

# SVELTE

Étude de Svelte (Framework compilateur JS) au travers d'une application (Switter)

Vidéo :

En utilisant Svelte on va créer une application nommée Switter

Cette application nous permettra de :

- Saisir des messages
- Devenir le nom de l'auteur
- Restriction sur le nombre de caractère à saisir dans la zone du message
- Un bouton SEND pour envoyer le message qui s'affiche ensuite avec toutes les informations sur le message
- Définir une valeur par défaut dans le cas ou certains champs ne sont pas renseigné
- Un bouton pour afficher/masquer la partie pour écrire un commentaire

Avec Svelte on peut faire communiquer les composants parents/enfants, apprendre à gérer des événements...

## INSTALLATION DE SVELTE et CREATION DU PROJET

Sur le site de [Svelte](#), on peut créer un template svelte en ligne de commande depuis NodeJS Déjà préalablement téléchargé.

```
PS C:\Users\59069> npx degit sveltejs/template switter
npx: installed 1 in 4.863s
> cloned sveltejs/template#master to switter
```

En utilisant VS Code, depuis le fichier **package.json** dans le dossier switter correspondant à l'application :

On ouvre un terminal puis on tape la commande suivante : **npm i** (i pour installer)

```
PS C:\Users\59069\switter> npm i
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@~2.3.1 (node_modules\rollup\node_modules\fsevents
):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin",
"arch":"any"} (current: {"os":"win32","arch":"x64"})

added 96 packages from 128 contributors and audited 97 packages in 12.266s

6 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

L'installation terminer, on peut lancer l'application générer

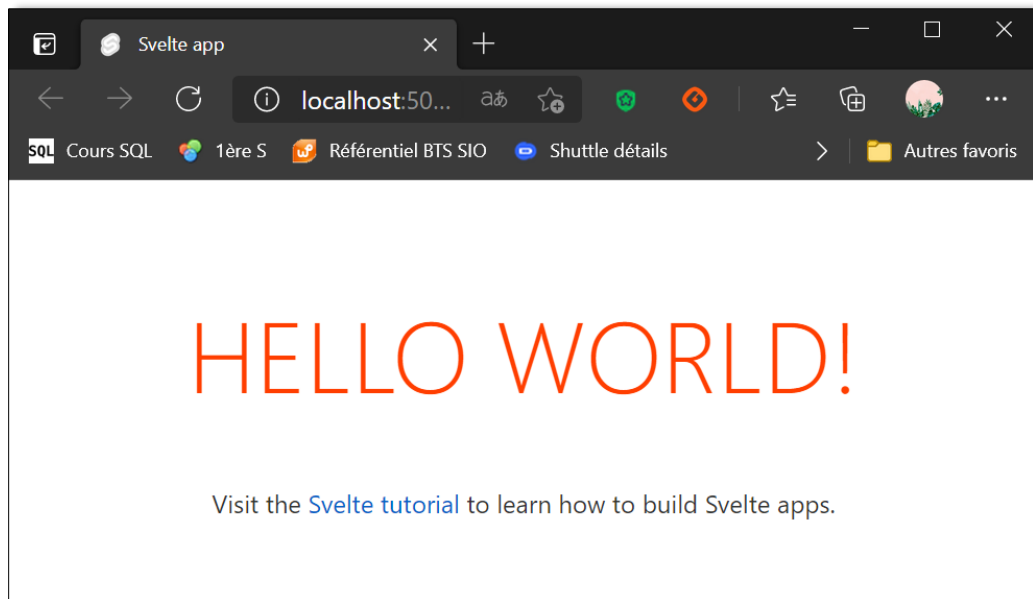
Commande : **npm run dev**

Your application is ready~! 🚀

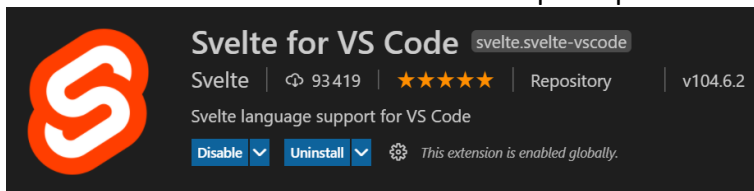
- Local: http://localhost:5000
- Network: Add `--host` to expose

# SVELTE

En cliquant sur le lien on obtient ceci :



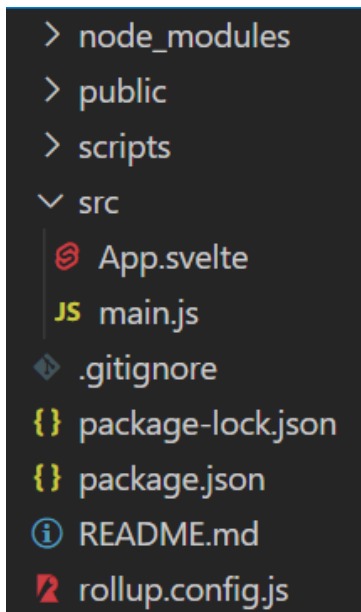
Installation d'une extension sur VS Code pour que svelte soit reconnue par notre IDE



Structure de l'application

On va principalement travailler dans le dossier **src**

- Des fichiers avec une extension propre à Svelte : « **.svelte** »



# SVELTE

## DEBUT DE L'APPLICATION

Dans le fichier **main.js**, on change la variable d'affichage par le nom de l'application « Switter » :

