TracFac

Student-Faculty Tracking and Communication System

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Mansoor Muzahid Shaik	**AP22110010019**
Nagur Meeravali Shaik	**AP22110010061**
Geetesh K	**AP22110010062**
Ram Gopal M	**AP22110010063**

Under the Guidance of

Mr. Singarapu Pavan Kumar,

Assistant professor, Dept.of CSE

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

April,2025

CERTIFICATE

Date: 17-04-2025

This is to certify that the project titled "TracFac - Student-Faculty Tracking and Communication System" submitted by Geetesh K (AP22110010062), Nagur Meeravali Shaik (AP22110010061), Mansoor Muzahid Shaik (AP22110010019), and Ram Gopal M (AP22110010063) in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering, is a record of bonafide work carried out by them under my supervision and guidance.

The project has been completed to my satisfaction and is worthy of consideration for the award of the degree.

Mr. Singarapu Pavan Kumar
Assistant Professor
Department of Computer Science and Engineering
SRM University-AP

ACKNOWLEDGEMENT

We express our sincere gratitude to our project guide, Mr. Singarapu Pavan Kumar, Assistant Professor, Department of CSE, SRM University-AP, for his invaluable guidance, constant encouragement, and support throughout the project duration.

We are thankful to the Department of Computer Science and Engineering, SRM University-AP, for providing us with the necessary infrastructure and resources to complete this project.

We would also like to acknowledge the support and cooperation of our team members, whose collective efforts made this project possible. Special thanks to our families and friends for their constant support and encouragement.

ABSTRACT

TracFac is a comprehensive web-based platform designed to facilitate direct communication between faculty and students within educational institutions. The system provides a centralized platform for academic interactions, program management, and student-faculty relationships.

The platform implements role-based access control for administrators, faculty, and students, with specialized interfaces for each user type. Key implemented functionalities include:

- Secure user authentication with JWT and role-based access control
- Profile management with file uploads (profile pictures, timetables, course tables)
- Program and section management with real-time updates
- Student enrollment with automatic credential generation
- Direct messaging between users with read status tracking
- File management with type validation and secure storage

Built using React.js for the frontend and Node.js with Express.js for the backend, TracFac uses MongoDB as its database. The system supports various academic initiatives including regular courses, mentor-mentee programs, and research projects.

Table Of Contents

1. Introduction

1.1 Purpose

TracFac is designed to address the communication gap between faculty and students in educational institutions. The system provides a centralized platform for academic interactions and program management, eliminating the need for external communication channels.

1.2 Scope

The implemented system includes:

- User authentication and role-based access control
- Profile management for students and faculty
- Program and section management
- Student enrollment
- Direct messaging between users
- File management (profile pictures, timetables, course tables)

1.3 Definitions and Abbreviations

- **TracFac:** Student-Faculty Tracking and Communication System
- **JWT:** JSON Web Token
- **REST:** Representational State Transfer
- **API:** Application Programming Interface
- **MERN:** MongoDB, Express.js, React.js, Node.js
- **MUI:** Material-UI

1.4 Overview

This document provides a comprehensive overview of the TracFac system, including its architecture, features, and implementation details. The system is built using modern web technologies and follows best practices for security and scalability.

2. Overall Description

2.1 Project Structure

Root Directory Structure

```
project-root/
├── client/                 # Frontend React application
│   ├── node_modules/       # Frontend dependencies
│   ├── public/             # Static files
│   ├── src/                # Source code
│   │   ├── assets/         # Static assets
│   │   ├── components/     # React components
│   │   ├── context/        # Context providers
│   │   ├── contexts/       # Additional context configurations
│   │   ├── pages/          # Page components
│   │   └── utils/          # Utility functions
│   ├── .gitignore          # Git ignore configuration
│   ├── eslint.config.js    # ESLint configuration
│   ├── index.html          # Main HTML file
│   ├── package.json        # Frontend dependencies
│   ├── package-lock.json   # Package lock file
│   ├── README.md           # Frontend documentation
│   └── vite.config.js      # Vite configuration
│
├── middleware/             # Authentication middleware
│   └── auth.js             # Authentication middleware
│
├── models/                 # Database models
│   ├── Chat.js             # Chat model schema
│   ├── Program.js          # Program model schema
│   ├── Section.js          # Section model schema
│   └── User.js             # User model schema
│
├── routes/                 # API routes
│   ├── admin.js            # Admin routes
│   ├── auth.js             # Authentication routes
│   ├── chat.js             # Chat functionality
│   ├── faculty.js          # Faculty management
│   ├── program.js          # Program management
│   ├── programAssignments.js # Program assignments
│   ├── sections.js         # Section management
│   ├── student.js          # Student routes
│   └── users.js            # User management
```

```
├── server/              # Server configuration
│   ├── middleware/      # Server middleware
│   ├── models/          # Server models
│   ├── routes/          # Server routes
│   │   ├── concerns.js # Concerns handling
│   │   └── program.js # Program operations
│   ├── utils/           # Utility functions
│   ├── app.js           # Express app configuration
│   └── package-lock.json # Dependencies lock
│
├── uploads/             # File storage
│   ├── course-tables/   # Course table documents
│   ├── profile-pics/    # User profile pictures
│   └── timetables/      # Timetable documents
│
├── .env                 # Environment variables
├── createAdmin.js       # Admin creation script
├── package.json         # Project dependencies
├── package-lock.json    # Dependencies lock
├── README.md            # Project documentation
├── server.js            # Main server entry
└── SRS.md               # Software Requirements Specification
```

