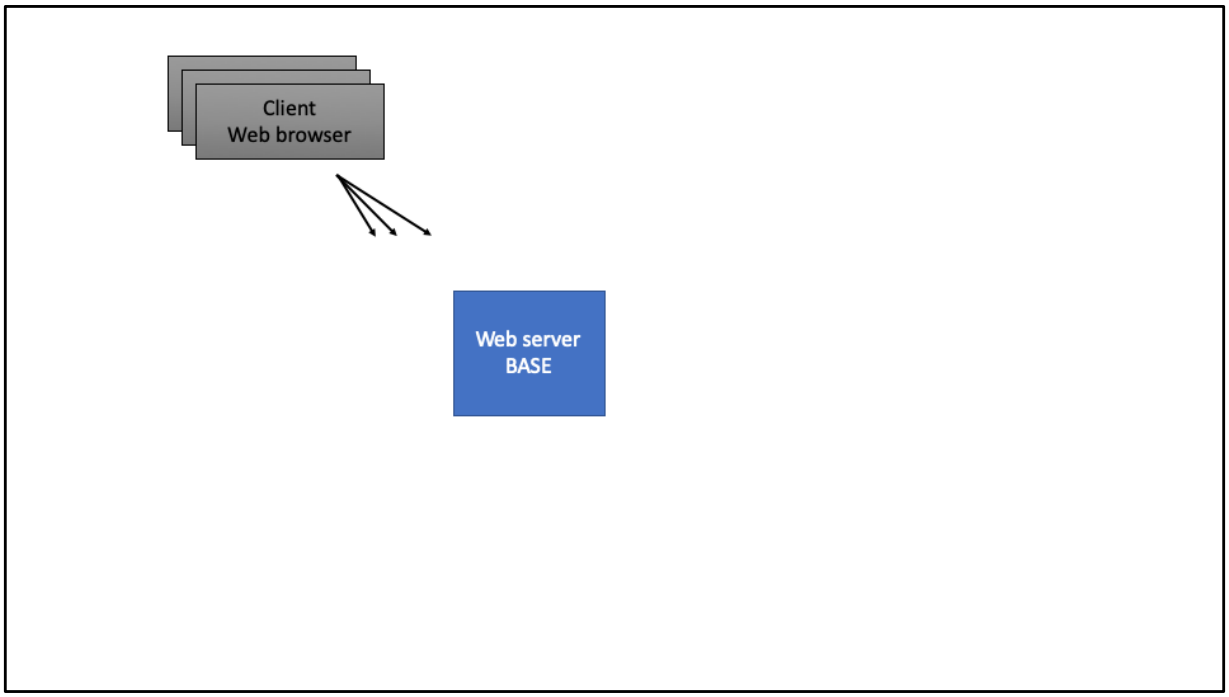
Grundsystemet
ETSN05

REST, SQL

Rasmus.ros@cs.lth.se

Maila mig gärna



Flera klienter, en server
Klienterna anropar servern
Grundsystemet heter BASE
Denna föreläsningen handlar om den blå rutan

REST APIs

- Server design principles and architecture
- Client/server separation

- http://apikatalogen.se/
- https://public-apis.xyz/

REST är inte teknik, en arkitekturstil
Gränssnitt använder REST APIer som data, tex.
Gränssnitt inte en del av REST APIet

- kan bygga ovanpå
- kan använda många APIer
- flera kan dela REST API

Vanlig HTTP utan modifieringar

HTTP

- Network protocol
- Hyper Text Transfer Protocol

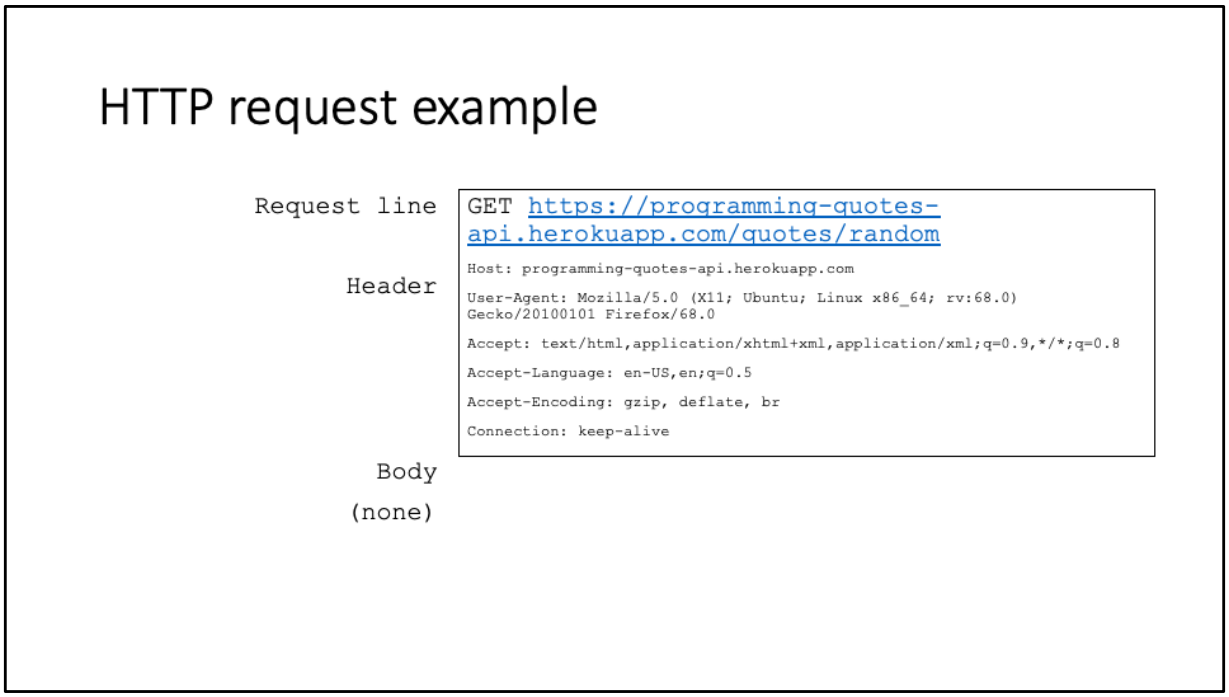
- Client request and server response
  - Verb and URL (and body?)
  - Status code (and body?)

