Rapport Adaptation des IHM 2016 - 2017
Projet RPS

Thibaut SORIANO - Zhengqin YAN - Bénédicte LAGOUGE - Arnaud LEGRAND

Introduction

RPS est un logiciel à destination des auto-entrepreneurs et des TPE ayant des réparateurs se déplaçant à domicile (télévision, électroménager, ...).
Notre application s'adapte à votre environnement de travail : au bureau l'affichage sera par défaut en mode administratif, alors qu'en dehors du bureau vous aurez l'affichage technique par défaut.
Les utilisateurs visés par notre application sont les techniciens et les administratifs. Nous allons ainsi diviser l'application en deux interfaces distinctes afin de nous adapter au mieux aux besoins de chaque type d'utilisateurs.
Dans la partie réservée au technicien, il est possible de consulter les détails de l'intervention courante (remarques du client, travail à effectuer). L'entête des informations concernant l'intervention suivante est aussi visible, en miniature. Il est possible de saisir des détails concernant l'exécution de l'intervention en cours. S'il n'y a pas d'intervention au moment de la consultation, la prochaine intervention est affichée. Sur le panneau de l'intervention courante, il est possible d'indiquer si le client a payé.
Dans la partie réservée à l'administration, il y est possible de consulter un historique des interventions et de prendre un nouveau rendez-vous pour une intervention. On a accès au prix de chaque intervention, et on peut savoir pour chaque intervention si elle a été payée.

Natif (Android) & **Cross-Plateform (Xamarin)**

Afin d'adapter l'application à l'environnement de l'utilisateur, si elle détecte le Wifi de l'entreprise, elle sera par défaut en mode administratif, sinon elle sera par défaut en mode technicien. Il est toujours possible de changer de mode manuellement.

L'utilisation d'une technologie native permet d'assurer à l'utilisateur une expérience plus confortable de l'application, en conservant les conventions d'interface Android. Il est aussi plus simple d'accéder aux capteurs de l'appareil. L'inconvénient est de viser un public moins grand, composé seulement des utilisateurs d'appareils Android. A l'aide de layout appropriés, l'application peut adapter son affichage à différentes tailles d'écrans.

L'utilisation d'une technologie cross-plateform possède la force de développer l'application sur différents systèmes d'exploitation tout en conservant la majorité des fonctionnalités de ceux-ci . Cependant, il y aura probablement plus de difficultés à s'adapter aux spécifications de chaque système d'exploitation.

Scénario #1 :

Fernandino est un réparateur de machines à laver et il partage son temps de travail entre le bureau et les déplacements chez les clients.
En début de journée, il démarre du bureau et lance son application. Celle-ci détecte le WiFi du bureau et ainsi sait que Fernandino est au bureau . Fernandino a alors accès directement à l'emploi du temps du jour et peut visualiser les tâches qu'il a à réaliser.
Lorsqu'il est en déplacement chez un client, et qu'il regarde son application, il a accès aux informations de l'intervention courante, sait en un regard combien de temps il lui reste avant la prochaine intervention.
Fernandino peut bien entendu naviguer entre la vue "emploi du temps" et la vue "intervention courante" à tout moment s'il le souhaite.

Web Component (Angular2) & Responsive (Bootstrap)

Il y aura différentes étapes d'adaptations :

- 1 (desktop) - environnement administratif par défaut + présentation normale ;
- 2 (Tablette) - environnement technique par défaut + présentation normale ;
- 3 (Mobile) - environnement technique par défaut + présentation verticale ;

Adaptation de l'écran : Présentation normale / Présentation verticale.
Adaptation de l'utilisateur : Environnement administratif / environnement technique

Scénario #2 :
Jean est commercial chez RPS, il est au bureau et un client l'appel pour avoir un rendez-vous car sa machine à laver ne fonctionne plus. Jean regarde sur l'Intranet de sa société, sur son ordinateur de bureau, s'il y a du créneau de libre. En effet, une intervention est possible aujourd'hui entre 16h et 18h, le client est rassuré.
Après avoir raccroché avec le client, Jean appelle le technicien pour lui indiquer cette nouvelle intervention, mais celui-ci lui indique qu'il est malade. Jean sera donc obligé d'aller chez le client pour honorer le rendez-vous. RPS lui permet d'accéder à sa page depuis son téléphone mobile, celui lui simplifiera la vie. Il ne sera pas nécessaire d'apporter son PC portable. Arrivé sur place, il n'a plus qu'à sélectionner le menu "Intervention" est faire comme s'il était lui-même le technicien, il pourra même prendre une photo avec son téléphone et l'insérer dans le rapport de l'intervention. L'intervention fini, dans sa voiture, un autre client l'appelle pour un autre rendez-vous. Toujours sur son téléphone mobile, il pourra re-basculer en mode administratif, et ne pas attendre d'arriver au bureau.

Tutoriels

Natif (Android)

Outil de développement / test :

Il existe deux IDE principaux pour développer en Android : Android Studio et Eclipse. Android Studio étant basé sur Intellij IDEA, j'ai opté pour celui-ci.
Pour tester l'application au cours du développement, j'ai utilisé mon téléphone Android. En effet, il est possible d'envoyer l'application sur le téléphone (après avoir installé le pilote du téléphone sur l'ordinateur) afin de la tester.
Pour les personnes ne disposant pas d'un appareil Android, il est possible d'envoyer l'application sur un smartphone virtuel généré par Android Studio. C'est plus lent, plus lourd, et l'interaction avec l'appareil et plus difficile, en comparaison à l'utilisation d'un appareil physique.

Pour moi (et mon ordinateur portable), l'installation d'Android Studio a été longue, et beaucoup de mises à jour ont été nécessaires pour finalement faire compiler le projet. Il y a un grand nombre de versions différentes existantes, cela peut poser problème. Il faut choisir une version pour Gradle (le moteur de production), pour le Sdk d'Android. Ce dernier choix dépend de l'appareil cible, ayant un smartphone en version 4.4.2 d'Android (version 19), j'ai spécifié ceci dans le fichier "build.gradle".

```
android {
    compileSdkVersion 24
    buildToolsVersion "23.0.2"

    defaultConfig {
        applicationId "ihm.rps"
        minSdkVersion 19
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
```

Extrait du fichier "build.gradle" (versions)

Android Studio est régulièrement amélioré, et plusieurs composants sont mis à jour, cela implique que l'utilisateur doit souvent faire des mises à jour au lancement de l'IDE.

Récupérer l'exemple :

J'ai déposé l'exemple sur mon dépôt Github. Le lien est :
https://github.com/ThibautSoriano/RPS.git

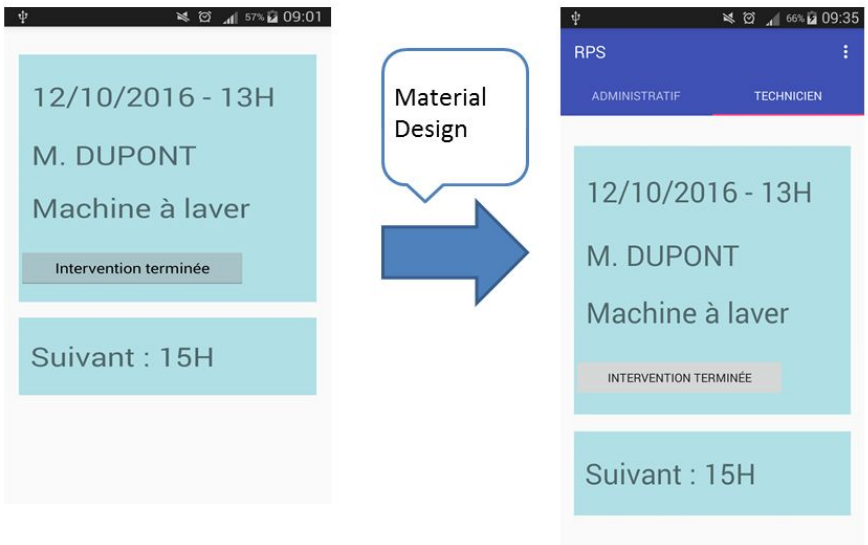
Technique pour tester le projet (après avoir installé Android Studio) :

- cloner le contenu du dépôt
- sur Android Studio, choisir import project (gradle), et donner le répertoire utilisé pour le clonage, en choisissant gradle.
- choisir un appareil virtuel (ou un physique, si possible) pour y envoyer l'application

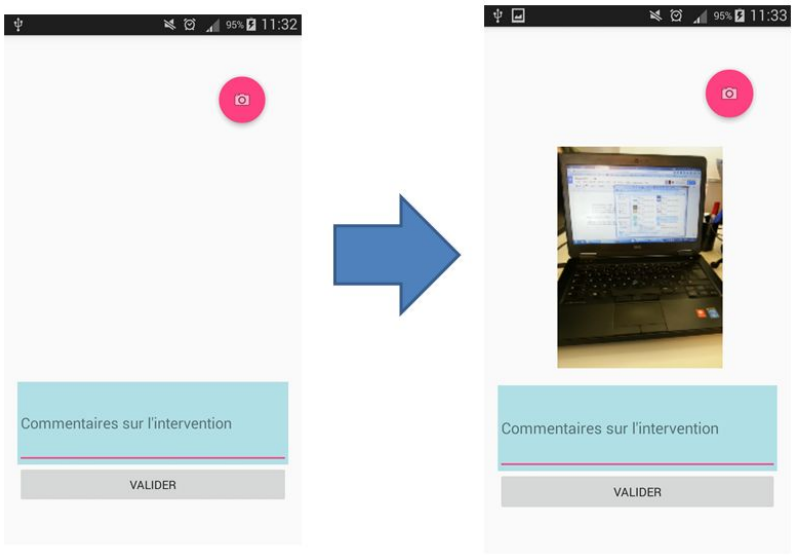
Etapes pour réaliser l'exemple :

L'application est divisée en deux vues : mode technicien et mode administratif. Sur l'application mobile, il faut mettre l'accent sur le mode technicien, j'ai donc commencé par là.

