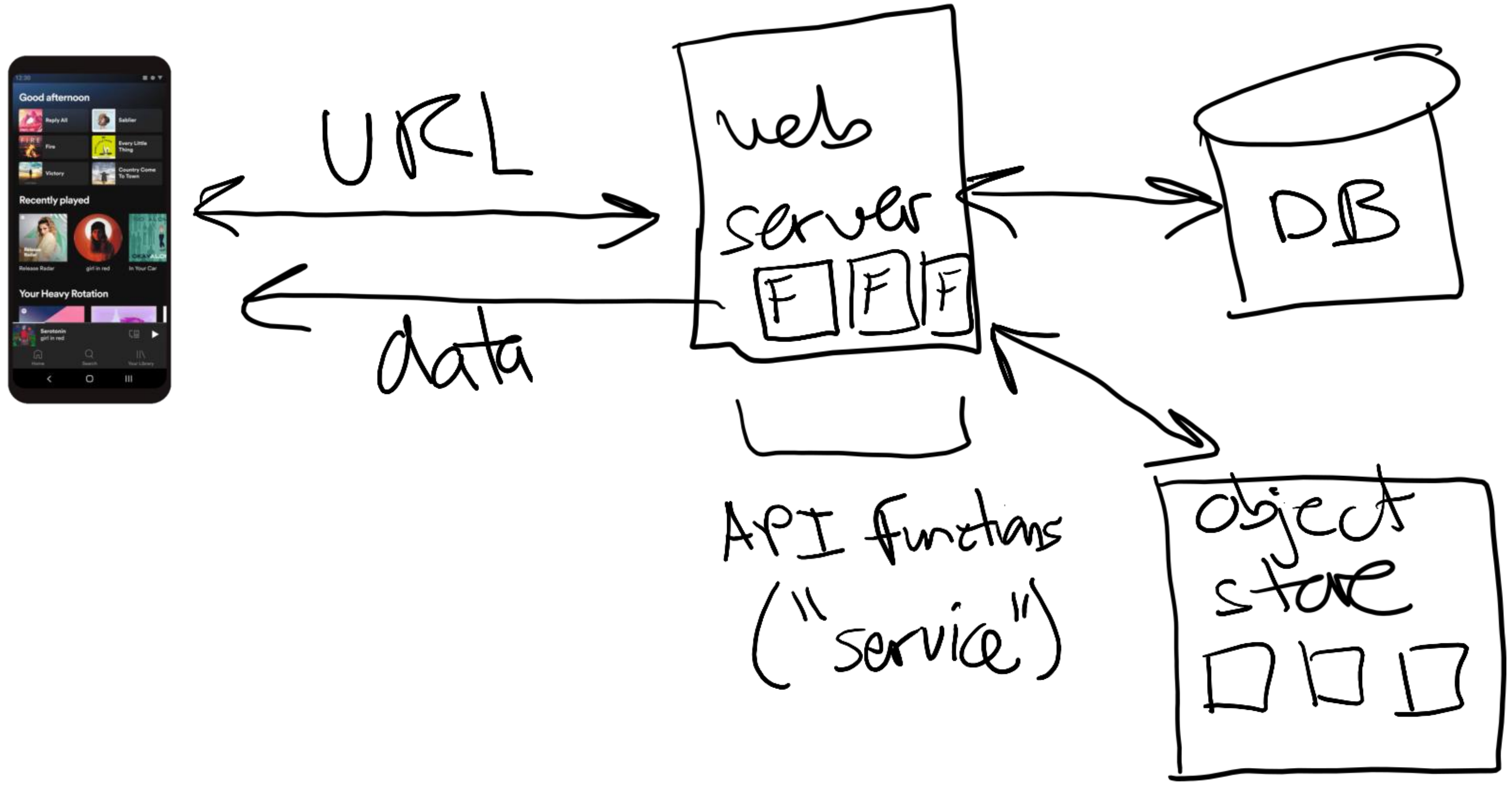
Web Services, Part 02

- Web services, part 02
- Data serialization and JSON
- Parameter passing
- Example: web service for the movielens DB

Northwestern University

Service-oriented architectures (SOA)

- **Redesign server-side as 1 or more services**
- **Expose the services using web technologies (HTTP / HTTPS)**

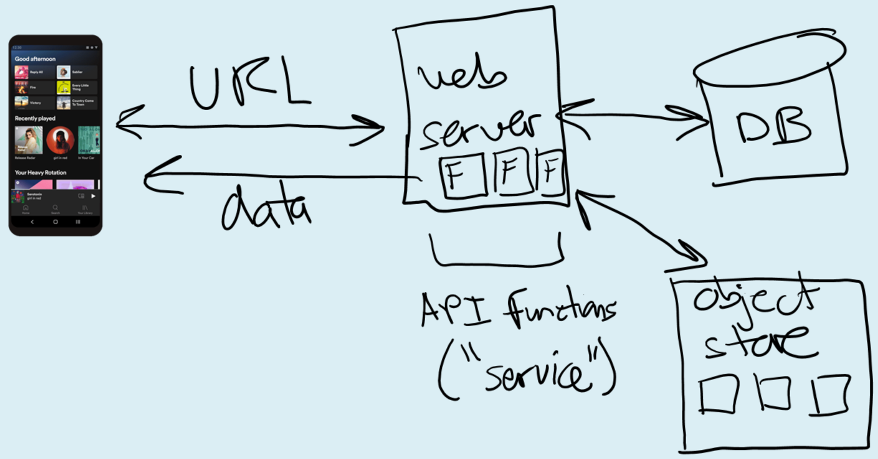


Performance?

- **Most important optimization in modern apps?**
- **Minimize trips between machines**

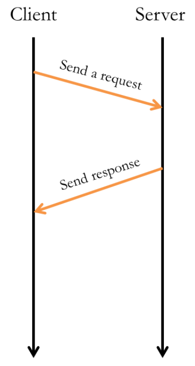
Here distance is the major factor, so language choice does not make much impact. Speed up which C++ like languages can bring are redundant for these use-cases.

