# miam | Web

Janvier–avril 2023

JUNIA
Grande école d'ingénieurs
HEI·ISEN·ISA

# Un petit recap

Cours 1
Je crée un fichier Html basic, un fichier js.
Je sais importer mon js dans mon html à l'aide d’une balise script
Je sais faire des fonctions en javascript
Mon js me permet de faire des actions et modifier de façon asynchrone le html pour exécuter des actions (changer une couleur)

Cours 2
Notion de DOM : une arborescence du site avec des balises / nodes.
C’est ce que j’ai fais dans mon propre html
J’apprend à parcourir le DOM pour récupérer des informations

Je vois les requêtes qui partent de mon front vers une api.
J’inspecte cette requête pour la reproduire
Je la reproduit à l'aide de XMLHttpRequest

Cours 3
Notion orientée Objet. On commence par voir comment ça fonctionne puis le sucre syntaxique de la class
Heritage
On change intègre les fonctions d’appel à l’api dans des classes bien propres avec de l'héritage. Je peux me re-servire de ces classes pour récupérer des annonces d’une autre catégorie par exemple avec très peu de code.

Cours 4
On se crée un serveur basic en node
On ordonne les route qu’il accepte
on traite un envoie d’annonce depuis notre script leboncoin et on enregistre la data dans un fichier à l'aide d’un service.

Next step
On se sert d’une BDD plutôt qu’un fichier
On Travail avec des frameworks qui font le travail plus proprement et plus facilement

On se fait un petit projet global avec un back qui peut récupérer des annonces et fournir la data à notre front qui permet d’afficher des annonces de plusieurs site.
On a réalisé un projet complet d'agrégateur d’annonces.

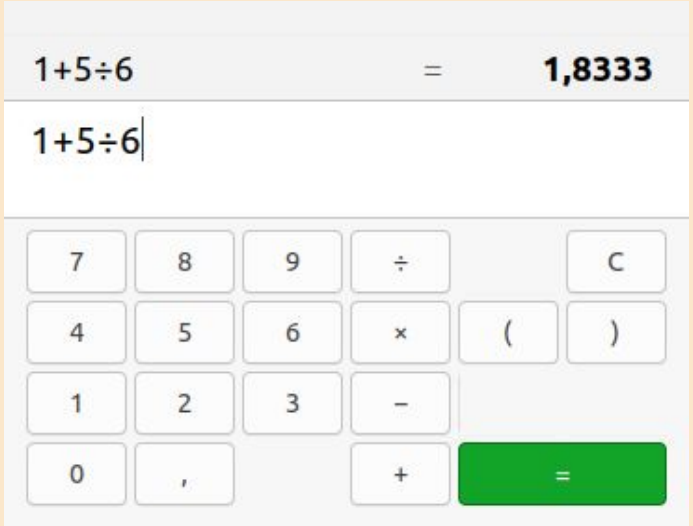
Entre deux : un exercice plus simple en reprenant les bases :
Le but : Une calculatrice ++

- Créer une page (Html + css) lié à un javascript pour réaliser une calculatrice
- Envoyer des événement à un serveur local :
    - Si opération valide : temps prit pour réaliser l'opération (c'est à dire le temps entre le premier caractère écrit et la demande de calcul)
    - Si opération invalide (par exemple l'utilisateur tape 1++2) un événement d'erreur
- Lors de ces événements le serveur renvoi un rapport global :
    - Si événement valide : %de calculs plus rapide que moi, moyenne de temps de calcul.
    - Si invalide : Nombre d'événement invalide, date de la dernière précédente erreure
- Le front affiche ces rapports

# Partie 1 : Le front pure html / css

Réaliser la partie purement html / css de la calculatrice avec

- les 9 chiffres et la virgule/le point
- les opération +, -, *, /
- les parenthèses
- un bouton pour effacer la dernière opération
- une zone où apparaîtra ce que je tape
- une zone avec le résultat
  ça doit ressembler à ceci



# Partie 2 : Le Js

2.1 créer un fichier js et l'importer

Nous avons vu dans le cours 1 step0 : nous avons deux fichier: le js et l'html


- Le fichier js est très simple il fait juste un console.log

=> cela va log quelque chose dans la console et je vais pouvoir voir que le fichier js est bien exécuté.