Vad är HTTP då?
Protokoll för nätverkstrafik
Internet är byggt på HTTP
HTTP är på byggt på TCP/IP osv.

En dator som lyssnar för inkommande trafik
Servern identifieras av URL, kan vara ett namn
Under laborationer och projektet kan vilken
Webbläsaren är klienten
Klienten initierar alltid anrop, servern kommer

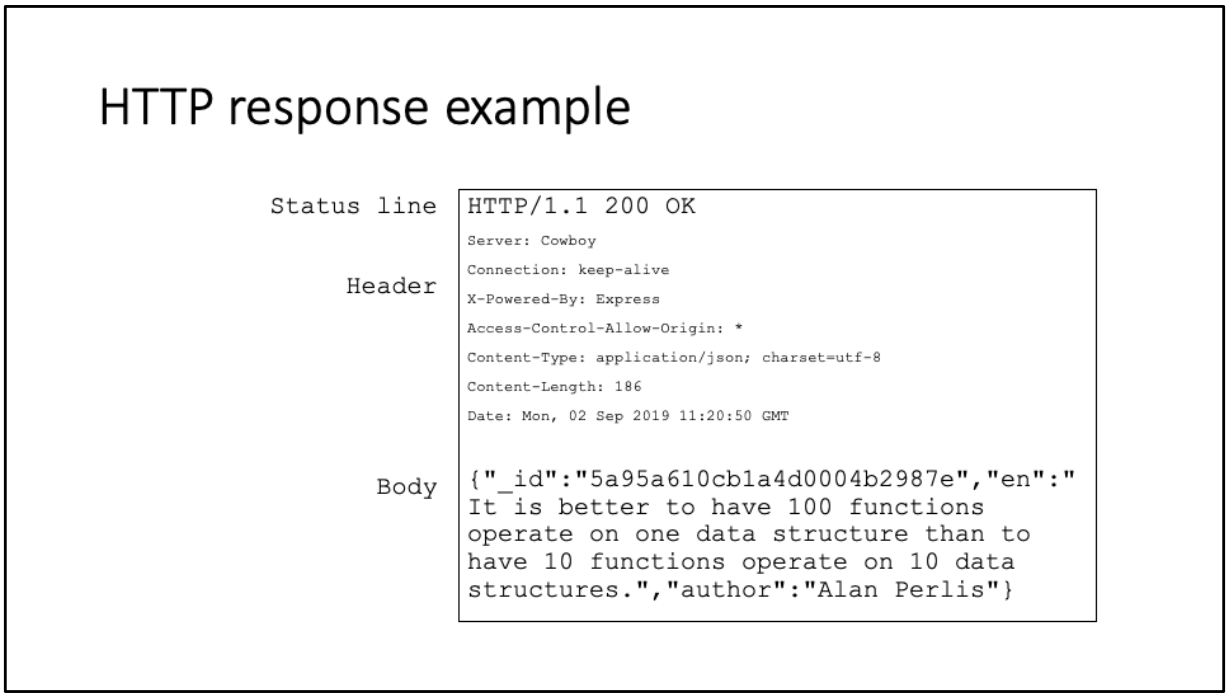
Anrop uppbyggt av verb och URL. Verb oftast
Status kod oftast 200 OK syns inte, har nog
Kommer se mycket 500 server error

HTTP request example



Gå med webbläsaren till https://programming-quotes-api.herokuapp.com/quotes/random
Det som står i rutan skickas då till servern
Första raden är det viktiga
Ibland skickar man även med en body, inte i detta fallet
Tex när man fyller i ett formulär

HTTP response example



Svaret blir 200 OK och en body

Formatterar det snyggare här näst

HTTP request/response example

```
GET https://programming-quotes-api.herokuapp.com/quotes/random
--------------------------------------------------------------
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
    "_id": "5a95a610cb1a4d0004b2987e",
    "en": "It is better to have 100 functions operate on one data
           structure than to have 10 functions operate on 10 data
           structures.",
    "author": "Alan Perlis"
}
```

Request där uppe, response under

REST APIet handlar om att få citat om programmering
Det specifika anropet är ett slumpmässigt

Datan som finns I bodyn är I formatet JSON
Generellt format. Smidigt att arbeta med från

HTTP stödjer många olika data format.
HTTP är kuveret och datan är medellanet
Header parametern content-type berättar
I projektet är det tänkt att alltid vara JSON till

HTTP request/response example 2

```
DELETE https://programming-quotes-api.herokuapp.com/quotes/random
-----------------------------------------------------------------
HTTP/1.1 403 FORBIDDEN
Content-Type: application/json; charset=utf-8

{
    "success": false,
    "message": "No token."
}
```

Finns många andra anrop att göra på REST
Servern känner bara till vissa anrop
Tex. delete på samma URL
Får 403 Forbidden, man behöver skicka med en "token"=

REST design

- Represent hierarchy of resources in URLs
- Use verbs
  - GET, DELETE
  - With body: PUT, POST
- Use status codes
  - 200 OK, 204 No content
  - 404 Not found, 403 Forbidden, 500 Server error

En resurs är ett koncept inom REST, ingen
En resurs är ett substantiv man vill utföra
URLen ska peka ut vilken resurs man vill göra

- GET = hämta information utan att förändra
- DELETE = ta bort det som står under URLen
- PUT = byt ut det som finns under URLen
- POST = lägg till nytt, uppdatera, osv.

