INSTITUTO
UNIVERSITÁRIO
DE LISBOA

API Development Project Report: Event & User Management API

Unidade Curricular: Armazenamento de Dados em Ambientes Descentralizados (ADAD)
Academic Year 2025/2026
1º Semestre
Coordenador: Prof. João Matos

Syed Hammad Ur Rehman Asghar - 123757
Robin FASSELER - 136767
Muhammad Tayyab Iqbal - 122082
Filipe Fernandes - 104956

09/11/2025

1. Project Overview and Goals

1.1 Project Context

The goal of this project was to develop a fully functional **REST API** using **Node.js** and **ExpressJS** to manage two core entities: **Events** and **Users**. This API supports standard CRUD operations and advanced data retrieval required for a supporting front-end application.

1.2 Technology Stack

- **Framework:** Node.js + ExpressJS
- **Database:** MongoDB
- **API Testing:** Postman
- **Programming Language:** JavaScript

1.3 Fulfillment of Component Requirements

This report serves to document how the project addressed the quality components, including **code organization**, **correct route definition**, **MongoDB Query quality**, **error handling**, and **API Documentation** (attached separately).

2. API Design and Data Modeling

2.1 Core Entities and Relationships

We defined two primary MongoDB collections and associated models:

1. **Users:** Stores user profile data.
2. **Events:** Stores event details and includes a mechanism for embedding or referencing **reviews/ratings**.

2.2 Project Structure and Organization

The code was organized into a modular structure to ensure clarity and maintainability:

- `index.js`: Server initialization and database connection.
- `routes/`: Express router files (`userRoutes.js`, `eventRoutes.js`).
- `models/`: MongoDB Schema definitions (`User.js`, `Event.js`).
- `controllers/`: Core business logic and database interaction (including all MongoDB Queries).
- `utils/`: Shared utility functions (e.g., error handling middleware).

3. Group Contribution and Endpoint Ownership

The project was divided logically into two main modules—User Management and Event Management—with additional responsibilities assigned for core functions like database connection and error handling.

Team Member	Core Endpoints Owned (Key Tasks)
Syed Hammad	1. GET /events 4. POST /users 7. DELETE /events/:id 10. PUT /users/:id 13. GET /events/star
Robin Fasseler	3. POST /events 6. GET /users/:id 9. PUT /events/:id 12. GET /events/ratings/:order 15. POST /users/:id/review/:event_id
M. Tayyab Iqbal	2. POST /events 5. GET /events/:id 8. DELETE /users/:id 11. GET / events/top:limit 14. GET / events/:year

Filipe Fernandes

```
Extra Endpoints:

16. GET /events/county/:county

17. GET /events/trending

18. GET /users/top

19. GET /users/active/:year
```

4. Implementation Details and Quality

4.1 Route Definition and Logic (`routes/` and `controllers/`)

All routes were defined using the Express Router to separate concerns. In the controllers, we adhered to the principle of "fat models, thin controllers" where business logic was kept as close to the data model as possible.

4.2 MongoDB Queries and Performance

For endpoints requiring complex data manipulation, we utilized specific MongoDB query types:

- **Pagination (Routes 1, 2):** Implemented using the `.skip()` and `.limit()` operators.
- **Average Score (Route 5, 11):** Used the **Aggregation Pipeline** with the `$group` and `$avg` stages, which is efficient for calculating dynamic metrics.

4.3 Error Handling (Crucial Requirement)

We implemented a centralized error handling middleware to ensure consistent responses, fulfilling the requirement for **correct error treatment with specification of error code**.

- **400 Bad Request:** Returned for invalid input or validation failures (e.g., missing fields in a POST body).
- **404 Not Found:** Returned when querying an entity by ID that does not exist in the database.
- **500 Internal Server Error:** Returned for unexpected server or database connection issues.
- **Implementation:** Error objects containing the HTTP status code and a descriptive message are thrown in the controllers and caught by the global error handler.

5. Conclusion

The team successfully developed and documented all mandatory and custom endpoints for the Event and User Management API, meeting the project's technical specifications and quality requirements. The clear division of work by endpoint and a focus on correct error handling ensured a stable and well-organized codebase.