SOFTWARE QUALITY ENGINEERING

**REPORT**

**(SE-G)**



**Submitted by**

Amna Shahid (21I-1148)

Fatima Qurban (21I-1195)

Ushna Nadeem (21I-1225)

**Submitted to**

Ma’am Saba Kanwal

**Checklist:**

**Package Name:** routes  
**File Name:** adminRoutes.js  
**Class / Interface Name:** N/A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S#** | **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| 1 | Naming Conventions | Are function names written in camelCase (e.g., toggleUserBlocked)? | Yes | None | N/A |
|  |  | Are variables named descriptively (e.g., user instead of u)? | Yes | None | N/A |
| 2 | Code Structure | Are middleware functions properly modularized? | Yes | Some redundancy in toggle logic for blocked and disabled attributes. | Refactor toggle logic into a utility function to ensure reusability. |
|  |  | Are HTTP methods aligned with RESTful standards? | Yes | None | N/A |
| 3 | Method Design | Do methods perform single responsibilities? | Partial | Toggle functionality is duplicated across different routes (users and blogs). | Refactor into a helper function to maintain DRY (Don’t Repeat Yourself) principles. |
| 4 | Exception Handling | Are exceptions handled correctly using try-catch blocks? | Yes | No centralized error-handling middleware for standardized response. | Create centralized error-handling middleware for better consistency and maintainability. |
|  |  | Are specific exceptions used instead of generic ones? | Yes | None | N/A |
| 5 | Code Readability | Are comments provided for complex logic? | Partial | Minimal comments explaining toggle logic and edge cases (e.g., Cannot block yourself). | Add descriptive comments to clarify the toggle logic and purpose of conditions. |
| 6 | Performance | Are repetitive queries (e.g., findById) minimized? | No | Duplicate database calls for findById within toggle routes. | Cache results or consolidate logic to avoid redundant queries. |
|  |  | Are unnecessary operations (e.g., redundant logging) avoided? | No | Use of console.log in blog disable route leads to unnecessary clutter in production logs. | Replace console.log with a logging library like winston for better performance and control. |
| 7 | Memory Management | Are resources (e.g., DB connections) properly managed and closed? | Yes | None | N/A |
| 8 | Security | Are user inputs validated to prevent injection or other vulnerabilities? | Partial | No validation for request payloads like blocked or block. | Use input validation middleware (e.g., express-validator) to validate payloads. |
| 9 | Maintainability | Is duplicate code minimized? | No | Toggle logic is repeated in both users and blogs routes. | Abstract toggle logic into a reusable helper function. |
| 10 | Test Coverage | Are unit tests provided for all endpoints? | Yes | Some edge cases (e.g., null values) are not covered in tests. | Add test cases to handle invalid inputs and null scenarios. |
| 11 | Test Design | Do tests follow the Arrange-Act-Assert (AAA) pattern? | Yes | None | N/A |
| 12 | Assertions | Are assertions used to verify expected outcomes? | Yes | None | N/A |
| 13 | Test Readability | Are test method names descriptive? | Yes | None | N/A |

**Summary:**

**Most Frequent Issues:**

* **Lack of Input Validation:** Request payloads (blocked, block) lack proper validation, leading to potential errors or vulnerabilities.
* **Use of console.log for Logging:** Errors and debug information are logged with console.log, which is unsuitable for production.
* **Redundant Code:** Toggle logic for blocked and disabled attributes is repeated across routes, violating DRY principles.
* **Complex Methods:** Methods like toggle functionality in /users/:id/disable and /blogs/:id/disable mix responsibilities, making them harder to maintain.
* **No Centralized Error Handling:** Error-handling logic is repeated in each route, leading to inconsistent responses.

**Suggested Refactoring:**

1. **Implement Input Validation:** Use middleware (e.g., express-validator) to validate request payloads and improve security.
2. **Use Logging Framework:** Replace console.log with a structured logging library like winston or pino for better error tracking.
3. **Refactor Toggle Logic:** Extract common toggle functionality into a helper function to reduce redundancy.
4. **Modularize Methods:** Break down large, multi-responsibility methods into smaller, more focused helper functions.
5. **Centralize Error Handling:** Introduce a centralized error-handling middleware to ensure consistent responses and cleaner route logic.

