  # Possible values: development,
production
PORT=8000  # API server port
LOG_LEVEL=info  # Log verbosity level

# 🔐 Authentication & Security
JWT_SECRET_KEY="your-secret-key"  # Secret key for JWT
authentication
JWT_EXPIRATION_HOURS=24  # Token expiration time (in hours)
PASSWORD_SALT_ROUNDS=12  # Bcrypt salt rounds for password
hashing
```

```
# 🏦 Database Configuration (MongoDB)
MONGODB_URI="mongodb+srv://user:password@cluster.mongodb.net/your_db"
MONGODB_DATABASE_NAME="your-database"
MONGODB_USERS_COLLECTION="users"
MONGODB_ORGANIZATIONS_COLLECTION="organizations"

# 🤖 AI Embedding & Matching (OpenAI)
OPENAI_API_KEY="your_openai_api_key"
OPENAI_EMBEDDING_MODEL="text-embedding-ada-002"

# 🔍 Vector Search (MongoDB Atlas)
MONGO_VECTOR_SEARCH_INDEX="vector_search_index"  # Index for AI-powered matching

# 🌐 OAuth (Google Authentication)
GOOGLE_CLIENT_ID="248657882307-ovuh63vcuced62v0pst2p8hi6alj1crg.apps.googleusercontent.com"
GOOGLE_CLIENT_SECRET="yGOCSPX-Kboz4c86nf735tgo7GFUiwJsUBiw"
GOOGLE_CALLBACK_URL="http://localhost:8000/api/auth/google/callback"

# 🔗 Cross-Origin Resource Sharing (CORS)
FRONTEND_URL="http://localhost:3000"  # Allowed frontend origin for API requests

# 📧 Email Notifications (Optional)
EMAIL_SENDER="noreply@yourdomain.com"
SMTP_SERVER="smtp.yourdomain.com"
SMTP_PORT=587
SMTP_USERNAME="your-email-username"
SMTP_PASSWORD="your-email-password"
```

IP Address	Comment	Status	Actions
205.175.106.95/32(includes your current IP address)	CauseConnect Database	Active	Edit Delete

Frontend (frontend/.env)

```
# 🚀 API Configuration
REACT_APP_BACKEND_URL="http://localhost:8000"  # API Base URL

# 🔐 Authentication & Security
REACT_APP_GOOGLE_CLIENT_ID="248657882307-ovuh63vcuced62v0pst2p8hi6alj1crg.apps.googleusercontent.com"

# 🌍 Deployment Environment
REACT_APP_ENV="development"  # Possible values: development, production
```

9. Appendix

MongoDB Atlas
C:\Users\Shun-Xi Wu>atlas
CLI tool to manage MongoDB Atlas. The Atlas CLI is a command line interface built specifically for MongoDB Atlas. You can manage your Atlas database deployments and Atlas Search from the terminal with short, intuitive commands.

Use the --help flag with any command for more info on that command.

To learn more, see our documentation:
https://www.mongodb.com/docs/atlas/cli/stable/connect-atlas-cli/

Usage:

```
  atlas [command]

Examples:
  # Display the help menu for the config command:
  atlas config --help


Available Commands:
  config                Configure and manage your user profiles.
  auth                  Manage the CLI's authentication state.
  setup                 Register, authenticate, create, and access an Atlas cluster.
  projects              Manage your Atlas projects.
  organizations         Manage your Atlas organizations.
  users                 Manage your Atlas users.
  teams                 Manage your Atlas teams.
  clusters              Manage clusters for your project.
  dbusers               Manage database users for your project.
  customDbRoles         Manage custom database roles for your project.
  accessLists           Manage the list of IP addresses that can access your Atlas project.
  alerts                Manage alerts for your project.
  backups               Manage cloud backups for your project.
  events                Manage events for your organization or project.
  metrics               Get metrics on the MongoDB process.
  performanceAdvisor    Learn more about slow queries and get suggestions to improve
database performance.
  logs                  Download host logs for your project.
  processes             Manage MongoDB processes for your project.
  privateEndpoints      Manage Atlas private endpoints.
  networking            Manage or configure network peering for your Atlas project.
  security              Manage security configuration for your project.
  integrations          Configure third-party integrations for your Atlas project.
  maintenanceWindows    Manage Atlas maintenance windows.
  customDns             Manage DNS configuration of Atlas project's clusters deployed to AWS.
  cloudProviders        Manage cloud provider access in Atlas using AWS IAM roles.
  streams               Manage your Atlas Stream Processing deployments.
  liveMigrations        Manage a Live Migration to Atlas for your organization.
  accessLogs            Return the access logs for a cluster.
  kubernetes            Manage Kubernetes resources.
  dataFederation        Data federation.
  auditing              Returns database auditing settings for MongoDB Cloud projects.
  deployments           Manage cloud and local deployments.
  federatedAuthentication Manage Atlas Federated Authentication.
  api                   Interact directly with any Atlas Admin API endpoint through the Atlas CLI,
streamlining script development.
```

```
  plugin              Manage plugins for the AtlasCLI.
  help                Help about any command
  completion          Generate the autocompletion script for the specified shell

Flags:
  -h, --help          help for atlas
  -P, --profile string   Name of the profile to use from your configuration file. To learn about
profiles for the Atlas CLI, see
https://dochub.mongodb.org/core/atlas-cli-save-connection-settings.
  -v, --version       version for atlas

Use "atlas [command] --help" for more information about a command.
```

```python
from pymongo.mongo_client import MongoClient

uri =
"mongodb+srv://Cluster13662:<db_password>@cluster13662.s1t3w.mongodb.net/?r
etryWrites=true&w=majority&appName=Cluster13662"

# Create a new client and connect to the server
client = MongoClient(uri)

# Send a ping to confirm a successful connection
try:
    client.admin.command('ping')
    print("Pinged your deployment. You successfully connected to MongoDB!")
except Exception as e:
    print(e)
```

Replace **<db_password>** with the password for the **Cluster13662** database user. Ensure any option params are

Save API Key Information

```
Public Key jxbbdzjo
Private Key b167304e-633a-4658-9e51-bd5a16f523d0
Organization ID 67a29616fd041b0750bc247a
Project ID 67a29616fd041b0750bc24a3
```