Status koder, 2XX betyder bra, 4XX klienten

Quotes API documentation

- Get all quotes
  - GET /quotes
- Get quotes by language
  - GET /quotes/lang/en
- Get quotes by page
  - GET /quotes/page/2
- Get random quote
  - GET /quotes/random
- Get random quote by language
  - GET /quotes/random/lang/sr
- Get quote by id
  - GET /quotes/id/5a6ce86f2af929789500e824
- Post vote
  - POST /quotes/vote
  - required params: quoteId, newVote (number from 1 to 5)
- Post quote (user)
  - POST /quotes
  - required params: token, author, en
  - optional: source, sr
  - author name should be from Wikipedia
- Update quote (admin)
  - PUT /quotes
  - required params: token, _id, author, en
  - optional: source, sr
- Delete quote (admin)
  - DELETE: /quotes
  - required params: token, _id

Hela programming quotes APIet
Till vänster behövs inte någon API nyckel,
Beskriver vilka fält JSON datan ska ha till höger
Lite sparsmakat dokumenterat

Quotes API documentation

- Get all quotes
  - GET /quotes
- Get quotes by language
  - GET /quotes/lang/en
- Get quotes by page
  - GET /quotes/page/2
- Get random quote
  - GET /quotes/random
- Get random quote by language
  - GET /quotes/random/lang/sr
- Get quote by id
  - GET /quotes/id/5a6ce86f2af929789500e824

- Post vote
  - POST /quotes/vote
  - required params: quoteId, newVote (number
- Update quote (admin)
  - PUT /quotes
  - required params: token, _id, author, en
  - optional: source, sr
- Delete quote (admin)
  - DELETE: /quotes
  - required params: token, _id

- What is the current resource?
- What other resources could there be?

Resurser = quotes
Extra resurser = authors, languages, editors?

Resources in Quotes API

1. /authors
    1. /authors/id/4
    2. /authors/id/4/quotes
    3. /authors/id/4/quotes/id/30 ?
2. /quotes
    1. (/quotes/id/30)
    2. (…)
    3. /quotes/id/30/authors ?
3. /languages

Hämta ut alla authors
Hämta ut author med id 4
Hämta ut alla quotes från author med id 4

/quotes finns redan
1.3 och 2.1 är redundanta, men kan samspela
2.3 är inte så vettig, en quote har alltid en author

Languages är definierat I dom flesta programspråk
Kan strukturera om APIet och ha languages

REST in Java

Jersey – REST in Java

```
@Path("foo")
public class FooResource {
    @GET
    @Produces("application/json")
    public List<Foo> getFoos() {
        // The method returns normal Java objects
        return ...;
    }
}
```

GET /foo

Jersey är ett bibliotek för REST
Deklarativt
Jersey är inte en server
Servern heter Jetty, det finns många andra
Jersey bor i en server så man slipper skriva
Annars får man skriva saker som: if request method equals GET and

Det finns sätt att göra REST enkelt i många

En Java klass motsvarar en REST resurs
En Java metod är ett verb och en URL

Snabel-a är annoteringar, Path, GET, DELETE, Produces, Consumes, ...
Normalt är annotering information till kompilatorn

Jersey – REST in Java

```
@Path("foo")
public class FooResource {
    @GET
    @Produces("application/json")
    public List<Foo> getFoos() {
        // The method returns normal Java objects
        return ...;
    }
    @PUT
    @Consumes("application/json")
    @Path("id/{fooId}")
    public void saveFoo(@PathParam("fooId") int id, Foo foo) {
        // Save to database
    }
}
```

GET /foo
PUT /foo/id/41

Man kan ha flera annoterade REST anrop I
Det nedre exemplet visar

- Man kan kombinera path från klass och
- Parametrar tas från URLen
- HTTP body kommer in

GSON – Java Objects to JSON

```
public class Foo {

    private final int id;
    private final int userId;
    private final String payload;
    private final long created;

    public Foo(int id, int userId, String payload, long created) {
        this.id = id;
        this.userId = userId;
        this.payload = payload;
        this.created = created;
    }

    public int getId() { return id; }

    public int getUserId() { return userId; }

    public String getPayload() { return payload; }

    public long getCreated() { return created; }
}
```

