API:er

Vad är ett API?

- Application Programming Interface
- Ett API tillhandahåller funktionalitet utan att exponera detaljer
- Ett API exekverar kod baserat på anrop utifrån.

Varför används API:er?

- API:er kan användas för att…
  - … abstrahera delar av en applikation
  - … ge åtkomst till funktionalitet åt flera klienter

- Ett API är alltså en fristående del av en eller flera applikationer

Olika typer av API:er

- SOAP – Simple Object Access Protocol
  - Framtaget av W3C (World Wide Web Concortium)
  - Introducerades 1998
  - Bygger på XML som dataformat
  - Hårda och strikta regler
- REST – Representational State Transfer
  - Framtaget av Roy Fielding
  - Introducerades 2000
  - Mjuka riktlinjer (Mer om detta senare)

Andra ramverk och verktyg värda att nämna.

- gRPC – google Remote Procedure Call
  - Ett Framework för att bygga APIer
  - Använder JSON
- tRPC – typeScript Remote Procedure Call
  - Används för att typsäkra data mellan client och server i TypeScript
- GraphQL – Graph Query Language
  - Mer av ett verktyg för att bygga queries till en server

SOAP

- Kan kommunicera via flertalet protokoll.
  - TCP, UDP, JMS, SMTP, HTTP
- Går att skriva i de flesta språk.
- Använder enbart XML
- Högre säkerhet
- Långsamt
- Funktionsdrivet

REST

- Använder HTTP för kommunikation
- Snabbt
- Går att skriva i alla språk
- Använder valfritt sätt att representera data.
- Datadrivet
- Lösa riktlinjer
- Cachning

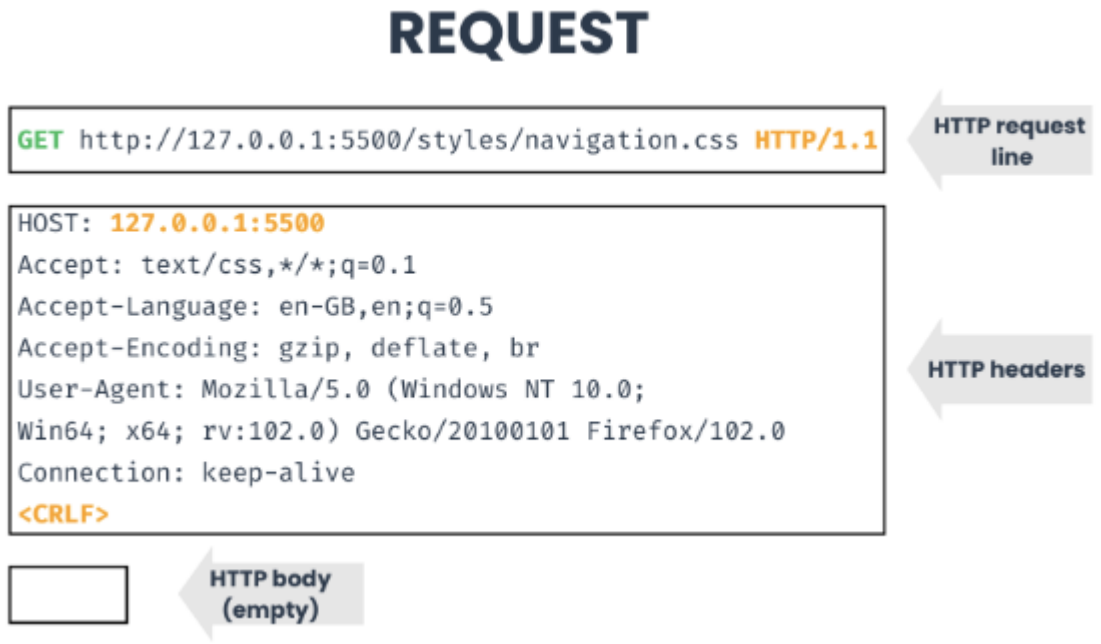
Hur kommunicerar vi via HTTP?

- HTTP (Hyper Text Transfer Protocol) bygger på:
  - Förfrågan (**Request**)
  - Svar (**Response**)

- En klient kommunicerar med ett API genom att skicka en förfrågan och sedan få ett svar.

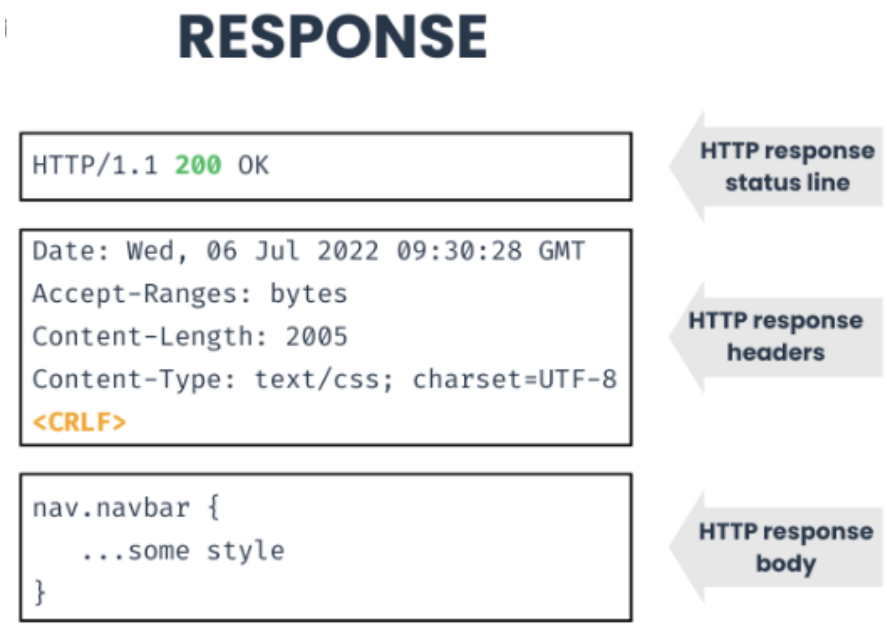
Uppbyggnad av en Förfrågan (**Request**)

