Capstone Project Phase B

Final Projects Portal: Topics, Partners, Supervisors and more

Project Code: 25-2-D-7

Supervisor:
Dr. Julia Sheidin

Students:
Eldar Gafarov - Eldar.Gafarov@e.braude.ac.il
Roee Bar - Roee.Bar@e.braude.ac.il

Git Repository:
https://github.com/MentorMatch

Table of contents

1. Problem Overview and Background

Final-year projects are a crucial component of engineering education at Braude College, allowing students to apply theoretical knowledge to real-world challenges. A critical phase in this process is the initial connection between students and project supervisors, which establishes the foundation for successful project completion.

1.1 Problem Statement

The current process of connecting students with project supervisors at Braude College is characterized by disorganized communication, limited information visibility, and inefficient workflows. Students struggle to identify available supervisors, understand their research interests, and determine their capacity to take on new projects. This fragmented approach relies heavily on email exchanges and static Excel spreadsheets, creating several critical problems:

- **Limited Visibility:** Students lack centralized information about available supervisors, their expertise areas, and current capacity.
- **Communication Inefficiency:** Both students and supervisors spend excessive time on repetitive email exchanges.
- **Administrative Burden:** Project coordinators manually track supervisor capacities and student assignments using spreadsheets.
- **Missed Opportunities:** Optimal matches between student interests and supervisor expertise are often overlooked.

1.2 Key Stakeholders

The key stakeholders are presented in Table 1.

Stakeholder	Role
Students	Responsible for selecting project topics, conducting research, and developing solutions. Primary users of the supervisor discovery and application features.
Supervisors	Faculty members who mentor, review proposals, and assess project work. Use the system to manage their profiles and review student applications.
Administrative Staff & Project Coordinators	Manage registration for Supervisors, handle administrative tasks, and ensure accurate tracking of supervision assignments. Oversee the entire project process, monitor matching progress, and ensure all students find appropriate supervision.

Table 1: Key Stakeholders and Their Roles

2. General Description

2.1 System Goals and Objectives

MentorMatch is a web-based platform designed to streamline the supervisor-student matching process for capstone projects at Braude College. The system extends the existing ProjectHub platform by addressing the critical first phase of the project lifecycle: Project Selection and Initiation.

Primary Objectives:

- Provide a centralized platform where students can discover available supervisors and their research interests
- Enable real-time visibility into supervisor capacity and availability
- Standardize the application process with structured forms and automated notifications
- Reduce administrative burden through automated tracking and reporting
- Eliminate double-booking conflicts through formal commitment mechanisms

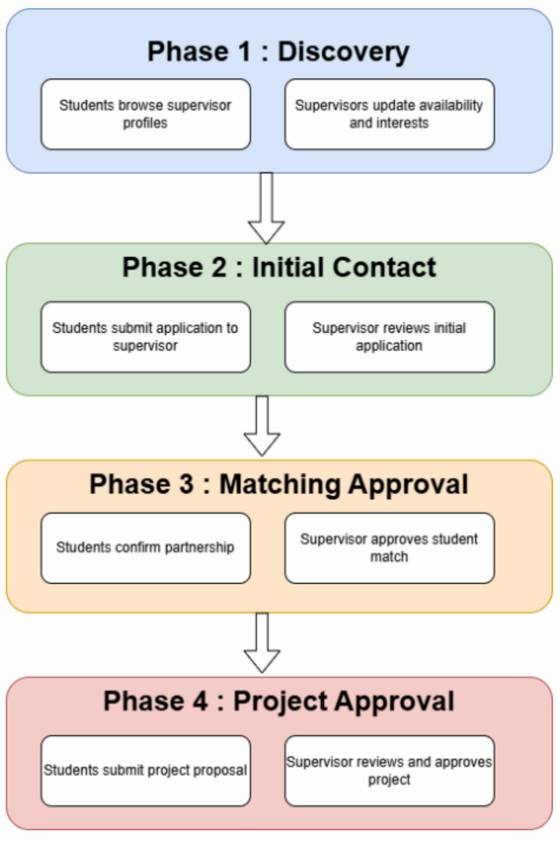
2.2 Target Users

The MentorMatch platform serves three primary user groups, each with role-specific interfaces and functionality:

- **Students:** Fourth-year engineering students seeking project supervisors. They can pair together, browse supervisor profiles, view capacity and research interests, and submit applications through a structured portal.
- **Supervisors:** Faculty members who mentor capstone projects. They manage their profiles, set capacity limits, review applications, and approve or decline student requests.
- **Administrators/Coordinators:** Project coordinators and administrative staff who oversee the matching process, monitor system-wide metrics, and manage supervisor capacities.

2.3 Process Workflow

The MentorMatch system implements a structured four-phase workflow for the student-supervisor matching process (Fig. 1):



Phase 1 : Discovery
Students browse supervisor profiles
Supervisors update availability and interests
Phase 2 : Initial Contact
Students submit application to supervisor
Supervisor reviews initial application
Phase 3 : Matching Approval
Students confirm partnership
Supervisor approves student match
Phase 4 : Project Approval
Students submit project proposal
Supervisor reviews and approves project

Figure 1: MentorMatch Process Workflow

3. Solution Description

3.1 System Architecture

MentorMatch follows a three-tier architecture, separating concerns between the presentation layer, business logic, and data storage (see Fig. 2). This modular design ensures maintainability, scalability, and clear separation of responsibilities.