Frontend Structure (client/src/components/)

```
components/
├── faculty/             # Faculty-specific components
│   ├── AdminDashboard.jsx
│   ├── AdminHeader.jsx
│   ├── AssignedPrograms.jsx
│   ├── Chat.jsx
│   ├── ErrorBoundary.jsx
│   ├── FacultyDashboard.jsx
│   ├── FacultyHeader.jsx
│   ├── FacultySectionDetails.jsx
│   ├── Login.jsx
│   ├── PrivateRoute.jsx
│   ├── Profile.jsx
│   ├── ProgramDetails.jsx
│   ├── ProgramManagement.jsx
│   ├── SectionDetails.jsx
│   ├── SectionManagement.jsx
│   ├── StudentDashboard.jsx
│   ├── StudentHeader.jsx
│   ├── StudentProfileView.jsx
│   └── StudentProgramDetails.jsx
```

Backend API Structure

- **Authentication:**
  - middleware/auth.js - JWT authentication middleware
  - routes/auth.js - Login and registration endpoints
- **User Management:**
  - models/User.js - User data schema
  - routes/users.js - User CRUD operations
  - routes/admin.js - Admin-specific operations
  - routes/faculty.js - Faculty management
  - routes/student.js - Student operations
- **Program Management:**
  - models/Program.js - Program schema
  - models/Section.js - Section schema
  - routes/program.js - Program operations
  - routes/programAssignments.js - Assignment handling
  - routes/sections.js - Section management
- **Communication:**
  - models/Chat.js - Chat schema
  - routes/chat.js - Messaging endpoints
  - server/routes/concerns.js - Concerns handling
- **File Management:**
  - uploads/profile-pics/ - Profile images
  - uploads/course-tables/ - Course documents
  - uploads/timetables/ - Schedule files

2.2 Technology Stack

Frontend

- React.js - UI framework

Used for building the entire user interface with component-based architecture.

- Vite - Build tool

Used for fast development and optimized production builds. We configured Vite to handle our React application with hot module replacement for quick development cycles.

- Material-UI - Component library

Used for consistent UI design across the application. For example, we used Material-UI's TextField, Button, and Card components throughout the application.

- Axios - HTTP client

Used for making API calls to our backend. For example, we created an API service that handles all HTTP requests with proper error handling and authentication headers.

- React Router - Navigation

Used for handling client-side routing. We implemented role-based routing where different users (admin, faculty, student) see different pages based on their role.

- Context API - State management

Used for managing global state like user authentication and program data. We created contexts for auth state and user data that are accessible throughout the application.

Backend

- Node.js - Runtime environment

Used as the server runtime. We created a RESTful API server that handles all backend operations including authentication, file uploads, and data management.

- Express.js - Web framework

Used for building the API endpoints. We created routes for user management, program management, and chat functionality with proper middleware for authentication and validation.

- MongoDB - Database

Used as the primary database. We store all application data including user profiles, programs, and chat messages in MongoDB collections.

- MongoDB Compass - Database management

Used for database visualization and management. We used MongoDB Compass to:

    - View and edit documents in our collections
    - Create and manage indexes for better query performance
    - Monitor database performance and query execution

    - Import and export data during development
    - Validate data structure and relationships
- JWT - Authentication

Used for user authentication. We implemented JWT-based authentication where users receive a token upon login that is used for subsequent API requests.

- Multer - File uploads

Used for handling file uploads. We configured Multer to handle different types of file uploads including profile pictures, timetables, and course tables with proper validation and storage.

Development Tools

- Postman - API testing

Used for testing API endpoints during development and validation phases. Created comprehensive test suites for each endpoint.

- MongoDB Compass - Database management

Used for database operations. We used MongoDB Compass to:

    - Create and manage database collections
    - Write and test MongoDB queries
    - Monitor database performance
    - Backup and restore data

3. Database Implementation

User Collection Implementation

We used MongoDB Compass to create and manage our User collection. Here's how we implemented it:

```
{
  "_id": ObjectId,
  "name": String,
  "email": String,
  "password": String,
  "role": String,
  "profile": {
    "facultyId": String,
    "studentId": String,
    "college": String,
    "branch": String,
    "department": String,
    "currentYear": Number,
    "currentSemester": Number,
    "profilePic": String,
    "timetablePic": String,
    "coursesTablePic": String,
    "gpaPdf": String,
    "currentGpa": Number,
```

```
    "socialLinks": {
      "linkedin": String,
      "github": String
    },
    "description": String,
    "courses": [String]
  }
}
```

Example usage in MongoDB Compass:

- Created indexes on email and role fields for faster queries
- Used the Schema Validation feature to ensure data integrity
- Created views for different user roles (admin, faculty, student)
- Used the Aggregation Pipeline Builder for complex queries

Program Collection Implementation

We used MongoDB Compass to manage our Program collection. Here's how we implemented it:

```
{
  "_id": ObjectId,
  "title": String,
  "description": String,
  "type": String,
  "sections": [{
    "name": String,
    "faculty": ObjectId,
    "students": [ObjectId],
    "capacity": Number
  }],
  "startDate": Date,
  "endDate": Date,
  "status": String
}
```

Example usage in MongoDB Compass:

- Created compound indexes for program search queries
- Used the Schema Validation feature to ensure proper data structure
- Created views for active, completed, and cancelled programs
- Used the Performance tab to optimize query performance

Chat Collection Implementation

We used MongoDB Compass to manage our Chat collection. Here's how we implemented it:

```
{
  "_id": ObjectId,
  "participants": [ObjectId],
  "messages": [{
    "sender": ObjectId,
    "content": String,
    "attachments": [String],
    "readBy": [ObjectId],
    "timestamp": Date
  }],
  "lastMessage": Date
}
```

Example usage in MongoDB Compass:

- Created TTL indexes for message cleanup
- Used the Schema Validation feature to ensure message structure
- Created views for unread messages and active chats
- Used the Performance tab to monitor chat query performance

4. System Features

4.1 User Authentication

- JWT-based authentication system
- Role-based access control (admin, faculty, student)
- Secure password hashing with bcrypt
- Session management with token expiration
- Admin-controlled password recovery

