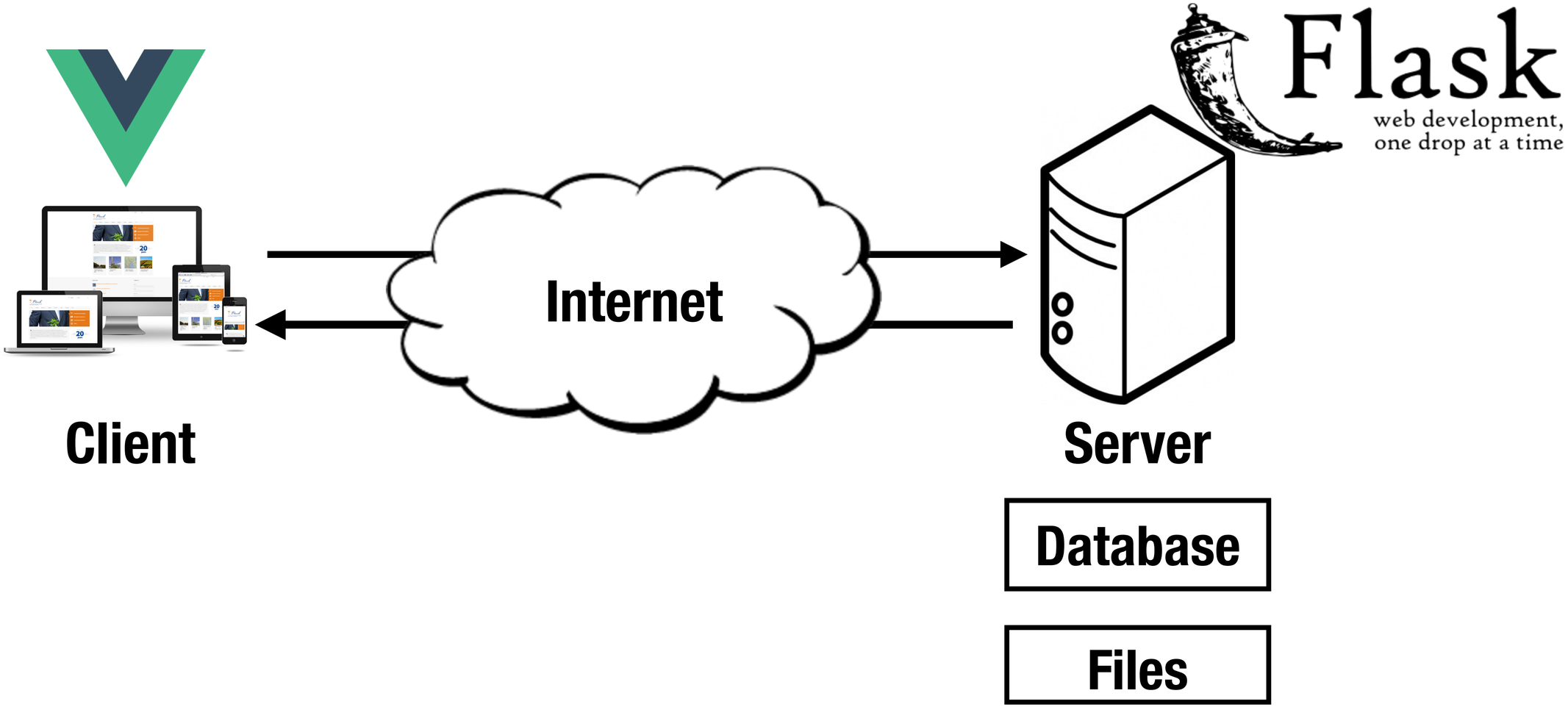
Web Programming

Decoupled REST API

Leander Jehl | University of Stavanger

Recap: Decoupled Client and server



Webserver Role:

Flask application using templates:

- Couple data and presentation
- Adjust HTML documents
- Implements business logic
- Implement display logic
- Manipulate data with forms

Using Vue.js and AJAX

- Serve initial HTML,JS, … files
- Serve data via AJAX & JSON
- Manipulate data via AJAX & JSON
- Can use data from other servers (if CORS allows)

Server-side APIs

- RESTful Web APIs
  - Accessing data independent from display
- Can maintain API independent from web application
- Can support different applications
- Can sell or offer the api to application developers

RESTful Web APIs

REST

- **RE**presentational **S**tate **T**ransfer
- REST is an architectural style (not a protocol)
  - Web service APIs are called RESTful
- Uniform interface separates clients from servers
  - Data storage is internal to the server
  - Servers are not concerned with the user's state
- Stateless
  - The client must provide all the information for the server to fulfill the request. No sessions.