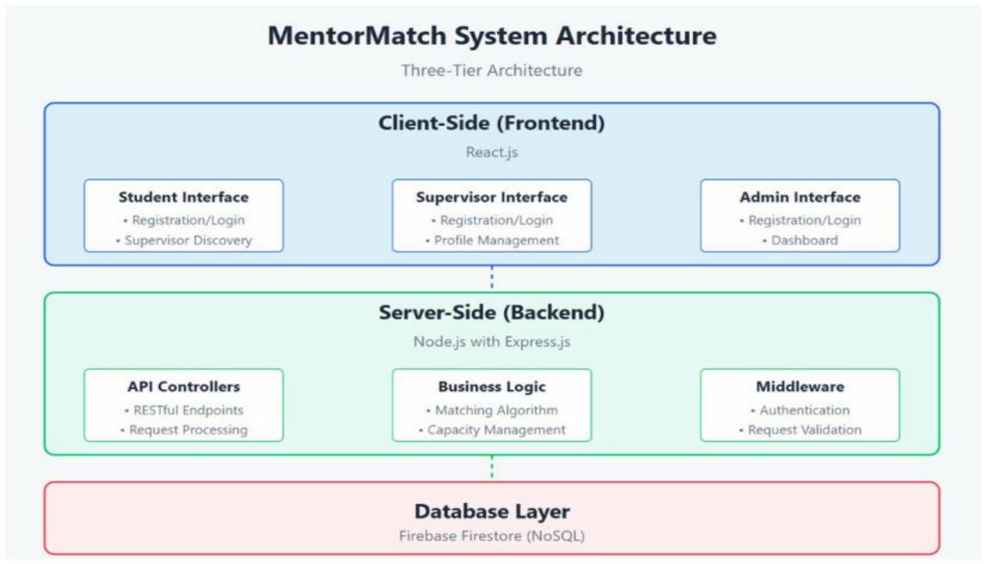


Figure 2: MentorMatch System Architecture

3.1.1 Architecture Layers:

- **Client-Side (Frontend):** Built with React.js and Next.js, providing three distinct interfaces for students, supervisors, and administrators. The frontend handles user interactions, state management, and communicates with the backend through RESTful APIs.
- **Server-Side (Backend):** Implemented using Node.js with Express.js, managing API controllers for RESTful endpoints, business logic for the matching algorithm and capacity management, and middleware for authentication and request validation.
- **Database Layer:** Firebase Firestore (NoSQL) serves as the primary database, storing user profiles, project applications, and matching data. Firebase Authentication handles user identity management.

3.2 Use Case Diagram

The use case diagram (Fig. 3) illustrates the primary interactions between system actors and the MentorMatch platform:

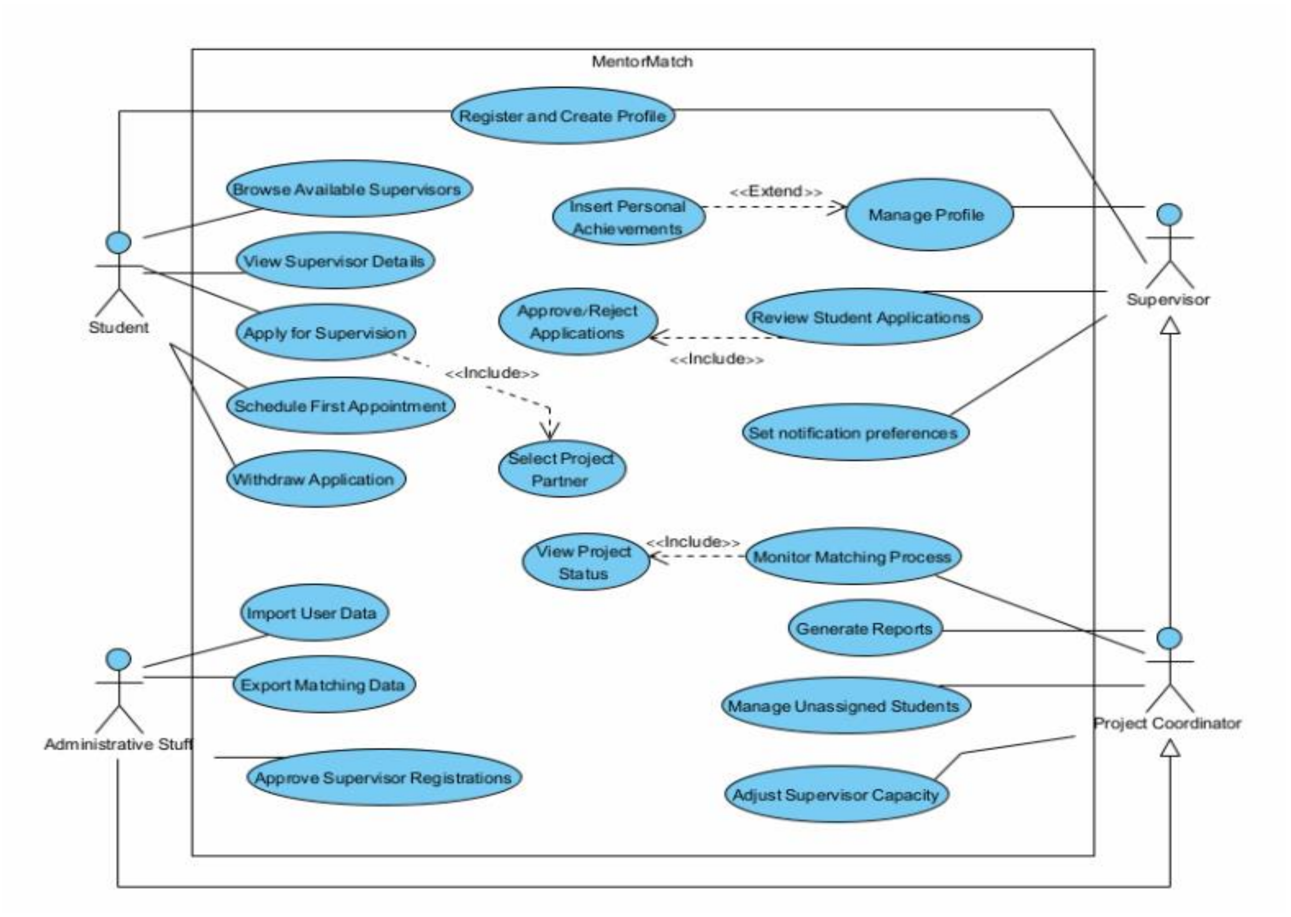


Figure 3: MentorMatch Use Case Diagram

3.3 Key Features Implemented

The following features were successfully implemented during Phase B development:

3.3.1 Student Features:

- User registration with email verification using Resend API
- Browsing available supervisors with filtering by department and research interests
- Viewing supervisor profiles with capacity status and expertise areas
- Submitting applications as a group (with project partners)
- Tracking application status in real-time

3.3.2 Supervisor Features:

- Dashboard with supervision overview (total applications, pending reviews, capacity status)

- Profile management with research interests and expertise tags
- Application review interface with approve/revise/decline functionality

3.3.3 Administrator Features:

- System-wide dashboard with key metrics (total students, unassigned students, active supervisors)
- Supervisor capacity management table with real-time availability tracking
- Overview of approved projects and pending applications
- Monitoring of matching progress across all supervisors

4. Development Process

4.1 Methodology

The development of MentorMatch followed a User-Centered Design (UCD) methodology, which prioritizes understanding user needs and incorporating feedback into iterative refinements. This approach was established during Phase A through stakeholder interviews and was continued throughout Phase B implementation. During mid-development, we presented an early prototype to Naomi, one of the project coordinators, to validate our design decisions and ensure the system addresses real user needs. Her feedback guided interface refinements before full implementation.

Development was conducted over the course of one semester, with the work divided between two team members. The initial division had one team member (Eldar) focusing on database architecture and understanding the outer API’s that should be used, while the other (Roee) handled frontend and backend. During the final weeks, significant refactoring was required to integrate and stabilize the codebase.

4.2 Tools and Technologies

The tools and technologies used in our project are presented in Table 2.

Category	**Technologies Used**
Frontend	Next.js, React.js, TypeScript, Tailwind CSS
Backend	Node.js, Express.js, RESTful APIs
Database	Firebase Firestore (NoSQL), Firebase Authentication
Email Service	Resend API (for email verification and notifications)
Deployment	Vercel (frontend hosting and serverless functions)
Version Control	Git, GitHub

Table 2: Development Tools and Technologies

4.3 Stakeholder Interaction

Throughout the development process, we maintained regular communication with stakeholders to gather feedback and validate our implementation. Weekly meetings with our supervisor, Dr. Julia Sheidin, who also serves as a project coordinator, provided continuous guidance and helped prioritize features based on real-world needs.

Additional feedback was collected from faculty members who reviewed the system and suggested enhancements such as displaying lecturer availability hours and adding research/development filters. Students also tested the platform and provided valuable usability feedback that informed interface improvements.

5. Challenges and Solutions

During the development of MentorMatch, we encountered several technical and process-related challenges that required creative solutions and adaptations.

5.1 Technical And Process Challenges

Challenge 1: Code Overload and Extensive Refactoring

During early development, we implemented numerous features and infrastructure components without fully evaluating their necessity. This included extensive end-to-end (E2E) testing frameworks, Kubernetes deployment configurations, and other advanced tooling. While these additions seemed valuable at the time, they significantly expanded the codebase complexity without contributing to the core user experience. The accumulated code became difficult to maintain and integrate, creating technical debt that slowed development progress.

Solution: We conducted a thorough code review and made the difficult decision to remove features that did not directly serve our stakeholders' needs. This refactoring effort, while time-consuming, resulted in a cleaner, more maintainable codebase.

Challenge 2: Shared Account Access Limitations

Several external services used in the project, including Resend and Vercel, were registered under a single team member's account. This meant only one developer had direct access to deployment pipelines, email service configurations, and environment variables. When that team member was unavailable or when urgent changes were needed, the other developer could not independently deploy updates or troubleshoot service-related issues. This bottleneck created delays and dependency on one person for critical operations.

Solution: We established a workflow where API keys and configuration details were shared securely between team members, allowing both to work with local development environments. For deployments, we coordinated schedules to ensure the account holder was available during critical release periods. For future projects, we would recommend creating shared team accounts from the start or using platforms that support multiple team members on free tiers, ensuring both developers have equal access to all project infrastructure.

Challenge 3: Code Integration and Quality

With two team members working on different parts of the system (database/backend and frontend), integrating the code revealed inconsistencies and bugs that required substantial debugging and refactoring in the final weeks of development.

Solution: We dedicated the final two weeks to comprehensive code review and bug fixing. Moving forward, more frequent integration and code reviews would prevent such accumulation of technical debt.

Challenge 4: Firebase Email Service Limitations

The initial plan was to use Firebase's built-in email service for user verification and notifications. However, we encountered two significant issues. First, Firebase's email functionality did not meet our requirements for customized email templates and reliable delivery. Second, Firebase required credit card registration even for the free tier, as a safeguard against exceeding usage limits. As students working on an academic project, we preferred to avoid providing payment information for a service that might incur unexpected charges.

Solution: We integrated the Resend API as our email service provider. Resend offers a free tier with limits of 100 emails per day and 3,000 emails per month, which is more than sufficient for our academic project's scope. Unlike Firebase, Resend does not require credit card registration and provides better control over email templates, delivery tracking, and error handling. This switch improved our email verification flow significantly while eliminating concerns about potential billing issues.

Challenge 5: Firebase Realtime Database Storage Costs

During initial development, we explored storing user profile photos in Firebase. However, we quickly encountered limitations with the free tier storage quotas, which would have been exceeded with even moderate usage.

Solution: We decided to defer the photo upload feature and focus on core functionality. The system currently operates without profile photos, which can be added in future iterations when proper storage infrastructure is budgeted.

Challenge 6: Time Management and Academic Workload

Balancing capstone project development with other academic courses and personal commitments created scheduling difficulties. Some weeks had limited progress due to exams or assignments in other subjects.

Solution: We established a weekly meeting schedule with our supervisor to maintain accountability and track progress. Breaking the project into smaller milestones helped us make consistent progress even during busier periods.

6. Results and Conclusions

6.1 Project Outcomes

The MentorMatch system was successfully developed and deployed as a functional web application that addresses the core challenges identified during Phase A. The platform provides a centralized solution for the student-supervisor matching process at Braude College.

Achieved Objectives:

- Centralized supervisor profiles with real-time capacity visibility
- Structured student application process eliminating duplicate email inquiries
- Administrative dashboard providing system-wide oversight
- Formal commitment mechanism preventing double-booking

6.2 Project Metrics Evaluation

In Phase A, we defined several success metrics for the MentorMatch system. To evaluate our performance against these metrics, we conducted live demonstrations with individual students and supervisors, gathering their feedback on the system's usability and value.