- le html contient une balise script qui permet d'importer ce js (pas besoin de la suite du fichier pour le moment)

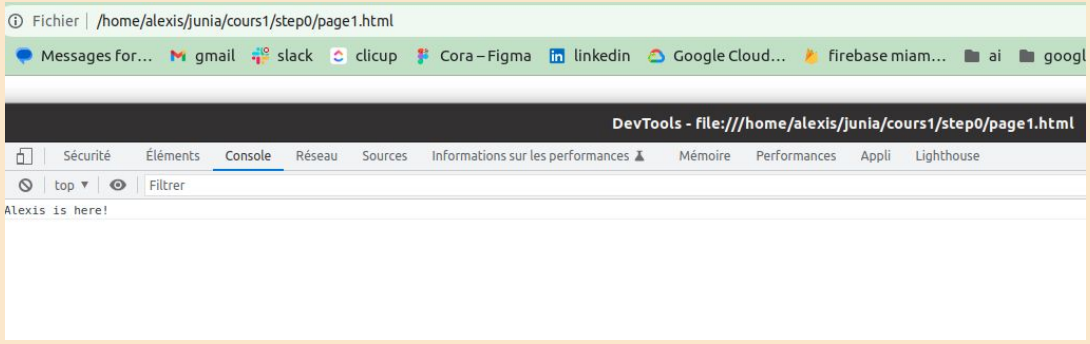
2.1 créer un fichier js et l'importer

Aide pour réussir : reprendre le cours1 step0. Faire un fichier js très simple avec un log. Regarder le html en particulier la balise script. Adapter le chemin de la balise script. La mettre dans mon propre html

Comment valider ?

1. J'ouvre le html dans mon navigateur
2. j'ouvre la console web
3. je recharge la page
4. je regarde si j'ai bien mon log

Voici un screen ou on voit
- la barre de mon navigateur avec l'adresse de mon fichier
- la console de mon navigateur (titre DevTools), je suis dans la rubrique Console
- Mon log : Alexis is Here!



## 2.2 les premières interactions : créer une fonction et l'appeler sur tous les boutons

Nous avons vu dans le cours1 step1 comment créer une fonction js :

```
console.log("Alexis is here!")

function changeToBlue() {
 console.log('blue')
}
…
```

Nous avons vu comment appeler ces fonctions sur nos boutons

```
<button onclick="changeToBlue()">Change to blue</button>
<button onclick="changeToRed()">Change to red</button>
<button onclick="changeToYellow()">Change to yellow</button>
```

2.2 les premières interactions : créer une fonction et l’appeler sur tous les boutons

Aide pour réussir : reprendre le cours1 step2.
Créer une fonction unique qui fait juste un console.log et utiliser cette même fonction sur tous nos boutons

Comment valider ?

1. J’ouvre le html dans mon navigateur
2. j’ouvre la console web
3. Je clic sur mes boutons
4. je regarde dans la console que j’ai bien mon log

## 2.3 les premières interactions : De quelles actions ai-je besoin ?

Pour le moment nous avons une fonction unique. Le but ici est d'en avoir plusieurs.

