Startup-Investor Platform: Vanilla PHP Architecture

Project Structure (Vanilla PHP)

```
startup-investor-platform/
├── public/
│   ├── index.php              # Main entry point with routing
│   ├── assets/
│   │   ├── css/
│   │   │   ├── main.css
│   │   │   ├── dashboard.css
│   │   │   ├── profiles.css
│   │   │   └── components.css
│   │   ├── js/
│   │   │   ├── main.js
│   │   │   ├── search.js
│   │   │   ├── messaging.js
│   │   │   ├── profiles.js
│   │   │   └── matching.js
│   │   └── images/
│   │       ├── logos/
│   │       ├── avatars/
│   │       └── icons/
│   ├── uploads/
│   │   ├── documents/
│   │   │   ├── pitch-decks/
│   │   │   ├── financials/
│   │   │   └── legal-docs/
│   │   ├── avatars/
│   │   └── company-logos/
│   └── .htaccess              # URL rewriting rules
├── src/
│   ├── Core/
│   │   ├── Application.php      # Main application class
│   │   ├── Router.php           # URL routing system
│   │   ├── Database.php         # Database connection & query builder
│   │   ├── Session.php          # Session management
│   │   ├── Security.php         # CSRF, validation, sanitization
│   │   ├── FileUpload.php       # Secure file handling
│   │   ├── Email.php            # Email sending system
│   │   └── Cache.php            # File-based caching system
│   ├── Models/
│   │   ├── BaseModel.php        # Base model with CRUD operations
│   │   ├── User.php             # User authentication & management
│   │   ├── Startup.php          # Startup profiles & data
│   │   ├── Investor.php         # Investor profiles & data
│   │   ├── Industry.php         # Industry categories
```

```
│   │   ├── Message.php          # Messaging system
│   │   ├── Match.php            # Matching algorithm results
│   │   ├── Document.php         # File management
│   │   └── Analytics.php        # User tracking & metrics
│   ├── Controllers/
│   │   ├── AuthController.php   # Login, register, logout
│   │   ├── DashboardController.php
│   │   ├── StartupController.php
│   │   ├── InvestorController.php
│   │   ├── SearchController.php
│   │   ├── MessageController.php
│   │   ├── MatchingController.php
│   │   ├── DocumentController.php
│   │   └── AdminController.php
│   ├── Services/
│   │   ├── MatchingService.php  # Core matching algorithm
│   │   ├── SearchService.php    # Advanced search & filtering
│   │   ├── NotificationService.php
│   │   ├── AnalyticsService.php
│   │   └── RecommendationService.php
│   ├── Views/
│   │   ├── layouts/
│   │   │   ├── main.php         # Main site layout
│   │   │   ├── auth.php         # Authentication layout
│   │   │   └── dashboard.php    # Dashboard layout
│   │   ├── auth/
│   │   │   ├── login.php
│   │   │   ├── register.php
│   │   │   ├── choose-type.php
│   │   │   └── verify-email.php
│   │   ├── dashboard/
│   │   │   ├── startup.php
│   │   │   ├── investor.php
│   │   │   └── admin.php
│   │   ├── profiles/
│   │   │   ├── startup/
│   │   │   │   ├── create.php
│   │   │   │   ├── edit.php
│   │   │   │   ├── view.php
│   │   │   │   └── public.php
│   │   │   └── investor/
│   │   │       ├── create.php
│   │   │       ├── edit.php
│   │   │       └── view.php
```

```
│   │   │   └── public.php
│   │   ├── search/
│   │   │   ├── startups.php
│   │   │   ├── investors.php
│   │   │   └── results.php
│   │   ├── messaging/
│   │   │   ├── inbox.php
│   │   │   ├── conversation.php
│   │   │   └── compose.php
│   │   ├── matching/
│   │   │   ├── recommendations.php
│   │   │   └── mutual-matches.php
│   │   └── components/
│   │       ├── header.php
│   │       ├── footer.php
│   │       ├── navigation.php
│   │       ├── startup-card.php
│   │       ├── investor-card.php
│   │       └── message-thread.php
│   └── Utils/
│       ├── Validator.php       # Input validation
│       ├── Sanitizer.php       # Data sanitization
│       ├── Helper.php          # Utility functions
│       ├── ImageProcessor.php  # Image handling & resizing
│       └── Logger.php          # Error & activity logging
├── config/
│   ├── config.php              # Main configuration
│   ├── database.php            # Database settings
│   ├── email.php               # Email configuration
│   └── app.php                 # Application settings
├── database/
│   ├── migrations/
│   │   ├── 001_create_users_table.sql
│   │   ├── 002_create_startups_table.sql
│   │   ├── 003_create_investors_table.sql
│   │   ├── 004_create_industries_table.sql
│   │   ├── 005_create_messages_table.sql
│   │   ├── 006_create_matches_table.sql
│   │   ├── 007_create_documents_table.sql
│   │   └── 008_create_analytics_table.sql
│   ├── seeds/
│   │   ├── industries.sql
│   │   └── demo_data.sql
│   └── schema.sql              # Complete database schema
```

```
├── storage/
│   ├── cache/              # File-based cache storage
│   ├── logs/               # Application logs
│   ├── sessions/           # Session files
│   └── temp/               # Temporary files
├── tests/
│   ├── unit/
│   │   ├── ModelTest.php
│   │   ├── ServiceTest.php
│   │   └── UtilTest.php
│   └── integration/
│       ├── AuthTest.php
│       ├── SearchTest.php
│       └── MatchingTest.php
├── scripts/
│   ├── migrate.php         # Database migration runner
│   ├── seed.php            # Data seeding script
│   ├── backup.php          # Database backup
│   └── cleanup.php         # File cleanup utilities
├── docs/
│   ├── API.md              # API documentation
│   ├── DATABASE.md         # Database schema docs
│   └── DEPLOYMENT.md       # Deployment guide
├── .env.example            # Environment template
├── .gitignore
├── composer.json           # For any external packages
├── README.md
└── phpunit.xml             # Testing configuration
```

