



ADR - Creation of HTTP-based API with or without HATEOAS for Antragsraum

Date	2023-10-12	
Attendees	Theodossios Aswestopoulos Maurice Laboureur Miguel Ferreira Rolf Roeßing Jens Reese Jörg Amelunxen	
Status	<div> Declined </div>	

Motivation:

We are designing our HTTP-based API for the "Antragsraum" to the client side of it. One of the critical decisions we need to make is whether to implement this API using the HATEOAS principle or not.

The decision has significant implications for our system architecture, particularly in terms of where the business logic is located (client vs server). If we use HATEOAS, much of the business logic will be moved to the backend, which may simplify the frontend but potentially increase the backend complexity. If we don't use HATEOAS, the frontend will carry more of the business logic, which of course will increase its complexity but simplify the backend.

Additional decision points to consider:

- Interoperability
- Scalability
- Maintainability
- Usability

Example

HATEOAS response

```
{
  "id": "123",
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "price": "$10.99",
  "_links": {
    "self": {
      "href": "http://api.example.com/books/123"
    },
    "reviews": {
      "href": "http://api.example.com/books/123/reviews"
    },
    "author": {
      "href": "http://api.example.com/authors/F.%20Scott%20Fitzgerald"
    },
    "purchase": {
      "href": "http://api.example.com/books/123/purchase"
    }
  }
}
```

In this response, the `_links` field contains links that describe possible actions related to the book.

- The `self` link gives the URL for accessing the book's own data.

- The `reviews` link leads to the book's reviews.
- The `author` link leads to information about the book's author.
- The `purchase` link is where a client can go to purchase the book.

The client can understand and navigate the API based on these links, without needing to know the URL structure in advance. This is the key feature of HATEOAS - it makes the API self-descriptive and easier to use and evolve over time.

Points For HATEOAS

- Decoupling: HATEOAS allows for a higher level of decoupling between the server and the client (as the delivered links contain a lot of information about potential "follow up use cases"). The server provides the necessary state transitions, making the client less dependent on specific URL structures.
- Future-proofing: With HATEOAS, changes to the URL structure or business logic can be made on the server without breaking the client, as long as the link relations do not change. This can make the API more resilient to changes and easier to evolve over time.
- Self-Descriptiveness: HATEOAS can make the API somewhat self-descriptive, which can improve its usability for developers. Clients can understand the API without needing to refer to external documentation (or at least less).

Points Against HATEOAS

- Increased Complexity: Implementing HATEOAS can add complexity to the backend, as it requires the server to provide meaningful links and relations in the responses.
- Limited Client Support: Not all HTTP clients and libraries support HATEOAS fully. (redux router has support)
- Overhead: HATEOAS adds extra overhead to each API response due to the inclusion of control information (links and relations). This could potentially impact the performance of the API, especially in high-load scenarios. (overhead should be limited)

Decision

- Quite some effort in the backend
- For us right now limited benefit
- Swagger already offers documentation
- Decision Against HATEOAS