<=>

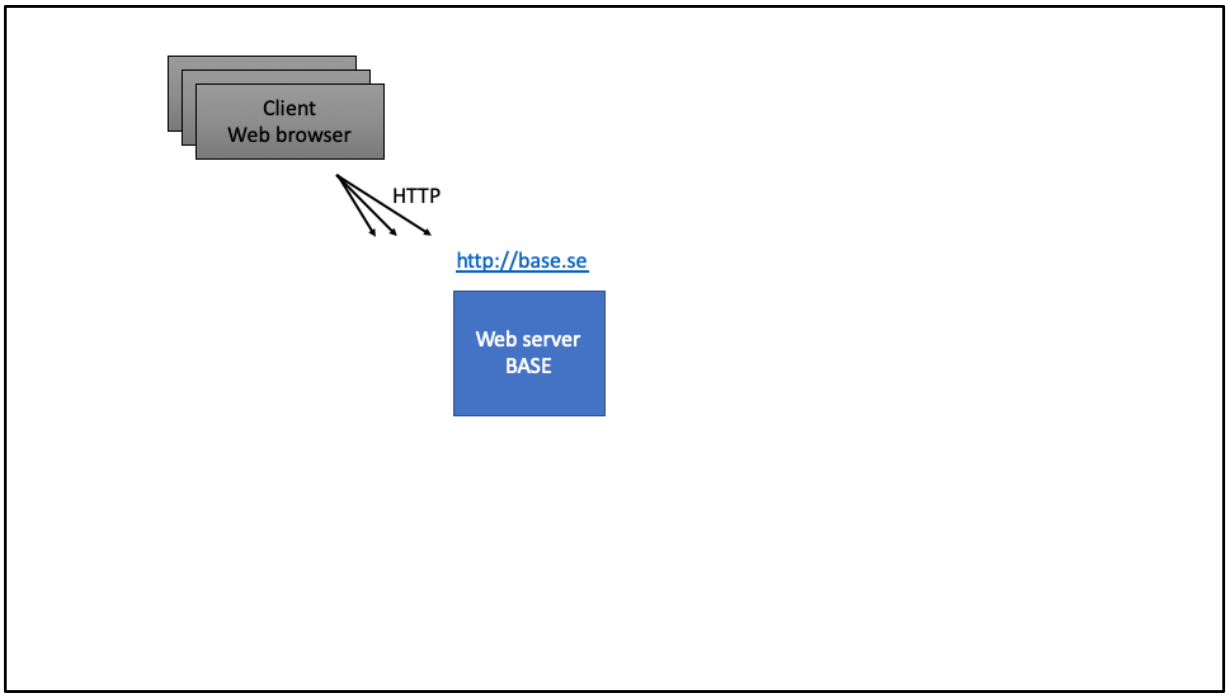
```
{
    "id": 21,
    "userId": 4,
    "payload": "a",
    "created": 129157912
}
```

- Maps and objects {"a": 1, "b":2}
- Lists and sets [{}, {}, {}]
- Nested objects, ints, longs, strings, …

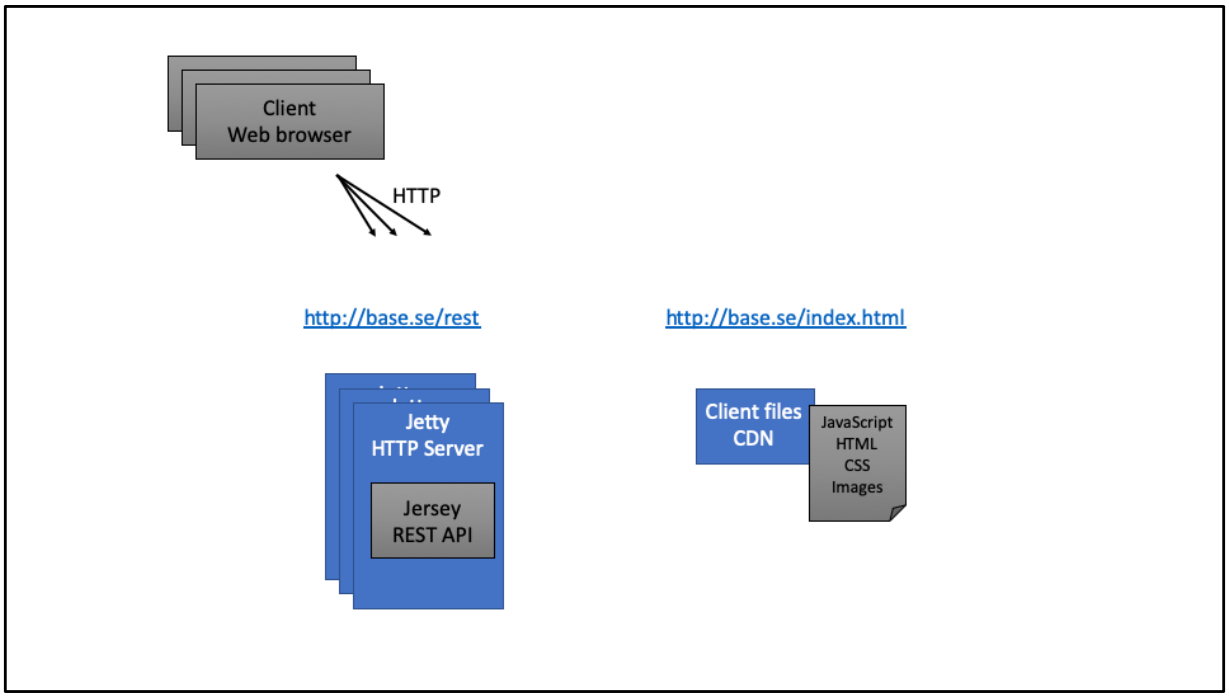
GSON översätter från sträng av JSON till java objekt, och vice versa
Konfigurerad till att användas bara när Consumes/Produces är application/json
När PUT och POST requests kommer till servern gör GSON Java objekt
När saker returneras från REST APIet som body i HTTP gör GSON sträng av ett Java objekt