Core Architecture Components

1. Application Entry Point (public/index.php)

php

```php
<?php
require_once '../src/Core/Application.php';

use Core\Application;
use Core\Router;

// Initialize application
$app = new Application();

// Set up routing
$router = new Router();

// Authentication routes
$router->get('/login', 'AuthController@showLogin');
$router->post('/login', 'AuthController@login');
$router->get('/register', 'AuthController@showRegister');
$router->post('/register', 'AuthController@register');
$router->get('/logout', 'AuthController@logout');

// Dashboard routes
$router->get('/dashboard', 'DashboardController@index');

// Profile routes
$router->get('/profile/create', 'ProfileController@create');
$router->post('/profile/store', 'ProfileController@store');
$router->get('/profile/edit', 'ProfileController@edit');
$router->post('/profile/update', 'ProfileController@update');

// Search routes
$router->get('/search/startups', 'SearchController@startups');
$router->get('/search/investors', 'SearchController@investors');
$router->post('/search/filter', 'SearchController@filter');

// API routes for AJAX
$router->post('/api/match', 'MatchingController@findMatches');
$router->post('/api/message', 'MessageController@send');
$router->get('/api/notifications', 'NotificationController@get');

// Handle the request
$router->dispatch($_SERVER['REQUEST_URI'], $_SERVER['REQUEST_METHOD']);
```

2. Core Database Class (src/Core/Database.php)

php

```php
<?php
namespace Core;

class Database {
    private static $instance = null;
    private $connection;
    private $host;
    private $username;
    private $password;
    private $database;

    private function __construct() {
        $config = require_once __DIR__ . '/../../config/database.php';
        $this->host = $config['host'];
        $this->username = $config['username'];
        $this->password = $config['password'];
        $this->database = $config['database'];

        $this->connect();
    }

    public static function getInstance() {
        if (self::$instance == null) {
            self::$instance = new Database();
        }
        return self::$instance;
    }

    private function connect() {
        try {
            $this->connection = new PDO(
                "mysql:host={$this->host};dbname={$this->database};charset=utf8mb4",
                $this->username,
                $this->password,
                [
                    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
                    PDO::ATTR_EMULATE_PREPARES => false,
                ]
            );
        } catch (PDOException $e) {
            throw new Exception("Database connection failed: " . $e->getMessage());
        }
```

```
    }

    public function query($sql, $params = []) {
        $stmt = $this->connection->prepare($sql);
        $stmt->execute($params);
        return $stmt;
    }

    public function fetch($sql, $params = []) {
        return $this->query($sql, $params)->fetch();
    }

    public function fetchAll($sql, $params = []) {
        return $this->query($sql, $params)->fetchAll();
    }

    public function insert($sql, $params = []) {
        $this->query($sql, $params);
        return $this->connection->lastInsertId();
    }

    public function update($sql, $params = []) {
        return $this->query($sql, $params)->rowCount();
    }

    public function delete($sql, $params = []) {
        return $this->query($sql, $params)->rowCount();
    }
}
```

