Rapport Adaptation des IHM
Emploi du temps
Groupe 10

AKHMADOV Baisangour
BORRY David
BOUCHER-THOUVENY Vincent
GRIVON Justin

Introduction

Dans le cadre de ce projet, nous avons développé un emploi du temps similaire à l'hyperplanning, nous avons choisi ce sujet car il présente des opportunités d'adaptation liées aux dispositifs (smartphone, PC) et aux utilisateurs (dans notre cas non voyants/malvoyants).
Nous avons chacun développé avec une technologie différente afin d'analyser les bénéfices et inconvénients de ces technologies et voir comment elles permettent l'adaptation de l'IHM de l'EDT sur différents supports.

Responsive Web: Bootstrap (Akhmadov Baisangour)

Le Responsive Web design consiste en le développement d'un site web dont la visualisation et l'utilisation est confortable sur des supports différents, dont notamment les appareils à petits écrans comme les smartphones et les appareils à plus grand écran comme les PC. Dans le cadre de ce projet nous avons utilisé le framework Bootstrap pour le Responsive web.

Description et installation

Bootstrap est un framework qui consiste en un ensemble de feuilles SASS (ici en version 4) qui donnent accès à des outils de composition et des composants. Il permet notamment d'adapter l'interface du site web à différents tailles d'écrans et donc différents appareils (Smartphone, tablette, PC). Bootstrap est utilisé par le biais de classes CSS prédéfinies qu'on utilise dans les balises HTML.
Pour installer bootstrap, le moyen le plus simple est d'ajouter la balise suivante dans le code HTML de la page qu'on développe:

```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css"
integrity="sha384-PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rRUpPvrszuE4W1povHYgTpBfshb"
crossorigin="anonymous">
```

Nous avons de plus utilisé du JS avec JQuery, afin de faciliter notamment la création des cases de l'EDT à partir de données JSON, pour utiliser JQuery on ajoute la balise suivante dans le code de la page:

```
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
   integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
              crossorigin="anonymous"></script>
```

On peut ensuite utiliser les classes Bootstrap au sein du code HTML:

```
<div class="nom de la classe bootstrap>...</div>
```

Avec Bootstrap il existe deux aspects essentiels pour l'adaptation:
1. Les différentes tailles d'écran, en effet dans bootstrap on distingue 5 catégories d'écran:

xs	sm	md	lg	xl
<= 576px de largeur	>= 576px de largeur (smartphone)	>= 768 px de largeur (tablette)	>= 992px de largeur (PC)	>= 1200px de largeur

Cela joue un rôle important dans le système de grille de bootstrap.

2. Le système de grille de Bootstrap est basé sur un système de 12 colonnes par ligne qui constituent la largeur du conteneur. On peut indiquer à un élément combien de ces colonnes il occupe, ce qui permet de déterminer quel sera sa taille et son placement.
De plus on peut indiquer différentes quantités de colonnes pour les différentes catégories d'écrans décrites précédemment, exemple:

```
<div class="col-lg-3 col-sm-12">
```

Ici on indique que le contenu du div prendra 3 colonnes pour les écrans larges et 12 pour les écrans petits. Cela permet par exemple d'afficher 4 images par ligne dans une grille d'image sur un grand écran (PC) et une seule image par ligne sur un petit écran (Smartphone).

Remarque: A noter que depuis Bootstrap 4, pour les écrans très petits on n'écrit plus "col-xs-12" mais "col-12"

Utilisation et tests

Etant donné qu'on utilise bootstrap, pour exécuter l'exemple il suffit d'ouvrir la page HTML. Pour les tests de l'adaptation nous avons utilisé le mode responsive (vue adaptative) de Firefox/Chrome qui permet de voir quel affichage on obtient selon différentes résolutions et qui permet aussi de sélectionner un appareil pour voir quel affichage on obtiendra sur celui-ci (Iphone 6, Galaxy S7, Ipad, Laptop…)

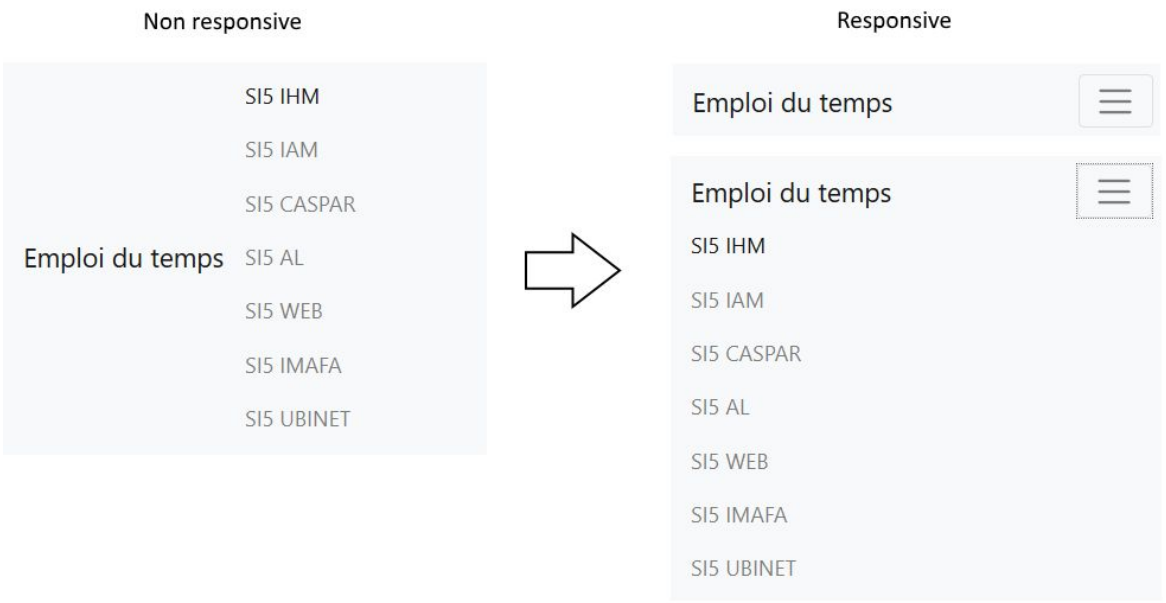
Réalisation

Pour le développement nous avons simplement eu besoin d'un éditeur de texte. (Notepad++, SublimeText par exemple)

Menu

On utilise le composant navbar Bootstrap pour le menu en haut de la page .

Ce menu est responsive, on affiche les boutons du menu normalement sur un grand écran (ordinateur), et on les cache lorsqu'on est sur un petit écran, l'utilisateur peut dans ce cas appuyer sur un bouton situé en haut à droite pour afficher les éléments du menu.



Affichage menu sur appareil mobile

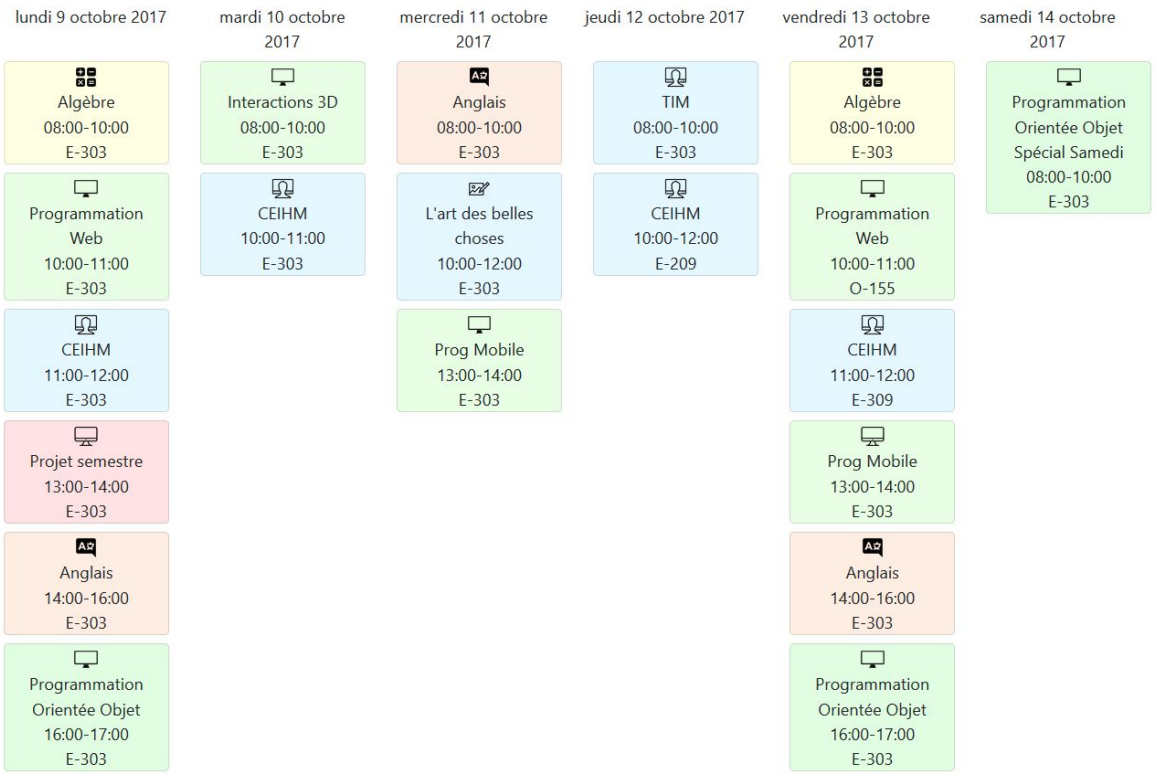
Voici la partie de code qui permet de créer le menu:

```
<nav id="navheader" class="navbar navbar-expand-lg navbar-light bg-light">
  <button class="navbar-toggler navbar-toggler-right" type="button" data-toggle="collapse"
data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item">
        <a class="nav-link" href="#">SI5 IHM</a>
      </li>
       ...Reste des options du menu
    </ul>
  </div>
</nav>
```

La partie <button> correspond au bouton qui permet d'afficher/cacher le menu sur un appareil de petite taille, et on peut voir la liste contenant les différents liens du menu au sein d'un div ayant la classe "collapse navbar-collapse". Cette classe englobe les différents liens du menu qui seront caché ou affiché selon la taille de l'écran.
On note aussi la classe "navbar-expand-lg" qui indique qu'on va afficher le menu en entier pour les appareils de catégorie lg ou supérieur, et que pour les appareils plus petit on aura un bouton comme on peut voir sur les images au-dessus.

Grille

Nous utilisons la disposition en grille de bootstrap que nous avons décrite précédemment pour placer les cases de l'emploi du temps et pour que la taille des cases s'adapte à la taille de l'affichage. Pour les cases en elle même nous avons utilisé la classe Bootstrap "card", on englobe les informations affichées dans la carte (nom du cours, horaire, salle) dans une balise div utilisant cette classe.



Grille EDT sur un PC/Tablette

Au niveau du code cela nous donne ceci (le code est simplifié, on ne met pas ici les styles et classes css utilisés qui ne proviennent pas de bootstrap):

```
<div class="container">
  <div id="edt" class="row">
    <div class="col-2">
      <div class="col card">
        Contenu créneau 1 jour 1 (lundi)
      </div>
      <div class="col card">
        Contenu créneau 2 jour 1 (lundi)
      </div>
    </div>
    <div class="col-2">
      <div class="col card">
        Contenu créneau 1 jour 2 (mardi)
      </div>
```

```
    </div>
    ...
  </div>
</div>
```

On a ici donc un div avec la classe “row” dans laquelle on crée des div “col” correspondant aux différents jours de la semaine, ces colonnes possèdent elles aussi des div “col” qui correspondent chacun à un créneau (une case sur la grille) de ce jour.

Pour l’adaptation sur les petits écrans (smartphone) nous avons développé 2 versions:

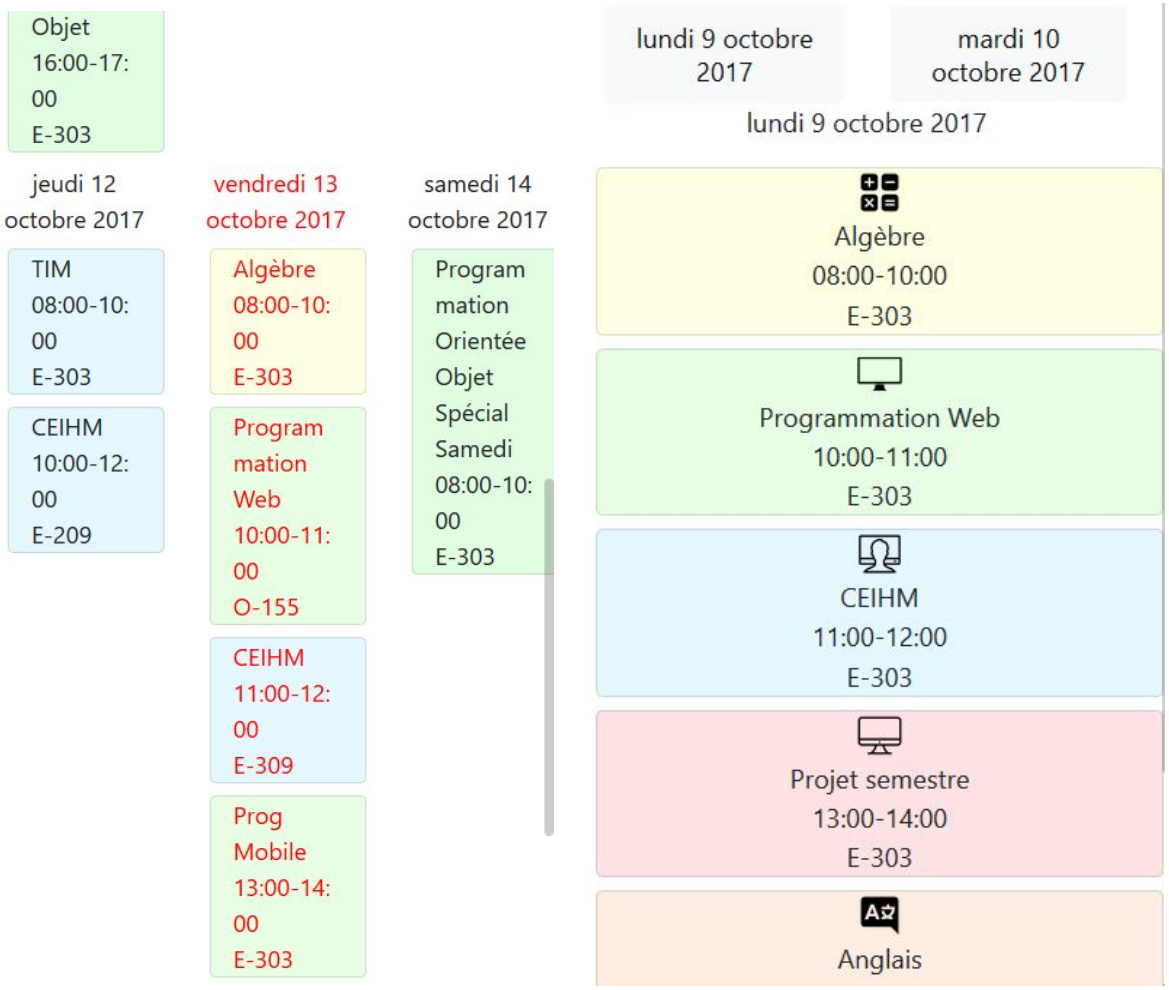
- Une qui adapte la grille aux petits écrans avec le système de grille responsive de bootstrap -> col-x col-md-x col-sm-x (x est le nombre de colonnes)
  Dans cette version, on met en évidence les cours du jour courant en mettant le texte en rouge pour les cases du jour actuel.
  Au niveau du code pour cet exemple on reprend l’exemple précédent sauf qu’au lieu d’avoir pour chaque jour un div:
  `<div class="col-2">`
  On aura:
  `<div class="col-md-2 col-4">`
  Ce qui va faire qu’on affichera que 3 jours par ligne au lieu de 6 sur les petits écrans (xs et sm)
- Une qui utilise un affichage différent pour les écrans mobiles, pour cela on utilise les classes “d-none” “d-md-none” et “d-md-block” qui permettent de cacher ou afficher le contenu HTML selon la taille de l’écran. L'idée c'est qu'on a un div avec l'interface mobile et un div avec la grille desktop. On utilise les classes décrites précédemment pour décider quelle interface afficher.

Concrètement pour cette 2e version, pour le contenant le code HTML de l’interface desktop on ajoute la classe “d-none” et “d-md-block”, cela indique que ce div ne sera pas affiché pour les petits écrans (sm et xs) et sera affiché pour les écran de taille md et plus.
Pour l'interface mobile, on utilise la classe “d-md-none” qui indique que le contenu de ce div sera caché pour les écrans de type md et supérieur.
Niveau code ça donne:

```
<div class="d-none d-md-block">
        ...Code HTML de l'interface bureau (grille de la semaine)
</div>
```

L'interface mobile énoncée précédemment affiche uniquement les cours d'un jour, le résultat est bien plus lisible, on peut afficher plus d'informations pour les différents cours et on peut naviguer entre les jours grâce à des boutons. On fait en sorte que lors de l'ouverture de l'EDT on affiche les cours du jour actuel.
Niveau code on a:

```
<div class="d-md-none">
       ...Code HTML de l'interface mobile (boutons, cases du jour sélectionné)
</div>
```



Mobile Version grille (gauche) / Version affichage adapté (droite)

Détails d'un cours

Lorsqu'on clique sur un des créneaux horaires les détails du cours sont affichés, on affiche le nom du cours, l'horaire, la salle, l'enseignant ainsi que d'éventuelles images/vidéos.

Pour l'affichage on utilise les "modal" bootstrap, qui permettent de faire afficher une pop up qui contient les informations détaillées du cours.

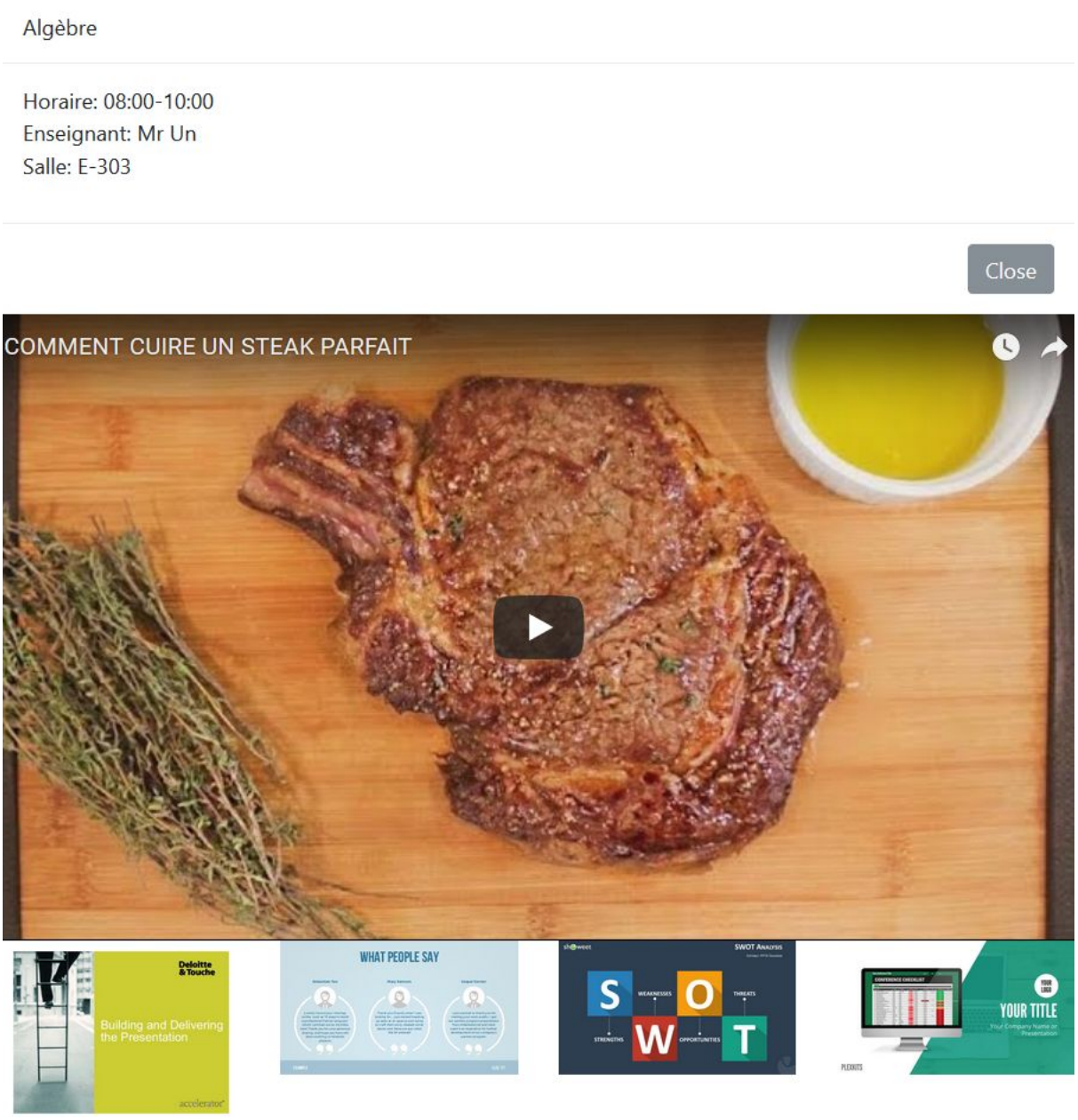
Le pop up ainsi que le texte contenu s'adapte à la taille de l'écran afin que le popup soit visible quel que soit la taille de l'écran (pas de scroll horizontal, uniquement un scroll vertical si il y a une grande quantité d'informations dans le pop up)

En ce qui concerne l'intégration d'images/vidéo:

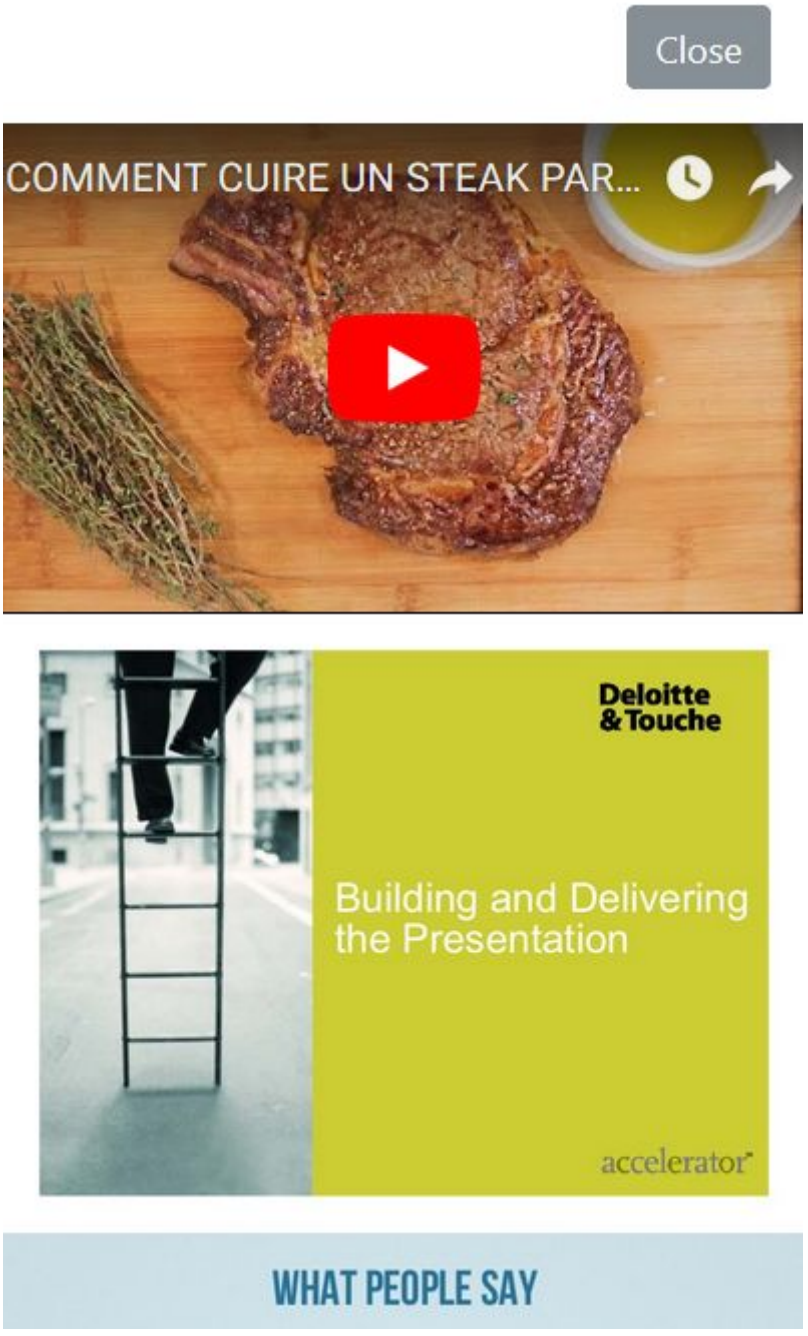
- Pour la vidéo, celle-ci correspond à une vidéo youtube qu'on intègre dans le pop up du site. On utilise la classe “embed-responsive” de bootstrap qui permet d'assurer que la taille de la vidéo s'adapte afin qu'elle soit contenue dans le pop up.
  On englobe la vidéo (balise <iframe>) dans une balise comme celle-ci:
  `<div class="col-12 embed-responsive embed-responsive-16by9">`

On peut bien voir la classe “embed-responsive” et la classe “embed-responsive-16by9” qui indique que la vidéo aura un “aspect ratio” de 16:9

- Pour les images, on utilise le système de grille de bootstrap, on affiche alors les images une par une sur un smartphone (une image par ligne), tandis qu'on augmente le nombre d'images qu'on affiche par ligne avec la taille de l'écran (jusqu'à 4 images par ligne).
  Niveau code, on englobe les images avec une balise comme celle-ci:
  `<div class="col-12 col-sm-12 col-md-6 col-lg-4 col-xl-3">`
  On indique ici que pour un écran très petit/petit on aura une image par ligne, pour les moyens 2, large 3 et très larges 4.



Affichage desktop (écran large): on a la vidéo et 4 images par ligne (dans le site, les images peuvent être cliquées pour les afficher séparément et dans une plus grande taille)



Affichage mobile, on n'affiche qu'une image par ligne, la vidéo s'est adapté correctement au changement de taille d'affichage

Code du modal (popup):

```
<div class="modal fade" id="infoModal">
       <div class="modal-dialog modal-lg" role="document" >
              ...Contenu du popup (header, corps, footer)
        </div>
</div>
```

On ajoute les donnée suivantes au correspondant aux cases de l'EDT pour qu'un clic sur une case ouvre le modal:

```
<div ... data-toggle="modal" data-target="#infoModal" ... >
```

On remarque que dans data-target on indique l’id du div correspondant au modal, ce qui permet de l’afficher lors du clic.

AngularJS (Justin GRIVON)

Description et installation

AngularJS est un framework javascript libre et open source développé par google qui se repose sur une architecture MVC afin de créer une page web, le langage utilisé est Typescript , un sur-ensemble de javascript.

Pour installer AngularJS on peut passer par npm :

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

Pour créer un nouveau projet, il suffit d'ouvrir un terminal et de marquer ceci:

```
ng new my-app
```

On peut ensuite lancer le serveur en allant dans le répertoire du projet (ici my-app) et lancé la commande:

```
cd my-app
ng serve --open
```

Mise en évidence et réalisation

Le but de cette expérience et de mettre en avant les avantages / inconvénients d'AngularJS au niveau de l'IHM d'une page web. Le test se fait sur la performance d'une page à être responsive, nous avons donc utilisé le mode responsive (vue adaptative) de Firefox/Chrome.

Les composants AngularJS utilisent le modèle MVC, ce qui signifie que chaque composant est complet et est indépendants des autres au niveau de l'affichage et des fonctions qu'il implémente.

Pour avoir une page responsive, l'idée est d'avoir un composant manager qui affiche tel ou tel composant en fonction de la taille de l'écran.

Composant manager:

app.component.ts

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent{
  private desktopDisplay: boolean;
  constructor() {
  }

  public onResize(event){
    if(window.screen.width > 768)
      this.desktopDisplay = true;
    else
      this.desktopDisplay = false;
  }
}
```

app.component.html

```
<div (window:resize)="onResize($event)"></div>
<div *ngIf="desktopDisplay; else mobileTemplate">
  <app-desktop></app-desktop>
</div>
<ng-template #mobileTemplate>
  <app-mobile></app-mobile>
</ng-template>
```

la méthode onResize s'exécute lorsque la taille de la fenêtre change. En fonction de sa largeur, une variable prend la valeur 1 ou 0 ce qui permet de choisir un composant d'un autre.

Voici l'affichage web des deux composants:

Mobile display

lundi mardi

lundi

Conception et évaluation IHM
O8:00-09:00
ALBA WINCKLER
O+308

Conception et évaluation IHM
9:00-10:00
ALBA WINCKLER
O+308

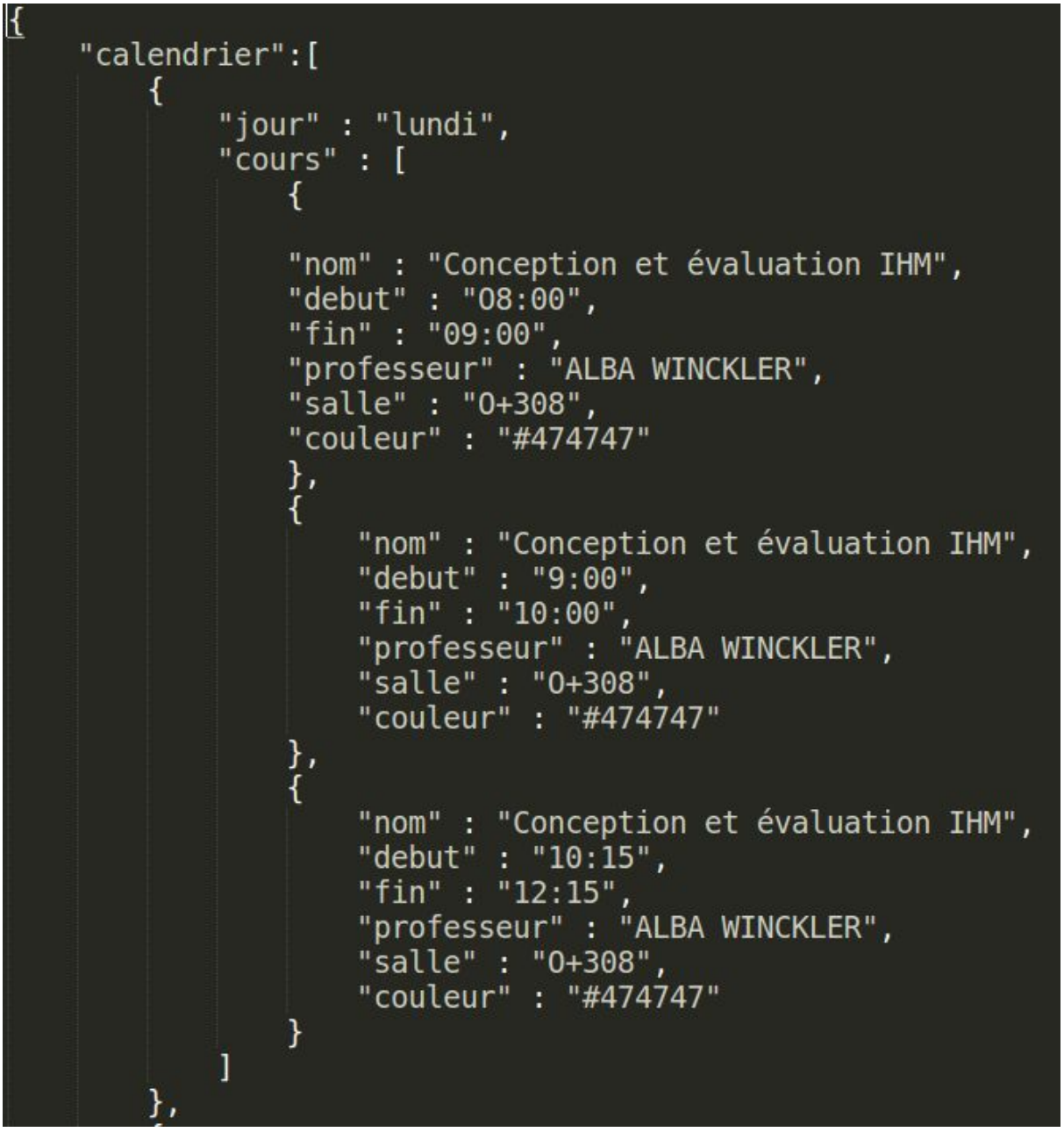
Conception et évaluation IHM
10:15-12:15
ALBA WINCKLER
O+308

Computer display

lundi	mardi	mercredi	jeudi	vendredi
Conception et évaluation IHM O8:00-09:00 ALBA WINCKLER O+308	Systèmes d'exploitation avancé O8:00-10:00 LAVIROTTE O+101	Intéragir dans un monde 3D 08:30-09:30 SAMOUN E+134	Techniques d'intéractions 08:00-10:00 MARONGLIU 0+302	Adaptation des interfaces 08:00-10:00 BREL 0+308
Conception et évaluation IHM 9:00-10:00 ALBA WINCKLER O+308	Systèmes d'exploitation avancé 10:15-12:15 LAVIROTTE O+101	Intéragir dans un monde 3D 09:30-10:30 SAMOUN E+134	Techniques d'intéractions 10:15-12:15 MARONGLIU 0+302	Adaptation des interfaces 10:15-12:15 BREL 0+308
Conception et évaluation IHM 10:15-12:15 ALBA WINCKLER O+308		Intéragir dans un monde 3D 10:45-12:15 SAMOUN E+134		

Les deux vues ont un code html différents, un css différents et des fonctions javascripts différentes. On a donc deux pages différentes pour les deux types de display.

Pour le remplissage de l'emploie du temps, un fichier json est placé dans le dossier /assets du projet:



```
{
    "calendrier":[
        {
            "jour" : "lundi",
            "cours" : [
                {

                "nom" : "Conception et évaluation IHM",
                "debut" : "08:00",
                "fin" : "09:00",
                "professeur" : "ALBA WINCKLER",
                "salle" : "O+308",
                "couleur" : "#474747"
                },
                {
                    "nom" : "Conception et évaluation IHM",
                    "debut" : "9:00",
                    "fin" : "10:00",
                    "professeur" : "ALBA WINCKLER",
                    "salle" : "O+308",
                    "couleur" : "#474747"
                },
                {
                    "nom" : "Conception et évaluation IHM",
                    "debut" : "10:15",
                    "fin" : "12:15",
                    "professeur" : "ALBA WINCKLER",
                    "salle" : "O+308",
                    "couleur" : "#474747"
                }
            ]
        },
```

Ce fichier est utilisé par un Service Angular qui est partagé avec le composant pour smartphone et celui pour écran d'ordinateur:

```
@Injectable()
export class CalendarService{

    constructor(private http: Http) {
        var obj;
        this.getCalendar().subscribe(data => obj=data, error => console.log(error));
    }

    public getCalendar(): Observable<Calendar> {
        return this.http.get("assets/calendar.json")
        .map(function(res){
           let data = res.json().calendrier;
           return data.map(obj => new Jour(obj));

        })
     }
}
```

qui génère l'emploie du temps grâce aux classes Calendar,Cours et Jour à partir du json en utilisant la fonction map

```
export interface Calendar{
    calendrier : Array<Jour>;
}
```

```
export class Cours{
    nom : string;
    debut : string;
    fin : string;
    professeur : string;
    salle : string;
    couleur : string;

    constructor(values: Object = {}) {
        Object.assign(this, values);
    }
}
```

```
export class Jour{
    jour : string;
    cours : Array<Cours>;

    constructor(values: Object = {}) {
        Object.assign(this, values);
    }
}
```

Ce service est alors utilisé par les deux composants gérant l'affichage:

```
export class DesktopComponent implements OnInit{
  private title = 'Computer display';
  private jours: Calendar;
  constructor(private DataService: CalendarService) {
    }
    ngOnInit(): void {
        this.getAllDays().subscribe(data => {
            this.jours = data;
        });
    }

    public getAllDays(): Observable<Calendar> {
        return this.DataService.getCalendar();
    }

}
```

```
export class MobileComponent implements OnInit{
  private title = 'Mobile display';
  private jours: Calendar;
  constructor(private DataService: CalendarService) {
  }
  ngOnInit(): void {
      this.getAllDays().subscribe(data => {
          this.jours = data;
      });
```

On peut alors utiliser l'attribut "jours" dans ses templates html :

```
<li *ngFor="let jour of jours" class="center-li">
```

Android natif (David BORRY)

Pour ce projet, j'ai voulu tirer parti des fonctionnalités du SDK d'Android pour créer une application d'emploi du temps adaptée à un public d'utilisateurs malvoyants. Grâce à des API de reconnaissance et de synthèse vocale, on peut en effet facilement avoir des informations sur le prochain cours de la journée sans avoir à regarder l'écran.

Description et installation

Le système d'exploitation Android est accompagné d'un SDK permettant de développer, débuguer et tester des applications exclusivement conçues pour des smartphones, tablettes, télévisions et objets connectés Android. Les applications natives sont développées avec **Java** et **XML**. Depuis 2015, **Android Studio** est l'IDE officiel du SDK. Il est grandement inspiré par les solutions de **Jetbrains**.

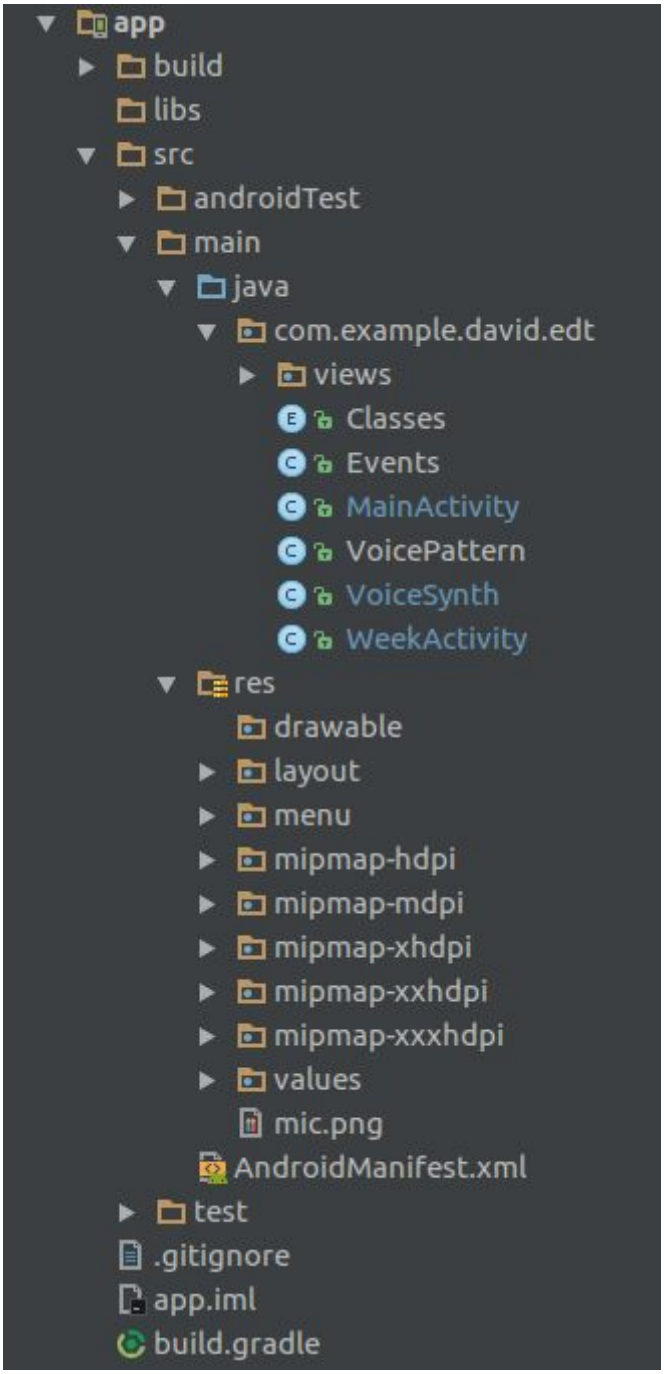
Pour installer et tester un projet d'application, il faut d'abord avoir avoir installé le SDK et l'IDE d'Android. Sur Ubuntu, on peut le faire avec le paquet **android-sdk**. Il faut ensuite télécharger Android Studio depuis son site officiel sous la forme d'une archive. Dans le répertoire bin se trouve un script, **studio.sh** qui permet de lancer l'IDE. Vous pouvez ensuite commencer à développer en Android sur votre PC.

Pour démarrer un nouveau projet, on choisit **File->new->new project**.
On vous demandera dans un premier temps de donner un nom pour votre application. Après avoir cliqué sur **next**, vous devez spécifier la version minimale d'android requise pour pourvoir l'utiliser. Par défaut, l'IDE en choisira une de sorte que votre application soit compatible avec 100% des appareils android enregistrés. Des versions d'API plus récentes offrent plus de fonctionnalités, mais viseront un public moins large.
Cliquez sur **next** une fois votre choix validé. Vous devez alors choisir le modèle de votre première **activité**, une des vues interactives composant une application. Nous reviendrons plus en détail sur le fonctionnement des activités par la suite. Après avoir choisi un modèle, donnez un nom à votre activité et cliquez sur **finish**. Votre projet android est correctement initialisé et prêt à l'emploi.

Vous pouvez voir dans la figure suivante l'arborescence générée après création du projet, plus particulièrement le répertoire **app/**. On y trouve tous les éléments essentiels au fonctionnement de l'application, à savoir:

- ***build.gradle*** : fichier de configuration de l'environnement de l'application. C'est ici que vous spécifiez la version du SDK d'android sur laquelle compiler, le type de compilation et les éventuelles dépendances pour importer des modules externes dans votre projet.

- ***AndroidManifest.xml*** : ce fichier permet de décrire l'application. C'est ici que sont spécifiés le nom de l'application, l'activité de lancement et les éventuelles permissions requises pour fonctionner correctement (géolocalisation, accès à internet…).



- ***res/*** : Comme son nom l'indique, c'est ici que sont stockées les ressources de l'application. On y trouve des images, texte pré-définis et, plus important, des fichiers xml pour décrire les activités. Ces derniers sont stockés dans le fichier layout et permettent, en plus de déclarer les différents composants d'une activité (bouton, texte, vue web) de définir des règles de styles pour adapter la vue aux dimensions de l'appareil.

- ***java/*** : Contient le code métier de l'application. C'est ici qu'on implémente la logique des activités mais aussi toutes les classes et méthodes java pouvant être utilisées par l'application. On peut enfin y implémenter de nouveaux composants exclusifs à l'application. Le SDK d'Android fournit des composants qui suffisent la plupart du temps pour des applications classiques, mais rien n'empêche le développeur de créer son propre composant avec ses propriétés et sa gestion des évènements.

Utilisation et tests

Comme il s'agit d'une application native, l'installation se fait sur smartphone ou tablette via un fichier **.apk** automatiquement généré par l'IDE après compilation. Ouvrez l'apk sur votre appareil. Si votre version d'Android est compatible, l'installation s'effectue rapidement et l'application et prête à l'emploi.

Au lancement, vous aurez directement la vue des cours de la semaine. En changeant l'orientation de l'appareil, vous pouvez voir que le nombre de jours affichés varie: 3 en mode portrait et 7 en mode paysage. Vous pouvez parcourir les heures et jours en balayant l'écran verticalement ou horizontalement.

La reconnaissance vocale est désactivée par défaut: vous pouvez l'activer en cochant l'option "Assistance auditive" dans le menu en-haut à droite. Pour la tester, maintenez le bouton enfoncé sur l'emploi du temps quelques instants. Une fenêtre de dialogue s'affichera pour que vous parliez dans le micro. Si votre phrase contient la séquence "**prochain cours**", la synthèse vocale décrira le nom, le lieu et l'horaire du prochain cours et la vue sera automatiquement centrée dessus. La synthèse vocale vous préviendra si la séquence n'est pas reconnue afin que vous puissiez recommencer.

Les tests ont été effectuées sur un **Galaxy S8** ainsi que des émulateurs d'appareils aux dimensions variées grâce aux machines virtuelles proposées par l'IDE. On a ainsi pu tester l'affichage de l'emploi du temps sous différentes dimensions et orientations. La synthèse vocale a également été testée sur deux voix différentes pour s'assurer que les textes Français étaient correctement retransmis.

Le SDK d'android permet de développer des applications avec un soin particulier pour l'adaptation: à la manière de bootstrap, on peut en effet rendre ses applications responsive en créant des règles de style pour, par exemple ajuster la taille d'un composant (image, bouton) aux dimensions de l'appareil. On dispose également d'une unité de mesure, **dp**, qui permet de donner des dimensions précises tout en s'adaptant à la taille de l'écran. On préfère l'utiliser plutôt que **px**.

Android permet aussi de s'adapter à la langue de l'utilisateur. En effet, nous verrons que le SDK permet de stocker des variables (textes, couleurs) dans des fichiers séparés que l'on peut réutiliser avec des identifiants. Ainsi, tous les textes sont contenus dans un fichier **string.xml** et ne sont pas en dur dans le code source. C'est très utile si on veut par exemple

déployer l'application dans plusieurs pays, on aura alors plusieurs fichiers **string_fr.xml, string_en.xml** qu'on pourra choisir dans l'application en fonction de la langue de l'utilisateur.

Enfin, dans le cas de notre projet, l'application s'adapte à un public d'utilisateurs malvoyants. Ils disposent d'une option permettant d'activer une assistance auditive utilisant la reconnaissance vocale pour avoir des informations sur le prochain cours de la semaine.

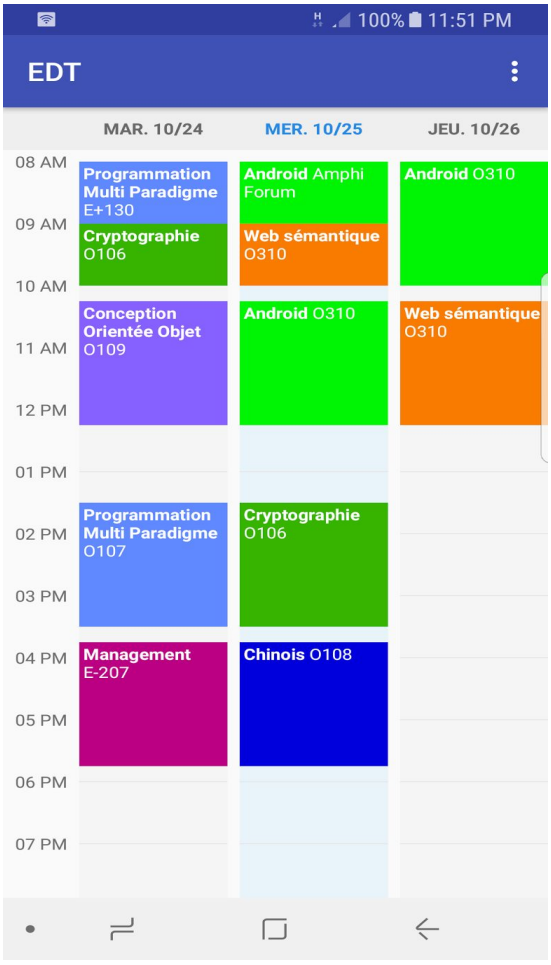
Réalisation

Si le développement des différentes fonctionnalités implique plusieurs classes et bibliothèques externes, toute l'application repose sur une seule activité.

Affichage des cours de la semaine

Comme j'ai voulu consacrer la plupart du développement à l'adaptation pour les utilisateurs malvoyants, j'ai utilisé un composant importé pour l'affichage des cours de la semaine.
On a donc un fichier **activity_week.xml** où l'on déclare un composant **WeekView**. Avant de remplir la vue avec une liste de cours, on utilise ce fichier xml pour définir diverses propriétés comme la taille de police, la hauteur d'une heure sur l'écran et les dimensions de la vue entière. Pour lui donner un aspect responsive, on fait en sorte qu'elle remplisse automatiquement l'écran avec la propriété "**match_parent"**

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/edtweek"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.example.david.edt.views.EDTWeekView
        android:id="@+id/weekView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:eventTextColor="@android:color/white"
        app:textSize="12sp"
        app:hourHeight="60dp"
        app:headerColumnPadding="8dp"
        app:headerColumnTextColor="#8f000000"
        app:headerRowPadding="12dp"
        app:columnGap="8dp"
        app:noOfVisibleDays="3"
        app:headerRowBackgroundColor="#ffefefef"
        app:dayBackgroundColor="#05000000"
        app:todayBackgroundColor="#1848adff"
        app:headerColumnBackground="#ffffffff"/>

</LinearLayout>
```



On définit ensuite la logique de l'activité dans le fichier **WeekActivity.java**. L'initialisation de l'emploi du temps se fait dés que l'activité est créée. Pour cela, on peut utiliser la méthode **onCreate**, qui s'exécutera automatiquement au lancement d'une activité. Après avoir récupéré le composant WeekView par son identifiant, on y charge donc la liste des cours. On doit aussi gérer l'interaction de clic long, au cas où

l'utilisateur a activé l'assistance auditive.

```
private void initMonth(){
    weekView.goToHour(8.00);
    weekView.setHourHeight(144);
    weekView.setMonthChangeListener((newYear, newMonth) -> {
            ArrayList<WeekViewEvent> eventsMonth = new ArrayList<>();
            List<WeekViewEvent> allEvents = events.getEvents();
            for(int i = 0; i < allEvents.size(); i++)
                if(allEvents.get(i).getStartTime().get(Calendar.MONTH) == newMonth-1)
                    eventsMonth.add(allEvents.get(i));

            Calendar calendar = Calendar.getInstance();
            if(newMonth == (calendar.get(Calendar.MONTH)+1))
                weekView.goToToday();

            return eventsMonth;
    });

        weekView.setEmptyViewLongPressListener((time) -> {
                Log.v("LONGCLICK", "Long click : "+mVoice);
                //speakOut("Bonjour");
                if(mVoice)
                    listen();
        });
```

Pour récupérer le composant WeekView par son id dans le code source, on peut utiliser cette commande :

```
weekView = (EDTWeekView)  findViewById(R.id.weekView);
```

Maintenant que notre WeekView est associée à une variable, on peut non seulement charger les cours dedans mais aussi modifier ses propriétés à tout moment grâce à des méthodes dédiées. Ainsi, on se sert du code logique pour gérer le changement d'orientation de l'appareil: on doit en effet mettre à jour le nombre de jours affichés dans la vue en fonction de l'orientation, 3 si mode portrait, 7 si paysage. Pour cela, on peut encore utiliser la fonction **onCreate** qui est exécutée à chaque fois que l'appareil change d'orientation.

```java
public void checkOrientation(){
    Log.v("ORIENTATION", "CHECKING");
    Resources res = getResources();
    if(res.getConfiguration().orientation == Configuration.ORIENTATION_PORTRAIT) {
        weekView.setNumberOfVisibleDays(3);
        Log.v("ORIENTATION", "Portrait");
    }

    else if(res.getConfiguration().orientation == Configuration.ORIENTATION_LANDSCAPE) {
        weekView.setNumberOfVisibleDays(7);
        Log.v("ORIENTATION", "Landscape");
    }
}
```

Assistance Auditive

Pour que l'application soit correctement adaptée aux utilisateurs malvoyants, on souhaite qu'il puisse demander oralement le prochain cours et qu'il obtienne une réponse pertinente par synthèse vocale. Pour cela, on doit d'abord implémenter les mécanismes de reconnaissance et de synthèse vocale. Heureusement, le SDK d'Android donne déjà accès à des bibliothèques très puissantes pour gérer ces deux éléments. Il suffit simplement de les lier à l'activité et aux interactions de l'utilisateur.

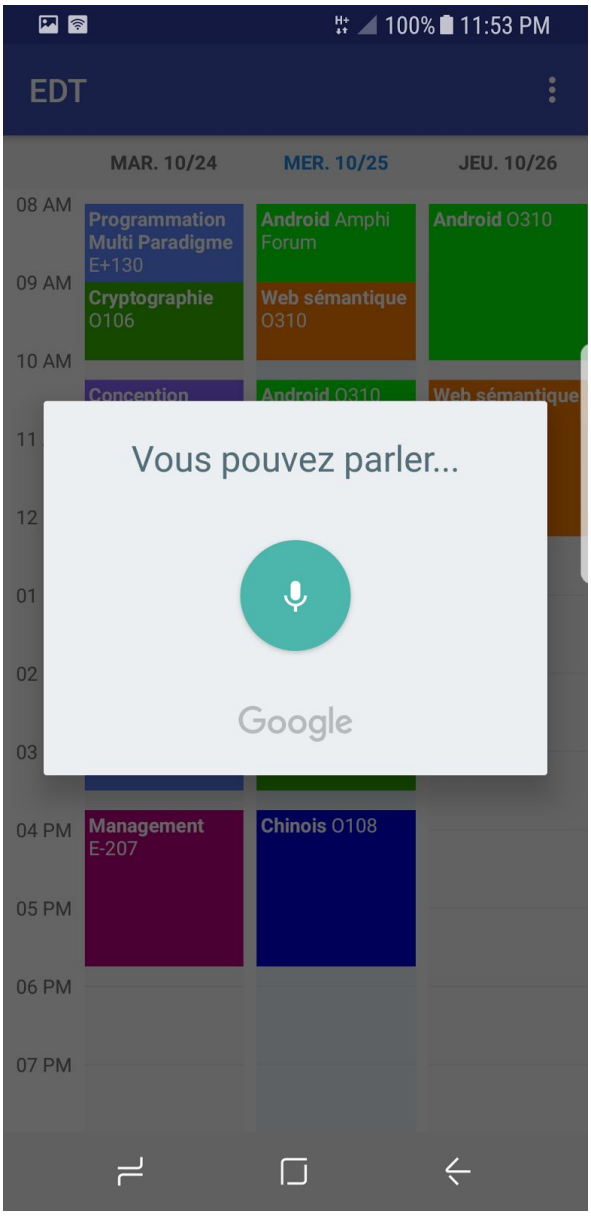
On doit donc d'abord utiliser la reconnaissance vocale dés que l'utilisateur maintient le doigt sur la vue. Une méthode est alors exécutée, et une nouvelle activité s'ouvre pour faire appel au service de reconnaissance vocale d'Android.

```java
private void listen(){
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault().getDisplayLanguage());
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Vous pouvez parler...");

    try{
        startActivityForResult(intent, REQ_CODE_SPEECH_INPUT);
    }

    catch(ActivityNotFoundException e){
        Toast.makeText(getApplicationContext(), "La reconnaissance vocale n'est pas supportée sur cet appareil", Toast.LENGTH_SHORT).show();
    }

}
```

On n'oublie pas de prévenir l'utilisateur si la reconnaissance vocale n'est pas disponible sur son appareil.

Une fois la fenêtre de dialogue ouverte, l'utilisateur peut parler devant le micro pendant un temps indéfini. Dés que le micro ne détecte plus de son distinguable, la reconnaissance vocale s'arrête et la fenêtre se ferme.

On revient alors à notre activité principale, où l'on peut récupérer le résultat de la reconnaissance vocale. Si un texte a bien été détecté par l'API de Google, on peut y accéder sous la forme d'une chaîne de caractères.



Le but maintenant est de voir si la séquence récupérée correspond à une opération particulière. Dans notre cas, la seule opération par reconnaissance vocale disponible pour l'instant est l'accès au prochain cours. On dispose d'une classe **VoicePattern** où l'on analyse un texte pour essayer de reconnaître une opération.
A chaque opération est associée une fonction de type booleen qui testera si le texte d'entrée correspond à une regex particulière.

Dans le cas de l'opération "prochain cours", on vérifie simplement si le texte contient la chaine "prochain cours". A l'avenir, cette classe peut faciliter l'ajout de nouvelles opérations dans l'application.

```
public class VoicePattern {

    private final String NEXT_COURSE = "prochain cours";

    public VoicePattern(){

    }

    public boolean isNextCours(String str) { return str.contains(NEXT_COURSE); }
}
```

La prochaine étape est la synthèse vocale. Là encore, on peut utiliser une API existante du SDK avec la classe **TextToSpeech**. Cette classe possède les méthodes nécessaires pour prononcer un texte en synthèse vocale et régler des paramètres comme le ton et la

fréquence de la voix. On doit d'abord gérer la synthèse vocale dans le cycle de vie de notre activité: on vérifie qu'elle est disponible au lancement dans la méthode **onStart**, et on n'oublie pas de la détruire à l'arrêt (**onStop**).

```
private void checkTTS(){
    Intent check = new Intent();
    check.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
    startActivityForResult(check, CHECK_CODE);
}
```

```
@Override
protected void onStop(){
    voiceSynth.shutdown();
    super.onStop();
}
```

Maintenant que la synthèse vocale est correctement configurée, on veut s'en servir lorsque l'utilisateur demande le prochain cours par reconnaissance vocale. On fait donc appel à la méthode **goToNextCourse** lorsque la requête contient "prochain cours". Cette méthode permet de rediriger la vue vers le prochain cours de la journée et, plus important, de décrire le cours par synthèse vocale.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data){
    super.onActivityResult(requestCode, resultCode, data);

    switch(requestCode){
        // .
        // .
        // .

        // Résultat reco vocale
        case REQ_CODE_SPEECH_INPUT:
            if(resultCode == RESULT_OK && null != data){
                ArrayList<String> result = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);

                //Contient "prochain cours"
                if(voicePattern.isNextCours(result.get(0)))
                    goToNextCourse();
                else
                    speakOut("Je n'ai pas compris, pouvez vous répéter svp?");
            }
            break;

        default:
            break;
    }
}
```

```
public String goToNextCourse(){
    Log.v("NEXTCOURSE","Next course");

    Calendar today = Calendar.getInstance();

    //weekView.setNumberOfVisibleDays(1);
    List<WeekViewEvent> weekViewEvents = events.getEvents();
    for(int i = 0; i < weekViewEvents.size(); i++) {
        WeekViewEvent event = weekViewEvents.get(i);
        //Log.v("COURSE", weekViewEvents.get(i).getName());
        if(today.compareTo(event.getStartTime()) < 0){
            Log.v("COURSE", event.getName() + " , " + event.getStartTime().get(Calendar.HOUR_OF_DAY));
            speakEvent(event);
            Calendar date = (Calendar) event.getStartTime().clone();
            weekView.goToDate(date);
            weekView.goToHour(event.getStartTime().get(Calendar.HOUR_OF_DAY));
            return event.getName();
        }
    }

    speakOut("Aucun cours prochainement");
    Log.v("COURSE", "Nothing found");
    return null;
}
```

Le composant WeekView permet de facilement rediriger la vue vers une journée et un horaire particulier avec les méthodes **goToDate** et **goToHour**. On parcourt la liste des cours enregistrés, qui a été préalablement triée par horaires croissants. Le premier cours se déroulant juste après l'heure actuelle est retenu comme le prochain cours. On redirige alors la vue et on le décrit par synthèse vocale avec la méthode **speakEvent**. Si on n'a rien trouvé, elle prévient l'utilisateur qu'aucun cours n'est prévu prochainement.

Les cours sont représentés par des objets **WeekViewEvent**, avec pour propriétés un nom, un lieu et une date précise. On veut que la synthèse vocale énonce ces informations automatiquement pour chaque évènement, tout en essayant de le faire de manière pertinente. Si par exemple un cours a lieu le jour même ou le lendemain, la voix prononcera "aujourd'hui" ou "demain" au lieu de "le 27 octobre".
Un cours sera donc par exemple décrit comme ceci:
Compilation demain à 8H, en E 107.

```
public void speakEvent(WeekViewEvent event){
    Calendar today = Calendar.getInstance();

    Calendar c = event.getStartTime();
    String name = event.getName();

    int min = c.get(Calendar.MINUTE);

    String day;
    if(c.get(Calendar.DAY_OF_MONTH) == today.get(Calendar.DAY_OF_MONTH))
        day = "aujourd'hui";
    else if(c.get(Calendar.DAY_OF_MONTH)== today.get(Calendar.DAY_OF_MONTH)+1)
        day = "demain";
    else
        day = JOURS[c.get(Calendar.DAY_OF_WEEK)-1]+" "+c.get(Calendar.DAY_OF_MONTH) + " " + MOIS[c.get(Calendar.MONTH)];

    String hour = c.get(Calendar.HOUR_OF_DAY)+ " heure " + (min != 0 ? min : "");
    String location = event.getLocation();

    speakOut(name+ ", " + day + " à "+hour+", en "+location);
}
```

Menu d'options

Comme l'assistance auditive est un paramètre et n'est pas activée par défaut au premier lancement de l'application, on veut pouvoir le modifier dans un menu d'options. Le SDK d'android permet à la manière des activités d'implémenter très simplement des menus déroulants qui peuvent être intégrés à une activité dans la barre d'action en-haut de l'écran. On doit d'abord créer un fichier **menu.xml** dans le répertoire **res/menu** du projet. Ici, on peut décrire notre menu déroulant et ses options comme pour les composants d'une activité, avec des identifiants, règles de style etc...

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <item android:id="@+id/exit"
        android:title="Quitter"/>

    <item android:id="@+id/checkAudioItem"
        android:title="Assistance auditive"
        android:icon="@mipmap/mic"
        android:checkable="true"/>

</menu>
```

On a donc deux options, une pour quitter l'application et une pour activer/desactiver l'assistance auditive. On précise que cette option est cochable par l'utilisateur. Cependant, si l'utilisateur sélectionne l'option dans le menu, elle ne sera pas cochée/décochée automatiquement, on doit l'implémenter dans le code logique. On doit donc inclure le menu dans notre activité au lancement dans **onCreateOptionMenu**. Lorsque le menu est initialisé, on récupère d'abord les paramètres de configuration de l'application. Pour savoir si l'assistance auditive est activée, on a un booléen qui est automatiquement enregistré à chaque modification, ce qui permet de le réutiliser après avoir relancé l'application.

