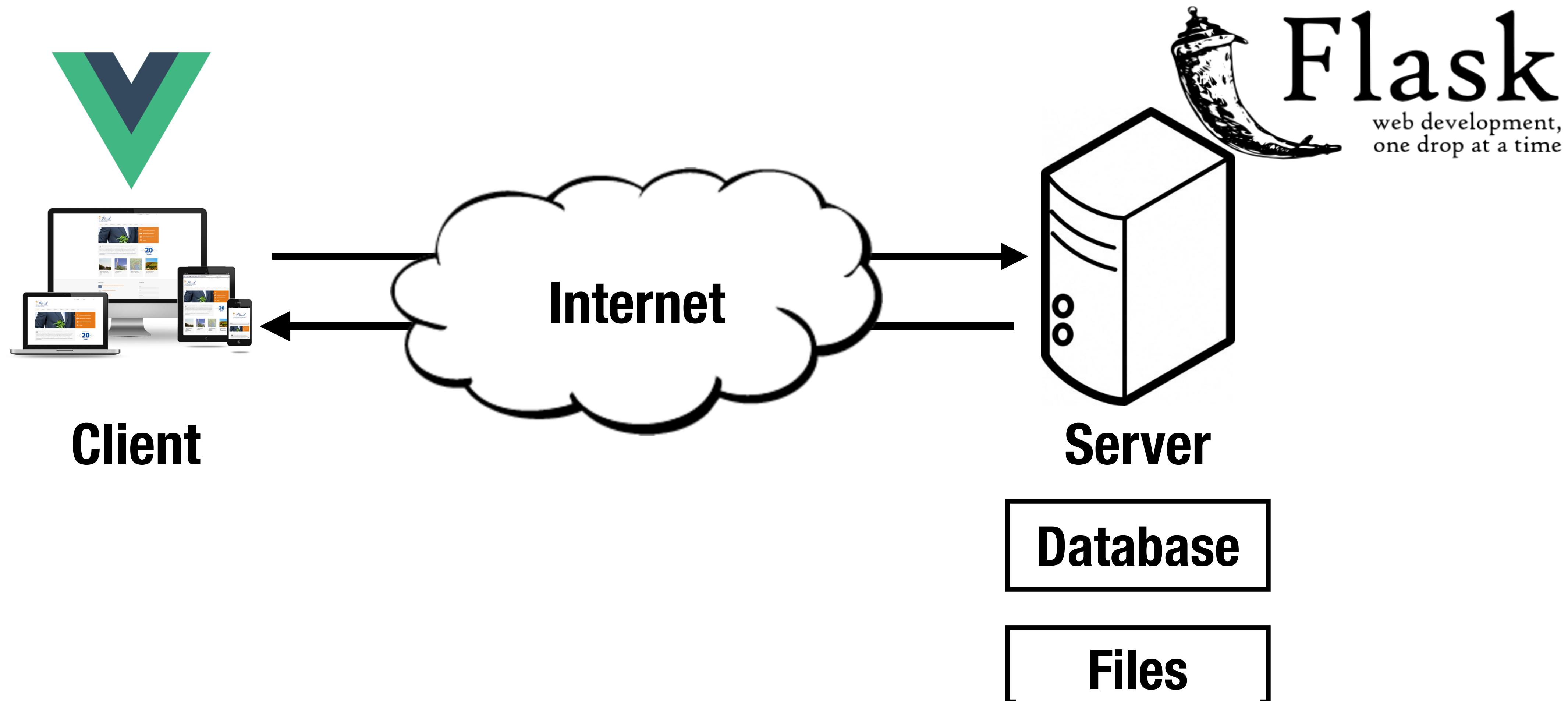


# Web Programming

# **Decoupled REST API**

# 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
<b>Collection URI</b> <code>http://example.com/resources</code>	<b>List</b> elements	<b>Replace</b> the entire collection	<b>Create</b> a new element in the collection	<b>Delete</b> the entire collection
<b>Element URI</b> <code>http://example.com/resources/item17</code>	<b>Retrieve</b> the representation of an element	<b>Replace</b> element <b>create</b> if it doesn't exist	generally not used	<b>Delete</b> the element

