RAPPORT PROJET TUTORÉ

SEMESTRE 4

Application mobile permettant aux étudiants de consulter leur emploi du temps et de créer des tâches pour facilement renseigner leurs devoirs

Eddy DRUET – B1

Année universitaire 2020 – 2021

TABLE DES MATIERES

INTRODUCTION

Dans ce rapport de projet tutoré, vous allez découvrir l'application mobile que j'ai réalisé par la présentation du déroulement du développement de celle-ci.

PRESENTATION DU PROJET

Le projet consiste en une application mobile pour Android et iOS. Celle-ci a pour objectif de proposer la consultation de son emploi du temps, et d'y gérer ses tâches à réaliser.

PUBLIC CIBLE

Le public cible par cette application mobile est les étudiants majoritairement, puisque tout l'intérêt et l'ergonomie sont organisés en ce sens, vous le verrez dans les fonctionnalités ci-dessous.

FONCTIONNALITES

Voici la liste des fonctionnalités principales souhaitées dans l'application mobile :

- **Consulter son emploi du temps**
  L'étudiant peut consulter son emploi du temps sous plusieurs temporalités : jour, semaine, mois.
  Le titre de la matière, l'heure, et la salle sont affichés.
- **Importer un emploi du temps**
  L'importation d'un calendrier extérieur est possible au sein de l'application, ex. un calendrier provenant de la plateforme Zimbra.
- **Consulter sa liste de tâches**
  L'étudiant peut consulter les tâches qu'il aura créées, et comme l'emploi du temps, elles sont consultables sous plusieurs temporalités : jour, semaine, mois.
  Les tâches sont en général associées à un cours spécifique dans l'emploi du temps.
  Elles peuvent être marquées comme « complétées ».
- **Ajouter des tâches**
  La création, la modification et la suppression de tâches est possible.
  Il est possible de définir le titre de la tâche, pour quel cours spécifique elle doit être associée (facultatif), la date et heure de début, une description, et trois modes de répétition : aucune répétition, la tâche est répétée à chaque cours spécifiquement désigné, ou encore à un jour précis (ex. tous les lundis à 8h).

Exemple concret d'utilisation de l'application :

J'ai importé mon emploi du temps de mes cours à l'IUT de Dijon-Auxerre sur l'application mobile. Il m'est maintenant possible de le consulter par celle-ci. Je sais qu'une interrogation écrite est proposée tous les débuts de travaux dirigés en cours d'Expression-communication, afin de ne jamais l'oublier, je crée une tâche correspondant à ce module avec un mode de répétition « chaque fois qu'il y a cette matière ».

FAIRE FACE A UNE PROBLEMATIQUE

La problématique principale qui m'a donné l'idée de réaliser ce projet, est qu'il m'est souvent arrivé d'oublier de noter des devoirs, car je n'avais pas forcément le temps d'écrire sur le bloc-notes alors qu'il était déjà l'heure de sortir de la salle. Pareillement, en cours d'Expression-communication, il y avait une interrogation écrite tous les débuts de travaux dirigés, mais je n'y pensais pas toujours.

L'attente majeure pour ce projet est ainsi une excellente ergonomie, c'est-à-dire, qu'il faut le moins de temps possible pour créer une tâche car cela permet de renseigner un nouveau devoir très rapidement en fin de cours par exemple.

Cela passera notamment par la possibilité de ne pas devoir à écrire le titre de la tâche, mais juste en cliquant sur un bouton écrivant le mot-clé « IE », « DS », « Partiel », « DM » …

Mais encore, avec les différents modes de répétition proposés.

CHOIX TECHNIQUES

Étant donné que le projet est une application mobile, de nombreux choix étaient envisageables, certains plus adaptés que d'autres, mais nous allons voir quels ont été mes critères et on va procéder par élimination.

CRITERE N°1 : ADEQUATION AVEC LE PROJET PERSONNEL ET PROFESSIONNEL (PPP)

L'attente première pour le projet tutoré de semestre 4, est qu'il soit adapté au PPP de l'étudiant.

Mon PPP est plutôt orienté vers le développement logiciel, c'est pourquoi cela élimine directement les technologies du web comme HTML, Javascript associés à un framework comme React Native par exemple.