Nous avons vu dans le cours 1 dans un premier temps chaque bouton avait sa propre action. Puis nous somme passés à une action unique avec un paramètre

Avant :

```
function changeToBlue() {
 console.log('blue')
}

function changeToRed() {
 console.log('red')
}

function changeToYellow() {
 console.log('yello')
}
```

```
<button onclick="changeToBlue()">Change to blue</button>
<button onclick="changeToRed()">Change to red</button>
<button onclick="changeToYellow()">Change to
yellow</button>
```

Après :

```
function changeColor(color) {
 document.getElementById("change_it").style.backgroundColor = color;
}
```

```
<button onclick="changeColor('#330aec')">Change to blue</button>
<button onclick="changeColor('rgb(250, 0, 0)')">Change to red</button>
<button onclick="changeColor('#c7f014')">Change to yellow</button>
```

Remplacer la fonction unique par plusieurs fonctions :

1. Quelles sont les actions similaires qui vont utiliser la même fonction avec un paramètre ?
2. Quelles sont les actions spécifiques qui ont besoin d'une autre fonction.

=> faire donc x fonction qui ne font qu'un console.log différent et les appeler correctement dans le html.

Aide pour réussir : reprendre le cours1 step3 pour avoir la fonction avec paramètre et l'appel.

Comment valider ?

- Commencez par bien réfléchir et essaye de trouver la solution vous même
- regardez les slides suivantes

Pour rappel :

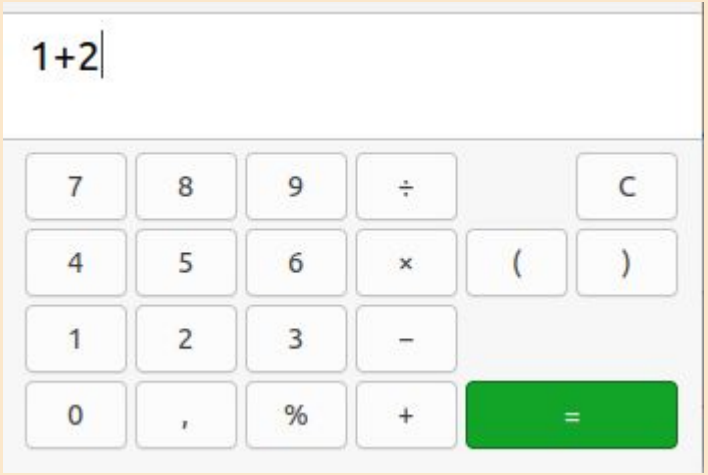
- les 9 chiffres et la virgule/le point
- les opération +, -, *, /
- les parenthèses
- un bouton pour effacer la dernière opération
- une zone où apparaîtra ce que je tape
- une zone avec le résultat

2.3.1 les premières implémentations - modification du dom

Nous allons commencer par implémenter une fonction d'écriture:
Quand je tape sur les touches numéraires, la virgule,  les opération ou parenthèses je veux simplement que le texte correspondant à la touche s'ajoute au texte prévu pour voir ce que je tape.

je passe de


à



2.3.1 les premières implémentations - modification du dom

Pour réaliser ceci:

1. Je dois récupérer un élément du dom
2. Je dois appliquer une opération (ajout d’un caractère)
3. je dois modifier un élément du dom

Dans le cours nous avons vu:

2.3.1 les premières implémentations - modification du dom

Aide pour réussir :

Comment valider ?

## 2.3.2 les premières implémentations - modification du dom

Nous allons maintenant implémenter la fonction de correction. Le but est similaire à la fonction précédente mais au lieu d’ajouter un caractère elle en retire un

2.3.2 les premières implémentations - modification du dom

Aide pour réussir :

Comment valider ?

## 2.3.3 les premières implémentations - modification du dom

Nous allons maintenant implémenter la fonction de calcul.
Elle a une partie similaire à celles d'avant :

1. Je dois récupérer un élément du dom
2. Je dois appliquer une opération
3. je dois modifier un élément du dom

Regardez la fonction 'eval' et essayez dans une console web

2.3.3 les premières implémentations - modification du dom

Aide pour réussir :

Comment valider ?


Vous avez votre première calculatrice !

