Assessment Form

JavaScript Web Applications Backend Teamwork Assignment

# Overall Scoring

Maximum points are 100. Maximum points for additional features are 25.

|  |  |  |
| --- | --- | --- |
| Requirement | Points possible | Points given |
| Must have requirements | x | x |
| Public |  |  |
| 1. **Register (POST /api/users)** | x | x |
| 1. Endpoint is implemented | 3 | 3 |
| 1. Hashing is used to store passwords | 1 | 1 |
| 1. Response returns new user data (without the password) or error message | 1 | 1 |
| 1. **Login (POST /api/session, POST /api/users/login, POST /api/login)** | x | x |
| 1. Endpoint is implemented | 3 | 3 |
| 1. Proper JWT response (does not contain password) | 1 | 1 |
| 1. Response returns JWT or error | 1 | 1 |
| 1. **Logout (DELETE /api/session, POST /api/users/logout, POST /api/logout)** | x | x |
| 1. Endpoint is implemented | 3 | 3 |
| 1. Token is invalidated | 1 | 1 |
| 1. Response returns success/error message | 1 | 1 |
| Private |  |  |
| Books |  |  |
| 1. **Retrieve all books (GET /api/books)** | x | x |
| 1. Endpoint is implemented | 3 | 3 |
| 1. Each book has a status – borrowed, unlisted, free | 1 | 1 |
| 1. Response returns all books in the library | 1 | 1 |
| 1. **Retrieve one book (GET /api/books/:id)** | x | x |
| 1. Endpoint is implemented | 3 | 3 |
| 1. Book is retrieved from the database by unique identifier (like id) | 1 | 1 |
| 1. Response returns one book or error (404) | 1 | 1 |
| 1. **Borrow a book (POST /api/books/:id)** | x | x |
| 1. Endpoint is implemented | 3 | 3 |
| 1. Book is retrieved from the database by unique identifier (like id) | 1 | 1 |
| 1. Check is performed whether the book can be borrowed | 1 | 1 |
| 1. Status of the book is updated in the database | 1 | 1 |
| 1. Response returns the book’s data or error | 1 | 1 |
| 1. **Return a book (DELETE /api/books/:id)** | x | X |
| 1. Endpoint is implemented | 2 | 2 |
| 1. Book is retrieved from the database by unique identifier (like id) | 1 | 1 |
| 1. Check is performed if the book is borrowed by the user | 1 | 1 |
| 1. Status of the book is updated in the database | 1 | 1 |
| 1. Response returns the book’s data or error | 1 | 1 |
| Reviews |  |  |
| 1. **Read book reviews (GET /api/books/:id/reviews)** | x | X |
| 1. Endpoint is implemented as sub-resource of books. Reviews is not a separate resource | 3 | 3 |
| 1. Each book to store its reviews (reviews is a property of book). Alternative approach to consider – store reviews in user. Separate entity in the database. | 1 | 1 |
| 1. Response returns the book’s reviews data or empty array, when no reviews | 1 | 1 |
| 1. **Create book review (POST /api/books/:id/reviews)** | x | X |
| 1. Endpoint is implemented | 2 | 2 |
| 1. Content field exists | 1 | 1 |
| 1. Review is added to the book or the user’s reviews (depending on where reviews are stored – per user or per book) | 2 | 2 |
| 1. Response returns the review’s data or an error message | 1 | 1 |
| 1. **Update book review (PUT /api/books/:id/reviews/:reviewId)** | x | x |
| 1. Endpoint is implemented | 2 | 2 |
| 1. User is able to edit only their own reviews | 1 | 1 |
| 1. Review is updated in the database | 1 | 1 |
| 1. Response returns the updated review’s data or error message | 1 | 1 |
| 1. **Delete book review (DELETE /api/books/:id/reviews/:reviewId)** | x | X |
| 1. Endpoint is implemented | 2 | 2 |
| 1. User is able to remove only their own reviews | 2 | 1 |
| 1. Review is deleted from the database. Alternative to deletion (for max points) – use **isDeleted** boolean property in the database. | 2 | 2 |
