First: system performance profiling

❖ Overview

The SkillSync backend is designed using lightweight technologies — Node.js, Express, and MongoDB — which offer fast startup and efficient request handling. Performance is primarily dependent on request processing time, database efficiency, and file handling during resume uploads.

❖ Performance-Conscious Design Choices

Area	Technique or Practice	Impact
Express Middleware	Centralized body parsing and CORS handling	Reduces redundant checks, speeds up parsing
Lightweight Auth Flow	Stateless JWT authentication	No database session storage = faster login
File Upload Handling	`multer` stores files locally, not in memory	Prevents server memory overload
Resume Parsing	Done asynchronously with `pdf-parse` and `mammoth`	Avoids blocking the event loop
Database Queries	Mongoose with indexed fields like `email`	Fast lookup during login/register
Role-based Access Control	Checked via middleware before hitting route logic	Prevents unnecessary processing on invalid users

---

❖ Observed Performance Behavior (during testing)

- Register/Login completes in under 200ms for valid inputs
- Resume analysis takes 1–2 seconds (due to file read/parse), which is acceptable for the feature
- No memory or CPU spikes observed during local testing
- Uploading resumes over 1MB showed no delay or crash

Second: Security Vulnerability Analysis

❖ Overview

The SkillSync backend implements essential security practices to protect user data, ensure authorized access, and prevent common vulnerabilities. Key areas addressed include **authentication**, **authorization**, **input handling**, and **data protection**.

❖ Security Measures Implemented

Area	Implementation	Protection Provided
Password Security	Passwords hashed using `bcrypt` before storing	Prevents password leaks if database is compromised
Authentication	JWT-based login, token stored in headers	Prevents session hijacking and allows stateless APIs
Authorization	Middleware-based role checking	Restricts access to protected routes (e.g., admin only)
File Upload Validation	Accepts only PDF/DOCX, rejects others	Mitigates risk of executing harmful files
CORS Configuration	Enabled via `cors` middleware	Prevents basic cross-origin issues for web frontend
Error Handling	Consistent response codes + try/catch wrappers	Prevents leaking stack traces or internal details
Environment Variables	Secrets (JWT, DB URI) stored in `.env`	Keeps sensitive info out of source code

---

❖ Potential Vulnerabilities Still Present

Issue Area	Risk Description	Recommended Action
Input Validation	No advanced schema-level validation for inputs	Use `express-validator` to validate inputs on routes
File Size Limits	Multer accepts any file size	Add file size restrictions to avoid disk exhaustion
Token Expiry Handling	No refresh tokens implemented	Implement refresh token mechanism for better control
Upload Storage	Files are stored locally in `uploads/`	Use cloud storage (e.g., S3) in production

---

❖ Threat Mitigation Summary

- **Brute-force attacks** are mitigated by bcrypt hashing (but could be improved with rate-limiting middleware like `express-rate-limit`)
- **Access control** is strongly enforced through layered role-check middleware
- **Sensitive data** like passwords and JWT secrets are securely handled and never exposed

---

Third: optimizations and fixes applied

❖ Refactoring & Architecture Improvements

Area	Optimization Applied	Benefit
Service Layer Design	Extracted business logic to services (e.g., `AuthService`)	Increased modularity, easier to test and maintain
Design Patterns	Singleton, Factory, Observer implemented	Improved code scalability and separation of concerns

Role Middleware	Centralized auth & role checks into reusable functions	Cleaner route logic and consistent access control
Resume Processing	Moved PDF/DOCX parsing into `resumeController` asynchronously	Prevented blocking the event loop during file handling
Error Handling	Standardized JSON error responses	Safer and more predictable client communication

---

❖ Bug Fixes During Testing

Bug / Issue	Resolution
Missing `path` import in resume upload	Added `const path = require('path')` to controller
Crashing on unauthorized resume upload	Added checks and returned `401` / `403` status codes
Token not recognized in Postman	Resolved by confirming correct `Bearer` format in header
Role misuse (mentor uploading resume)	Fixed via `restrictToRole('student')` middleware
MongoDB not showing `skillsync` DB	Triggered collection creation by registering a user

---

❖ Tools and Libraries Used

- **Multer** for safe file uploads (handled via disk storage)
- **JWT** for stateless, secure user sessions
- **Mongoose** ODM for schema enforcement and queries
- **PDF-parse / Mammoth** for extracting content from resume files
- **Bcrypt** for password hashing

❖ Summary

SkillSync's backend evolved significantly with strong architectural refactoring, secure middleware, and practical optimizations. Testing and debugging phases helped surface and resolve real-world issues, resulting in a secure, scalable, and maintainable system.