Au contraire, il est préférable donc, de me diriger vers des technologies utilisées dans le domaine applicatif comme des langages tels que Java, C#, C++, Dart…

- ⇨ **Retenu pour le moment** : technologies domaine applicatif, Java, C#, C++

CRITERE N°2 : MON NIVEAU DE CONNAISSANCE DANS LES TECHNOLOGIES

L'intérêt du projet étant d'apprendre ou encore approfondir une technologie informatique, cela ne sert donc strictement à rien de reprendre une technologie que je maîtrise déjà très bien comme Java puisque nous l'avons étudié de fond en comble durant les deux années.

Au contraire, des langages comme C#, C++ sont des langages que je maîtrise de façon basique, tout comme le framework .NET. Cela m'a donc paru une très bonne idée, de les approfondir par le biais de ce projet.

- ⇨ **Retenu pour le moment** : C#, C++

CRITERE N°3 : COMPATIBILITE ANDROID ET IOS

Le développement d'une application mobile est très dépendant de la plateforme sur laquelle on développe, faute des technologies propriétaires des deux grands acteurs du marché : Google avec son système d'exploitation Android, et Apple avec iOS.

L'un utilise pour base le langage Java, tandis que l'autre utilise aujourd'hui, le langage Swift.

Cela nécessiterait de développer l'application deux fois, j'ai donc choisi des technologies permettant de développer pour les deux en même temps.

- ⇨ **Retenu pour le moment** : C#, C++ avec technologies compatibles à la fois Android et iOS

CHOIX FINAL

Les différentes technologies répondants aux critères précédents sont :

- Le langage C# avec le framework Xamarin

- Le framework Flutter

J'ai choisi le langage C# avec le framework Xamarin, car j'ai préféré en apprendre davantage sur C#, qui est plus utilisé dans le domaine applicatif que Flutter, et est également plus ancien.

⇨ **Technologies retenues** : langage C# associé au framework Xamarin

STRUCTURE DU PROJET

Dans cette partie, nous allons voir la structure adoptée pour le projet adaptée à l'utilisation optimale de C# et Xamarin.

ARCHITECTURE DU CODE

La méthode de conception du code conseillée par Xamarin est le « Modèle-vue-vue modèle ».

Celle-ci a pour objectif de séparer la vue (interface utilisateur), de la logique et de l'accès aux données ; notamment par l'usage de liaisons (bindings, que l'on verra à la partie Liaison couche métier, interface utilisateur) et d'événements (que l'on verra également à la partie Programmation du visuel des pages).

Pour simplifier, voici un schéma explicatif :

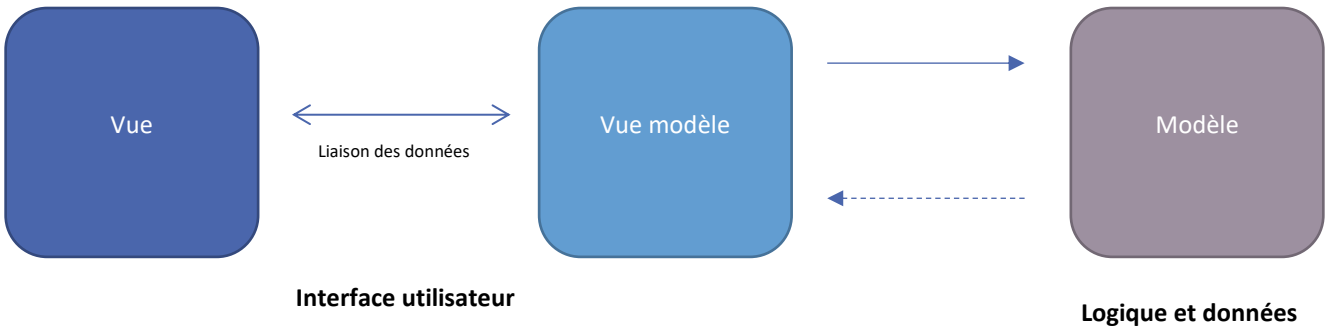


Schéma 1 : Modèle-vue-vue modèle

ARBORESCENCE

L'arborescence des fichiers du projet est :

- **Répertoire « Views »**
  Contient toutes les pages de l'application (accueil, calendrier, etc.)
  - **Répertoire « Components »**
    Contient tous les morceaux constituant l'interface utilisateur (en-tête, barre de navigation, bouton, etc.)
- **Répertoire « ViewModels »**
  Contient toutes les vues modèles associées aux vues, elles sont la liaison entre les données et l'interface utilisateur.
- **Répertoire « Models »**
  Contient tous les modèles, c'est-à-dire, les classes stockant les données ainsi que la logique de l'application.
- **Répertoire « Adapters »**

Contient tous les adaptateurs entre des bibliothèques extérieures et les modèles de l'application. Ils permettent de convertir les objets fournis par une bibliothèque en les objets de notre application.