Uniform interface

- Resources are identified by URIs
- Operations are performed on resources
- Resources are manipulated through representations
  - Representation contains enough information for the client to modify/delete it on the server
  - Representations are typically in JSON or XML format

RESTful web APIs

- HTTP based
- Resources are identified by URIs
  - E.g., http://example.com/resources/
- Operations correspond to standard HTTP methods
  - GET, PUT, POST, DELETE
- Media type is JSON

Typical RESTful API

	GET	PUT	POST	DELETE
Collection URI `http://example.com/resources`	**List** elements	**Replace** the entire collection	**Create** a new element in the collection	**Delete** the entire collection
Element URI `http://example.com/resources/item17`	**Retrieve** the representation of an element	**Replace** element **create** if it doesn't exist	generally not used	**Delete** the element

Example

examples/ajax/vue/playlist

Brr

Band name

Add Song

My favorite
This band

Second favorite
This other band

Example

examples/ajax/vue/playlist

```
@app.route("/playlist", methods=["GET"])
def getplace():
   ...

@app.route("/add", methods=["POST"])
def addSong():
    ...

@app.route("/remove", methods=["POST"])
def removeSong():
    ...
```

Not Rest.
If application grows, will be difficult to know what is removed.

Example

examples/ajax/vue/playlist-rest

```
@app.route("/songs", methods=["GET"])
def getplace():
   ...

@app.route("/songs", methods=["POST"])
def addSong():
    ...

@app.route("/songs/<int:id>", methods=["DELETE"])
def removeSong():
    ...
```

```
let response = await fetch("/song", {
    method: "DELETE",
    headers: {
        "Content-Type": "app
    },
    body: JSON.stringify({name: song.name, band: song.band}),
});
```

Can use **GET, POST, PUT, DELETE**

Exercises #1

github.com/dat310-2024/info/tree/master/
exercises/ajax/rest

Error handling and Validation

Example

examples/ajax/vue/playlist-error

Server side:

- JSON decoding may fail:

```
try:
    song = json.loads(request.data)
except json.decoder.JSONDecodeError as err:
    print("Decoding error: ", err)
    abort(400, "unable to decode JSON")
```

- Fields may be missing

```
name = song.get("name")
band = song.get("band")
if not name or not band:
    abort(400, "name or band missing")
```

- Return error to client

```
@app.errorhandler(400)
def request_error(e):
    response = e.get_response()
    # replace the body with JSON
    response.data = json.dumps({
        "code": e.code,
        "name": e.name,
        "description": e.description,
    })
    response.content_type = "application/
json"
    return response
```

Example

examples/ajax/vue/playlist-error

Client side:

- Fetch may fail on connection error

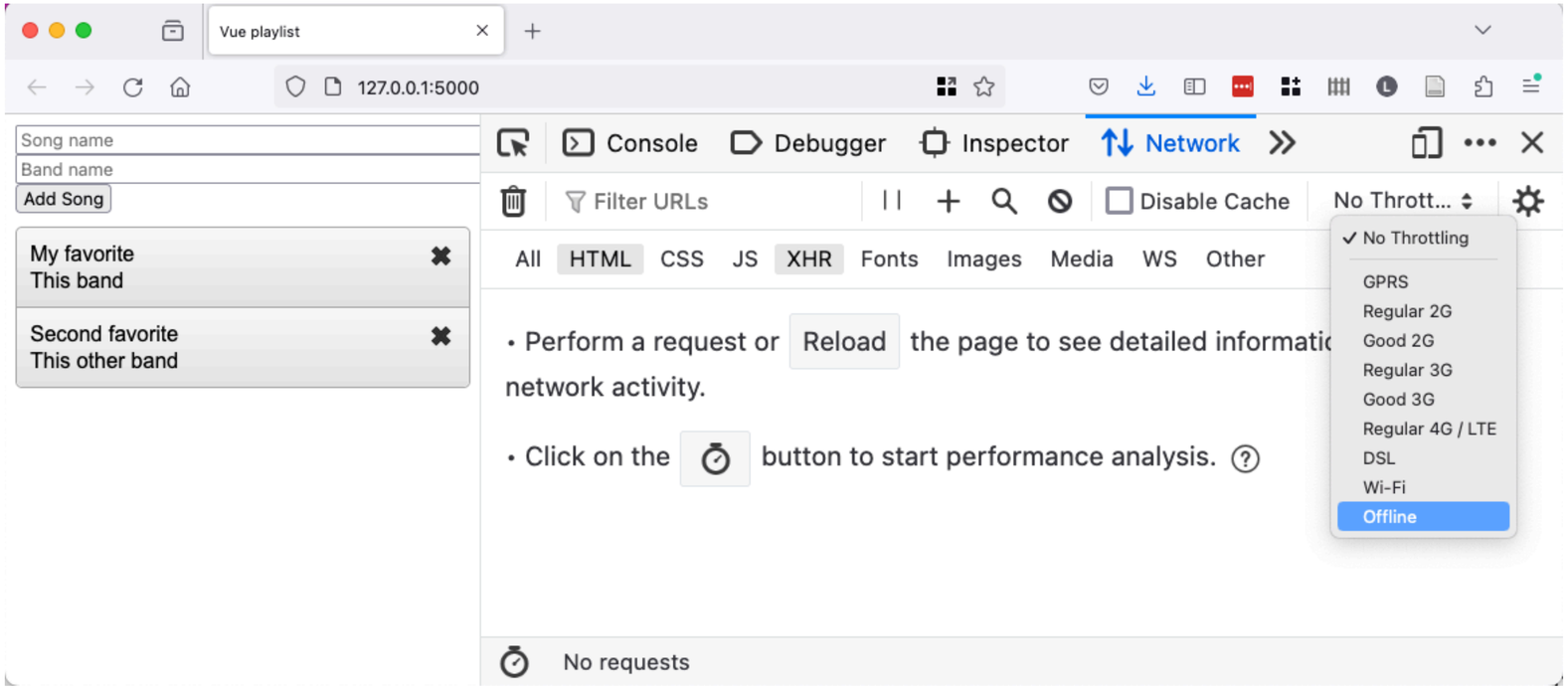
```
try {
    let response = await fetch("/songs");
} catch (e) {
    this.error = "Connection error: " + e
}
```

- Handle error status

```
let result = await response.json()
if (response.status == 200){
    this.playlist = result;
}
else {
    this.error = "Request error: " +
        response.status + " " +
        response.statusText;
```