# Partie 3 : C’est la classe

Ce qu’on a vu en cours

création d’une classe

```
class BaseCalculator {
 constructor() {
   // Do some stuff
 }
}

let baseCalculator = new BaseCalculator();
```

Le but : remplacer dans notre js les fonction en les intégrant dans une classe.

Comment réussir :

- Un copier / coller des fonctions dans la classe doit faire le gros du travaille
- Point d’attention : si j’avais des variables elles vont passer en variable d’instance avec utilisation de ‘this’, ainsi que les methodes
- Changer les fonctions utilisées dans les onClick du html
- mot clef function à supprimer

Comment valider :

- La calculatrice doit fonctionner de la même manière

# Partie 4 : Bonus : le retour arrière

Ce qu’on a vu en cours :

Manipulation de liste :
let maListe = [2, 3, 4]
Fonction push, pop, foreach, map
accès direct : maListe[0], maListe[1], maListe[maListe.length - 1]

```
> let maListe = [2,3,4];
< undefined
> maListe
< ▶ (3) [2, 3, 4]
> maListe.push(5);
< 4
> maListe
< ▶ (4) [2, 3, 4, 5]
> let maValue = maListe.pop();
< undefined
> maValue
< 5
> let maValue = maListe.pop();
< undefined
> maValue
< 4
> maListe
< ▶ (2) [2, 3]
> maListe[0];
< 2
> maListe[maListe.length - 1];
< 3
```

Le but : Avoir une touche qui permet de revenir en arrière.
Si je tape ‘1+2’ revenir en arrière va mettre mon input à ‘1+’ comme si j’avvait fait le coorect pour retirer le dernier caractère
Si le tape ‘1+2’ puis ‘=’ j’ai mon résultat, revenir en arrière me ramène à ‘1+2’ et je peux continuer d’écrire et faire ‘1+2+3’ facilement

Comment réussir :

- enregistrer sur ma classe une variable d’instance qui sera une liste
- Ajouter mes actions (toutes ou presque ?) dans cette liste
- être capable de récupérer les valeurs de cette liste (pensez à les supprimer ou non ?)
- savoir utiliser cette valeure récupérée pour modifier mon input
- penser aux cas limites (rien dans la liste etc.)

Comment valider :

- La calculatrice doit fonctionner de la même manière + créez vous un bon usecase ou j’arrive à revenir en arrière après l’affichage d’un résultat.
- vérifier les cas limites

# Partie 5 : serveur

Ce qu’on a vu en cours :

Créer un projet NestJs, détails en cours 6, 7 et 8
Récap des commandes:

Install et Init du projet :

```
npm i -g @nestjs/cli
```

```
nest new nestjs-project
```

Création d’un module + controller + service + entity

nest g module annonces --no-spec
nest g controller annonces --no-spec
nest g service annonces --no-spec

Entity a faire a la main

Un handler Post pour créer une annonce / envoyer de la data 2 manières:

```
@Post()
createAnnonce(
  @Body('id') id: string,
  @Body('title') title: string,
  @Body('price') price: number,
): Promise<Annonce> {
  return this.annoncesService.createAnnonce(id, title, price);
}
```

```
@Post('dto')
createAnnonceDto(
  @Body() createAnnonceDto: CreateAnnonceDto,
): Promise<Annonce> {
  return this.annoncesService.createAnnonceDto(createAnnonceDto);
}
```

Rappel : but du serveur

- Envoyer des événement à un serveur local :
  - Si opération valide : temps prit pour réaliser l'opération (c'est à dire le temps entre le premier caractère écrit et la demande de calcul)
  - Si opération invalide (par exemple l'utilisateur tape 1++2) un événement d'erreur
- Lors de ces événements le serveur renvoi un rapport global :
  - Si événement valide : %de calculs plus rapide que moi, moyenne de temps de calcul.
  - Si invalide : Nombre d'événement invalide, date de la dernière précédente erreure

Comment réussir ?

