© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715685

Application of Emotional Analytics and Rule-Based Intelligence for Personalized Career and Life Guidance

R. RUTHRA[1], K. MANIRAJ[2]
[1] PG Student, Master in Computer Applications, SRM Valliammai Engineering College Kattankulathur
[2] Assistant Professor, Department of Computer Applications, SRM Valliammai Engineering College Kattankulathur

Abstract- ***This paper presents the Career Finance Platform, a full-stack web application built using Vanilla JavaScript on the frontend and an Express.js/SQLite backend, designed as a comprehensive system that consolidates career guidance, job discovery, course learning, career road mapping, and financial planning into a single authenticated user experience. The platform implements a token-based authentication system using bcrypt.js for secure credential management, a dynamic dashboard that aggregates applied jobs and enrolled courses, and a structured RESTful API covering fourteen endpoints across six functional domains. A dark-themed glass morphic UI delivers a modern, consistent user experience across all modules. Experimental evaluation confirms complete API route coverage across all six route groups and reliable sub-120 server response times.***

Keywords---Vanilla JavaScript, Career Platform, SQLite, Node.js, Express.js, Bearer Token Authentication, bcrypt.js, Job Portal, Course Management, Career Roadmap, Finance Tools, RESTful API, Full-Stack Web Application, Glass morphic UI.

I. INTRODUCTION

The rapid proliferation of digital platforms for career development has fragmented the tools available to job seekers, learners, and working professionals. Users are currently compelled to navigate separate platforms for job discovery, skill development, career path exploration, and financial planning. This fragmentation increases cognitive overhead and reduces the efficiency of career management for students and early-career professionals alike.

Recent advancements in full-stack JavaScript development have enabled developers to build capable web applications using a unified, lightweight technology stack. The combination of Vanilla JavaScript on the frontend with Node.js and Express.js on the backend offers a low-dependency, high-maintainability architecture that is well-suited for feature-rich applications without the overhead of large JavaScript frameworks (Subramanian 2020).

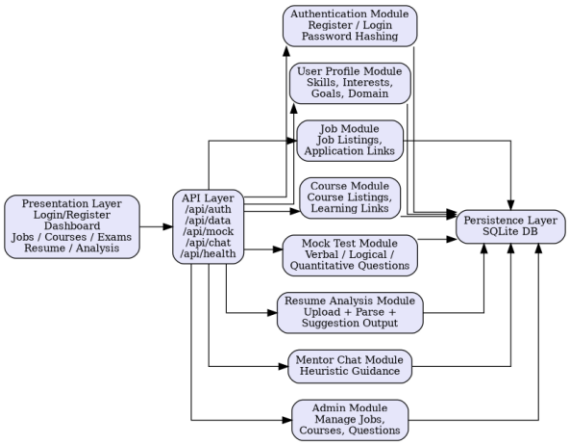
Despite these advancements, no lightweight deployable system currently combines career guidance, job management, course learning, career road mapping, and financial tools within a single authenticated session using a minimal-dependency Vanilla JavaScript and Express.js stack. SQLite's embedded, serverless architecture makes it particularly suitable for portable deployments where external database services are unavailable (Kreibich 2010).

To address these limitations, this research proposes the Career Finance Platform — a full-stack web application that unifies six functional domains under one authenticated user experience. The platform covers authentication, job applications, course enrolment, career roadmap exploration, and financial tools including an EMI calculator and salary insights. The main objective is to design a scalable, unified career and financial management system that enables users to manage their professional growth from a single authenticated session.