BASE

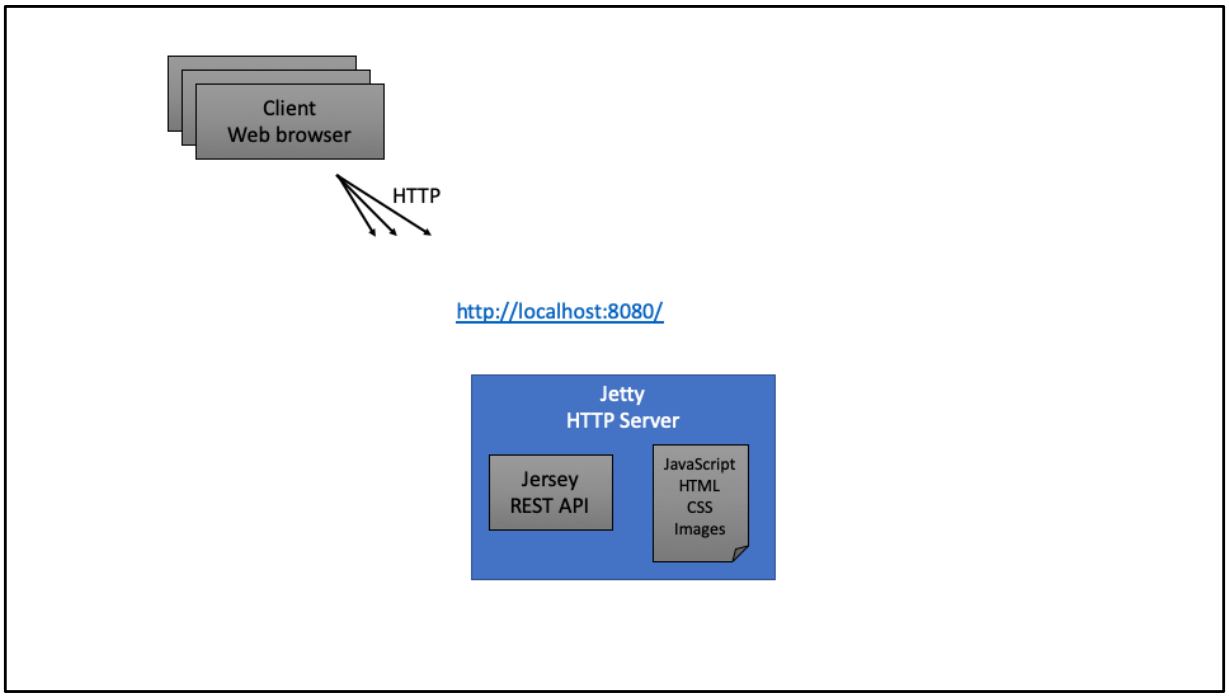
Overview



Nu vet vi att klienterna pratar HTTP med servern
Servern har en URL

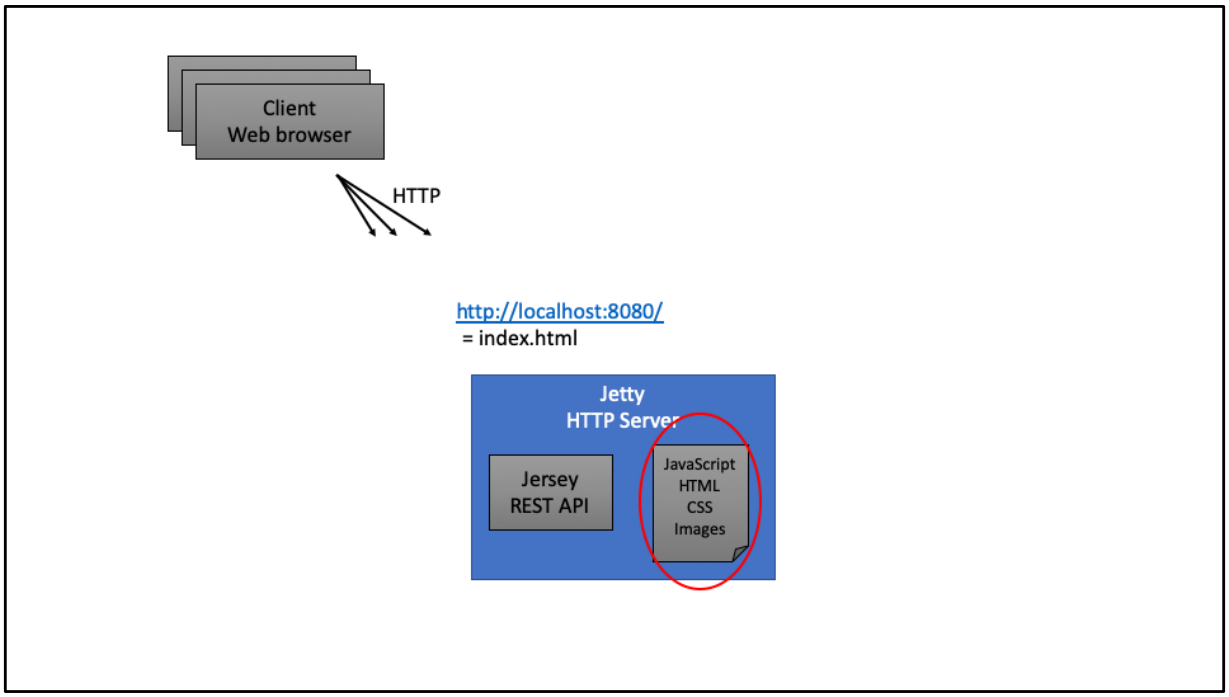


REST APIet och statiska filer kan ligga på olika
HTML, CSS, Javascript, bilder, är statiska filer
Det är precis samma sak för servern, bara data
hand om