4.2 Profile Management

- Student academic information (college, branch, department, year, semester, GPA)
- Faculty professional information (department, specialization)
- Profile picture uploads with Multer
- Timetable and course table uploads
- Social links (LinkedIn, GitHub)

4.3 Program Management

- Program creation and management by administrators
- Section management within programs
- Student enrollment tracking
- Faculty assignments to programs
- Real-time updates for program changes

4.4 Student Enrollment

- Student registration by administrators
- Automatic login credential generation
- Program and section enrollment
- Enrollment notifications
- Enrollment status tracking

4.5 Messaging

- Direct messaging between users
- Group conversations for programs
- Message notifications
- File attachments
- Message read status tracking

4.6 File Management

- Profile picture uploads with validation
- Timetable and course table uploads
- File type validation (images, PDFs)
- Secure storage with access control
- Automatic file cleanup for updates

5. Non-Functional Requirements

5.1 Performance Requirements

- Page load time under 3 seconds
- Support for 1000+ concurrent users
- System responsiveness during peak usage
- Efficient file upload and download
- Quick response time for database queries

5.2 Security Requirements

- Secure user authentication
- Role-based access control
- Data encryption for sensitive information
- Input validation and sanitization
- File type validation and security
- Protection against common web vulnerabilities

5.3 Reliability Requirements

- 99.9% system uptime
- Data backup and recovery procedures
- Error handling and logging
- System monitoring and maintenance
- Graceful degradation during high load

6. Other Requirements

6.1 Database Requirements

- MongoDB as primary database
- Data consistency and integrity
- Efficient indexing and querying
- Regular backup procedures
- Data migration capabilities

6.2 Documentation Requirements

- User documentation
- API documentation
- System architecture documentation
- Deployment documentation
- Maintenance documentation

6.3 Training Requirements

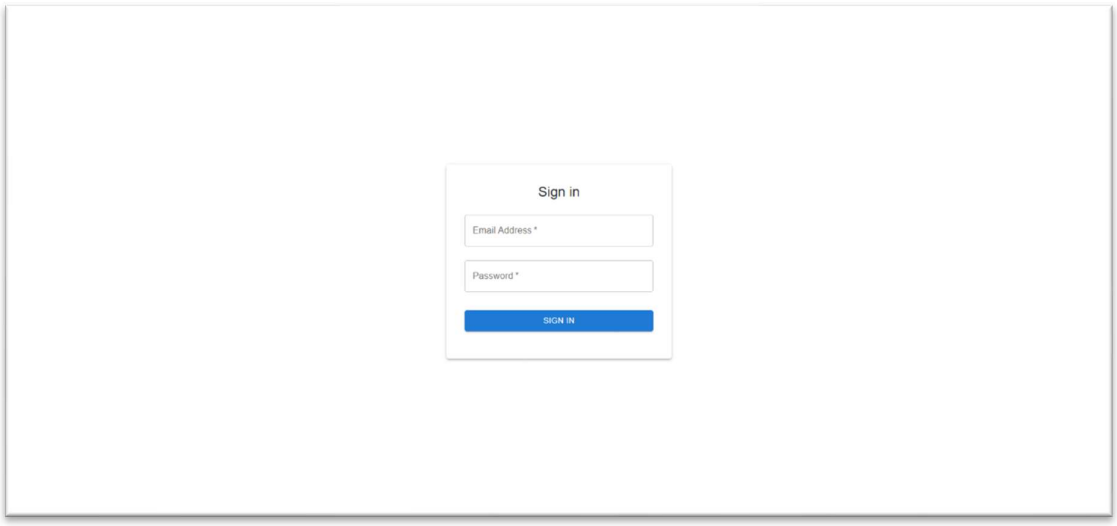
- User training materials
- Administrator training
- Technical support documentation
- FAQ and troubleshooting guides

