Academic Staff Application System

Datkaiym Sagynbekova  
Kocaeli University  
Information Systems Engineering  
211307102@kocaeli.edu .tr

Julina Romialison  
Kocaeli University  
Information Systems Engineering  
211307106@kocaeli.edu .trHadeel Taqi  
Kocaeli University  
Information Systems Engineering  
211307108@kocaeli.edu .tr

***Abstract*—This report presents the design and implementation of a web-based Academic Staff Application System developed as a university project. The system streamlines the submission, evaluation, and scoring of academic applications based on university-specific guidelines (KOÜ Atama Yönergesi). Built using Node.js, MongoDB, and React, the project handles multiple user roles, category-based scoring, secure authentication, and file handling. This document outlines the architectural decisions, implementation details, challenges encountered, and future enhancements.**

***Keywords—Academic application, Node.js, MongoDB, React, Role-based access control, Scoring system, MERN stack***

# Introduction

Universities often encounter challenges in managing academic staff applications due to the complexities of evaluation criteria and guidelines. The objective of this project is to automate the evaluation process, facilitating efficient interactions among various user roles, including applicants, administrators, jury members, and managers. The project, available at “<https://github.com/jujuGthb/AcademicApp>” is developed based on Kocaeli University’s application regulations, aiming to translate these guidelines into a scalable and secure digital platform.

# System Architecture

The system adopts a modular client-server architecture, illustrated in *Figure 1* and *Figure 2*, consisting of three main layers: a React-based frontend, a Node.js (Express) backend, and a MongoDB database. The backend handles core logic, routing, and validation through middleware like Helmet and custom role guards. MongoDB is used for document-based storage, with file uploads managed via GridFS. Integration with AWS S3 is planned for future scalability. Authentication uses short-lived JWTs, hashed and stored securely on the server. On the frontend, React and Formik manage user interfaces and forms, communicating with the backend through RESTful APIs.

Architecture Overview:

* Frontend: React + Formik (with planned enhancements)
* Backend: Node.js + Express + JWT + Helmet
* Database: MongoDB (transitioning to AWS S3 for file storage)
* Deployment: Continuous Delivery (CD) approach

# User Roles & Workflow

As shown in *Table 1* applicants upload documents via a multi-step form. Jury members assigned to announcements can comment and recommend decisions. Managers configure scoring requirements dynamically.

# Scoring Logic

The scoring logic in this system adheres strictly to Kocaeli University's application regulations. Each announcement defines a set of categories, such as A, B, and C, where each category contains one or more items (e.g., A1, A2, B1). Categories may include a “k” coefficient, which acts as a multiplier on the base score, allowing for differential weighting based on institutional priorities. The overall score for an applicant is calculated on the backend using a formula that sums all item scores, each multiplied by its assigned weight and any applicable coefficient. In the case of co-authored works, specific rules apply to adjust the score, such as dividing or scaling based on the number of contributors.

The scoring logic is fully embedded in the backend and database, and dynamically responds to data changes. Managers can edit both categories and their corresponding weights through a dedicated administrative interface. Additionally, the system enforces both minimum and maximum score constraints per category. It also supports real-time validation to determine whether applicants satisfy category-specific requirements—for example, confirming that the combined score of items A1 through A4 meets or exceeds a threshold of 80.

1. User Roles

| **User Role** | ***Permissions*** |
| --- | --- |
| **Applicant** *(default)* | - Register and authenticate via TC number  -Submit application form  -Upload supporting documents  -Track application status |
| **Jury Member** | - View assigned announcements  -Review applicant documents  -Submit scores and comments |
| **Manager** | - Create/edit announcements  -Define category weights and minimum score rules  -Assign jury members  -Review jury evaluations |
| **Administrator** | -Full access to all system modules  -Manage users, roles, and system settings  -Override and audit submissions |

* Role-based access control (RBAC) is enforced server-side. Each action is authorized based on role-specific permissions defined in the backend logic. Users can hold only one role per session.

# Database Schema

MongoDB is used for flexible document storage. Main collections include:

* Users (with roles and hashed credentials)
* Announcements (with embedded scoring requirements)
* Categories
* Applications (linked to users and announcements)
* Files (currently stored locally, future S3 integration planned)

The system uses MongoDB with Mongoose for schema modeling. Core collections include User, Category, Announcement, and Application.

**Users** are stored with their TC number, personal details (name, surname, birth date), contact info (email, phone number), password (hashed), and optional education, CV, and profile picture fields. Each user is assigned a role: applicant, jury, manager, or admin. Users can log in, generate tokens, and are automatically filtered by isActive.

**Categories** represent evaluation areas like A, B, C, etc. Each category can define a minScore, maxScore, and whether a minimum threshold is required. Categories hold multiple items (e.g., A1, A2), each with a basePoints value and a flag indicating if the “k” coefficient applies to that item.

**Announcements** represent academic position postings. Each includes a title, faculty, description, and open positions (Dr. Öğr. Üyesi, Doçent, Profesör). Requirements per category (asgari, azami, kCoefficient) are stored in a flexible key-value Map, allowing dynamic updates. Announcements also track the deadline and creator.

