Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

# 5) Services

*Emmanuel Benoist*

Spring Term 2021

Computer Science Division

# Services

- Last week
- Introduction
- GET
- POST parameters
- Conclusion

# Last week

# Last week

- **SQL: language to work with databases**

  Select $*$ from table1, table2 where table1.key1 = table2.key2↘
  → order by fiel1

- **MySQL Node for Node RED**
  - Must be configured to connect the DB
  - Receives the query in `msg.topic`
  - Generates a JSON object on the output `msg.payload`

# Introduction

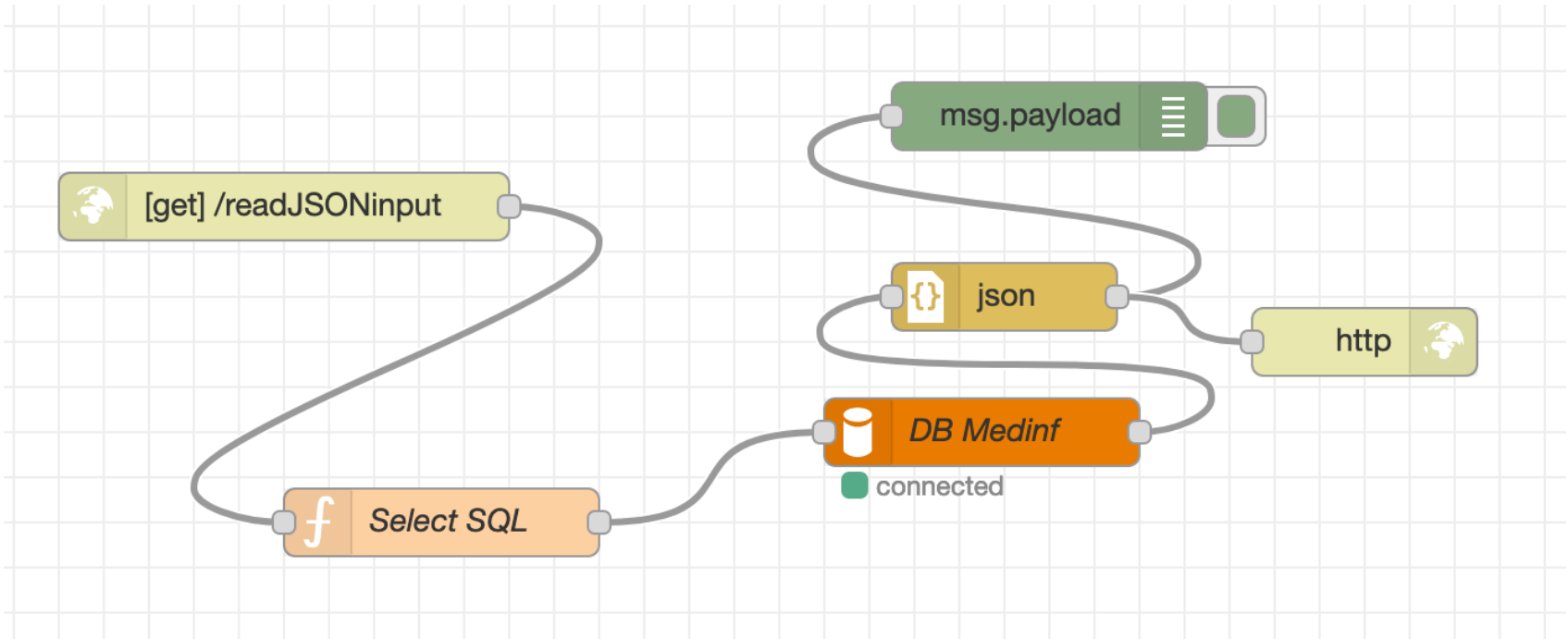
# Motivation

- **Computer communicate using JSON strings**
  - Client / Server communication (Web and Apps)
  - Server / Server communication
- **Get information from the server**
  - HTTP GET method
  - Read information from the server
  - Parameters are given inside the Query String (the URL)
- **Send information to the server**
  - HTTP POST method
  - Send information to the server
  - Information is larger and must be written inside the body