- **Reduce search time for supervisors:** Achieved during live demos, students confirmed that browsing supervisor profiles with real-time capacity information eliminates the need to send multiple inquiry emails.
- **User satisfaction:** Achieved feedback collected from both students and supervisors during demonstrations was consistently positive. Users appreciated the centralized platform, clear visibility of availability, and structured application process.
- **Decrease unmatched students by registration deadline:** Not directly measured as the system was not deployed during a full academic cycle, this metric could not be quantitatively evaluated. However, the platform provides all necessary tools to support this goal.
- **Task management efficiency:** Achieved, supervisors confirmed the system would reduce email overhead,

6.3 User Feedback

The system was tested through two main approaches: prototype evaluation during development and hands-on user testing of the final system.

During mid-development, we presented an early prototype to Dr.Naomi, one of the project coordinators, to validate our design decisions. Her feedback helped us refine the interface and ensure the system addresses real administrative needs.

For final system testing, we conducted individual sessions with students and faculty members. Each participant interacted with the platform by performing tasks relevant to their role students browsed supervisors and submitted applications, while supervisors reviewed their dashboard and managed applications. After using the system, we collected their feedback:

Students Feedback:

"Finally I can see which supervisors are available and what topics interest them."

"Seeing supervisor capacity in real-time means I can stop wasting time on supervisors who are already full."

Supervisors Feedback:

"This will finally reduce the email spam, no more duplicate inquiries from students who already found someone else."

"Students can see my availability upfront, so I receive more relevant inquiries."

6.4 Features Not Implemented

Due to time constraints and the refactoring required during development, some features from the original Phase A requirements were not implemented:

- **Supervisor project suggestions:** The feature allowing supervisors to post specific project topics for students to choose from was not implemented. Currently, students propose their own project ideas in applications.
- **Profile photos:** User profile photo uploads were deferred due to Firebase storage cost considerations.
- **Excel import/export:** The ability to import student data from external systems (Gilboa) and export matching data was not completed.

These features are documented for future development iterations.

7. Lessons Learned

The development of MentorMatch provided valuable insights into software engineering practices and project management. The following lessons emerged from our experience:

7.1 Development Approach

Focus on Incremental Development:

The most significant lesson was the importance of building features step by step rather than attempting to implement everything simultaneously. Our initial approach of adding many features in parallel led to integration issues and technical debt. A more incremental approach would have allowed for earlier detection of problems and smoother integration.

Validate Feature Necessity:

Significant time was spent on features like E2E testing infrastructure and Kubernetes deployment that did not contribute to the core user experience. Before implementing any feature, we should have asked: "Does this directly address a stakeholder need?" This validation step would have prevented wasted effort.

7.2 Team Collaboration

Frequent Integration:

Working separately on frontend and backend components created integration challenges. More frequent integration points and code reviews throughout development would have identified issues earlier. In future projects, we would establish daily or every-other-day integration practices.

Clear Communication:

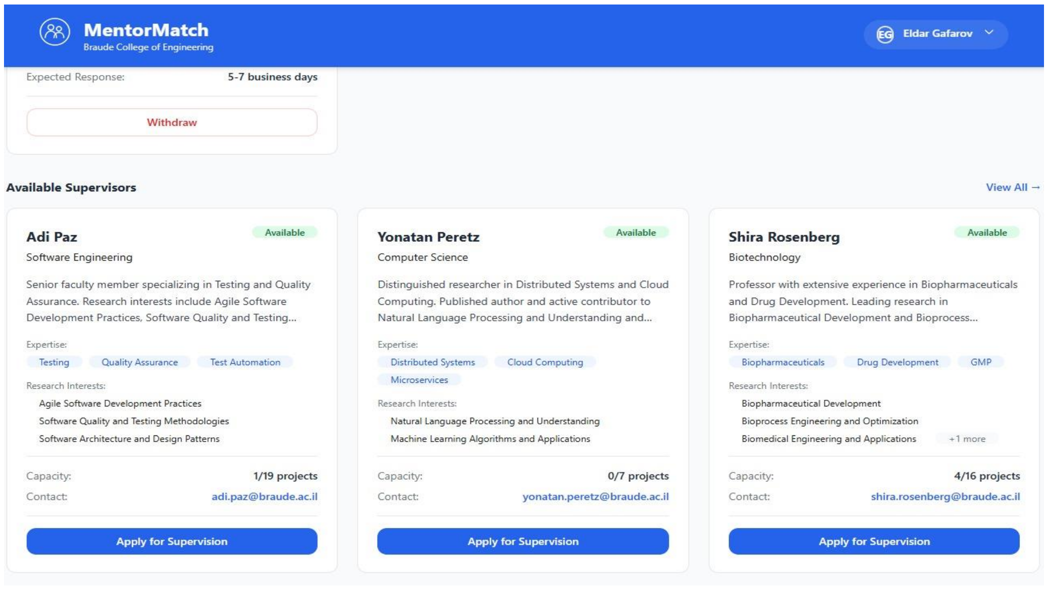
Regular communication about progress, blockers, and design decisions is essential. When team members work on separate components, assumptions can diverge. Establishing regular sync meetings and shared documentation would improve alignment.

7.3 What We Would Do Differently

- **Integrate early and often:** Merge code frequently to catch integration issues before they compound.
- **Prioritize ruthlessly:** Focus on features that directly serve users, not technical "nice-to-haves."
- **Plan for refactoring:** Allocate time in the schedule for code cleanup and technical debt management.