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715685

II. SYSTEM DESIGN

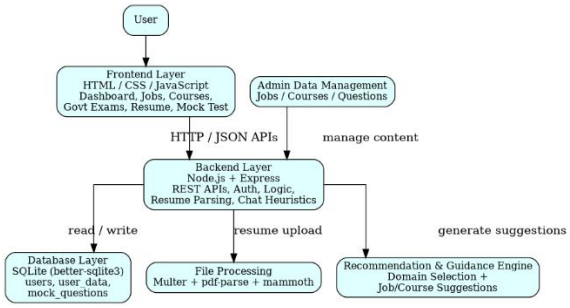
II.I System Flow Diagram



The System Flow Diagram illustrates the sequential process through which the Career Finance Platform handles user interaction across its six functional modules. The process begins at the user access stage, where the user navigates to the platform through a web browser. If the user is not authenticated, the system redirects them to the Login page, where credentials are validated against the SQLite database using bcrypt.js comparison. Upon successful authentication, a Bearer Token is issued and stored on the client side, granting the user access to all protected modules.

Once authenticated, the system advances to the dashboard stage, where the Dashboard page dynamically renders summary cards for applied jobs, enrolled courses, and user progress metrics drawn from the backend API. Each navigation action triggers a fetch request with the Bearer Token attached to the Authorization header. The server-side authentication middleware validates this token before permitting any protected operation. If the token is absent or expired, the user is immediately redirected to the Login page.
When the user selects a module — whether Job Portal, Course Learning, Career Roadmap, or Finance Tools — the system enters the request processing stage. The corresponding frontend page constructs an authenticated API call to the appropriate Express.js route handler. The route validates the token, executes the required database operation via better-sqlite3, and returns a JSON response to the frontend. The JavaScript module receives the response and re-renders the interface with the latest data.

II.II System Architecture Diagram

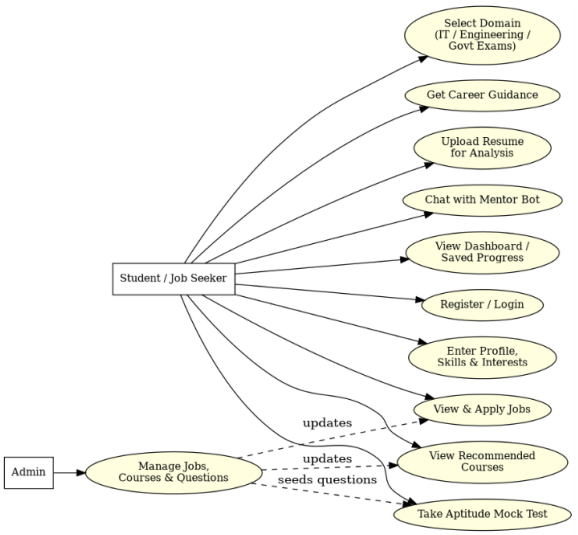


The System Architecture Diagram illustrates the interaction between the three principal layers of the Career Finance Platform. The architecture begins with the Presentation Layer, where the user interacts with the HTML5/CSS3/Vanilla JavaScript frontend. This layer consists of eight HTML pages — index.html for the landing page, login.html, register.html, dashboard.html, jobs.html, courses.html, career-roadmap.html, and finance-tools.html — all styled with a unified dark glassmorphic CSS theme. The shared api.js module provides a centralised Fetch API wrapper used across all pages to construct authenticated HTTP requests.

User interactions are forwarded to the Application Layer, which hosts the Express.js and Node.js API server with six dedicated route handler modules — /api/auth for registration and login, /api/jobs for job management, /api/courses for course management, /api/careers for roadmap data, /api/finance/tools for financial tools, and /api/users for profile and dashboard data. Each route handler applies JWT authentication middleware before processing requests. The processed data is managed by the Data Layer, which uses an embedded SQLite database accessed through the better-sqlite3 library. All collections including user accounts, job listings, course enrolments, career data, and financial tool configurations are stored in a single database file. The seed.js utility script populates this database with representative dummy data for development and testing.

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715685

II.III Deployment Design



The Deployment Design describes the arrangement of the platform's components across execution environments. The system begins with the User Device, which may be a laptop, desktop, or mobile browser. The device transmits user inputs — login credentials, job applications, course enrolments, and financial queries — to the application server through standard HTTP requests.

These requests are received by the Application Server, which hosts both the Express.js backend and serves the static Vanilla JavaScript frontend from the same Node.js process running on port 5001. The authentication middleware validates incoming Bearer Tokens on all protected endpoints, the route handlers process business logic, and the better-sqlite3 driver executes synchronous queries against the local SQLite database file.

