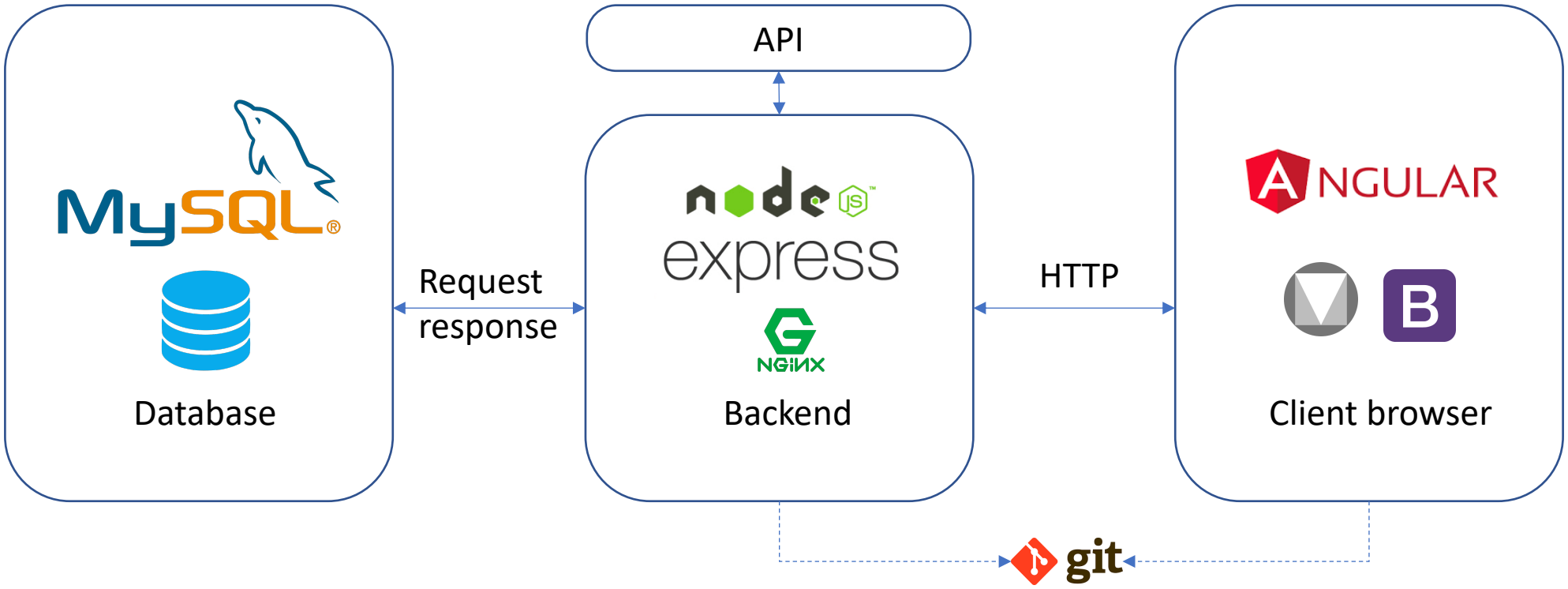
Cours transverse I

Houssem Ben Mahfoudh

Semestre de printemps 2020

Architecture



Full Stack Application

Web Service

WS

- Add in app.module.ts :

import { HttpClientModule, HttpInterceptor, HttpRequest, HttpHandler, HTTP_INTERCEPTORS } from '@angular/common/http';

- Add in imports[]:

HttpClientModule

WS

- Add in your component:

```
import { HttpClient } from '@angular/common/http';

constructor(private httpClient: HttpClient){}

test() {
this.httpClient.get("http://localhost:3000/", { responseType: 'text' }).
subscribe(res => {console.log(res);})
}
```

Nodejs - Access control

```
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
  next();
});
```

WebSocket

Socket.io

- Javascript library for realtime web application based on websocket protocol.

- It enables bi-directional communication between clients and server.

