

# Alumni Portal

Complete Project Architecture & Documentation

CAHCET Alumni Network

Generated: January 29, 2026

---

## Table of Contents

1. Project Overview
2. Client (Frontend) Structure
3. Server (Backend) Structure
4. Authentication & Security
5. Data Models & Relationships
6. API Endpoints
7. Request Flow Examples
8. Importance Summary

# 1. Project Overview

## What is Alumni Portal?

The Alumni Portal is a full-stack web application built with React (frontend) and Node.js/Express (backend) that connects college alumni with current students. It facilitates networking, job opportunities, event management, and profile sharing among CAHCET (C Abdul Hakeem College of Engineering & Technology) community members.

## Technology Stack

Layer	Technology	Purpose
<b>Frontend</b>	React 19, React Router, Axios	User interface & navigation
<b>Backend</b>	Node.js, Express 5	API server & business logic
<b>Database</b>	MongoDB, Mongoose	Data storage & validation
<b>Authentication</b>	JWT, bcryptjs	Secure login & permissions
<b>File Upload</b>	Multer	Handle file uploads

## 2. Client (Frontend) Structure

The client folder contains all React components, pages, and utilities for the user interface. It runs on `http://localhost:3000` during development.

### `/client/package.json`

**Role:** Project configuration and dependency manifest

**Why Used:** Defines all npm packages required to run the React application

**Importance:**  CRITICAL

#### Key Dependencies:

- **react (19.2.4)** - UI framework
- **react-router-dom (7.13.0)** - Page navigation
- **axios (1.13.3)** - HTTP requests to backend
- **jwt-decode (4.0.0)** - Decode authentication tokens

### `/client/public/` - Static Assets

File	Purpose	Importance
<b>index.html</b>	Main HTML file where React app loads	
manifest.json	Progressive Web App metadata	
robots.txt	Search engine crawling rules	

### `/client/src/index.js`

**Role:** React application entry point

**Why Used:** Mounts the React app to the DOM (to index.html)

**Importance:** ★★★★★ CRITICAL

```
import React from "react"; import ReactDOM from "react-dom/client"; import
App from "./App"; const root =
ReactDOM.createRoot(document.getElementById("root")); root.render(<App />);
```

## `/client/src/App.js`

**Role:** Main routing and navigation component

**Why Used:** Controls all page routes and access protection

**Importance:** ★★★★★ CRITICAL

### Key Features:

- ProtectedRoute - Prevents unauthorized access
- Admin routes (approvals, bulk upload, carousel, events)
- Alumni routes (profile, directory)
- Public routes (login, register)

## `/client/src/pages/` - Page Components

File	Purpose	Accessible By
<b>Login.js</b>	User authentication page	Everyone (public)
<b>Register.js</b>	User registration & signup	Everyone (public)
<b>Dashboard.js</b>	Main hub/navigation center	All logged-in users
<b>AdminApprovals.js</b>	Manage pending user approvals	Admins only
<b>AdminBulkUpload.js</b>	<b>Bulk upload users via CSV</b>	<b>Admins only (NEW)</b>
<b>AdminCarousel.js</b>	Manage carousel/banner images	Admins only
<b>AdminCreateEvent.js</b>	Create new events	Admins only

File	Purpose	Accessible By
<b>AlumniDirectory.js</b>	Browse and search alumni	All users
<b>AlumniProfile.js</b>	View and edit personal profile	Alumni users
<b>Events.js</b>	View all events	All users
<b>Jobs.js</b>	Browse job postings	All users
<b>PostJob.js</b>	Alumni post job openings	Alumni users

## 🎨 `/client/src/components/` - Reusable UI Components

Component	Purpose	Used In
<b>Navbar.js</b>	Navigation header with logo and logout	Every page
<b>ImageCarousel.js</b>	Rotating banner with images	Dashboard

## 🔧 `/client/src/services/api.js`

**Role:** Centralized HTTP client for backend communication

**Why Used:** All API requests go through this file

**Importance:** ★★★★★ CRITICAL

**Key Feature - Auto-Token Injection:** Automatically adds JWT token to every API request header

```
api.interceptors.request.use((config) => { const token =
localStorage.getItem("token"); if (token) { config.headers.Authorization =
`Bearer ${token}`; } return config; });
```

## 🎨 `/client/src/styles/theme.js`

**Role:** Centralized design system and color palette

**Why Used:** Keeps UI consistent across all pages

**Importance:**  Important

**Contains:** Colors, fonts, spacing, button styles, card styles

## [`/client/src/utils/constants.js`](#)

**Role:** App-wide constant values

**Why Used:** Store fixed values (dropdown options, labels, etc.)

**Importance:**  Nice-to-have

## [`/client/src/assets/`](#)

**Role:** Images, logos, and icons

**Contents:** logo.png, favicon, banner images

**Importance:**  Visual elements

## 3. Server (Backend) Structure

The server folder contains the Express.js API that handles all business logic, database operations, and authentication. It runs on `http://localhost:5000` during development.