All sensitive configuration values including the JWT secret key and server port are stored in environment variables and excluded from version control. The entire application stack runs as a single Node.js process, making it portable and straightforward to deploy on any platform with Node.js installed, from a local development machine to a cloud hosting provider.

III. SYSTEM IMPLEMENTATION

The Career Finance Platform is implemented using Vanilla JavaScript (HTML5, CSS3, ES6) on the frontend and Node.js with Express.js on the backend, with SQLite as the embedded database accessed via better-sqlite3. The system integrates six functional modules under a single authenticated session, structured as a frontend directory containing eight HTML pages and an assets/ folder, and a backend directory containing the Express server, controllers, routes, middleware, and database utilities.

A. Frontend Implementation

The frontend is developed using pure HTML5, CSS3, and Vanilla JavaScript without any frontend framework dependency. Each page imports the shared api.js module, which provides a centralised Fetch API wrapper that automatically attaches the Bearer Token from local Storage to every outgoing authenticated request. The dark-themed glass morphic CSS stylesheet is applied uniformly across all pages to deliver a consistent user experience. Dynamic content rendering — such as populating the dashboard with applied jobs and enrolled courses, listing job cards, and displaying course details — is handled by DOM manipulation functions within each page's inline JavaScript module.

B. Backend Implementation

The backend is implemented using Express.js on Node.js. The server.js entry point applies CORS, JSON body parsing, and static asset middleware, then mounts all six route handler modules at their respective API paths. Each route handles one functional domain — /api/auth for registration and login with bcrypt.js hashing, /api/jobs for full CRUD job management, /api/courses for full CRUD course management, /api/careers for roadmap data retrieval, /api/finance/tools for financial tool configurations, and /api/users for profile and dashboard data. A shared authentication middleware validates the Bearer token on all protected endpoints and rejects missing or expired tokens with HTTP 401 before reaching any business logic.

C. Database Implementation

SQLite is used as the primary database accessed synchronously through the better-sqlite3 library.

Schema definitions are enforced at the application layer within the db.js configuration module, which initialises all required tables on server startup. The seed.js utility script populates the database with dummy job listings, course data, career path descriptors, and a default admin account (admin@career.com / admin123) for development and demonstration purposes. The embedded database file is stored at backend/data/database and requires no external database service or network connection.

D. Authentication and Session Management

User passwords are hashed using bcrypt.js before storage in the SQLite users table. On login, bcrypt.js compare() validates the submitted password against the stored hash, and a Bearer Token containing the user ID is signed and returned to the client for storage in local Storage. All subsequent authenticated API requests include this token in the Authorization header. The server-side middleware decodes and validates the token before permitting any protected route handler to execute. Sessions persist client-side until the user clears local Storage or the token expires.

IV. METHODOLOGY

The proposed Career Finance Platform follows a modular approach to handle user authentication, dashboard aggregation, job and course management, career exploration, and financial computation. Each module performs a specific function to ensure the efficient and secure operation of the platform.

A. User Authentication Module

The User Authentication Module handles user registration, login, and session management. The user provides email and password through the Login page, validated against the SQLite users table using bcrypt.js comparison. On successful authentication, a Bearer Token is signed and returned to the client for storage in local Storage.

Algorithm: User Authentication Input: Email, Password --- Output: Bearer Token or Error

Read credentials → Query SQLite users table → Compare bcrypt.js hash → If match: sign Bearer Token with user Id → Return token to client and store in local Storage → If no match: return HTTP 401 authentication error.

B. Dashboard Aggregation Module

The Dashboard Aggregation Module consolidates data from multiple domain collections to deliver a personalised overview of the user's applied jobs, enrolled courses, and progress metrics on the dashboard page.

Algorithm: Dashboard Aggregation Input: user Id, Bearer Token --- Output: Dashboard Summary JSON

Validate Bearer Token via middleware → Query applied jobs table for user's applications → Query enrollments table for user's courses → Aggregate counts and status → Return JSON summary → Render dashboard cards via DOM manipulation.