Server side

Npm install socket.io

```
const socketIO = require('socket.io')
const server = http.createServer(app)
const io = socketIO(server)
io.on('connection', socket => {
console.log('New client connected')
 socket.on('new-message', (message) => {
         io.emit('message', {text: message});
     console.log(message);
   });
socket.on('disconnect', () => {
console.log('user disconnected') })
});
server.listen(3000, function () {
       console.log('app listening on port 3000! ')
})
```

Client side

Npm install socket.io-client
npm install rxjs-compat

Create a chat.service.ts

Add to app.module.ts :
import { ChatService } from './chat.service';
import { FormsModule } from '@angular/forms';
Add to imports:[FormsModule]
Add to providers:[ChatService]

Chat service

```
import * as io from 'socket.io-client';
import { Observable } from 'rxjs/Observable';
export class ChatService {
private url = 'http://localhost:3000';
private socket;
constructor() {this.socket = io(this.url);}
public sendMessage(message) {this.socket.emit('new-message', message);}
public getMessages() {let observable = new Observable(observer => { this.socket = io(this.url);
this.socket.on('message', (data) => {observer.next(data.text); });
return () => {this.socket.disconnect();}; })
return observable;}
}
```

Component.ts

Add to the app.component.ts

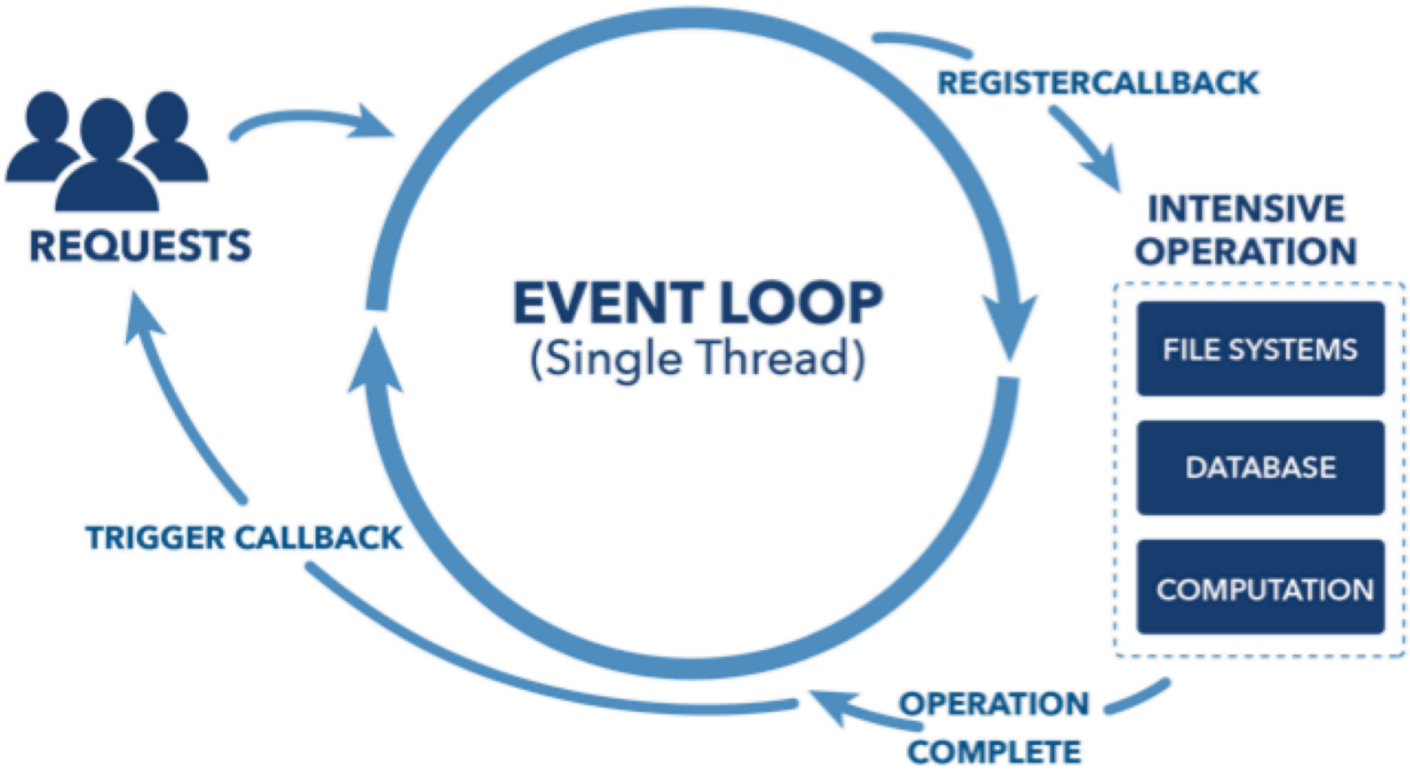
```
import { ChatService } from './chat.service’;

message: string;
messages: string[] = [];
constructor(private chatService: ChatService,private httpClient: HttpClient) { }
ngOnInit() {
this.chatService .getMessages()
.subscribe((message: string) => { this.messages.push(message);});}
sendMessage() { this.chatService.sendMessage(this.message);
this.message = '';}
```