# GET

# The GET requests

- We reuse the same example as last week:
- Input : GET for the URL: `readJSONGET`
- Output: The JSON String containing patient / patients data
- Flow:



# The GET requests

- **We have the following code inside the function node:**

```
if(typeof(msg.req.query.patient)!== "undefined"){
    var pID = msg.req.query.patient;
    node.log("UserID␣="+pID);
    msg.topic="select␣*␣from␣patient,␣vital_sign␣where␣patient.↘
    →patientID␣=␣vital_sign.patientID";
    msg.topic+="␣and␣patient.patientID="+pID;
}
else{
     msg.topic="select␣*␣from␣patient";
}
return msg;
```

# GET parameters

- **If one has no parameters**
  - We have the list of all the patients
- **If one has one parameter** `patient=1`
  - We display the results for one single patient (patient number 1 in this case).

# Send GET requests

- **From within a Browser**
  - Type the URL : `http://localhost:1880/readJSONGET`
- **You can write parameters in the URL**
  - It starts with ?
  - Contains pairs: `attribute=value`
  - Pairs are separated using `&`
  - `https://www.mysite.com/prog?val1=1&val2=45&user=bie1`
  - `http://localhost:1880/readJSONGET?patient=1`

# Connect to a server using GET

- **One can use CURL to contact HTTP(S) servers**
  - A program that downloads the content of a page
  - Is used to connect servers
  - Can be used for tests
- **Syntax**

  $ curl http://*localhost:1880/readJSONGET*

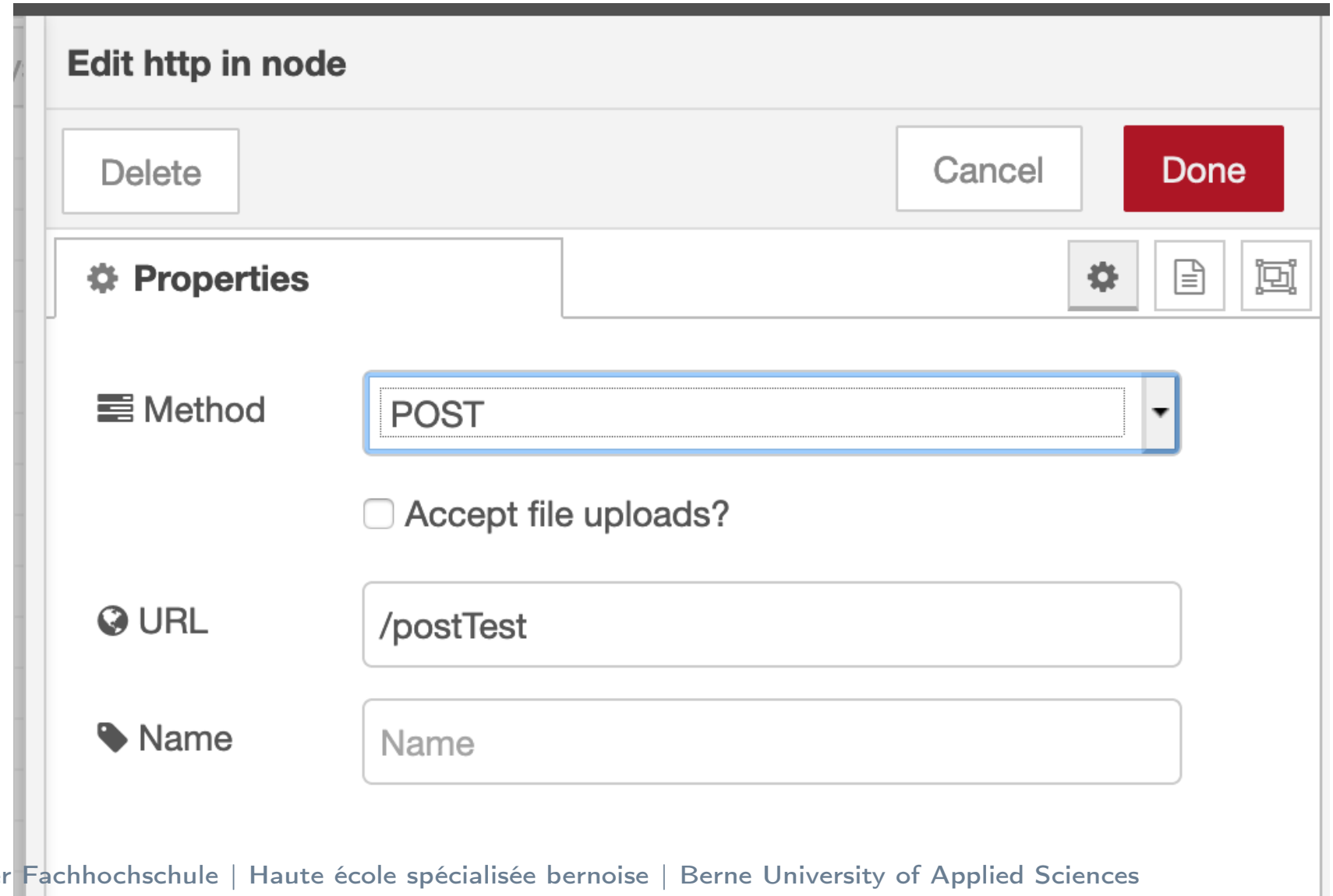
  [{"patientID":1,"MRN":"24598","name":"Laurie","↘
  →first_name":"Hugh","gender":1, ...
- **With a parameter**