DESCRIPTION DU PLAN

Pour la suite de ce rapport, nous allons voir les différentes phases que j'ai suivi pour le développement de l'application, je vais y expliquer les principes utilisés, et montrer comment je l'ai ai mis en œuvre pour développer les fonctionnalités.

Entre autres, nous verrons :

1) **Développement de l'interface utilisateur**
   Démarche depuis la création des maquettes, jusqu'à l'intégration en C# avec Xamarin.
   Explication des principes propres à la création d'interface utilisateur sous Xamarin.
2) **Développement des fonctionnalités**
   L'interface utilisateur étant réalisée, la démarche concernant le développement des fonctionnalités y sera expliquée.
3) **Liaison des fonctionnalités avec l'interface utilisateur**
   L'interface utilisateur et les fonctionnalités étant développées, il restera à les relier ensemble par le biais de différents moyens pratiques offerts par Xamarin.

DEVELOPPEMENT INTERFACE UTILISATEUR

La première phase du développement de ce projet, a été de réaliser, toute la partie interface utilisateur, de la conception en fil de fer pour réfléchir à l'agencement des pages, et à l'ergonomie ; puis une maquette du design avec réflexion sur le style graphique, code couleurs… et enfin l'intégration de l'interface utilisateur finale sur Xamarin.

MAQUETTE WIREFRAME[1]

En premier lieu, puisque l'application, pour rappel, exige une très grande ergonomie, j'ai réfléchi à celle-ci en réalisant une maquette en fil de fer à l'aide du logiciel Adobe XD, un outil de conception d'expérience utilisateur.

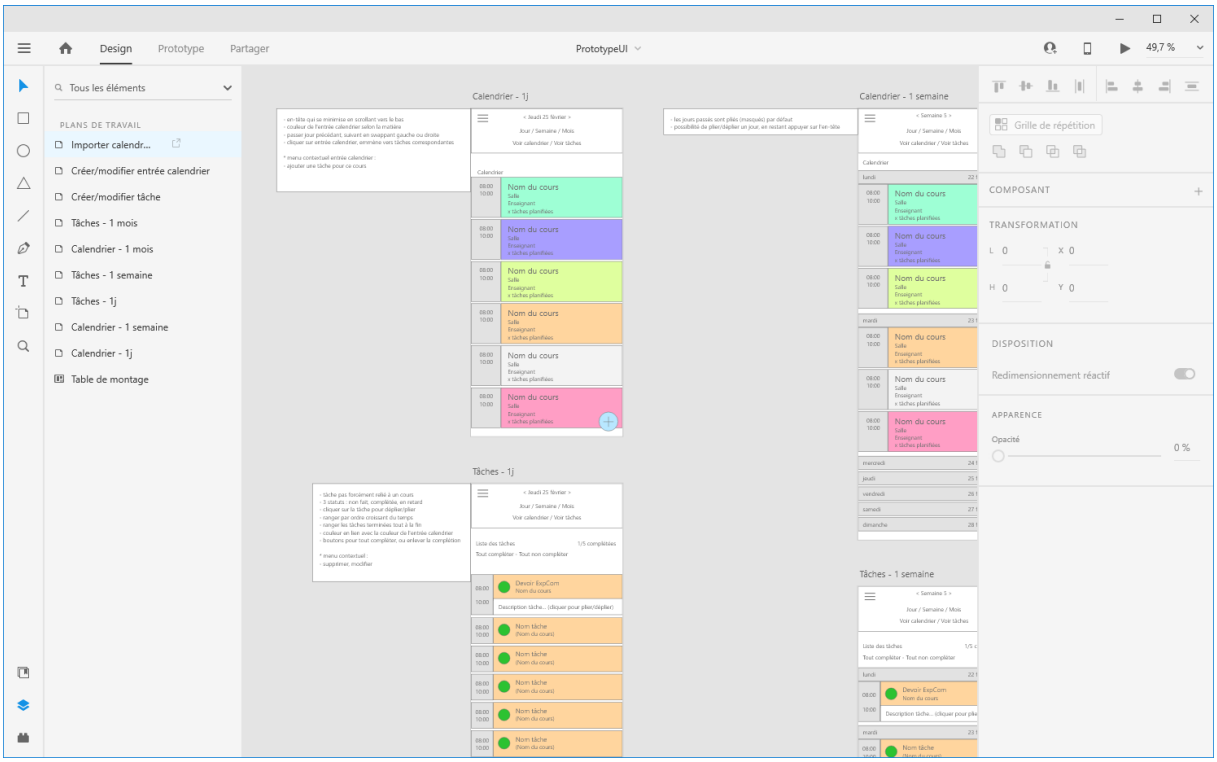


Illustration 1 : Adobe XD

J'ai commencé par réfléchir par les pages que je devais proposer en accord avec les fonctionnalités voulues, puis j'ai réfléchi à l'agencement des blocs constituants la page, afin que l'utilisateur puisse créer et consulter ses tâches le plus rapidement.