```
import App from "./App.svelte";

const app = new App({
  target: document.body,
  props: {
    name: "Switter",
  },
});

export default app;
```

Puis dans le fichier **App.vue** :

- On rajoute un textarea (zone de texte, commentaire)

```
<textarea cols="50" rows="5"></textarea>
```

Dans la balise <main>, on retire le « Hello » pour avoir seulement le nom de l'application définie précédemment dans le fichier js

**Avant :**

```
<h1>Hello {name}!</h1>
```

**Après :**

```
<h1>{name}</h1>
```

**>> Résultat :**



# SVELTE

Ensuite on veut écouter les saisies qui se font dans le <textarea> pour se faire il faut écrire :

```
<textarea cols="50" rows="5" on:input={updateMessage}></textarea>
```

Cela permet d'écouter les évènements **input** qui va exécuter une fonction nommée **updateMessage**

↓

Et cette fonction permet de montrer les évènements inscrits dans le <textarea> dans la console

```
<script>
  export let name;

  function updateMessage(event){
    console.log(event);
  }
</script>
```

>> **Résultat :**

Chaque fois que l'on tape :

Hello

Cela s'affiche dans la console :

```
App.svelte:5
▶ InputEvent {isTrusted: true, data: "H", isComposing: false, inputType: "insertText", dataTransfer: null, ...}
App.svelte:5
▶ InputEvent {isTrusted: true, data: "e", isComposing: false, inputType: "insertText", dataTransfer: null, ...}
App.svelte:5
▶ InputEvent {isTrusted: true, data: "l", isComposing: false, inputType: "insertText", dataTransfer: null, ...}
App.svelte:5
▶ InputEvent {isTrusted: true, data: "l", isComposing: false, inputType: "insertText", dataTransfer: null, ...}
App.svelte:5
▶ InputEvent {isTrusted: true, data: "o", isComposing: false, inputType: "insertText", dataTransfer: null, ...}
```

Dans la même fonction :

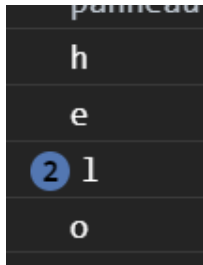
```
console.log(event.data);
```

>> **Résultat :**

hello

# SVELTE

Dans la console :



```
h  
e  
2 l  
o
```

On a directement les lettres qui apparaissent.

Dans la fonction, maintenant, on change l'affichage :

```
console.log(event.target.value);
```

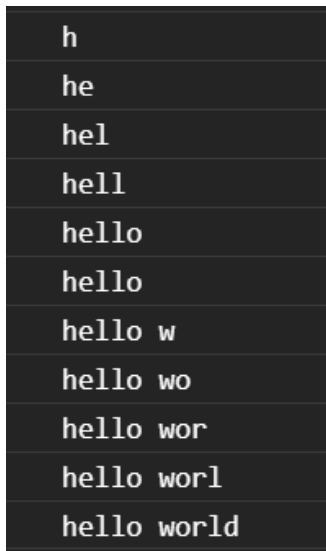
*event.target.value* : Permet d'afficher dans la console la valeur final au fur et à mesure qu'on le tape

>> Résultat :



```
hello world
```

Dans la console :



```
h  
he  
hel  
hell  
hello  
hello  
hello w  
hello wo  
hello wor  
hello worl  
hello world
```

# SVELTE

Maintenant, on change le fonctionnement de la fonction : ce qui est saisi on le récupérer dans une variable pour afficher le mot le final sur la page

