

Fondamentaux du Web

HTML, CSS, JavaScript

Lab 4 — Layouts & Responsive

Faire en sorte que Jàng Afrig marche sur téléphone

Licence 1 DUT1-Informatique — 2025–2026

Dr. El Hadji Bassirou TOURÉ
École Supérieure Polytechnique
Département Génie Informatique
Université Cheikh Anta Diop de Dakar

Informations pratiques

Type	Lab guidé
Séance	S6 — deuxième Lab CSS
Prérequis	Lab 3 terminé (5 pages stylisées avec tag v0.3)
Validation	App lisible sur téléphone + tag v0.4 sur GitHub
Rapport	1-2 pages : ce que j'ai fait + difficultés + lien repo

Objectifs du Lab

À la fin de ce Lab, vous aurez :

1. Compris pourquoi votre app actuelle est **inutilisable sur téléphone**
2. Ajouté la balise `<meta viewport>` pour le mobile
3. Découvert et utilisé les **media queries** CSS
4. Adopté l'approche **mobile-first**
5. Adapté la **nav**, les **cartes**, et le **tableau** pour mobile
6. Ajouté une **grille de drapeaux** responsive sur la page À propos
7. Testé votre app dans le **mode mobile de DevTools**
8. Posé le tag v0.4 sur GitHub

Table des matières

1	Le problème : votre app n'est pas mobile	2
1.1	Faites le test maintenant	2
1.2	Pourquoi est-ce si moche ?	2
1.3	Ce qu'on va faire aujourd'hui	2
2	Le viewport meta tag (la ligne magique)	3
2.1	Ajouter la ligne dans les 5 pages	3
2.2	Ce que fait cette ligne	3

3	Mobile-first : la philosophie	4
3.1	Deux approches possibles	4
3.2	Pourquoi mobile-first ?	4
3.3	Les 3 breakpoints qu'on va utiliser	4
4	Anatomie d'une media query	5
4.1	Syntaxe	5
4.2	Exemple complet : un titre qui change de taille	5
5	Étape 1 : Adapter le main pour mobile	6
6	Étape 2 : Adapter la nav	6
7	Étape 3 : Adapter les cartes	8
8	Étape 4 : Une grille de drapeaux sur la page À propos	9
8.1	Récupérer les images	9
8.2	HTML : la grille	9
8.3	CSS : la grille responsive	9
9	Étape 5 : Adapter le tableau du palmarès	11
9.1	Modifier scores.html	11
9.2	Ajouter le CSS	11
10	Tester avec DevTools	12
10.1	Le mode mobile de DevTools	12
10.2	Tester les 3 breakpoints	12
10.3	Aperçu visuel attendu	12
11	Exercices d'application	13
12	Aide-mémoire	14
12.1	Le b.a.-ba du responsive	14
12.2	Les 3 breakpoints	14
12.3	Méthodologie mobile-first	14
12.4	Tester sur vrai téléphone	14
13	Routine Git	15
14	Rapport à produire	15
15	Résumé et transition	16
15.1	Ce que vous avez accompli	16
15.2	Le bloc CSS est terminé	16
15.3	Ce qu'on fera au CM3 et au Lab 5	16

1 Le problème : votre app n'est pas mobile

1.1 Faites le test maintenant

Ouvrez votre Jàng Afrig dans le navigateur. Maintenant :

1. Appuyez sur F12 (ou Cmd+Opt+I sur Mac) pour ouvrir DevTools
2. Cliquez sur l'icône **téléphone/tablette** en haut à gauche du panneau (ou Ctrl+Shift+M)
3. Choisissez un appareil dans la liste : **iPhone SE** ou **Pixel 5**

Vous voyez l'horreur : la nav déborde, les cartes sont écrasées ou minuscules, le tableau du palmarès dépasse l'écran à droite et oblige à scroller horizontalement.