Component.html

```
<input [(ngModel)]="message" />
<button (click)="sendMessage()">Send</button>
<div *ngFor="let message of messages">
{{message}}
</div>
```

Call back

NodeJs



Asynchronous non blocking I/O event driven architecture

https://webapplog.com/event-loop/

Callback

- A callback is a function that is called by another function which received it as an argument.

```
function greeting(name) {
  alert('Bonjour ' + name);
}

function processUserInput(callback) {
  var name = prompt('Entrez votre nom.');
  callback(name);
}

processUserInput(greeting);
```

https://developer.mozilla.org/fr/docs/Glossaire/Fonction_de_rappel

Real world example

Read file[1]

```
fs.readFile(funFileName, function(err, file) {
    if(err)
        handleError(err);
    console.log("file: ", file)
})
```

HTTP request[2]

```
T.get('search/tweets', params, function(err, data, response) {
  if(!err){
    // This is where the magic will happen
  } else {
    console.log(err);
  }
})
```

[1] https://medium.com/javascript-in-plain-english/callbacks-in-node-js-how-why-when-ac293f0403ca
[2] https://codeburst.io/javascript-what-the-heck-is-a-callback-aba4da2deced

Callback Hell

```
const createUser = function(username, password, picture, callback) {
   dataBase.createUser(username, password, (error, userID) => {
       if (error) {
           callback(error)
       } else {
           cloudinary.uploadPicture(picture, (error, path) => {
               if (error) {
                   callback(error)
               } else {
                   dataBase.updatePicture(userID, path, (error) => {
                       if (error) {
                           callback(error);
                       } else {
                           callback(null);
                       }
                   })
               }
           })
       }
   })
};
```

https://medium.com/javascript-in-plain-english/callbacks-in-node-js-how-why-when-ac293f0403ca

Callback Hell

```
request('http://www.somepage.com', function (firstError,
firstResponse, firstBody) {
    if(firstError){
        // Handle error.
    }
    else {
        request(`http://www.somepage.com/${firstBody.someValue}`,
function (secondError, secondResponse, secondBody) {
            if(secondError){
                // Handle error.
            }
            else {
                // Use secondBody for something
            }
        });
    }
});
```

https://medium.com/codebuddies/getting-to-know-asynchronous-javascript-callbacks-promises-and-async-await-17e0673281ee

Promise

- A promise is an object that wraps an asynchronous operation and notifies when it’s done.

```
var promise1 = new Promise(function(resolve, reject) {
  resolve('Success!');
});

promise1.then(function(value) {
  console.log(value);
  // expected output: "Success!"
});
```

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Promise/then

Promise

```
const loadImage = url => {
  return new Promise(function(resolve, reject) {

    //Open a new XHR
    var request = new XMLHttpRequest();
    request.open('GET', url);

    // When the request loads, check whether it was successful
    request.onload = function() {
      if (request.status === 200) {
        // If successful, resolve the promise
        resolve(request.response);
      } else {
        // Otherwise, reject the promise
        reject(Error('An error occurred while loading image. error code:' + request.statusText));
      }
    };

    request.send();
  });
```

```
const embedImage = url => {
  loadImage(url).then(function(result) {
      const img = new Image();
      var imageURL = window.URL.createObjectURL(result);
      img.src = imageURL;
      document.querySelector('body').appendChild(img);
    },
    function(err) {
      console.log(err);
    });
}
```

https://codeburst.io/javascript-promises-explained-with-simple-real-life-analogies-dd6908092138

Async/await

- Async and Await are extensions of promises
- Non blocking
- New way to deal with asynchronous function
- Chaining promises

Async/await

```
function doubleAfter2Seconds(x) {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve(x * 2);
    }, 2000);
  });
}

function addPromise(x){
  return new Promise(resolve => {
    doubleAfter2Seconds(10).then((a) => {
      doubleAfter2Seconds(20).then((b) => {
        doubleAfter2Seconds(30).then((c) => {
          resolve(x + a + b + c);
        })
      })
    })
  });
}

addPromise(10).then((sum) => {
  console.log(sum);
});
```

vs

```
function doubleAfter2Seconds(x) {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve(x * 2);
    }, 2000);
  });
}

async function addAsync(x) {
  const a = await doubleAfter2Seconds(10);
  const b = await doubleAfter2Seconds(20);
  const c = await doubleAfter2Seconds(30);
  return x + a + b + c;
}


addAsync(10).then((sum) => {
  console.log(sum);
});
```

https://codeburst.io/javascript-es-2017-learn-async-await-by-example-48acc58bad65