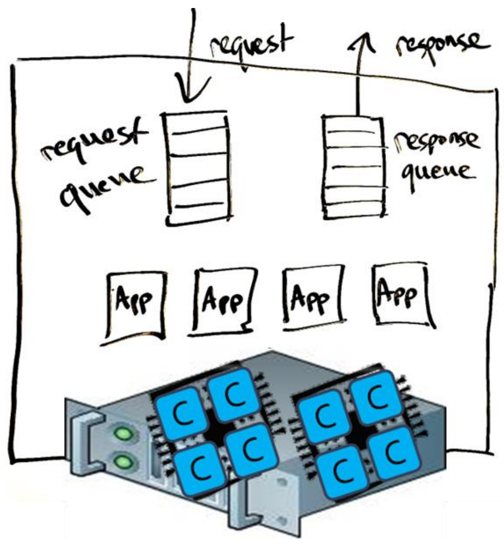
  - *Client-to-server AND server-to-server within cloud*
  - *Size of payload (msg) much less important than trip itself*
  - *Minimize trips / API calls*
  - *Batch SQL queries into one SQL program*

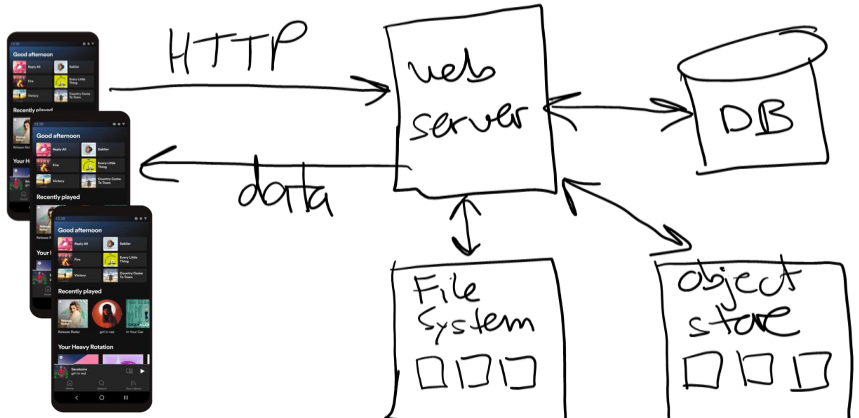


Why use Web Services for building applications?

- Web community has already solved the problems designers face:
  - Support for concurrent users
  - Client-side configuration
  - Security / encryption
  - Supports most programming languages
  - Lots of programming frameworks

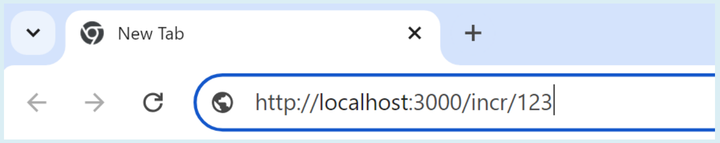






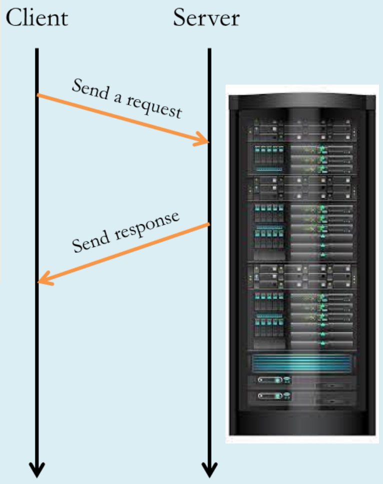
Example from last time

- A simple calculator web service:



http://localhost:3000/incr/123

```python
import requests

url = 'http://localhost:3000/'

x = input('Enter integer x> ')
url = url + 'incr/' + x
response = requests.get(url)
print(response.text)
```

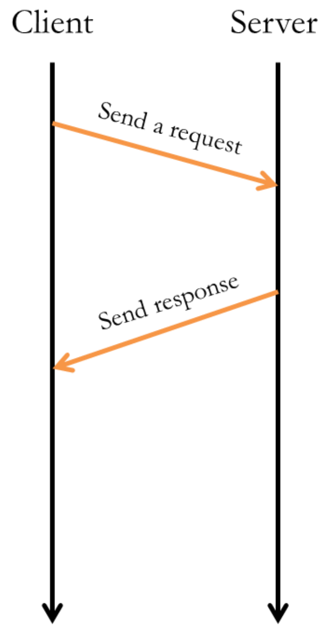


```javascript
const express = require('express');
const app = express();

// main():
app.listen(3000, () => {
  console.log('**SERVER: running...');
});

// increment x:
app.get('/incr/:x', (req, res) => {

  try {
    console.log('**call to /incr');
    let x = parseInt(req.params.x);
    if (isNaN(x))
      throw new Error('x not a number');
    let y = x + 1;
    res.send(y.toString());
    return;
  }
  catch(err) {
    res.status(400).send(err.message);
  }

});
```

HTTP requests and responses

Requests (most common)

- **GET**: to request a data
- **POST**: to post data to the server, and perhaps get data back

