ScholarBridge: Final Project Roadmap & Implementation Guide

Document Version: 1.2
Last Updated: October 7, 2025
Author: Gemini (Project Lead)

1. Executive Summary

This document outlines the finalized plan to integrate an intelligent **"Best Fit" Matching Algorithm** into the ScholarBridge platform. The primary objective is to transform the Trust Dashboard from a simple list of all applications into a prioritized, actionable workbench.

This feature will dramatically improve the user experience for trusts by saving them significant time and effort. It will also serve as the cornerstone for a potential research paper by applying a heuristic-based algorithm to a real-world matching problem.

The implementation is designed to be highly efficient, scalable, and integrated seamlessly into the existing project structure with minimal, non-breaking changes.

2. The Core Feature: "Best Fit" Matching Algorithm

2.1. Problem Statement

Trusts are often overwhelmed by the sheer volume of scholarship applications, many of which are not relevant to their specific charter (e.g., wrong course, location, or gender). This leads to wasted review time and potential for overlooking ideal candidates.

2.2. The Solution: A Prioritized Workbench

We will implement a backend algorithm that calculates a **Match Score (0-100)** for every application relative to each trust's specific funding preferences. The Trust Dashboard will then, by default, show a **smart-filtered list** of only *eligible* applications, sorted from the highest score to the lowest.

2.3. Finalized Scoring Model (Version 2.0)

The algorithm is based on a weighted scoring model that mimics real-world decision-making by prioritizing non-negotiable "gatekeeper" criteria.

Parameter	Data Source (Student)	Weight	Justification
Gender Match	student_profiles.gender	**35 pts**	The highest weight. A hard, exclusionary filter for gender-specific trusts (e.g., "for women in STEM").
Course Match	applications.current_course_name	**30 pts**	A primary gatekeeper. A trust for medical students considers engineering applications ineligible.
Location Match	student_profiles.address (city)	**15 pts**	A common gatekeeper for geographically-focused trusts.
Financial Need	SUM of all family_members.monthly_income	**15 pts**	The core mission filter. Prioritizes students who fall below the trust's defined income ceiling.
Academic Merit	education_history.grade	**5 pts**	A refining factor/tie-breaker. Gives a slight edge to students who meet the academic threshold.
Total Max Score		**100 pts**	

2.4. Handling Edge Cases

- **"Any" Preference:** If a trust leaves a preference field blank (e.g., "Preferred Courses"), the algorithm will treat it as a wildcard, automatically awarding full points for that category to all applications.
- **Tie-Breaker Logic:** If multiple applications have the same Match Score, they will be sorted using a fair, three-level hierarchy:
  1. **Primary:** match_score (Descending)
  2. **Secondary:** total_family_income (Ascending - greater need is prioritized)
  3. **Tertiary:** application.created_at (Ascending - first-come, first-served)

3. Step-by-Step Implementation Plan

Phase 1: Database Modification (One-Time Change)

We must add a single jsonb column to the trusts table to store their funding preferences. This is a non-breaking change.

Execute this SQL command on your NeonDB database:

```
ALTER TABLE trusts
ADD COLUMN preferences jsonb DEFAULT '{}'::jsonb;
```

This command is idempotent and safe to run even if the column already exists.

Phase 2: Backend Implementation (Node.js/Express)

Step 2.1: Create an Endpoint for Trusts to Set Their Preferences

A trust needs to be able to save their preferences. We will create a new, protected endpoint for this.

File: scholarbridge-api/src/routes/trustRoutes.js
Action: Add a new route.

```
// ... existing imports
const { getDashboardApplications, updateTrustPreferences, getTrustPreferences } =
require('../controllers/trustController'); // Import new controllers

// ... existing routes

// NEW ROUTES: For a trust to get and update their own preferences
router.get('/me/preferences', authMiddleware, requireRole('trust'), getTrustPreferences);
router.put('/me/preferences', authMiddleware, requireRole('trust'), updateTrustPreferences);
```