3. Base Model Class (src/Models/BaseModel.php)

php

```php
<?php
namespace Models;

use Core\Database;

abstract class BaseModel {
    protected $db;
    protected $table;
    protected $primaryKey = 'id';

    public function __construct() {
        $this->db = Database::getInstance();
    }

    public function find($id) {
        $sql = "SELECT * FROM {$this->table} WHERE {$this->primaryKey} = ?";
        return $this->db->fetch($sql, [$id]);
    }

    public function findBy($column, $value) {
        $sql = "SELECT * FROM {$this->table} WHERE {$column} = ?";
        return $this->db->fetch($sql, [$value]);
    }

    public function all() {
        $sql = "SELECT * FROM {$this->table}";
        return $this->db->fetchAll($sql);
    }

    public function create($data) {
        $columns = implode(', ', array_keys($data));
        $placeholders = ':' . implode(', :', array_keys($data));

        $sql = "INSERT INTO {$this->table} ({$columns}) VALUES ({$placeholders})";

        return $this->db->insert($sql, $data);
    }

    public function update($id, $data) {
        $setClause = implode(' = ?, ', array_keys($data)) . ' = ?';
        $sql = "UPDATE {$this->table} SET {$setClause} WHERE {$this->primaryKey} = ?";

        $params = array_values($data);
```

```
        $params[] = $id;

        return $this->db->update($sql, $params);
    }

    public function delete($id) {
        $sql = "DELETE FROM {$this->table} WHERE {$this->primaryKey} = ?";
        return $this->db->delete($sql, [$id]);
    }

    public function where($conditions = [], $limit = null, $offset = null) {
        $sql = "SELECT * FROM {$this->table}";
        $params = [];

        if (!empty($conditions)) {
            $whereClause = [];
            foreach ($conditions as $column => $value) {
                $whereClause[] = "{$column} = ?";
                $params[] = $value;
            }
            $sql .= " WHERE " . implode(' AND ', $whereClause);
        }

        if ($limit) {
            $sql .= " LIMIT {$limit}";
            if ($offset) {
                $sql .= " OFFSET {$offset}";
            }
        }

        return $this->db->fetchAll($sql, $params);
    }
}
```