```
<script>
  export let name;
  let message = "";

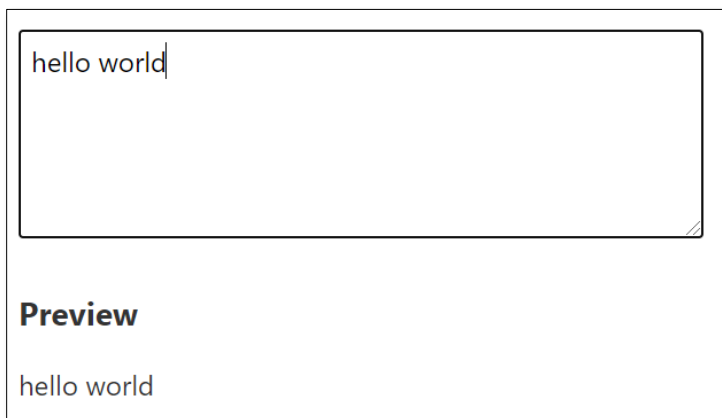
  function updateMessage(event){
    console.log(event.target.value);
    message = event.targer.value;
  }
</script>
```

Cette variable *message* va récupérer la valeur de *event.target.value*

Pour l'affichage on crée une `<div>` où on affiche le message :

```
<div>
  <h3>Preview</h3>
  {message}
</div>
```

>> Résultat :



The screenshot shows a web application interface. At the top, there is a large text input field containing the text "hello world". Below this input field, there is a section titled "Preview" in bold. Under the "Preview" title, the text "hello world" is displayed, which is a direct reflection of the text entered in the input field above.

On a le message complet dans la partie **Preview**.

Voilà comment on fait communiqué la partie HTML et JavaScript avec un composant Svelte, en utilisant des évènements et des fonctions selon le besoin.

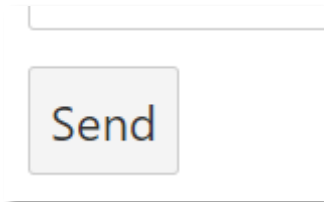
# SVELTE

Ajout d'un bouton

Code :

```
<button>Send</button>
```

>> Résultat



On va rajouter une évènement click sur le bouton qui fera appel à une fonction

```
<button on:click={saveMessage}>Send</button>
```

Puis on crée la fonction *saveMessage*

```
function saveMessage(){
  const newMessage = {
    id: Date.now(),
    text: message,
    author: 'Yanissa'
  };
  messages = [newMessage, ...messages];
  console.log('messages', messages);
}
```

Dans cette fonction crée un nouvel objet *newMessage* qui va contenir un id = la date du jour avec la fonction *Date.now()*

Le texte qui sera le message que l'on a initialisé précédemment

Et l'auteur : pour le moment j'ai mis mon prénom

Puis on a créé un tableau

```
let messages = [];
```

Ce tableau va afficher le nouveau message ainsi que les autres messages mais sous forme d'objets

Pour l'affichage :

```
<div>
  <h2>Messages</h2>
  {messages}
</div>
```

# SVELTE

>> Résultat

Après avoir tapé le message puis cliquer sur le bouton

Hello world

Dans la partie Message on a ceci qui apparaît :

Messages

[object Object]

Dans la console : on voit bien le message crée

```
messages ▾ [{...}] ⓘ  
  ▾ 0:  
    author: "Yanissa"  
    id: 1615672443798  
    text: "Hello world"  
    ▶ __proto__: Object  
  length: 1  
  ▶ __proto__: Array(0)
```

On verra comment afficher les objets dans un tableau

Itération :

Il faut utiliser *each* ... as ...

En quelque sorte cela veut dire : pour chaque message et après le as cela représente l'instance couramment itérée

Donc cela va afficher les messages les uns après les autres qui sont dans le tableau messages

Les instructions en Svelte comme le *each* commence par un # et se termine par un /

```
<h2>Messages</h2>  
{#each messages as message}  
  <div></div>  
{/each}
```

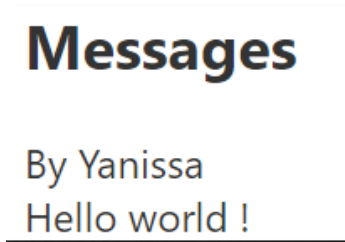
Dans cette instruction, on va afficher l'auteur du message et son contenu :