- Header
  - Request line – HTTP method, URI, HTTP version
  - Optional Request Headers (En samling key:value-pairs)
- Blank line (Separerar Header med Body)
- Body (Optional)
  - Data som man vill skicka med till servern. T.ex. Sessionsinformation, kundvagn, login-information.



Uppbyggnad av ett Svar – **Response**

- Header
  - HTTP Version
  - Status Code (tresiffrig kod som beskriver resultatet)
  - Reason Phrase (Status text, text som beskriver statuskoden)
  - Optional HTTP Headers (key:value - pairs)
- Blank Line
- Body (Optional)
  - Data eller innehåll



HTTP Methods

- **GET** – Förfrågan om att få data
- **POST** – Förfrågan om att spara data
- **PUT** – Förfrågan om att uppdatera data
- **DELETE** – Förfrågan om att ta bort data
- **PATCH** – Förfrågan om att uppdatera viss del av data

- Mfl…

HTTP Methods – CRUD Operations

HTTP Method	CRUD
POST	Create
GET	Read
PUT	Update/Replace
PATCH	Partial update/Modify
DELETE	Delete

URL – Uniform Resource Locator

- En URL är uppbyggd på följande sätt
  - **Schema** (exempel: http, https, ftp osv.)
  - **Host** (exempel: localhost, www.iths.se)
  - **Path** (exempel: /courses, /watch, /about)
  - **Query string** (exempel: ?v=6sUbt-Qp6Pg)
- Exempel URL: https://www.youtube.com/watch?v=6sUbt-Qp6Pg

Request Body

- Information till servern vid specifika requests
- **GET** och **DELETE** requests brukar inte ha någon body
- **POST**, **PUT**, **PATCH** bör ha en body för att ge API:et information om vad som ska sparas/ändras
- En body kan innehålla data av olika format.
  - HTML
  - JSON
  - XML
- Maxlängd på en body är inte specificerad av protokollet.
  - Däremot har webbläsare och servrar begränsningar.

Statuskoder

- 100-199: Informativa koder
- 200-299: Successful
- 300-399: Redirection
- 400-499: Client Error
- 500-599: Server Error

Lista över definerade statuskoder:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Första steget till att bygga ett API?

- API – Specification
  - Tydlig beskrivning om syfte och mål för API:et
  - API:ets URL
  - Vilka HTTP-metoder som stöds
  - Beskrivning om hur förfrågningar hanteras
  - Beskrivning av hur svar är uppbyggda
  - Hur autentisering utförs
  - Vilka olika statuskoder som används
  - Annan viktig information

REST API

- **REST** står för
  - **RE**presentational
  - **S**tate
  - **T**ransfer
- Begränsningar för ett REST API
  - Unika sökvägar för alla resurser
  - Förändring genom representation
  - Självbeskrivande meddelanden
  - Driva all kommunikation med (Hypermedia as the Engine of Application State)
    - Kort sammanfattat att klienten ska inte behöva känna till serverns status.

Arkitektoniska Begränsingar för REST

1. Enhetligt interface (Uniform Interface)
   - Definiera sökvägar och ändra inte!
2. Client – Server
   - Server och klient ska kunna utvecklas separat, så länge interfacet emellan dem inte ändras
3. Lägeslös (Stateless)
   - Klienten ska tillhandahålla all information om dess läge, servern ska inte spara anropshistorik från någon enskild klient.
4. Cachebart (Cacheable)
   - Resurser ska (när applicerbart) deklarera sig som cachebara.

5. System baserat på lager (Layered System)
   - REST låter oss dela upp delar av applikationen på olika servrar

6. Kod på begäran (Code on demand) *valbart
   - Oftast returnerar man statiskt innehåll (XML/JSON). Men man får också skicka exekverbar kod.

Alla tidigare nämnda begränsningar hjälper oss bygga ett äkta RESTful API, men ibland måste man bryta mot någon av dessa. Det är okej, det är fortfarande ett RESTful API (men inte ett äkta sådant!)

Källor

- https://restfulapi.net/what-is-an-api/
- https://www.ibm.com/docs/en/cics-ts/5.3?topic=concepts-http-protocol
- https://developer.mozilla.org/en-US/docs/Web/HTTP
- https://www.w3.org/TR/soap/
- https://trpc.io/docs
- https://grpc.io/docs/
- https://graphql.org/