7. Screenshots

7.1 User Interface

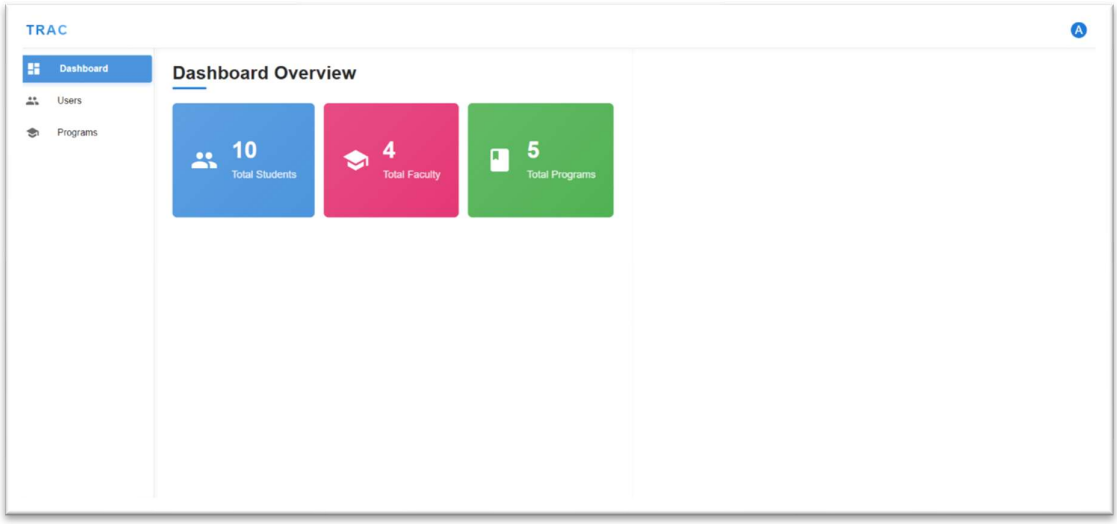
Sign In Screen

User authentication interface for students, faculty, and administrators



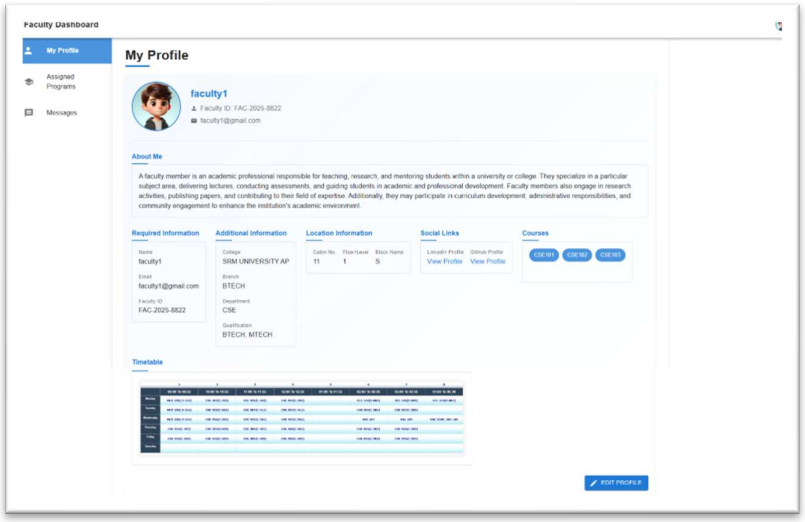
Admin Dashboard

Comprehensive management interface for system administrators



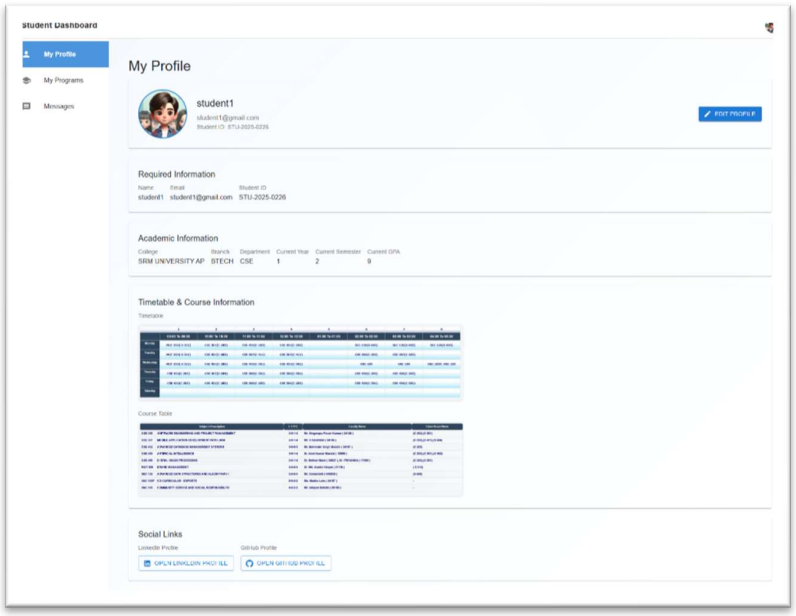
Faculty Dashboard

Interface for faculty to manage programs and communicate with students



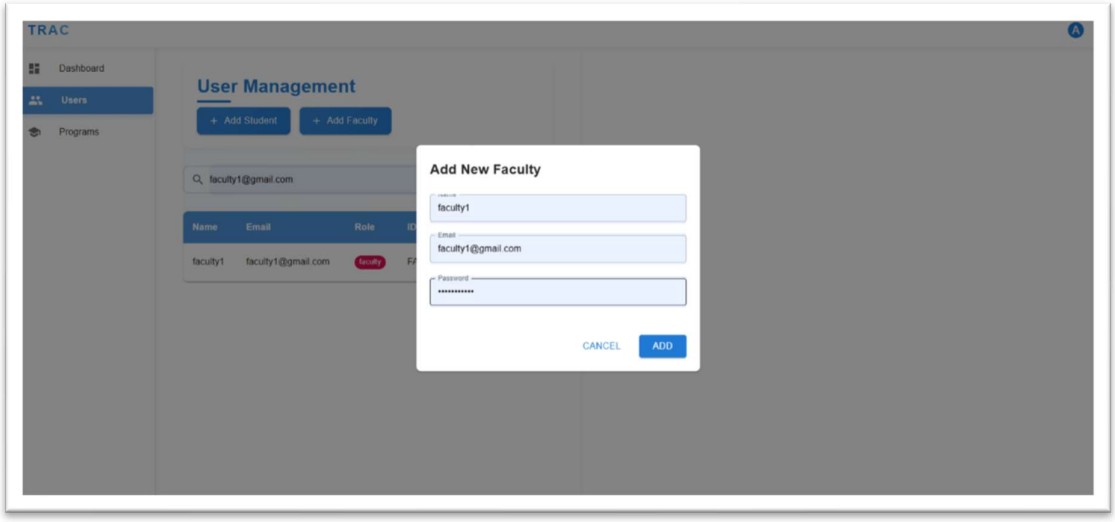
Student Dashboard

Main interface for students to view programs and communicate with faculty



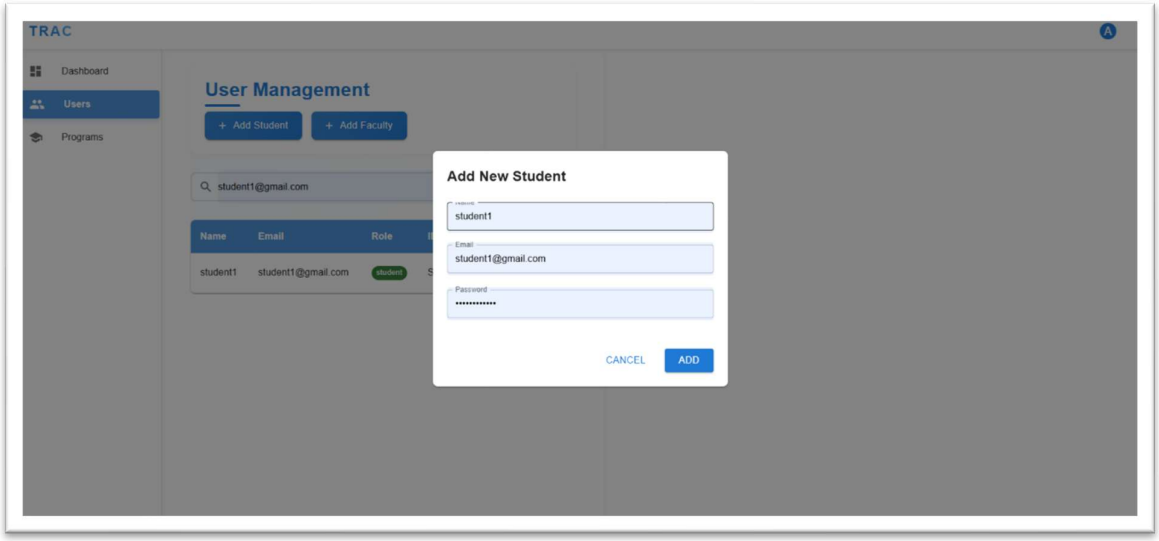
Add Faculty

Interface for administrators to add new faculty members to the system



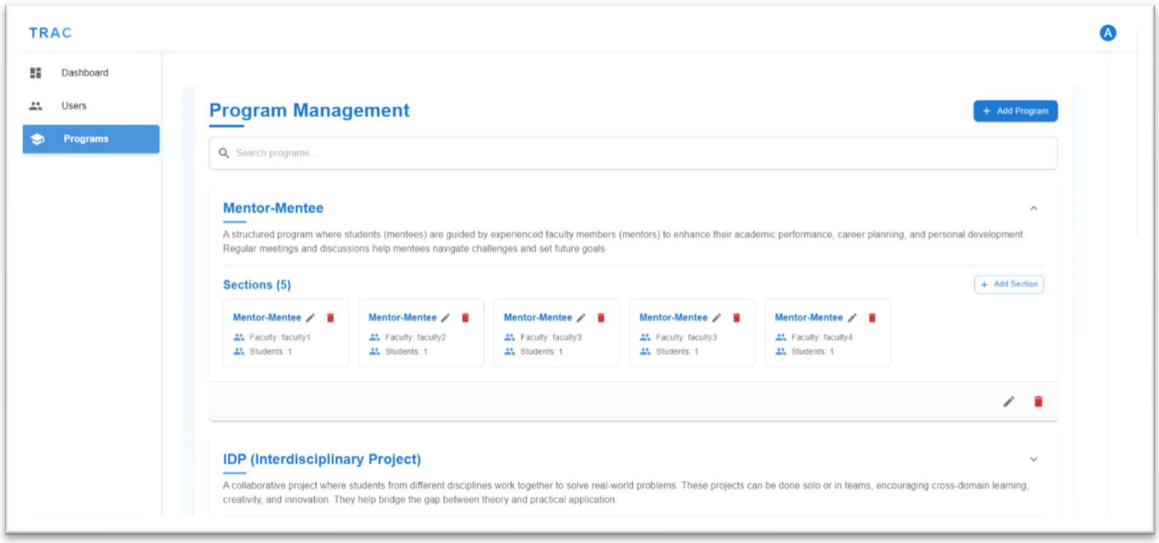
Add Student

Interface for administrators to add new students to the system



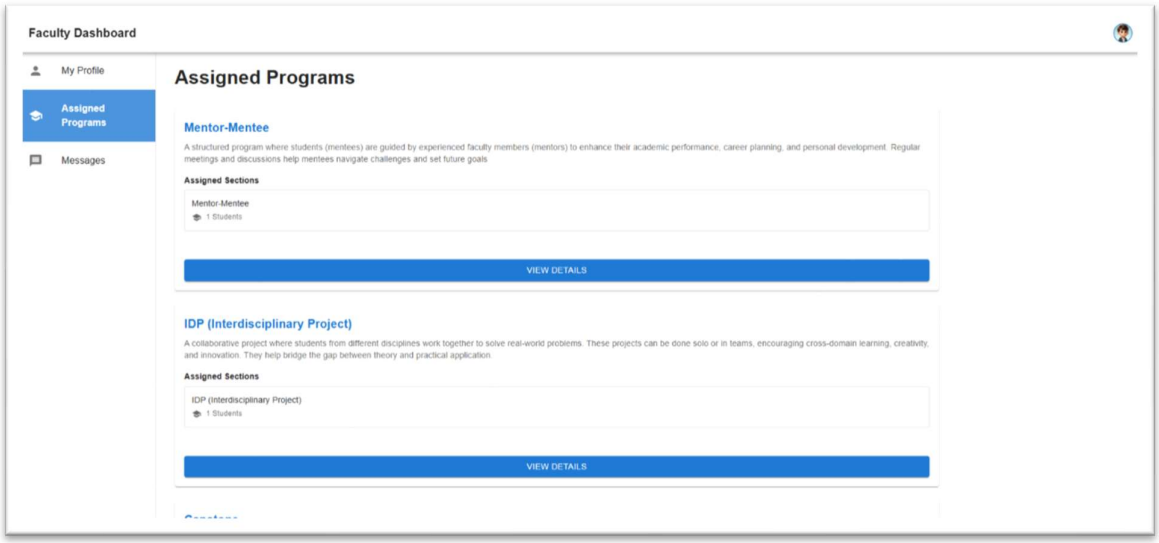
Program Management

Interface for administrators to create and manage academic programs



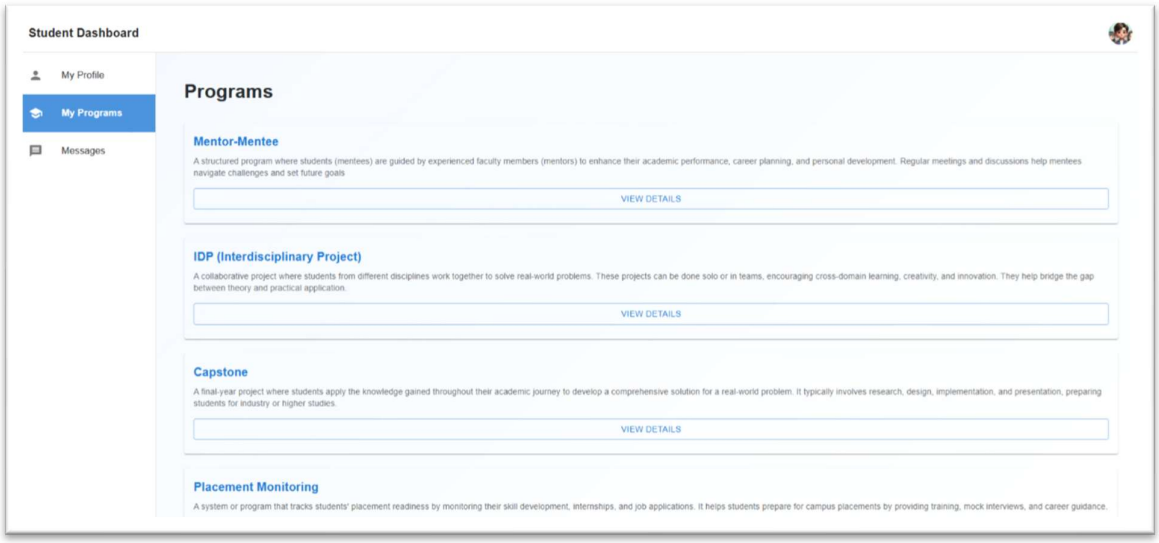
Faculty Programs

Interface for faculty to manage their assigned programs



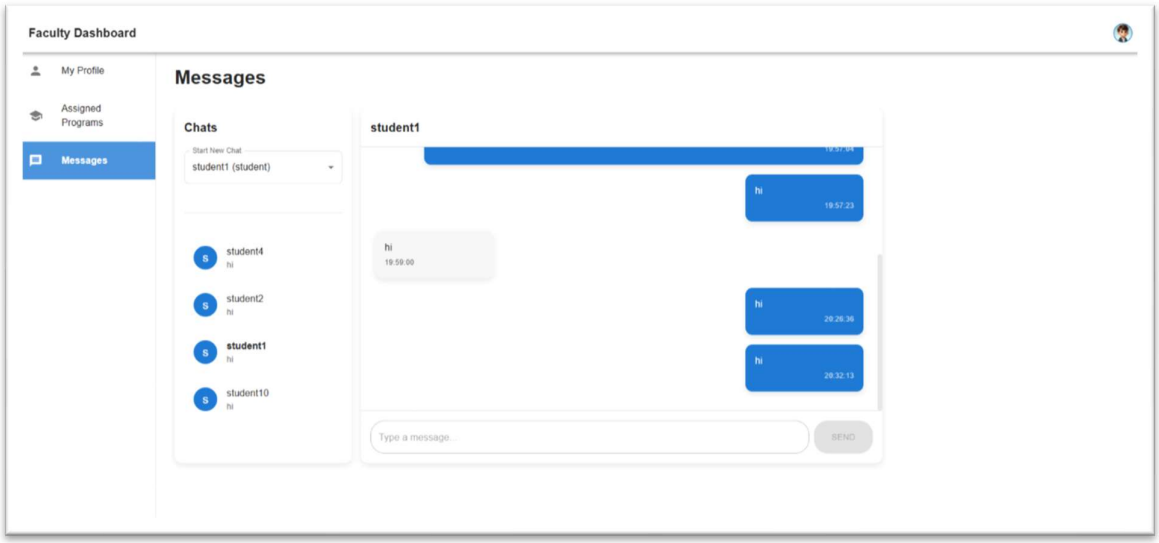