En partant d'un projet vierge, j'ai donc pu arriver à une vue contenant les éléments nécessaires (figure sur la gauche). C'est une vue trop simple. Lors d'un retour professeur, on m'a suggéré d'intégrer du Material Design pour Android. J'ai donc mis en place un système d'onglets :



Un des objectifs était d'utiliser les capteurs, de façon à comparer la facilité d'utilisation entre natif et cross-platform. J'ai donc fait une vue se déclenchant par un clic sur "intervention terminée" permettant d'utiliser l'appareil photo. J'ai souhaité afficher la photo prise dans une ImageView placée dans la vue, pour cela il a fallu surcharger la méthode "onActivityResult". C'était plutôt facile à trouver, ainsi que l'appel à l'appareil photo. Pour l'exemple, je ne fais qu'afficher la photo dans l'ImageView, mais il serait tout à fait possible de la stocker ou encore de l'envoyer sur un Cloud pour l'utilisateur (code dans ReportActivity.java). J'obtiens la vue suivante :



Toujours dans le même objectif d'utiliser les capteurs, j'ai utilisé le Wifi. Afin de donner les droits à l'application pour utiliser le Wifi, il faut ajouter deux lignes dans le Manifest.xml :

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Le code permettant de consulter le Wifi prend assez peu de place, et a marché du premier coup, c'était assez simple (code dans MainActivity.java).
Si on détecte le réseau Wifi "Unice-HotSpot", on lance l'application sur le premier onglet (mode administratif -> on suppose que l'utilisateur est à son travail, le nom du réseau Wifi est à titre d'exemple), sinon on la lance sur le deuxième onglet (mode technicien -> on suppose qu'il est en intervention à l'extérieur). Il est bien entendu possible de changer d'onglet facilement (par appui sur le titre de l'onglet, ou encore en glissant le doigt vers l'onglet souhaité).

Une fois la partie réservée au technicien terminée, j'ai commencé la partie dédiée à l'administratif. Cette partie doit contenir un emploi du temps. Etant donné que ce n'était pas la partie principale pour mon application (ce sont les technologies Web qui ont mis l'accent sur cette partie), j'ai souhaité la traiter rapidement. J'ai cherché un composant "emploi du temps", ou "calendrier", mais à part les classiques "Date Time Picker", je n'ai rien trouvé. Cette recherche étant infructueuse, je me suis rabattu sur une "ListView" classique, que je remplis à la volée.

Emploi du temps sous forme d'une simple liste

C'est un emploi du temps très rudimentaire, mais comme dit précédemment, ce n'était pas l'objectif principal de mon application.

Capacité d'adaptation :

Selon le sujet choisi, l'application doit s'adapter à deux choses : l'utilisateur (son rôle : technicien ou administratif) et l'environnement (en intervention à l'extérieur ou présent à son bureau).

L'application s'adapte au rôle de l'utilisateur, il y a en effet deux onglets, chacun d'entre eux correspond à un rôle.
L'application s'adapte donc à l'environnement par le Wifi. Si on se connecte au Wifi de l'école Unice-HotSpot (pour l'exemple), et qu'on lance l'application, elle s'ouvrira sur le premier onglet (administratif), et sinon, elle s'ouvrira sur le deuxième.

Mon avis :

J'ai trouvé très longue et laborieuse l'installation et la mise en place d'Android Studio, ainsi que d'un premier projet vierge compilant. Il y a une très grande diversité en versions, et un grand nombre de mises à jour à effectuer, c'est une richesse et un défaut à la fois. De mon point de vue, c'était surtout un défaut, car ça m'a pris du temps pour trouver des versions compatibles et venir à bout des mises à jour. J'ai trouvé assez complexe l'utilisation du système de design, peut-être parce que je ne m'en étais jamais servi auparavant.

J'ai trouvé très pratique l'utilisation des capteurs : c'était le point clé de mon application native, et ça a finalement été le plus rapide et le plus simple pour moi. J'ai trouvé riche le Material Design, donnant une apparence agréable.

Cross-Plateform (Xamarin)

1. Introduction

Xamarin est une plateforme mobile qui permet de développer des applications cross-platform. À ma connaissance il existe deux façons de développer en Xamarin :

- la première consiste à développer des applications en utilisant Xamarin Forms. Ce framework permet de développer des applications en C# et de les déployer sur la ou les plateformes ciblées (Android, iOS ou Windows Phone).

- la seconde consiste à développer des applications natives (pour les plateformes ciblées par le projet), tout en ayant une partie du code commune , écrite dans le langage C#.

La première façon de faire a l'avantage d'un développement rapide mais ne semble pas pouvoir convenir pour une application devant utiliser de nombreuses fonctionnalités spécifiques à une plateforme donnée.
La deuxième façon de faire permet des interfaces plus proches de chaque plateforme, car on doit développer une interface différente pour chaque plateforme.Elle nécessite cependant une connaissance du développement d'applications sur chaque plateforme visée.

Avant le projet, mes connaissances en programmation mobiles se limitaient à Android. De plus,en sachant que le temps imparti pour le projet était plutôt court et que l'application doit utiliser des fonctionnalités (appareil photo, accès à Internet) développables sous Xamarin Forms, j'ai choisi d'utiliser la première solution.

Ceci m'a ainsi permis de développer une application cross-platform Android/iOS et également de détecter les réelles forces et faiblesses d'utiliser uniquement Xamarin Forms, que j'expliciterai plus tard.

2. Détails de la réalisation de l'exemple

a. L'organisation du projet sous Xamarin Studio

Lors de la création d'un projet cross-platform, Xamarin Studio crée en fait quatre sous-projets : le premier contient le code commun en C# (Xamarin.Forms), les trois autres les codes natifs spécifiques aux trois plateformes couvertes par Xamarin (Android,iOS,Windows Phone).
Dans le cadre de ce projet utilisant uniquement Xamarin Forms, nous utiliserons principalement le premier sous-projet.

b. Le développement de l'interface

Comme Android attribue à des vues des activités, Xamarin Forms attribue à des vues des pages.Une page est composée de deux fichiers :

- un fichier XAML, qui permet de déterminer la partie visuelle de l'interface.
- un fichier C#, qui permet de définir les interactions de l'utilisateur avec la vue définie le premier fichier.

L'interface de ce projet est divisée en deux vues principales, qui répondent chacune à un besoin utilisateur précis: la visualisation du planning à court terme lorsque l'utilisateur est en intervention, et la visualisation de l'emploi du temps lorsque l'utilisateur prend la casquette d'un administrateur.

i. La phase d'initialisation et les objets "backend"

La phase d'initialisation se fait dans la classe App.

App est une classe héritant de Application, et est la classe appelée par Xamarin.Forms pour initialiser l'application. Après avoir initialisé toutes les données que

nous voulons utiliser pour “remplir” l’application (des objets Client,Categorie et RDV permettant simplement de modéliser des rendez-vous), nous appelons la ligne suivante :

```
MainPage = new NavigationPage(new AdaptationDesIHMSPage());
```

L’appel de NavigationPage se fait pour pouvoir nous permettre par la suite de naviguer entre les pages de l’application (créer une page à partir d’une autre, revenir à la page précédente, etc).
Quand à MainPage, il s’agit d’un attribut de App qui permet de déterminer la page de commencement de l’application.Dans notre cas il s’agit de AdaptationDesIHMSPage.

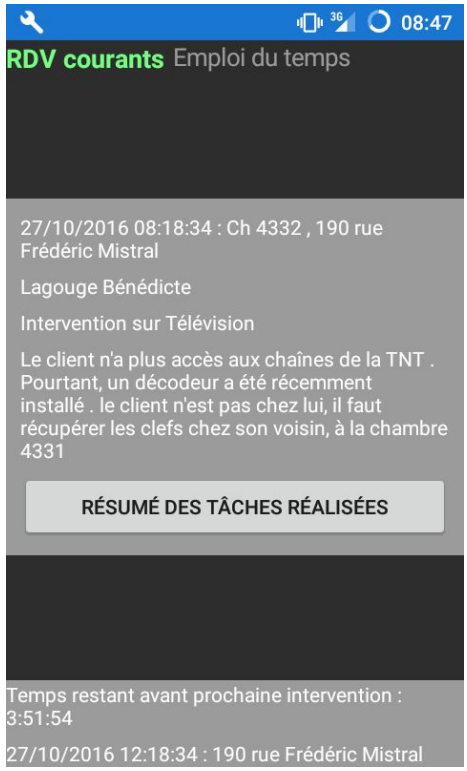
ii. Le déplacement entre les vues : présentation de la CarouselPage

Les deux vues principales sont explicitées dans la page AdaptationDesIHMSPage.Cette page a la particularité d’hériter du type de page CarouselPage, qui permet de se déplacer très simplement d’une vue à l’autre, en glissant son doigt sur l’écran de gauche à droite et de droite à gauche. Voici le XAML simplifié de AdaptationDesIHMSPage (nous rentrerons plus en détail dans le contenu de chaque vue par la suite) :

```
<?xml version="1.0" encoding="utf-8"?>
<CarouselPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="AdaptationDesIHMS.AdaptationDesIHMSPage"
              BackgroundColor="#2D2D2D">

    <ContentPage x:Name="page1">
        <!-- organisation de la vue "intervention courante"-->
    </ContentPage>


    <ContentPage x:Name="page2">
        <!-- organisation de la vue "emploi du temps"-->
    </ContentPage>

</CarouselPage>
```

Chaque ContentPage contenu par une CarouselPage correspond à une vue différente de cette CarouselPage.

iii. La vue “Intervention courante” et ses sous-vues



Cette vue est divisée en trois grandes parties.

La première est une entête permettant à l’utilisateur d’avoir un indicatif sur la vue actuellement affichée et dans quel sens il devra glisser son doigt pour accéder à l’autre vue.

La deuxième est le résumé de l’intervention courante en train d’être effectuée. Elle contient toutes les informations disponibles de celle-ci.Le bouton de cette partie permet d’accéder à la validation de l’intervention(nous verrons cela plus en détail bientôt).

La troisième partie est composée du bref résumé de l’intervention qui suivra (la date/heure et l’adresse pour permettre à l’utilisateur de situer cette intervention dans le temps et l‘espace) et d’un compteur qui compte à rebours le temps restant avant cette intervention.