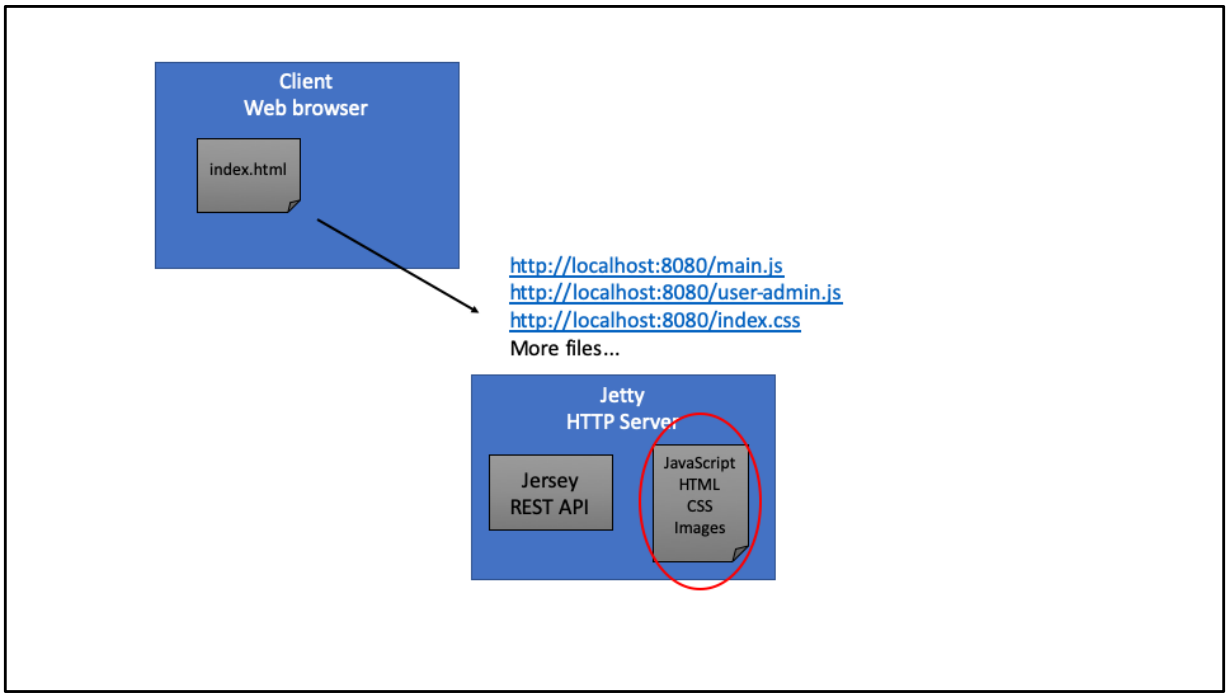


I BASE ligger det på samma server
I projektet är det samlat under en mapp: webassets
(precis som all java kod är samlad under en

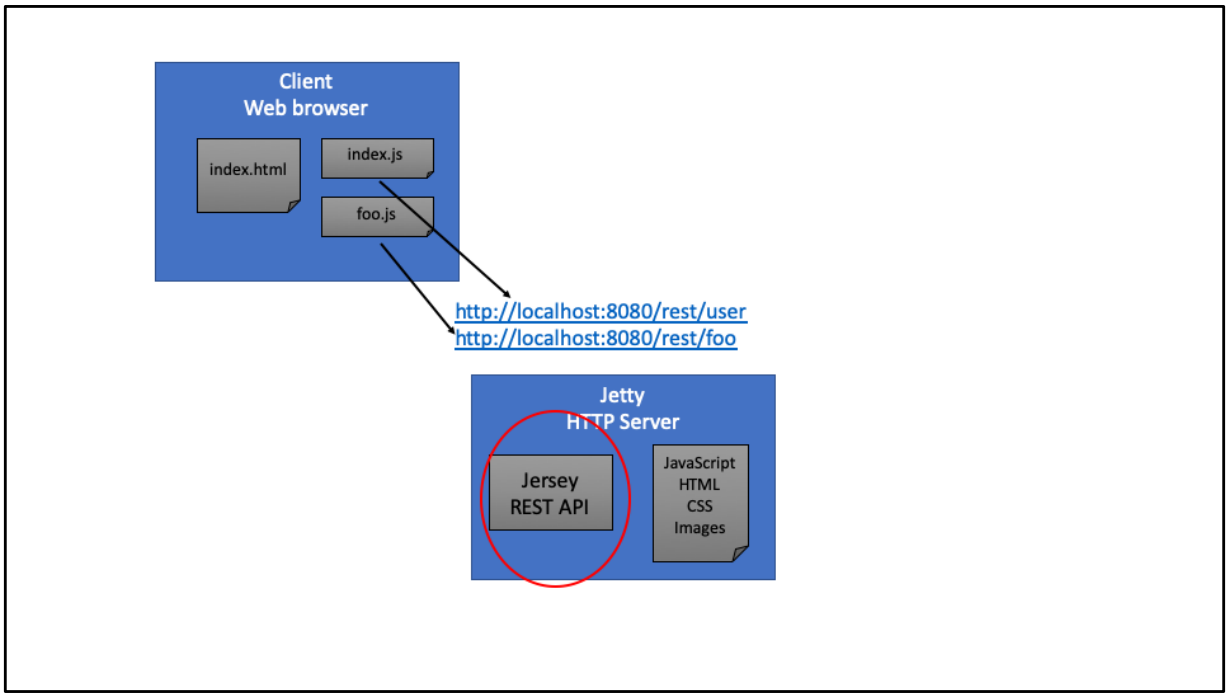
URLen bytt till localhost
localhost används när klient och server är
Notera porten 8080, när default 80 används
Måste explicit öppna port 80, men 8080 är
Kan inte göra det på skoldatorerna