Student Programs

View of programs available to students



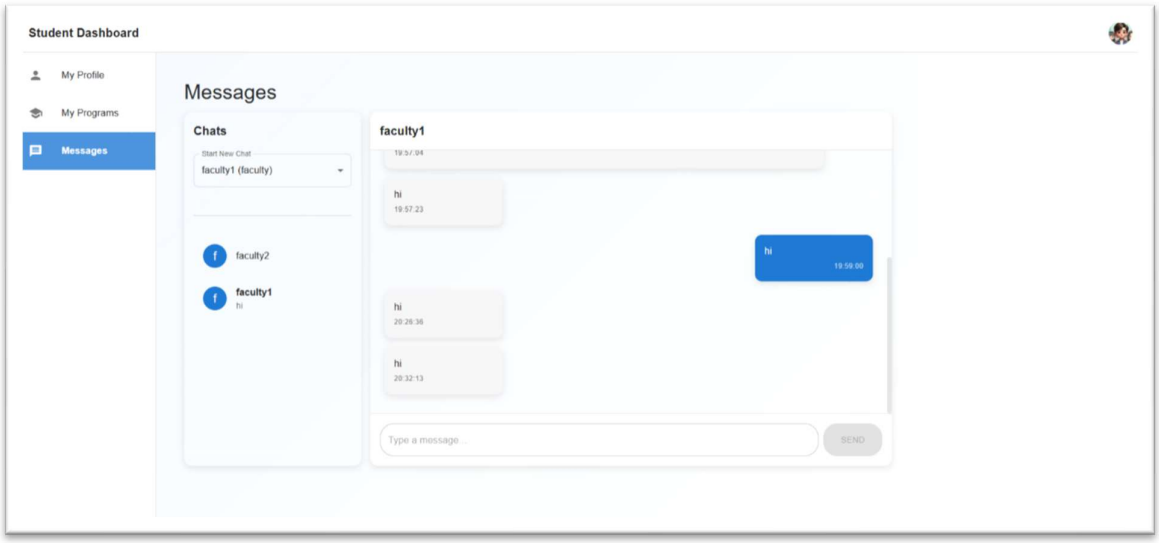
Faculty Messages

Messaging interface for faculty to communicate with students



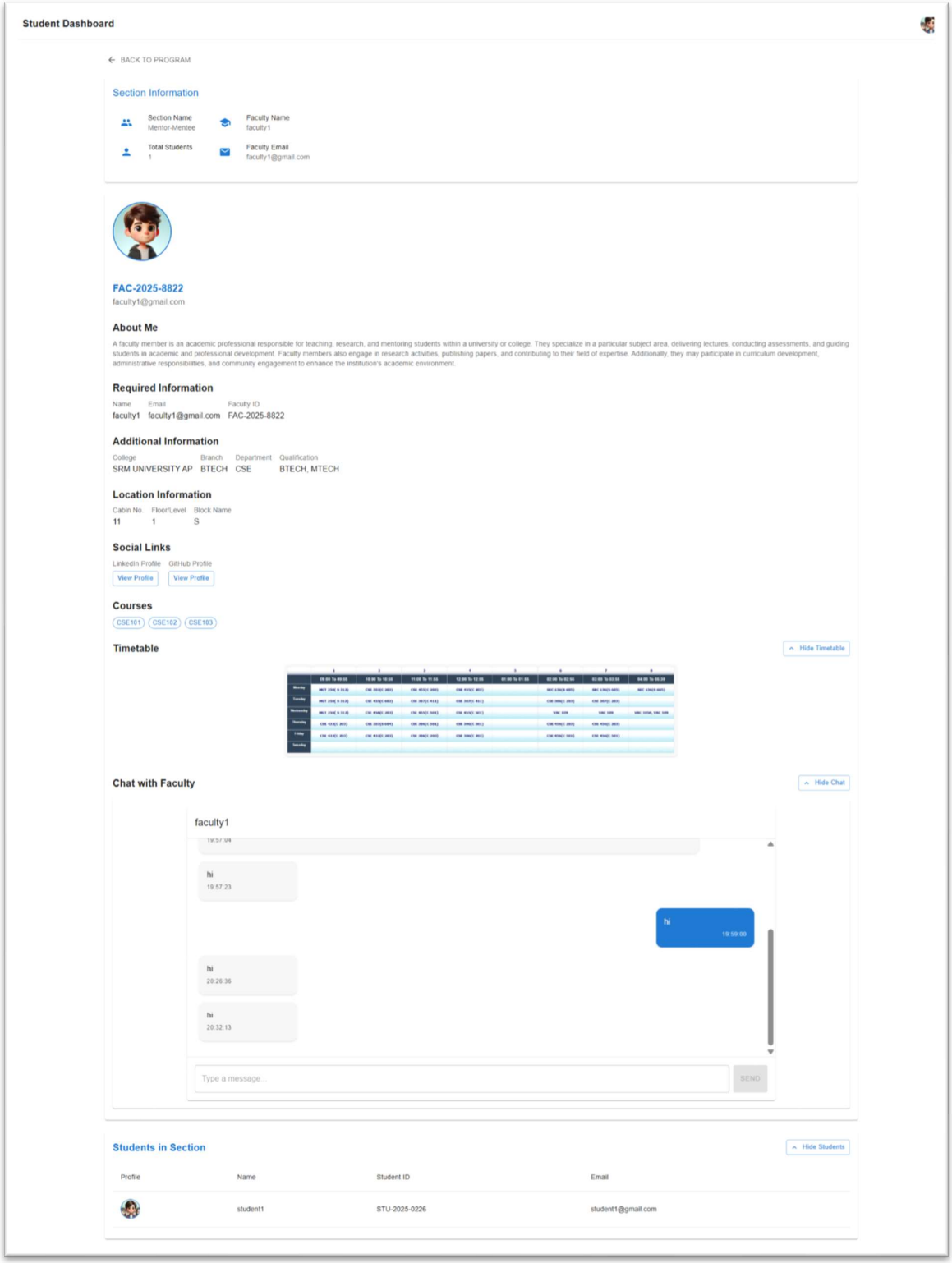
Student Messages

Messaging interface for students to communicate with faculty



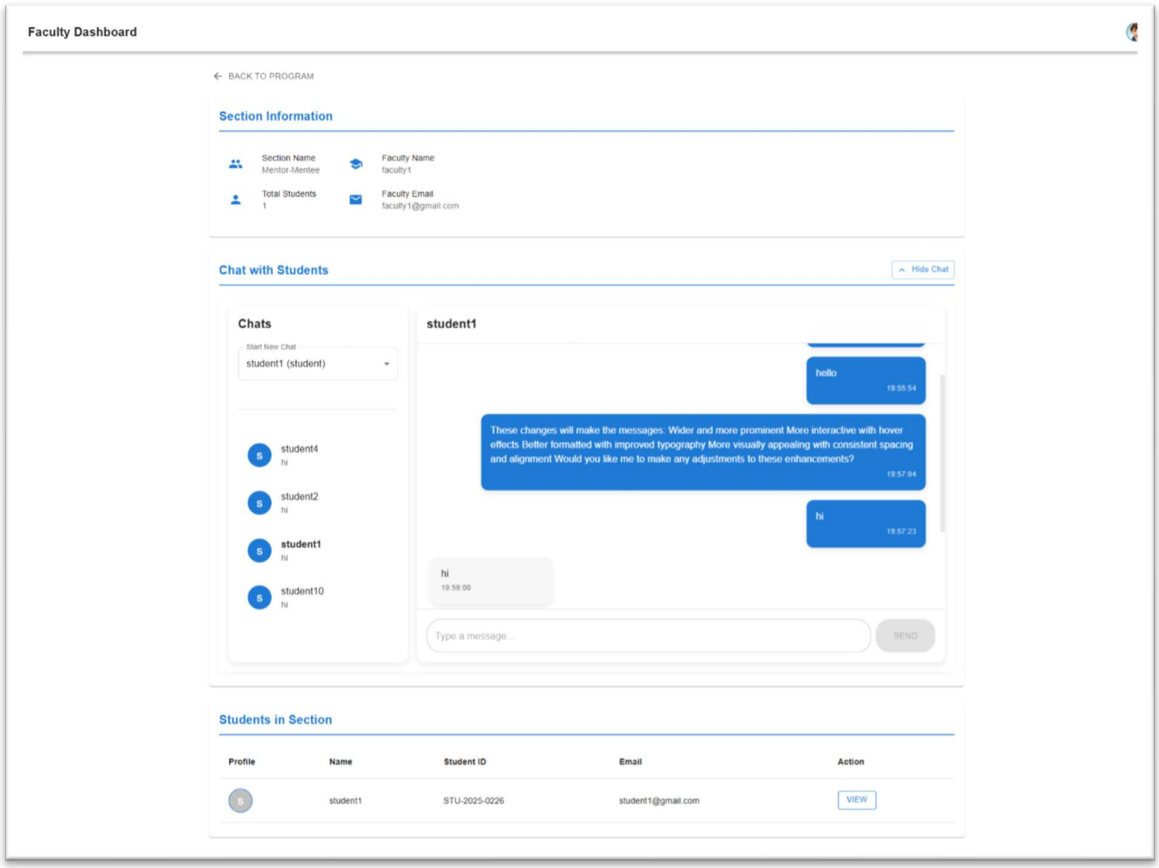
Student Section Overview

Comprehensive view of sections available to students



Faculty Section Overview

Comprehensive view of sections managed by faculty



8. Testing

Manual Functionality Testing

- **Authentication Testing:**
  - User registration validation
  - Login with different user roles
  - Password reset functionality
  - Session management
  - Authorization checks for protected routes
- **User Management Testing:**
  - Profile creation and updates
  - File upload functionality
  - Role-based access control
  - User search and filtering
- **Program Management Testing:**
  - Program creation and modification
  - Section assignment
  - Student enrollment
  - Faculty assignment
- **Communication Testing:**
  - Message sending and receiving between users