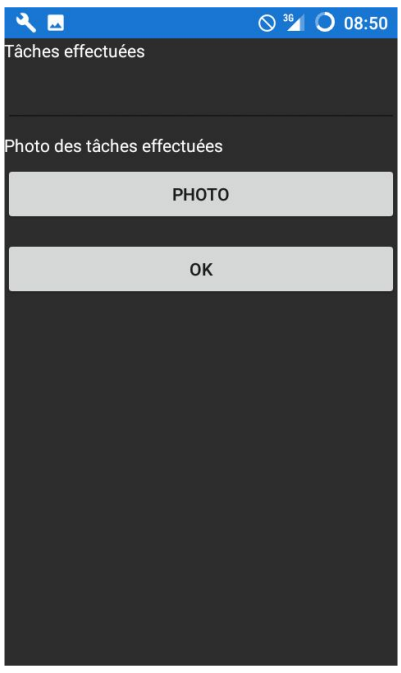
En regardant de plus près le bout de XAML définissant cette vue, on retrouve bien la même structure, chaque partie étant une StackLayout :

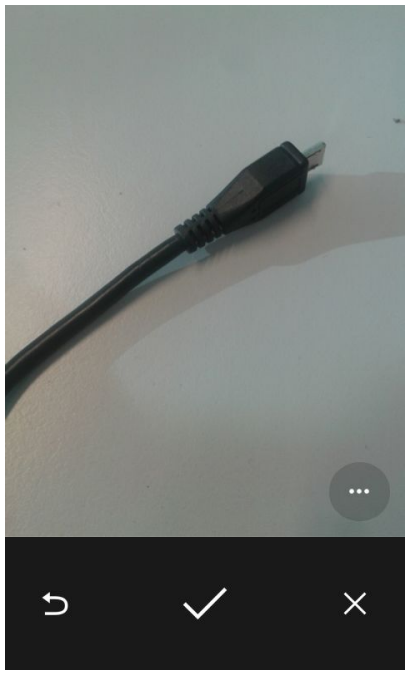
```
<StackLayout>
        <StackLayout Orientation="Horizontal" VerticalOptions="Start">
            <Label Text="RDV courants" FontAttributes="Bold" TextColor="#75FF83" FontSize="18"/>
            <Label Text="Emploi du temps" HorizontalOptions="End" TextColor="#9B9B9B" FontSize="17"/>
        </StackLayout>

        <StackLayout VerticalOptions="CenterAndExpand" BackgroundColor="Blue" Padding="10">
            <Label x:Name="enteteCourant" TextColor="White"/>
            <Label x:Name="clientNomPrenomCourant" TextColor="White"/>
            <Label x:Name="categorie" TextColor="White"/>
            <Label x:Name="remarquesCourant" TextColor="White"/>

            <Button Text="Résumé des tâches réalisées" x:Name="finished" Clicked="onButtonFinished"/>
        </StackLayout>

        <StackLayout VerticalOptions="End" BackgroundColor="Blue" >
            <StackLayout>
                <Label x:Name="compteur" Text="COMPTEUR" TextColor="White" />
            </StackLayout>
            <Label x:Name="resumeSuivant" TextColor="White"/>
        </StackLayout>
    </StackLayout>
```

Comme l'indique le XAML,le bouton "Résumé des tâches réalisées" déclenche la méthode onButtonFinished() lorsqu'il est cliqué, cette méthode est définie dans la partie C# de AdaptationDesIHMSPage et crée une nouvelle vue de la façon suivante :

```
Navigation.PushAsync(new InformationsRDV());
```

Voici cette vue :



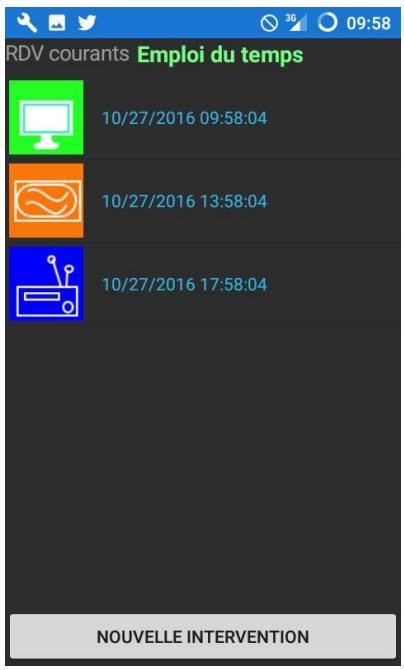


Elle permet dans un premier temps de renseigner par écrit les tâches effectuées lors de l'intervention puis de prendre une photo de ce qui a été réalisé. Lorsqu'on appuie sur le bouton du bas, les informations sont enregistrées et on revient automatiquement à la vue

principale, qui sera légèrement modifiée : l'ancien rendez-vous suivant devient le courant, et un nouveau rendez-vous prend place en tant que rendez-vous suivant.
Pour ce qui est de la gestion du compteur, on initialise le temps restant comme le nombre seconde entre la date de la prochaine intervention et de l'intervention courante, puis on utilise l'objet Device.StartTimer en lui indiquant d'effectuer chaque seconde la vérification du temps restant.Si il reste du temps, on diminue d'une seconde.Sinon, ça veut dire que l'utilisateur est en retard et une notification est lancée pour l'en informer.

```
Device.StartTimer(TimeSpan.FromSeconds(1), () =>
{
    if (temps_restant > 0)
    {
        updateCountDown();
        return true; //continue
    }
    else {
        var Notifier = CrossLocalNotifications.Current;
        Notifier.Show("Retard", "La prochaine intervention est censée commencer maintenant");
        return false; //not continue
    }
});
```

iv. La vue "emploi du temps" et ses sous-vues



Cette vue est constituée de trois parties.
La première partie est l'entête qu'on retrouve également dans la vue précédente.
La deuxième partie est une liste de rendez-vous qui correspond à l'emploi du temps de la journée.
La troisième partie est un bouton permettant d'accéder à la fonctionnalité d'ajout d'une nouvelle intervention.

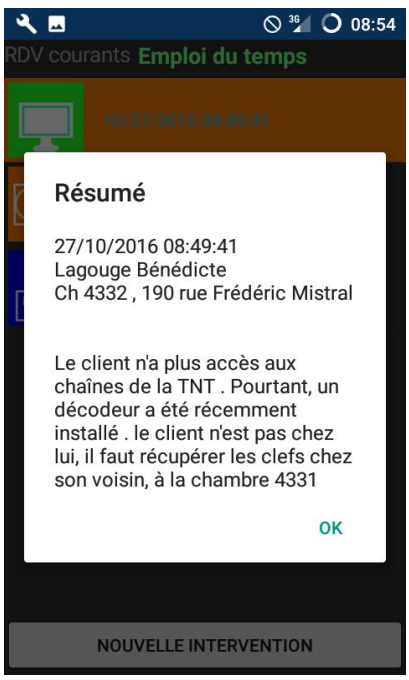
Voici le XAML correspondant :

```
<StackLayout Orientation="Vertical" x:Name="header2">
    <StackLayout Orientation="Horizontal" VerticalOptions="Start" >
        <Label Text="RDV courants" HorizontalOptions="Start" TextColor="#9B9B9B" FontSize="17"/>
        <Label Text="Emploi du temps"
            HorizontalOptions="End"
            FontAttributes="Bold"
            TextColor="#75FF83"
            FontSize="18"/>
    </StackLayout>

    <ListView x:Name="rdv_list">
        <ListView.ItemTemplate>
            <DataTemplate>
                <ImageCell ImageSource="CategorieRDV.Image_ressource" Text="{Binding DateRDV}"/>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>

    <Button Text="Nouvelle intervention" x:Name="button_nouvelle_intervention" Clicked="onNewIntervention"/>
</StackLayout>
```

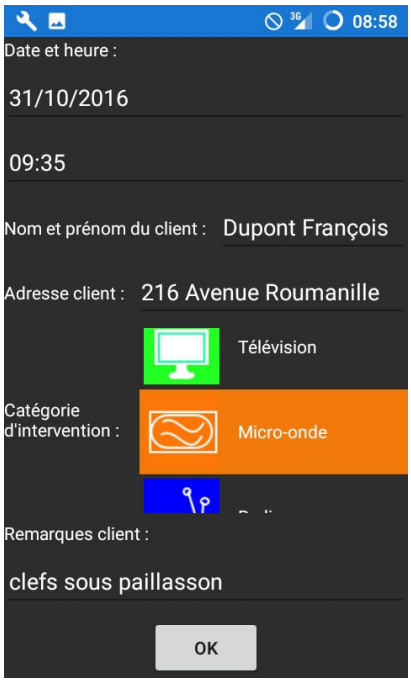
Chaque élément de la liste est un bref résumé d'un rendez-vous : la catégorie est indiquée par une image, puis viennent l'heure et l'adresse.
À cette liste a été associé une méthode OnItemTapped qui permet d'afficher les informations complètes du rendez-vous sélectionné.

```
this.rdv_list.ItemTapped += OnItemTapped;

private void OnItemTapped(object sender, ItemTappedEventArgs e)
{
    if (e.Item != null)
    {
        RDV rdvTapped = (RDV)e.Item;
        String str = rdvTapped.DateRDV + "\n"
                        + rdvTapped.ClientRDV.Nom +"\n"
                        + rdvTapped.ClientRDV.Adresse + "\n\n\n"
                        + rdvTapped.ClientRDV.Remarques;
        DisplayAlert("Résumé",str, "OK");
    }
}
```

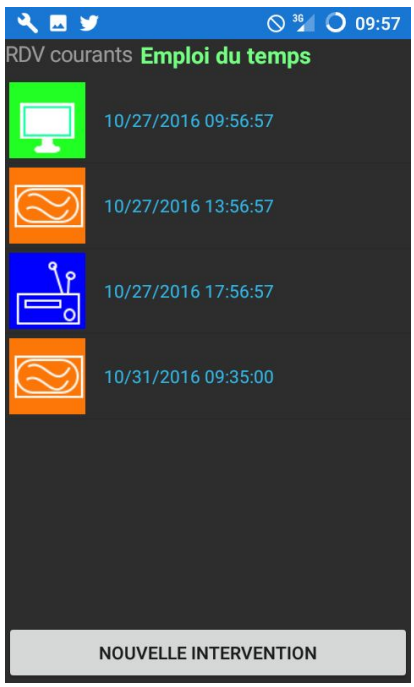
Ainsi voici ce qu'on obtient lorsqu'on clique sur un élément de la liste :



Le bouton “Nouvelle intervention” permet d’accéder à une instance de NouvelleInterventionPage.
Voici cette vue :