### `/server/package.json`

**Role:** Backend dependencies and scripts configuration

**Importance:**  CRITICAL

**Key Dependencies:**

- **express (5.2.1)** - Web server framework
- **mongoose (9.1.5)** - MongoDB object modeling
- **bcryptjs (3.0.3)** - Password hashing
- **jsonwebtoken (9.0.3)** - JWT token creation
- **cors (2.8.6)** - Cross-origin resource sharing
- **dotenv (17.2.3)** - Environment variables
- **multer (2.0.2)** - File upload handling

### `/server/server.js`

**Role:** Main server entry point

**Why Used:** Sets up Express app, connects to database, registers API routes

**Importance:**  CRITICAL

**What it does:**

- Load environment variables from .env file
- Connect to MongoDB database
- Set up Express middleware (CORS, JSON parsing)
- Register all API routes

- Start listening on PORT (default 5000)

## `/server/.env`

**Role:** Secret configuration and environment variables

**Importance:**  CRITICAL - SECURITY!

**⚠️ WARNING:** This file should NEVER be committed to git. It contains sensitive data like database passwords and JWT secrets.

**Must contain:**

```
MONGO_URI=mongodb+srv://username:password@cluster.mongodb.net/database
JWT_SECRET=your_secret_key_here PORT=5000 NODE_ENV=development
```

## `/server/config/db.js`

**Role:** Database connection setup

**Why Used:** Connects to MongoDB using Mongoose

**Importance:**  CRITICAL

```
const mongoose = require("mongoose");
const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI);
    console.log(`MongoDB Connected: ${conn.connection.host}`);
  } catch (error) {
    console.error(`Error: ${error.message}`);
    process.exit(1);
  }
};

module.exports = connectDB;
```

## `/server/models/` - Database Schemas

These files define how data is structured and validated in MongoDB using Mongoose schemas.

Model	Purpose	Key Fields
<b>User.js</b>	User accounts (login credentials)	name, email, password (hashed), role, status, timestamps
<b>AlumniProfile.js</b>	Extended alumni information	userId (reference), registerNumber, department, batchYear, company, skills, bio
<b>Event.js</b>	Events and conferences	title, description, date, location, createdBy (admin)
<b>Job.js</b>	Job postings	title, company, description, salary, location, postedBy (alumni)
<b>CarousellImage.js</b>	Banner/carousel images	imageUrl, title, description, uploadedBy (admin)

## `/server/controllers/` - Business Logic

Controllers contain the business logic that handles what happens when API requests arrive.

Controller	Purpose	Key Functions
<b>authController.js</b>	Handle login and registration	register(), login(), validateToken()
<b>adminController.js</b>	Admin-specific operations	Currently empty - can be expanded
<b>alumniController.js</b>	Alumni profile operations	createOrUpdateProfile(), getAllProfiles(), getMyProfile()
<b>eventController.js</b>	Event management	createEvent(), getEvents(), deleteEvent()
<b>jobController.js</b>	Job posting management	postJob(), getJobs(), deleteJob()

Controller	Purpose	Key Functions
<b>carouselController.js</b>	Carousel image management	uploadImage(), getImages(), deleteImage()

## `/server/routes/` - API Endpoints

Routes connect HTTP requests to controller functions. They define the API endpoints and which middleware to use.

Route File	Base URL	Key Endpoints
<b>authRoutes.js</b>	/api/auth	POST /register, POST /login
<b>adminRoutes.js</b>	/api/admin	GET /pending-users, PUT /approve/:id, <b>POST /bulk-upload (NEW)</b>
<b>alumniRoutes.js</b>	/api/alumni	GET /profiles, POST /profile, GET /profile/:id
<b>jobRoutes.js</b>	/api/jobs	GET /list, POST /post, DELETE /:id
<b>eventRoutes.js</b>	/api/events	GET /list, POST /create, DELETE /:id
<b>carouselRoutes.js</b>	/api/carousel	GET /images, POST /upload, DELETE /:id

## `/server/middleware/` - Security & Permission Checks

Middleware functions check permissions and validate requests before they reach controllers.