  - Message read status tracking
  - Direct messaging between faculty and students

9. Front-End Description

9.1 Technology Stack

React.js

Material-UI

Axios

React Router

Context API

React Hooks

9.2 Component Structure

Example Component Hierarchy:

```
components/
├── faculty/          # Faculty-specific components
│   ├── AdminDashboard.jsx
│   ├── AdminHeader.jsx
│   ├── AssignedPrograms.jsx
│   ├── Chat.jsx
│   ├── ErrorBoundary.jsx
│   ├── FacultyDashboard.jsx
│   ├── FacultyHeader.jsx
│   ├── FacultySectionDetails.jsx
│   ├── Login.jsx
│   ├── PrivateRoute.jsx
│   ├── Profile.jsx
│   ├── ProgramDetails.jsx
│   ├── ProgramManagement.jsx
│   ├── SectionDetails.jsx
│   ├── SectionManagement.jsx
│   ├── StudentDashboard.jsx
│   ├── StudentHeader.jsx
│   ├── StudentProfileView.jsx
│   └── StudentProgramDetails.jsx
```

10. Back-End Description

10.1 Technology Stack

Node.js

Express.js

MongoDB

MongoDB Compass

JWT

Multer

10.2 API Implementation

Authentication API

- POST /api/auth/login - User authentication with JWT
- POST /api/auth/register - User registration (admin only)