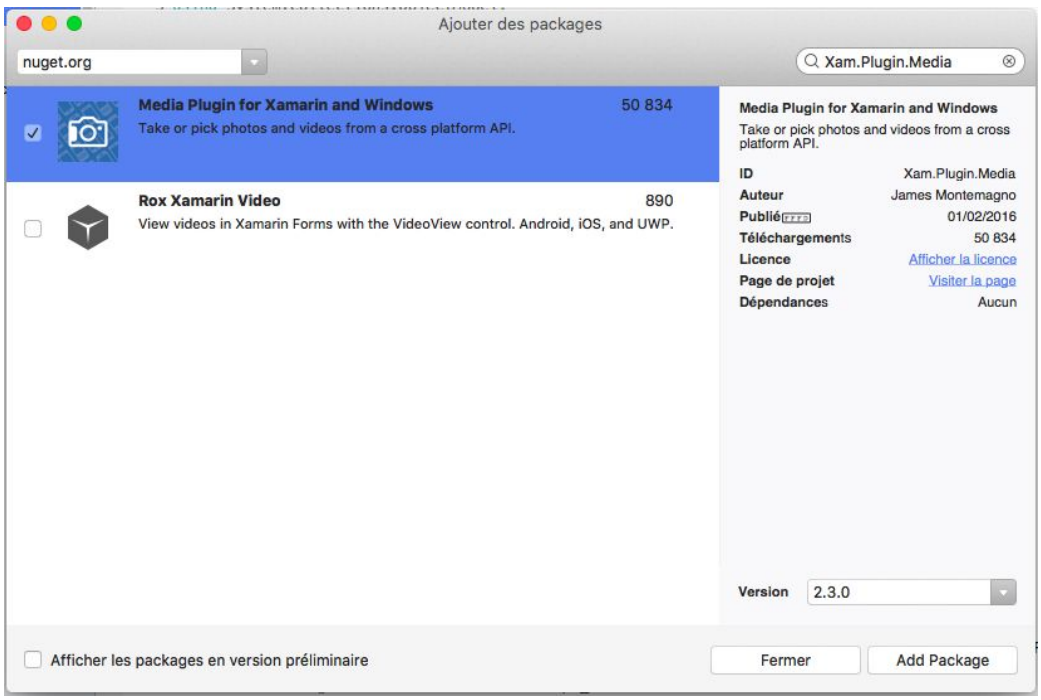
Elle permet de rentrer les différentes informations nécessaires pour définir un rendez-vous, puis de valider ce rendez-vous en appuyant sur le bouton du bas. En appuyant sur ce bouton, on enregistre le nouveau rendez-vous dans une collection de la classe statique InfosRDVPersist et la liste de la vue principale est mise à jour pour inclure ce nouveau rendez-vous.



c. L’accès aux capteurs via des librairies

Pour ce projet, j’ai dû accéder à l’appareil photo et au gestionnaire de réseau.
Il faut savoir que Xamarin Forms seul ne permet pas ce genre de fonctionnalités. Il y a donc deux choix : coder en natif l’accès aux capteur et relier ce code au code C#, ou utiliser une des nombreuses librairies additionnelles mise à notre disposition pour accéder aux capteurs via Xamarin Forms.
Restant dans la logique d’utiliser uniquement Xamarin Forms,j ’ai choisi la deuxième solution.

Pour utiliser correctement chacune des librairies citées ci-dessous, il faut les installer dans le sous-projet contenant le code C# et tous les sous-projets correspondant aux plateformes visées.
Cette application devant être déployée sur Android et iOS, les librairies ont été installées dans les trois sous-projets correspondants. Pour installer une librairie, il suffit de faire un clic droit sur le répertoire “Packages” du projet et cliquer sur “Ajouter un package”.Dans la fenêtre qui apparaît, il suffit alors de rechercher puis sélectionner les packages devant être installés :



La première librairie qui a été utilisée est le plugin Xam.Plugin.Media, dont le code source et le tutoriel se trouvent à cette adresse : https://github.com/jamesmontemagno/MediaPlugin.

Ce plugin supporte les version d'Android supérieures à 14 et les versions iOS et iOS Unified supérieures à 7.

Après avoir installé la librairie dans les trois sous-projets, il faut ajouter les permissions d'accès à la caméra. Pour ce faire :

- sur Android : modifier le fichier AndroidManifest en y ajoutant ces deux lignes permettant l'accès à la caméra de l'appareil et à la possibilité d'écriture de fichier dans la mémoire du téléphone (pour enregistrer les photos)

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.CAMERA" />
```

- sur iOS : modifier le fichier Info.plist en autorisant les champs `NSCameraUsageDescription` et `NSPhotoLibraryUsageDescription`.

Une fois les permissions ajoutées, nous pouvons commencer à utiliser la librairie dans notre sous-projet Xamarin Forms.

Pour appeler les fonctionnalités de cette librairie, il faut appeler en début de fichier :

```
using Plugin.Media.Abstractions;
```

Voici le code pour lancer l'appareil photo, prendre une photo, l'enregistrer et l'afficher. Les commentaires expliquent le fonctionnement du code :

```
//création d'un événement lorsque qu'un bouton est cliqué
button_photo.Clicked += async (sender, args) =>
      {
        //vérification de la disponiblité de l'appareil photo
          if (!CrossMedia.Current.IsCameraAvailable || !CrossMedia.Current.IsTakePhotoSupported)
          {
              DisplayAlert("No Camera", ":( No camera avaialble.", "OK");
              return;
          }

         //lancement de la tâche de prise de photo.La photo est alors stockée dans le téléphone
    //dans le dossier PhotosAppli, et pour nom [dateactuelle].jpg
          var file = await CrossMedia.Current.TakePhotoAsync(new Plugin.Media.Abstractions.StoreCameraMediaOptions
          {

              Directory = "PhotosAppli",
            Name = DateTime.Now.ToString()+".jpg"
          });

    //si le fichier est null c'est qu'on a décidé de ne pas prendre de photo.
          if (file == null)
              return;

    //sinon on l'affiche.
          InfosRDVCourantPersist.ImagePath = file.Path;

          image.Source = ImageSource.FromStream(() =>
          {
              var stream = file.GetStream();
              file.Dispose();
              return stream;
          });
      };
```

La deuxième librairie utilisée est le plugin Xam.Plugin.Connectivity, dont le code source et le tutoriel se trouve à cette adresse : https://github.com/jamesmontemagno/ConnectivityPlugin

Il faut également ajouter une permission au sous-projet Android pour utiliser cette librairie : il faut pour ce faire modifier le fichier Assembly.info pour y rajouter la permission nécessaire :

```
[assembly: UsesFeature("android.hardware.wifi", Required = false)]
```

L'utilisation faite de cette librairie est plutôt simple dans le cadre de ce projet : au lancement de l'application, on regarde si le téléphone est bien connecté à un WiFi et on commence l'application sur la bonne vue :