```
<div>  
  <h2>Messages</h2>  
  {#each messages as message}  
    <div>By {message.author}</div>  
    <div>{message.text}</div>  
  {/each}  
</div>
```



# SVELTE

>> Résultat :

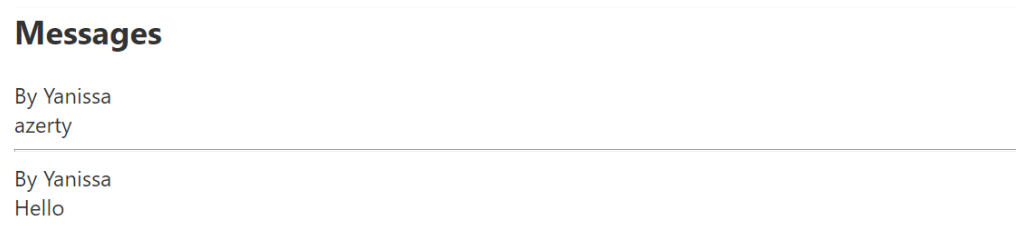


On a ajouté une ligne de séparation avec la balise <hr>

Code :

```
<div>{message}  
<hr>
```

>> Résultat :



Donc pour itérer les éléments d'une collection, on utilise l'instruction *each*

Le binding

```
<textarea cols="50" rows="5" bind:value={message}></textarea>  
<br>
```

Permet de lier un élément JS au html on utilise le binding

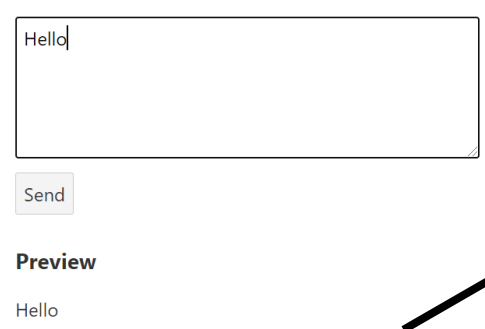
On n'a plus besoin d'utiliser une fonction

Vider le textarea après avoir cliqué sur le bouton :

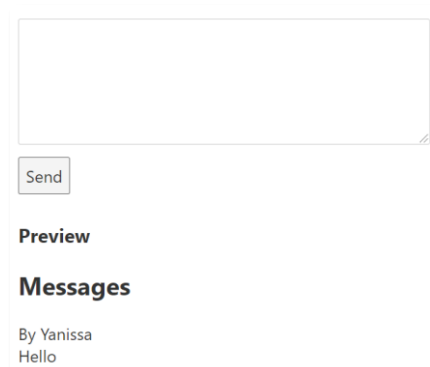
On va réinitialiser la valeur de message

```
messages = [newMessage, ...messages];  
message = "";
```

>> Résultat :



Après le clic sur le bouton **Send**



# SVELTE

## Création de component en Svelte

Un component qui va afficher le textarea et le bouton et gère l'envoi de nouveau message :  
Création du fichier Message.svelte et dans ce fichier on retrouve les éléments pour écrire le message et le bouton pour l'envoyer

```
<script>
  let message = "";
  function saveMessage(){
    const newMessage = {
      id: Date.now(),
      text: message,
      author: 'Yanissa'
    };
    messages = [newMessage, ...messages];
    message = "";
  }
</script>

<style></style>

<textarea cols="50" rows="5" bind:value={message}></textarea>
<br>
<button on:click={saveMessage}>Send</button>
```

Dans le fichier App.svelte comme message n'est plus définie on a cette erreur :

```
Uncaught ReferenceError: message is not defined    App.svelte:38
    at Object.create [as c] (App.svelte:38)
    at init (index.mjs:1499)
    at new App (App.svelte:4)
    at main.js:3
    at main.js:8
```

De ce fait on va importer le fichier Message.svelte crée précédemment :

```
import Message from "../Message.svelte";
```

```
<Message />
```

>> Résultat :

En cliquant sur Send après avoir écrit un message on obtient :

```
Uncaught ReferenceError: messages is not defined    Message.svelte:10
    at HTMLButtonElement.saveMessage (Message.svelte:10)
```

Message est devenu l'enfant de App.svelte et nous verrons comment faire communiquer enfant et parent

On vérifie qu'on récupère bien la variable *newMessage* par sécurité

# SVELTE

Code :

```
//messages = [newMessage, ...messages];  
console.log('newMessage', newMessage);  
message = "";
```

>> Résultat :