# Example

🔄 Examples/ajax/rest/playlist-norest

Add Song

My favorite  
This band



Second favorite  
This other band



# Example

## 🔄 Examples/ajax/rest/playlist-norest

```
@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/rest/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/dat310-2025/info/tree/master/](https://github.com/dat310-2025/info/tree/master/exercises/ajax/rest)  
**exercises/ajax/rest**

# **Error handling and Validation**

# Example

🕒 Examples/ajax/rest/playlist-errors

## 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/rest/playlist-norest](#)

## Client side:

- Fetch may fail on connection error

```
try {  
  let response = await fetch("/songs");  
} catch (e) {  
  let msg = 'Error fetching playlist: ' + error  
  showError(msg);  
}
```

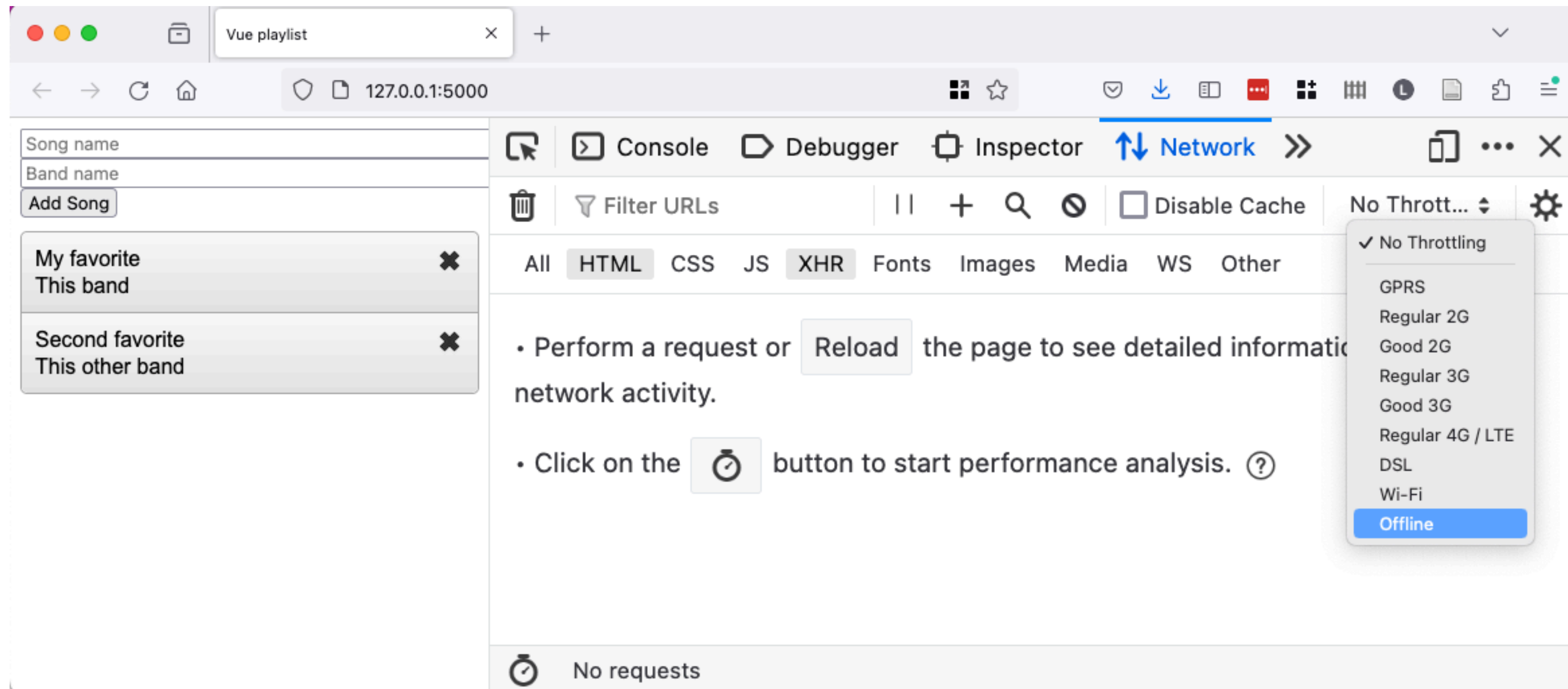
- Handle error status

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