```
if (Plugin.Connectivity.CrossConnectivity.Current.IsConnected)
    this.CurrentPage = this.page2;
else
    this.CurrentPage = this.page1;
```

ci-dessus, page1 étant la vue "Intervention courante" et page2 la vue "Emploi du temps"

3. Avec quel outil de développement, de tests ?

On peut développer sous Xamarin en utilisant soit Xamarin Studio (sur Mac ou Windows) , soit Microsoft Visual Studio. Étant donné que mon but aura été de créer une application pouvant être déployée sur Android et iOS, j'ai choisi d'utiliser Xamarin Studio sur Mac. Ceci a l'avantage d'avoir un émulateur iOS directement intégré à Xamarin Studio, ce qui facilite les tests.
Afin d'installer Xamarin Studio sur Mac, il suffit de le télécharger à cette adresse : https://www.xamarin.com/download .

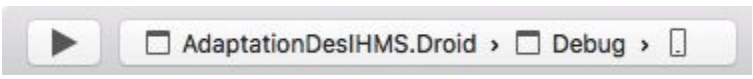
L'application est compatible à tout appareil Android à partir de la version 19 et à tout appareil iOS à partir de la version 5. Elle a été testée en déployant l'application via Xamarin Studio sur un smartphone Android de version 5.1.1 et un émulateur d'iPhone.
Les tests sont assez compliqués à réaliser sur l'émulateur (lenteur de l'émulateur,latences et plantage fréquents), mais tout fonctionne parfaitement sur le système réel Android. Ainsi, l'application a été testée en profondeur sur Android, mais je pense que si quelqu'un déploie l'application sur un iPhone il aura accès à toutes les fonctionnalités.

4. Comment déploie-t-on et exécute-t-on l'exemple ?

Expliquer la façon dont Xamarin Forms se déploie sur un système donné (piti schéma)

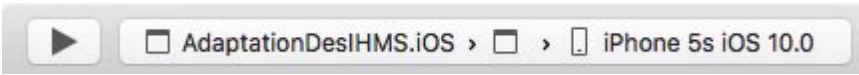
a. Sur Android

Via Xamarin Studio, il suffit de sélectionner les paramètres suivants dans le menu en haut à gauche et lancer le déploiement en cliquant sur le bouton Play.



b. Sur iOS

Je n'ai pu réussir qu'à utiliser un émulateur de Xamarin Studio et pas un système réel.
J'ai utilisé l'émulateur pour l'iPhone 5S, car il semblerait que ce soit l'iPhone le plus utilisé jusqu'à présent.
Pour ce faire, il faut utiliser les paramètres suivants dans le menu en haut à gauche lancer le déploiement en appuyant sur le bouton Play.
Il faut prendre son mal en patience car ceci est très lent et a tendance a causer pas mal de latences sur l'ordinateur.



5. Comment teste-t-on les capacités d'adaptations ?

a. Adaptation au support

La première adaptation qui peut venir à l'esprit lorsqu'on parle de technologie cross-platform est bien entendu l'adaptation au support. C'est également celle sur laquelle j'ai passé le plus d'efforts, malgré les difficultés de tests sur émulateur iOS.

En effet, l'exemple que j'ai développé s'adapte à deux plateformes : Android et iOS . Cette adaptation a été travaillée en essayant de créer une application qui puisse convenir aux deux plateformes.
Il a donc fallu par exemple prendre en compte le fait que la différence du nombre et des rôles des boutons physiques des systèmes Android et iOS. L'interface a donc été pensée pour ne pas avoir besoin d'utiliser de bouton physique...mais n'entrave pas l'utilisation de ceux-ci : si on utilise Android, on pourra toujours revenir en arrière avec le bouton physique correspondant.
Je n'ai pas vérifié si l'adaptation se fait correctement sur Windows Phone, mais vu que le code a été exclusivement développé à l'aide Xamarin Forms, il suffirait juste de rajouter les librairies et les permissions nécessaires au sous-projet Windows Phone pour pouvoir déployer sur cette plateforme.

b. Adaptation à l'utilisateur

Il s'agit de la deuxième adaptation sur laquelle j'ai concentré mon effort.

Comme indiqué dans la partie décrivant les interfaces, une réflexion importante a été attachée à l'adaptation de la visualisation d'un planning selon deux utilisations différentes : la gestion d'emploi du temps et la visualisation de l'intervention courante.
Ainsi, l'utilisateur en mode "intervention" va avoir rapidement accès en un regard aux informations concrètes dont il a besoin pour faire son travail.
Quand à l'utilisateur en mode "administration", il va avoir accès à une interface lui permettant de gérer simplement les rendez-vous.

c. Adaptation à l'environnement

En récupérant une information correspondant à l'environnement direct du téléphone (détection de connexion à un WiFi), on peut alors deviner si l'utilisateur est rentré au bureau ou pas. Ceci permet une utilisation plus intuitive de l'interface : l'utilisateur n'a pas à changer de vue, l'application le fait automatiquement pour lui.

On peut tout de même noter un petit bémol sur cette partie, qui est dûe à la limitation de la librairie utilisée (ou de mon manque de prise en main de celle-ci) : l'application détecte uniquement si le téléphone est relié par WiFi à une connexion quelconque, pas une connexion précise.

6. Petite conclusion sur les avantages et inconvénients de Xamarin Cross Platform

Avantages :

1. Portable sur plusieurs plateformes (ici testé sur Android et iOS) grâce à une surcouche en C#
2. Économie de développement et pas la peine de connaître en profondeur le fonctionnement des OS sur lesquels on veut développer.
3. Accès à un ensemble de librairies disponible pour Xamarin.Forms qui permettent d'ajouter des fonctionnalités communes aux plateformes sur lesquelles on veut porter l'application (notifications, photos)

Inconvénients :

1. Pertes de performances par rapport à du natif
2. Difficultés d'accès aux capteurs

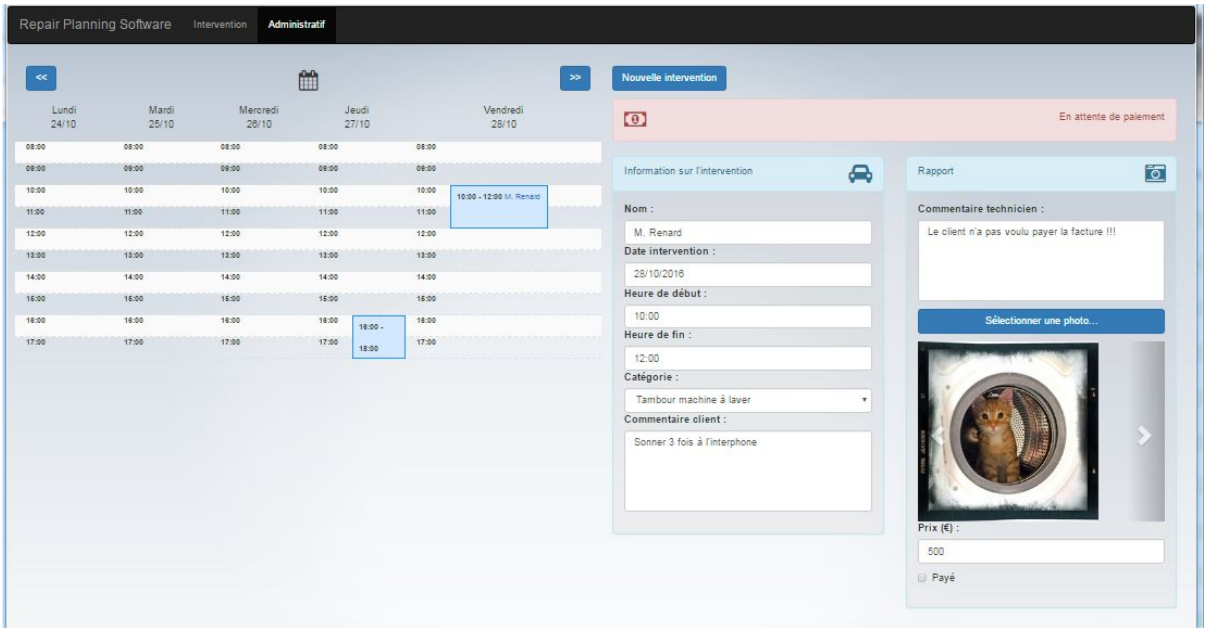
Responsive (Bootstrap)

Mise en place de l'environnement de travail

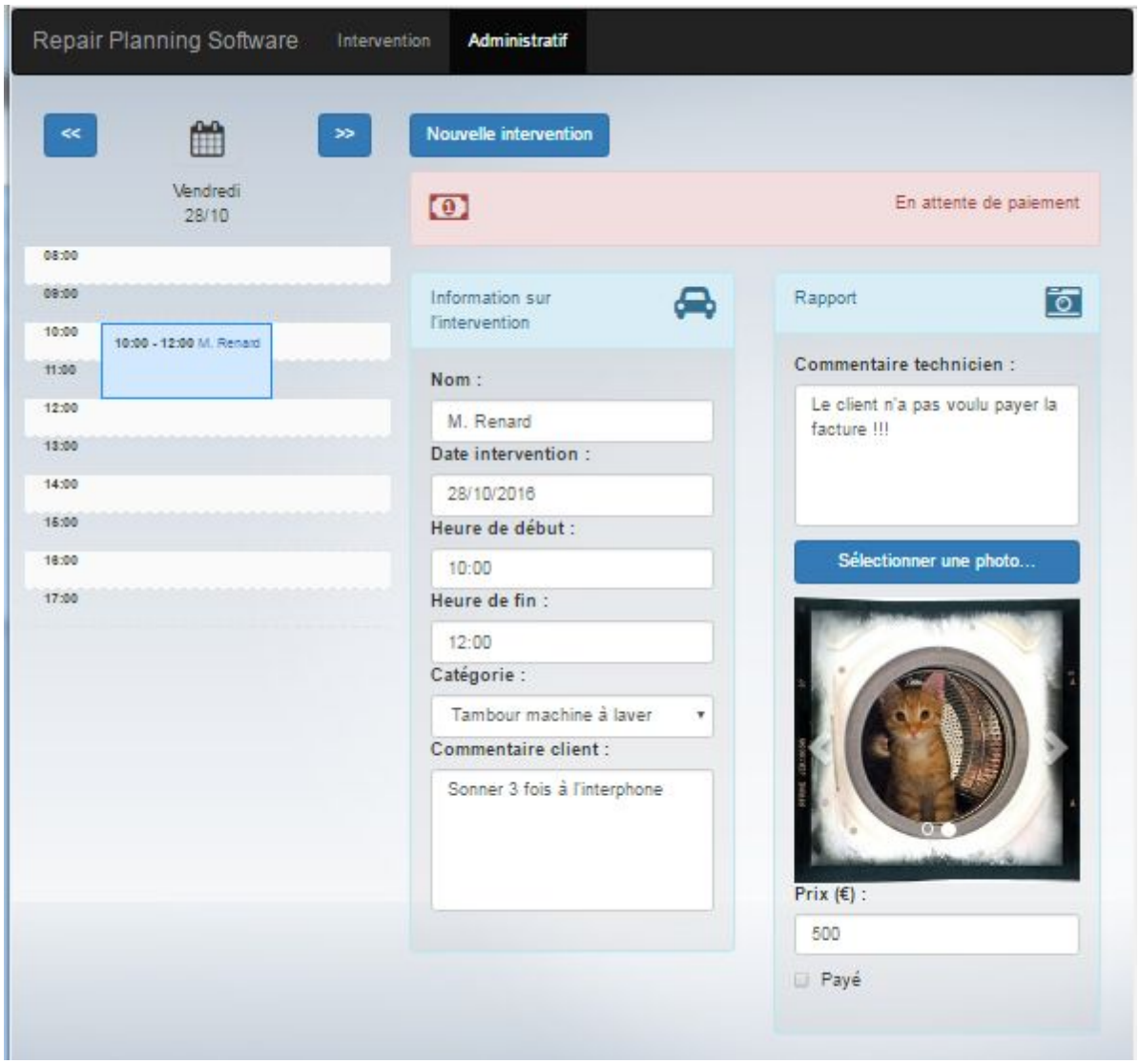
Pour ce projet, nous avons choisi d'utiliser Eclipse Neon et le serveur Web TomCat.
Créer un projet Web dynamique Java. Définir un nouveau serveur "TomCat" pour avoir l'intégration à Eclipse.
Dans le répertoire \WebContent, ajouter deux nouveaux fichier "administratif.jsp" et "intervention.jsp" correspondant à nos pages Web.

Page "administratif"

Visuel à obtenir :



Pour une tablette :



Nous allons commencer par la page administrative. Elle se comporte de 5 parties :

- Les feuilles de styles (.css dans le header de la page)
- Le menu de navigation
- Le calendrier
- Le détail d'une intervention
- Les fichiers Javascript en bas de page et fichier à part

Les feuilles de styles et les fichiers Javascript

Pour intégrer Bootstrap à notre page il est nécessaire d'ajouter aussi des dépendances. Comme : JQuery, JQuery UI, Boostrap, notre CSS propre (rps.css), et d'autre CSS correspondant aux différents composant que nous allons utiliser.