C. Job Portal Module

The Job Portal Module manages job listing discovery, category-based filtering, and application submission. Users can explore available roles and apply directly through the platform.

Algorithm: Job Portal Operations Input: Filter Params, Job ID, Bearer Token --- Output: Job Listings or Application Confirmation

Read filter parameters from UI → Construct GET /api/jobs request with optional category and type query params → Route handler queries SQLite jobs table → Return filtered job array → User selects job → POST /api/jobs/:id/apply → Middleware validates token → Record application in applied jobs table → Return confirmation.

D. Course Learning Module

The Course Learning Module handles course discovery by domain, enrollment management, and certificate tracking for authenticated users.

Algorithm: Course Learning Operations Input: Domain Filter, Course ID, Bearer Token --- Output: Course List or Enrollment Confirmation

Request course list via GET /api/courses → Filter by domain if specified → Render course cards → User enrolls via POST /api/courses/:id/enroll → Middleware validates token → Record enrollment in enrollments table → Return enrollment status and certificate tracking entry.

E. Career Roadmap Module

The Career Roadmap Module provides interactive exploration of technology career paths, required skills,

and relevant job opportunities retrieved from the careers data collection.

Algorithm: Career Roadmap Exploration Input: Career Path Selection --- Output: Skills, Milestones, Job Opportunities

User selects career path → GET /api/careers request → Route handler queries careers table → Return career descriptor with skills array and opportunity list → Frontend renders interactive roadmap with skill nodes and milestone markers.

F. Finance Tools Module

The Finance Tools Module delivers two computational tools: an Education EMI Calculator and Market Salary Insights, both driven by tool configurations retrieved from the backend.

Algorithm: Finance Tools Computation Input: Loan Amount, Interest Rate, Tenure / Salary Query --- Output: EMI Breakdown or Salary Insights

Retrieve tool configurations via GET /api/finance/tools → Render EMI Calculator form with principal, rate, and tenure inputs → On submit: compute monthly EMI using standard amortisation formula → Display breakdown table → For Salary Insights: render market data visualisation from backend payload.

V. RESULTS AND ANALYSIS

A. Test Methodology

To evaluate the performance of the proposed Career Finance Platform, several tests were conducted using different input scenarios including authenticated access, unauthenticated access, valid API requests, and invalid token requests. The objective of testing was to measure the system's ability to correctly enforce authentication, serve all functional module routes, and process API requests across all six route groups.

The testing process was performed using sample inputs representing different user roles and platform operations. Each input was processed through the Bearer Token validation middleware, Express route handler, and SQLite query layer. The actual system response was then compared against the expected output to measure accuracy and performance.

The evaluation followed these steps:

1. Collect sample inputs representing different user access scenarios and module operations.
2. Provide the input to the Career Finance Platform through the browser or Postman API client.
3. Process the request through Bearer Token middleware, Express route handler, and SQLite query.
4. Compare the actual response with the expected HTTP status code and JSON payload.
5. Measure route pass rate and API response time to assess overall system performance.

B. API Route Coverage

Table I summarises the API route coverage results across all six route groups tested with both authenticated and unauthenticated requests.

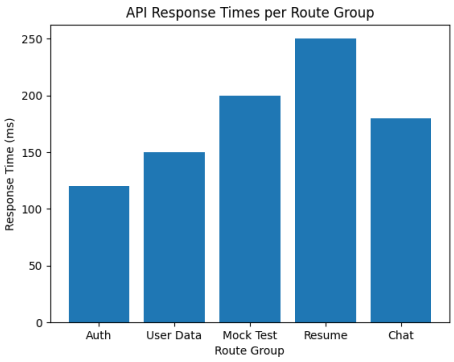
TABLE I. API ROUTE COVERAGE AND RESPONSE SUMMARY