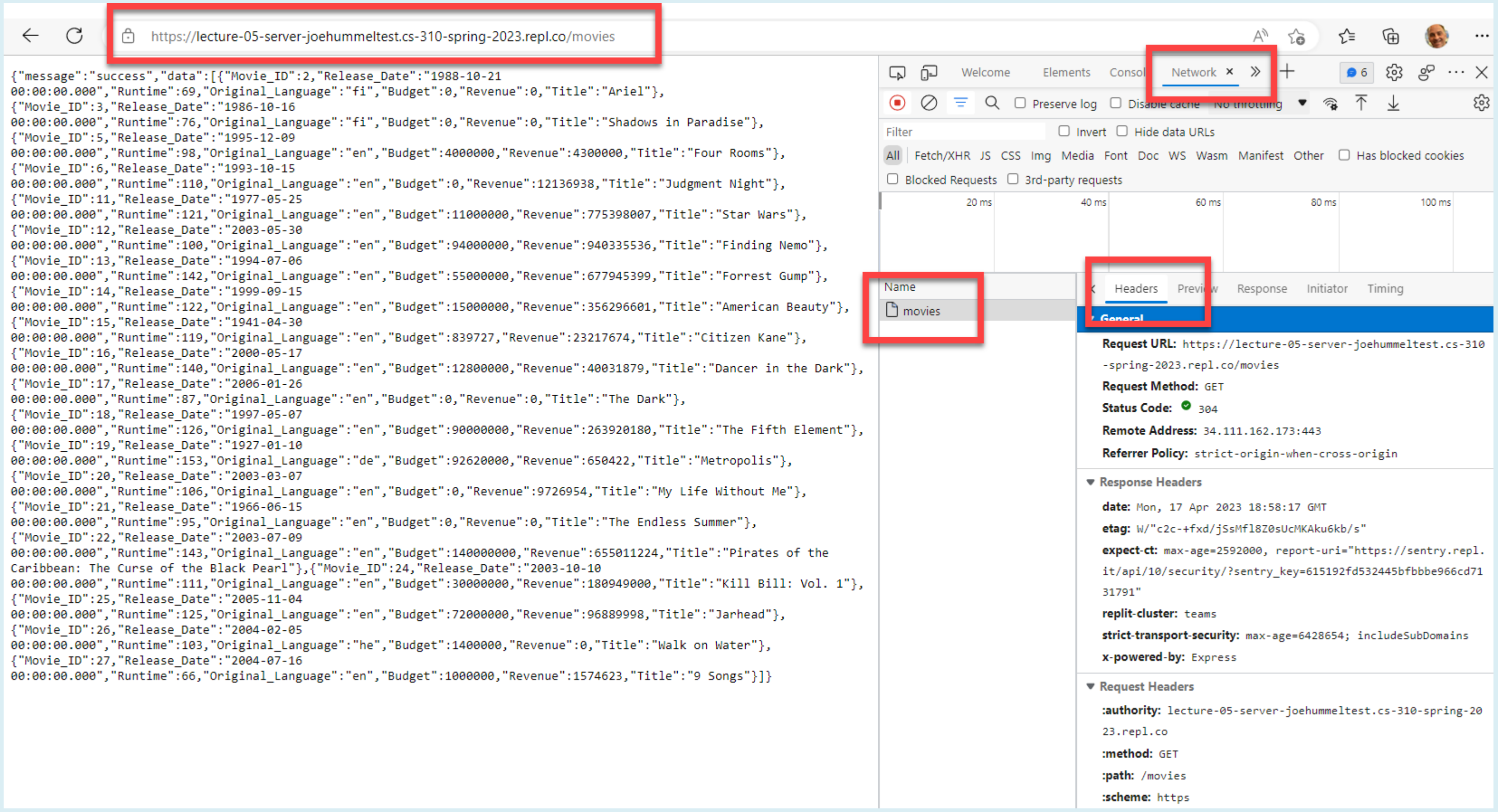


Response codes

- **200 OK**: success
- **301 Moved Permanently**
- **403 Forbidden**
- **404 Not Found**
- **500 Internal Server Error**

Viewing requests & responses

- **Use web browser's developer tools**
  - *Open browser, right-click on page, Inspect, Network tab, browse to URL*



```
{"message":"success","data":[{"Movie_ID":2,"Release_Date":"1988-10-21
00:00:00.000","Runtime":69,"Original_Language":"fi","Budget":0,"Revenue":0,"Title":"Ariel"},
{"Movie_ID":3,"Release_Date":"1986-10-16
00:00:00.000","Runtime":76,"Original_Language":"fi","Budget":0,"Revenue":0,"Title":"Shadows in Paradise"},
{"Movie_ID":5,"Release_Date":"1995-12-09
00:00:00.000","Runtime":98,"Original_Language":"en","Budget":4000000,"Revenue":4300000,"Title":"Four Rooms"},
{"Movie_ID":6,"Release_Date":"1993-10-15
00:00:00.000","Runtime":110,"Original_Language":"en","Budget":0,"Revenue":12136938,"Title":"Judgment Night"},
{"Movie_ID":11,"Release_Date":"1977-05-25
00:00:00.000","Runtime":121,"Original_Language":"en","Budget":11000000,"Revenue":775398007,"Title":"Star Wars"},
{"Movie_ID":12,"Release_Date":"2003-05-30
00:00:00.000","Runtime":100,"Original_Language":"en","Budget":94000000,"Revenue":940335536,"Title":"Finding Nemo"},
{"Movie_ID":13,"Release_Date":"1994-07-06
00:00:00.000","Runtime":142,"Original_Language":"en","Budget":55000000,"Revenue":677945399,"Title":"Forrest Gump"},
{"Movie_ID":14,"Release_Date":"1999-09-15
00:00:00.000","Runtime":122,"Original_Language":"en","Budget":15000000,"Revenue":356296601,"Title":"American Beauty"},
{"Movie_ID":15,"Release_Date":"1941-04-30
00:00:00.000","Runtime":119,"Original_Language":"en","Budget":839727,"Revenue":23217674,"Title":"Citizen Kane"},
{"Movie_ID":16,"Release_Date":"2000-05-17
00:00:00.000","Runtime":140,"Original_Language":"en","Budget":12800000,"Revenue":40031879,"Title":"Dancer in the Dark"},
{"Movie_ID":17,"Release_Date":"2006-01-26
00:00:00.000","Runtime":87,"Original_Language":"en","Budget":0,"Revenue":0,"Title":"The Dark"},
{"Movie_ID":18,"Release_Date":"1997-05-07
00:00:00.000","Runtime":126,"Original_Language":"en","Budget":90000000,"Revenue":263920180,"Title":"The Fifth Element"},
{"Movie_ID":19,"Release_Date":"1927-01-10
00:00:00.000","Runtime":153,"Original_Language":"de","Budget":92620000,"Revenue":650422,"Title":"Metropolis"},
{"Movie_ID":20,"Release_Date":"2003-03-07
00:00:00.000","Runtime":106,"Original_Language":"en","Budget":0,"Revenue":9726954,"Title":"My Life Without Me"},
{"Movie_ID":21,"Release_Date":"1966-06-15
00:00:00.000","Runtime":95,"Original_Language":"en","Budget":0,"Revenue":0,"Title":"The Endless Summer"},
{"Movie_ID":22,"Release_Date":"2003-07-09
00:00:00.000","Runtime":143,"Original_Language":"en","Budget":140000000,"Revenue":655011224,"Title":"Pirates of the
Caribbean: The Curse of the Black Pearl"},{"Movie_ID":24,"Release_Date":"2003-10-10
00:00:00.000","Runtime":111,"Original_Language":"en","Budget":30000000,"Revenue":180949000,"Title":"Kill Bill: Vol. 1"},
{"Movie_ID":25,"Release_Date":"2005-11-04
00:00:00.000","Runtime":125,"Original_Language":"en","Budget":72000000,"Revenue":96889998,"Title":"Jarhead"},
{"Movie_ID":26,"Release_Date":"2004-02-05
00:00:00.000","Runtime":103,"Original_Language":"he","Budget":1400000,"Revenue":0,"Title":"Walk on Water"},
{"Movie_ID":27,"Release_Date":"2004-07-16
00:00:00.000","Runtime":66,"Original_Language":"en","Budget":1000000,"Revenue":1574623,"Title":"9 Songs"}]}
```