```
<link rel="stylesheet" href="css/jquery-ui.min.css">
<link rel="stylesheet" href="css/bootstrap.min.css">
<link rel="stylesheet" href="css/font-awesome.min.css">
<link rel="stylesheet" href="css/calendar.css">
<link rel="stylesheet" href="css/bootstrap-datetimepicker.min.css" />
<link rel="stylesheet" href="css/rps.css">
```

Une méta information est nécessaire aussi pour permettre les fonctionnalités nomades (Touch zooming, ...)

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

Pour les liens vers les fichiers Javascript, c'est à la fin du body qu'il faut les mettre :

```
<script type="text/javascript" src="js/jquery.min.js"></script>
    <script type="text/javascript" src="js/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/bootstrap.min.js"></script>
    <script type="text/javascript" src="js/underscore-min.js"></script>
    <script type="text/javascript" src="js/language/fr-FR.js"></script>
    <script type="text/javascript" src="js/calendar.min.js"></script>
    <script type="text/javascript" src="js/moment-with-locales.min.js"></script>
    <script type="text/javascript"
src="js/bootstrap-datetimepicker.min.js"></script>
    <script type="text/javascript"
         src="js/locales/bootstrap-datepicker.locale-perso.js"></script>
    <script type="text/javascript" src="js/administratif.js"></script>
```

On pourra remarquer qu'il n'y a pas qu'un fichier JS Bootstrap... Le nombre de fichier JS paraît plutôt important par rapport à ce qu'on aurait pu s'attendre, au vu de la petite taille de l'application. Cela va aussi ralentir le chargement de la première page du site Web.

Le menu de navigation

Pour afficher le menu de navigation en haut de la page, qui s'adapte lorsqu'on est sur un téléphone mobile, on utilise la classe 'navbar', avec en plus la classe 'navbar-inverse' pour avoir le fond noir.

A l'intérieur de la balise <body> ajouter, en premier, le menu :

```
<nav class="navbar navbar-inverse">
```

A l'intérieur de cette balise, il faut définir le titre qui s'affiche à gauche, et les menus, qui sont remplacés par trois traits verticaux si l'écran est très petit. Le titre est en entier (Repair Planning Software) quand l'écran le permet, si non on le raccourcit (RPS). Pour avoir ce comportement, nous avons utilisé la classe 'row'.

Le calendrier

Bootstrap ne propose pas de calendrier/planning par défaut, il faut se tourner vers des composants qui utilisent Boostrap (par exemple : Boostrap Calendar).

https://github.com/Serhioromano/bootstrap-calendar

Le comportement que nous souhaitons avoir est, lorsque la taille le permet, on affiche 5 jours, sinon, si c'est trop petit, on n'affiche qu'un seul jour. Par défaut ce comportement n'est pas possible avec le composant Calendar. Pour ce faire, il faut définir 5 calendriers en mode "jour", les placer les uns à côté des autres, puis en faire disparaître 4 quand la résolution de l'écran diminue.

```
<div class="col-xs-12 col-sm-12 col-md-4 col-lg-4 col-calendrier">
                <div class="jour-entete">
                Vendredi<br />28/10
                </div>
        <div id="calendarVe" class="jour-perso"></div>
        </div>
```

Un calendrier se défini via une commande Javascript. Cette commande remplace la balise concerné par le composant Calendar :

```
var calendarVe = $("#calendarVe").calendar({
 view: 'day',
 tmpl_path: '/web-bootstrap/tmpls/',
```

```
        tmpl_cache: false,
        day: '2016-10-28',
        time_start: '08:00',
        time_end: '18:00',
        time_split: '60',
        events_source: 'events.json'
    });
    calendarVe.setLanguage("fr-FR");
    calendarVe.view();
```

Le calendrier utilise la technologie des IFrame. Ce qui nous oblige à repenser l'agencement de notre page. Car à l'origine nous souhaitions avoir le calendrier à gauche, sur grand écran, et en haut sur petit écran, mais ce deuxième point ne sera pas possible car lors du redimensionnement, le calendrier n'est pas pris en compte au niveau hauteur, et les détails se mettent derrière le calendrier. Pour corriger ce problème nous devons mettre le calendrier à droite, pour qu'il soit en-dessous des détails en petite taille. Mais il faudra décaler les balises du calendrier à gauche dans le cas où l'écran est grand. Ce qui nous donne :

```
<div class="col-xs-12 col-sm-4 col-sm-pull-8 col-md-6 col-md-pull-6 col-lg-6 col-lg-pull-6">
```

Au passage on peut remarquer la lourdeur de l'écriture Bootstrap, les classes s'accumulent pour faire ce que nous voulons. Pour ne pas avoir de surprise il est préférable de définir tous les cas possibles...

A noter aussi, pour ne pas avoir de problème avec les grandes tailles d'écran, en tout cas au niveau largeur, il ne faut pas oublier de définir le conteneur principal avec la classe 'container-fluid', et non pas 'container', car si non le contenu reste au centre de l'écran, il ne prend pas toute la largeur.

La définition des 'row' à 12 colonnes nous a un peu posé problème car nous avions 5 jours + 2 panels de détails (donc 7 colonnes). Nous ne voulions pas introduire trop d'espace. Nous avons donc ajouté des 'row' dans des 'row'. Pour les jours du calendrier on a été "contraint" de définir que le jour sélectionné serait deux fois plus grand que les autres jours, pour prendre toute la largeur. Ce qui n'est pas plus mal en finalité.

Problème rencontré : Le padding est trop important au niveau du calendrier, on ne voit pas les détails. *Solution* : Pour résoudre ce problème nous avons créé des classes CSS pour modifier le comportement des 'col-' Bootstrap et du calendrier. Il faut supprimer aussi le padding et la marge, puis diminuer la taille de la font du calendrier :

```
.col-calendrier {
padding-right: 0px;
        padding-left: 0px;
        margin-right: 0px;
        margin-left: 0px;
}
.container-perso {
margin: 10px;
}
.jour-perso {
font-size: x-small;
}
```

Nous précisons cela pour mettre en évidence que pour avoir le comportement souhaité, nous devons mettre les mains dans le CSS.

Pour ne pas avoir l'entête de la colonne sur le format "Jour", pour le personnaliser, il faut modifier le fichier \WebContent\tmpls\day.html pour supprimer les lignes suivantes au début du fichier :

```
<div class="row-fluid clearfix cal-row-head">
        <div class="span1 col-xs-1 cal-cell"><%= cal.locale.time %></div>
        <div class="span11 col-xs-11 cal-cell"><%= cal.locale.events %></div>
</div>
```

Lors de la mise en place des entêtes des jours, certains entêtes passe automatiquement sur 2 lignes. Il faut donc passer toutes les entêtes sur 2 lignes.

Le composant Calendar permet d'insérer des rendez-vous (events) de différentes manières. Nous choisissons d'utiliser l'injection d'un fichier JSON, qui en production serait généré dynamiquement côté serveur.
Problème rencontré : lorsque que plusieurs rendez-vous sont présents le même jour, en mode "jour", les rendez-vous ne sont pas simplement empiler les uns en dessous des autres, mais en plus décalé vers la droite. Comme la taille du calendrier n'est pas assez grande, il n'y a que le premier rendez-vous du jour qui est bien positionné au niveau des heures, les suivants sont placés aléatoirement en dessous. *Solution possible* : faire un fork du composant pour corriger le problème d'affichage.

Ajouter des boutons de navigation et un sélecteur de date pour pouvoir choisir n'importe quelle date. Pour cela il faut récupérer la librairie bootstrap datepicker https://eonasdan.github.io/bootstrap-datetimepicker/.
Problème rencontré : Le datepicker ne peut pas se mettre sur une image ou un bouton. *Solution* : il faut donc définir un input comme datepicker, le cacher, et le faire ré-apparaître furtivement lors de l'appui sur le bouton du calendrier.

```
$("#imgDate").on("click", function() {
        $('#calendarDate').show();
        $('#calendarDate').focus();
        $('#calendarDate').hide();
});
```

Le détail d'une intervention
Pour les détails, il faut définir deux panels avec la classe 'panel-info', un pour les informations génériques sur l'intervention, et un autre pour le rapport (avec photo). A rajouter aussi, une classe personnelle 'background-transparent' pour avoir un effet de transparence que je n'avais pas par défaut.

Ajout de la bibliothèque d'image Font Awesome, que nous allons utiliser pour nos icônes :

```
<link rel="stylesheet" href="css/font-awesome.min.css">
```

Au niveau des entêtes des détails, on ajoute une icône. Pour que les icônes soient à droite on utilise une 'row' (10 pour le texte, 2 pour l'icône).
Problème rencontré : l'image est soit trop éloigné du bord droit, soit dépasse le bord droit ! *Solution* : ajouter un alignement à droite sur le div de l'image, et passer les colonnes en 8/4. Le texte de l'entête sera toujours sur 2 lignes.

Ajout d'un carrousel dans le rapport pour afficher les photos prises lors de l'intervention.
Ajout d'une alerte si on est en attente de paiement.
Ajout d'un bouton pour sélectionner une photo
https://www.abeautifulsite.net/whipping-file-inputs-into-shape-with-bootstrap-3 .
Ajout d'une image de fond global à la page :

```
body {
background-image:
        url(../img/background.jpg);
background-size: cover
}
```

Ajout des contrôles des détails (nom, date, commentaire, ...) à l'intérieur d'un 'form-group'. Un élément d'un formulaire se définit avec deux balises, une balise 'label' avec l'attribut 'for' qui pointe sur la deuxième balise ('input' par exemple).

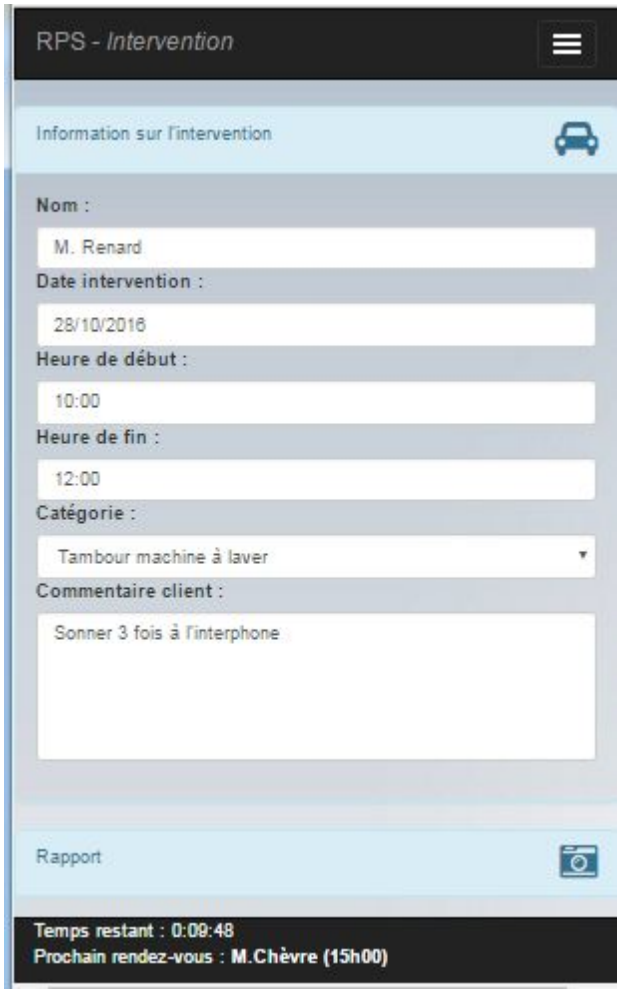
Problème rencontré : Le 'textarea' est par défaut redimensionnable. Ce qui amène un comportement incohérent, car le 'textarea' peut être plus grand que le panel ! *Solution* : définir une classe personnelle 'noresize', et l'ajouter à la balise en question.