  $ curl http://*localhost:1880/readJSONGET?patient=1*

# POST parameters

# POST Parameters

- **We change the method of the HTTP input node**
  - From GET to POST



# Send a POST request

- **Generated by HTML Formulars**
  - For web servers
  - Browser generates the content
- **But not in our case**
  - We want Program to Program interface (API)
  - We want to send a JSON object
- **Content:**

```
{
  "name" : "bie1"
}
```

# Send the POST request

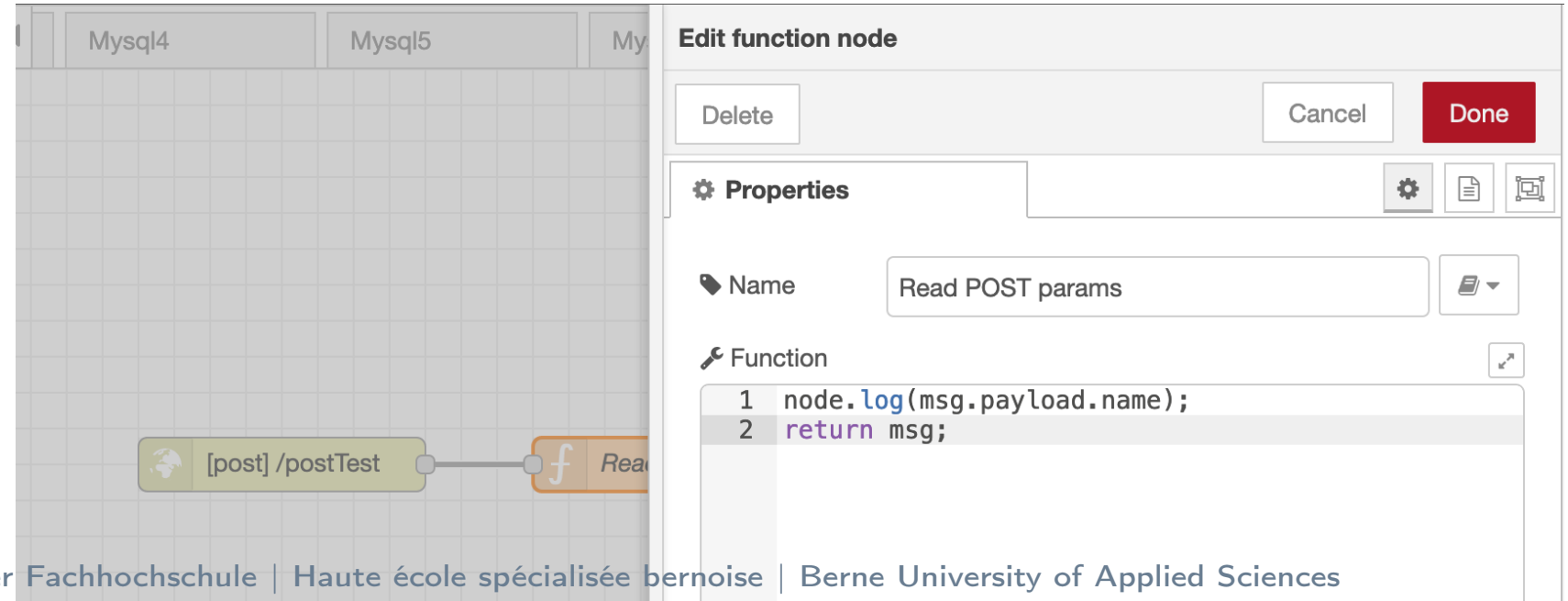
- **We use CURL**

  $ curl −X POST −d'{"name":"BIE1"}' −H "Content−type↘
  →:␣application/json" http://localhost:1880/postTest

- **We use JavaScript to read the input**

  node.log(msg.payload.name);
  return msg;

- **JSON is parsed into `msg.payload`**



# POST Request

- **We send a more complex JSON Query**

```
{
  "patient":2,
  "temperature":''37.5'',
  "notes":"Everything␣in␣order"
}
```