Middleware	What it Checks	Purpose
<b>authMiddleware.js</b>	Is user logged in?	Verify JWT token is valid and hasn't expired
<b>adminMiddleware.js</b>	Is user an admin?	Only allow admins to access admin-only routes

Middleware	What it Checks	Purpose
<b>roleMiddleware.js</b>	Does user have required role?	Role-based access control (admin, alumni, student)

## How Middleware Works (Example):

```
router.put("/approve/:id", auth, admin, approveUser); // 1. auth checks if
JWT token exists and is valid // 2. admin checks if user has admin role // 3.
approveUser executes only if both checks pass
```

**Importance:** ★★★★★ CRITICAL - SECURITY

## `/server/seedAdmin.js` & `/server/seedAlumni.js`

**Role:** Populate database with test data

**Why Used:** Quick testing without manual data entry

**How to Use:** Run `node seedAdmin.js` or `node seedAlumni.js` in terminal

**Importance:** ★★ Development Helper

## 4. Authentication & Security

### JWT (JSON Web Token) Authentication

The application uses JWT for stateless authentication. When users log in, they receive a token that proves they're authenticated without storing session data on the server.

### Authentication Flow Diagram

```
User enters email & password ↓ POST /api/auth/login ↓ authController validates credentials ↓  
Password matches? → Generate JWT token ↓ Send token to frontend (stored in localStorage) ↓  
User sends token with every API request ↓ authMiddleware verifies token signature & expiry ↓  
Access granted or denied
```

### Password Security

Passwords are hashed using **bcryptjs** (bcrypt algorithm) before storing in database. This means:

- Plain passwords are NEVER stored
- Even database leaks don't expose passwords
- Passwords are salted (random data added) for extra security

### Token Storage

The JWT token is stored in browser's `localStorage`. The **api.js** interceptor automatically includes this token in every API request header:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

### Role-Based Access Control (RBAC)

Three user roles control what features users can access:

Role	Capabilities
<b>ADMIN</b>	<ul style="list-style-type: none"><li>• Approve pending users</li><li>• Bulk upload users (NEW)</li><li>• Manage carousel images</li><li>• Create events</li><li>• View all user data</li></ul>
<b>ALUMNI / STUDENT</b>	<ul style="list-style-type: none"><li>• Create and edit own profile</li><li>• Browse alumni directory</li><li>• Apply to jobs</li><li>• Post job openings</li><li>• Register for events</li></ul>

## User Status States

Users go through different status states during their lifecycle:

- **PENDING** - Just registered, waiting for admin approval
- **APPROVED** - Admin approved, can access platform
- **BLOCKED** - Admin blocked user, cannot login

# 5. Data Models & Relationships

## Entity Relationship Diagram

```
User (Account) |--- role: "admin" | "alumni" | "student" |--- status: "pending" | "approved" | "blocked" |--- name, email, password (hashed) |--- 1:1 → AlumniProfile (Optional) | |--- registerNumber, department, batchYear | |--- currentCompany, designation | |--- skills, bio, linkedinUrl | |--- 1:M → Job (Posted by alumni) | |--- title, description | |--- salary, location | |--- postedBy: userId | |--- 1:M → Event (Attending) | |--- title, date | |--- description, location | |--- createdBy: userId Admin User |--- 1:M → CarouselImage (Manages) | |--- imageUrl, title | |--- uploadedBy: userId | |--- 1:M → Event (Creates) | |--- createdBy: userId
```

## Key Relationships

From	To	Type	Optional?
User	AlumniProfile	One-to-One	Yes - can exist without profile
AlumniProfile	User	Many-to-One	No - must reference a user
Job	User	Many-to-One	No - must have poster
Event	User	Many-to-One	No - must have creator
CarouselImage	User	Many-to-One	No - must have uploader

# 6. Bulk Upload Feature (NEW)

## Overview

The bulk upload feature allows admins to add multiple users to the system at once by uploading a CSV file. This is much faster than registering users one-by-one.