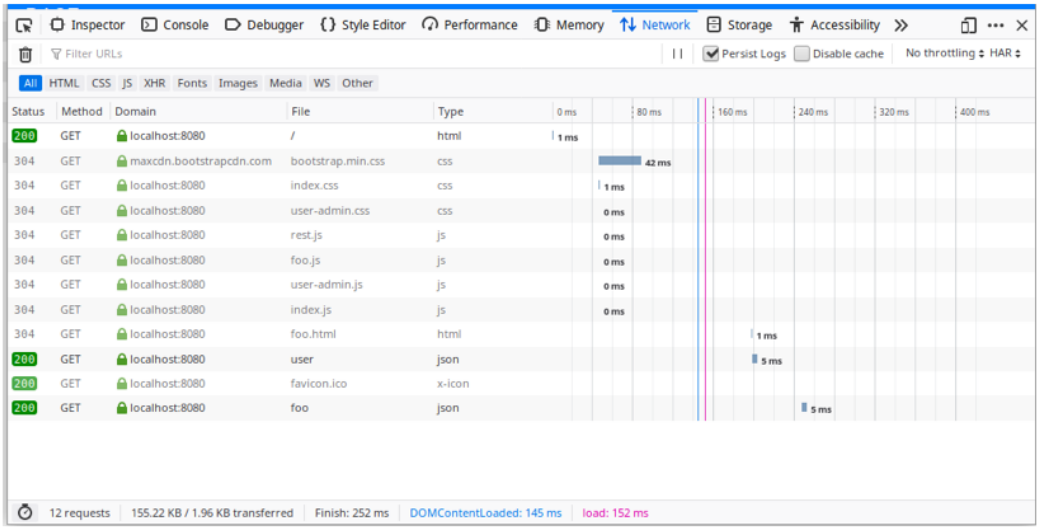
Hur går ett vanligt anrop till när klienten ska
Skriver in adressen på vår server I webbläsaren
Roten, dvs /, pekar på index.html
Det är en HTML fil som ligger I webassets



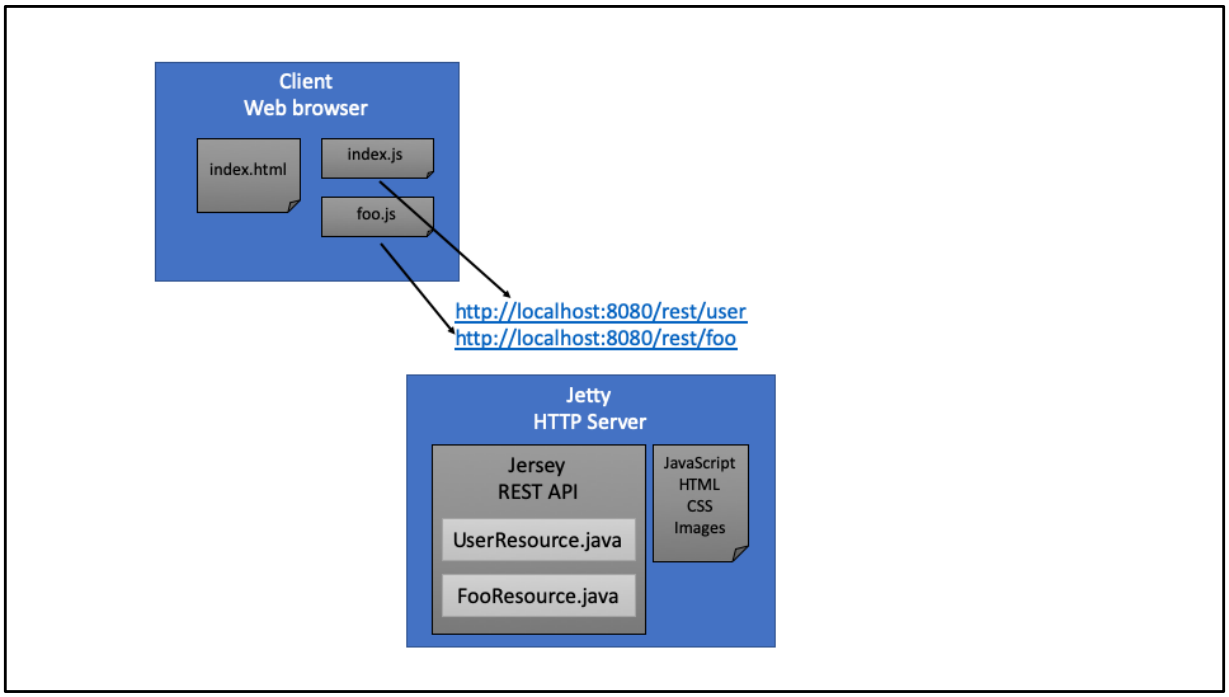
Index.html laddas in till användaren
Där står att det behövs mer filer från servern
Alla filer laddas direkt (lite långsamt men ok)
Norm idag att ha många filer, men kan laddas
osv