1.2 Pourquoi est-ce si moche ?

Deux raisons :

1. Vous n'avez pas la balise `<meta viewport>` dans vos pages, donc le téléphone affiche votre site comme s'il avait 980 pixels de large (et zoome dehors).
2. Vous n'avez écrit **aucune règle CSS spécifique au mobile**. Tout est calculé pour un écran d'ordinateur (1100px de large dans votre main).

1.3 Ce qu'on va faire aujourd'hui

- Ajouter le `<meta viewport>` (10 secondes)
- Adopter une approche **mobile-first** : on écrit d'abord pour téléphone, puis on adapte pour grand écran avec des **media queries**
- Adapter nav, cartes, tableau, et ajouter une grille de drapeaux

Statistique mondiale

En 2025, plus de **60%** du trafic web mondial vient des téléphones. Au Sénégal, ce chiffre dépasse **80%**. Si votre site n'est pas *responsive*, vous abandonnez 8 utilisateurs sur 10. C'est inacceptable.

2 Le viewport meta tag (la ligne magique)

2.1 Ajouter la ligne dans les 5 pages

Dans le `<head>` de **chaque** page HTML, ajoutez cette ligne juste après `<meta charset="UTF-8">` :

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Enregistrez les 5 fichiers. Rafraîchissez votre simulation mobile dans DevTools : la page n'est plus dézoomée. Les éléments ont maintenant leur taille réelle. ✓

2.2 Ce que fait cette ligne

- `width=device-width` : la largeur de la page égale celle de l'écran réel du téléphone (par exemple 375px sur iPhone SE)
- `initial-scale=1` : pas de zoom au chargement

Avant/après cette ligne

Sans `<meta viewport>` : le téléphone simule un écran de 980px et dézoome pour faire tenir la page. Résultat : texte minuscule.

Avec `<meta viewport>` : la page s'affiche à la vraie taille de l'écran. Résultat : texte lisible mais...la nav déborde toujours. Il faut maintenant écrire des règles CSS pour mobile.

À ne jamais oublier

Mettez `<meta viewport>` sur **toutes** les pages HTML que vous créerez dans votre vie. C'est la première ligne après `<meta charset>`. C'est devenu un automatisme pour tous les développeurs web depuis 2010.

3 Mobile-first : la philosophie

3.1 Deux approches possibles

- **Desktop-first** : on écrit le CSS pour grand écran, puis on adapte pour mobile.
- **Mobile-first** : on écrit le CSS pour mobile, puis on adapte pour grand écran.

3.2 Pourquoi mobile-first ?

1. Le mobile est plus **contraint** (petit écran, peu d'espace). Si on règle d'abord ce cas difficile, ajouter de l'espace pour grand écran est facile.
2. Le **layout par défaut** qu'on écrit (sans media query) est celui qui sera vu par les téléphones. C'est le cas le plus fréquent.
3. Les **media queries de type min-width** sont plus lisibles que **max-width**.

3.3 Les 3 breakpoints qu'on va utiliser

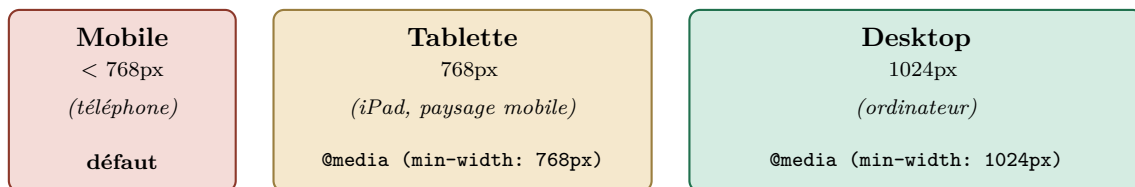


Fig. 1 : Les 3 breakpoints utilisés dans ce Lab. Le CSS de base (sans media query) est pour mobile.

La règle mentale

- Sans aucune media query → s'applique à **tous** les écrans (donc en mobile-first, c'est mon style mobile)
- @media (min-width: 768px) {...} → s'ajoute pour les écrans **plus larges que 768px**
- @media (min-width: 1024px) {...} → s'ajoute pour les écrans **plus larges que 1024px**

Plus l'écran est large, plus de règles s'empilent en se cumulant.

4 Anatomie d'une media query

Une media query est une **condition** : si l'écran respecte une certaine largeur, alors le CSS à l'intérieur s'applique.

4.1 Syntaxe

```
@media (min-width: 768px) {  
  /* Toutes les règles ici ne s'appliquent que */  
  /* si l'écran fait au moins 768 pixels de large */  
  
  .carte-mode {  
    flex: 1 1 250px;  
  }  
  
  main {  
    padding: 0 40px;  
  }  
}
```

4.2 Exemple complet : un titre qui change de taille

```
/* Mobile (par défaut) */  
header h1 {  
  font-size: 1.8rem;  
}  
  
/* Tablette et plus */  
@media (min-width: 768px) {  
  header h1 {  
    font-size: 2.2rem;  
  }  
}  
  
/* Desktop et plus */  
@media (min-width: 1024px) {  
  header h1 {  
    font-size: 2.8rem;  
  }  
}
```

Comment lire ce code :

- Sur téléphone (< 768px) : le titre est à 1.8rem
- Sur tablette (768-1023px) : le titre devient 2.2rem
- Sur desktop (≥ 1024px) : le titre devient 2.8rem

Pourquoi on n'écrit pas max-width ?

On pourrait écrire `@media (max-width: 767px) {...}` pour viser uniquement le mobile. Mais en mobile-first, on préfère :

1. Écrire le mobile *sans condition* (plus simple)
2. *Ajouter* pour les écrans plus grands avec `min-width`

On a moins de doublons et le code est plus facile à lire.

5 Étape 1 : Adapter le main pour mobile

Dans votre `style.css` actuel, vous avez probablement :

```
main {  
  max-width: 1100px;  
  margin: 30px auto;  
  padding: 0 20px;  
}
```

Le `max-width: 1100px` est OK pour mobile (la largeur sera celle de l'écran), mais le `padding: 0 20px` peut être trop sur petit écran. **Modifiez en :**

```
main {  
  max-width: 1100px;  
  margin: 20px auto;  
  padding: 0 15px;  
}  
  
@media (min-width: 768px) {  
  main {  
    padding: 0 30px;  
  }  
}  
  
@media (min-width: 1024px) {  
  main {  
    padding: 0 40px;  
  }  
}
```

Enregistrez. La marge intérieure du `main` grandit avec l'écran. Sur mobile : 15px. Sur tablette : 30px. Sur desktop : 40px. ✓

6 Étape 2 : Adapter la nav

Votre nav actuelle utilise `display: flex` avec `justify-content: center`. Sur mobile, les 4 liens débordent. **Solution simple :** les empiler en colonne sur petit écran, en ligne sur grand écran.

Remplacez le bloc `nav` de votre CSS par :

```
/* ===== Navigation (mobile par défaut) ===== */  
nav {  
  background-color: var(--jaune);  
  display: flex;  
  flex-direction: column;  
  text-align: center;  
}  
  
nav a {  
  color: var(--noir);  
  text-decoration: none;  
  font-weight: 600;  
  padding: 12px 15px;  
  transition: background-color 0.2s;  
  border-bottom: 1px solid rgba(0, 0, 0, 0.1);  
}
```

```
nav a:hover {
    background-color: var(--vert);
    color: white;
}

/* Tablette et plus : on revient en ligne */
@media (min-width: 768px) {
    nav {
        flex-direction: row;
        justify-content: center;
        flex-wrap: wrap;
    }

    nav a {
        border-bottom: none;
        padding: 15px 25px;
    }
}
```

Enregistrez. Testez en redimensionnant la fenêtre :

- Petit écran : les liens sont **empilés verticalement**
 - Grand écran : ils reviennent **en ligne horizontale**
- ✓

Le hamburger menu

Sur les *vrais* sites, on remplace souvent la nav mobile par une icône hamburger () qui ouvre le menu au clic. Cela demande du JavaScript, donc on s'en occupera au-delà de ce cours. Pour le L1, on fait la version simple : les liens en colonne. C'est tout à fait acceptable.

7 Étape 3 : Adapter les cartes

Vos cartes `.carte-mode` ont `flex: 1 1 200px`, donc elles se mettent déjà en grille flexible. Sur mobile, le `1 1 200px` peut donner 1 carte par ligne (parce que 200px est plus large que la moitié de l'écran). Sur tablette, 2 cartes par ligne. Sur desktop, 3 ou plus.

C'est déjà bien. Mais on peut affiner pour avoir un comportement plus prévisible :

Modifiez `.carte-mode` dans votre CSS :

```
/* ===== Cartes de modes (mobile : 1 colonne) ===== */
.modes-grille {
  display: flex;
  flex-direction: column;
  gap: 12px;
}

.carte-mode {
  padding: 16px;
  border-radius: 8px;
  color: var(--noir);
  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);
}

/* Tablette : 2 colonnes */
@media (min-width: 768px) {
  .modes-grille {
    flex-direction: row;
    flex-wrap: wrap;
  }
  .carte-mode {
    flex: 1 1 calc(50% - 12px);
  }
}

/* Desktop : 3 colonnes */
@media (min-width: 1024px) {
  .carte-mode {
    flex: 1 1 calc(33.333% - 12px);
  }
}
```

Enregistrez. Maintenant :

- Mobile : **1 carte par ligne**
- Tablette : **2 cartes par ligne**
- Desktop : **3 cartes par ligne**



calc()

`calc(50% - 12px)` permet de *calculer* une valeur en mêlant unités. Ici : la moitié de la largeur du parent, moins 12 pixels (pour compenser le `gap`). C'est très utile pour les layouts.

8 Étape 4 : Une grille de drapeaux sur la page À propos

Sur `apropos.html`, ajoutez une nouvelle section qui affiche une **galerie de drapeaux africains**.

8.1 Récupérer les images

Téléchargez 6 drapeaux (clic droit → **Enregistrer la cible sous**) dans votre dossier `images/` :

- Sénégal : <https://flagcdn.com/w320/sn.png> → `sn.png`
- Mali : <https://flagcdn.com/w320/ml.png> → `ml.png`
- Maroc : <https://flagcdn.com/w320/ma.png> → `ma.png`
- Côte d'Ivoire : <https://flagcdn.com/w320/ci.png> → `ci.png`
- Égypte : <https://flagcdn.com/w320/eg.png> → `eg.png`
- Afrique du Sud : <https://flagcdn.com/w320/za.png> → `za.png`

8.2 HTML : la grille

Ajoutez à `apropos.html`, dans le `<main>` :

```
<section>
  <h2>Quelques pays au programme</h2>
  <div class="grille-drapeaux">
    <figure>
      
      <figcaption>Sénégal</figcaption>
    </figure>
    <figure>
      
      <figcaption>Mali</figcaption>
    </figure>
    <figure>
      
      <figcaption>Maroc</figcaption>
    </figure>
    <figure>
      
      <figcaption>Côte d'Ivoire</figcaption>
    </figure>
    <figure>
      
      <figcaption>Égypte</figcaption>
    </figure>
    <figure>
      
      <figcaption>Afrique du Sud</figcaption>
    </figure>
  </div>
</section>
```

8.3 CSS : la grille responsive

Ajoutez à `style.css` :

```
/* ===== Grille de drapeaux (mobile : 2 par ligne) ===== */
.grille-drapeaux {
  display: flex;
```

```

    flex-wrap: wrap;
    gap: 15px;
    margin-top: 15px;
}

.grille-drapeaux figure {
    flex: 1 1 calc(50% - 15px);
    text-align: center;
    background-color: white;
    padding: 10px;
    border-radius: 6px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.08);
}

.grille-drapeaux img {
    width: 100%;
    height: auto;
    border-radius: 4px;
}

.grille-drapeaux figcaption {
    margin-top: 6px;
    font-size: 0.9rem;
    font-weight: 600;
}

/* Tablette : 3 par ligne */
@media (min-width: 768px) {
    .grille-drapeaux figure {
        flex: 1 1 calc(33.333% - 15px);
    }
}

/* Desktop : 6 par ligne */
@media (min-width: 1024px) {
    .grille-drapeaux figure {
        flex: 1 1 calc(16.666% - 15px);
    }
}

```

Enregistrez. Sur la page `apropos.html`, vous voyez les 6 drapeaux disposés :

- Mobile : 2 par ligne
- Tablette : 3 par ligne
- Desktop : 6 par ligne (alignés)

✓

<figure> et <figcaption>

Ces deux balises HTML5 sémantiques servent à associer une image à sa légende. Plus propre que d'enchaîner `<img>` et `<p>` anonymes : un lecteur d'écran annoncera figure : drapeau du Sénégal, légende Sénégal.

9 Étape 5 : Adapter le tableau du palmarès

Le tableau a 5 colonnes (Rang, Pseudo, Mode, Score, Date). Sur mobile, ça déborde et il faut scroller horizontalement. Plusieurs solutions existent. La plus simple : permettre le **scroll horizontal contrôlé** dans un conteneur.

9.1 Modifier scores.html

Enveloppez le tableau dans un `<div>` :

```
<div class="tableau-conteneur">
  <table>
    <!-- ... lignes du tableau ... -->
  </table>
</div>
```

9.2 Ajouter le CSS

```
/* ===== Conteneur de tableau (scroll horizontal sur mobile) ===== */
.tableau-conteneur {
  overflow-x: auto;
  -webkit-overflow-scrolling: touch;
  margin-top: 15px;
}

table {
  min-width: 480px;
}

/* Police plus petite sur mobile */
@media (max-width: 767px) {
  th, td {
    font-size: 0.85rem;
    padding: 8px;
  }
}
```

Enregistrez. Sur mobile, le tableau a maintenant un **ascenseur horizontal local** (juste pour le tableau, pas pour toute la page). C'est élégant. ✓

Pourquoi cette approche ?

Le tableau du palmarès a 5 colonnes *essentiels*. On ne peut pas en cacher sans perdre de l'info. Le scroll horizontal localisé est la solution la plus simple et la plus courante. Plus tard, en L2, vous verrez d'autres techniques (cartes par ligne, transformation en accordéon).

10 Tester avec DevTools

10.1 Le mode mobile de DevTools

Dans Chrome ou Firefox :

1. Ouvrez DevTools (F12)
2. Cliquez sur l'icône **téléphone/tablette** en haut à gauche
3. Sélectionnez un appareil dans la liste déroulante :
 - iPhone SE (375×667) — petit téléphone
 - iPhone 14 Pro (393×852) — téléphone moderne
 - iPad Mini (768×1024) — tablette
 - Responsive (taille libre, redimensionnez à la souris)

10.2 Tester les 3 breakpoints

1. Choisissez **iPhone SE** (375px) : tout doit s'afficher en colonne
2. Choisissez **iPad Mini** (768px) : la nav passe en ligne, les cartes en 2 colonnes
3. Passez en **Responsive** et étirez à 1100px : 3 cartes par ligne, 6 drapeaux par ligne

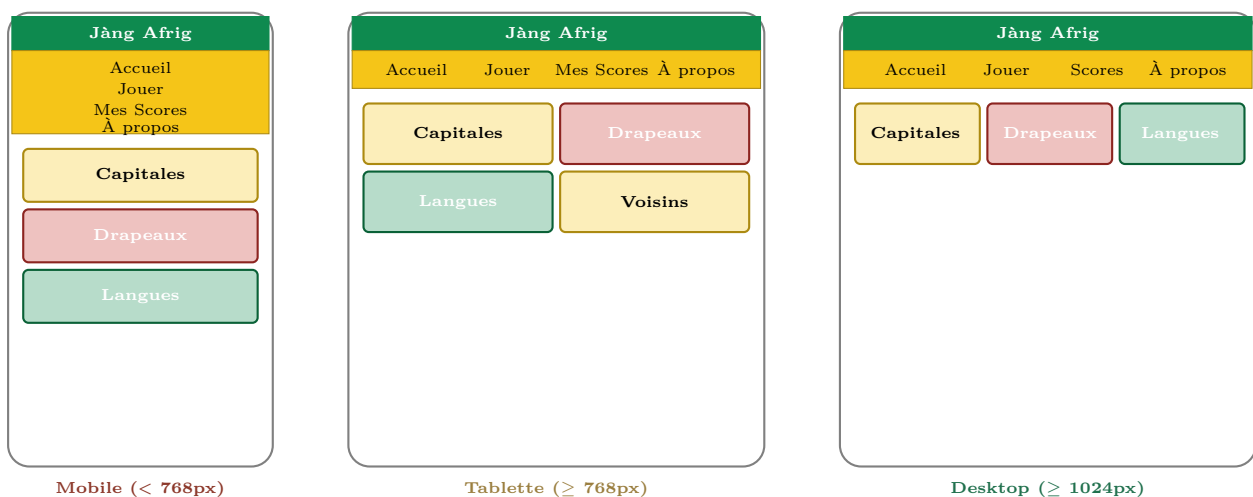
Tester sur un vrai téléphone (bonus)

Si vous voulez voir Jàng Afrig sur *votre* téléphone réel :

1. Connectez votre PC et votre téléphone au **même Wi-Fi**
2. Dans VS Code, faites tourner Live Server
3. Notez l'adresse IP de votre PC (ipconfig sur Windows, ifconfig sur Mac/Linux)
4. Sur votre téléphone, ouvrez le navigateur et tapez :
`http://192.168.X.X:5500/index.html`

Ça marche, c'est très satisfaisant.

10.3 Aperçu visuel attendu



La même page affichée différemment selon la largeur. C'est ça, le responsive.

11 Exercices d'application

Trois mini-exercices

Exercice 1 — Footer responsive

Faites en sorte que sur mobile, le footer affiche le copyright sur une ligne et le lien En savoir plus sur la ligne suivante (au lieu de tout sur une ligne qui déborde). Indice : `display: flex; flex-direction: column; gap: 8px;`.

Exercice 2 — Image héros responsive

Sur la page d'accueil (`index.html`), ajoutez juste après le `<header>` une grande image héros (par exemple un drapeau de l'Union Africaine).

- Largeur 100% de l'écran
- Hauteur max 200px sur mobile, 300px sur desktop
- Centrée, avec `object-fit: cover`

Exercice 3 — Formulaire responsive

Sur `quiz.html`, faites en sorte que le formulaire occupe au maximum :

- 100% de la largeur sur mobile
- 500px centrés sur tablette et desktop

Indice : `form { max-width: 500px; margin: 0 auto; }`.

Bonus — Affichage sticky

Faites en sorte que la nav reste **collée en haut** de l'écran quand on scrolle. Une seule ligne de CSS suffit : `position: sticky; top: 0;` sur nav.

12 Aide-mémoire

12.1 Le b.a.-ba du responsive

Élément	Rôle
<code><meta name="viewport" content="..."></code>	La ligne magique pour le mobile
<code>@media (min-width: 768px)</code>	Règles pour tablette et plus
<code>@media (min-width: 1024px)</code>	Règles pour desktop et plus
<code>flex-direction: column / row</code>	Empiler / aligner
<code>flex-wrap: wrap</code>	Retour à la ligne
<code>flex: 1 1 calc(50% - 15px)</code>	2 colonnes flexibles
<code>overflow-x: auto</code>	Scroll horizontal local
<code>position: sticky; top: 0;</code>	Élément collé en haut au scroll

12.2 Les 3 breakpoints

Plage	Cible	Media query
<code>< 768px</code>	Mobile (par défaut)	— (CSS de base)
<code>768 – 1023px</code>	Tablette / paysage	<code>@media (min-width: 768px)</code>
<code>≥ 1024px</code>	Desktop	<code>@media (min-width: 1024px)</code>

12.3 Méthodologie mobile-first

1. Écrire d'abord le CSS pour mobile (**sans** media query)
2. Tester sur DevTools mode mobile (iPhone SE 375px)
3. Ajouter `@media (min-width: 768px)` pour ce qui doit changer en tablette
4. Tester en mode tablette (iPad Mini 768px)
5. Ajouter `@media (min-width: 1024px)` pour le desktop
6. Tester en mode responsive à 1100px

12.4 Tester sur vrai téléphone

1. PC et téléphone sur le même Wi-Fi
2. `ipconfig` (Windows) / `ifconfig` (Mac/Linux) pour récupérer l'IP du PC
3. Sur téléphone : `http://IP-DU-PC:5500/index.html`

13 Routine Git

```
git status
git add .
git commit -m "v0.4 : responsive mobile + grille drapeaux"
git push
git tag v0.4
git push --tags
```

Vérifiez sur <https://github.com/votre-username/jang-afrig> :

- Le `<meta viewport>` est dans les 5 pages
- `style.css` contient les media queries
- Les 6 drapeaux sont dans `images/`
- Le tag `v0.4` est visible

14 Rapport à produire

Rapport du Lab 4 — obligatoire

Format : 1-2 pages, déposé dans `rapports/rapport-lab-4-NOM-Prenom.pdf`.

Contenu (4 sections) :

1. **Ce que j'ai fait** : viewport ajouté, media queries écrites, nav/cartes/tableau adaptés, grille de drapeaux, exercices d'application.
2. **Difficultés rencontrées** : layout qui casse, breakpoints qui ne marchent pas, scroll horizontal indésirable.
3. **Comment j'ai résolu** : DevTools mode mobile, MDN, aide d'un camarade.
4. **Lien vers mon repo avec le tag v0.4** :
ex. <https://github.com/fatou-diop/jang-afrig/releases/tag/v0.4>

Bonus dans le rapport : **2 captures d'écran** de la même page : une en mode mobile (375px), une en mode desktop (1100px). C'est votre preuve fière que vous savez faire du responsive.

15 Résumé et transition

15.1 Ce que vous avez accompli

- Ajouté `<meta viewport>` aux 5 pages ✓
- Adopté l'approche **mobile-first** ✓
- Maîtrisé les **media queries** avec 3 breakpoints ✓
- Adapté nav, cartes, formulaire, tableau ✓
- Construit une grille de drapeaux responsive ✓
- Testé dans le mode mobile de DevTools ✓
- Posé le tag v0.4 ✓

15.2 Le bloc CSS est terminé

Vous savez maintenant tout le CSS dont vous avez besoin pour ce semestre : sélecteurs, cascade, box model, Flexbox, variables, media queries, responsive. C'est solide.

15.3 Ce qu'on fera au CM3 et au Lab 5

CM3 (S7) : JavaScript débarque. Le langage qui anime les pages. On verra variables, fonctions, conditions, boucles. Plus de pages statiques — on va commencer à *réagir* aux clics de l'utilisateur.

Lab 5 (S8) : JS pratique. Vous découvrirez la console DevTools, écrirez vos premières lignes de JS, et ferez bouger un élément de Jàng Afrig avec un clic.

À ce stade du semestre

Votre application a **une vraie interface professionnelle** qui marche **sur n'importe quel appareil**. Plus de la moitié du chemin est faite.
Manque maintenant le **cerveau** : la logique JavaScript qui rend l'app *interactive*. C'est ce qu'on attaque dans la prochaine séance.

Lab 4 — Terminé ✓

Votre app fonctionne sur tous les écrans, du téléphone à l'ordinateur.

Place à JavaScript : on va lui donner vie.

Rendez-vous au CM3 !