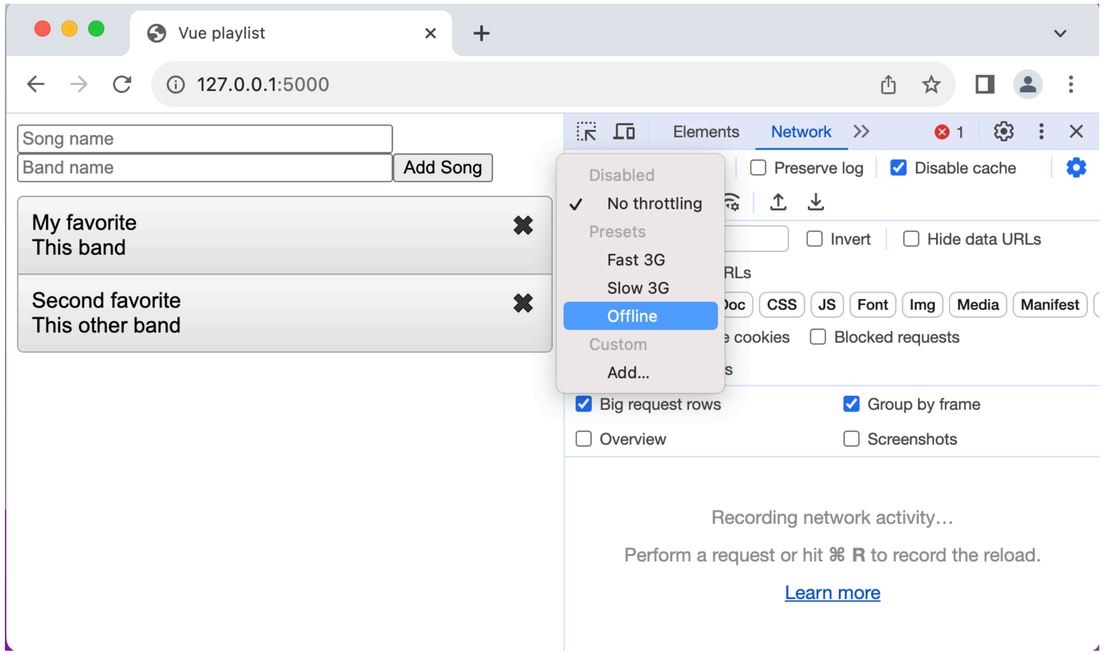
Connection error

- Test connection error handling by offline mode in dev tools



Connection error

- Test connection error handling by offline mode in dev tools

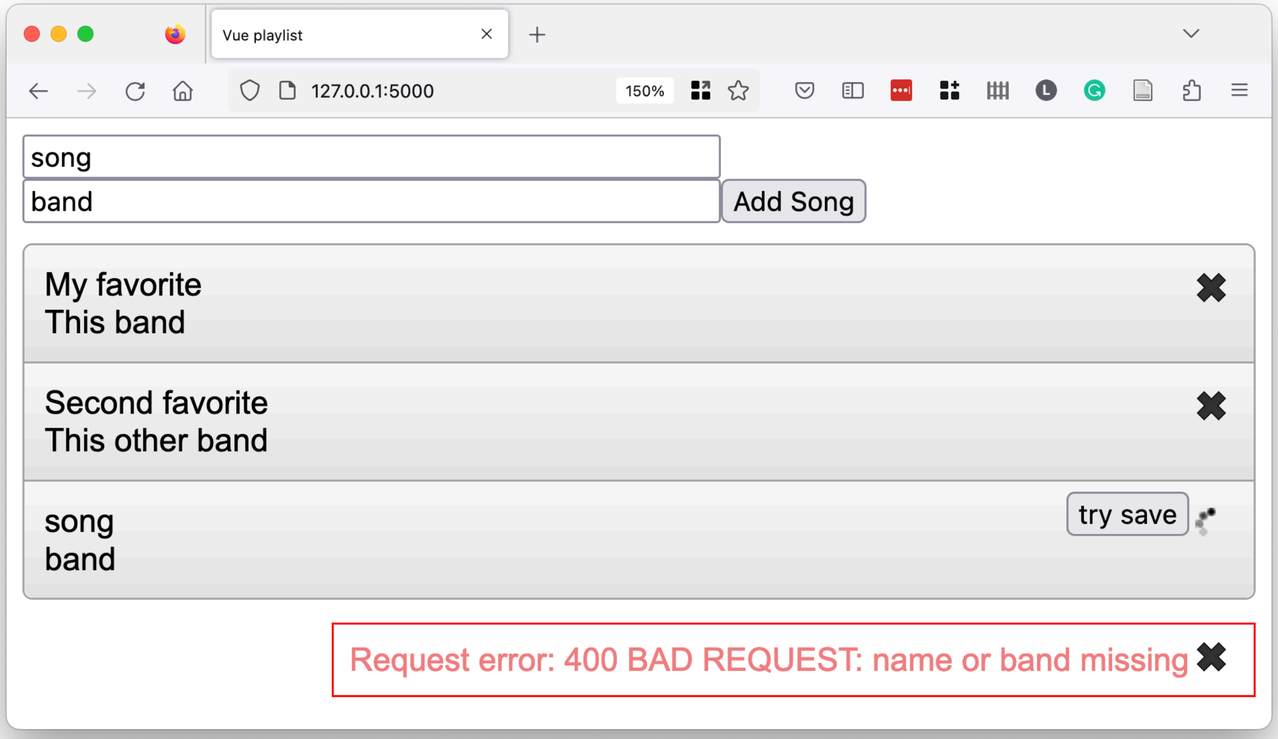


Example

examples/ajax/vue/playlist-error

Client side:

- Inform user on errors



Example

examples/ajax/vue/playlist-error

Server side:

- Check for missing fields

```
name = song.get("name")
band = song.get("band")
if not name or not band:
    abort(400, "name or band missing")
```

Client side:

- Prevent missing fields

```
addSong: async function(song) {
    if (!song.name || !song.band){
        this.error = “Enter name and band."
        return;
     }
```

Example

examples/ajax/vue/playlist-error

Server side:

- Check for missing fields

```
name = song.get("name")
band = song.get("band")
if not name or not band:
    abort(400, "name or band missing")
```

- Ensures data have good quality

Client side:

- Prevent missing fields

```
addSong: async function(song) {
    if (!song.name || !song.band){
        this.error = “Enter name and band."
        return;
     }
```

- Helps user fill the form

Client side validation and **server side validation**
Different purpose, **both needed**.

Validation

Server side:

- Ensures data have good quality
- Cannot be circumvented by users/ attackers

Client side:

- Helps user fill the form
- Can be circumvented by users/ attackers

References

- REST API tutorial
  - http://www.restapitutorial.com/