```
newMessage                                     Message.svelte:11  
▶ {id: 1615685157620, text: "hello", author: "Yanissa"}
```

## PASSER DE PROPS A COMPOSANT ENFANT :

Le problème à ce stade est qu'on n'arrive pas à faire communiquer le message jusqu'au parent

Solutions

>> Du parent vers l'enfant :

Par exemple : si l'on veut communiquer à Message le nom de l'auteur

Pour faire communiquer le parent vers l'enfant, on utilise des props

Et pour utiliser un props

On va directement l'utiliser dans le composant que l'on a envie de passer

Ici on déclare que *author* = «Bob» dans l'élément parent

```
<Message author="Bob"/>
```

Dans l'élément enfant on va déclarer la variable *author* et pour qu'elle soit accessible depuis l'extérieur on ajoute comme préfixe *export*

```
export let author;
```

Grace à «export», maintenant *author* = « Bob »

Dans la fonction, on remplace l'*author* par la variable que l'on vient de créer

Avant :

```
author: 'Yanissa'
```

Après :

```
author: author
```

>> Résultat dans la console après avoir envoyer un message :

```
newMessage                                     Message.svelte:13  
▶ {id: 1615937790282, text: "Hello", author: "Bob"}
```

# SVELTE

>> De l'enfant vers le parent :

Il faut utiliser des *event*

Création de la fonction `addMessage` et son event :

```
function addMessage(event){  
  console.log(event);  
}
```

Appel de l'event :

```
<Message author="Bob" on:message={addMessage}/>
```

Maintenant on va dispatcher ce *custom event* mis à disposition par Svelte grâce à une méthode :

```
import { createEventDispatcher } from "svelte";
```

On fait ensuite appel à cette méthode dans une variable qui va nous permettre de créer des custom event:

```
const dispatch = createEventDispatcher();
```

Et dans notre fonction `saveMessage`, on utilise cette fonction créée précédemment pour les messages si bien que lorsque l'on va cliquer sur le bouton *Send*, cela va appeler la fonction `saveMessage` qui crée notre nouveau message et ensuite on dispatch un custom event :

```
function saveMessage(){  
  const newMessage = {  
    id: Date.now(),  
    text: message,  
    author: author  
  };  
  //messages = [newMessage, ...messages];  
  console.log('newMessage', newMessage);  
  dispatch('message', newMessage);  
  message = "";  
}
```

>> Test :

```
App.svelte:8  
▶ CustomEvent {isTrusted: false, detail: {...}, type: "message", target:  
  null, currentTarget: null, ...}
```

On a bien un custom Event avec les détails de notre `newMessage` (objet)

```
▶ detail: {id: 1615950486549, text: "Hello", author: "Bob"}
```

# SVELTE

On va récupérer les informations  
Puis afficher le message :

```
function addMessage(event){  
  console.log(event.detail);  
  messages = [event.detail, ...messages];  
}
```

On rajoute cet event dans le tableau message pour qu'il soit afficher

>> Résultat

Le message s'affiche bien :

## Messages

By Bob  
Hello world

Dans la console :

```
newMessage                                     Message.svelte:16  
▶ {id: 1615976432046, text: "Hello world", author: "Bob"}  
App.svelte:8  
▶ {id: 1615976432046, text: "Hello world", author: "Bob"}
```

### Rappel :

Ici, nous avons un component qui reçoit les saisies de l'utilisateur (textarea + bouton)  
Quand on clique sur le bouton cela appelle la fonction *saveMessage* qui crée un nouveau message et elle nous permet aussi de communiquer avec le parent grâce au dispatch custom event

### METTRE LE NOM DE L'AUTEUR EN GRAS :

Code dans la balise <style>

```
.author {  
  font-weight: bold;  
}
```

Dans l'HTML :

```
<div class="author">By {message.author}</div>
```

>> Résultat :

**By Bob**  
azerty

# SVELTE

Donnez la possibilité à l'utilisateur la possibilité de saisir lui mm qui il est :

## Dans l'élément enfant :

Créer un *input* de type *text* où l'on va binder la variable *author*

```
<input type="text" bind:value={author}>
```

On va déclarer la variable *author*

### Avant :

```
export let author;
```

### Après :

```
let author = "";
```

Plus besoin de l'exporter et on l'initialise comme champ vide

## Dans l'élément parent :

### Avant :

```
<Message author="Bob" on:message={addMessage}/>
```

### Après :