**Applications** connect users to announcements. Each application includes the submitted form data, current status (draft, submitted, approved, etc.), total and detailed scores, and evaluations from assigned jury members. Each jury can upload a PDF evaluation, and submission times are logged.

This schema supports dynamic score validation, flexible role-based access, and scalable evaluation logic aligned with Kocaeli University’s academic appointment regulations.

# API Design

REST APIs cover user authentication, application submission, file upload, scoring calculation, and comment handling. Pagination is implemented for listing large datasets. SOAP is used for TC ID number verification.

## Authentication Routes

These endpoints handle user registration, login, and logout.

| **Method** | **Endpoint** | **Description** | **Access** |
| --- | --- | --- | --- |
| *POST* | /api/auth/register | Register a new user | Public |
| *POST* | /api/auth/login | Authenticate user and return token | Public |
| *POST* | /api/auth/logout | Logout and invalidate session | Public |

## User Management Routes

These routes manage user data. Access is restricted based on roles.

| **Method** | **Endpoint** | **Description** | **Access** |
| --- | --- | --- | --- |
| *GET* | /api/users/profile | Retrieve current user's profile | Authenticated |
| *PATCH* | /api/users/profile | Update user profile information | Authenticated |
| *GET* | /api/users/list | List all users | Admin only |
| *POST* | /api/users/create | Create a new user | Admin only |
| *GET* | /api/users/:id | Retrieve user by ID | Admin only |

## Category and Item Routes

These endpoints handle user registration, login, and logout.

| **Method** | **Endpoint** | **Description** | **Access** |
| --- | --- | --- | --- |
| *GET* | /api/categories | Fetch all categories with items | Public |
| *GET* | /api/categories/:code | Get category by its unique code | Public |
| *POST* | /api/categories | Create a new category | Admin only |
| *PUT* | /api/categories/:code | Update a category and its items | Admin only |
| *DELETE* | /api/categories/:code | Delete a category | Admin only |

## Application Submission Route

This is the core functionality where applicants submit their academic applications. It includes automatic scoring and validation against position-specific requirements.

| **Method** | **Endpoint** | **Description** | **Access** |
| --- | --- | --- | --- |
| *POST* | /api/application/submit | Submit an application for scoring and requirement check | Authenticated |

All other API endpoints are documented and available in the project’s GitHub repository.

# Frontend Implementation

# The frontend is built using React and Formik for form management. Multi-step forms guide applicants through submissions. Jury members and managers have role-based interfaces. Texts are currently hardcoded, with future plans for i18n.

# Challenges

The Kocaeli University application regulations complex scoring and category dependencies required deep analysis and testing to convert into programmable rules. Early drafts, based on CI, faced frequent integration errors, prompting us to switch to a CD-focused rebuild for smoother development.

As an educational project, our team lacked experience with large systems and role separation, causing delays in design clarity. TC ID form validation, with SOAP integration for Turkish authentication, added extra complexity. Without a mentor, we relied solely on self-learning and documentation, which slowed progress and limited guidance.

##### Acknowledgment

We would like to express our sincere gratitude to our friend **Francky Ronsard** *(Student ID 211307100)* for his continuous support, insightful suggestions, and technical assistance throughout the development of this project. His contributions were instrumental in shaping our understanding and improving the quality of our work.

We also extend our appreciation to the **open-source community**, whose freely available tools, libraries, and documentation significantly empowered our development process. This project would not have been possible without the collective effort and generosity of developers worldwide.

##### References

**[1] L. Lukas**, “Node.js Express JWT Authentication with MongoDB & Roles,” *Corbado Blog*, Jan. 15, 2025. [Online]. Available: https://www.corbado.com/blog/nodejs-express-mongodb-jwt-authentication-roles.

**[2] N. Patil**, “Mastering Role-Based Authorization in Your MERN App: A Comprehensive Guide with JWT and Passport.js,” *Medium*, Oct. 5, 2023. [Online]. Available: https://medium.com/@nil041297/mastering-role-based-authorization-in-your-mern-app-a-comprehensive-guide-with-jwt-and-passport-js-d7693143b33b.

**[3] S. Dubey**, “MERN: Creating a Full Stack Authentication System Using JWT and Bcrypt — Part 1,” *Stackademic Blog*, Jan. 31, 2024. [Online]. Available: https://blog.stackademic.com/mern-creating-a-full-stack-authentication-system-using-jwt-and-bcrypt-part-1-b0c773ec4af0.

**[4] N. Chowdhury**, “Full Stack Learning Management System (LMS) with MERN Stack: A Complete Guide,” *Dev.to*, Sep. 2024. [Online]. Available: https://dev.to/nadim\_ch0wdhury/full-stack-learning-management-system-lms-with-mern-stack-a-complete-guide-4dg0.

**[5] A. Sharma**, “NodeJS and MongoDB Application Authentication by JWT,” *LoginRadius Blog*, Mar. 20, 2020. [Online]. Available: https://www.loginradius.com/blog/engineering/nodejs-and-mongodb-application-authentication-by-jwt/.

**[6] Yogndrr**, “Yogndrr/MERN-School-Management-System,” *GitHub Repository*, 2025. [Online]. Available: https://github.com/Yogndrr/MERN-School-Management-System.