- PUT /api/auth/profile - Profile management with file uploads

User Management API

- GET /api/users - List all users
- GET /api/users/:id - Get specific user details
- PUT /api/users/:id - Update user information

Program Management API

- GET /api/programs - List all programs
- POST /api/programs - Create new program
- GET /api/programs/:id - Get program details
- PUT /api/programs/:id - Update program

- DELETE /api/programs/:id - Delete program

Communication API

- GET /api/chat - Get user chats
- POST /api/chat - Create new chat
- POST /api/chat/:id/message - Send message
- GET /api/concerns - Get concerns
- POST /api/concerns - Submit new concern

10.3 API Security Implementation

- JWT-based authentication
- Role-based access control
- Input validation and sanitization
- File upload security with Multer
- Rate limiting for API endpoints

10.4 API Error Handling

- Standardized error responses
- HTTP status code usage
- Error logging and monitoring
- Client-friendly error messages

11. Conclusion

TracFac provides a comprehensive platform for student-faculty communication and program management. The implementation demonstrates the effectiveness of modern web technologies in creating a secure and user-friendly platform.

Key achievements include:

- Role-based access control
- Program and section management
- Direct messaging capabilities
- Student enrollment system
- File management system
- Real-time updates

The system meets the core requirements and provides a foundation for future enhancements.

12. Future Work

Potential areas of future improvement include:

- Integration of real-time notifications via WebSocket or Firebase.
- Implementation of mobile applications (Android/iOS).
- Automated program recommendation engine using machine learning.
- Advanced reporting and analytics for administrators.
- Enhanced file management with version control features.

13. References

- MongoDB Documentation
- MongoDB Compass Documentation
- Express.js Documentation
- React.js Documentation
- Node.js Documentation
- JWT Documentation
- Material-UI Documentation
- Axios Documentation
- React Router Documentation
- Multer Documentation
- bcrypt Documentation