```
<Message on:message={addMessage}/>
```

Plus besoin de définir *author* ici

>> Résultat :

**By Quetsiah**

Yo

**By Yanissa**

Hello

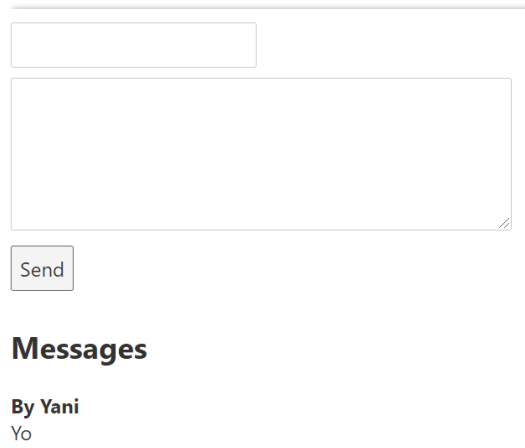
# SVELTE

Pour vider le champ pour saisir l'*author* après le clic sur le bouton :

Comme pour message, dans la fonction *saveMessage*, après avoir dispatcher le custom event on vide le champ texte et maintenant le champ pour l'auteur comme ceci :

```
author = "";
```

>> Résultat :



The screenshot shows a web interface with a text input field at the top, followed by a large text area. Below the text area is a 'Send' button. Underneath the button is a section titled 'Messages'. In this section, there is a message that says 'By Yani' followed by 'Yo' on the next line.

Ajout de la date du message :

On va utiliser : **Intl.DateTimeFormat**

Fichier *Message.svelte* :

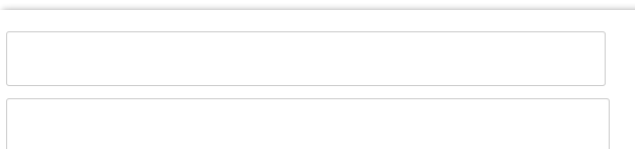
On rajoute une propriété date qui donnera une nouvelle instance de date donc *new Date()*

```
function saveMessage(){
  const newMessage = {
    id: Date.now(),
    text: message,
    author: author,
    date : new Date()
```

```
<input class="text" type="text" bind:value={author}> <br>
<style>
  .text {
    width:395px;
  }
</style>
```

↓

>> Résultat :



The screenshot shows a web interface with two text input fields stacked vertically. The top field is empty, and the bottom field is also empty.

# SVELTE

Fichier *App.svelte* :

On rajoute dans la boucle l'affichage de la date

```
<div class="author">By {message.author} on {message.date}</div>
```

>> Résultat :

**By Yanissa on Fri Mar 19 2021 09:37:38 GMT-0400 (heure normale de l'Atlantique)**  
Hello

La date s'affiche comme ceci

On va modifier le format avec la fonction **Intl.DateTimeFormat**

On crée un objet options qui va répertorier tous les paramètres d'affichage de la date (si on veut la date en toutes lettres etc...) ceci grâce à la documentation :

Je n'ai pas fait comme dans la vidéo, mettre la date en anglais mais plutôt en français

```
const options = {  
  weekday : "long",  
  year : "numeric",  
  month : "long",  
  day : "numeric",  
  hour12 : false,  
  hour : "2-digit",  
  minute : "2-digit"  
};
```

Dans la variable `formatter` : on met un constructeur qui est fourni par **Intl.DateTimeFormat**

```
const formatter = new Intl.DateTimeFormat("fr-FR", options);
```

Et on l'utilise dans notre boucle :

```
<div class="author">By {message.author} on {formatter.format(message.date)} </div>
```

>> Résultat :

**By Yanissa on vendredi 19 mars 2021, 09:50**  
Hello



# SVELTE

## REACTIVER SVELTE

Pour tester cette réactivité, on va compter le nombre de caractères saisi par l'utilisateur

*Message.svelte*

On crée une variable nbCaractere

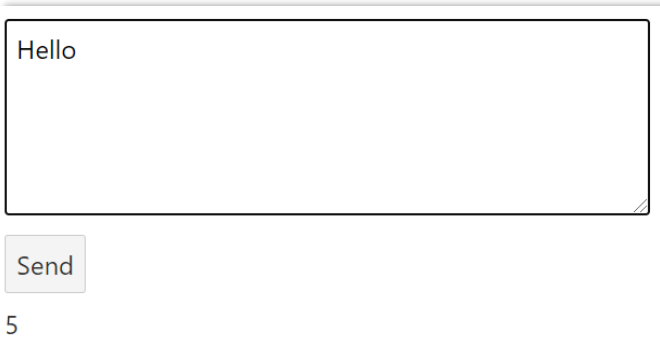
\$ : >> ce symbole veut dire que l'on veut recalculer la valeur de la variable nbCracters qui correspond à la longueur du message saisie