Request URL: `https://lecture-05-server-joehummeltest.cs-310-spring-2023.repl.co/movies`
Request Method: `GET`
Status Code: `304`
Remote Address: `34.111.162.173:443`
Referrer Policy: `strict-origin-when-cross-origin`

▼ Response Headers

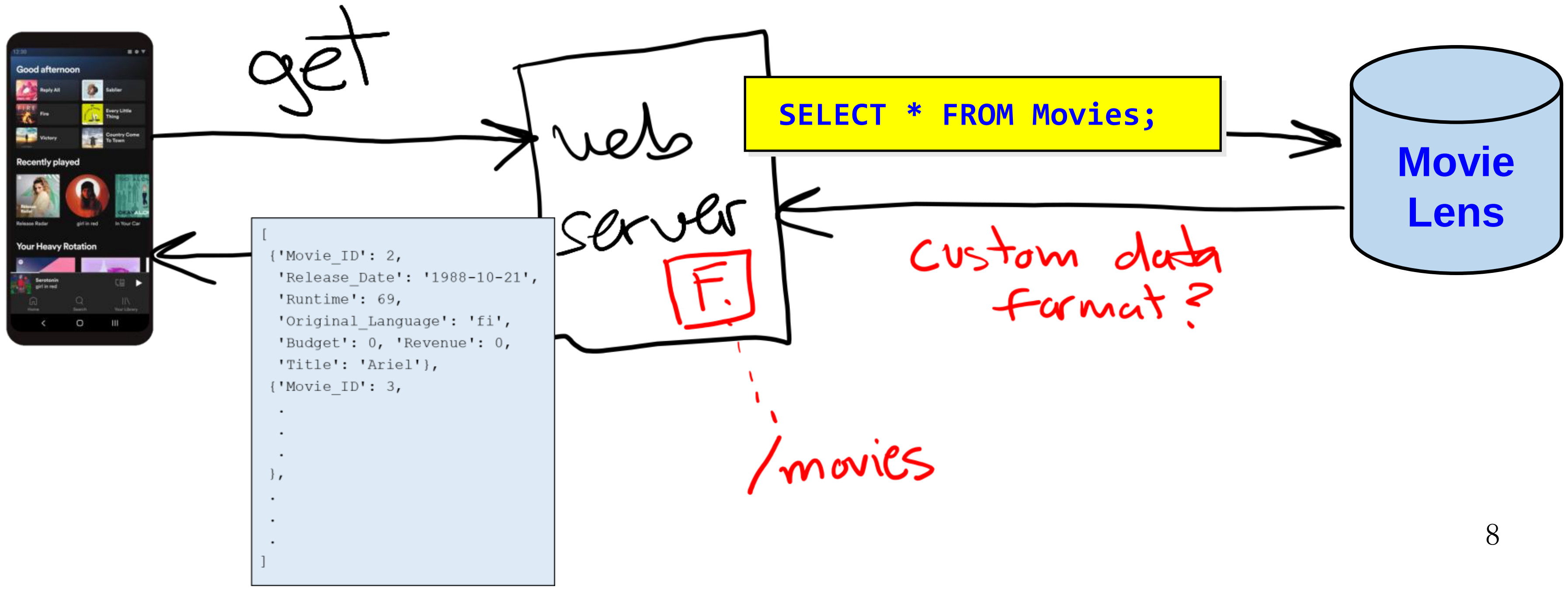
date: `Mon, 17 Apr 2023 18:58:17 GMT`
etag: `W/"c2c-+fxd/jSsMfl8Z0sUcMKAku6kb/s"`
expect-ct: `max-age=2592000, report-uri="https://sentry.repl.it/api/10/security/?sentry_key=615192fd532445bfbbbe966cd7131791"`
replit-cluster: `teams`
strict-transport-security: `max-age=6428654; includeSubDomains`
x-powered-by: `Express`

▼ Request Headers

:authority: `lecture-05-server-joehummeltest.cs-310-spring-2023.repl.co`
:method: `GET`
:path: `/movies`
:scheme: `https`

Serialization

- Data needs to be sent over the network…
- **Serialization** is the process of converting data / objects into a data stream that can be stored or transmitted over network…
  - *we don't want to send 0's and 1's, the two computers may be different platforms --- e.g. Windows x86 and Mac M1 --- where data formats are different*
  - *JSON (JS object notation) is a common choice --- platform-neutral and language-neutral*



```
SELECT * FROM Movies;
```

```
[
 {'Movie_ID': 2,
  'Release_Date': '1988-10-21',
  'Runtime': 69,
  'Original_Language': 'fi',
  'Budget': 0, 'Revenue': 0,
  'Title': 'Ariel'},
 {'Movie_ID': 3,
  .
  .
  .
 },
 .
 .
 .
]
```

JSON – JavaScript Object Notation

- A data format used by most web services
- Allows an arbitrary amount of **nesting**
- Whitespace is ignored

Basic components are:

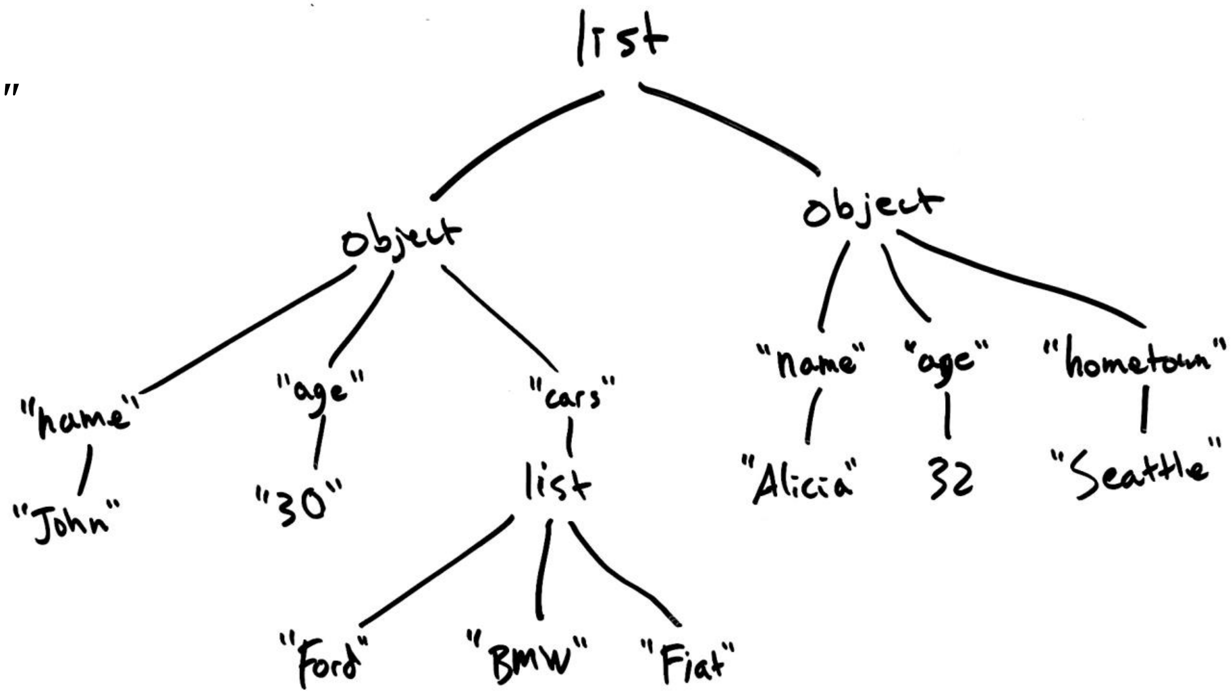
- `[ ]` for ordered lists
  - Items are separated by commas
  - Items can be any JSON
- `{ }` for unordered dictionaries/objects
  - Key: value pairs are separated by commas
  - Keys must be strings (text)
  - Values can be any JSON
- Numbers, `true, false, null`
- Strings in quotes

```
[
 {'Movie_ID': 2,
  'Release_Date': '1988-10-21',
  'Runtime': 69,
  'Original_Language': 'fi',
  'Budget': 0, 'Revenue': 0,
  'Title': 'Ariel'
 },
 {'Movie_ID': 3,
  .
  .
  .
 },
 .
 .
 .
]
```

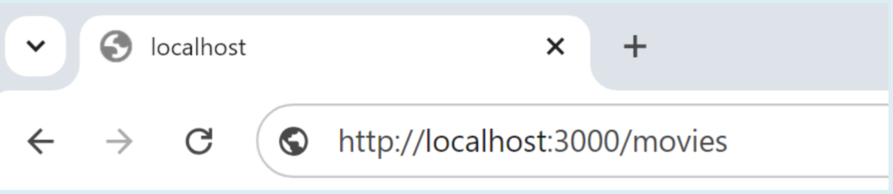
Credits: slides from Prof. Tarzia's CS 310 materials

Example: JSON graph

```
[
  {
    "name": "John",
    "age": 30,
    "cars": ["Ford", "BMW", "Fiat"]
  },
  {
    "name": "Alicia",
    "age": 32,
    "hometown": "Seattle"
  }
]
```

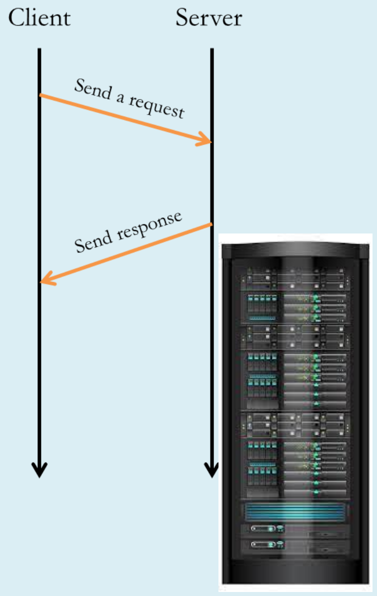


Sending a JSON response



localhost

http://localhost:3000/movies

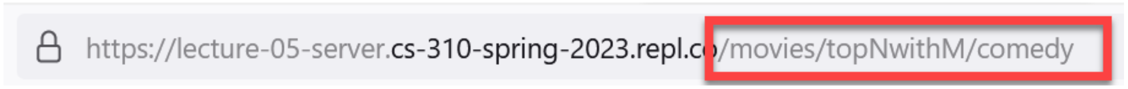


res.send() -> This is used for sending over the strings.
res.json() -> This is used to send over the JSON objects.

```
const express = require('express');
const app = express();

// main():
app.listen(3000, () => {
  console.log('**SERVER: running...');
});

// increment x:
app.get('/movies', (req, res) => {
  try {
    .
    .
    .

    // send response in JSON format:
    res.json( {"message": "success",
               "data":     rows} );

    return;
  }
  catch(err) {
    res.status(500).json({"message": …});
  }
});
```

Parameter passing

1. URL parameters



https://lecture-05-server.cs-310-spring-2023.repl.co/movies/topNwithM/comedy

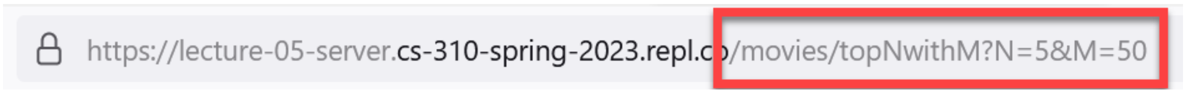
- **Parameters required (they form part of the path)**
- More natural, easier to read?

```
//
// Retrieves top N movies having at least M reviews,
// but focusing on the given genre.
//
app.get('/movies/topNwithM/:genre', (req, res) => {

  genre = req.params.genre;
```

Parameter passing (cont'd)

- **Two approaches**

  2. *Query strings*



https://lecture-05-server.cs-310-spring-2023.repl.co/movies/topNwithM?N=5&M=50

- **Parameters are optional**

These need not to be passed in the URL everytime. If not specified, their default value is used.