File: scholarbridge-api/src/controllers/trustController.js
Action: Add the getTrustPreferences and updateTrustPreferences controller functions.

```
// ... existing functions

// New controller to GET trust preferences
exports.getTrustPreferences = async (req, res) => {
  try {
    const { rows } = await db.query('SELECT preferences FROM trusts WHERE user_id = $1',
[req.user.id]);
    if (rows.length === 0) return res.status(404).json({ error: 'Trust profile not found.' });
    res.json(rows[0].preferences);
  } catch (err) {
    console.error('Error fetching trust preferences:', err);
    res.status(500).json({ error: 'Server error' });
  }
};

// New controller to UPDATE trust preferences
exports.updateTrustPreferences = async (req, res) => {
  try {
    const trustUserId = req.user.id;
    const preferences = req.body;

    const query = `
      UPDATE trusts
      SET preferences = $1, updated_at = now()
      WHERE user_id = $2
      RETURNING user_id, preferences;
    `;
    const { rows } = await db.query(query, [JSON.stringify(preferences), trustUserId]);
    if (rows.length === 0) return res.status(404).json({ error: 'Trust profile not found.' });
    res.json({ message: 'Preferences updated successfully.', preferences:
rows[0].preferences });
  } catch (err) {
    console.error('Error updating trust preferences:', err);
    res.status(500).json({ error: 'Server error' });
  }
};
```

Step 2.2: Implement the Main "Best Fit" Dashboard Endpoint

This is the core of the backend logic. It will perform the smart filtering, scoring, and sorting in a single, efficient query.

File: scholarbridge-api/src/routes/trustRoutes.js
Action: Add the new dashboard route.

```
// NEW ROUTE: For the trust's main dashboard view
router.get('/dashboard/applications', authMiddleware, requireRole('trust'),
getDashboardApplications);
```

File: scholarbridge-api/src/controllers/trustController.js
Action: Add the getDashboardApplications controller function with the final, complete query.

```
// New controller for the Trust Dashboard
exports.getDashboardApplications = async (req, res) => {
  const trustUserId = req.user.id;
  const { view } = req.query; // Check for ?view=all

  const query = `
    WITH
    -- Step 1: Get the preferences for the currently logged-in trust
    trust_prefs AS (
      SELECT
        preferences->>'preferred_gender' AS p_gender,
        jsonb_array_to_text_array(preferences->'preferred_courses') AS p_courses,
        jsonb_array_to_text_array(preferences->'preferred_cities') AS p_cities,
        (preferences->>'max_family_income_lpa')::numeric AS p_max_income,
        (preferences->>'min_academic_percentage')::numeric AS p_min_grade
      FROM trusts
      WHERE user_id = $1
    ),
    -- Step 2: Pre-calculate academic and financial info for each application
    application_details AS (
      SELECT
        app.id AS application_id,
        -- Calculate Weighted Academic Score from education_history (most recent 3
years)
        COALESCE(AVG(
          CASE
            WHEN eh.year_of_passing = (SELECT MAX(year_of_passing) FROM
education_history WHERE application_id = app.id) THEN eh.grade::numeric * 0.5
```

```
                WHEN eh.year_of_passing = (SELECT MAX(year_of_passing) FROM
education_history WHERE application_id = app.id) - 1 THEN eh.grade::numeric * 0.3
                ELSE eh.grade::numeric * 0.2
              END
            ), 0) AS weighted_academic_score,
            -- Calculate Total Family Income in Lakhs Per Annum by summing all family members
            COALESCE(SUM(fm.monthly_income), 0) * 12 / 100000 AS total_family_income_lpa
          FROM
            applications app
          LEFT JOIN education_history eh ON app.id = eh.application_id
          LEFT JOIN family_members fm ON app.id = fm.application_id
          GROUP BY app.id
        )
        -- Step 3: Main query to filter, score, and sort applications
        SELECT
          a.id AS application_id,
          sp.full_name,
          a.current_course_name,
          sp.address->>'city' as city,
          ad.total_family_income_lpa,
          ad.weighted_academic_score,
          -- THE FINAL SCORING ALGORITHM
          (
            CASE WHEN sp.gender = tp.p_gender OR tp.p_gender IS NULL OR tp.p_gender =
'Any' THEN 35 ELSE 0 END +
            CASE WHEN tp.p_courses IS NULL OR a.current_course_name = ANY(tp.p_courses)
THEN 30 ELSE 0 END +
            CASE WHEN tp.p_cities IS NULL OR sp.address->>'city' = ANY(tp.p_cities) THEN 15
ELSE 0 END +
            CASE WHEN tp.p_max_income IS NULL OR ad.total_family_income_lpa <=
tp.p_max_income THEN 15 ELSE 0 END +
            CASE WHEN tp.p_min_grade IS NULL OR ad.weighted_academic_score >=
tp.p_min_grade THEN 5 ELSE 0 END
          ) AS match_score
        FROM
          applications a
        JOIN
          student_profiles sp ON a.student_user_id = sp.user_id
        JOIN
          application_details ad ON a.id = ad.application_id
        CROSS JOIN
```

```
        trust_prefs tp
      -- The "Smart Filter" is active unless view=all
      WHERE a.status = 'submitted'
      ${view !== 'all' ? `
        AND (tp.p_gender IS NULL OR tp.p_gender = 'Any' OR sp.gender = tp.p_gender)
        AND (tp.p_courses IS NULL OR a.current_course_name = ANY(tp.p_courses))
        AND (tp.p_cities IS NULL OR sp.address->>'city' = ANY(tp.p_cities))
      ` : ''}
      -- THE FINAL SORTING HIERARCHY
      ORDER BY
        match_score DESC,
        ad.total_family_income_lpa ASC,
        a.created_at ASC;
    `;

    try {
      const { rows } = await db.query(query, [trustUserId]);
      res.json(rows);
    } catch (err) {
      console.error('Error fetching dashboard applications:', err);
      res.status(500).json({ error: 'Server error' });
    }
};

// Note: This implementation assumes you have the following helper function in your
PostgreSQL database.
// You only need to run this CREATE FUNCTION command once.
/*
CREATE OR REPLACE FUNCTION jsonb_array_to_text_array(p_jsonb jsonb)
RETURNS text[] LANGUAGE sql IMMUTABLE AS $$    SELECT ARRAY(SELECT value FROM
jsonb_array_elements_text(p_jsonb));$$;
*/
```

