Intro :

- Qui suis-je
- Qui êtes-vous ? votre niveau ?
  - CSS
    - [ ] flex box : déjà utilisé
    - [ ] css grid ? idem ? déjà utilisé / pas sûr
    - [ ] media query ? déjà utilisé, déjà entendu parlé
    - [ ] bonnes pratiques conventions : BEM, rscss ? déj utilisé
    - [ ] sass, less, postcss, css in js ? déjà utilisé
  - JS
    - [ ] promises ? déjà utilisé — ptetre un petit cours dessus

      - [ ]
      ```
      new Promise()

      fetch('google.com')
      .then(() => {})
      ```

    - [x] template string ? déjà vu

      - [x]
      ```
      var nomDuProf = 'Géraud'
      var str = `mon nom est ${nomDuProf}`
      ```

    - [ ] object destructuring ? non — un petit cours dessus

      - [ ]
      ```
      let prof = {
        nom: 'Henrion',
        prénom: 'Géraud'
      }

      let {nom, prenom} = prof

      let nom = prof.nom
      let prenom = prof.prenom
      ```

      - [ ]

    - [ ] object spread ? à voir ensemble

      - [ ]
      ```
      let prof = {
      nom: 'Henrion',
      prenom: 'Géraud',
      ```

```
}

let entreprise {
nom_entreprise: 'Nartex'
}


let result = {
...prof,
...entreprise,
}

result = {
nom: 'Henrion',
prenom: 'Géraud',
nom_entreprise: 'Nartex'
}

```

- [ ] array spread ? prévoir un cours

```
let array1 = [1, 2, 3]
let array2 = [3, 4]

[
...array1,
...array2,
]

// [1, 2, 3, 3, 4]


```

- [ ] array.map ? à voir en cours

```
let array1 = [1, 2, 3]

let array2 = array1.map(number => number * 2)

// [2, 4, 6]
```

- Typescript ?

```
let age: number = 27
let nom: string = 'Géraud'


```

- Programmation en général
  - [ ] architecture SPA -> à voir
  - [ ] web worker ? pas vraiment le sujet mais pq pas
  - [ ] frameworks JS que vous connaissez ?
    - [ ] vue
    - [ ] angular
    - [ ] react
    - [ ] jQuery
  - [ ] notion d'effet de bord ? -> je reviendrai dessus
    - [ ]
    - [ ]
  - [ ] état global, état local ? -> je reviendrai dessus
  - [ ] fonction pure / impure ? -> on reviendra dessus
    - [ ] pas d'effet de bord -> n'impacte pas l'extérieur
    - [ ] toujours le même résultat pour les mêmes arguments
- Présentation du contenu de cette UE (plan ci-dessous)

```


-webkit-transform: translate(100px);
        transform: translate(100px);

// mixin

@mixin transform (translate(100px));




```

Plan

- introduction au responsive design
    - Histoire rapide du CSS
    - Le mobile, un autre medium, différentes contraintes
    - Les différents types de mise en page
    - Patterns responsive
    - Exercices
- Qu'est-ce qu'un composant ?
    - un brique d'interface indépendante et réutilisable
- qu'est-ce qu'une single page application ?
    - responsabilité de la navigation et de la génération du markup est déléguée au client
    - le serveur n'est plus responsable plus que de contenir les données
- la philosophie du framework React :
    - https://www.youtube.com/watch?v=FleElijoXtk&list=PLIvCYh5AD3HwVHc7wvMSrluM6mhtDGRpR&index=1
    - déclaratif vs impératif
    - exprimer interface utilisateur sous la forme d'un arbre de composants encapsulés
    - source de vérité : DOM vs app state
        - la vue est une fonction directe du state
        - cohérence entre ce qui est affiché et ce qui est en mémoire
    - la render loop
    - unidirectionnal data flow
    - components all the way down
    - JSX
        - syntactic sugar (sucre syntaxique)
        - https://www.youtube.com/watch?v=Z9XppQBgArc&list=PLIvCYh5AD3HwVHc7wvMSrluM6mhtDGRpR&index=3
    - data, props, functions, functions, functions, functions...
    - propriétés en entrée, événements en sortie
    - surface d'API minimale
    - hooks
    - fonctionnement ? (virtual DOM, diffing)
- Qu'est-ce que Cordova
- Installation :
    - nodejs
    - npm, yarn
    - create-react-app

- Projet : une application de chasse au trésor
    - React + Ionic + Cordova
    - Localisation, Nfc, ...

Cours de Web Mobile

Introduction au responsive design

Histoire rapide de la mise en page en CSS

- historiquement, les documents étaient uniquement en HTML et la mise en page et l'apparence étaient gérés par des balises
    - `<font>`, etc.
    - des bidouilleurs ont réussi à y remédier en faisant des mises en page à l'aide de tableaux
- Puis le CSS est créé en 1998 /* DATE à vérifier */ pour apporter des styles et des couleurs
    - à ce moment naît la mise en page à base de float: left
    - premières librairies CSS fournissant une grille
- Nouvelle évolution : arrivée des media queries permettant de modifier les règles CSS en fonction des dimensions de la fenêtre !
    - naissance des premiers frameworks CSS, notamment bootstrap et foundation,
    - populaires pour leurs grilles
    - populaires la facilité avec laquelle ils permettent d'implémenter des patterns responsive
- Arrivée de Flexbox puis de Grid
    - fin des hacks
    - et nouveaux patterns
    - donner exercice sur les grenouilles s'ils connaissent pas encore

Le mobile, un autre medium, différentes contraintes

- La méthode de navigation
    - effet tunnel plutôt que arborescence
- La place à l'écran
    - menus moins remplis, menus cachés
- La résolution
    - écran rétina x2, x3, x4
    - notion de résolution / pixel / dp
        - 1px != 1px
        - 1px == 1dp
- La taille des doigts

    - plus grosses zones cliquables (40dp mini)
    - plus d'espace entre les éléments (8dp mini)
- La bande passante
    - prix / délais
    - taille des images
    - taille du Javascript
- La latence
    - popularité des web app dites "SPA" ou "Ajax" (ce qu'on va faire)
- Le hors ligne
    - utilisation des web workers
- La performance
    - taille du javascript
    - optimisations
        - (attention à ne pas trop en faire, surtout au début !)

Patterns responsive

Ici, donner des exemples de sites internet et proposer des exos ??

- Passage d'une ligne à une colonne
- Grille responsive
- Afficher / masquer un élément
    - pour des raisons de place
    - pour des raisons esthétiques
    - pour la performance
- Changement d'ordre des éléments
- Réduire la taille du texte
- Changer la photo
    - selon la taille de l'écran
    - selon l'orientation de l'écran !
- Le hamburger menu
    - Iconique du web mobile
    - Fut très populaire, remplacé petit à petit par la bottom bar
    - Tendance à trop l'utiliser, peut poser des problèmes d'accessibilité

Javascript moderne

es6, es2015, es2020 ...

JavaScript = ECMAScript — ECMA étant l'organisation s'occupant des standards du langage.

JS = JavaScript, ES = ECMASCRIPT

(il existe d'autres langages basés sur ECMAScript, comme ActionScript ou TypeScript)

En 2015 après des années de stagnation le langage a fait un bond en avant en sortant sa 6e édition appelée es6 (renommée plus tard en es2015)

- une nouvelle édition chaque année,
  - nouveau nommage : es20XX
- un système de proposal public en plusieurs étapes
  - https://github.com/tc39/proposals
  - tout le monde peut voir et participer aux débats
  - décision prise par le commité TC39, qui inclut ECMA et des représentants des navigateurs

Es6 connu pour avoir apporté beaucoup de nouvelles fonctionnalités, visibles ici :

https://github.com/daumann/ECMAScript-new-features-list

Compatibilité

https://caniuse.com/

https://babeljs.io/

ES modules : `import` `export`

possibilité de définir des modules JS

- un module = scope totalement isolé
- imports
  - nommés
  - default
- exports
  - nommés
  - default

https://github.com/daumann/ECMAScript-new-features-list/blob/master/ES2015.MD#modules https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/import https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/export

Suite des exos : mettez votre résultat sur le discord

`let`, `const` (and `var`)

https://github.com/daumann/ECMAScript-new-features-list/blob/master/ES2015.MD#let--const

- `let` et `const` élminent une certaine catégorie de bugs
  - exemple de gotcha

```
for (var i = 0; i < 9; i++) {
 setTimeout(() => console.log(i), 1000);
}
```

- quand utiliser l'un ou l'autre ?
-
  - `let` -> le contenu de la variable pourra être **réassigné**
  - `const` -> le contenu ne pourra pas pas être **réassigné**
    - attention ça ne veut pas dire que c'est immuable
    - un objet ou un tableau déclaré avec `const` sera toujours modifiable, donc *pas constant* à proprement parler
- ce qu'i lfaut retenir : éviter `var` et utiliser `let` ou `const` à la place
  - et respecter les convention du projet / de l'équipe
  - beaucoup de débat sans fin sur quoi utiliser et quand...
- à titre personnel j'utilise beaucoup `const`

```
// Appliquer la nouvelle syntaxe
var age = 21
var name = "Jane"

age = age + 1
```

```
// Retirer la nouvelle syntaxe
const profession = "apiculteur"
let salary = 1337

salary = salary + 100
```

Enhanced Object Literals

https://github.com/daumann/ECMAScript-new-features-list/blob/master/ES2015.MD#enhanced-object-literals

```
// Appliquer la nouvelle syntaxe
let age = 21
let name = "Jane"

let person = { age: age, name: name }
```

```
// Retirer la nouvelle syntaxe
let profession = "apiculteur"
let salary = 1337

let job = { salary, profession }
```

Destructuring

https://github.com/daumann/ECMAScript-new-features-list/blob/master/ES2015.MD#destructuring

Objets

Appliquer la syntaxe :

```
// Appliquer la nouvelle syntaxe
let person = { age: 21, name: "Jane" }

let age = person.age
let name = person.name
```

Retirer la syntaxe

```
// Retirer la nouvelle syntaxe
let job = { salary: 1337, profession: "apiculteur" }

let { salary, profession } = job
```

Tableaux

```
// Appliquer la nouvelle syntaxe
let names = ["Jane", "John"]

let john = person[1]
let jane = person[0]
```

```
// Retirer la nouvelle syntaxe
let jobDescription = ["apiculteur", 1337]

let [profession, salary] = jobDescription
```

Opérateur spread : `...`

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/Syntaxe_d%C3%A9composition

tableau

construction

```
// Appliquer la nouvelle syntaxe
let salariesSemester1 = [1300, 1301, 1303, 1304, 1305, 1306]
let salariesSemester2 = [1307, 1308, 1309, 1310, 1311, 1312]

let fullYear = group1.concat(group2)
```

```
// Retirer la nouvelle syntaxe
let group1 = ["Jean", "Jeanne"]
let group2 = ["John", "Jeannette"]

let everyone = [...group1, ...group2]
```

déconstruction

```
// Appliquer la nouvelle syntaxe
let everyone = ["Jean", "Jeanne", "John", "Jeannette"]

let first = everyone[0]
let everyoneElse = everyone.slice(1)
```

```
// Retirer la nouvelle syntaxe
let salaries = [12000, 24000, 24000]

let [firstYearSalary, ...otherYears] = salaries
```

paramètres

https://github.com/daumann/ECMAScript-new-features-list/blob/master/ES2015.MD#default--rest--spread

récupération des arguments :

```
// old way
function example () {
  console.log(arguments) // c'est une variable magique
}

example(1, 2, 3)
```

```
// new way
function example (...args) {
  console.log(args)
}

example(1, 2, 3)
```

```
// bonus
function example (first, ...args) {
  console.log(first, args)
}

example(1, 2, 3)
```

application des arguments

```
// old way
function add (a, b, c) {
  return a + b + c
}

let result = add.apply(null, [2, 4, 6])
console.log(result)
```

```
// new way
function add (a, b) {
  return a + b
}

let result = add(...[2, 4, 6])
console.log(result)

let result2 = add(2, ...[4, 6])
console.log(result)
```

objet

construction

```
// Appliquer la nouvelle syntaxe
let person = { age: 21, name: "Jane" }

let olderPerson = Object.assign({}, person, {age: 22})
```

```
// Retirer la nouvelle syntaxe
let job = { salary: 1337, title: "apiculteur" }

let betterPaid = {
  ...job,
  salary: 2337,
}
```

Qu'est-ce qu'une Single Page Application ?

Dans une SPA (pas la balnéo), l'architecture consiste à déléguer la responsabilité de la navigation et de la génération du HTML au client, tous les échanges de donnée se faisant en Ajax vers des services externes.

Le serveur ne se charge plus que de la gestion des données (sauvegarde, communication)

- la page web est chargée UNE fois
- et ensuite, toute la navigation + le rendu + la récupération des données est faite côté CLIENT
  - -> faut gérer le rendu soi-même
  - -> faut gérer les changements d'url soi-même aussi
  - en général, c'est super dur à faire juste sans rien
    - pour ça que les gros frameworks existent
    - ça scale beaucoup mieux

Qu'est-ce qu'un composant (component) ?

En web, un composant est une brique d'interface qui est :

- indépendant
  - isolé
- réutilisable
- et composable (on peut en imbriquer plusieurs entre eux et en former de nouveaux)

- contient sa propre mise en page (**view**) -> isolé
- peut avoir son propre état interne (**state**) -> ex: focus

- et potentiellement sa propre logique métier
  - faire des requêtes
  - modifier une base de donéne
  - etc.
- mais vu de l'extérieur c'est une boîte noire à la quelle on va modifier ses propriétés (**props**) et écouter ses événements en retour (**events**)
  - `<Component color="red" onClick={() => alert('coucou')} />`

https://rscss.io/components.html

https://material.io/components

https://ant.design/components/button/

Ce concept se retrouve dans tous les frameworks JS modernes (mettre des exemples) :

- Angular
- Vue
- React
- et même jQuery

Vocabulaire :

- Props (propriétés / configuration)
  - configuration provenant de l'exterieur

```
<Component color="red" />

props === { color: 'red' }
```

- State
  - état interne : menu ouvert ou fermé, survolé, focus, etc.
    - en général ce qui disparaît au rechargement de la page (= volatile)
  - état partagé : utilisateur actuel, cache, url
    - informations partagées entre plusieurs composants
  - état de l'application = somme de tous les états de ses composants
- View
  - reflet de l'état de l'application
  - `f(state) = view`, `f` étant notre code
- Events
  - ce qui se passe dans l'application, fait évoluer son état
  - `event + state = newState`

    - `f(newState) = newView`
- Lifecycle / cycle de vie
    - un composant naît, vit et meurt
        - exemple widget météo
        - page météo qui affiche notamment un widget avec la température
            - ce widget fait des requêtes régulièrement pour connapitre la météo
            - au changement de page, on l'affiche plus
            - -> on va détruire
        - on peut éxécuter des callbacks à ces moments,
            - création
                - démarrer des trucs
            - destruction
                - on les éteint
        - notamment pour la gestion de la mémoire (
            - démarrer et arrêter un setInterval par ex
            - écouteurs d'événements
    - on peut exécuter des callbacks à ces moments,
    - notamment pour la gestion de la mémoire (
        - démarrer et arrêter un setInterval par ex
        - écouteurs d'événements
    - View
        - l'apparence du composant à un moment donné
        - state = état interne de l'app
        - f = notre application
        - `view = f(state)`
        - `event + state = newState`
        - `newView = f(newState)`

Installations

Installer Nodejs

Via homebrew sur macos

Via chocolatey sur windows (powershell ou cmd)

Via votre package manager si sur linux

Installer android studio + adb + sdk si sous android ou sous windows/linux

Suivre les instructions ici https://ionicframework.com/docs/developing/android (on utilisera Capacitor donc ignorer la section Cordova)

À l'étape " Installing the Android SDK "

1. lancer android studio
2. au moment où il y a un message "Please provide the path to the android SDK" cliquer sur "cancel" et reconfirmer (cliquer sur "NO" au second message d'erreur)
3. si ça revient, faire pareil
4. aller dans paramètres / settings
5. rechercher SDK
6. à droite de "Android SDK location" cliquer sur "Edit"
7. garder les paramètres par défaut et faire "next", "next", "next"
8. ça peut prendre un peu de temps le temps de tout télécharger + installer
9. Si vous avez un android, installez aussi la version du sdk correspondant à celle de votre tél

Ajout des commandes au path : https://developer.android.com/studio/command-line/variables

Exemple `~/.bashrc` :

```
export ANDROID_SDK_ROOT=$HOME/Library/Android/sdk

# avdmanager, sdkmanager
export PATH=$PATH:$ANDROID_SDK_ROOT/tools/bin

# adb, logcat
export PATH=$PATH:$ANDROID_SDK_ROOT/platform-tools

# emulator
export PATH=$PATH:$ANDROID_SDK_ROOT/emulator
```

Installer xcode si sous mac et sous iOS

https://ionicframework.com/docs/developing/ios (on utilisera Capacitor donc ignorer la section Cordova)

+ cocoapods

bug xcode select :

```
sudo xcode-select -s /Applications/Xcode.app/Contents/Developer
```

Installer ionic cli

```
npm install -g @ionic/cli
```

- créer un projet : `ionic start`

Installer React Devtool

https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=fr

https://addons.mozilla.org/fr/firefox/addon/react-devtools/

React

Intro :

construire son propre micro-framework déclaratif en 15 lignes de js

- https://stackblitz.com/edit/js-nlyzrr?file=index.html

métaphores react

- https://egghead.io/talks/javascript-drawing-the-invisible-react-explained-in-five-visual-metaphors
- pdf

la philosophie du framework React :

- https://www.youtube.com/watch?v=FleEIijoXtk&list=PLIvCYh5AD3HwVHc7wvMSrluM6mhtDGRpR&index=1
- déclaratif vs impératif
- exprimer interface utilisateur sous la forme d'un arbre de composants encapsulés
- source de vérité : DOM vs app state
    - la vue est une fonction directe du state
    - cohérence entre ce qui est affiché et ce qui est en mémoire

- la render loop
- unidirectionnal data flow
- components all the way down
- JSX
    - syntactic sugar (sucre syntaxique)
    - https://www.youtube.com/watch?v=Z9XppQBgArc&list=PLIvCYh5AD3HwVHc7wvMSrluM6mhtDGRpR&index=3
- data, props, functions, functions, functions, functions...
- propriétés en entrée, événements en sortie
- surface d'API minimale
- hooks
- fonctionnement ? (virtual DOM, diffing)

hooks : https://reactjs.org/docs/hooks-intro.html

Installation de :

- eslint + beautify on save
- ?

Exercice créer une app react simple, à laide de create-react-app

Notions :

- ajouter de l'interactivité (state, etc.)
    - exemple : entrez votre nom + votre nom en reverse
    - bonus : éditable dans les deux sens
    - autre exemple : todo liste
- introduction aux hooks
- créer un hook personnalisé
- faire un composant
- répéter ce composant
- lift state up
- react-router
- useEffect (ex : récupérer des données)

Exos :

- Faire une requête vers une API (une recherche Pokédex ?)
    - https://pokeapi.co/api/v2/pokemon?limit=50
    - Afficher une liste de pokémon

    - La rendre filtrable
- Faire une mini app de TODO list (persistence en local)
- Faire une mini app de prise de photos (+ partage ?)
    - changer le thème : https://ionicframework.com/docs/theming/color-generator
- Puis on peut rentrer dans le gros du projet

Pour aller plus loin :

https://fr.reactjs.org/docs/hello-world.html

https://fr.reactjs.org/tutorial/tutorial.html

Ionic, Capacitor

Capacitor + Ionic

Capacitor

Capacitor = plateforme qui permet de faire une appli mobile qui va contenir une webview + plugins

Appli mobile = couche native (swift pour iOS, kotlin/java pour Android)

Webview = iframe qui est DANS une appli mobile = une simple page web

Webview -> contrôlée par notre "coquille" (l'appli mobile) qui va lui ajouter des plugins

Plugins = vont permettre d'étendre les fonctionnalités du navigateur

- à la base un navigateur c'est sandboxé (bac à sable) donc pas d'accès aux fichiers, caméra, géolocalisation, notifs push
- les plugins donnent accès à tout ça

Ionic

Lot de fonctionnalités qui regroupe :

- des plugins
    - natifs
    - des Fallbacks qui fonctionnent en web !
        - par exmple, une interface pour prendre en photo avec la caméra
        - https://medium.com/@maxlynch/building-the-progressive-web-app-os-57daebcb69c1
- des composants (web) d'apparence native

- mimétisme plateforme -> pour look and feel "natif"
- performant
- cross-framework
  - basé sur une techno -> web components (pas le sujet du cours)

Intro sur async/await

Suivre tutoriel ici : https://ionicframework.com/docs/react/your-first-app

- version plus lisible de l'étape 5

```
async function savePicture(
    photo: CameraPhoto,
    fileName: string
  ): Promise<Photo> {
    // "hybrid" will detect Cordova or Capacitor;
    if (isPlatform("hybrid")) {
      const file = await readFile({
        path: photo.path!,
      });

      const savedFile = await writeFile({
        path: fileName,
        data: file.data,
        directory: FilesystemDirectory.Data,
      });
      // Display the new image by rewriting the 'file://' path to HTTP
      // Details: https://ionicframework.com/docs/building/webview#file-protocol
      return {
        filepath: savedFile.uri,
        webviewPath: Capacitor.convertFileSrc(savedFile.uri),
      };
    }

    const base64Data = await base64FromPath(photo.webPath!);

    await writeFile({
      path: fileName,
      data: base64Data,
      directory: FilesystemDirectory.Data,
```

```
    });

    return {
      filepath: fileName,
      webviewPath: photo.webPath,
    };
  }
```

Exercices :

- ajouter une étape pour demander le nom de la photo avant de l'enregistrer
- optimiser le chargement initial des photos on les parallélisant (Promise.all)

ionic

- surcouche sur cordova
- composants prédéfinis qui communiquent avec le téléphone
- mimétisme interface native
- des utilitaires pour aller plus vite (démarrer un projet)

cordova

- webview (= navigateur embarqué dans une appli mobile)
- plugins
  - communiquent avec les fonctions natives du téléphone

cross-plateforme

- cordova / ionic
- flutter
  - en train de monter en puissance
  - même philosophie que react
- react native
  - react, avec un autre interpréteur
- XAML, nativescript

- Qu'est-ce que Cordova
- Installation :
  - nodejs
  - npm, yarn
  - ionic starter

Projet : une application de chasse au trésor

React + Ionic + Cordova

Localisation, Nfc, ...

Bibliographie

- https://dev.to/ziizium/site-layouts-in-css-2n6o
- https://dev.to/ziizium/introduction-to-responsive-web-design-45g6
- https://ivomynttinen.com/blog/ios-design-guidelines
- https://developer.apple.com/design/human-interface-guidelines/ios/icons-and-images/image-size-and-resolution/
- https://vimeo.com/169809377
- https://www.smashingmagazine.com/2016/09/the-thumb-zone-designing-for-mobile-users/
- https://responsivedesign.is/patterns/