Les prochaines pages passent en revue chaque page de l'interface utilisateur, avec un encadré qui explique globalement les actions possibles par l'utilisateur.

[1] Fil de fer

CALENDRIER, VUE JOURNALIERE

C'est la page d'accueil, la première page lorsque l'on lance l'application. Elle affiche l'emploi du temps en vue journalière à la date du jour par défaut.

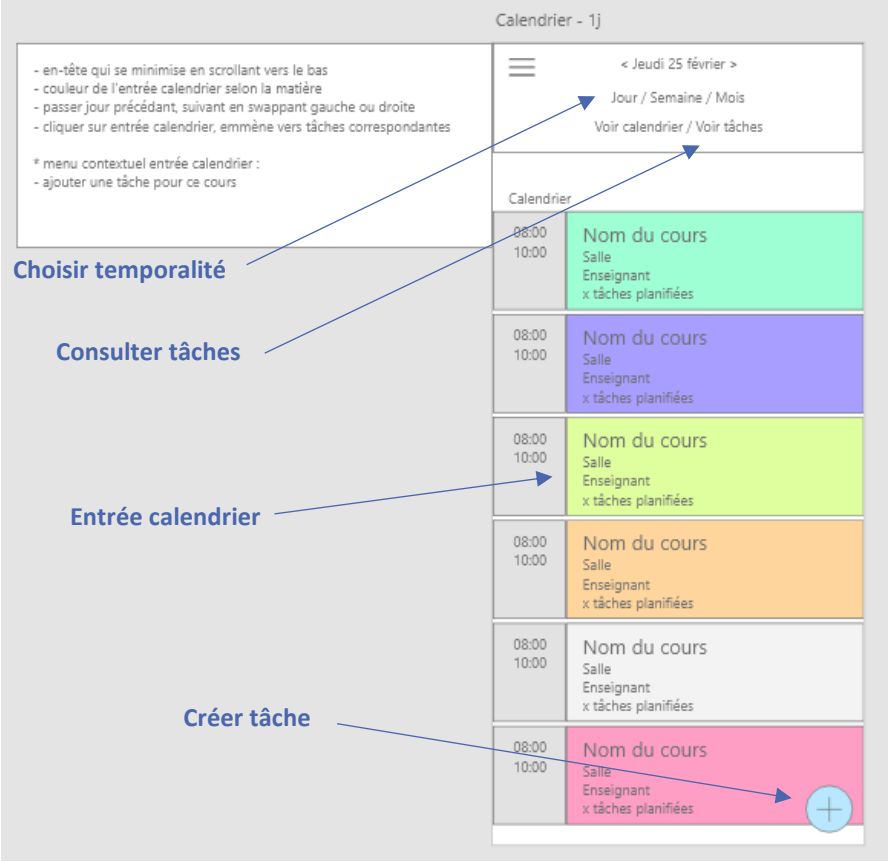


Illustration 2 : Wireframe de la page calendrier en vue journalière

LISTE DES TACHES, VUE JOURNALIERE

Il est possible de consulter les tâches à une date donnée ici. Chaque tâche peut correspondre à un cours ou non.