Phase 3: Frontend Implementation (React)

Step 3.1: Create the "Funding Preferences" Page

This new page allows trusts to control the algorithm's parameters.

- **Where it will be displayed:** A new link titled **"Funding Preferences"** will be added to the main navigation sidebar or header for logged-in trusts. This makes it easily

discoverable and accessible.

- **File:** scholarbridge-client/src/pages/trust/TrustPreferencesPage.jsx (New File)
- **Action:** Create a new page component for trusts to manage their preferences. The form on this page will directly update the preferences field in the database.

```
import React, { useState, useEffect } from 'react';
import apiClient from '../../api/apiClient';

const TrustPreferencesPage = () => {
  const [preferences, setPreferences] = useState({
    preferred_gender: 'Any',
    preferred_courses: [], // Store as an array
    preferred_cities: [],
    max_family_income_lpa: '',
    min_academic_percentage: ''
  });
  const [loading, setLoading] = useState(false);

  // Fetch current preferences on load to populate the form
  useEffect(() => {
    const fetchPrefs = async () => {
      try {
        const { data } = await apiClient.get('/trusts/me/preferences');
        // Set state, ensuring arrays are handled correctly from text input
        setPreferences({
          ...data,
          preferred_courses: data.preferred_courses || [],
          preferred_cities: data.preferred_cities || [],
        });
      } catch (error) {
        console.error("Could not fetch preferences", error);
      }
    };
    fetchPrefs();
  }, []);

  const handleSave = async () => {
    setLoading(true);
    try {
      const payload = { ...preferences };
      if (!payload.max_family_income_lpa) delete payload.max_family_income_lpa;
```

```
        if (!payload.min_academic_percentage) delete payload.min_academic_percentage;

        await apiClient.put('/trusts/me/preferences', payload);
        alert('Preferences saved successfully!');
      } catch (error) {
        console.error('Failed to save preferences', error);
        alert('An error occurred.');
      } finally {
        setLoading(false);
      }
    };

    return (
      <div className="p-6">
        <h1 className="text-2xl font-bold mb-4">Funding Preferences</h1>
        <p className="mb-6 text-gray-600">Set your criteria here to see the most relevant applications first. Leave fields blank to consider all applicants for that category.</p>
        {/* Form JSX goes here: a dropdown for Gender, text inputs for courses/cities, number inputs for income/percentage, and a save button */}
      </div>
    );
};

export default TrustPreferencesPage;
```

Step 3.2: Upgrade the Trust Dashboard

This is the main landing page for trusts, where they see the prioritized list of applicants.