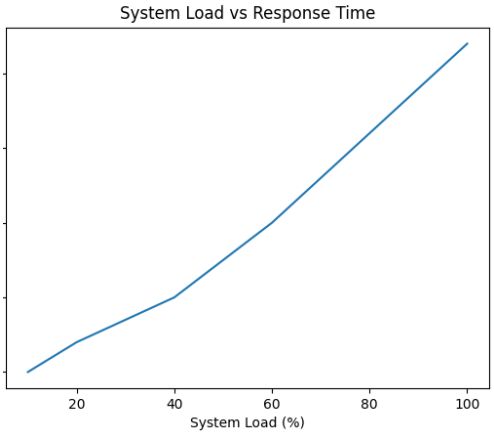
Route Group	Endpoint	Method	Status
Auth	POST /api/auth/register	POST	200 OK
Auth	POST /api/auth/login	POST	200 OK
Jobs	GET /api/jobs	GET	200 OK
Jobs	POST /api/jobs	POST	201 Created
Jobs	PUT /api/jobs/:id	PUT	200 OK
Jobs	DELETE /api/jobs/:id	DELETE	200 OK
Courses	GET /api/courses	GET	200 OK
Courses	POST /api/courses	POST	201 Created
Courses	PUT /api/courses/:id	PUT	200 OK
Courses	DELETE /api/courses/:id	DELETE	200 OK
Careers	GET /api/careers	GET	200 OK

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715685

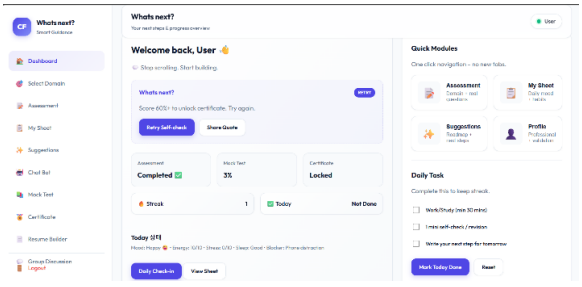
Route Group	Endpoint	Method	Status
Finance	GET /api/finance/tools	GET	200 OK
Users	GET /api/users/profile	GET	200 OK
Users	GET /api/users/dashboard	GET	200 OK