**Checklist:**

**Package Name:** middleware  
**File Name:** authMiddleware.js  
**Class / Interface Name:** N/A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S#** | **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| 1 | Naming Conventions | Are function names written in camelCase (e.g., authMiddleware)? | Yes | None | N/A |
|  |  | Are variables and parameters named descriptively (e.g., decoded, token)? | Yes | None | N/A |
|  |  | Are constants written in uppercase with underscores (e.g., JWT\_SECRET)? | Yes | None | N/A |
| 2 | Code Structure | Are access modifiers (e.g., private, public) used appropriately? | N/A | Middleware functions don't use access modifiers. | N/A |
|  |  | Are middleware functions properly modularized and reusable? | Yes | None | N/A |
|  |  | Are dependencies imported and utilized correctly? | Yes | None | N/A |
| 3 | Method Design | Does the middleware perform a single responsibility? | Yes | None | N/A |
|  |  | Are parameters kept to a reasonable limit? | Yes | None | N/A |
| 4 | Exception Handling | Are exceptions handled correctly using try-catch blocks? | Yes | Logging within catch is done via console.log, which is not suitable for production. | Replace console.log with a structured logging library like winston or pino. |
|  |  | Are specific exceptions used instead of generic ones? | Yes | None | N/A |
|  |  | Is logging implemented within catch blocks for debugging purposes? | Partial | Logging is present but uses console.log, which is not a best practice for production. | Use a structured logging library for better error tracking and maintainability. |
| 5 | Code Readability | Are meaningful comments provided for complex logic? | No | No comments explaining token verification or error handling logic. | Add comments to explain token extraction, verification, and associated error responses. |
|  |  | Is there consistent indentation (4 spaces or a tab size of 4)? | Yes | None | N/A |
|  |  | Are blank lines used appropriately to separate code blocks for better readability? | Yes | None | N/A |
|  |  | Are meaningful names used for variables and methods? | Yes | None | N/A |
| 6 | Performance | Are computationally expensive operations (e.g., token verification) handled efficiently? | Yes | None | N/A |
|  |  | Are unnecessary operations (e.g., redundant computations) avoided? | Yes | None | N/A |
| 7 | Memory Management | Are resources (e.g., tokens) properly validated and disposed after use? | Yes | None | N/A |
| 8 | Security | Is token validation performed to prevent unauthorized access? | Yes | None | N/A |
|  |  | Is sensitive information (e.g., secrets) handled securely? | Yes | None | N/A |
| 9 | Maintainability | Is the middleware reusable for other routes or scenarios? | Yes | None | N/A |
| 10 | Test Coverage | Are unit tests provided for all critical scenarios? | Partial | Edge cases, such as malformed tokens or missing process.env.JWT\_SECRET, are not covered. | Add tests for malformed tokens, missing cookies, and unexpected errors. |
| 11 | Test Design | Do tests follow the Arrange-Act-Assert (AAA) pattern? | Yes | None | N/A |
| 12 | Assertions | Are assertions used to verify expected outcomes? | Yes | None | N/A |
| 13 | Test Readability | Are test method names descriptive? | Yes | None | N/A |

**Summary:**

**Most Frequent Issues:**

**Logging with console.log:** Errors are logged with console.log, which is unsuitable for production environments.

* **Lack of Comments:** The logic for token validation and error handling lacks inline comments for better readability.
* **Incomplete Edge Case Tests:** Tests do not cover scenarios like malformed tokens, missing cookies, or absent environment variables.

**Suggested Refactoring:**

1. **Use a Logging Framework:** Replace console.log with a logging library like winston or pino to handle errors systematically.
2. **Add Inline Comments:** Document token validation, error handling, and edge cases to improve code readability and maintainability.
3. **Expand Test Coverage:** Add additional test cases for edge scenarios, such as malformed tokens, missing cookies, and unexpected errors.

