REST API

REST (Representational State Transfer) er en arkitekturstil for distribuerede systemer, der bygger på standard HTTP-metoder til at kommunikere mellem klient og server. Et REST API (Application Programming Interface) følger disse principper og tillader CRUD-operationer (Create, Read, Update, Delete) via HTTP-protokollen.

REST API'er anvender ressourcer, der identificeres via unikke URIs (Uniform Resource Identifiers). Hver ressource kan manipuleres ved hjælp af følgende HTTP-metoder:

- **GET**: Henter data fra en ressource.
- **POST**: Opretter en ny ressource.
- **PUT**: Opdaterer en eksisterende ressource.
- **DELETE**: Sletter en ressource.

REST API Principper

For at et API kan betragtes som RESTful, skal det overholde følgende principper:

1. **Klient-server arkitektur**: Klienten sender forespørgsler til serveren, der behandler og returnerer svar. De to dele er adskilt for at sikre skalerbarhed og fleksibilitet.
2. **Stateless**: Serveren gemmer ikke nogen klienttilstand mellem anmodninger. Hver forespørgsel skal indeholde al nødvendig information.
3. **Cachebarhed**: Svar fra serveren kan caches for at forbedre ydeevnen.
4. **Ensartet grænseflade**: API'et skal have konsistente og forudsigelige strukturer for ressourcer og deres repræsentationer.
5. **Lagopdelt system**: Et REST API kan implementeres i flere lag (f.eks. load balancers, proxier) uden at klienten behøver at kende til den underliggende infrastruktur.
6. **Kode-on-demand (valgfrit)**: Serveren kan returnere eksekverbar kode (f.eks. JavaScript) for at udvide klientfunktionalitet.

Struktur og Eksempel

Et typisk REST API strukturerer sine ressourcer på en måde, der afspejler en logisk datamodel. Et eksempel på en RESTful ressource kunne være:

```
GET https://api.example.com/users/123
```

Dette kunne returnere følgende JSON-svar:

```
{
```

```
    "id": 123,
    "name": "John Doe",
    "email": "john.doe@example.com"
}
```

For at oprette en ny bruger kan en **POST**-anmodning sendes:

```
POST https://api.example.com/users
```

Med følgende JSON-body:

```
{
    "name": "Jane Doe",
    "email": "jane.doe@example.com"
}
```

Autentifikation og Sikkerhed

REST API'er kræver ofte autentifikation for at begrænse adgang. De mest almindelige metoder inkluderer:

- **API-nøgler**: En unik nøgle sendes med hver anmodning.
- **OAuth 2.0**: En mere sikker metode, der bruger adgangstokens.
- **JWT (JSON Web Token)**: En token-baseret metode, der sikrer stateless autentifikation.

For at beskytte et API mod angreb anbefales det at implementere:

- HTTPS for krypteret kommunikation.
- Rate limiting for at forhindre DoS-angreb.
- Input validering for at undgå SQL Injection og XSS.

Versionering

For at sikre bagudkompatibilitet kan et REST API versioneres. Dette gøres typisk ved:

- **URL-baseret versionering**: https://api.example.com/v1/users
- **Header-baseret versionering**: Tilføjelse af en Accept header, f.eks. Accept: application/vnd.example.v1+json

REST vs. SOAP

REST adskiller sig fra SOAP (Simple Object Access Protocol) ved at være lettere og baseret på standard HTTP-metoder. SOAP bruger XML og er mere kompleks, hvilket gør REST til en foretrukken løsning i moderne webudvikling.

Kilder og Yderligere Information

- REST API Design Guide
- https://blog.postman.com/rest-api-examples/