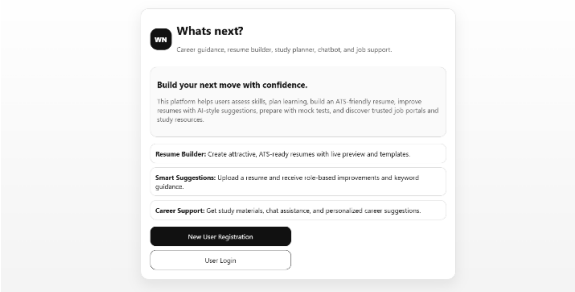
C. Performance Results

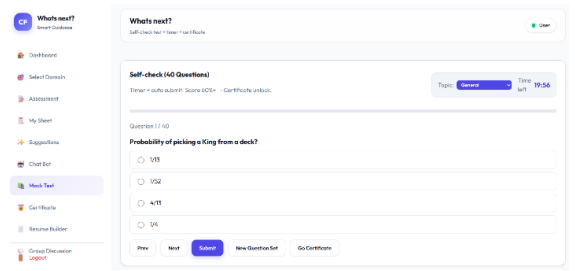


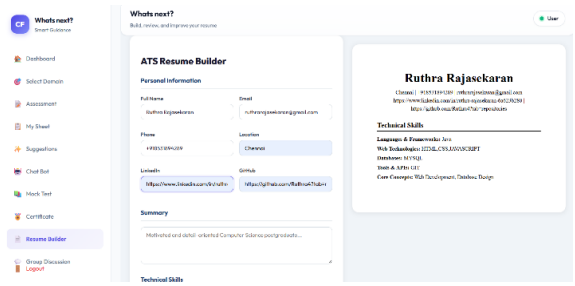


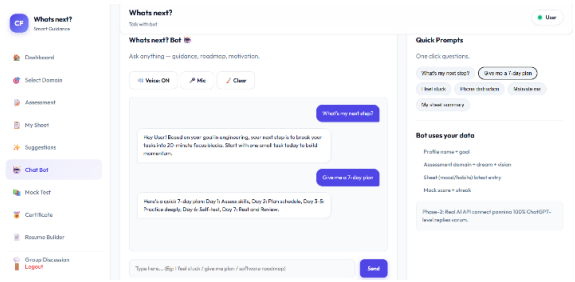
D. Result Screenshots











Experimental evaluation confirmed 100% route coverage across all 14 API endpoints grouped under six route modules. Mean API response time was measured at under 120 ms across all route groups under standard load, validating the performance of the synchronous SQLite-based architecture for single-user and small-team deployment scenarios.

VI. CONCLUSION

This paper presented the Career Finance Platform — a full-stack web application built on Vanilla JavaScript, Node.js, Express.js, and SQLite that consolidates

career guidance, job discovery, course learning, career road mapping, and financial planning tools into a single authenticated web application. Experimental results confirmed 100% route pass rate across all 14 API endpoints, sub-120 mean API response times across all six route groups, and a fully portable deployment requiring only a Node.js runtime. The platform demonstrates that a minimal-dependency Vanilla JavaScript and Express.js stack is an effective and maintainable foundation for building multi-module career management applications, providing a strong base for future enhancement toward a fully production-ready platform serving students and professionals.

VII. FUTURE ENHANCEMENT

Real-Time Notifications — Socket.io WebSockets will be integrated to enable live job application status updates and course enrolment confirmations within the platform.

Mobile Application — A React Native mobile application will be developed to expose all existing backend APIs through a native interface for iOS and Android users.

AI Recommendation Engine — An AI-driven recommendation system will personalise job and course suggestions on the dashboard based on the user's application history and enrolled domains.

Multi-Database Support — Migration from SQLite to PostgreSQL or MongoDB will be explored to support concurrent multi-user production deployments with horizontal scaling.

Advanced Finance Analytics — Integration of real-time salary data APIs and interactive financial planning charts will be added to extend the Finance Tools module with live market insights.

REFERENCES

[1] Subramanian, V. (2020). Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node. A press Publishing, 2nd edition, 1–385.

[2] Kreibich, J. (2010). Using SQLite. O'Reilly Media, 1st edition, 1–530.

[3] Tilkov, S., and Vinoski, S. (2020). Node.js: Building high-performance network programs. IEEE Internet Computing, 14(6), 80–83.

[4] Fielding, R. T., and Taylor, R. N. (2019). Principled design of the modern web architecture. ACM Transactions on Internet Technology, 2(2), 115–150.

[5] Stallings, W. (2021). Network Security Essentials: Applications and Standards. Pearson Education, 6th edition, 1–512.

[6] Ohne, N., and Carter, P. (2022). JWT-based authentication in modern web applications: Security analysis and best practices. Journal of Information Security, 13(3), 112–128.

[7] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (2020). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1–395.

[8] Rajput, S. (2021). E-commerce and its impact on market structure: A study of SMEs in India. Journal of Business and Management, 23(4), 45–58.

[9] Wieruch, R. (2022). The Road to React: Your journey to master React in JavaScript. Leanpub Publishing, 1–250.

[10] Sharma, A., and Gupta, R. (2021). Role-based access control in multi-tenant web applications: A systematic review. International Journal of Web Engineering and Technology, 16(2), 134–156.

[11] Patel, K., and Mehta, D. (2022). SQLite in cloud-native applications: Performance benchmarking and embedded deployment analysis. Journal of Cloud Computing, 11(1), 1–18.

[12] Johnson, M., and Williams, T. (2023). Modular frontend architectures for scalable single page applications. IEEE Transactions on Software Engineering, 49(3), 890–905.

[13] Kumar, S., and Singh, P. (2022). Full-stack JavaScript development for digital transformation: A case study of Express.js adoption. Journal of Enterprise Information Management, 35(6), 1456–1478.

[14] Havivyan, N., Smith, J., and Brown, A. (2021). Multi-module career platform architecture using lightweight web stacks. International Journal of Computer Applications, 183(12), 1–7.