Reprendre le projet du cours ou en créer un nouveau, supprimer les choses inutiles pour ne garder pour le moment que la base : le app.module.ts de base. (et le main.ts)
On va ensuite l'enrichir avec nos propre modules

Se poser les bonnes questions

1. De combien de models ai-je besoin en base ?
2. Quelle sera leur définition ?
3. Quelles informations vais-je envoyer au serveur?
4. Quelles routes vais-je donc faire ?

1. De combien de models ai-je besoin en base ?

Le plus simple est de faire deux models :

- Le premier pour les événements de succès
- Le second pour ceux d’erreurs

Vous pouvez donc réaliser 2 modules avec service et controller associés (vide pour le moment)

2: Quelle sera leur définition ?

Le premier devra avoir comme information le temps pris pour l’opération, je vous propose d’enregistrer cela sous format d’un int en millisecondes.
Le second devra avoir un timestamp pour savoir quand l’erreur est arrivée.

Les deux pour avoir comme primary key un id en auto-increment -> je n’ai pas à m'en soucier, à chaque création cette id va s’incrémenter tout seul gérer par la base

Vous pouvez donc créer les deux fichier entity avec

- une column id avec le decorator @PrimaryGeneratedColumn pour les deux
- une column timeTakenMs en int pour le temps prit en ms pour le succès
- une column created_at au format Date pour la date de création (regardez le decorator @CreateDateColumn)

3: Quelles informations vais-je envoyer au serveur?

Pour un événement de succès je vais devoir envoyer le temps que l’utilisateur a prit pour faire l’opération

Pour un événement d’erreur il n’y a finalement rien à envoyer, vous avez directement la date de l’erreur lors de la création du record en base dans le champ created_at

## 4 Quelles routes vais-je donc faire ?

Je vais avoir besoin de deux routes : la première pour le succès, la seconde pour l'erreur.
Je donne une information au serveur je vais donc a priori faire un Post, d'autant plus pour le première cas ou j'ai vraiment une information à donner (le temps prit)

Vous pouvez donc ajouter la fonction au controller de chaque module. Le path de la route est assez automatique. pour vous aider :
Vous pouvez donc reprendre la route vu en cours avec le body :

```
@Post()
createAnnonce(
  @Body('id') id: string,
  @Body('title') title: string,
  @Body('price') price: number,
): Promise<Annonce> {
  return this.annoncesService.createAnnonce(id, title, price);
}
```

Vous allez bien sûr  changer le nom createAnnonce ainsi que les paramètres de la fonction

Côté service la aussi regardez le cours et reprenez cette fonction :

```
async createAnnonce(id: string, title: string, price: number): Promise<Annonce> {
  const annonce = new Annonce();
  annonce.id = id;
  annonce.title = title;
  annonce.price = price;
  await annonce.save();

  return annonce;
}
```

Vous avez ici bien sûr moins de paramètre a mettre pour coller au model, certains ont même dejà des valeurs par defaut

Comment valider ?

- Utilisez un utilitaire type Postman pour faire vos requêtes
- pensez a mettre des logs
- Vous pouvez aussi vous faire une route Get pour regarder facilement ce que vous avez en base (plutôt que d’aller voir directement en sql)
- Faites vos requêtes Post -> verifier que cela a bien inséré ce que vous voulez en base

# Partie 6 : calculatrice to server

Ce qu’on a vu en cours:

Faire une requête avec XmlHttpRequest :

```
let url = 'https://api.leboncoin.fr/finder/search'
let dataraw = {
  "some_data": "the_data"
};
 var xhttp = new XMLHttpRequest();
xhttp.open("POST", url, true);
xhttp.setRequestHeader("Content-type", "application/json");
xhttp.send(JSON.stringify(dataraw));
```

Traiter la réponse :

```
xhttp.onload = function() {
  console.log(this.responseText);
};
```

Rappel : but. Lors d'une action il faut simplement faire ces appels back et afficher la réponse

Comment réussir ?