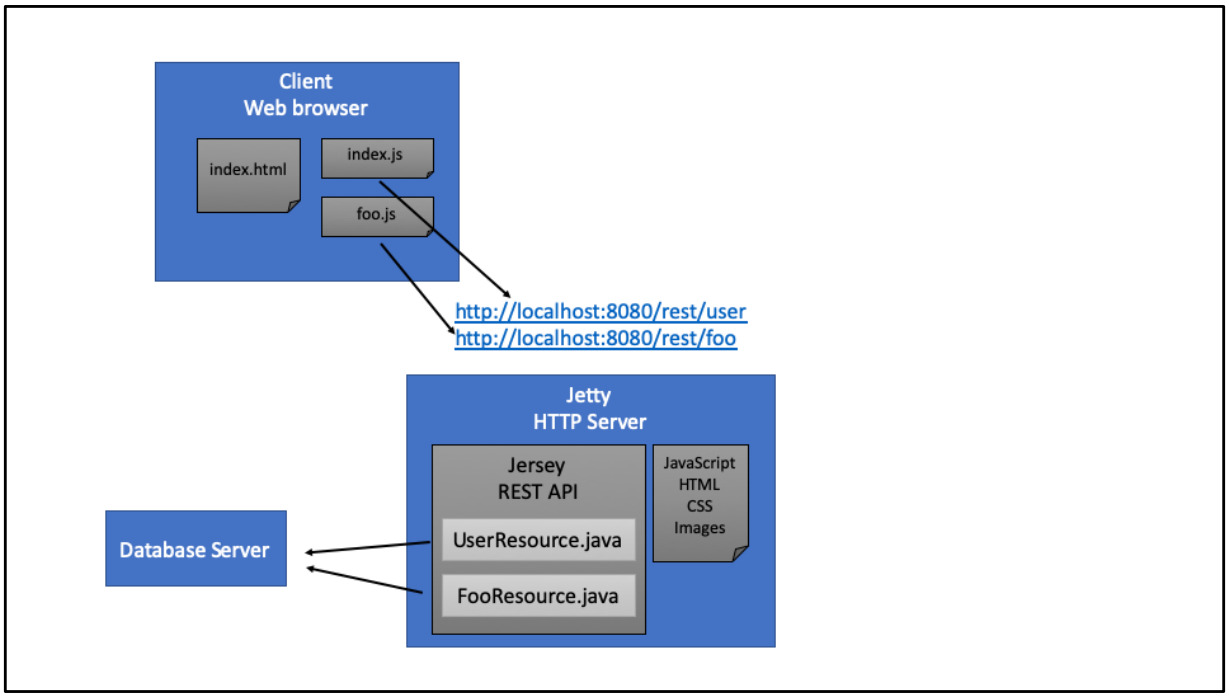
JavaScripten är det som anropar REST APIet
Index.js frågar om användaren är inloggad
Om inte dirigeras man om till login annars
foo.js hämtar ut användarens data
Foo är ett generiskt namn som ni kan bygga
data



Web developer tools från firefox
Visar tidslinje över nedladdad data
304 är statiskt innehåll, betyder att det redan är laddat I webbläsaren



I Jersey finns java klasser för båda resurser
Var får dom sin data ifrån?



Dom anropar i sin tur en databas
Databasen skyddas av servern
Klienten har inte direkt access till databasen

SQL

Database overview

How to store data?

- In Java Collections?
  - Loose data when server is restarted...
- In files?
  - Depends on file system...
- In Database Management Systems (DBMS)?
  - "Made for this"
  - Atomicy, consistency, isolation, durability
  - Different views
  - ...

Java minne tappas när servern stängs av, ytterst

SQL

- Structured Query Language
- Language for managing data in a relational database
- H2 is one system for this, PostgreSQL another

Data stored in tables (relational database)

- Students:
  - Student name
  - Birth year
  - Program
- Courses:
  - Course name
  - Teacher
  - Book
- Course selections:
  - Student name
  - Course name

Relation mellan student och kurs lagras I en egen tabell
Många till många relation

SQL keys

- Students:
  - Student name (primary key)
  - Birth year
  - Program
- Courses:
  - Course name (primary key)
  - Teacher
  - Book
- Course selections:
  - Student name (foreign key, cascade)
  - Course name (foreign key, cascade)

Primary key är unik, bara en per tabell
Foreign key är en relation till en annan tabell
Oftast används id istället för namn
Tar man bort student eller kurs kan course selections tas bort automatiskt, inte default

Some example statements

- create table Respondents (
  id integer auto_increment,
  name varchar(100),
  primary key (id));
- insert into Respondents (name)
  values('Martin Höst');
- select * from Respondents;

Id genereras automatiskt, behöver man inte ange

Design of databases

- Important in order to avoid duplicated values, inconsistencies, …
- Taught in courses on Databases

Det är ok om det inte är perfekt
Många koncept som ni inte känner till
Normalisering, index, osv.

Token based authentication/authorization

```
POST https://base.se/login
{
    "username": "Rasmus",
    "password": "Password123"
}

HTTP/1.1 204 No Content
Set-Cookie: USER_TOKEN=a3e195b159c119f189a1ab891
```

- User table: name, id, role
- Session table: token, userId
- Jersey `@RolesAllowed("Admin")`

Authentication, vem är du?
Authorization, vad får du göra?
Istället för att skicka med användar/lösenord I varje request, skicka token
Tokens skapas när man loggar in av servern och ges till klienten
Klienten skickar sedan med sin token I varje request
Kan "logga ut" genom att ta bort cookien programmatiskt eller rensa cookies I webbläsaren