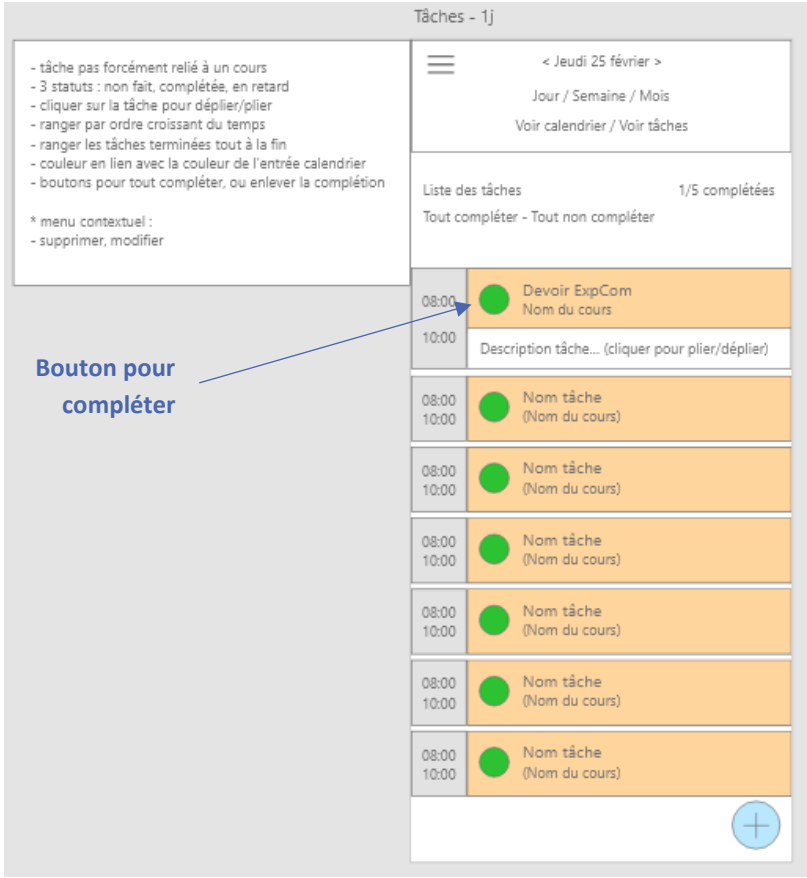


Illustration 3 : Wireframe de la page tâches en vue journalière

CALENDRIER, VUE HEBDOMADAIRE

La même page que la vue journalière, mais les entrées calendrier sont catégorisées par jour.

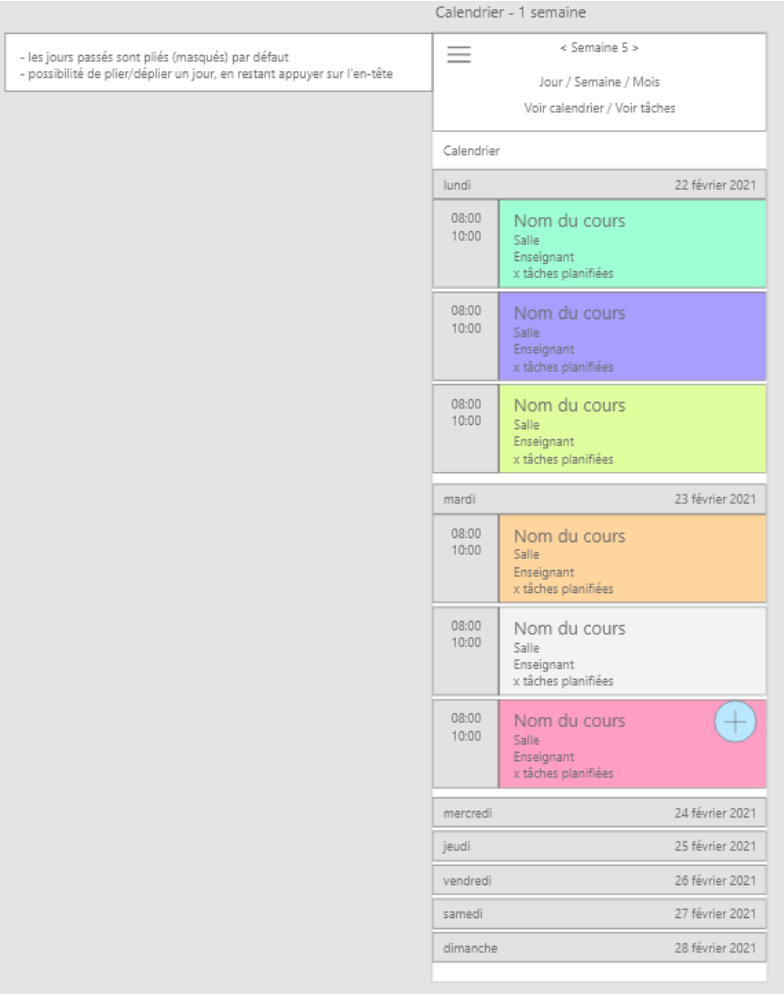


Illustration 4 : Wireframe de la page calendrier en vue hebdomadaire

LISTE DES TACHES, VUE HEBDOMADAIRE

La même page que la vue journalière, mais les tâches sont catégorisées par jour.