**Checklist:**

**Package Name:** routes  
**File Name:** blogRoutes.js  
**Class / Interface Name:** N/A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S#** | **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| 1 | Naming Conventions | Are function names written in camelCase (e.g., getBlogPosts)? | Yes | None | NN /A |
|  |  | Are variables named descriptively (e.g., req.body.text instead of req.body.t)? | Yes | None | N/A |
| 2 | Code Structure | Are middleware functions properly modularized and reusable? | Partial | Middleware like getBlog is used but not extended for broader reusability across routes. | Refactor and expand the middleware for reuse across similar endpoints. |
|  |  | Are route paths consistent and aligned with RESTful standards? | Yes | None | N/A |
| 3 | Method Design | Do methods perform single responsibilities? | Partial | Some methods mix logic for pagination, sorting, and filtering in a single function. | Separate pagination, sorting, and filtering logic into helper functions. |
|  |  | Are parameters kept to a reasonable limit? | Yes | None | N/A |
| 4 | Exception Handlin | Are exceptions handled correctly using try-catch blocks? | Yes | None | NA |
|  |  | Are specific exceptions used instead of generic ones? | Yes | None | N/A |
|  |  | Is logging implemented for debugging and error tracking? | Partial | console.log is used for debugging in some routes, which is unsuitable for production. | Replace console.log with a logging library like winston or pino. |
| 5 | Code Readability | Are meaningful comments provided for complex logic? | Partial | Minimal comments for explaining pagination and sorting logic. | Add descriptive comments to clarify logic for pagination, sorting, and filtering. |
|  |  | Is there consistent indentation and spacing? | Yes | None | N/A |
| 6 | Performance | Are repetitive queries (e.g., findById) minimized? | No | Repeated calls to findById for blog and user data within the same request pipeline. | Use caching or consolidate queries where applicable. |
|  |  | Are unnecessary operations avoided? | Partial | Rate calculation performs unnecessary operations when ratings are invalid (out of range). | Validate ratings before calculations to avoid redundant operations. |
| 7 | Memory Management | Are resources like database connections and streams properly handled? | Yes | None | N/A |
| 8 | Security | Is user input validated to prevent injection or other vulnerabilities? | Partial | Missingvalidation for parameters like rate, sortBy, and sortOrder. | Use middleware like express-validator for input validation. |
| 9 | Maintainability | Is duplicate code minimized? | Partial | Pagination and sorting logic are duplicated across multiple routes. | Refactor pagination and sorting logic into shared utility functions. |
| 10 | Test Coverage | Are unit tests provided for all endpoints? | Yes | None | N/A |
|  |  | Are edge cases (e.g., invalid parameters or missing data) covered in tests? | Partial | Some edge cases, like invalid query parameters, are not thoroughly tested. | Add test cases for invalid query parameters and empty results. |
| 11 | Test Design | Do tests follow the Arrange-Act-Assert (AAA) pattern? | Yes | None | N/A |
| 12 | Assertions | Are assertions used to verify expected outcomes? | Yes | None | N/A |
| 13 | Test Readability | Are tesmethod names descriptive? | Yes | None | N/A |

**Summary:**

**Most Frequent Issues:**

* **Logging with console.log:** Errors and debug information are logged with console.log, which is unsuitable for production environments.
* **Duplicate Pagination Logic:** Pagination and sorting logic is repeated across multiple routes, reducing maintainability.
* **Lack of Input Validation:** Parameters like rate, sortBy, and sortOrder lack validation, making the API vulnerable to invalid data.
* **Overloaded Methods:** Some methods mix multiple responsibilities (e.g., /author/:id handles pagination, sorting, and filtering).

**Suggested Refactoring:**

1. **Use a Logging Framework:** Replace console.log with a structured logging library like winston or pino for better error tracking and debugging.
2. **Refactor Pagination Logic:** Extract pagination and sorting logic into reusable utility functions to avoid code duplication.
3. **Implement Input Validation:** Use middleware like express-validator to validate query parameters and request bodies.
4. **Modularize Methods:** Break down methods that handle multiple responsibilities into smaller, focused functions.
5. **Expand Test Coverage:** Add test cases for edge scenarios like invalid query parameters, missing required fields, and empty results.