## Files Involved

File	Location	What it Does
<b>AdminBulkUpload.js</b>	client/src/pages/	Frontend page with file upload UI
<b>adminRoutes.js</b>	server/routes/	Defines POST /api/admin/bulk-upload endpoint
<b>App.js</b>	client/src/	Defines route /admin/bulk-upload
<b>Dashboard.js</b>	client/src/pages/	Shows "Bulk Upload Users" card

## CSV Format Required

```
name,email,password,role John Doe,john@example.com,password123,alumni Jane
Smith,jane@example.com,password456,alumni Ahmed
Khan,ahmed@example.com,pass789,student
```

## Upload Process

User selects CSV file ↓ Frontend parses CSV ↓ Shows preview of users to upload ↓ User clicks "Upload Users" ↓ POST /api/admin/bulk-upload (with user data) ↓ Server validates each user:  
 - Check required fields - Check email doesn't exist - Hash password - Save to database ↓  
 Return results (success count, failed count) ↓ Show detailed report to admin

## Backend Validation

- **Required fields:** name, email, password
- **Email uniqueness:** No duplicate emails allowed
- **Password hashing:** All passwords hashed with bcryptjs
- **Default role:** "alumni" if not specified
- **Auto-approval:** Bulk users marked as "approved" by default

## 7. Complete Request Flow Examples

### Example 1: User Registration

**User Action:** Clicks "Register" and fills form

1. User fills form: name, email, password, role
2. Register.js calls axios.post("/api/auth/register", data)
3. api.js doesn't add token (no login yet)
4. Server receives POST /api/auth/register
5. authRoutes directs to authController.register()
6. Controller checks if email exists
7. Controller hashes password with bcryptjs
8. Controller saves new User with status="pending"
9. Server responds: "Registration successful"
10. Frontend shows: "Waiting for admin approval"

### Example 2: Admin Bulk Upload

**Admin Action:** Uploads CSV with 50 new users

1. Admin on AdminBulkUpload.js selects users.csv
2. Frontend parses CSV → 50 user objects
3. Admin clicks "Upload Users"
4. api.js adds JWT token automatically
5. POST /api/admin/bulk-upload with 50 users
6. authMiddleware checks token ✓
7. adminMiddleware checks if user is admin ✓
8. adminRoutes directs to controller
9. Controller loops through each user:
  - Validate name, email, password present
  - Check if email exists (fail if duplicate)
  - Hash password
  - Save to database
10. Collect results: 48 succeeded, 2 failed
11. Send back detailed report
12. Frontend displays success/failure list

### Example 3: View Alumni Directory

**User Action:** Clicks "View All Alumni"

1. User clicks "View All Alumni" link 2. Router navigates to /alumni/directory 3. AlumniDirectory.js loads 4. useEffect calls fetchProfiles() 5. api.js adds JWT token 6. GET /api/alumni/profiles 7. authMiddleware checks token ✓ 8. alumniRoutes directs to controller.getAllProfiles() 9. Controller queries: - Get all approved alumni from User collection - Get all AlumniProfiles from AlumniProfile collection - Merge them (show all users + their profiles if exists) 10. Return array of 150 alumni 11. Frontend displays cards with name, department, company

## 8. Folder & File Importance Summary

### Critical Files (★★★★★ Must-Have)

File/Folder	Location	Why Critical
server.js	server/	Starts the entire backend server
.env	server/	Contains database credentials and secrets
config/db.js	server/config/	Connects to MongoDB database
models/	server/models/	Defines all data structures
routes/	server/routes/	Defines all API endpoints
controllers/	server/controllers/	Contains all business logic
middleware/	server/middleware/	Protects routes with auth/permissions
App.js	client/src/	Defines all page routes
index.js	client/src/	Starts React application
services/api.js	client/src/services/	Handles all backend communication
pages/	client/src/pages/	All user-facing pages

### Important Files (★★★★)

- **package.json** (both client and server) - Dependencies management
- **Navbar.js** - Navigation on every page
- **Dashboard.js** - Main hub showing all features
- **public/index.html** - HTML entry point for React

## Useful Files ( ★ ★ ★ )

- **styles/theme.js** - Design consistency
- **components/** - Reusable UI components
- **seedAdmin.js & seedAlumni.js** - Test data

## Nice-to-Have Files ( ★ ★ )

- **utils/constants.js** - App-wide constants
- **assets/** - Images and logos
- **.gitignore** - Ignore node\_modules and .env

# Conclusion

The Alumni Portal is a well-structured full-stack application with clear separation between frontend and backend. Each folder and file has a specific purpose contributing to the overall system:

- **Frontend** - Provides user interface and handles user interactions
- **Backend** - Manages data, enforces business rules, and secures the application
- **Database** - Stores all user data with proper relationships
- **Authentication** - Secures access with JWT tokens and password hashing
- **Bulk Upload** - New feature allowing efficient mass user registration

Understanding each file's role helps with maintenance, debugging, and adding new features.

---

Alumni Portal Documentation | Complete Architecture Guide

Generated: January 29, 2026

For questions or updates, refer to the respective source files.