```
.noresize {
resize: none;
}
```

Ajout de code JS fake pour avoir une expérience utilisateur (ajouter une intervention, cliquer sur une intervention existante, ...).

Page "Intervention"

Visuel à obtenir :



Au niveau des dépendances, nous avons un CSS commun avec la page administratif. Mais le JS personnalisé est propre à cette page.

Pour la partie intervention, Il faut dupliquer le code de la partie administrative concernant les détails de l'intervention.
Supprimer le bouton "nouvelle intervention"

Puis ajouter d'un footer, pour gérer l'affichage du temps restant avant le début de l'intervention, et l'affichage du nom du prochain client. Pour cela, utiliser un 'navbar' avec la classe 'navbar-fixed-bottom' en plus.

Tests

Pour tester, nous avons utilisé les fonctionnalités avancés du navigateur Chrome (outils de développement). Ce qui permet de vérifier le code HTML et Javascript généré par notre site Web, mais aussi de simuler différents matériels (téléphones portables, tablettes, ...)

Déploiement

Copier les fichiers suivant dans TOMCAT_HOME/webapps :

- target/web-bootstrap-1.0.0.war
- Tout le répertoire WebContent/

Conclusion

Avantages :

- Visuel agréable
- S'adapte facilement à la taille de l'écran
- Adapté pour les sites Web avec une présentation simple

Inconvénients :

- Accumulation des classes dans les balises HTML

- On est souvent limité par le Framework
- Comportement par défaut pas toujours adapté

Et demain…Bootstrap 4

- Prise en compte des écrans extralarges
- Intégration de FlexBox (alignement verticale, …)

Web composants (Angular2)

A. Introduction de Angular2

Angular2 est un framework pour construire des applications clientes en HTML et le langage JavaScript/TypeScript (TypeScript est un superset de JavaScript).

Le framework se compose de plusieurs bibliothèques, certains d'entre eux “core” et en option.

Nous écrivons des applications Angulaires en composant des modèles HTML avec le balisage “Angularized”, l'écriture des classes de composants pour gérer ces modèles. Ajout d'une logique d'application dans les services et les services de boxe et de composants dans les modules.

B. Démarrage de Angular2

J’ai suivi le tutoriel officiel de Angular2 pour démarrer l’application.

J’ai installé Npm et NodeJs (en dernières versions) pour pouvoir installer les paquets et tourner le serveur. (figure B.1)

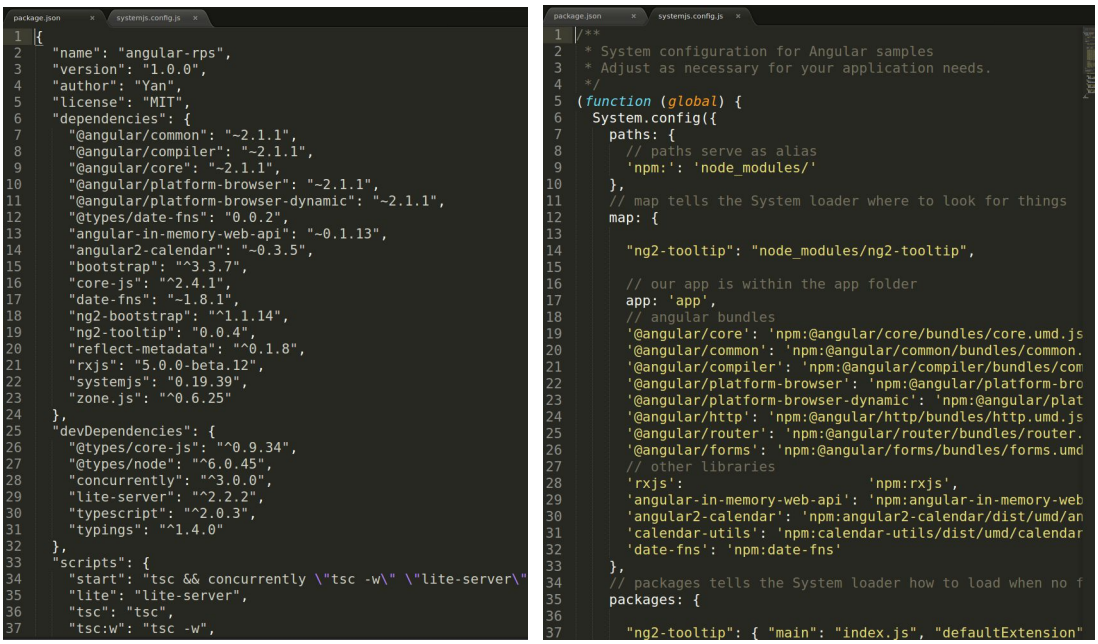
```
user@Axxx:~/Angular2/RPS_Adaptation_Origin$ npm -v
3.10.8
user@Axxx:~/Angular2/RPS_Adaptation_Origin$ nodejs -v
v6.9.1
```

Figure B.1

Une application Angular2 a plusieurs fichiers de configurations:

1. *package.json* identifie paquet NPM dépendances pour le projet.
2. *tsconfig.json* définit la façon dont le compilateur “typescript” génère JavaScript à partir des fichiers du projet.
3. systemjs.config.js fournit des informations à un module loader où trouver les modules d'application, et enregistre tous les paquets nécessaires. Il contient également d'autres paquets qui seront nécessaires plus tard.

Pour installer les paquets et les modules desquelles j’ai besoin, j’ai configuré *package.json* et *systemjs.config.js (figure B.2)*



```
{
  "name": "angular-rps",
  "version": "1.0.0",
  "author": "Yan",
  "license": "MIT",
  "dependencies": {
    "@angular/common": "~2.1.1",
    "@angular/compiler": "~2.1.1",
    "@angular/core": "~2.1.1",
    "@angular/platform-browser": "~2.1.1",
    "@angular/platform-browser-dynamic": "~2.1.1",
    "@types/date-fns": "0.0.2",
    "angular-in-memory-web-api": "~0.1.13",
    "angular2-calendar": "~0.3.5",
    "bootstrap": "^3.3.7",
    "core-js": "^2.4.1",
    "date-fns": "~1.8.1",
    "ng2-bootstrap": "^1.1.14",
    "ng2-tooltip": "0.0.4",
    "reflect-metadata": "^0.1.8",
    "rxjs": "5.0.0-beta.12",
    "systemjs": "0.19.39",
    "zone.js": "^0.6.25"
  },
  "devDependencies": {
    "@types/core-js": "^0.9.34",
    "@types/node": "^6.0.45",
    "concurrently": "^3.0.0",
    "lite-server": "^2.2.2",
    "typescript": "^2.0.3",
    "typings": "^1.4.0"
  },
  "scripts": {
    "start": "tsc && concurrently \"tsc -w\" \"lite-server\"
    "lite": "lite-server",
    "tsc": "tsc",
    "tsc:w": "tsc -w",
```

```
/**
 * System configuration for Angular samples
 * Adjust as necessary for your application needs.
 */
(function (global) {
  System.config({
    paths: {
      // paths serve as alias
      'npm:': 'node_modules/'
    },
    // map tells the System loader where to look for things
    map: {

      "ng2-tooltip": "node_modules/ng2-tooltip",

      // our app is within the app folder
      app: 'app',
      // angular bundles
      '@angular/core': 'npm:@angular/core/bundles/core.umd.js
      '@angular/common': 'npm:@angular/common/bundles/common.
      '@angular/compiler': 'npm:@angular/compiler/bundles/com
      '@angular/platform-browser': 'npm:@angular/platform-bro
      '@angular/platform-browser-dynamic': 'npm:@angular/plat
      '@angular/http': 'npm:@angular/http/bundles/http.umd.js
      '@angular/router': 'npm:@angular/router/bundles/router.
      '@angular/forms': 'npm:@angular/forms/bundles/forms.umd
      // other libraries
      'rxjs':                      'npm:rxjs',
      'angular-in-memory-web-api': 'npm:angular-in-memory-web
      'angular2-calendar': 'npm:angular2-calendar/dist/umd/an
      'calendar-utils': 'npm:calendar-utils/dist/umd/calendar
      'date-fns': 'npm:date-fns'
    },
    // packages tells the System loader how to load when no f
    packages: {

      "ng2-tooltip": { "main": "index.js", "defaultExtension"
```

Figure B.2

La dernière étape de configuration est “npm install” en commande pour télécharger les paquets.

Pour démarrer l’application, il faut un *“index.html”* que le lite-serveur peut appeler. Il faut aussi un fichier “*app.module.ts*” qui est le “entry point” de l’application et les composants fichiers.

Après créer le “*index.html*” et les composants, on fait “npm start” en commande pour démarrer l’application sur serveur localhost.

Voici ce que j’ai eu la première fois que j’ai lancer mon application après des simples configurations. (Figure B.3)

Figure B.3

C. Réalisation de l'exemple (outils de développement)

Comme expliqué au début du rapport, principalement notre application permet auto-entrepreneur de consulter des interventions et de prendre un nouveau rendez-vous pour une intervention.

Nous avons considéré trois cas d'utilisations possibles:

1. (desktop) - environnement administratif (souvent dans son bureau);
2. (Tablette) - environnement technique;
3. (Mobile) - environnement technique;

Comme mes deux collègues travaillent sur l'application mobile, j'ai décidé de me focaliser sur l'environnement administratif pour que notre utilisateur puisse utiliser notre application dans son bureau avec un desktop.

C.a. Bootstrap

J'ai choisi d'utiliser "bootstrap" comme mon collègue Arnaud. Mais mon but principale n'est pas de réaliser "Responsive Design". Je peux me profiter d'utiliser les "widgets" offerts par bootstrap et aussi son grid system.
Voici ce que j'ai configuré pour utiliser bootstrap.

Dans les balises <head> de" index.html"

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
```

Dans "package.json"

```
"bootstrap": "^3.3.7",
```

J'ai fait "npm install" pour installer "bootstrap" sur mon app.

C.b. Calendrier

J'ai utilisé "angular2-calendar" (un composant angular2 publié sur internet, offert par mattlewis92) pour lister toutes les interventions de l'entreprise.

Dans "package.json"

```
"angular2-calendar": "~0.3.5",
```

Dans les balises <head> de" index.html"

```
<!-- 1. Load libraries -->
<link rel="stylesheet" href="node_modules/angular2-calendar/dist/css/angular2-calendar.css">
```

Dans app.module.ts

```
import { CalendarModule } from 'angular2-calendar';
```

C.c. Tooltip

J'ai utilisé "ng2-tooltip" pour que l'utilisateur puisse avoir plus d'information s'il met son pointeur sur les widgets.

