# Review Service API Design

## Assumptions

* The API assumes authentication and authorization are handled at a different layer (e.g., API gateway or middleware).
* All requests reaching the API are pre-validated for authentication and authorization.
* Users can submit only one review per product, identified by their `userId`, but can update their existing review.
* Reviews will be moderated externally, and no content filtering is applied at the API level.
* Reviews belong to products, so they are nested under `/products/{id}/reviews` for creation and listing.
* Individual review actions (fetch, update, delete) use `/reviews/{id}` as reviews have unique IDs.
* The database structure follows a NoSQL schema to store reviews in a flexible format.

## Database Choice: MongoDB

### Reasoning:

* Flexible Schema: Reviews can have optional fields such as ratings, text comments, and metadata.
* Scalability: MongoDB is optimized for large-scale applications with high read/write operations.
* Document-based Storage: Each review can be stored as a document, making it easy to retrieve by product ID or user ID.

## Database Schema

// Reviews Collection  
  
{  
 \_id: ObjectId, // MongoDB's default unique ID  
 productId: String, // ID of the product being reviewed  
 userId: String, // unique user id  
 rating: Number, // Optional: Rating (e.g., 1-5)  
 comment: String, // Optional: Text comment  
 reviewerName: String,  
 reviewerEmail: String,  
 metadata: Object, // Optional: Additional metadata  
 createdAt: Date, // Timestamp of review creation  
 updatedAt: Date // Optional: Timestamp of review update  
}

## API Endpoints

### 1. Create a Review

Method: POST

Endpoint: /products/{id}/reviews

#### Request Payload:

{  
 "userId": "user123",  
 "rating": 5,  
 "comment": "Great product!",  
 "reviewerName": "John Doe",  
 "reviewerEmail": "john.doe@example.com",  
 "metadata": {"key": "value"}  
}

#### Response Payload:

{  
 "id": "unique\_review\_id",  
 "productId": "{id}",  
 "userId": "user123",  
 "rating": 5,  
 "comment": "Great product!",  
 "reviewerName": "John Doe",  
 "reviewerEmail": "john.doe@example.com",  
 "metadata": {"key": "value"},  
 "createdAt": "2024-10-27T10:00:00Z"  
}

#### Error Responses:

* 400 Bad Request:

{  
 "error": "Bad Request",  
 "message": "The request payload is invalid. Must contain at least one of rating or comment."  
}

* 409 Conflict:

{  
 "error": "Conflict",  
 "message": "A review already exists for this product by the same user."  
}

### 2. Get Reviews for a Product

Method: GET

Endpoint: /products/{id}/reviews

#### Query Parameters:

* - page (optional, default: 1) → Pagination support
* - pageSize (optional, default: 10) → Limits the number of reviews returned

#### Example Request:

GET /products/{id}/reviews?page=2&pageSize=5

#### Response Payload:

{  
 "reviews": [  
 {  
 "id": "unique\_review\_id\_6",  
 "productId": "{id}",  
 "userId": "user123",  
 "rating": 5,  
 "comment": "Great quality!",  
 "reviewerName": "Alice Brown",  
 "createdAt": "2024-10-25T14:00:00Z",  
 "metadata": {}  
 },  
 {  
 "id": "unique\_review\_id\_7",  
 "productId": "{id}",  
 "userId": "user456",  
 "rating": 3,  
 "comment": "Average product.",  
 "reviewerName": "Bob White",  
 "createdAt": "2024-10-24T16:30:00Z",  
 "metadata": {}  
 },  
 {  
 "id": "unique\_review\_id\_8",  
 "productId": "{id}",  
 "userId": "user789",  
 "rating": 4,  
 "comment": "Would buy again.",  
 "reviewerName": "Charlie Green",  
 "createdAt": "2024-10-23T18:15:00Z",  
 "metadata": {}  
 },  
 {  
 "id": "unique\_review\_id\_9",  
 "productId": "{id}",  
 "userId": "user101",  
 "rating": 2,  
 "comment": "Not as expected.",  
 "reviewerName": "David Black",  
 "createdAt": "2024-10-22T19:45:00Z",  
 "metadata": {}  
 },  
 {  
 "id": "unique\_review\_id\_10",  
 "productId": "{id}",  
 "userId": "user112",  
 "rating": 5,  
 "comment": "Loved it!",  
 "reviewerName": "Emily White",  
 "createdAt": "2024-10-21T21:30:00Z",  
 "metadata": {}  
 }  
 ],  
 "total": 15,  
 "currentPage": 2,  
 "totalPages": 3  
}

#### Error Responses:

* 404 Not Found:

{  
 "error": "Product not found",  
 "message": "The product with ID '123' does not exist."  
}

### 3. Get a Single Review by ID

Method: GET

Endpoint: /reviews/{id}

#### Example Request:

GET /reviews/{id}

#### Response Payload:

{  
 "id": "unique\_review\_id",  
 "productId": "{id}",  
 "userId": "user123",  
 "rating": 5,  
 "comment": "Great product!",  
 "reviewerName": "John Doe",  
 "reviewerEmail": "john.doe@example.com",  
 "createdAt": "2024-10-27T10:00:00Z",  
 "metadata": {}  
}

#### Error Responses:

* 404 Not Found:

{  
 "error": "Review not found",  
 "message": "The review with ID '123' does not exist."  
}

### 4. Update a Review

Method: PUT

Endpoint: /reviews/{id}

#### Request Payload:

{  
 "rating": 4,  
 "comment": "Updated comment.",  
 "metadata": {"updatedKey":"updatedValue"}  
}

#### Response Payload:

{  
 "id": "unique\_review\_id",  
 "productId": "{id}",  
 "userId": "user123",  
 "rating": 4,  
 "comment": "Updated comment.",  
 "reviewerName": "John Doe",  
 "reviewerEmail": "john.doe@example.com",  
 "createdAt": "2024-10-27T10:00:00Z",  
 "updatedAt": "2024-10-27T11:00:00Z",  
 "metadata": {"updatedKey":"updatedValue"}  
}

#### Error Responses:

* 404 Not Found:

{  
 "error": "Review not found",  
 "message": "The review with ID '123' does not exist."  
}

* 400 Bad Request:

{  
 "error": "Bad Request",  
 "message": "The request payload is invalid. Must contain at least one of rating or comment."  
}

### 5. Delete a Review

Method: DELETE

Endpoint: /reviews/{id}

#### Example Request:

DELETE /reviews/{id}

#### Response Payload:

{  
 "message": "Review deleted successfully",  
 "id": "123",  
 "deletedAt": "2024-10-27T12:00:00Z"  
}