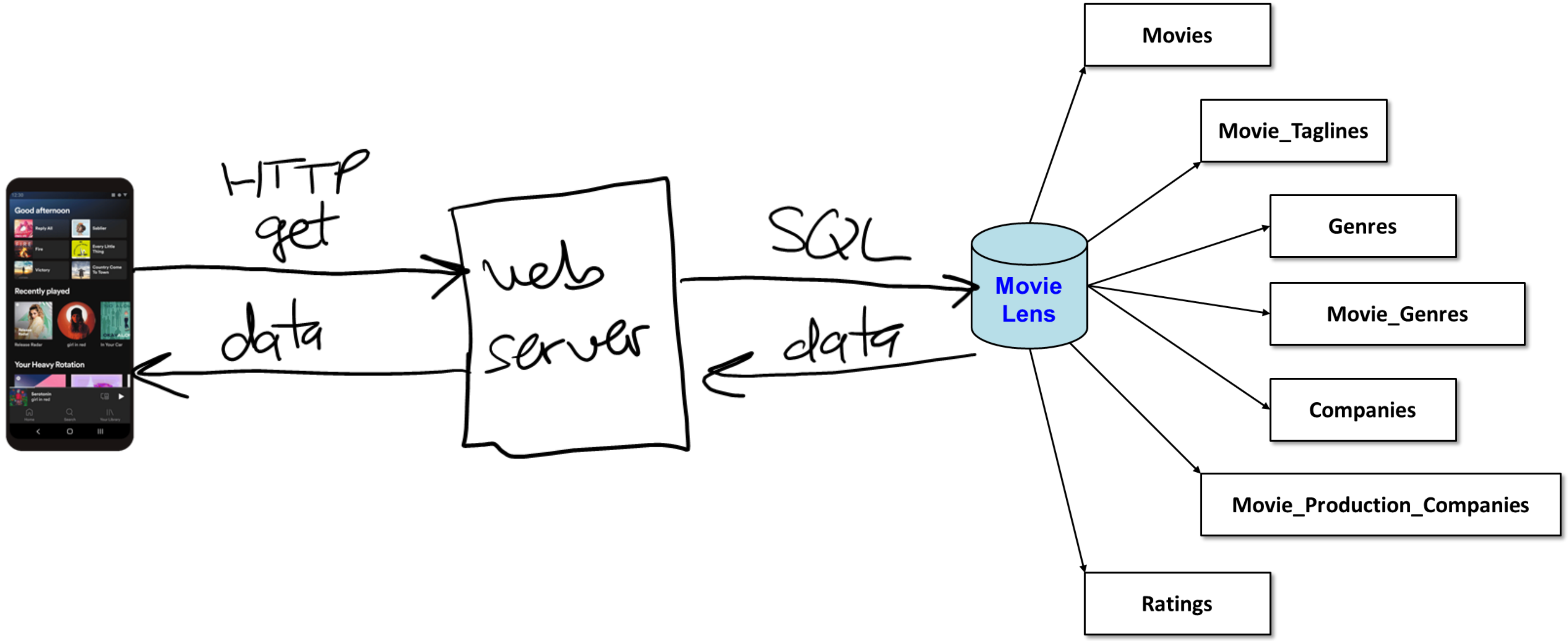
- Can have any # of params

```
//
// Retrieves top N movies having at least M reviews.
//
app.get('/movies/topNwithM', (req, res) => {

  N = 10;  // defaults:
  M = 100;

  if (req.query.N) {
    N = parseInt(req.query.N);
  }
```

