# Code Review & Improvements

Below is the review of your Node.js + MongoDB Event Booking API project based on the uploaded codebase. You mentioned that you spent around 6 hours 30 minutes building it. This review considers structure, code quality, security, performance, best practices, and maintainability.

## Overall Rating

Rating: 3.5 / 5

Given the time constraints, the project is decent. However, there is room for improvement in terms of validation, error handling, modularity, and security practices.

## Strengths

✅ Clear project structure with src folder separation.

✅ Usage of TypeScript for type safety.

✅ Middleware-based authentication setup.

✅ Environment variables usage for sensitive configuration.

✅ Organized routes and controllers separation.

✅ Models defined using Mongoose schemas.

## Key Improvements

1. \*\*Validation\*\* – Use a library like Joi or Zod for validating incoming request bodies in controllers before processing.

2. \*\*Error Handling Middleware\*\* – Implement a global error handler to avoid repetitive try/catch in every controller.

3. \*\*Consistent Response Format\*\* – Create a unified success/error response format to maintain consistency.

4. \*\*Security\*\* – Hash passwords before saving using bcrypt. Currently, password handling logic needs verification.

5. \*\*Token Expiration & Refresh\*\* – Implement token expiry checks and refresh token flow for better session management.

6. \*\*Type Safety in Services\*\* – Ensure ObjectId types are handled using `mongoose.Types.ObjectId` to avoid CastError.

7. \*\*DTO Layer\*\* – Introduce a Data Transfer Object pattern for cleaner controller-service communication.

8. \*\*Logging\*\* – Add Winston or pino logging for better debugging and production monitoring.

9. \*\*Testing\*\* – No unit or integration tests found; add Jest or Mocha tests for API endpoints.

10. \*\*Deployment Readiness\*\* – Add production scripts, Dockerfile, and AWS/GCP deployment configurations.

11. \*\*Pagination & Filtering\*\* – For `GET` APIs fetching multiple records, implement pagination to avoid performance bottlenecks.

12. \*\*Database Indexes\*\* – Add indexes to frequently queried fields for performance gains.

13. \*\*API Documentation\*\* – Provide API documentation via Swagger/OpenAPI for easier collaboration.

14. \*\*Input Sanitization\*\* – Sanitize input to prevent injection attacks.

15. \*\*Code Comments\*\* – Add more inline comments for complex business logic.

## File-Specific Notes

\*\*auth.controller.ts\*\* – Missing request validation. Error handling can be centralized.

\*\*user.controller.ts\*\* – Lacks try/catch in some async calls. Ensure user data is properly sanitized.

\*\*Authenticate.ts\*\* – Decoding JWT is fine, but handle expired token error cases gracefully.

\*\*Booking.ts Model\*\* – No schema-level validations (e.g., required fields, enum constraints).

\*\*db.config.ts\*\* – Use mongoose connection options like `useNewUrlParser` and `useUnifiedTopology`.

\*\*decodeUser.ts\*\* – Needs null checks before accessing decoded data.