8. Application Screenshots

8.1 Student Interface - Available Supervisors


MentorMatch
Braude College of Engineering
EG Eldar Gafarov

Expected Response: 5-7 business days
Withdraw

Available Supervisors — View All →

Adi Paz — Available
Software Engineering
Senior faculty member specializing in Testing and Quality Assurance. Research interests include Agile Software Development Practices, Software Quality and Testing...
Expertise: Testing, Quality Assurance, Test Automation
Research Interests:
Agile Software Development Practices
Software Quality and Testing Methodologies
Software Architecture and Design Patterns
Capacity: 1/19 projects
Contact: adi.paz@braude.ac.il
Apply for Supervision

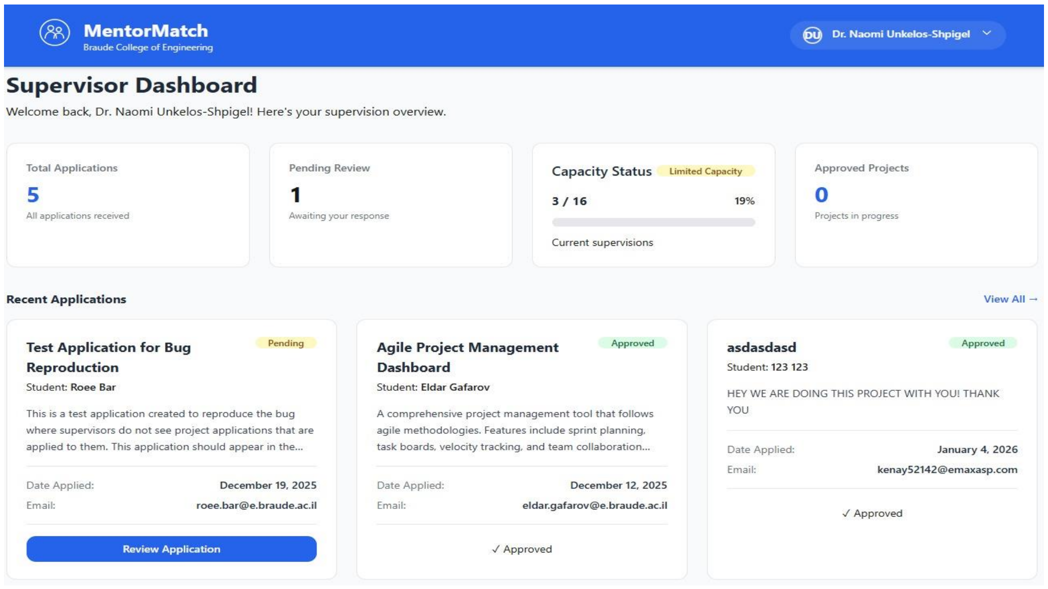
Yonatan Peretz — Available
Computer Science
Distinguished researcher in Distributed Systems and Cloud Computing. Published author and active contributor to Natural Language Processing and Understanding and...
Expertise: Distributed Systems, Cloud Computing, Microservices
Research Interests:
Natural Language Processing and Understanding
Machine Learning Algorithms and Applications
Capacity: 0/7 projects
Contact: yonatan.peretz@braude.ac.il
Apply for Supervision

Shira Rosenberg — Available
Biotechnology
Professor with extensive experience in Biopharmaceuticals and Drug Development. Leading research in Biopharmaceutical Development and Bioprocess...
Expertise: Biopharmaceuticals, Drug Development, GMP
Research Interests:
Biopharmaceutical Development
Bioprocess Engineering and Optimization
Biomedical Engineering and Applications +1 more
Capacity: 4/16 projects
Contact: shira.rosenberg@braude.ac.il
Apply for Supervision

Figure 4: Student View - Browsing Available Supervisors

Students can browse supervisor profiles displayed as cards showing name, department, expertise tags, research interests, current capacity, and contact information. The "Apply for Supervision" button initiates the application process.

8.2 Supervisor Dashboard


MentorMatch
Braude College of Engineering
DU Dr. Naomi Unkelos-Shpigel

Supervisor Dashboard
Welcome back, Dr. Naomi Unkelos-Shpigel! Here's your supervision overview.

Total Applications: 5 — All applications received
Pending Review: 1 — Awaiting your response
Capacity Status (Limited Capacity): 3 / 16 — 19% — Current supervisions
Approved Projects: 0 — Projects in progress

Recent Applications — View All →

Test Application for Bug Reproduction — Pending
Student: Roee Bar
This is a test application created to reproduce the bug where supervisors do not see project applications that are applied to them. This application should appear in the...
Date Applied: December 19, 2025
Email: roee.bar@e.braude.ac.il
Review Application

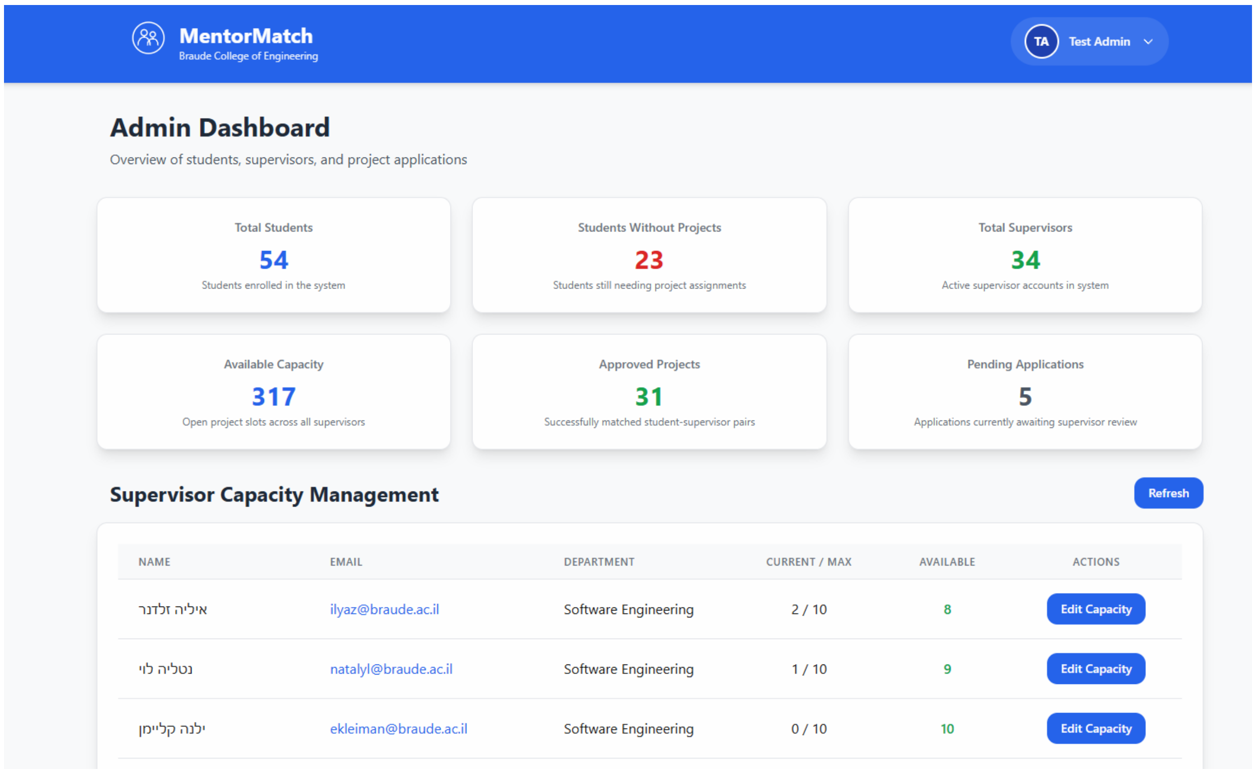
Agile Project Management Dashboard — Approved
Student: Eldar Gafarov
A comprehensive project management tool that follows agile methodologies. Features include sprint planning, task boards, velocity tracking, and team collaboration...
Date Applied: December 12, 2025
Email: eldar.gafarov@e.braude.ac.il
✓ Approved