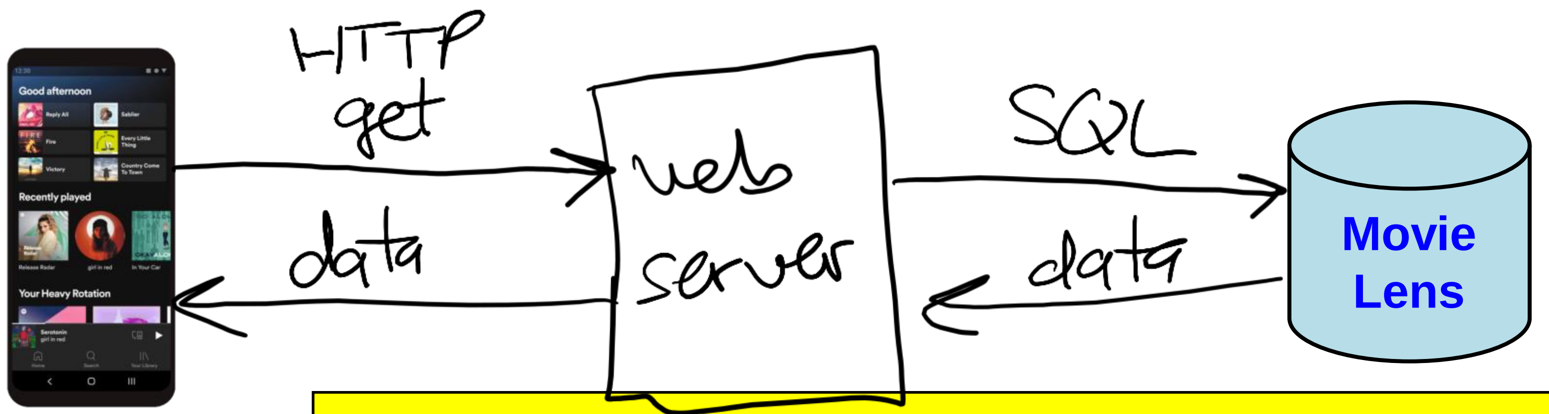
EXAMPLE

A more realistic Example

- Let's build a web service for the MovieLens database

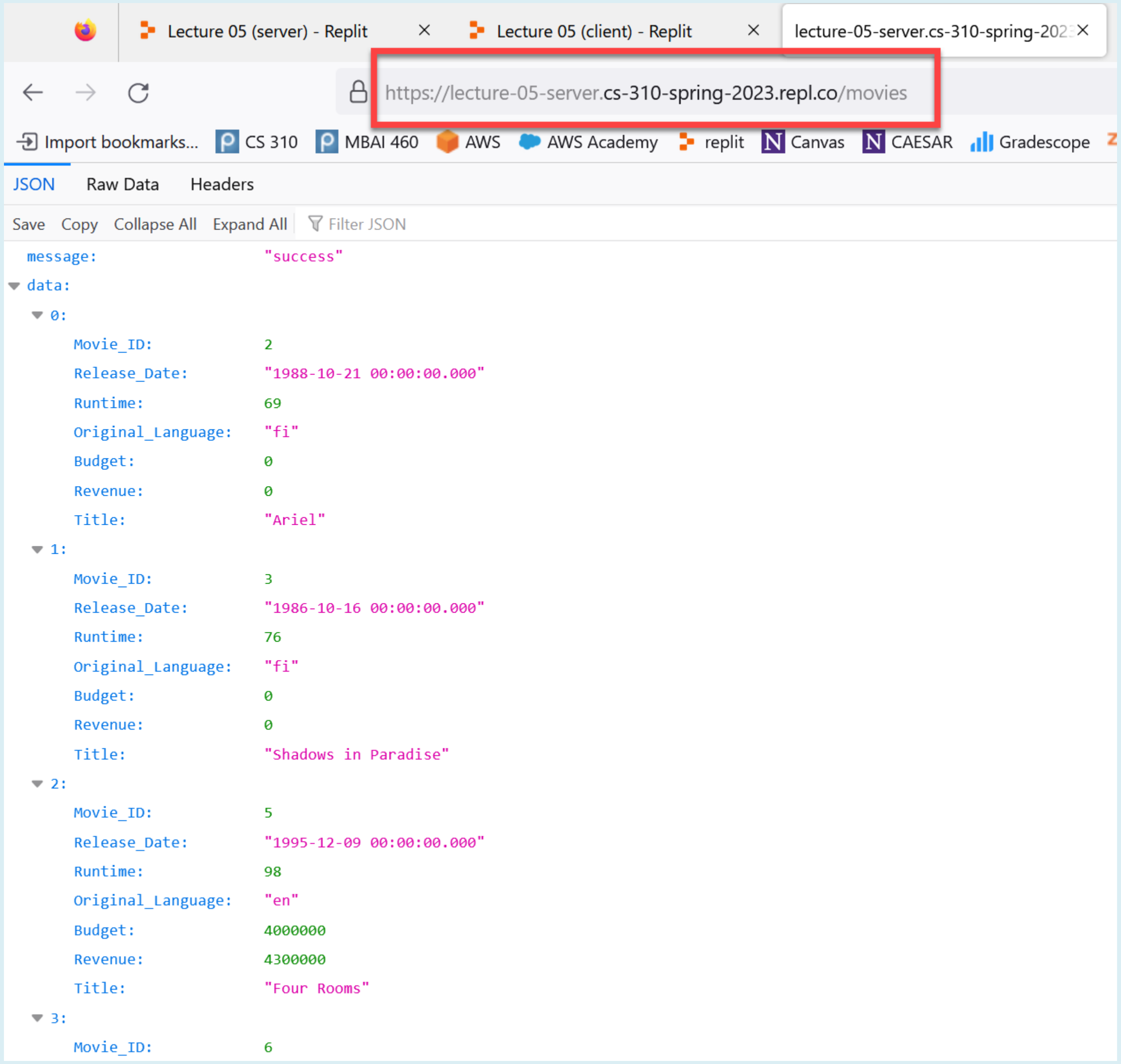


MovieLens web service



```
app.get('/movies', (req, res) => {…});

app.get('/movies/top10', (req, res) => {…});

// top N with at least M reviews (defaults to N=10, M=100):
app.get('/movies/topNwithM', (req, res) => {…});

// top N with at least M reviews, in given genre:
app.get('/movies/topNwithM/:genre', (req, res) => {…});


.
.
.
```

Demo using browser as client



Lecture 05 (server) - Replit
Lecture 05 (client) - Replit
lecture-05-server.cs-310-spring-202
Import bookmarks... CS 310 MBAI 460 AWS AWS Academy replit Canvas CAESAR Gradescope
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON

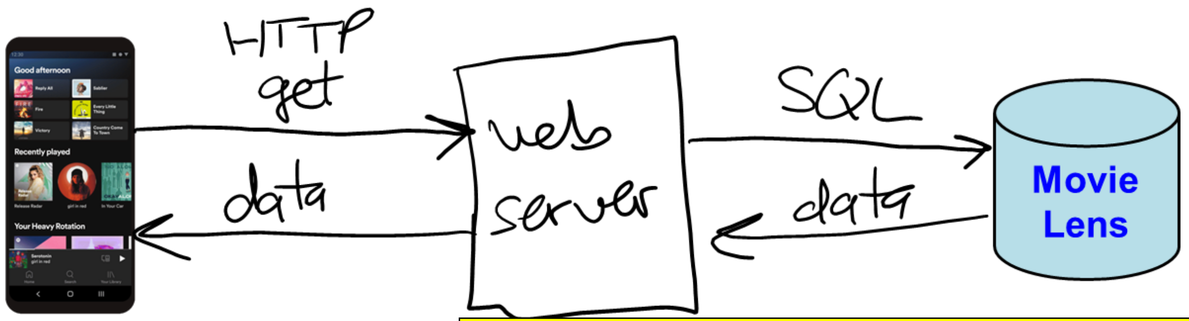
https://lecture-05-server.cs-310-spring-2023.repl.co/movies

```
message:              "success"
data:
  0:
    Movie_ID:           2
    Release_Date:       "1988-10-21 00:00:00.000"
    Runtime:            69
    Original_Language:  "fi"
    Budget:             0
    Revenue:            0
    Title:              "Ariel"
  1:
    Movie_ID:           3
    Release_Date:       "1986-10-16 00:00:00.000"
    Runtime:            76
    Original_Language:  "fi"
    Budget:             0
    Revenue:            0
    Title:              "Shadows in Paradise"
  2:
    Movie_ID:           5
    Release_Date:       "1995-12-09 00:00:00.000"
    Runtime:            98
    Original_Language:  "en"
    Budget:             4000000
    Revenue:            4300000
    Title:              "Four Rooms"
  3:
    Movie_ID:           6
```