```
$: nbCharacters = message.length;
```

Maintenant on affiche le nombre de caractères

```
<span>{nbCharacters}</span>
```

>> Résultat :



The screenshot shows a simple web interface. At the top is a text input field containing the word 'Hello'. Below the input field is a button labeled 'Send'. Directly beneath the 'Send' button, the number '5' is displayed, representing the character count of the text in the input field.

Le nombre de caractère s'affiche en dessous du bouton.

Idée :

Au bout d'un nombre de caractères le bouton soit désactiver

On crée une variable *disable* qui prend en compte le faite que si le message est plus long que le nombre de caractère définie donc ici je mettrai 20 que *disable* soit vraie sinon renvoie faux

Code :

```
$: disabled = message.length > 20 ? true : false;
```

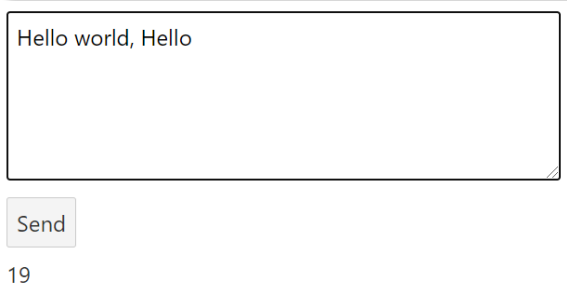
Et sur le bouton :

```
<button on:click={saveMessage} disabled={disabled}>Se
```

# SVELTE

>> Résultat :

**Avant** la limite de caractères :

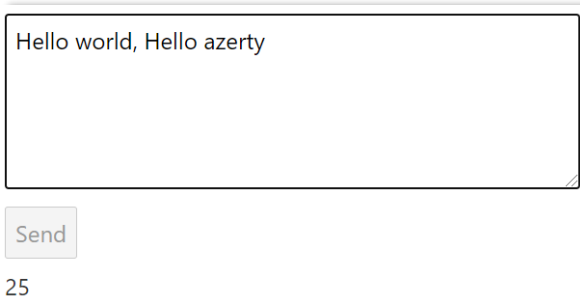


Hello world, Hello

Send

19

**Après** avoir dépassé le nb de caractères, le bouton se désactive



Hello world, Hello azerty

Send

25

## AJOUT D'UNE CLASSE CSS CONDITIONNELLEMENT

Quand l'utilisateur approche de la limite de caractère, pour ajouter un peu de style et donner des indications à l'utilisateur

Par exemple :

- **orange** si la limite de caractères est presque atteinte
- **rouge** quand la limite de caractères est dépassée

On va créer cette classe CSS nommé alerte

```
.alert{  
  color : ■ orangered;  
}
```

On a redéfini le nombre de caractères max dans une variable

Puis on a utilisé dans la classe conditionnelle

```
let maxLenght = 20  
$: nbCaracters = message.length;  
$: disabled = message.length > maxLenght ? true : false;
```

# SVELTE

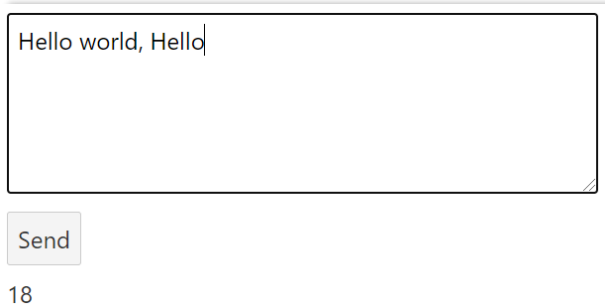
Puis dans l'affichage

On crée la classe cependant pour que celle-ci soit conditionnelle on l'écrit comme ci « `class : alert` » puis entre accolade on ajoute la condition que l'on souhaite (ici que la classe `alert` mette le nb caractère en orange dès que celui-ci est dépassé) :