asdasdasd — Approved
Student: 123 123
HEY WE ARE DOING THIS PROJECT WITH YOU! THANK YOU
Date Applied: January 4, 2026
Email: kenay52142@emaxasp.com
✓ Approved

Figure 5: Supervisor Dashboard - Application Management

The supervisor dashboard displays key metrics including total applications, pending reviews, capacity status with a visual progress bar, and approved projects. Recent applications are shown with student names, project titles, dates, and status badges (Pending/Approved).

8.3 Admin Dashboard



Admin Dashboard

Overview of students, supervisors, and project applications

Total Students	Students Without Projects	Total Supervisors
54	23	34
Students enrolled in the system	Students still needing project assignments	Active supervisor accounts in system

Available Capacity	Approved Projects	Pending Applications
317	31	5
Open project slots across all supervisors	Successfully matched student-supervisor pairs	Applications currently awaiting supervisor review

Supervisor Capacity Management

NAME	EMAIL	DEPARTMENT	CURRENT / MAX	AVAILABLE	ACTIONS
איליה זלדנר	ilyaz@braude.ac.il	Software Engineering	2 / 10	8	Edit Capacity
נטליה לוי	natalyl@braude.ac.il	Software Engineering	1 / 10	9	Edit Capacity
ילנה קליימן	ekleiman@braude.ac.il	Software Engineering	0 / 10	10	Edit Capacity

Figure 6: Admin Dashboard - System Overview

The admin dashboard provides system-wide metrics including total enrolled students, students without projects, active supervisors, available capacity across all supervisors, approved projects, and pending applications. The Supervisor Capacity Management table allows administrators to view and edit individual supervisor capacities.

9. Appendix A: User Guide

9.1 Introduction

This guide provides instructions for using the MentorMatch platform. The system is designed to facilitate the student-supervisor matching process for capstone projects at Braude College.

9.2 Getting Started

9.2.1 Accessing the Platform:

Navigate to the MentorMatch URL in your web browser. The system is optimized for modern browsers including Chrome, Firefox, Safari, and Edge.

9.3 Student Operations

9.3.1 Registration:

1. Click the "Register" button on the home page
2. Enter your Braude email address and create a password
3. Fill in your profile information (name, department, academic year)
4. Submit the registration form
5. Check your email for a verification link and click it to activate your account

Browsing Supervisors:

After logging in, the main dashboard displays available students to pair with and afterwards supervisors. Each supervisor card shows their name, department, research interests, expertise tags, and current capacity. Supervisors marked "Available" have open slots for new projects.

Viewing Supervisor Details:

Click on a supervisor card to view their full profile, including detailed research interests, contact information, and current supervision capacity.

Applying for Supervision:

1. Click "Apply for Supervision" on a supervisor's card
2. Fill in your project idea and relevant skills
3. Submit the application

Tracking Applications:

Your submitted applications appear in the dashboard with status indicators: Pending (awaiting review), Approved (accepted by supervisor), or Declined.

9.4 Supervisor Operations

Supervisor Test Credentials: Username : ilyaz@braude.ac.il , Password : Test123!

Dashboard Overview:

The supervisor dashboard displays metrics for Total Applications, Pending Reviews, Capacity Status, and Approved Projects. Recent applications are listed below with student information and review options.

Reviewing Applications:

1. Click "Review Application" on a pending application
2. Review the student's project proposal and qualifications
3. Click "Approve" to accept the student or "Decline" to reject or “Revise” to revise the proposal.

Managing Profile:

Access your profile through the user menu to update research interests, expertise tags, and supervision capacity.

9.5 Administrator Operations

Admin Test Credentials: Username : admin@braude.ac.il , Password : Test123!

System Monitoring:

The admin dashboard provides an overview of system-wide metrics, including total students, unassigned students, active supervisors, available capacity, approved projects, and pending applications.

Capacity Management:

The Supervisor Capacity Management table displays all supervisors with their current and maximum capacity. Use the "Edit Capacity" button to adjust a supervisor's maximum project limit.