**Checklist:**

**Package Name:** root  
**File Name:** server.js  
**Class / Interface Name:** N/A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S#** | **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| 1 | Naming Conventions | Are variables named descriptively (e.g., corsOptions instead of options)? | Yes | None | N/A |
|  |  | Are route paths named consistently (e.g., /user, /blog, /search)? | Yes | None | N/A |
| 2 | Code Structure | Is middleware usage consistent across routes? | Partial | Middleware like authMiddleware is applied selectively, leaving some routes unprotected. | Ensure all sensitive routes use authMiddleware unless explicitly designed for public access. |
|  |  | Are cross-origin headers properly configured? | Yes | None | N/A |
|  |  | Is configuration information (e.g., CORS, database connection) centralized? | No | CORS and MongoDB options are hardcoded within the file. | Extract configuration options to a separate config file for reusability and maintainability. |
| 3 | Method Design | Are methods performing single responsibilities? | Partial | The POST /login method handles both authentication and response generation. | Separate authentication logic from response generation into dedicated functions or services. |
|  |  | Are parameters kept to a reasonable limit? | Yes | None | N/A |
| 4 | Exception Handling | Are exceptions handled correctly using try-catch blocks? | Partial | Some routes (e.g., /login) do not handle errors consistently or send informative responses. | Standardize exception handling across all routes for consistency. |
|  |  | Is logging implemented for debugging and error tracking? | Yes | However, console.error is used, which is not ideal for production. | Use a structured logging library like winston or pino for error logging. |
|  |  | Are error responses consistent across routes? | Partial | Error messages differ in structure and HTTP status codes are sometimes inappropriate. | Define and use consistent error response structures for all routes. |
| 5 | Code Readability | Are meaningful comments provided for complex logic? | Partial | Comments for middleware, CORS setup, and database connections are missing. | Add comments to explain complex setups and design decisions. |
|  |  | Is there consistent indentation and spacing? | Yes | None | N/A |
| 6 | Performance | Are redundant database queries or calculations avoided? | Partial | The /login route queries the database without considering cached or optimized approaches. | Use caching or indexes for repeated lookups to improve performance. |
|  |  | Is middleware optimized for efficiency? | Yes | None | N/A |
| 7 | Security | Are user inputs validated to prevent injection or other vulnerabilities? | Partial | Missing validation for POST /register and POST /login payloads. | Use middleware like express-validator to validate inputs. |
|  |  | Are sensitive information (e.g., passwords) stored securely? | No | Passwords are stored in plain text, which is highly insecure. | Use hashing algorithms like bcrypt to securely store passwords. |
|  |  | Are JWTs generated and validated securely? | Partial | The JWT is generated without specifying critical options like expiresIn. | Include secure options like expiresIn and secure flags for cookie handling. |
|  |  | Are CORS settings restrictive enough to prevent unauthorized access? | Partial | Current CORS settings allow broad access (localhost), which may be exploited. | Restrict CORS settings to specific trusted origins. |
| 8 | Maintainability | Is duplicate code minimized? | Partial | Repeated logic in /register and /login for handling responses and errors. | Abstract common logic into helper functions. |
| 9 | Test Coverage | Are unit tests provided for all routes? | Yes | None | N/A |
|  |  | Are edge cases (e.g., invalid inputs, blocked users) tested? | Yes | Some edge cases like malformed JWTs or invalid request bodies are not tested. | Add test cases for these scenarios to improve coverage. |
| 10 | Test Design | Do tests follow the Arrange-Act-Assert (AAA) pattern? | Yes | None | N/A |
| 11 | Assertions | Are assertions used to verify expected outcomes? | Yes | None | N/A |
| 12 | Test Readability | Are test method names descriptive? | Yes | None | N/A |

**Summary:**

**Most Frequent Issues:**

* **Lack of Secure Password Handling:** Passwords are stored in plain text, posing a severe security risk.
* **Incomplete Input Validation:** Input validation for critical routes like /register and /login is missing.
* **Inconsistent Error Handling:** Some routes do not send consistent error responses, leading to potential confusion for clients.
* **Broad CORS Settings:** Current CORS settings allow access from localhost, which may not be restrictive enough.
* **Hardcoded Configuration:** CORS and MongoDB configurations are hardcoded, reducing maintainability.

**Suggested Refactoring:**

1. **Implement Secure Password Storage:** Use hashing algorithms like bcrypt to store passwords securely.
2. **Add Input Validation:** Use middleware like express-validator to validate user input in routes like /register and /login.
3. **Standardize Error Handling:** Use a centralized error-handling middleware to send consistent responses across routes.
4. **Enhance CORS Configuration:** Restrict CORS access to trusted origins to improve security.
5. **Abstract Configuration Options:** Move CORS and database configurations to a separate config file for better maintainability.
6. **Improve JWT Handling:** Add options like expires In for JWT generation and secure flags for cookies.