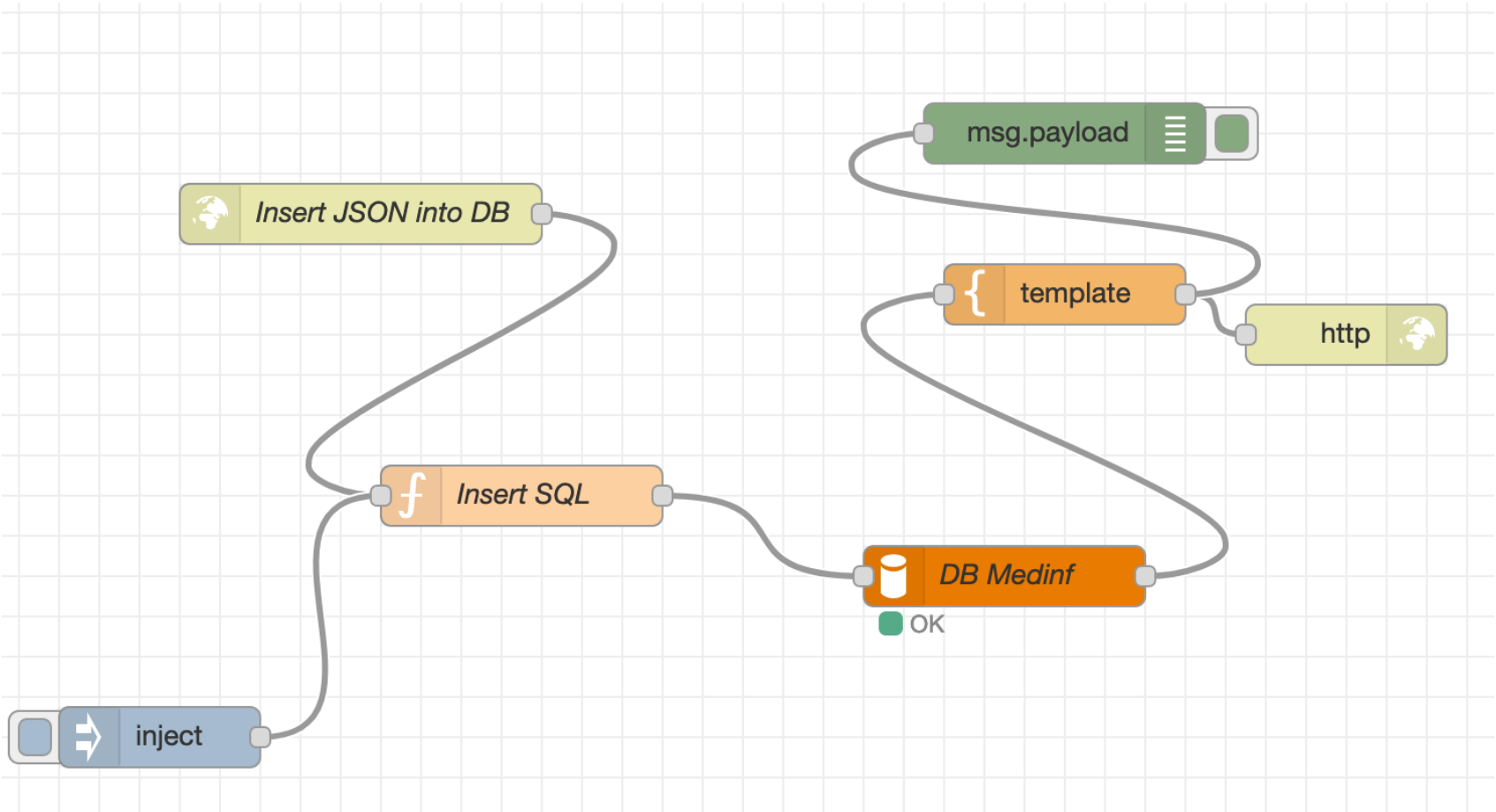
- **We can read the input in JavaScript (Function node)**

```
node.log(msg.payload.patient+"␣"+msg.payload.↘
→temperature);
```

- **CURL command**

```
$ curl −X POST −d'␣{"patient":2,"temperature":"37.5",␣"↘
→notes":"Everything␣in␣order"}' −H "Content−type:␣↘
→application/json" http://localhost:1880/insertData
```

# Example


msg.payload
Insert JSON into DB
template
http
Insert SQL
DB Medinf
OK
inject

# SQL Insert Query

- **We want to insert a new data into the table** `vital_sign`
  - `vital_signID` an autoincrement primary key
  - `patientID` foreign key to the patient table
  - `signID` foreign key to the sign table
  - `value` varchar
  - `time` timestamp (default is now)
  - `note` string
- **SQL looks like**

insert into vital_sign (patientID, signID, value, note) VALUES (2,1,'37.5','Everything␣in␣order');

# Define new SQL

- **Function node**

```
node.log(msg.payload.patient+"␣"+msg.payload.↘
→temperature);
sql = "insert␣into␣vital_sign␣(patientID,signID,value,note)␣↘
→VALUES␣(";
sql += msg.payload.patient+",1,"+msg.payload.↘
→temperature+",'"+msg.payload.notes+"')␣";
//We output the Generated query
node.log(sql);
msg.topic = sql;
return msg;
```

# Conclusion

# GET and POST Requests

- **GET to read information**
  - Parameter is in the query string
  - Can be read in `msg.req.query.XXXXXX`
- **POST to send information to the server**
  - Can contain much larger data
  - We used JSON
  - needs CURL to be tested
  - Can be read at `msg.payload`
- **JSON is used to communicate with the other applications**
  - See the App part.