10. Appendix B: Maintenance Guide

10.1 Technology Stack

Technology	Purpose	Version
Next.js	Full-stack React framework	14.2.33
React	UI library	18.3.1
TypeScript	Type-safe JavaScript	5.x
TailwindCSS	Utility-first CSS framework	3.4.1
Firebase	User authentication	12.6.0
Firestore	NoSQL database	12.6.0
Firebase Admin	Server-side Firebase SDK	13.6.0
Zod	Schema validation	4.1.13
Vercel	Hosting and deployment	Latest

10.2 Development Environment Setup

10.2.1 Prerequisites

- **Node.js:** Version 20 or above
- **npm or yarn:** For dependency management\
- **Firebase CLI:** `npm install -g firebase-tools`
- **Git:** For version control

10.2.2 Installing Dependencies

Clone the repository and install dependencies:

```
git clone https://github.com/Roee-Bar/MentorMatch.git
cd MentorMatch
npm install
```

10.2.3 Environment Variables

Create a .env.local file in the project root:

```
# Firebase Client Configuration (Public)
NEXT_PUBLIC_FIREBASE_API_KEY=your_api_key_here
NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN=your_auth_domain
NEXT_PUBLIC_FIREBASE_PROJECT_ID=your_project_id
NEXT_PUBLIC_FIREBASE_STORAGE_BUCKET=your_storage_bucket
NEXT_PUBLIC_FIREBASE_MESSAGING_SENDER_ID=your_sender_id
NEXT_PUBLIC_FIREBASE_APP_ID=your_app_id

# Firebase Admin Configuration (Server-side only)
FIREBASE_ADMIN_PROJECT_ID=your_project_id
FIREBASE_ADMIN_CLIENT_EMAIL=your_service_account_email
FIREBASE_ADMIN_PRIVATE_KEY="-----BEGIN PRIVATE KEY-----\n...\n-----END PRIVATE KEY----
-\n"
```

10.2.4 Running the Development Server

```
npm run dev           # Start development server on http://localhost:3000
npm run build         # Create production build
npm start             # Start production server
npm run lint          # Run ESLint
npm run type-check    # TypeScript type checking
```

10.2.5 Project Structure

```
mentormatch/
├── app/
│   ├── api/                    # Next.js API routes (28 files)
│   │   └── [admin, applications, auth, partnerships, projects, students, supervisors,
users]
│   ├── authenticated/          # Protected routes (admin, student, supervisor
dashboards)
│   ├── components/             # Reusable UI components (display, feedback, form,
layout, shared)
│   ├── login/, register/, verify-email/
│   ├── page.tsx, layout.tsx, globals.css
├── lib/
│   ├── middleware/             # Auth, authorization, error handling, validation (6
files)
│   ├── services/               # Business logic layer (18 files)
│   │   └── [admin, applications, partnerships, projects, students, supervisors,
users, email]
│   ├── hooks/                  # React custom hooks (17 files)
│   ├── styles/                 # Shared styles and utilities
│   ├── utils/                  # Utility functions
│   ├── auth.ts, firebase.ts, firebase-admin.ts, logger.ts
│   └── constants.ts, routes.ts, auth-errors.ts
├── types/                      # TypeScript definitions (api.ts, database.ts)
├── scripts/                    # Data migration and admin scripts (13 files)
├── docs/                       # Documentation
├── public/                     # Static assets
├── firestore.rules             # Security rules
├── firestore.indexes.json      # Database indexes
├── vercel.json                 # Deployment config
├── next.config.js, tailwind.config.ts, tsconfig.json
└── package.json, README.md
```

10.3 Backend Architecture

10.3.1 API Routes Structure

All API routes are located in app/api/ and follow Next.js 14 App Router conventions.

10.3.2 Core API Endpoints

Endpoint	Method	Role
Authentication		
/api/auth/register	POST	-
/api/auth/verify-email	POST	-
Users		
/api/users	GET	Admin
/api/users/[id]	GET	Self/Admin
/api/users/[id]	PUT	Self/Admin
Students		
/api/students	GET	All authenticated
/api/students/[id]	GET	All authenticated
/api/students/[id]	PUT	Self/Admin
/api/students/[id]/applications	GET	Self/Admin/Supervisor
/api/students/[id]/partnership-requests	GET	Self/Admin/Supervisor
/api/students/available-partners	GET	Student
/api/students/unmatched	GET	Admin
Supervisors		
/api/supervisors	GET	All authenticated
/api/supervisors/[id]	GET	All authenticated
/api/supervisors/[id]	PUT	Self/Admin
/api/supervisors/[id]/projects	GET	All authenticated
Projects		
/api/projects	GET	All authenticated
/api/projects	POST	Admin
/api/projects/[id]	GET	All authenticated

Applications		
/api/applications	GET	Admin
/api/applications	POST	All authenticated
/api/applications/[id]	GET	Involved parties
/api/applications/[id]	PUT	Involved parties
/api/applications/[id]	DELETE	Involved parties
/api/applications/[id]/status	PATCH	Supervisor/Admin
/api/applications/[id]/resubmit	POST	Student/Admin
Partnerships		
/api/partnerships/request	POST	All authenticated
/api/partnerships/[id]/respond	POST	Target student
/api/partnerships/[id]	DELETE	Involved parties/Admin
/api/partnerships/unpair	POST	All authenticated
Admin		
/api/admin/stats	GET	Admin
/api/admin/fix-match-status	POST	Admin
/api/admin/supervisors	GET	Admin
/api/admin/supervisors/[id]/capacity	PATCH	Admin

10.4 Frontend Architecture

10.4.1 Next.js App Router

MentorMatch uses Next.js 14 App Router with the following structure:

- **Server Components: Default for most pages**
- **Client Components: Used for interactive UI (marked with `'use client'`)**
- **Layouts:** Shared layouts for authenticated routes

10.4.2 Authenticated Routes

Protected routes are wrapped in app/authenticated/layout.tsx:

```
// Checks authentication status
// Redirects to login if not authenticated
// Implements role-based routing (student → /student, supervisor → /supervisor, etc.)
```

10.4.3 Component Organization

Components follow a hierarchical structure:

- **Pages:**Top-level routes in app/
- **Layout:** Shared layouts in app/components/layout/
- **Display:**Data display in app/components/display/
- **Form:** Input components in app/components/form/
- **Feedback:**Loading/error states in app/components/feedback/
- **Shared:** Reusable utilities in app/components/shared/

10.4.4 State Management

- **Server State:** Fetched via API, cached by Next.js
- **Client State:** React hooks
- **Custom Hooks:** In lib/hooks/ for reusable logic
- **Auth State:** Firebase Auth + useAuth hook