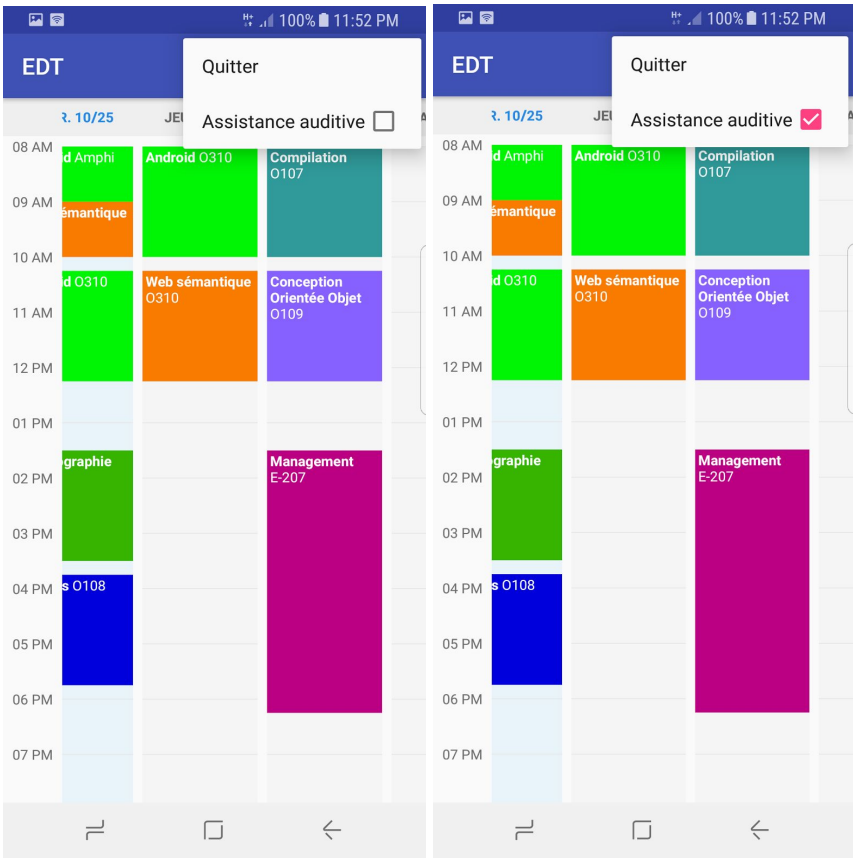
```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    getMenuInflater().inflate(R.menu.menu, menu);

    boolean v = mSettings.getBoolean("vocal_mode", false);
    menu.findItem(R.id.checkAudioItem).setChecked(v);

    return true;
}
```

On doit ensuite gérer les évènements du menu: fermer l'application si on a choisi "quitter", activer/désactiver la reconnaissance vocale si l'assistance auditive est sélectionnée. On peut récupérer les options par leur identifiants comme pour les composants d'activité, il suffit

ensuite d'adapter le comportement de l'application.

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()){
        case R.id.exit:
            finish();
            System.exit(0);

        case R.id.checkAudioItem:
            item.setChecked(!item.isChecked());
             setVoice(item.isChecked());
             break;

    }

    return super.onOptionsItemSelected(item);
}
```

setVoice permet d'enregistrer le paramètre d'assistance auditive dans le fichier de configuration. On réutilise ce paramètre lors d'un long clic pour savoir s'il faut ou non lancer la reconnaissance vocale.



React Native (Vincent BOUCHER-THOUVENY)

Description et installation

React Native est un cadriciel développé par Facebook qui permet de créer des applications natives dans plusieurs plateformes avec une seule base de code: le JSX, un mélange de Javascript et de XML.

Pour développer dans cet environnement il faut se munir d'un environnement Node et d'un IDE adapté tel que IntelliJ Ultimate ou Webstorm.

Nouveau projet

Lors de la création d'un nouveau projet on fait :

```
npm i -g react-native-cli
```

Puis on exécute ce qui suit, qui crée un template par défaut et installe les dépendances :

```
react-native init MyApp
```

Projet en cours

Lorsqu'on se trouve avec un projet déjà initialisé on installe les dépendances avec :

```
npm i
```

Utilisation et tests

L'utilisation de React Native permet de tester son code rapidement à l'aide d'un packager qui s'occupe de transpiler le code avant de l'envoyer dans l'application qui est prête à recevoir le code à exécuter. Ainsi pour Android et iOS par exemple, nul besoin de réinstaller l'apk ou l'ipa, le nouvelle version est installées instantanément.

Comme un projet React Native est un projet Node, on peut bénéficier de l'utilisation des scripts NPM. Voici un exemple d'une liste de scripts qui se trouvent dans le fichier package.json qui aident au développement RN :

```
"scripts": {
    "start": "node node_modules/react-native/local-cli/cli.js start",
    "adb-shake": "adb shell input keyevent 82",
    "adb-packager": "adb reverse tcp:8081 tcp:8081",
    "adb-backend": "adb reverse tcp:8082 tcp:8082",
    "android": "cd android && gradlew clean && cd .. && node node_modules/react-native/local-cli/cli.js run-android",
    "android-clean": "cd android && gradlew clean",
    "android-build": "cd android && gradlew clean && gradlew assembleRelease",
    "android-debug": "cd android && gradlew clean && gradlew installDebug",
    "android-log": "node node_modules/react-native/local-cli/cli.js log-android",
    "android-install": "cd android && gradlew clean && gradlew installRelease",
    "ios": "node node_modules/react-native/local-cli/cli.js run-ios",
    "web": "cross-env NODE_ENV=development webpack-dev-server --progress --colors --inline",
    "web-build": "cross-env NODE_ENV=production webpack",
    "web-build-watch": "cross-env NODE_ENV=production webpack --progress --colors --watch",
    "web-serve": "cross-env NODE_ENV=production forever start --no-colors -c http-server web",
    "web-serve-stop": "forever stopall --no-colors",
    "test": "jest"
},
```

Ainsi le projet est prêt à être lancé selon les configurations supportées :

```
npm run android|ios|web
```

Vous voici en présence d'un superbe emploi du temps adapté à la largeur de votre plateforme ! Sur mobile, l'affichage par jour vous est proposé, alors que sur des écrans plus larges vous aurez plutôt un affichage semaine par semaine.

Le développement a été fait à partir de Google Chrome et d'un Samsung Galaxy S6 edge et un S8. Par navigateur, le développement est plus rapide, il a donc été plus facile de tester d'abord sur navigateur puis sur mobile.

Réalisation

Responsivité

Au plus haut de l'application on définit ce que signifie "être sur mobile", ce qui au final signifie plutôt "avoir un écran trop petit pour l'affichage semaine par semaine.

```
const isMobile = () => Dimensions.get('window').width <= 800;
```

Lors du rendering de l'application alors on pose les conditions :

```
export default class App extends Component<{}> {

  state = {
    isMobile: isMobile(),
  };

  componentDidMount() {...}

  componentWillUnmount() {...}

  _updateViewport = () => {...};

  render() {
    return (
      <View style={{flex: 1, backgroundColor: '#fff'}}>
        <View style={{height: 56, backgroundColor: '#05B3F0', flexDirection: 'row', alignItems: 'center'}}>
          <Text style={{fontFamily: 'Lato', color: '#fff', fontSize: 20, marginLeft: 20}}>
            {'Emploi du temps ' + (this.state.isMobile ? 'par jour' : 'par semaine')}
          </Text>
        </View>
        {this.state.isMobile ? (
          <DayViewer
            items={ITEMS}/>
        ) : (
          <WeekViewer
            items={ITEMS}/>
        )}
      </View>
    );
  }
}
```

On voit donc bien que l'application utilisera soit un DayViewer, soit un WeekViewer, si l'écran est considéré comme "mobile".

Visualisation par jour

Un DayViewer est un composant déclaré tel que suit :

```
export default class DayViewer extends Component {

  static propTypes = {
    items: PropTypes.arrayOf(PropTypes.shape({"duration": PropTypes.number.isRequired...})),
  };

  state = {
    currentDate: new Date(),
    dayOffset: 0,
  };

  _incrementOffset = (increment) => {
    this.setState({dayOffset: this.state.dayOffset + increment});
  };

  render() {
    let activeDay = new Date(+this.state.currentDate + 86400000 * this.state.dayOffset);
    let dayString = activeDay.toJSON().slice(0, 10);
    return (
      <View style={{flex: 1}}>
        <View style={{"backgroundColor": '#05B3F0'...}}>
          <TouchableOpacity/>
          <Text style={{fontFamily: 'Lato', color: '#fff', fontSize: 22}}>
            {dayString}
          </Text>
          <TouchableOpacity/>
        </View>
        <ScrollView style={{flex: 1}}>
          <Day
            date={activeDay}
            items={this.props.items}/>
        </ScrollView>
      </View>
    );
  }

}
```

Des items lui sont passés en propriétés et il garde en mémoire le date initiale ainsi que le décalage effectué avec les flèches de l'interface.

Visualisation par semaine

Un WeekViewer est un composant déclaré tel que suit :

```
export default class WeekViewer extends Component {

  static propTypes = {
    items: PropTypes.arrayOf(PropTypes.shape({"duration": PropTypes.number.isRequired...})),
  };

  state = {
    currentDate: getMonday(new Date()),
    weekOffset: 0,
  };

  _incrementOffset = (increment) => {
    this.setState({weekOffset: this.state.weekOffset + increment});
  };

  render() {
    let activeWeek = new Date(+this.state.currentDate + 86400000 * 7 * this.state.weekOffset);
    let currentWeekdaysAsNumber = new Array(7).fill(0).map((v, i) => i);
    let weekNumber = getWeekNumber(activeWeek);
    return (
      <View style={{flex: 1}}>
        <View style={{"backgroundColor": '#05B3F0'...}}>
          <TouchableOpacity/>
          <Text/>
          <TouchableOpacity/>
        </View>
        <View style={{flexDirection: 'row'}}>
          {dayNames.map(name => (
            <View key={name} style={{"alignItems": 'center'...}}>
              <Text style={{fontFamily: 'Lato', color: '#666', fontSize: 15}}>
                {name}
              </Text>
            </View>
          ))}
        </View>
        <ScrollView style={{flex: 1}}>
          <View style={{flex: 1, flexDirection: 'row'}}>
            {currentWeekdaysAsNumber.map((v, i) => (
              <View
                key={i}
                style={{flex: 1}}>
                <Day/>
              </View>
            ))}
          </View>
        </ScrollView>
      </View>
    );
  }
```

Des items lui sont passés en propriétés et il garde en mémoire le lundi de la semaine actuelle pour se repérer et également le décalage par semaine effectué avec les flèches de l’interface.

Composants globaux

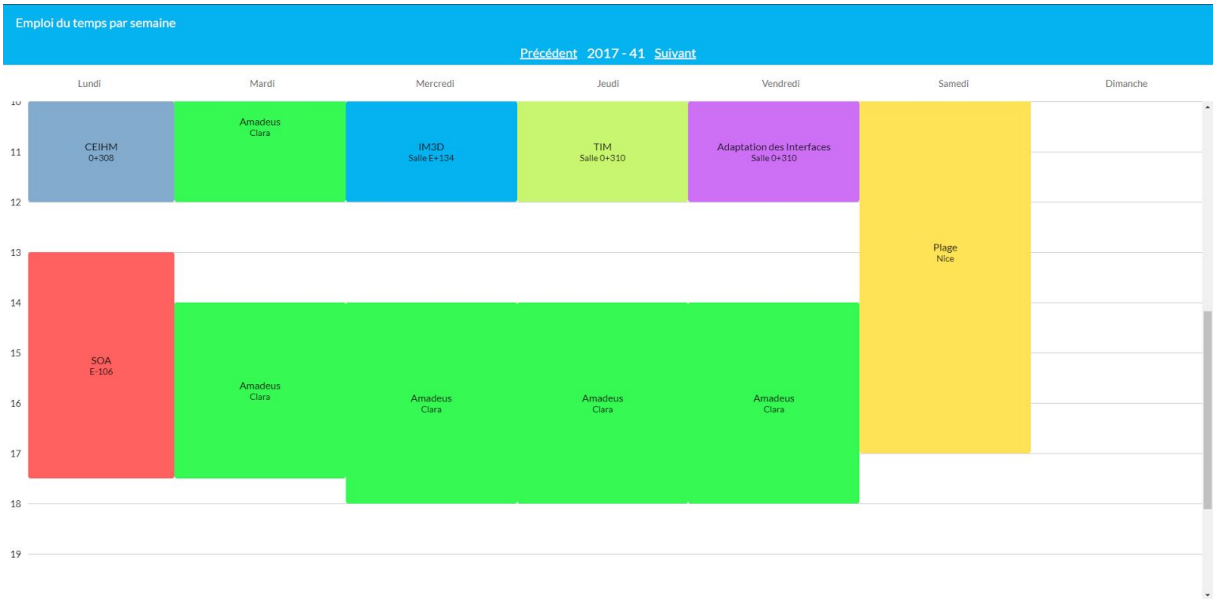
Un Day se définit comme suit :

```
export default class Day extends Component {

  static propTypes = {
    showIndicators: PropTypes.bool,
    date: PropTypes.instanceOf(Date).isRequired,
    items: PropTypes.arrayOf(PropTypes.shape({"duration": PropTypes.number.isRequired...})),
  };

  static defaultProps = {
    showIndicators: true,
    items: [],
  };

  render() {
    let currentDayHours = new Array(24).fill(0).map((v, i) => i);
    let dayHeight = 60;
    let hourIndicatorWidth = 40;
    return (
      <View style={{position: 'relative', flexDirection: 'column', height: dayHeight * currentDayHours.length}}>
        {currentDayHours.map((v, i) => {...})}
        {this.props.items.map(item => {
          if (new Date(item.start).setHours(0, 0, 0, 0) !== new Date(this.props.date).setHours(0, 0, 0, 0)) {
            return;
          }
          return (
            <View key={item.start} style={{"position": 'absolute'...}}>
              <Text/>
              <Text/>
            </View>
          );
        })}
      </View>
    );
  }

}
```

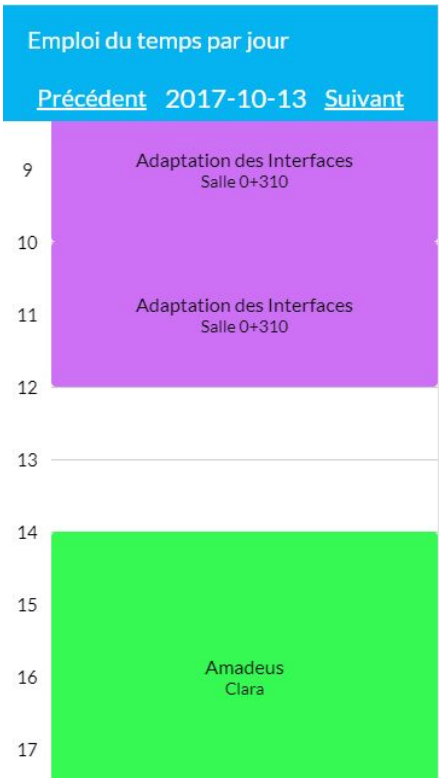
Il affiche ou non les indicateurs par heure sur le côté, possède la date qu'il doit afficher, ainsi que la liste des objets à afficher.

Affichage

Pour écran de largeur supérieure à 800 px, on a :



Pour écran inférieur à cette largeur, on a :



Conclusion

Le développement de l'EDT avec ces 4 technologies nous permet d'observer les bénéfices et les limites de celles-ci liées à l'adaptation.

Bootstrap permet une adaptation selon le dispositif efficace et simple d'utilisation, avec de nombreux avantages:

- Facile à mettre en place et facile à utiliser, bootstrap fonctionne avec les classes html, il suffit d'utiliser les classes proposées par bootstrap au sein du html (on n'a pas besoin d'utiliser de JS ou autre langage)
- Système de grille à 12 colonnes permet la plupart du temps d'adapter l'affichage à différentes tailles de dispositif. De plus les lignes et colonnes peuvent être imbriquées ce qui permet de découper encore plus si nécessaire.
- Bootstrap permet de n'afficher des éléments que pour certaines tailles d'écran, on peut donc avoir un html complètement différent si on est sur mobile ou sur PC par exemple, ce qui augmente les possibilités d'adaptation.

Mais aussi des inconvénients:

- Lorsqu'on a besoin d'avoir un affichage différent et que la grille bootstrap ne suffit pas, on peut faire comme décrit précédemment (du HTML qui n'est affiché que pour un certain type d'écran). Le problème est qu'on a affaire à un code lourd avec de la duplication.
- Pas de réutilisabilité contrairement à Angular/React qui permettent avec les composants d'avoir des éléments communs aux différentes interfaces et qu'on peut réutiliser dans les deux.
- Bootstrap est complet mais est aussi assez lourd, surtout si on ne souhaite pas utiliser les "components" bootstrap mais uniquement les layout (grille)

En ce qui concerne **Angular**, l'approche par Web components présente des avantages particuliers:

- Grande liberté sur la création d'une page
- Facilité à créer et gérer des pages qui sans angular aurait été un enfer: Dans le cadre d'une page utilisant des dizaines de composants qui sont affichés ou non selon certaines conditions, cela impliquerait sur une page html classique, d'avoir du code html qui potentiellement ne servirait pas et du code javascript long et compliqué.
- Découplage fort lié au MVC, ce qui permet d'avoir du code plus facilement maintenable
- Possibilité de réutiliser des composants pour différentes IHM (version mobile, version PC) ce qui permet d'avoir un code plus lisible et d'éviter la duplication de code dans certaines situations.

Néanmoins, utilisé seul, il présente aussi des inconvénients:

- Code lourd et répétitif à écrire dans le cas où l'on souhaite juste modifier l'affichage d'éléments présent dans deux vues différentes, se tourner vers une solution bootstrap semble ici plus approprié.
- AngularJS étant un framework, nous avons naturellement des pertes de performances comparé à du JavaScript pure.

Pour palier au problème du code lourd sous certaines conditions, on peut utiliser bootstrap avec AngularJS afin d'éviter de créer des composants qui ont juste un css différent.

Avec **Android natif**, on peut profiter d'un ensemble d'outils pour adapter facilement son applications à différents environnements et types d'utilisateurs

Points positifs :

- SDK et IDE très complets, permettant grâce à modèles d'activités de créer très rapidement un prototype d'application.

- Description des activités dans un fichier séparé du code métier, cela permet un code plus lisible et respectant le pattern MVC.

- Possibilité de créer soi-même de nouveaux composants, ce qui laisse une liberté créative conséquente au développeur.

- Accès aux différents capteurs (micro, caméra) très simple grâce à des API spéciales fournies par le SDK.

Points négatifs :

- Ne concerne que les appareils Android. Si la plupart des smartphones et objets connectés tournent aujourd'hui sous Android, iOS occupe également une part non négligeable du marché, ce qui réduit le nombre de clients potentiels.

- Installation lourde et parfois longue de l'environnement. Cela dépend grandement de votre OS et de sa configuration, mais entre l'installation du SDK, de l'IDE et de l'outil d'émulation d'Android, il faut compter au moins 2GO de téléchargement et d'installation sur son appareil.

- Contraintes liées à la version d'Android visée. On ne peut pas publier simplement une application pour tous les appareils existants, on doit définir une version d'android minimale requise pour utiliser certaines API et, pour des fonctionnalités plus récentes, cela implique qu'un public moins large sera visé.

- Mieux vaut tester l'application sur un appareil physique que sur une machine virtuelle, qui sera plus faible en performances et donc donnera des résultats moins pertinents.

React Native possède égalements ses avantages et ses inconvénients :

Avantages :

- Une seule base de code suffit pour atteindre de multiples plateformes.
- Le testing est très rapide grâce au système de “packaging” qui fait que l’on n’a pas a réinstaller l’application à chaque modification.
- Chaque composant possède en un seul fichier: son interface, son style, ses fonctions métiers et ses données métiers.
- Le pattern Flow inventé par Facebook qui permet d’assurer que l’application sera fluide et optimisée.

Inconvénients :

- Le socle d’un projet standard destiné à la publication est significatif et son installation peut faire perdre de précieuses minutes.
- L’affichage sur les différents environnements peut varier légèrement selon la plateforme.
- Les composants qu’il est impossible d’implémenter en Javascript sont pontés et donc nécessitent un travail de mise en place en plus.