/movies

```
//
// Retrieve all movies in the database:
//
app.get('/movies', (req, res) => {
  try {
    console.log("**call to /movies");

    let sql = "Select * From Movies Order By Movie_ID;";
    let params = [];

    // execute the SQL:
    movielens.all(sql, params, (err, rows) => {
        if (err) {
          res.status(500).json( {"message": err.message, "data": []} );
          return;
        }

        // send response in JSON format:
        console.log("sending response");
        res.json( {"message": "success", "data": rows} );
    });

    console.log("about to return");
    return;
   }
   catch(err) { res.status(500).json({"message": err.message, "data": []}); }
});
```

Other service functions



```
app.get('/movies', (req, res) => {…});

app.get('/movies/top10', (req, res) => {…});

// top N with at least M reviews (defaults to N=10, M=100):
app.get('/movies/topNwithM', (req, res) => {…});

// top N with at least M reviews, in given genre:
app.get('/movies/topNwithM/:genre', (req, res) => {…});
```

http://localhost:3000/movies/top10

http://localhost:3000/movies/topNwithM?N=5&M=50

http://localhost:3000/movies/topNwithM/Drama?N=3&M=200

/movies/topNwithM/:genre

```
//
// Retrieve top N movies with at least M reviews, in
// the given genre. Defaults of N=10 and M=100.
//
app.get('/movies/topNwithM/:genre', (req, res) => {
  try {
     let N = 10;  // defaults:
     let M = 100;

     let genre = req.params.genre;

     if (req.query.N)
       N = parseInt(req.query.N);
     if (req.query.M)
       M = parseInt(req.query.M);

     let sql = `…`;
     let params = [genre, M, N];

     movielens.all(sql, params, (err, rows) => {
        if (err) {
          res.status(500).json({ "message": err.message, "data": [] });
          return;
        }

        // send response in JSON format:
        res.json({ "message": "success", "data": rows });
     });
  }
  catch(err) { res.status(500).json({"message": err.message, "data": []}); }
});
```

```
var sql = `Select Movies.Movie_ID, Title, Genre_Name,
                  Count(Rating) As NumReviews, Avg(Rating) As AvgRating
    From Movies
    Inner Join Ratings on Movies.Movie_ID = Ratings.Movie_ID
    Inner Join Movie_Genres on Movies.Movie_ID = Movie_Genres.Movie_ID
    Inner Join Genres on Genres.Genre_ID = Movie_Genres.Genre_ID
    Where Genre_Name like ?
    Group By Movies.Movie_ID
    Having NumReviews >= ?
    Order By AvgRating DESC, Title ASC
    Limit ?;
`;
```

Invalid parameters?

```
//
// Retrieve top N movies with at least M reviews, in
// the given genre. Defaults of N=10 and M=100.
//
app.get('/movies/topNwithM/:genre', (req, res) => {
  try {
     let N = 10;  // defaults:
     let M = 100;

     let genre = req.params.genre;

     if (req.query.N)
       N = parseInt(req.query.N);
     if (req.query.M)
       M = parseInt(req.query.M);

     let sql = `…`;
     let params = [genre, M, N];

     movielens.all(sql, params, (err, rows) => {
        if (err) {
          res.status(500).json({ "message": err.message, "data": [] });
          return;
        }

        // send response in JSON format:
        res.json({ "message": "success", "data": rows });
     });
  }
  catch(err) { res.status(500).json({"message": err.message, "data": []}); }
});
```

```
if (isNaN(N)) {
  res.status(400).json({ message: "N is not a number",
                         data: [] });
  return;
}
if (isNaN(M)) {
  res.status(400).json({ message: "M is not a number",
                         data: [] });
  return;
}
```

Client-side

```
import requests

baseurl = 'http://localhost:3000'  ## no / at the end

api_movies = '/movies'
api_top10 = '/top10'
api_topNwithM = '/movies/topNwithM'

# get all the movies:
url = baseurl + api_movies

response = requests.get(url)

# let's look at what we got back:
print(type(response))

print(response.status_code)
```

```
# deserialize to Python objects:
body = response.json()

print(type(body))  # dictionary

msg = body['message']
print("message:", msg)

movies = body['data']
print("# of movies:", len(movies))
for i in range(0,10): # print first 10 rows
  print(movies[i])
```

```
{'message': …,
 'data': …   }
```

ORM

- ORM = Object Relational Mapping
- Relational data is often turned into client-side objects

```python
import jsons

class Movie:
  Movie_ID: int
  Title: str

.
.
.

# deserialize to Python objects:
body = res.json()

rows = body['data']
movies = []
for i in range(0,10): # map to Movie objects:
    m = jsons.load(rows[i], Movie)
    movies.append(m)
```

```python
for m in movies:
  print(m.Movie_ID, ":", m.Title)
```

Paging

- **There are 45,000 movies**

- **Options:**

  1. *Download all to client & display 1 page at a time*

```
i = 0
pagesize = 10
N = len(movies)

#
# show movies one page at a time:
#
while i < N:
  for r in range(i, min(i + pagesize, N)):
    row = movies[r]
    print(row["Movie_ID"], row["Title"])

  i += pagesize

  nextpage = input("Another page? [y/n] ")
  if nextpage == 'n':
    break
```

Is downloading 45,000 movies a good idea?

On a decent internet connection, yes. The general rule of thumb is to minimize trips to/from the server. ==> one BIG message is better than lots of smaller messages…

What if you want web server to paginate?

- **Options:**

1. *Use SQL's* ***Limit*** *clause*
   - Client passes offset and page size to the server
   - **Warning**: the server still selects all the rows, then returns the rows you want. ==> slow if there are a large number of rows (millions?)

```
SELECT *
FROM Movies
ORDER BY Movie_ID ASC
LIMIT offset, pagesize;
```

2. *Use a* ***WHERE*** *clause on primary key*
   - Client passes largest key from previous page, & pagesize
   - Much faster when you have large tables, but requires use of primary key / indexed column

```
SELECT *
FROM Movies
ORDER BY Movie_ID ASC
WHERE Movie_ID > prevkey
LIMIT pagesize;
```

That's it, thank you!