- **Where it will be displayed:** This is the existing **TrustDashboard.jsx** page, which is the default view when a trust logs in. We are enhancing it, not replacing it.
- **New Elements on the Page:**
  1. **Toggle Switch:** A toggle labeled "Only show applications matching my preferences" will be placed prominently at the top of the page, below the main "Applicant Dashboard" title. It will be enabled by default.
  2. **Match Score Column:** The main applications table will now feature a new first column titled "Match Score", which will display the score visually.
- **File:** scholarbridge-client/src/pages/trust/TrustDashboard.jsx (Existing File)
- **Action:** Modify the component to use the new endpoint, include the toggle switch, and display the score.

```
import React, { useState, useEffect } from 'react';
```

```
import apiClient from '../../api/apiClient';

// A simple component to visualize the score
const MatchScoreBar = ({ score }) => {
  const color = score > 75 ? 'bg-green-500' : score > 50 ? 'bg-yellow-500' : 'bg-red-500';
  return (
    <div className="w-full bg-gray-200 rounded-full h-4" title={`Match: ${score}%`}>
      <div className={`${color} h-4 rounded-full text-xs text-white flex items-center
justify-center`} style={{ width: `${score}%` }}>
        {score > 10 && `${score}%`}
      </div>
    </div>
  );
};

const TrustDashboard = () => {
  const [applications, setApplications] = useState([]);
  const [loading, setLoading] = useState(true);
  const [showAll, setShowAll] = useState(false); // State for the toggle

  useEffect(() => {
    const fetchApplications = async () => {
      setLoading(true);
      try {
        // The URL now dynamically changes based on the toggle state
        const url = `/trusts/dashboard/applications?view=${showAll ? 'all' : 'filtered'}`;
        const response = await apiClient.get(url);
        setApplications(response.data);
      } catch (error) {
        console.error("Failed to fetch applications", error);
      } finally {
        setLoading(false);
      }
    };
    fetchApplications();
  }, [showAll]); // Re-fetch when the 'showAll' toggle changes

  return (
    <div className="p-6">
      <h1 className="text-2xl font-bold">Applicant Dashboard</h1>
```

```
        {/* The "Show All" Toggle Switch - Displayed at the top */}
        <div className="my-4 p-4 bg-blue-50 rounded-lg">
          <label className="flex items-center space-x-3 cursor-pointer">
            <input
              type="checkbox"
              checked={!showAll}
              onChange={() => setShowAll(prev => !prev)}
              className="h-4 w-4 rounded"
            />
            <span className="font-medium text-gray-700">Only show applications
matching my preferences</span>
          </label>
        </div>

        {/* Application Table */}
        {loading ? <p>Loading applications...</p> : (
          <div className="overflow-x-auto">
            <table className="min-w-full bg-white shadow-md rounded-lg">
              <thead className="bg-gray-100">
                <tr>
                  <th className="py-3 px-4 text-left w-48">Match Score</th>
                  <th className="py-3 px-4 text-left">Student Name</th>
                  <th className="py-3 px-4 text-left">Course</th>
                  <th className="py-3 px-4 text-left">City</th>
                  {/* ... other headers */}
                </tr>
              </thead>
              <tbody>
                {applications.map(app => (
                  <tr key={app.application_id} className="border-b hover:bg-gray-50">
                    <td className="px-4 py-2">
                      <MatchScoreBar score={Math.round(app.match_score)} />
                    </td>
                    <td className="px-4 py-2 font-medium">{app.full_name}</td>
                    <td className="px-4 py-2">{app.current_course_name}</td>
                    <td className="px-4 py-2">{app.city}</td>
                    {/* ... other data cells */}
                  </tr>
                ))}
              </tbody>
            </table>
```

```
            </div>
          )}
        </div>
    );
};

export default TrustDashboard;
```

4. The Research Paper Angle

With this implementation, you have a perfect foundation for a research paper.

- **Title Suggestion:** "A Weighted Heuristic Algorithm for Optimizing Funder-Applicant Matching on a Digital Philanthropic Platform."
- **Methodology:** Describe the finalized V2.0 scoring model in detail. Justify the weights based on the "Pyramid of Eligibility" concept.
- **Results:** After launching, you can analyze the data. Your key hypothesis to prove is: **"Does a higher Match Score correlate with a higher rate of funding approval?"** If you can show a positive correlation, your paper will have a strong, data-backed conclusion.

This roadmap provides a complete, end-to-end plan for delivering a sophisticated and high-impact feature.