4. Matching Service (src/Services/MatchingService.php)

php

```php
<?php
namespace Services;

use Models\Startup;
use Models\Investor;
use Models\Match;

class MatchingService {
    private $startup;
    private $investor;
    private $match;

    public function __construct() {
        $this->startup = new Startup();
        $this->investor = new Investor();
        $this->match = new Match();
    }

    public function findMatchesForStartup($startupId) {
        $startup = $this->startup->find($startupId);
        if (!$startup) return [];

        // Get potential investor matches
        $sql = "
            SELECT i.*, u.first_name, u.last_name, u.email
            FROM investors i
            JOIN users u ON i.user_id = u.id
            WHERE JSON_CONTAINS(i.preferred_industries, ?)
            AND i.investment_range_min <= ?
            AND i.investment_range_max >= ?
            AND JSON_CONTAINS(i.investment_stages, ?)
            AND i.availability_status = 'actively_investing'
        ";

        $params = [
            json_encode([$startup['industry_id']]),
            $startup['funding_goal'],
            $startup['funding_goal'],
            json_encode([$startup['funding_type']])
        ];

        $potentialMatches = $this->investor->db->fetchAll($sql, $params);
```

```
    $matches = [];
    foreach ($potentialMatches as $investor) {
        $score = $this->calculateMatchScore($startup, $investor);
        if ($score > 60) { // Only show high-quality matches
            $matches[] = [
                'investor' => $investor,
                'score' => $score,
                'reasons' => $this->getMatchReasons($startup, $investor)
            ];
        }
    }

    // Sort by match score
    usort($matches, function($a, $b) {
        return $b['score'] <=> $a['score'];
    });

    return array_slice($matches, 0, 10); // Return top 10 matches
}

private function calculateMatchScore($startup, $investor) {
    $score = 0;

    // Industry match (30 points)
    $investorIndustries = json_decode($investor['preferred_industries'], true);
    if (in_array($startup['industry_id'], $investorIndustries)) {
        $score += 30;
    }

    // Stage match (25 points)
    $investorStages = json_decode($investor['investment_stages'], true);
    if (in_array($startup['stage'], $investorStages)) {
        $score += 25;
    }

    // Investment size match (20 points)
    if ($startup['funding_goal'] >= $investor['investment_range_min'] &&
        $startup['funding_goal'] <= $investor['investment_range_max']) {
        $score += 20;
    }

    // Geographic proximity (15 points)
    if ($this->sameRegion($startup['location'], $investor['location'])) {
        $score += 15;
```

```
    }

    // Track record relevance (10 points)
    $portfolioCompanies = json_decode($investor['portfolio_companies'], true) ?? [];
    foreach ($portfolioCompanies as $company) {
        if ($company['industry_id'] == $startup['industry_id']) {
            $score += 10;
            break;
        }
    }

    return min($score, 100); // Cap at 100
}

private function getMatchReasons($startup, $investor) {
    $reasons = [];

    $investorIndustries = json_decode($investor['preferred_industries'], true);
    if (in_array($startup['industry_id'], $investorIndustries)) {
        $reasons[] = 'Industry expertise match';
    }

    $investorStages = json_decode($investor['investment_stages'], true);
    if (in_array($startup['stage'], $investorStages)) {
        $reasons[] = 'Investment stage alignment';
    }

    if ($startup['funding_goal'] >= $investor['investment_range_min'] &&
        $startup['funding_goal'] <= $investor['investment_range_max']) {
        $reasons[] = 'Investment size fit';
    }

    if ($this->sameRegion($startup['location'], $investor['location'])) {
        $reasons[] = 'Geographic proximity';
    }

    return $reasons;
}

private function sameRegion($location1, $location2) {
    // Simple region matching - can be enhanced
    $region1 = explode(',', $location1)[1] ?? '';
    $region2 = explode(',', $location2)[1] ?? '';
    return trim($region1) === trim($region2);
```

```
    }
}
```

Key Database Tables

Users Table

```sql
CREATE TABLE users (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    user_type ENUM('startup', 'investor') NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    phone VARCHAR(20),
    location VARCHAR(255),
    email_verified_at TIMESTAMP NULL,
    profile_completed BOOLEAN DEFAULT FALSE,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    INDEX idx_email (email),
    INDEX idx_user_type (user_type),
    INDEX idx_location (location)
);
```

Startups Table

sql

```sql
CREATE TABLE startups (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    user_id INT UNSIGNED NOT NULL,
    company_name VARCHAR(255) NOT NULL,
    slug VARCHAR(255) UNIQUE NOT NULL,
    description TEXT,
    industry_id INT UNSIGNED,
    stage ENUM('idea', 'prototype', 'mvp', 'early_revenue', 'growth') NOT NULL,
    employee_count ENUM('1', '2-5', '6-10', '11-25', '26-50', '51+'),
    website VARCHAR(255),
    logo_url VARCHAR(255),
    pitch_deck_url VARCHAR(255),
    funding_goal DECIMAL(15,2),
    funding_type ENUM('seed', 'series_a', 'series_b', 'debt', 'grant'),
    location VARCHAR(255),
    is_featured BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    INDEX idx_industry (industry_id),
    INDEX idx_stage (stage),
    INDEX idx_location (location),
    INDEX idx_funding_goal (funding_goal),
    FULLTEXT idx_search (company_name, description)
);
```

Performance Optimizations Built-In

1. Database Indexing Strategy

- **Composite indexes** for common query patterns
- **Full-text search** indexes for company/description searches
- **Foreign key indexes** for join performance
- **Geographic indexes** for location-based matching

2. Caching Layer

- **File-based caching** for database query results
- **Template caching** for rendered views
- **Asset optimization** with minification and compression

- **Image caching** with automatic resizing

3. Query Optimization

- **Prepared statements** to prevent SQL injection and improve performance
- **Connection pooling** through persistent connections
- **Lazy loading** for related data
- **Query result pagination** for large datasets

4. Security Features

- **CSRF protection** for all forms
- **Input validation** and sanitization
- **Secure file uploads** with type and size restrictions
- **Password hashing** with PHP's password_hash()
- **Session security** with secure cookie settings

This vanilla PHP architecture gives you complete control while building in performance optimizations and security from day one. You can start building immediately and scale efficiently as your user base grows.