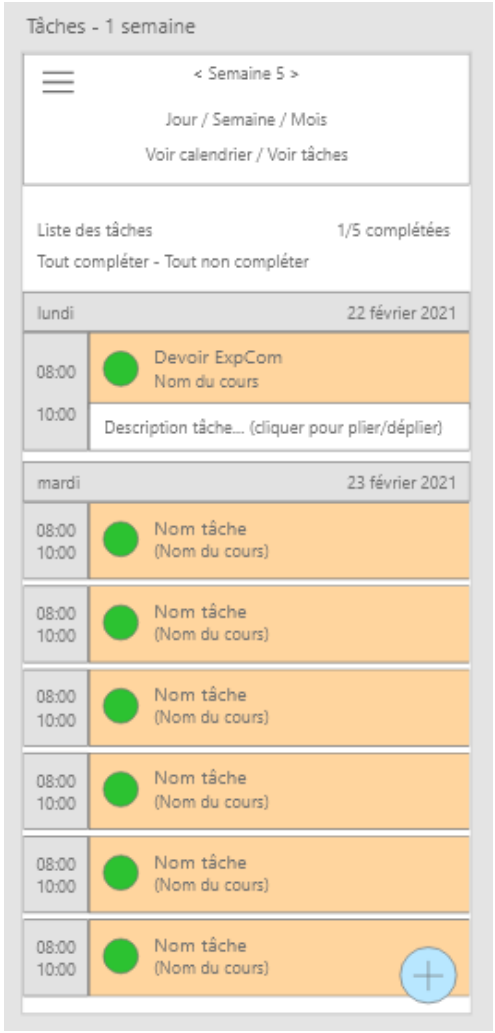


Illustration 5 : Wireframe de la page tâches en vue hebdomadaire

CREER, MODIFIER UNE TACHE

Ceci est la page pour créer ou modifier une tâche. Les champs « date de début », « titre » sont obligatoires.

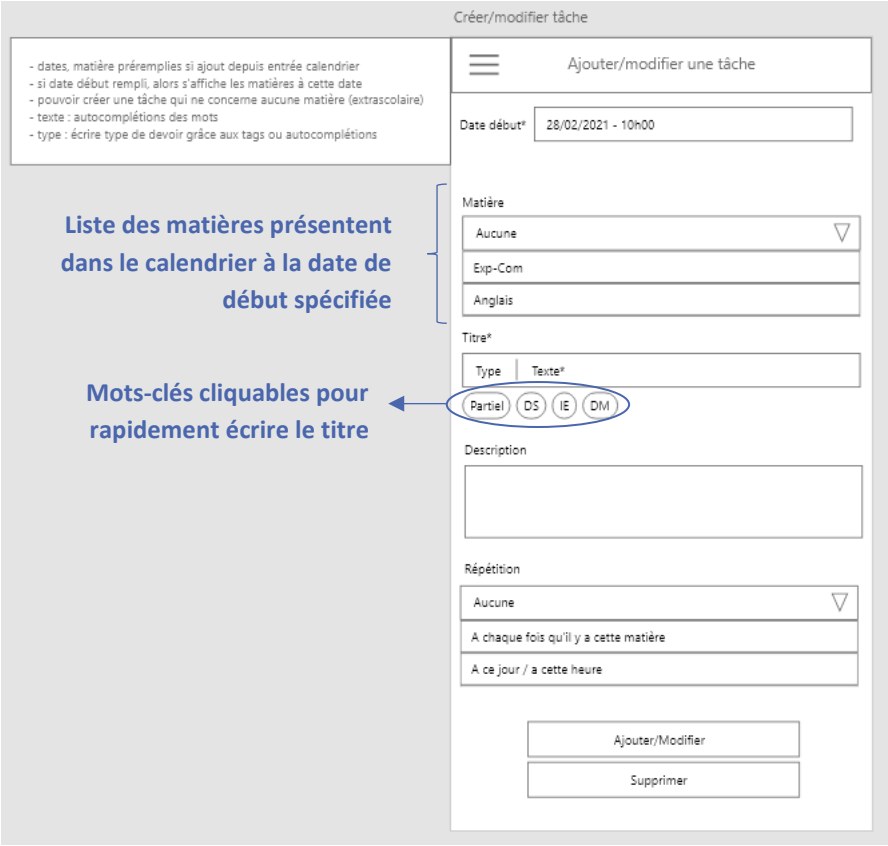


Illustration 6 : