Rapport Adaptation des interfaces

Groupe 6 : projet E-commerce

TABLE DES MATIÈRES

Introduction

L'exemple choisi est un site d'E-commerce proposant à ses clients un catalogue de jeux vidéos. Disposant de nombreux articles, il est essentiel que la navigation se fasse le plus facilement et intuitivement possible pour assurer le confort de l'utilisateur, et ce quel que soit le dispositif utilisé. En effet, une application conçue uniquement pour une seule plateforme s'avère souvent inutilisable sur d'autres supports à cause de son interface qui ne parvient pas à s'adapter.

L'application que nous avons choisie va nous permettre de proposer des adaptations principalement centrées sur la taille de l'écran grâce à plusieurs technologies (Angular 2, Bootstrap, Foundation et Xamarin). Dans notre cas, le catalogue de jeux vidéos devra donc être affiché de manière pertinente selon le dispositif et sa taille pour faciliter sa lecture. De plus, les menus de l'applications devront également s'adapter pour éviter de rendre la navigation contraignante. Enfin, la fiche détaillée de chaque jeu devra proposer un contenu en adéquation avec la taille de l'écran.

Tutoriel application Angular 2

Prérequis et installation

Installez la dernière version de NodeJS si ce n'est pas déjà fait (4.6.x) https://nodejs.org/en/download/ (vous pouvez vérifier la version avec la commande **node –v**)

Assurez-vous d'avoir npm en version 3.10.x . Pour vérifier votre version de npm, vous pouvez utiliser la commande **npm –v**. Pour mettre à jour npm, utilisez la commande **npm install npm@latest -g**

Clonez le répertoire du projet hébergé sur github.com avec la commande **git clone https://github.com/cc83/Adaptation_IHM**

Si vous n'avez pas git, vous pouvez le télécharger ici : https://git-scm.com/

Avec un terminal, déplacez-vous à la racine du projet, puis lancez l'installation avec la commande **npm install** (cela peut prendre un certain temps, vérifiez également que vous disposez d'Internet durant l'installation).

Une fois l'installation terminée, vous pouvez désormais utiliser la commande **npm start** depuis la racine du projet. Cela a pour effet de lancer le serveur Angular 2 et ouvrira une page de navigateur Internet contenant la première page Web du projet.

Informations complémentaires

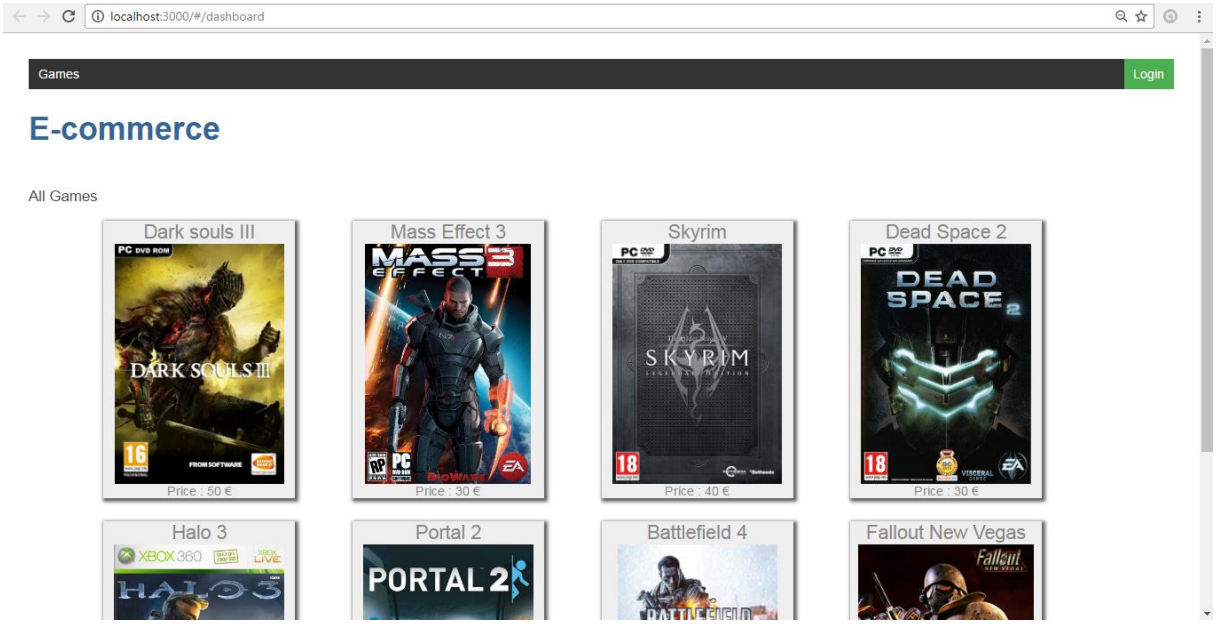
Ce projet est basé sur le tutoriel Angular 2 mis en place sur le site officiel : https://angular.io/docs/ts/latest/quickstart.html . J'ai ensuite amélioré cet exemple grâce à de nombreuses recherches sur les possibilités offertes par Angular 2 en terme d'adaptation au dispositif et de réutilisation des composants.

Le contenu fonctionnel du projet a été dans l'ensemble simplifié pour se concentrer sur l'essentiel, à savoir l'adaptation de l'application au dispositif (taille de l'écran) ainsi que la réutilisation des composants dans des contextes différents.

Pour ce projet, j'ai utilisé l'éditeur de texte Atom, celui offrant une autocomplétion très pratique en css ainsi qu'en Typescript, le langage utilisé pour définir les composants. Il permet également de détecter les erreurs Typescript.

Test des capacités d'adaptation à l'écran

Si tout s'est bien passé, votre navigateur a dû s'ouvrir après la commande **npm start**, et affiche désormais une page Web de ce type :

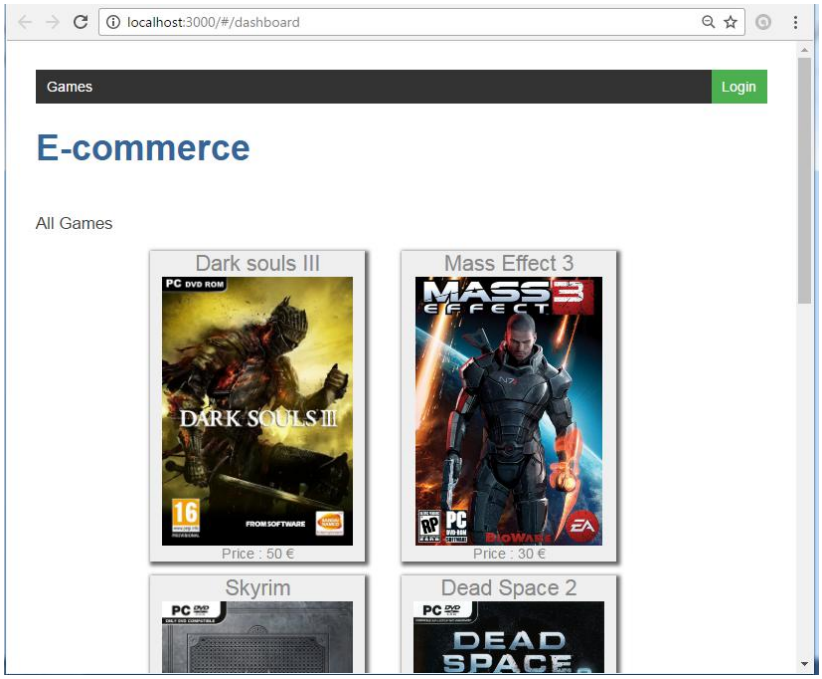


Cet affichage sous forme de grandes vignettes des jeux est disponible lorsque la taille de l'écran est suffisamment large. Ici chaque jeu est un composant Game dont le style d'affichage a été déterminé via le CSS interne au composant, une caractéristique très pratique d'Angular 2. Ainsi, l'affichage du jeu ne dépend que de lui-même et ne sera pas pollué par du CSS extérieur.

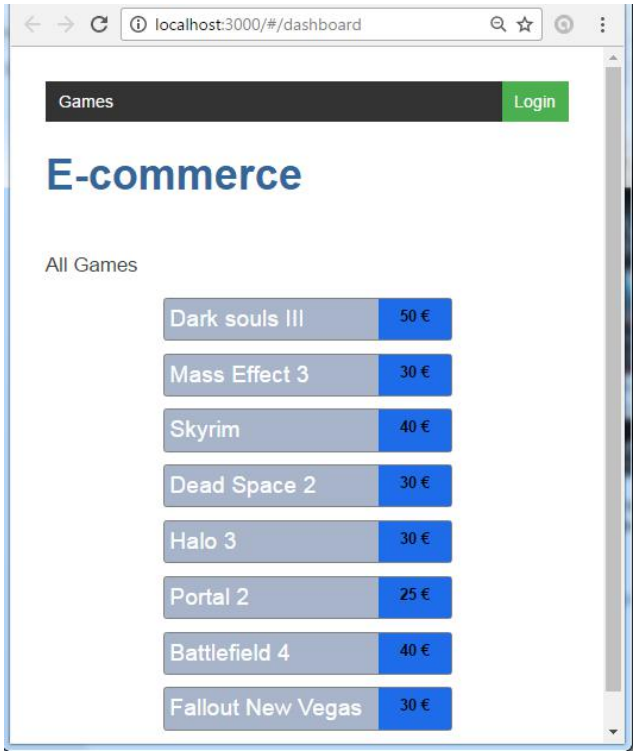
Tous ces composants Game sont contenus dans un composant Dashboard qui se contente donc de les afficher comme bon lui semble. De plus, la liste des jeux est un paramètre d'entrée du composant : ainsi il est tout à fait possible d'afficher d'autres jeux en fournissant une autre liste de Game au Dashboard. Par défaut, la liste des jeux est récupérée via un service GameService, rendant complètement indépendante la manière dont les jeux sont stockés et assurant là encore une réutilisation de ce service.

Enfin, un dernier composant est visible sur cette page : la barre des menus TopMenu située tout en haut de la page. Celle-ci dispose simplement d'un raccourci vers la page principale et permet également de se connecter via un composant Login dont nous reparlerons plus tard.

Redimensionnez la fenêtre à l'aide des capacités de votre navigateur ou simplement à la main, le composant TopMenu s'adapte à la largeur de l'écran. Les composants Game se resserrent.

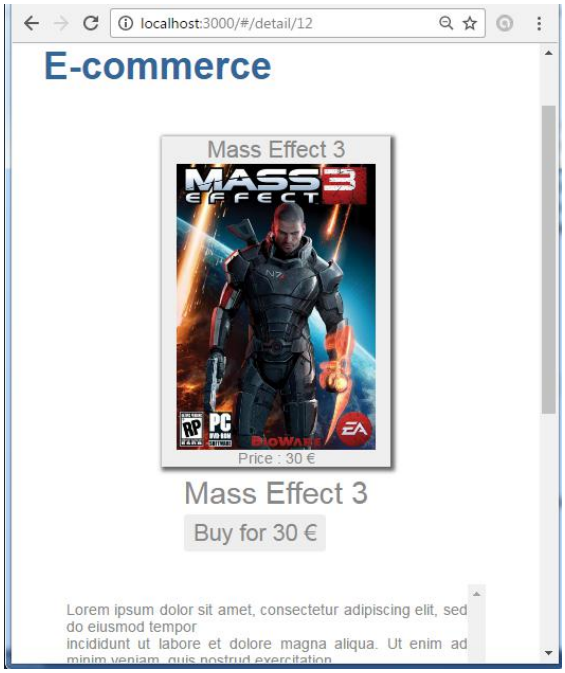


En rétrécissant encore la largeur de la fenêtre, on remarque que l'affichage change : les composants Game sont désormais affichés sous forme de liste et leur style a changé pour mieux s'adapter :



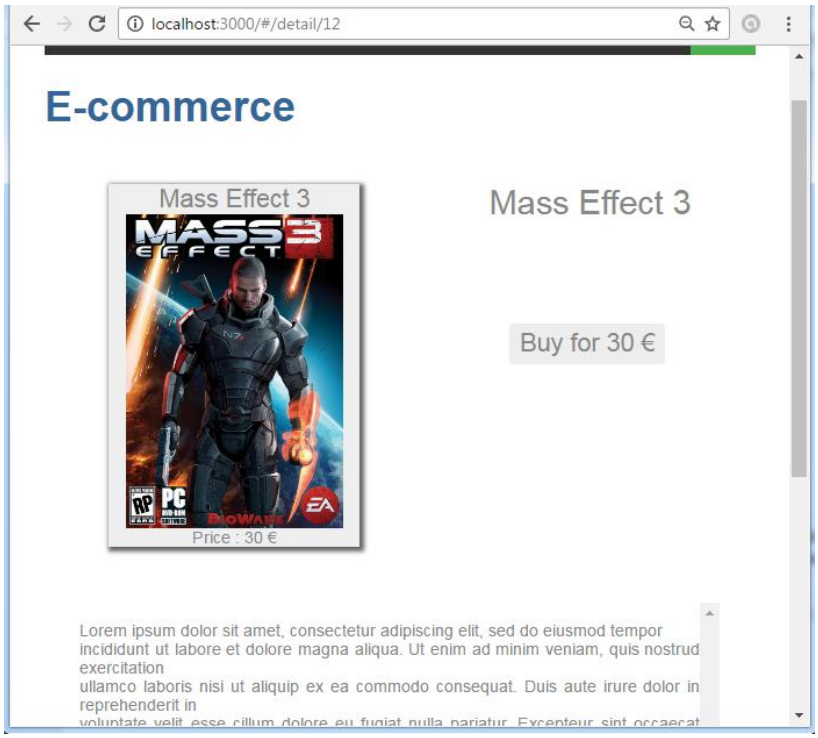
Ce changement de style des composants Game est rendu possible non pas grâce à l'utilisation des Media Queries de CSS3 mais grâce à un module Angular 2 (ng2-responsive). Ce module permet de détecter la taille de l'écran et de modifier l'affichage d'un composant. Ici, c'est le composant Dashboard qui détecte la taille réduite de l'écran et choisit d'une part de passer à un affichage en mode liste à une seule colonne, mais il donne également en paramètre aux composants Game une certaine valeur qui permet de modifier leur affichage pour ne plus avoir d'images.

Quel que soit l'affichage des composants Game, il est toujours possible de cliquer dessus. Cela redirige vers une page affichant les détails du jeu via le composant GameDetail. Avec la même taille d'écran que précédemment on obtient la page Web suivante :



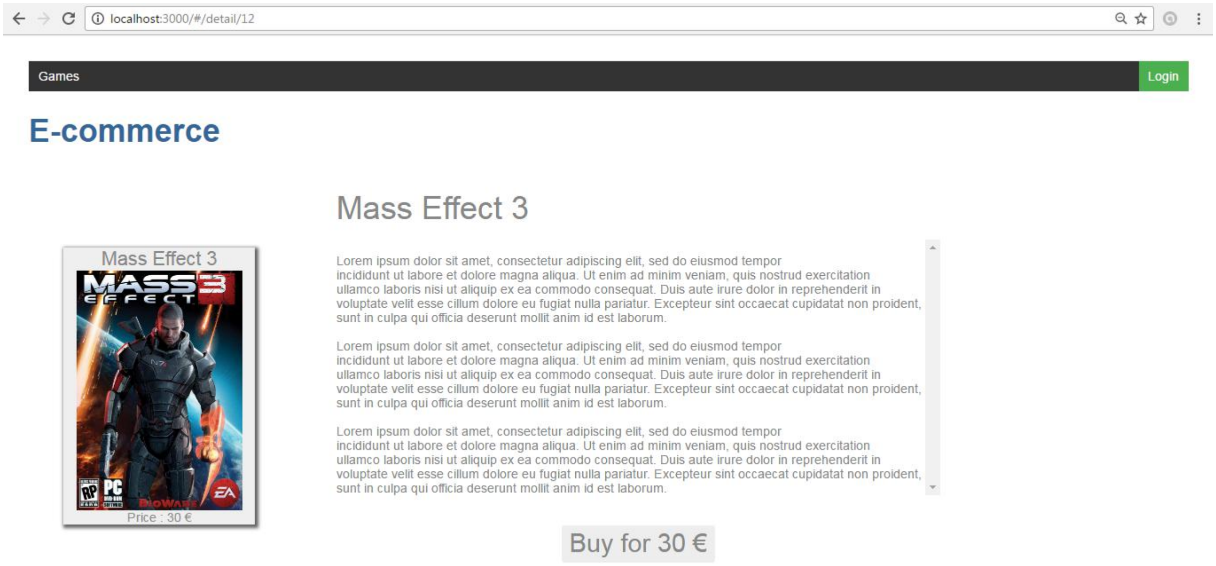
On remarque donc qu'on a réutilisé l'affichage d'un composant Game avec sa vignette. De plus, les informations supplémentaires du jeu sont affichées en dessous (notamment le bouton pour acheter et la description).

Lorsqu'on agrandit un peu la largeur de la fenêtre, on passe à un affichage en mode tablette :



Désormais, le bouton pour acheter le jeu et le titre se retrouvent à droite de la vignette, tandis que la description reste en dessous.

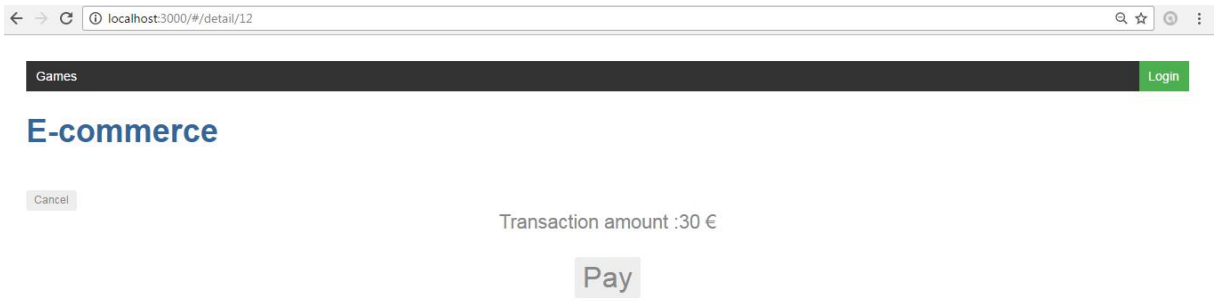
Enfin, en agrandissant encore la fenêtre pour arriver à un affichage Desktop, on obtient la page suivante :



La grande largeur de l'écran nous permet d'afficher directement la description à droite de la vignette, contrairement au mode tablette.

Ces trois versions de l'affichage ont été réalisées grâce au module responsive d'Angular 2 en le combinant avec les Media Queries de CSS3. En effet, le module permet de restructurer la page Web en fonction de la taille de l'écran ce qui affectera directement le DOM, tandis que CSS ne permet que de modifier le style. Dans ce cas précis, je me suis donc servi du module Angular pour changer la disposition des balises lors d'un affichage Desktop ou Tablette/Téléphone (en sortant la description du jeu lorsque l'écran est trop petit pour la mettre en dessous), ceci rendant par la suite le CSS des Media Queries beaucoup plus simple. Cela a donc permis de rendre le composant GameDetail responsive.

On peut ensuite choisir d'acheter le jeu (l'achat ne sera pas pris en compte). Cela permettra d'afficher le composant Buy .



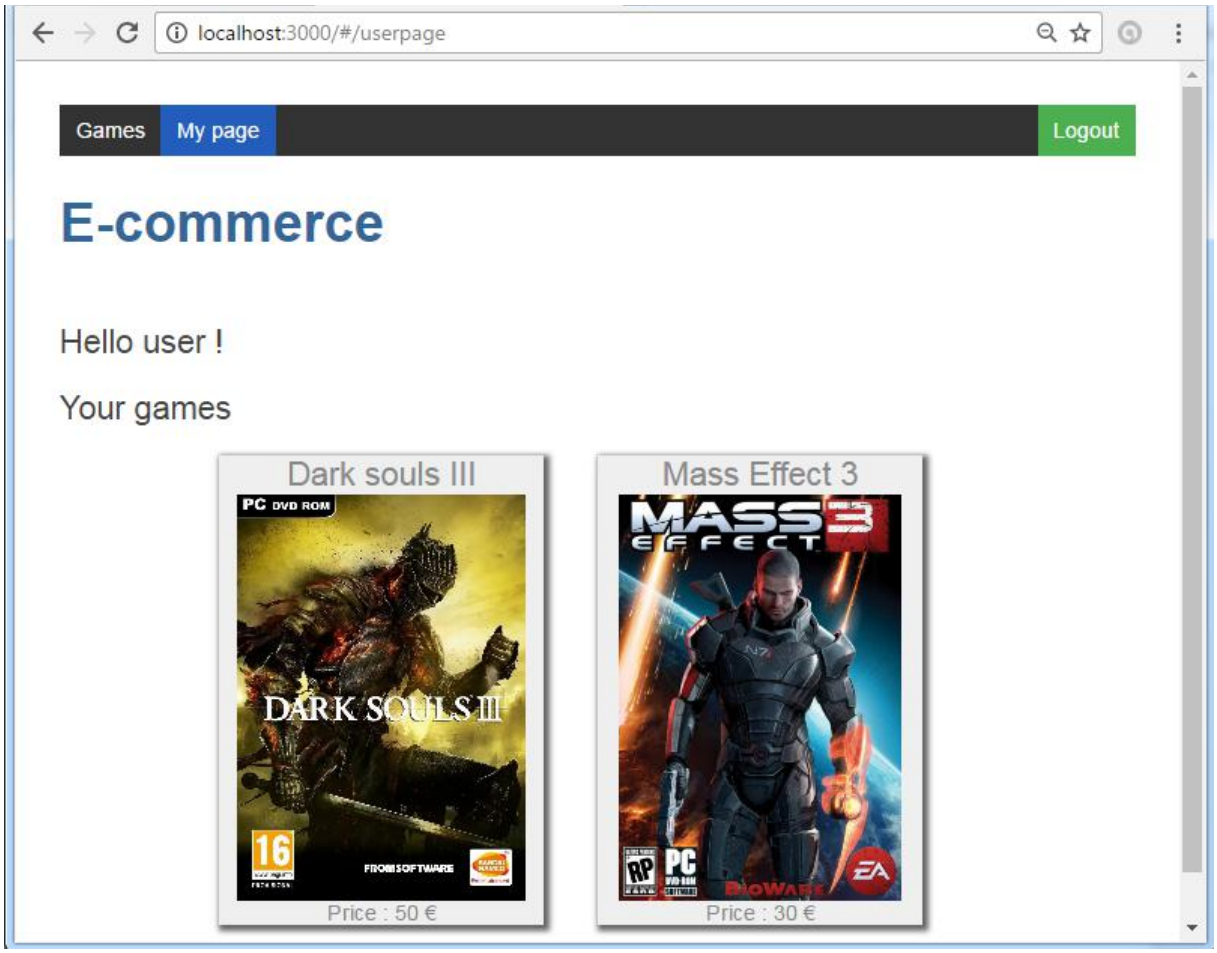
Celui-ci est très simple puisqu'il ne dispose que d'une entrée (le prix) et une sortie (la validation du paiement). Evidemment on pourrait imaginer un composant demandant un numéro de carte bleue ou autre et effectuant une vérification avec un service bancaire, faute de temps cela n'a pas été possible. Cette simplicité permet à ce composant d'être fortement réutilisable pour n'importe quelle application : il suffit simplement d'indiquer le montant de la transaction et il se chargera du reste tout en renvoyant une valeur permettant de savoir si la transaction s'est bien déroulée ou non. Enfin, son CSS interne lui permet également de s'adapter à la largeur de l'écran.

Enfin, revenons maintenant à l'accueil. Il est possible de se connecter à l'application via le bouton login situé à droite de la barre des menus. Cela affichera ensuite le composant Login (dont le fond a été colorié en vert):

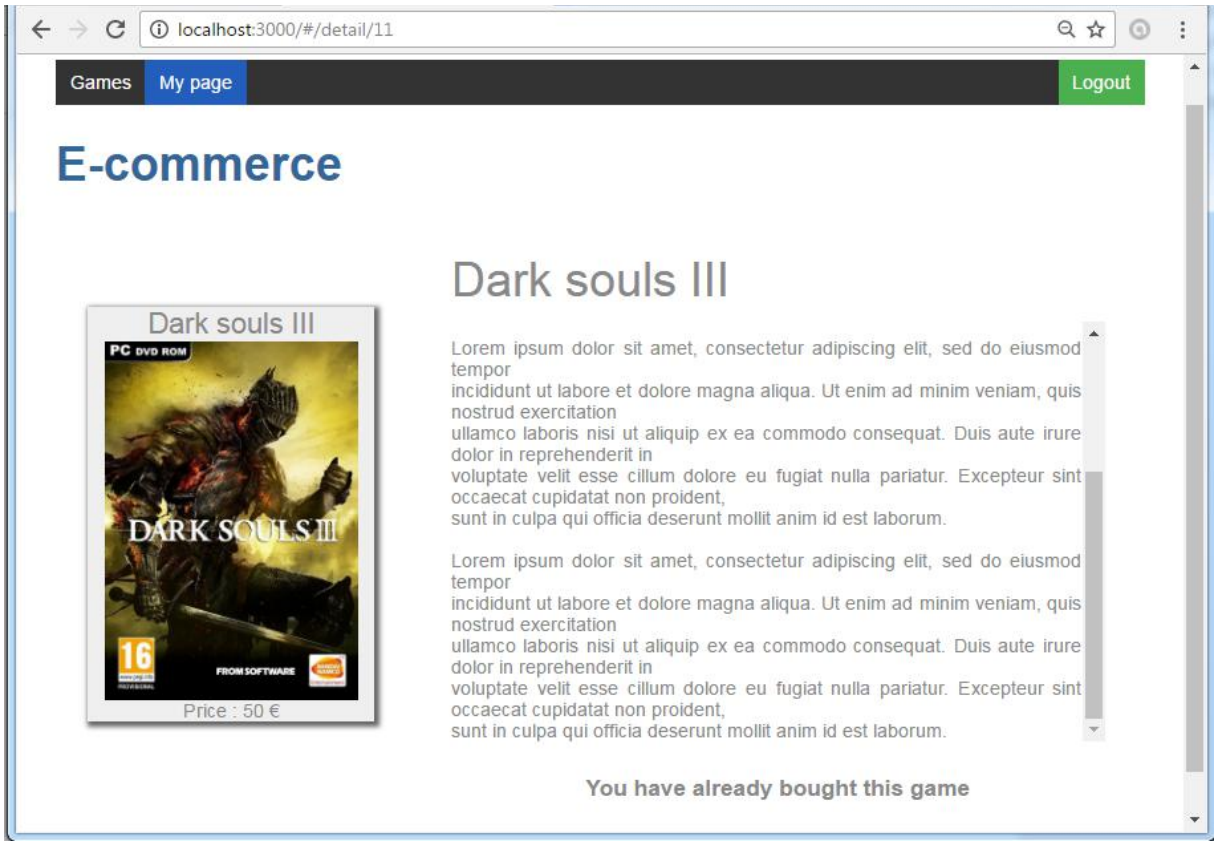
Le composant Login est également tout à fait réutilisable : celui-ci prend en entrée un login et un mot de passe puis il demande à un service si la combinaison est valide après un clic sur le bouton Send. La seule chose à changer serait le service à appeler.

Pour cette version, il n'y a pas d'utilisateur : un simple clic sur Send permet de se connecter. Les sessions ont ici simplement été modélisées par un service global commun à tous les composants par souci de temps, mais il existe des modules Angular 2 permettant de travailler avec de véritables sessions (voir auth0). Une fois cela fait, un nouvel onglet va apparaître dans la barre des menus : My page.

Lorsque l'on clique sur My page, on a alors accès à la page personnelle d'un utilisateur.



On remarque que pour cette page, on a réutilisé le composant Dashboard, mais en fournissant en paramètres seulement les jeux que l'utilisateur a acheté. Ce composant a donc exactement le même comportement que précédemment, et s'affichera également sous forme de liste pour une taille de l'écran plus petite. Il est aussi toujours possible d'accéder à la fiche technique d'un jeu en cliquant dessus, la seule différence sera l'absence du bouton d'achat :



En effet, nous sommes là encore redirigés vers le composant GameDetail, et celui-ci va d'abord appeler le service de session pour savoir si le jeu présenté a déjà été acheté ou non. Evidemment, le comportement du composant GameDetail vis-à-vis de la taille de l'écran est le même que précédemment.

Forces et faiblesses d'Angular 2 pour l'adaptation visée

Globalement, la technologie de Web Components proposée par Angular 2 se marie très bien avec les adaptations de type dispositif. En effet, la définition de composants dont le style d'affichage est complètement indépendant du style de la page. Cela permet donc d'avoir le même affichage et la même adaptation au dispositif d'un composant quel que soit le contexte d'utilisation.

De plus, cette isolation du style du composant facilite grandement l'édition de ses propriétés CSS puisqu'elles ne peuvent pas entrer en conflit avec d'autres propriétés. Cette technologie permet aussi de fournir des paramètres aux composants dans le but de changer leur type d'affichage, permettant ainsi de déléguer la tâche d'adaptation au composant parent (le conteneur) si besoin.

Enfin, Angular 2 dispose d'un module prenant en charge l'adaptation au dispositif (ng2-responsive) et permettant de changer (entre autres) la structure html d'un composant selon la taille de l'écran. Cela s'avère très pratique lorsque l'on souhaite rajouter ou enlever du contenu, l'avantage par rapport aux Media Queries de CSS3 étant de ne pas polluer le DOM. En effet, en CSS on se contenterait de cacher les balises qu'on ne souhaite pas voir apparaître alors qu'avec Angular 2, ces balises n'apparaîtront pas dans le DOM. Il est également possible de combiner ces deux solutions pour rendre l'adaptation plus simple à mettre en œuvre. Concernant l'adaptation au dispositif non présentée dans ce projet, ce module Angular 2 permet d'aller plus loin en détectant le type d'appareil (Windows phone, iPhone, iPad, etc..), le navigateur utilisé ou encore l'orientation portrait/paysage. Bref en ayant autant de possibilités à disposition et sachant que l'on peut aussi utiliser des librairies externes telles

Bootstrap, on se retrouve avec un éventail très complet de moyens permettant d'adapter une interface selon le dispositif.

Angular 2 ne souffre pas de beaucoup de défauts, mais son module responsive oblige tout de même le développeur à dupliquer du code html s'il veut réorganiser ses balises selon la taille de l'écran. En effet, le contenu des balises à réorganiser devra être réécrit à chaque réorganisation sans qu'il n'y ait de possibilité de factorisation. On peut également déplorer l'absence d'un système automatisé de tests pour interface graphique, même si la concurrence ne fait pas mieux. Enfin, Angular 2 est une technologie très récente et évoluant rapidement, ce qui réduit la présence de documentation et de tutoriels à jour, rendant (pour l'instant) le développement quelque peu difficile.

Concernant la réutilisation de composants, Angular 2 est également pratique puisqu'il est possible de les réutiliser sans les changer (par exemple un composant de paiement) mais aussi de leur fournir des paramètres pour s'adapter comme dit plus haut. Ils peuvent également retourner des valeurs à leur composant parent, offrant ainsi une interaction pratique. La présence de services permet également d'ajouter une couche d'abstraction au fonctionnement des composants, améliorant ainsi leur réutilisabilité. L'héritage est également pris en compte via Typescript, offrant là encore une couche d'abstraction utile dans ce cadre d'adaptation.

Tutoriel application Bootstrap

Prérequis et installation

Bootstrap ne nécessite pas d'installation particulière, il suffit d'ajouter les fichiers CSS et les fichiers JavaScript au projet afin de pouvoir l'utiliser. Il faut aussi ajouter jQuery au projet étant donnant que Bootstrap utilise certaines de ses fonctionnalités.

Il suffit d'ouvrir la page index.html pour ouvrir le site.

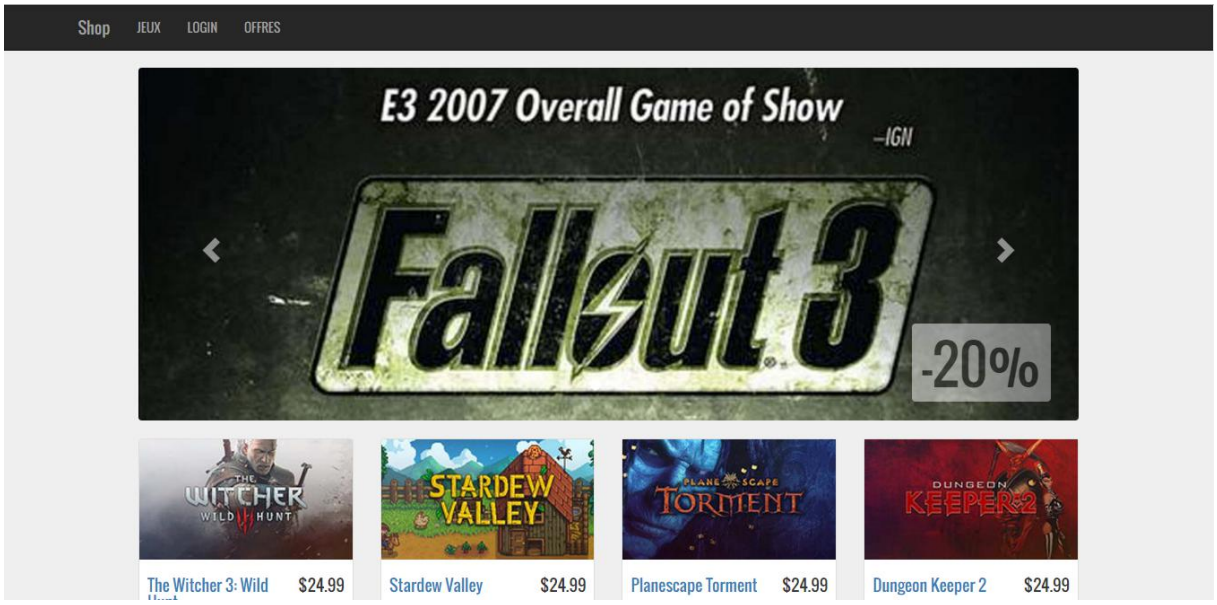
Informations complémentaires

Ce projet est basé sur divers exemples d'utilisation de Bootstrap tirés du site W3School. Les fonctionnalités du site ont été simplifiées au maximum afin de se focaliser sur la partie adaptation qui nous intéresse pour ce projet.

Le projet a été effectué sous Visual Studio et l'adaptation a été testée grâce à Google Chrome qui permet d'afficher le site sous divers formats mobiles.

Test des capacités d'adaptation à l'écran

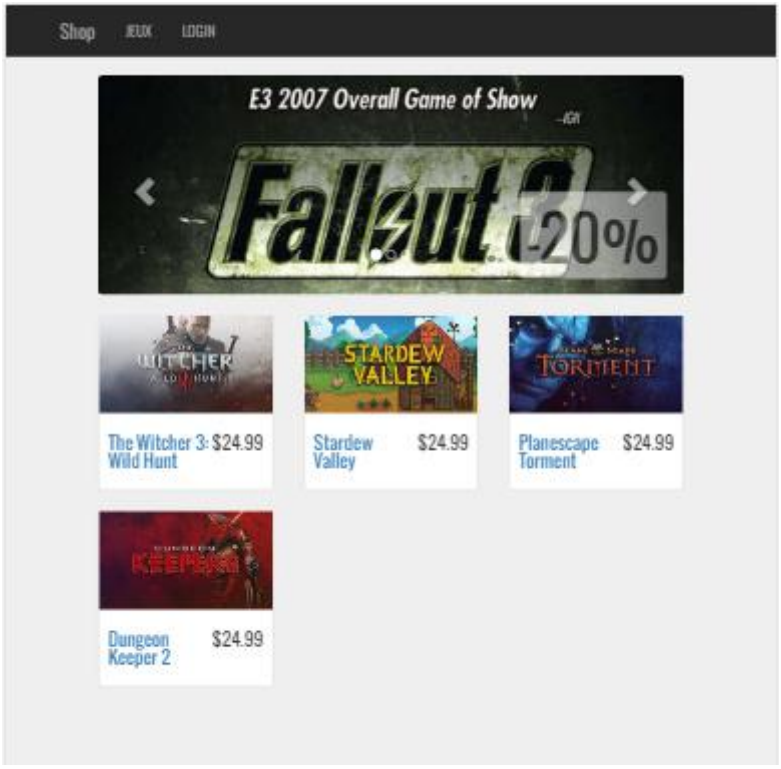
Un fois le site ouvert, vous devriez obtenir la page d'accueil suivante:



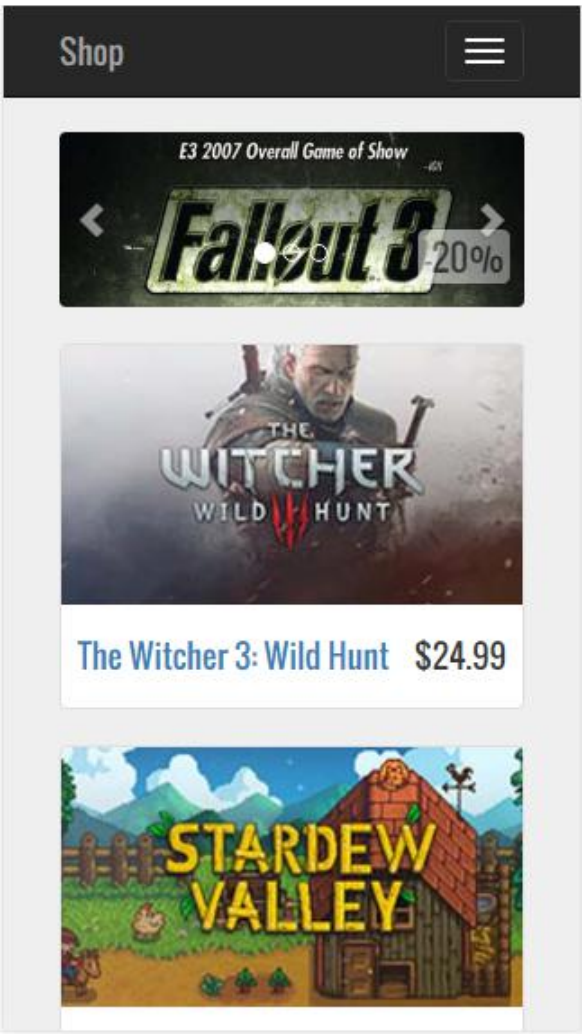
On a un *carousel* qui permet de faire défiler les jeux actuellement en promotion, celui-ci défile de manière automatique mais on peut également utiliser les flèches afin de les faire défiler

manuellement. Le *carousel* existe de base dans Bootstrap, il faut l'intégrer dans un *carousel-holder* puis ajouter les items que l'on voit qu'il contienne.

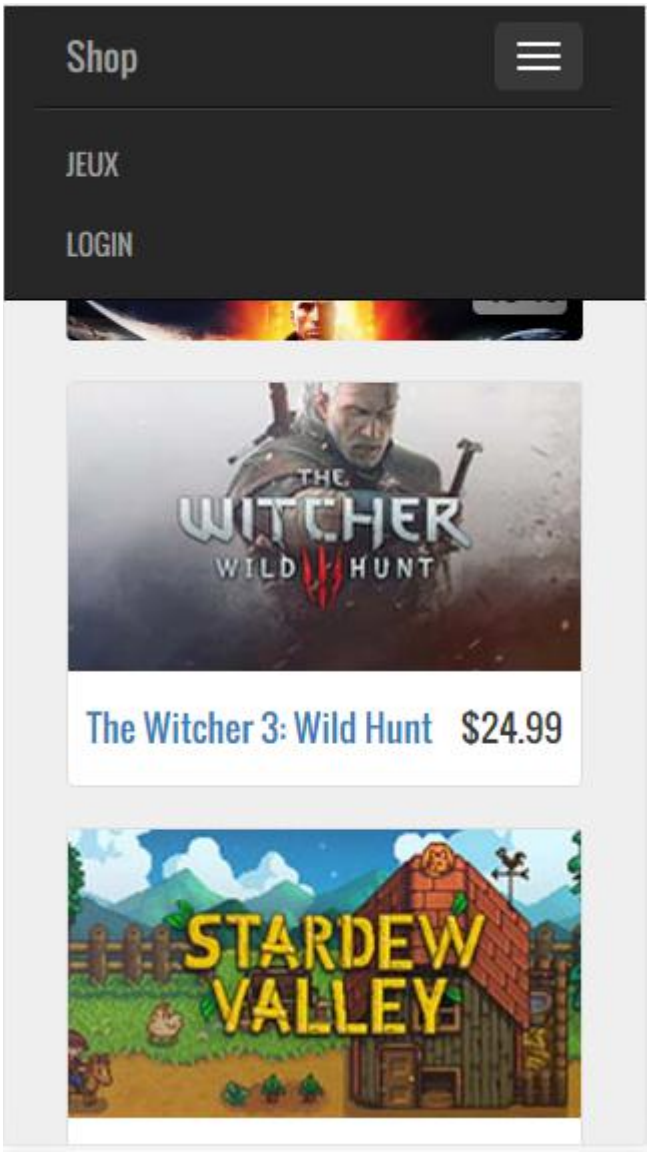
Au format desktop, on peut voir que les jeux sont affichés en lignes de 4. Le nombre de jeux par ligne va s'adapter en fonction de la taille du dispositif sur lequel est visualisé le site. On utilise pour cela les *container* de Bootstrap, ainsi que les *rows* et les *cols* qui nous permettent de placer les différents composants de la page sur la grille de 12 colonnes définie par Bootstrap. Au format large, les images prennent 3 colonnes chacune, aux formats medium et small (ex: tablettes), elles en prennent 4, et 12 pour le format extra-small (smartphones).



On peut voir sur cette image l'affichage du site sur iPad (vertical), la taille des images du *carousel* s'adaptent, et les jeux sont affichés par trois sur chaque ligne. Le menu lui reste inchangé par rapport à la version desktop.

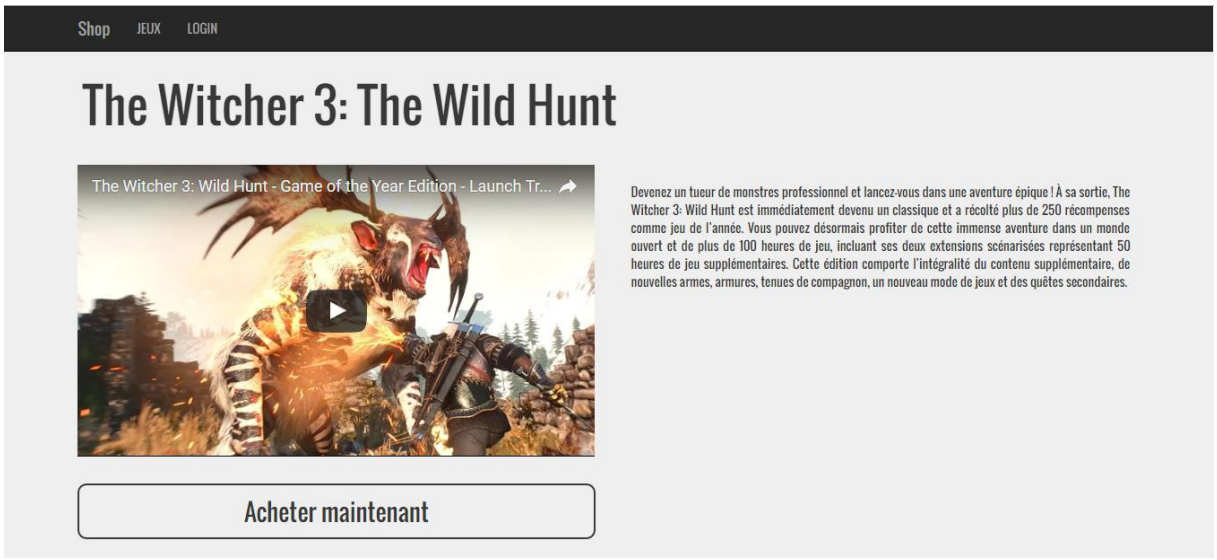


Sur cette image, on peut voir l'affichage du site sur iPhone 5, encore une fois le *carousel* s'est adapté à la taille de l'écran. Les jeux sont désormais affichés un à un sur chaque ligne. Cette fois, le menu est modifié afin d'être plus facilement utilisable sur smartphone sans être trop encombrant. On a un bouton qui va permettre d'afficher les items du menu lorsque l'on appuie dessus:



Le fait de modifier le nombre de jeux par ligne permet d'adapter leurs tailles à l'écran et ainsi faciliter la visualisation des informations affichées. Si on avait le même affichage sur smartphone que sur desktop, il y a de fortes chances que les informations deviennent illisibles.

Lorsque l'on clique sur un jeu (ex: The Witcher), le site nous redirige sur une page de présentation du jeu:



Cette page comporte une vidéo, une description du jeu et un bouton permettant d'acheter le jeu. La div contenant l'iframe de la vidéo est de la classe embed-responsive et l'iframe est de la classe embed-responsive-item. Ce sont ces options qui vont permettre d'adapter la taille de la vidéo à l'écran utilisé pour visionner le site. La version iPad est très similaire à celle-ci.

Sur iPhone, le texte ne pouvant plus être affiché à côté de la vidéo de manière lisible, on le fait passer sous le bouton “Acheter maintenant”. On utilise des media queries pour adapter la taille du bouton, en modifiant la taille de la police.

Avantages et inconvénients de Bootstrap

Un des principaux avantages de Bootstrap est sa simplicité d’utilisation. On a seulement besoin de HTML et CSS (en plus des fichiers JavaScript nécessaires pour le faire fonctionner) pour créer nos pages responsive. On se sert principalement des *class* définies dans les fichiers CSS de Bootstrap, notamment des *class* : container, row, col-... . De plus, Bootstrap étant largement utilisé, sa communauté est assez importante, celui-ci dispose donc d’une documentation fournie qui permet de le prendre en main assez vite. De plus, le design responsive est très facile à tester, étant donné que celui-ci s’adapte en temps réel lorsqu’on modifie la taille de la fenêtre d’affichage, contrairement à Foundation qui nécessite d’actualiser la page pour que celle-ci s’affiche correctement.

Dans le cadre de cette adaptation, il est assez pertinent d’utiliser Bootstrap. En effet, sa disposition en grille est tout à fait adaptée à un site de e-commerce (ici, vente de jeux vidéo) car elle permet d’afficher très facilement les items vendus, et d’adapter leur affichage en fonction du dispositif utilisé pour visionner le site. Une des seules choses que l’on pourrait reprocher à Bootstrap est le fait que sa grille diminue la liberté en terme de design.

Tutoriel Foundation 6

Prérequis et installation

Pour pouvoir développer avec Foundation, il est nécessaire d'avoir Sass installé sur son ordinateur. Sass est un préprocesseur CSS, c'est à dire que Sass définit un nouveau langage pour les feuilles de style, et que ce langage est compilé côté serveur en feuille de style CSS. Sass utilise sa propre syntaxe et les fichiers Sass ont bien souvent l'extension .scss (en tous cas pour Foundation, cependant une extension .sass existe aussi mais la syntaxe est légèrement différente). Afin de fonctionner, Sass a besoin de Ruby, qui est un langage de programmation.

Pour installer Ruby sur Windows, le meilleur moyen est d'utiliser http://rubyinstaller.org/, qui est un installateur simple et rapide. Une fois Ruby installé, il faut lancer une console en mode administrateur et lancer la commande *gem install sass.* Une fois ces étapes accomplies, Sass est installé sur la machine.

Les étapes précédentes sont nécessaires uniquement si l'on souhaite développer avec Foundation. Puisque les fichiers scss sont compilés en fichiers css, un projet Foundation fonctionnel ne nécessite ni Sass ni Ruby pour fonctionner. Si des comportements avancés sont implémentés sur le site web (type ajax et comportements client/serveur), il faudra peut-être déployer la page sur un serveur (le plus pratique étant de lancer un serveur local).

Dans notre cas, la page est très basique, il suffit donc d'ouvrir le fichier **index.html** situé à la racine du projet.

Informations complémentaires

Le site web créé dans le cadre de ce projet s'inspire des standards du e-commerce actuels (liste d'items, carousel en haut de page, top-bar pour naviguer, etc). Un effort concernant l'aspect graphique du site a été fait afin de rendre la démonstration plus réaliste et de se rapprocher d'une adaptation réelle. Un point supplémentaire à souligner est que afin de se mettre "dans les conditions" du cours, le site web a d'abord été réalisé dans sa version "desktop". Le développement de sites web de nos jours a tendance à être mobile-first, c'est à dire qu'on développe d'abord la version mobile puis on ajoute des éléments pour aboutir à la version bureau du site. Comme le défi est ici d'adapter une interface existante, la version mobile a donc été développée en adaptant la version desktop, qui a été réalisée en premier lieu.

Le site web a été développé avec l'IDE WebStorm, et a été testé sur Google Chrome uniquement. Afin de tester l'adaptation, deux méthode sont été utilisées :

1) en premier lieu, redimensionnement simple de la fenêtre du navigateur
2) une fois la version finale atteinte, le site web a été testé dans le simulateur de devices de Google Chrome (ctrl+shit+i puis ctrl+shift+m pour l'activer). Le simulateur de devices de

Google Chrome permet d'émuler parfaitement le site web en fonction des différents types de smartphones/tablettes (plusieurs sont proposés dans l'émulateur).

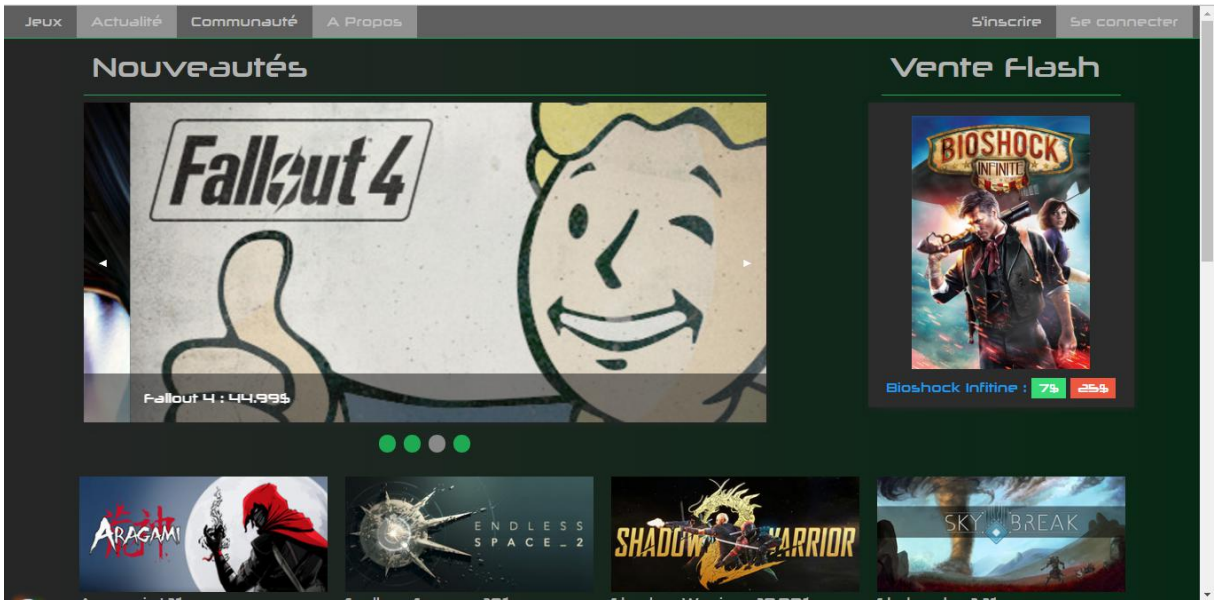
Dernière information : les deux fichiers créés à la main dans le cadre de ce projet sont **index.html, bioshock.html et app.scss** (pour les évaluateurs).

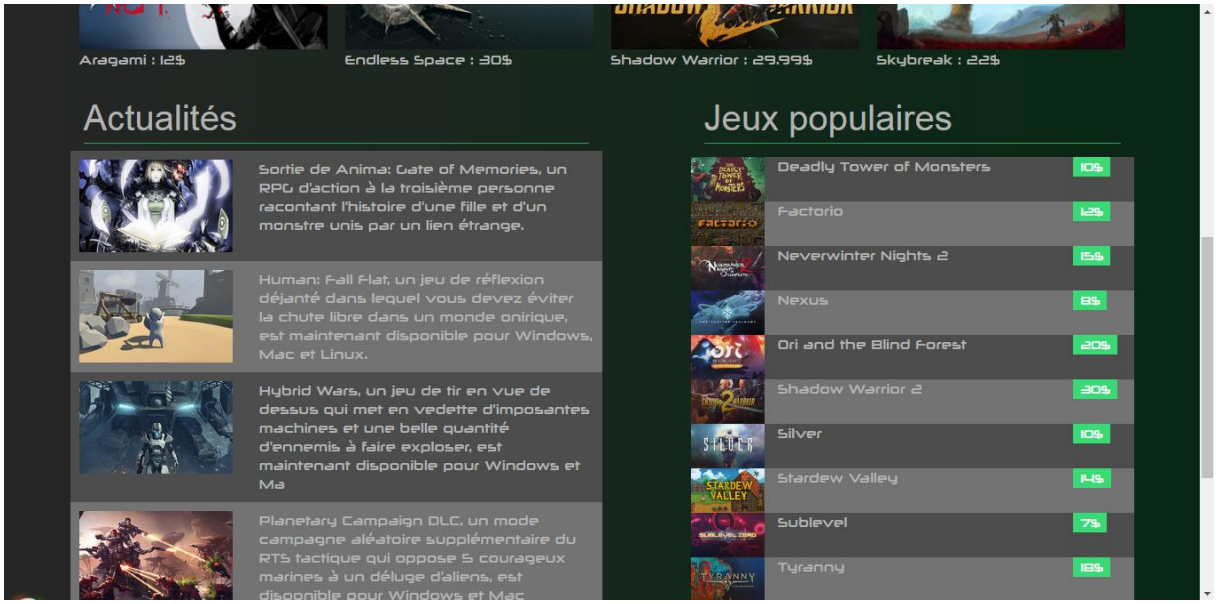
Tour d'horizon de la page d'accueil en mode desktop (large)

La page au format desktop (index.html avec une fenêtre de plus de 1024 pixels de large) a été construite en utilisant les attributs de colonnes "large-x". Quelques explications : Foundation, comme beaucoup de frameworks responsives, fonctionne avec un système de grille. Cette grille fait la largeur de la fenêtre et est constituée de 12 colonnes. Ces 12 colonnes (columns dans le code) sont placées dans des lignes (row dans le code).

Foundation fonctionne avec trois dimensions de fenêtre : small, medium et large. Les fenêtres smalls sont les fenêtres de moins de 640px de large, les mediums de plus de 640px, et les large de plus de 1024px. Lorsque que l'on spécifie la taille d'une zone en colonne, on peut spécifier cette taille pour les trois formats de fenêtre. Ainsi, un attribut avec la class *class="large-6 medium-9 small-12 columns"* prendra 6 colonnes sur un grand écran (soit la moitié de l'écran), 9 sur un écran moyen et tout l'écran sur une petite fenêtre. La plupart des frameworks se basent sur ce principe assez simple, et c'est à partir de ce principe que ce site a été construit.

Bien évidemment, nous allons présenter ici le layout pour la taille d'écran **large.**





Analysons cette page. Tout en haut se trouve une barre de navigation “top-menu”. Ce menu situé en haut de la page dispose d’un certain nombre d’options, qui ne tiendront certainement pas en largeur sur un petit écran. Nous avons ensuite deux catégories :

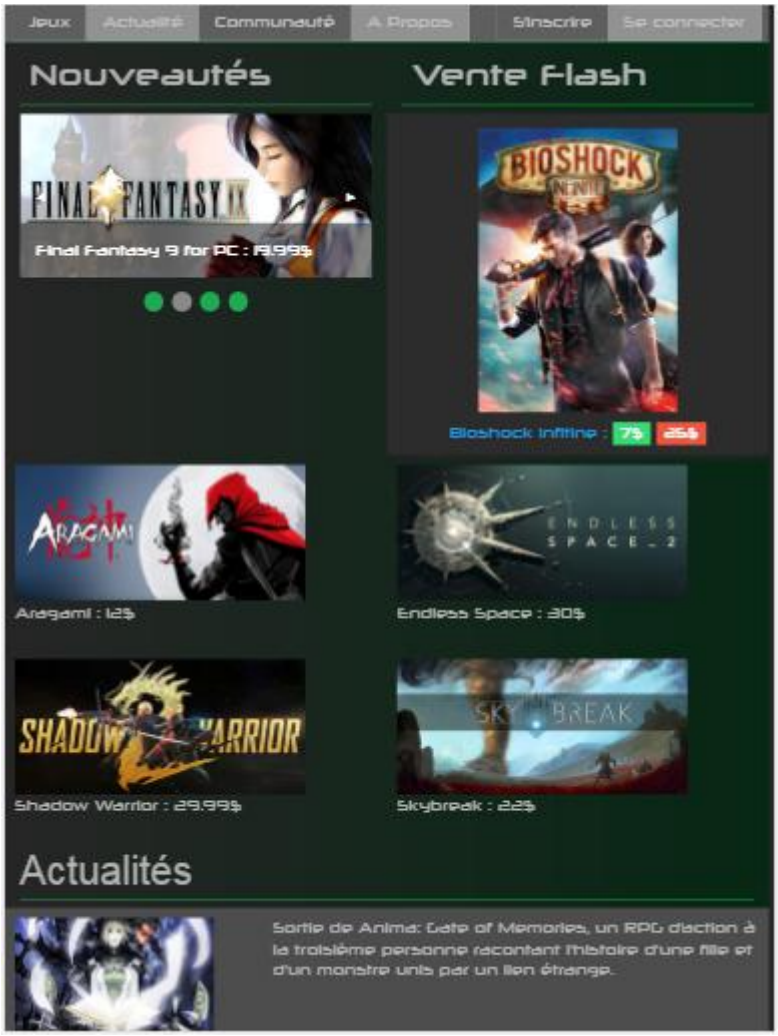
- Nouveautés : cette catégorie est composée d’un carousel (composant “Orbit” de Foundation). Ce composant est plutôt massif, il prend donc 8 colonnes sur le layout.
- Vente Flash : cette catégorie met en avant un article du catalogue en promotion. Elle est donc placée en haut de la page, près du centre de l’attention directement à côté du carousel. Comme cette catégorie présente un unique produit, elle prend seulement 3 colonnes de large. Une colonne est laissée vide pour créer un peu d’espace.
- Nouveautés secondaires : ces quatres articles font partie des nouveaux jeux, mais ils sont de moindre importance pour le vendeur que les éléments du carousel ou de la vente flash, ils sont donc placés juste en dessous mais sont quand même mis en valeur : chaque article prend 3 colonnes de large.
- Actualités : les actualités du site et du domaine des articles vendus par le site. Cette catégorie prend la moitié du reste de la page, et les news sont empilées verticalement, comme dans un journal. Les fonds changent de couleur à chaque article successif (un sur deux) afin de créer une meilleur lisibilité/séparation visuelle
- Jeux Populaires : il s’agit des articles populaires du catalogue. Cette catégorie prend elle aussi la moitié du reste de la page (6 colonnes) et les articles sont empilés verticalement.

Quelques “tricks” sur les colonnes ont été faits dans les articles d’actualité, dans les jeux populaires, etc. afin d’obtenir le layout désiré. Par exemple, pour la vente flash, une sous grille a été créée dans le container de la catégorie “Vente Flash”, avec une image qui prend 12 colonnes en mode large et un texte qui en prend 12 aussi.

Il n’y a rien de vraiment compliqué dans cette page. Cependant, il est fortement recommandé d’utiliser le template de Foundation pour le carousel (composant Orbit), car c’est un composant capricieux (exemple : il faut recharger la page quand on redimensionne la fenêtre car le composant Orbit ne se redimensionne plus jamais une fois chargé) et dont le code ne s’invente pas.

Tour d’horizon de la page d’accueil en mode tablet (medium)

Nous allons présenter ici l’adaptation de l’interface desktop à des écrans medium, c’est à dire des fenêtres de la taille d’une tablette. Nous rappelons que ces écrans font plus de 640px de large mais moins de 1024px.

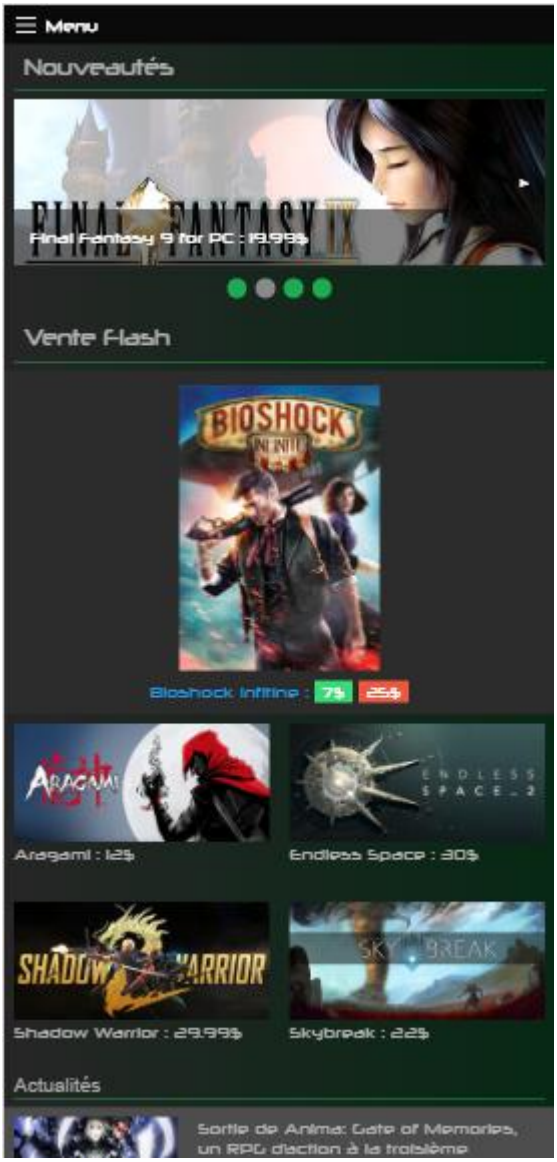


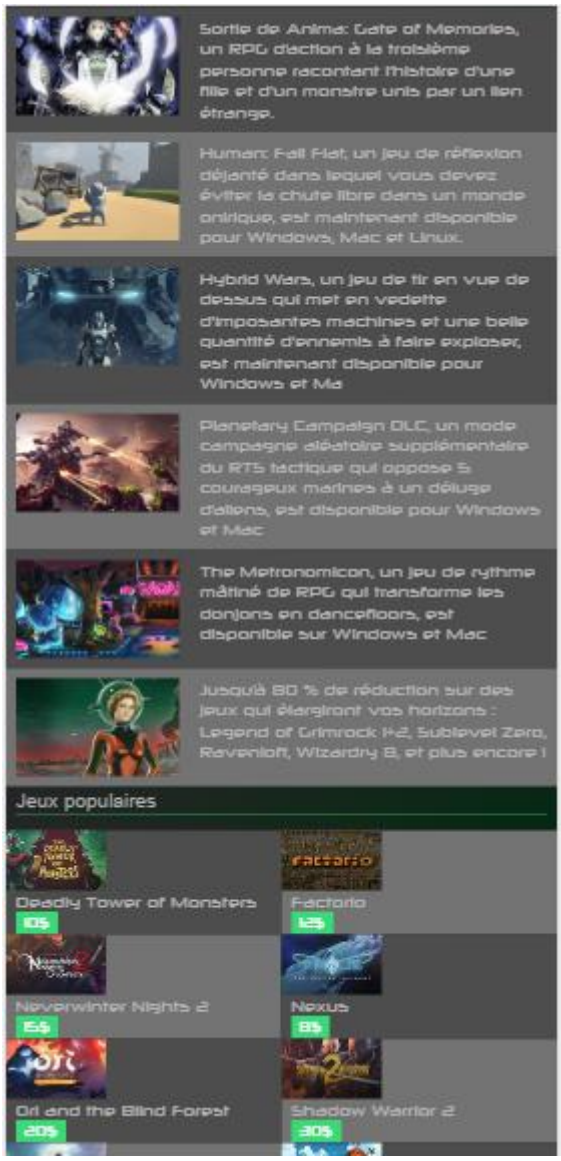
Comparons cette version avec la version desktop :

- Il y a toujours assez de place pour que le menu du haut ne collapse pas.
- Le carousel a été réduit et partage maintenant la moitié de l'écran avec la vente flash, cet arrangement est intéressant puisqu'il donne l'impression de feuilleter un magazine, les colonnes prennent chacun une moitié de l'écran
- Ce système de "deux colonnes" se prolonge sur les articles en nouveauté secondaire : ces articles prennent maintenant 6 colonnes au lieu de 3 et les couples d'articles se stackent verticalement
- On a donc pour cette partie de la page rassemblé les éléments sur deux colonnes de tailles égales
- Pour les actualités, nous avons fait l'inverse. En effet ici l'information textuelle est beaucoup plus dense, il n'est donc pas envisageable de laisser autant de texte sur une moitié de page sur un petit écran : les articles prennent donc toute la largeur de la page et s'empilent verticalement
- Les articles populaires contiennent peu d'information, vu qu'il ne reste plus que cette catégorie elle prend donc tout l'écran, cependant, les articles sont organisés sur deux colonnes distinctes pour garder en cohérence : il n'y a pas assez de contenu dans les articles populaires pour leur faire remplir toute la largeur de la fenêtre.

Tour d’horizon de la page d’accueil en mode smartphone (small)

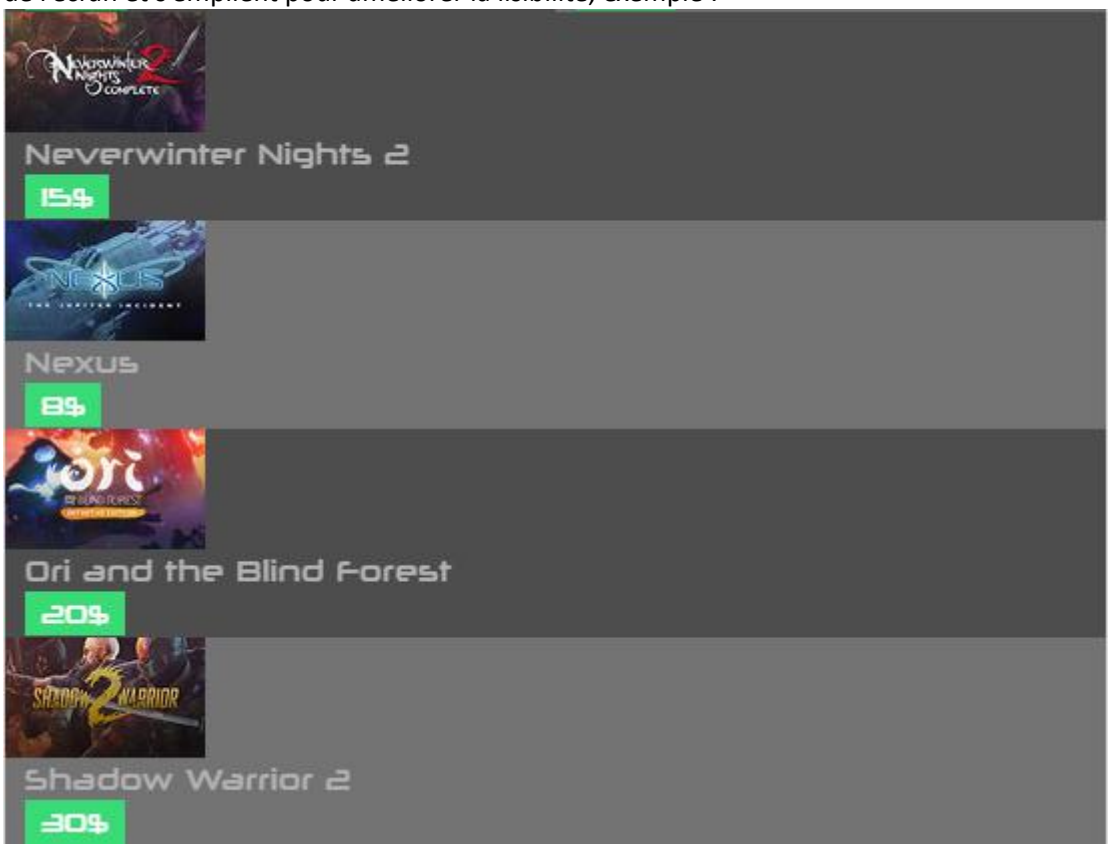
Nous allons présenter ici l’adaptation de l’interface desktop à des écrans small, c’est à dire des fenêtres de la taille d’un smartphone. Nous rappelons que ces écrans font moins de 640px de large.





Comparons cette version avec les deux autres versions :

- L'écran est devenu trop petit pour le menu, ainsi celui-ci s'effondre et se retrouve caché derrière un bouton type burger-menu. Lorsque l'on clique sur l'icône de menu, celui-ci s'affiche. C'est le seul composant fourni par Foundation pour réaliser cette opération. C'est un pattern très limité et dont l'usabilité est questionnable. Il permet cependant de cacher le surplus d'information (mais manque d'affordance).

- Il n’y a plus assez de place pour garder le carousel et la vente flash sur la même ligne : chaque composant prend donc maintenant 12 colonnes et est empilé verticalement.
- Il y a cependant toujours assez de place pour les nouveautés secondaires et tout reste lisible, aucun changement de ce côté ci. Idem pour les articles d’actualité.
- Les articles populaires ne changent pas de layout, cependant, on aurait pu le changer. L’information affichée est tellement faible qu’on peut se permettre de garder ce pattern. Cependant, si elle venait à grossir, il faudrait que les jeux populaires prennent toute la largeur de l’écran et s’empilent pour améliorer la lisibilité, exemple :
- 

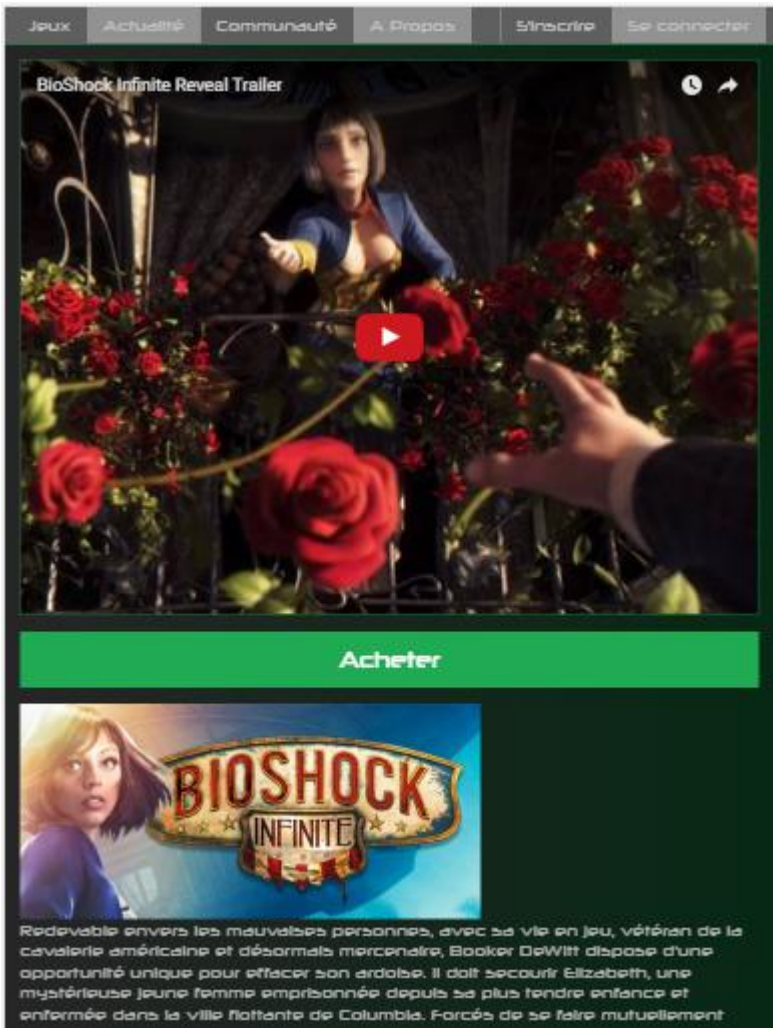
On remarque donc que pour la page d’accueil, le layout change considérablement en fonction du format de la page. On adapte surtout les éléments et leur taille dans la grille en fonction de la visibilité désirée et de la densité du contenu de ces éléments. Par exemple, le carousel prend toujours une bonne partie de la page et est toujours placé en haut quel que soit la taille de la fenêtre car ce sont avant tout les items que l’on veut vendre. Les articles doivent rester lisibles, donc quand l’écran réduit en taille on augmente la proportion qu’ils prennent à l’écran afin que la taille de la police et la lisibilité restent constantes. Les items de moindre importance comme les jeux populaires peuvent être stackés et verticalement et horizontalement car ce ne sont pas ceux que l’on cherche le plus à vendre ou ceux qui présente le plus de contenu, contrairement à la vente flash ou aux nouveaux secondaires.

Tour d’horizon de la page d’un jeu

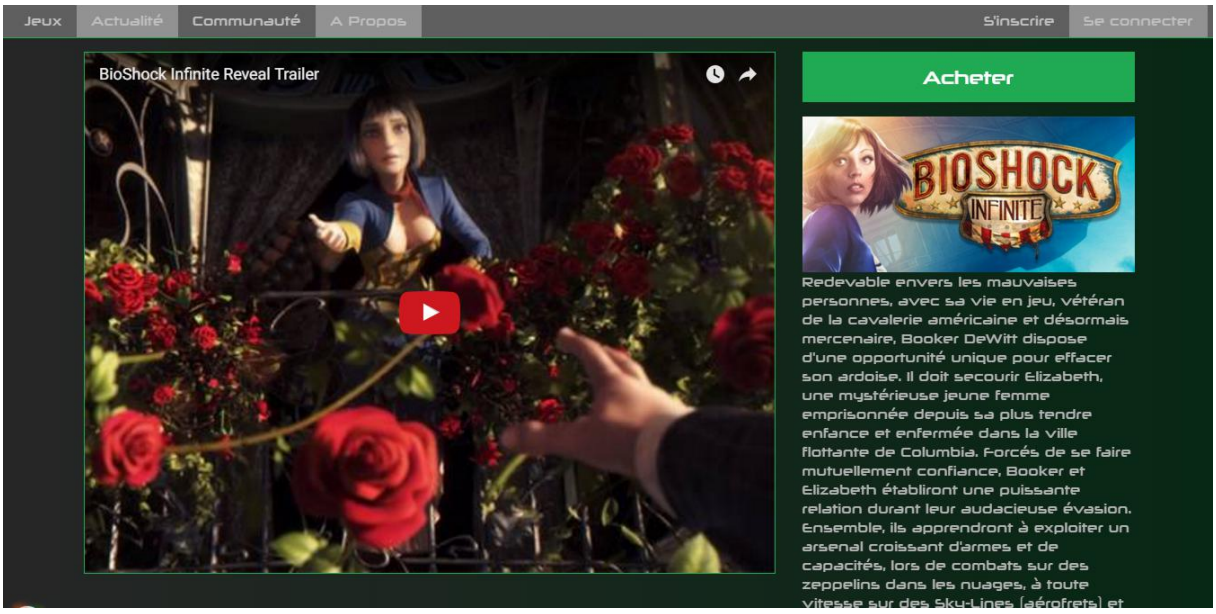
La page de détails d’un jeu est une page très classique, au layout assez simpliste. Elle sert à mettre en avant le comportement de Foundation quand on veut intégrer un lecteur vidéo et un peu plus de contenu textuel que sur la page d’accueil. Nous allons présenter d’un coup les trois versions :

- Version smartphone :

- Version tablet :



- Version desktop :



On remarquera que les versions tablet et smartphone ne diffèrent entre elle que par la présence ou la non présence du menu “étendu”. Sinon le layout est le même. La version desktop quant à elle adopte un layout tout à fait différent. Analysons les points clé :

- Sur cette page, le contenu le plus important est définitivement la vidéo. Elle est donc placée en haut à gauche et occupe une majeure partie de l’écran sur la version desktop. Sur les versions medium et small, la vidéo occupe même toute la largeur de l’écran.
- Viens ensuite le bouton d’achat et la jaquette “numérique” du jeu. Ces éléments sont placés par ordre d’importance, stackés verticalement pour insister sur leur importance sur les petites versions, placés à droite du contenu multimédia car la place le permet sur la version desktop.
- Le texte est placé à la fin car c’est à priori l’information que l’on va consulter en dernier, comme c’est une information dense elle prend toute la largeur de l’écran dans les petites versions.

Avantages et inconvénients de Foundation

Le principal concurrent de Foundation est Bootstrap. Le développeur de ce site web avait une bonne expérience de Bootstrap, d’où le choix de tester Foundation. L’analyse suivante se basera donc sur l’utilisation de Foundation dans le cadre de ce projet et sur des comparaisons avec Bootstrap.

Tout d’abord, il est beaucoup plus compliqué de faire fonctionner Foundation au démarrage que Bootstrap. Foundation nécessite d’installer Ruby, puis d’installer Sass pour fonctionner correctement. Ce n’est pas forcément une étape très évidente pour certains designers web n’ayant qu’une expérience limitée en programmation. De plus, Foundation est moins léger dans le sens où il impose une utilisation de Sass. Sass n’est pas un langage très compliqué pour qui connaît bien le CSS, et de nos jours c’est une bonne chose de connaître un langage comme LESS ou Sass. Cependant le fait qu’un framework tel que Foundation impose de lui même le choix du préprocesseur CSS est discutable et pourrait être une raison de ne pas utiliser Foundation. L’utilisation du npm start est par ailleurs assez fastidieuse, en effet seul le fichier app.scss est compilé par défaut, et il faut chercher dans le script gulp pour le savoir. Foundation a pris quelques heures d’installation pour tout comprendre et faire le bon setup, Bootstrap avait pris vingt minutes. Foundation utilise par ailleurs d’autres packages créés par Foundation, seul problème : les noms des fichiers n’ont rien à voir avec Foundation. Par exemple, Motion-ui est une petite librairie utilisée entre autres par le carousel, mais rien n’indique à quoi il sert ou s’il fait partie du framework. Par ailleurs, Motion-Ui n’est pas fourni dans tous les templates de projet.

Un autre problème de Foundation est sa documentation : celle-ci est parfois assez floue et se repose trop sur les exemples. Il est difficile de comprendre comment fonctionnent vraiment les composants : impossible d’explorer facilement le framework. En résumé, la documentation propose dans la plupart des cas des bouts de code à insérer et modifier sur sa page, mais nulle part n’est expliquée la logique derrière les éléments. C’est un point assez embêtant qui n’aide pas à utiliser efficacement Foundation.

De plus, la communauté autour de Foundation est bien faible. Il est difficile de trouver des questions/réponses et le taux de réponse est globalement faible (et le plupart du temps les réponses sont moins maîtrisées que sur Bootstrap par exemple). Des sites comme StackOverflow illustrent cette situation : 4000 questions pour Foundation contre 71000 questions pour Bootstrap.

Pour l'adaptation, l'impression est mitigée mais généralement positive. Foundation est globalement assez simple dans sa prise main, il n'y a pas de dizaines de noms de classe à rallonge dans le html, les composants sont simples et les exemples rapides à mettre en oeuvre. Le système de grille est tout à fait standard et facile à prendre en main, on pourra juste se demander si 3 points de ruptures au lieu de 4 suffisent (on utilise généralement les 4 tailles smartphone, tablet portrait, tablet paysage/small desktop, desktop/large desktop). Dans notre cas assez simpliste nous n'avions pas besoin de plus, mais des projets d'envergure préféreront peut-être un meilleur réglage du responsive. Deux petits bémols constatés lors du développement :

- la page doit être rechargée lorsque l'on redimensionne la fenêtre sinon certains éléments fonctionnent mal (comme le carousel), c'est plutôt embêtant lorsque l'on essaie de jouer en temps réel avec la taille de la fenêtre
- le framework semble avoir du mal à gérer l'overflow : les combinaisons texte/image provoquent parfois des retours à la ligne inattendus alors qu'il a bien été spécifié un nombre de colonnes qui tient sur la largeur courante de la page et qui ne nécessite pas un retour à la ligne : par exemple pour les jeux populaires, si le titre du jeu est trop long alors l'item glitch ou est renvoyé à la ligne. Nous n'avons pas réussi à expliquer ce comportement.

En conclusion, Foundation est un outil pouvant s'avérer pratique car la simplicité de ses composants permet de travailler très rapidement. Il implique cependant de maîtriser un certain nombre de choses, comme Sass ou la procédure d'installation avant d'être opérationnel. On lui préférera cependant Bootstrap pour sa communauté, sa simplicité d'installation, sa documentation et sa stabilité globale. Pour finir sur une touche positive, nous avons par contre apprécié le look and feel de Foundation et l'aspect de ses composants graphiques, par rapport à ceux de ses concurrents. Même si le système de grille est le même qu'ailleurs, étrangement le site ressemble moins à un "bootstrap-clone" que beaucoup d'autres.

Tutoriel Xamarin

Prérequis et installation

Selon le système d'exploitation, il faut installer l'environnement de développement Xamarin:

https://www.xamarin.com/download

Cloner le repository de code qui est hébergé sur GitLab en utilisant git clone:

git clone https://gitlab.com/shafdanny/gamestore.git

Ensuite, ouvrez le fichier de projet avec Xamarin Studio ou Visual Studio.

Pour tester l'application sur le PC, nous avons besoins des émulateurs. Pour tester l'application Android, il faut installer Android Virtual Device ou Xamarin Android Player (https://developer.xamarin.com/releases/android/android-player/).

Sous Mac, un Simulator pour tester l'application iOS est installé avec Xcode.

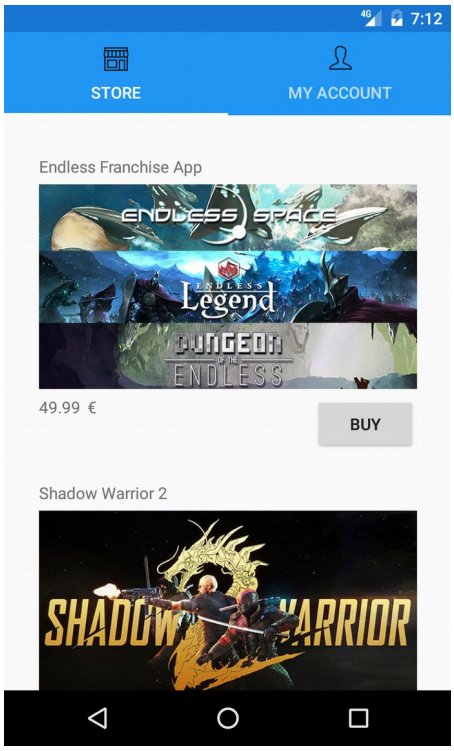
Informations complémentaires

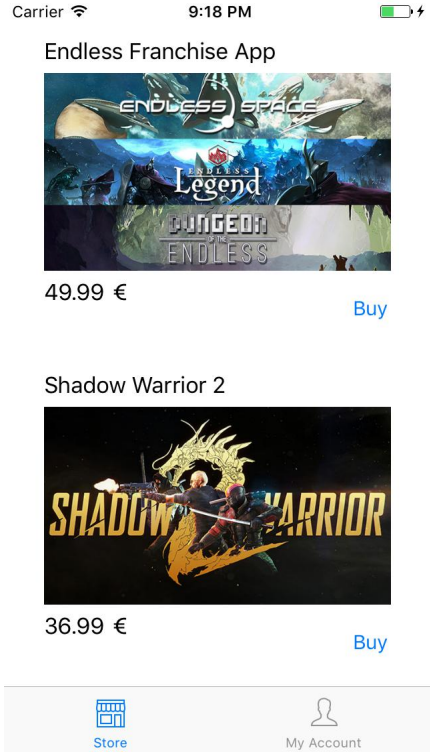
Le but de ce développement est d'essayer utiliser au maximum le Xamarin.Forms afin d'avoir un maximum de partage de code, sans réécrire l'IHM pour chaque platform. Puisque le Xamarin.Forms est assez nouveau, donc il n'a pas encore beaucoup de fonctionnalités, notamment les interfaces plus complexes ne sont pas encore disponible. Et contrairement aux développements précédents (Angular 2, Bootstrap, Foundation), nous sommes que sur le mobile. Donc, il n'y aura pas d'adaptation pour les grandes écrans.

Comparaison UI Android et iOS

Pour l'adaptation avec Xamarin, nous avons choisi de focaliser sur l'adaptation selon le platform mobile, dans ce cas Android ou iOS. Ces deux platform mobile a chacun un style ou paradigm de design pour leur UI/UX. Nous avons sur Android le Material Design qui définit le bonne pratique pour développer une interface sous Android (https://material.google.com). Pour iOS, il n'y a pas de nom spécifique pour le style de design, mais il a bien une implémentation spécifique pour les éléments des interfaces, et qu'un utilisateur iOS (ou même pas) peut reconnaître si une interface est bien de style iOS ou pas (https://developer.apple.com/ios/human-interface-guidelines). L'avantage principale de développement cross-platform sur Xamarin est nous pouvons créer des interfaces natives pour chaque platform, soit indépendamment (donc il faut définir chaque interface pour chaque platform) ou bien utiliser le Xamarin.Forms qui va générer les interfaces correspondant à chaque platform à partir d'un fichier XAML. Comme indiqué précédemment, nous avons choisi d'utiliser le Xamarin.Forms.

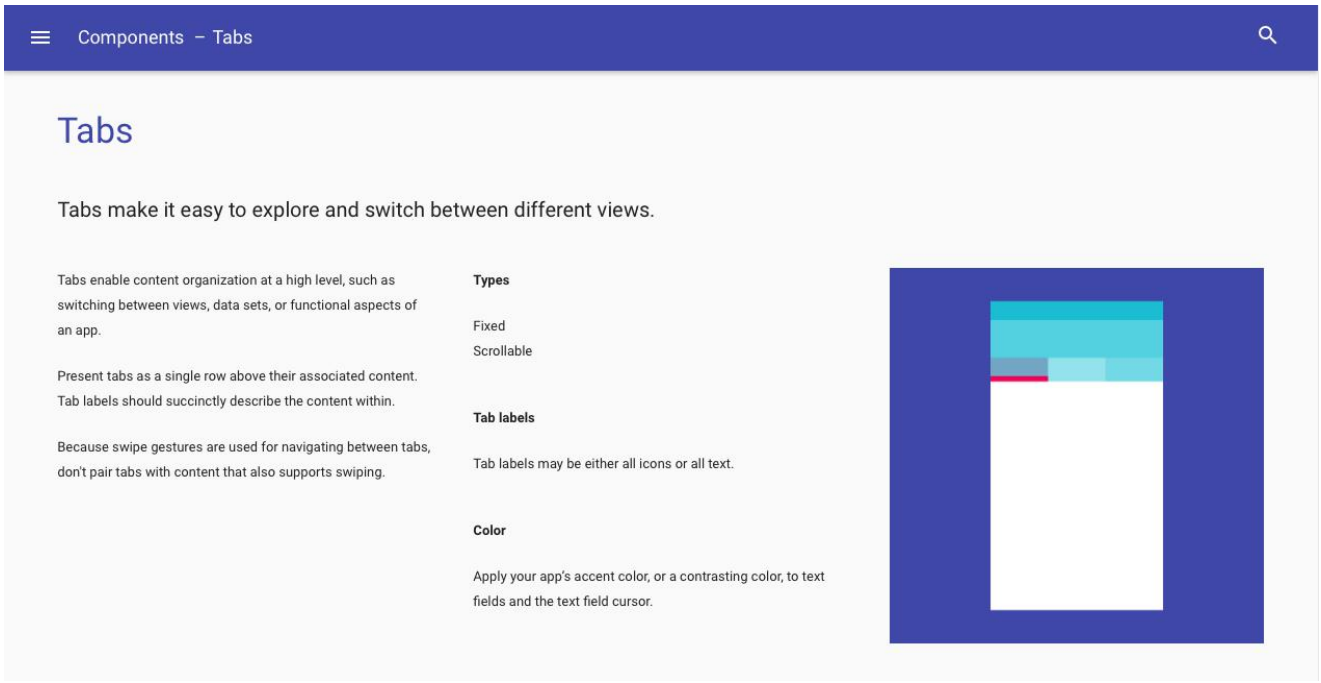
Voici la page de notre magasin avec une liste de jeux que nous pouvons acheter:



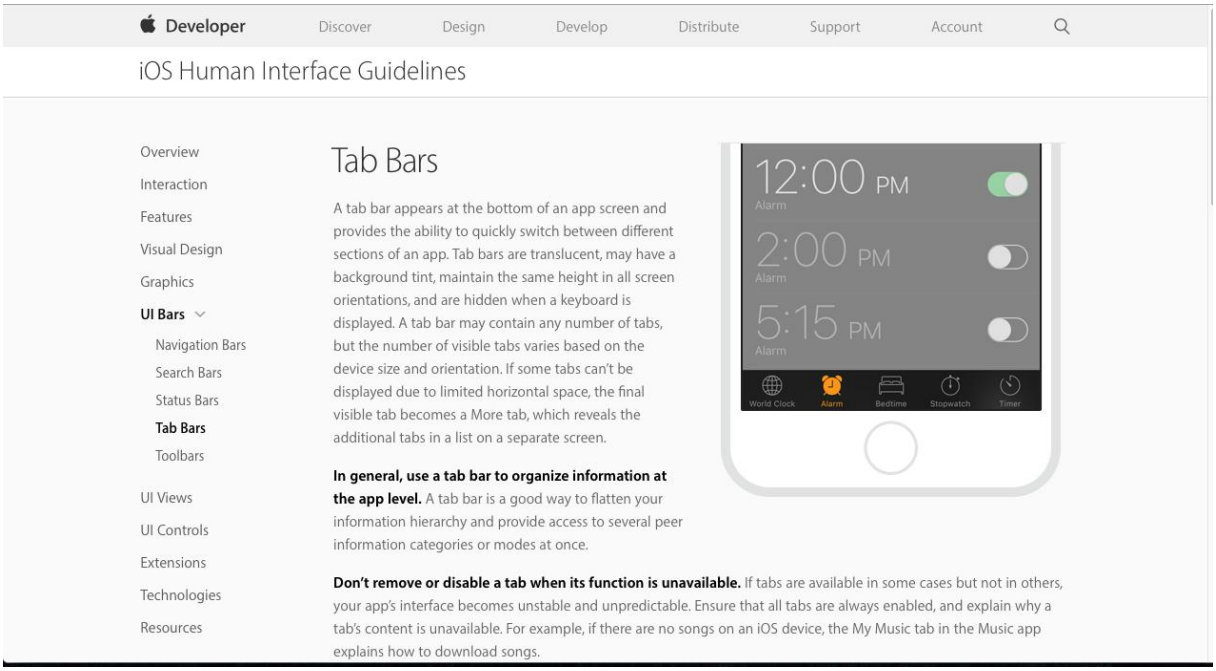


Pour réaliser cela, nous utilisons des composants de l'interface disponible avec Xamarin.Forms. Pour les deux pages avec des onglets, nous utilisons un TabbedPage. Ensuite, pour afficher une liste de jeux, nous utilisons un layout de type ScrollView. Cela nous permet de scroller dans une liste d'items. Chaque article de jeu à sa propre vue, qui est définie avec un StackLayout. Donc, nous pouvons placer plusieurs éléments dans le layout. Dans chaque vue d'article, nous mettons un text de titre de jeu, une image, un text avec le prix, et un bouton pour acheter le jeu.

On remarque les différences entre les deux interfaces, notamment dans le placement des onglets de page. Ces deux manières différentes correspond bien aux styles de chaque platform. Pour vérifier cela, il suffit d'aller vers le site Material Design de Google, et iOS Human Interface Guidelines. Voici les deux sites avec les consignes concernant les pages avec les onglets:



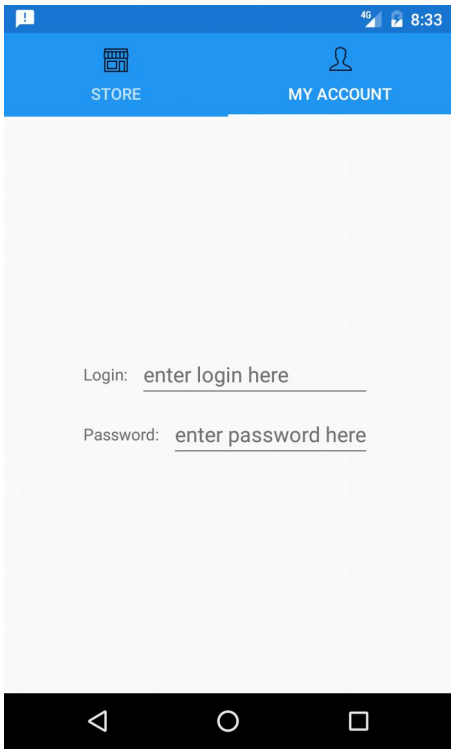
Android : https://material.google.com/components/tabs.html#

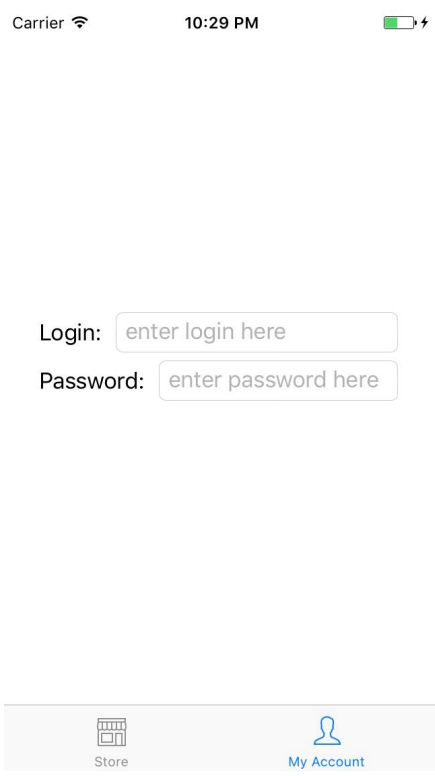


iOS: https://developer.apple.com/ios/human-interface-guidelines/ui-bars/tab-bars/

Cela est un exemple de même type de layout, visé différemment par les deux platforms mobiles. Grâce au Xamarin.Forms, nous n'avons pas besoin de nous occuper avec ces different methods d'implémentation, car toute est fait automatiquement.

Il y a beaucoup d'exemples où nous pouvons voir les différentes approches pour les mêmes types d'éléments dans une interface. La page de login suivant montre que sous Android, le champ de saisie de text est représenté juste avec un sous ligne, alors que sous iOS, c'est une boite de champ de text.

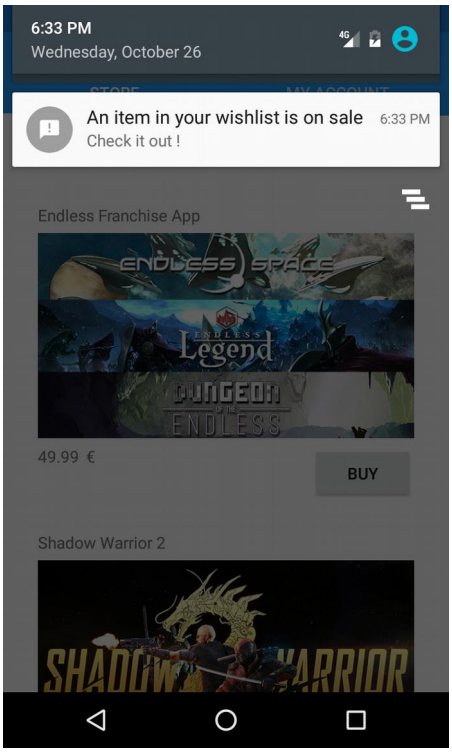


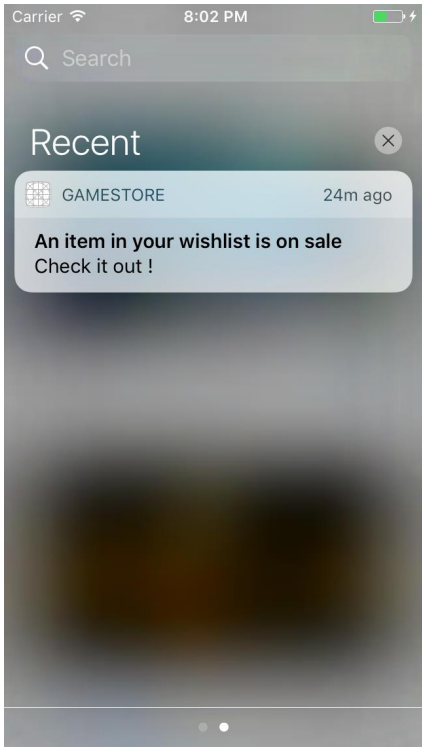


Le positionnement des éléments est fait avec l'attribut HorizontalOptions et VerticalOptions, qui nous permet de définir la position des enfants dans un parent. Dans ce cas, la vue de My Account est un ContentPage qui contient un StackLayout. Ce StackLayout a comme attribut HorizontalOptions et VerticalOptions égal à Center. Donc, tous les éléments dans StackLayout vont être centré horizontalement et verticalement. Ensuite, il nous reste de définir deux StackLayout avec Orientation égal à Horizontal. Cela nous permet d'empiler les éléments horizontalement. Il suffit après de mettre un Label et un Entry dans chaque StackLayout.

Un élément qui n'est pas disponible ou adaptée pour la technologie web ou responsive web design et qui reste unique pour l'application mobile est les notifications. Malheureusement, le package NuGet qui permet de faire les notifications pour Android et iOS ne marche pas très bien sous iOS (il y a un problème de formatage de message à afficher). Donc, nous avons besoin d'écrire le code spécifique pour le cible iOS afin d'utiliser la notification.

Notification:





Avantages et inconvénients de Xamarin

Le premier avantage d'utiliser Xamarin est nous utilisons seulement un seul langage de programmation pour développer une application mobile sous différents platforms. En plus, l'application généré est native, donc nous aurons la performance d'une application native, ce qui n'est pas forcément le cas si on utilise une technologie web pour faire le cross-platform.

Ensuite, avec Xamarin.Forms, nous n'avons pas besoin de définir plusieur fois l'interface pour chaque platform. Il suffit d'utiliser un seul fichier XAML pour une interface qui va être transformé automatiquement aux interfaces de nos cibles (dans ce projet, Android et iOS).

L'inconvénient à ce-moment est pour Xamarin.Forms, l'utilisation est assez limité. Au niveau des interfaces, il est encore conseillé de ne pas utiliser le Xamarin.Forms pour développer des applications pour un produit final. Le Xamarin.Forms est plutôt utilisé pour faire un prototypage rapide avec des interfaces simple. Par exemple, il est difficile d'intégrer un CarrouselView, car le package qui permet de faire cela n'est plus disponible dans la version stable de Xamarin.

En plus, l'utilisation de Xamarin.Forms limite les bibliothèques ou NuGet packages que l'on peut utiliser pour aider le développement. Cela est dû à la version de compilateur spécifique pour Xamarin.Forms qui n'est pas forcément compatible avec les autres packages. Donc, nous avons des choix limités pour les librairies que nous pouvons intégrer avec Xamarin.Forms. Cela présente des difficultés si nous avons une grande application qui a beaucoup de dépendances.

Conclusion

En résumé, toutes les technologies que nous avons utilisées offrent des options d'adaptations inégales. Angular 2 offre un éventail très complet de fonctionnalités pour gérer l'interface de l'application (notamment avec son module ng2-responsive), et sa technologie des Web Components permet une isolation du style de chaque composant offrant ainsi un développement pour l'adaptation à la taille de l'écran simple et efficace. De plus, il est toujours possible d'utiliser les Media Queries ainsi que des librairies externes telles que Bootstrap ce qui rend ce langage très pratique pour cette adaptation, malgré un petit défaut de duplication de code pour le module responsive. Enfin, Angular 2 est un langage qui évolue très vite, c'est pourquoi il n'est pas impossible qu'il soit rendu obsolète relativement vite par une version plus avancée de lui même.

Plus généralement, la technologie de Web Components proposée par Angular 2 permet de développer des composants indépendants et réutilisables au sein de l'application ou même sur d'autres projets, ce qui est très intéressant du point de vue architectural. En effet, ceux ci présentent de nombreuses options telles que le passage de paramètres, l'héritage ou encore la possibilité d'en faire des services, ces possibilités favorisant l'abstraction et l'insertion dans un contexte différent.

Pour la technologie cross-platform comme Xamarin, il y encore beaucoup des inconvénients par rapport à la développement native, mais cela est déjà un bon début vers un seul environnement de développement pour toutes les platforms mobiles. Si on vise à développer une application mobile avec des fonctionnalités native et utiliser les interfaces native, mais avec la possibilité d'utiliser une seule langage et environnement de développement pour plusieurs cibles, le cross-platform de Xamarin est le meilleur choix.

Les frameworks tel que Bootstrap et Foundation proposent quant à eux des solutions plus statiques mais plus faciles à mettre en oeuvre, où la majorité du travail sera de réaliser un design pertinent et son adaptation plus que de concevoir des comportements complexes. Ce sont en effet des frameworks essentiellement graphiques, proposant des librairies de composants HTML/CSS, où l'utilisation du Javascript est limitée. Ce sont avant tout des frameworks de styling. Ces frameworks sont très pertinents de par la manière dont ils fonctionnent : le système de grille est un outil puissant, et la grille responsive (avec ses 3 ou 4 points de rupture selon le framework) fait des miracles au vu de l'effort demandé pour l'utiliser au niveau adaptation à la taille de l'écran. Ce qui différencie réellement ces types de framework les uns des autres sont leur documentation, leur communauté et leur aspect graphique. Ces trois points sont absoluments déterminants. Bootstrap, de par son utilisation très répandue, produira des sites au look moins original, mais si Bootstrap est aussi massivement utilisé c'est grâce à sa documentation exemplaire et sa communauté nombreuse. Foundation est moins facile à utiliser mais produit des sites au look moins "cloné". Cependant, pour une problématique d'adaptation pure, sans penser à l'aspect attractif et graphique des choses, on préférera un framework ayant fait ses preuves comme Bootstrap. Il faut enfin souligner que ces frameworks ne permettent (et surtout ne sont pas conçus) pour implémenter des comportements complexes, que ce soit au niveau des interactions ou au niveau architecture logicielle. Pour des projets complexes de grande envergure, on pourra combiner un framework graphique avec une technologie telle que Knockout.js ou des moteurs encore plus massifs tels que Angular et KendoUI. Pour les projets de moindre envergure, que l'on veut porter facilement sur différentes tailles d'écran, ces frameworks remplissent très bien leur rôle. Dans le cadre d'un projet comme un site de e-commerce proposant un grand catalogue, il faudra bien évidemment les combiner à autre chose.