10.5 Database Architecture

10.5.1 Firestore Collections

Collection	Purpose	Key Fields
users	User profiles and roles	**id (PK),** email, name, role, department, emailVerified, createdAt
students	Student-specific data	**id (PK),** email, firstName, lastName, skills[], **partnerId (FK → students)**, partnershipStatus, matchStatus, **assignedSupervisorId (FK → supervisors)**
supervisors	Supervisor-specific data	**id (PK),** email, firstName, lastName, expertiseAreas[], researchInterests[], maxCapacity, currentCapacity, availabilityStatus, isActive, isApproved
projects	Projects	**id (PK),** projectCode, title, description, **studentIds[] (FK → students), supervisorId (FK → supervisors),** status, phase
applications	Student applications	**id (PK), studentId (FK → students), supervisorId (FK → supervisors)**, projectTitle, projectDescription, status, hasPartner, **linkedApplicationId (FK → applications),** isLeadApplication, supervisorFeedback, dateApplied
partnership_requests	Student partnership requests	**id (PK), requesterId (FK → students), targetStudentId (FK → students),** status, createdAt, respondedAt
admins	Admin user data	**id (PK),** email, firstName, lastName, adminRole, permissions[], isActive
email_verification_tokens	Email verification tokens	**token (PK), userId (FK → users),** email, createdAt, expiresAt, used
capacity_changes	Audit log for capacity changes	**supervisorId (FK → supervisors), adminId (FK → admins),** oldMaxCapacity, newMaxCapacity, reason, timestamp

Note: All Firestore operations go through the backend API with Firebase Admin SDK. Client-side access is blocked by security rules.

10.5.2 Security Rules

Firestore security rules are defined in firestore.rules:

- **All client access blocked**: `allow read, write: if false`
- **Admin SDK only**: All database operations go through backend API routes
- **Server-side validation**: Authorization and validation happen in middleware
- **No direct client access:** Firebase client SDK only used for authentication

10.6 Development Workflows

10.6.1 Git Workflow

Follow these conventions:

- **Branches:** Create feature branches from `main`
- **Commits:** Descriptive commit messages
- **Pull Requests:** Required for all changes to `main`
- **Code Review:** At least one approval required

10.6.2 Adding New Features

Example: Adding a new API endpoint

1. **Define Zod schema** (`lib/middleware/validation.ts`):

```
export const MyFeatureSchema = z.object({ field: z.string() });
```

2. **Create service function** (`lib/services/my-feature/`):

```
export const myFeatureService = { async create(data) { /* business logic */ } };
```

3. **Create API route** (`app/api/my-feature/route.ts`):

```
import { apiHandler } from '@/lib/middleware/apiHandler';
import { myFeatureService } from '@/lib/services/my-feature';
export const POST = apiHandler(async (req, { user }) => {
  const result = await myFeatureService.create(await req.json());
  return ApiResponse.success(result);
}, { requireAuth: true, requiredRole: 'student' });
```

4. **Create frontend hook** (`lib/hooks/useMyFeature.ts`):

```
export function useMyFeature() { /* hook logic */ }
```

5. **Create UI component** (`app/components/MyFeature.tsx`):

```
'use client';
export function MyFeature() {
  const { data } = useMyFeature();
  return (/* component JSX */);
}
```

10.7 Debugging and Troubleshooting

10.7.1 HTTP Status Codes Reference

Error Code	Meaning	Common Cause	Solution
400	Bad Request	Missing/incorrect request parameters	Validate request payload against Zod schema, check required fields
401	Unauthorized	Not authenticated or invalid token	Verify user is logged in, check Firebase Auth token validity
403	Forbidden	Insufficient permissions	Verify user role matches endpoint requirements
404	Not Found	Resource doesn't exist	Check endpoint URL, verify document ID exists in Firestore
500	Internal Server Error	Server-side failure	Check Vercel logs, verify database connection, review error context

10.7.2 Common Issues and Solutions

Problem	Cause	Solution
"Firebase not initialized" error	Missing environment variables	Verify .env.local exists and contains all required Firebase config
"Unauthorized" on API calls	Missing or invalid auth token	Ensure user is logged in, check Firebase Auth state in browser console
"Role not found" error	User doesn't have role assigned	Check users collection in Firestore has role field set correctly
Build fails with type errors	TypeScript compilation errors	Run npm run type-check to see detailed type errors
Firestore permission denied	Security rules rejecting request	Verify user authentication and check Firestore security rules in Firebase Console
"Partner email must be valid"	Email doesn't match domain	Ensure email ends with @e.braude.ac.il

10.8 MentorMatch-Specific Features

10.8.1 Security Policies

Email Restrictions: Only `@e.braude.ac.il` emails allowed.
Password Requirements: Min 8 chars, 1 uppercase, 1 lowercase, 1 number.
Access Control: Email verification required,Role-Based Access Control, Firebase Auth.
Maintenance: Quarterly user reviews, remove graduated students annually, audit admin access.

10.9 Future Maintenance

Monitoring

- Centralized logging via lib/logger.ts (view in Vercel/Firebase Console)
- Track API response times and daily active users (DAU)

Optimization

- Fix N+1 query problems
- Add composite Firestore indexes
- Implement pagination for large lists
- Archive data older than 2 years
- Use batch operations for bulk writes
- Add client-side caching

Security

- Monthly: Run npm audit
- Quarterly: Review Firebase Auth rules
- Annually: Audit admin accounts, rotate service keys

11. References

1. Malka, N. & Krispin, N. (2024). ProjectHub: Final Projects Portal. Braude College Capstone Project.
2. Memorial University of Newfoundland. Research Project Management Guide for Researchers.
3. Nwanakezie, H., & Ogona, I. (2021). Task Development Procedures for Effective Educational Management.
4. Sanders, E.B.-N. & Stappers, P.J. (2008). Co-creation and the new landscapes of design. CoDesign, 4(1), 5-18.
5. Firebase Documentation. https://firebase.google.com/docs
6. Next.js Documentation. https://nextjs.org/docs
7. Resend API Documentation. https://resend.com/docs