Dans "package.json"

```
"ng2-tooltip": "0.0.4",
```

Dans app.module.ts

```
import { TooltipModule } from "ng2-tooltip";
```

Voici l'architecture de mon application. (Pour me bien profiter de réutilisation des composants, j'ai séparé les composants et je les ai mis dans des sous-dossiers)

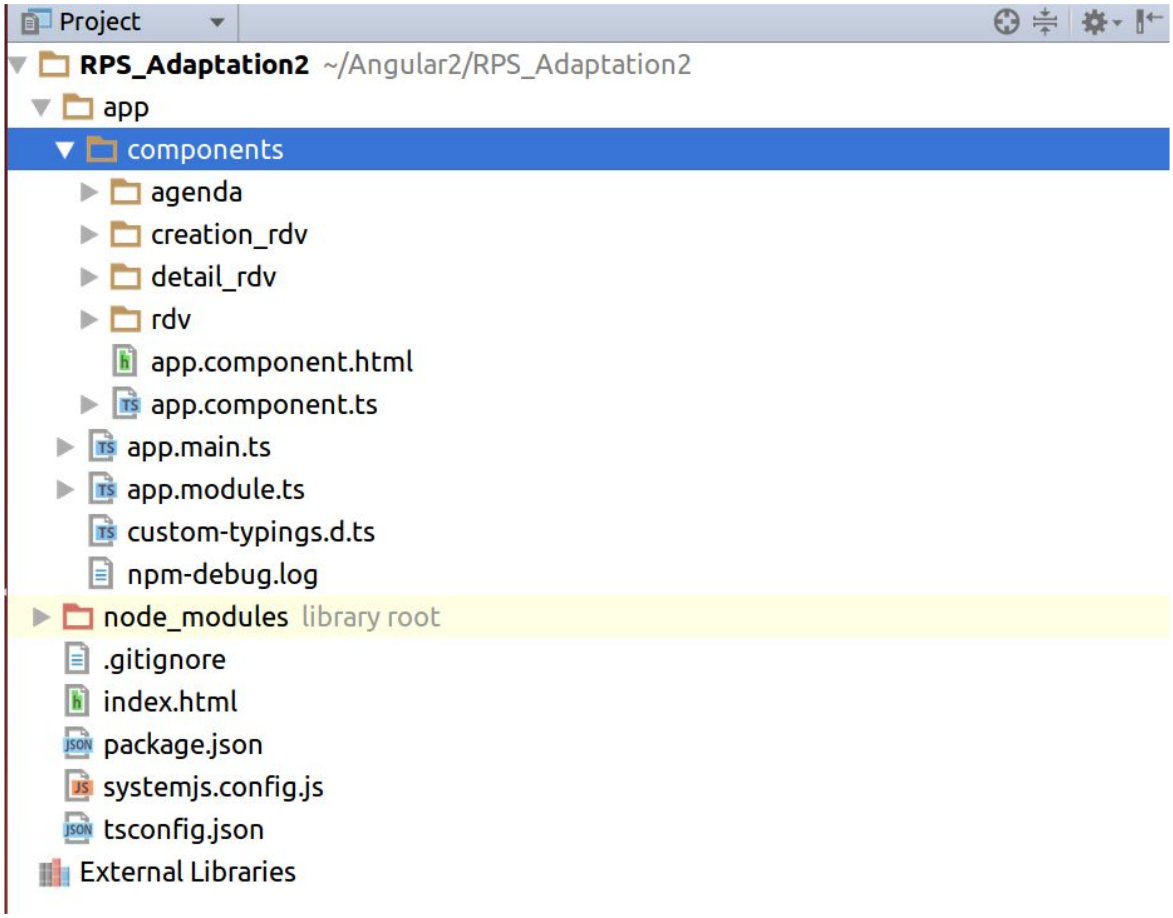


Figure C.1

C.d. Outil de test

J'ai utilisé "npm-test" comme outil de test. Quand on tape *"npm test [-- <args>]" en commande, le script test de ce paquet (qui est "-- <args>")* va se lancer.

Voici l'interface avec "angular2-calendar", "tooltip" en utilisant "bootstrap3":

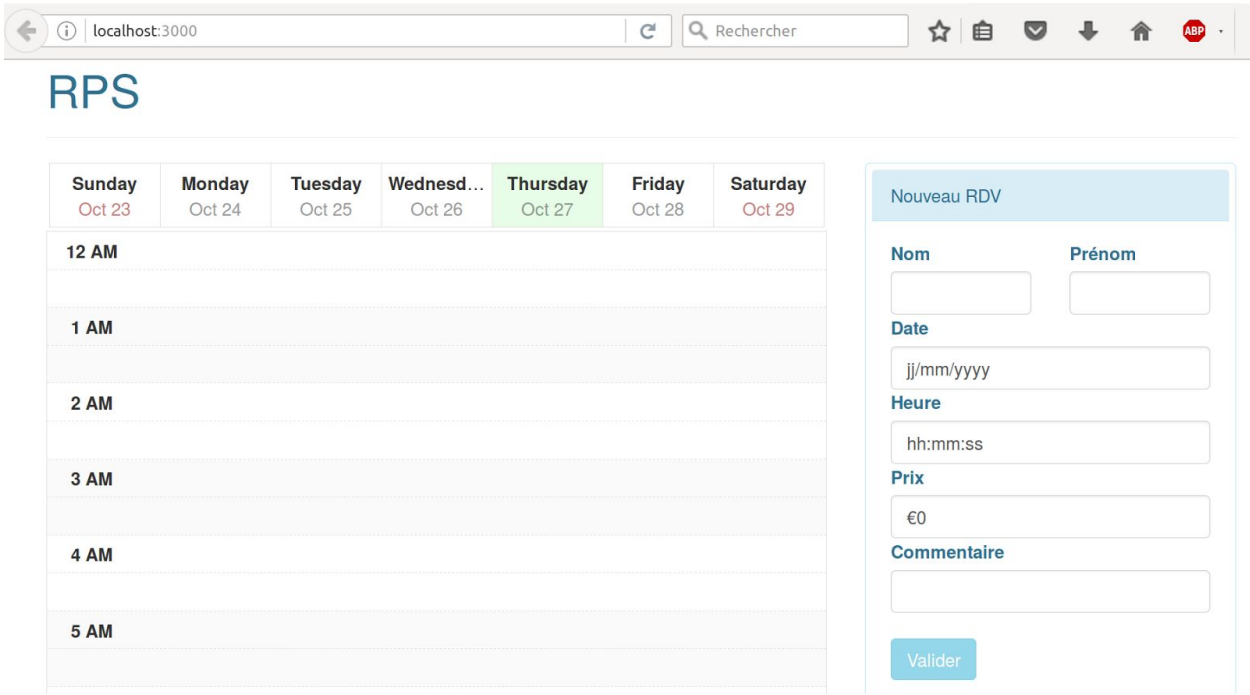


Figure C.2

D. Avantages et Inconvénients

Avantages :

1. Composants Web réutilisables : Nous pouvons amortir les coûts de développement par réutilisation. Ça rend aussi plus maintenable en cas de changement du code. Par exemple, le développement du mode "technicien" sera plus facile car le développeur peut réutiliser les composants de mode "administratif".
2. Un framework complet avec une grande communauté : Angular2 nous offre beaucoup de fonctionnalités puissantes et la facilité de coopérer avec les autres technologies (JQuery, Bootstrap, ...) Il nous offre aussi un excellent tutoriel officiel. Par exemple, l'application peut s'adapter facilement aux différents écrans grâce à bootstrap.

Inconvénients :

1. Un manque de stabilité : Comme Angular2 a été en version beta il n'y a pas très longtemps, il n'est pas encore très stable. Ce qui demandera au développeur de réajuster son code assez souvent.
2. Notre exemple demande juste une simple application avec peu de fonctionnalités. Ce n'est peut-être pas une bonne idée de mettre en place l'application par Angular2 en considérant sa lourdeur (beaucoup de choses à configurer) et sa vitesse de chargement. Angular2 est plutôt adapté à un gros site qu'à un petit site Web. (Au premier chargement, la page doit charger tous les modules et les scripts)

Synthèse

Si on regarde les deux technologies Web sélectionnées, un avantage d'Angular 2 est de pouvoir réutiliser facilement les composants. Car dans le cas de BootStrap, il a fallu faire un copier / coller des balises de la page Administratif vers la page Intervention, pour ensuite rajouter les éléments spécifiques à ce deuxième écran. Mais Angular 2 a pour défaut sa lourdeur, car même pour un petit site ne disposant que de quelques fonctionnalités, il faut charger tous les éléments proposés dans ses frameworks lors du premier chargement de la page.
Dans l'introduction, nous avons indiqué que la solution devait être capable de détecter si elle était lancé depuis le bureau ou chez le client, pour afficher un écran par défaut. Nous n'avons pas réussi à le mettre en oeuvre pour les technologies du Web. Pour les technologie plus proches du matériel, la solution de la détection du Wifi a été implémenté pour cette problématique.

Côté mobile, le principal avantage du natif est l'accès aux capteurs de l'appareil beaucoup plus simple. Du côté cross-platform, l'accès aux capteurs se fait plus difficilement mais on gagne en temps de développement,c'est un choix à faire selon l'application qu'on souhaite développer. De plus, une technologie native offre un design spécifique à la plateforme, contrairement à une technologie Cross-Platform, qui ne permet qu'un design plus générique mais toujours plus rapide à développer. Il serait cependant possible d'améliorer le design Cross-Plateform par l'ajout de feuilles de style pour chaque plateforme couverte. Enfin,le temps de lancement d'une application Cross-Platform est très élevé en comparaison à une technologie native.

De manière générale, les technologies choisies ont des objectifs différents en termes d'adaptation. Les technologies du Web permettent de limiter les problèmes liés à l'adaptation à la taille de l'écran mais restent imparfaites dans la communication avec le dispositif physique de l'appareil utilisé, où les technologies mobiles restent imbattables. Mais de plus en plus, le Web commence à rattraper son retard, par exemple, il semblerait qu'avec HTML5, il y ait des possibilités pour interagir avec l'appareil photo d'un Smartphone (que nous n'avons pas eu le temps d'étudier en profondeur).