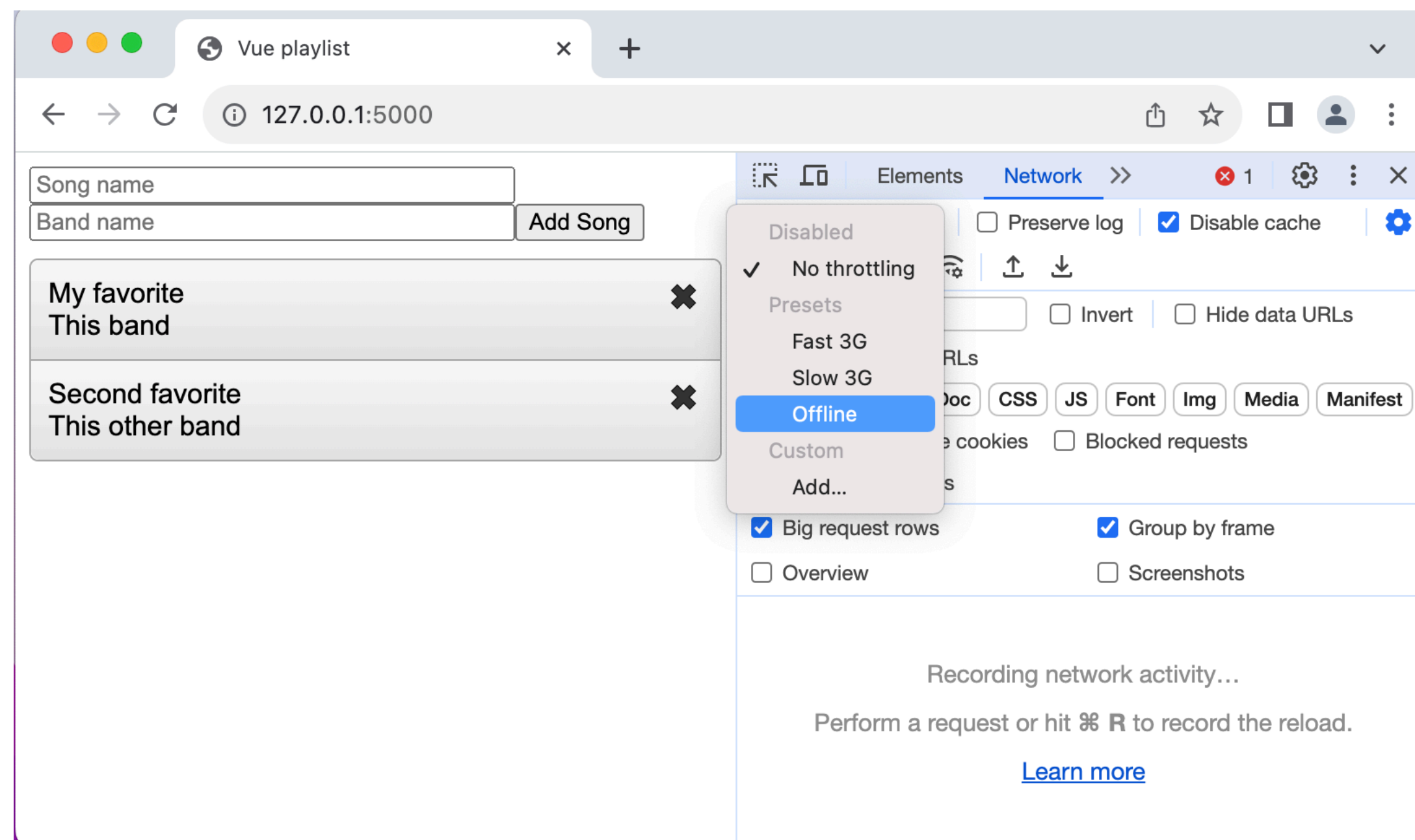
# Connection error

- 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

The screenshot shows a web browser window titled "Vue playlist" at the address "127.0.0.1:5000". The application has two input fields, "song" and "band", and an "Add Song" button. Below these are three playlist items, each with a close button (X):

- My favorite  
This band
- Second favorite  
This other band
- song  
band

At the bottom of the third item is a "try save" button. A red error message box at the bottom of the browser window reads: "Request error: 400 BAD REQUEST: name or band missing".

# 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

# Exercises #2



[github.com/dat310-2025/info/tree/master/](https://github.com/dat310-2025/info/tree/master/exercises/ajax/rest)  
**exercises/ajax/rest**

# References

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