```
<span class:alert={nbCaracters > maxLength}>{nbCaracters}</span>
```

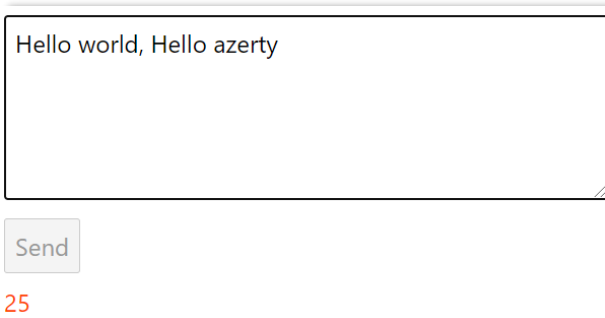
>> Résultat :

**Avant** la limite de caractères :



A screenshot of a web form. It features a text input field containing the text "Hello world, Hello". Below the input field is a "Send" button. At the bottom of the form, the number "18" is displayed in a standard black font, indicating the current character count.

**Après** avoir dépassé le nb de caractères



A screenshot of the same web form as before, but now the text input field contains "Hello world, Hello azerty". The "Send" button remains below it. At the bottom, the number "25" is displayed in an orange font, indicating that the character limit has been reached.

## COTE TEMPLATE

On peut ajouter du texte conditionnellement en ajoutant un if

Ici on a rajouté un message quand le nb caractères est dépassé

```
{#if disabled}  
  <span class="alert">(message trop long)</span>  
{/if}
```

# SVELTE

>> Résultat :

**Avant** la limite de caractères :

Hello World

Send

11

**Après** avoir dépassé le nb de caractères

Hello World, Hello azerty

Send

25 (message trop long)

## DEFINIR UNE VALEUR PAR DEFAULT

Définir une valeur par défaut si le champ auteur n'est pas rempli

>> Code

```
author: author || 'anonyme',
```

>> Résultat :

Quand on envoie le message sans auteur :

### Messages

**By anonyme on vendredi 19 mars 2021, 17:50**  
Hello

# SVELTE

Quand on envoie le message avec un nom d'auteur renseigné

## Messages

By Yanissa on vendredi 19 mars 2021, 17:52

Hello

By anonyme on vendredi 19 mars 2021, 17:50

Hello

## AFFICHER OU MASQUER UN COMPONENT

Masquer ou afficher la zone pour écrire le message

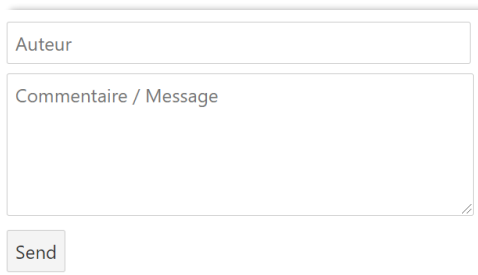
On va créer une variable *isVisible* = *true* (par défaut on affiche le component)

```
let isVisible = true;
```

On va tester si notre component Message *isVisible*

```
{#if isVisible}  
<Message on:message={addMessage}/>  
{/if}
```

>> Résultat :



Auteur

Commentaire / Message

Send

Cependant si *isVisible* = *false*, le component disparaît :

# SWITTER

Visit the [Svelte tutorial](#) to learn how to build Svelte apps.

## Messages

# SVELTE

On va maintenant mettre un bouton pour donner la possibilité à l'utilisateur d'afficher ou non la zone pour écrire un message

En créant une fonction :

```
function toggle(){  
  isVisible = !isVisible;  
}
```

Puis l'associer à un bouton

```
<button on:click={toggle}>Cacher</button>
```

>> Résultat :

Cacher

**Messages**

Quand on clique sur le bouton :

Cacher

Auteur

Commentaire / Message

Send

0

**Messages**

# SVELTE

On va changer le bouton

**Cacher** pour cacher la zone de texte et une fois la zone cacher le bouton montre **Afficher** pour afficher cette zone

```
<button on:click={toggle}>{isVisible ? 'Cacher' : 'Afficher'}</button>
```

RESULTAT FINAL :

# SWITTER

Visit the [Svelte tutorial](#) to learn how to build Svelte apps.

Afficher

## Messages

Cacher

Yanissa

Salut

Send

5

## Messages

Quand le message est envoyé :

## Messages

**By Yanissa on vendredi 19 mars 2021, 18:14**

Salut

# SVELTE

## Conclusion :

Je trouve que Svelte est facile à comprendre et un outil assez maniable.  
Il ressemble un peu à VueJS mais comporte ces propres spécificités.