| 1. Response returns the deleted review’s data or an error message | 1 | 1 |
| Version control and issue tracking | x | X |
| * **Git** is used for version control | 1 | 1 |
| * [**Kanban**](https://en.wikipedia.org/wiki/Kanban_(development)) is used for project management – [GitLab issue boards](https://docs.gitlab.com/ee/user/project/issue_board.html) , [Trello](https://trello.com/) , etc. | 3 | 3 |
| Code style and formatting | x | X |
| * **ESLint** is used | 2 | 2 |
| * Code style is consistent | 2 | 2 |
| * OOP, Single Responsibility principles are followed | 3 | 3 |
| * Correct naming is used | 1 | 1 |
| * Code is clean | 1 | 1 |
| * Code is self-documenting | 1 | 1 |
| Backend requirements | x | X |
| * **Express** is used | 1 | 1 |
| * **MySQL/MariaDB** is used | 1 | 1 |
| * REST architectural principles are followed | 5 | 5 |
| * Services are used for data access and business logic | 2 | 2 |
| * At least five types of database tables created | 3 | 3 |
| * At least two types of relations in the database provided | 2 | 2 |
| * Data validation is applied. All data received from the client is validated through validation middleware | 3 | 3 |
| * Error handling is applied | 1 | 1 |
| * Unit tests for at least one of the functionalities | 1 | 0 |
| * Unit tests are present for the majority of features | 5 | 0 |
| * Unit tests are isolated | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| Additional Features |  |  |
| Should requirements | x | x |
| Private |  |  |
| **Retrieve all books (GET /api/books)** | x | x |
| 1. Query parameters used: server-side pagination, filtering, sorting | 2 | 2 |
| **Rate book (PUT /api/books/:id/rate)** | x | x |
| 1. User is able to rate a book they have borrowed and returned | 1 | 1 |
| 1. Check is performed if the user, posting the rating, has borrowed and returned the book | 1 | 1 |
| 1. Response returns the book’s data or error message | 1 | 1 |
| **Like reviews (PUT /api/reviews/:id/votes)** | X | X |
| 1. User is able to like or dislike a review | 1 | 1 |
| 1. Check if a vote exists from the authenticated user (by id or username). Vote is added (if it does not exist) or removed (if exists) | 1 | 1 |
| 1. Response returns the review’s data or an error message | 1 | 1 |
| Administration |  |  |
| Accessible only for admin users |  |  |
| **CRUD books/reviews** | x | X |
| 1. Admin user is able to create, read, update, and delete any book or review | 5 | 5 |
| **Ban users (PUT /api/admin/users/:id/banstatus)** | X | X |
| 1. Admin user is able to ban users | 1 | 1 |
| 1. Expiration date field exists on banstatus model and is used for restrictions | 1 | 1 |
| 1. Banned user is restricted from every operation in the library, except reading | 1 | 1 |
| 1. Response returns the user data or an error message | 1 | 1 |
| **Delete users (DELETE /api/admin/users/:id)** | x | X |
| 1. Admin user is able to delete users from the library system | 2 | 2 |
| 1. Response returns the user data or an error message | ? | ? |
| Could requirements | x | X |
| **Reading points / Gamification** | x | X |
| 1. Users get reading points for: every book they borrowed and returned, writing a review, having a review liked | 2 | 2 |
| 1. Level system – users gain levels, depending on the reading points they have | 2 | 0 |
| 1. Users lose some percentage of points if they are banned, depending on how long they are banned for | 2 | 2 |

|  |  |  |
| --- | --- | --- |
| **Summary** | Possible | Given |
| Must have requirements | 100 | 92 |
| Additional features | 25 | 23 |

\* To update the calculated fields: select all (Ctrl + A), then press the F9 button (works only on Desktop)