Si vous avez bien réussi la partie 5 vous avez déjà réussi à faire ces requêtes vers votre serveur back, par exemple avec postman. Le but est ici double:

1. répliquer ces requêtes depuis le front avec l'outil XmlHttpRequest.
2. Traiter le retour pour afficher la réponse

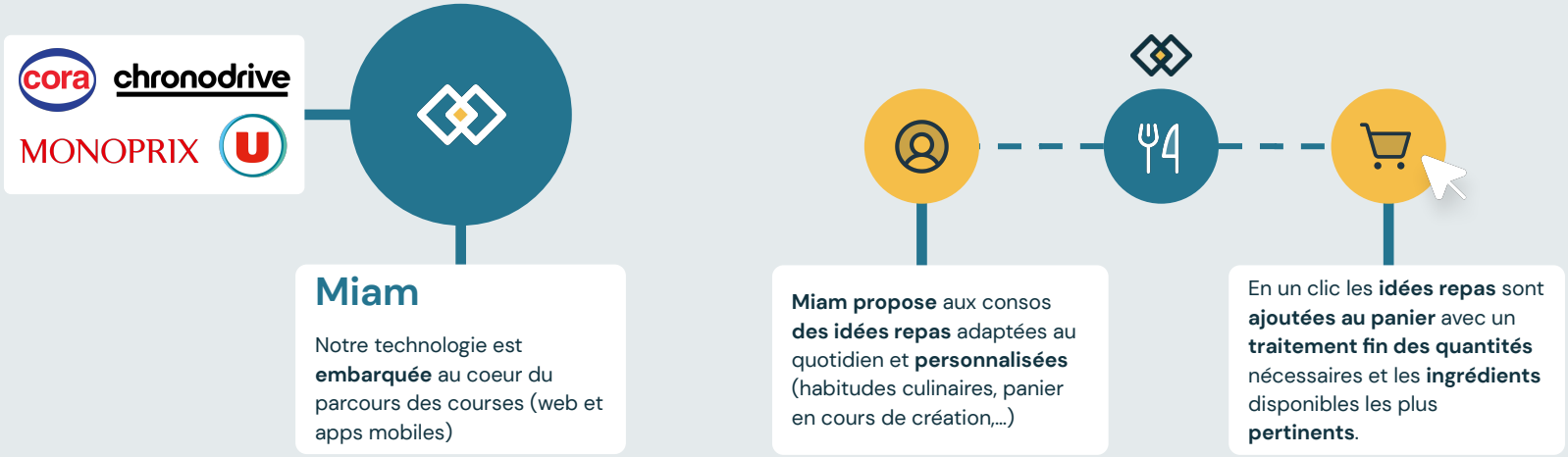
- utiliser les console.log pour regarder si / comment les requêtes partent du front
- utiliser la console chrome pour voir les requêtes qui partent et reviennent du front
- utiliser les console.log pour regarder comment les requêtes arrivent / partent du back

Comment valider :

- reprenez les cas de succès / erreur que vous avez déjà pu utiliser pour vérifier que votre calculatrice fonctionne. Elle doit toujours fonctionner mais ajouter un petit rapport en plus.

# A propos de Miam

Miam réinvente et simplifie les courses alimentaires.




## Miam

Notre technologie est **embarquée** au coeur du parcours des courses (web et apps mobiles)

**Miam propose** aux consos **des idées repas** adaptées au quotidien et **personnalisées** (habitudes culinaires, panier en cours de création,...)

En un clic les **idées repas** sont **ajoutées au panier** avec un **traitement fin des quantités** nécessaires et les **ingrédients** disponibles les plus **pertinents**.

Qui sommes-nous ? Une startup tech lilloise, qui développe depuis 2019 des solutions d'intelligence artificielle à destination de la grande distribution.

Plus de détails sur [https://miam.tech](https://miam.tech)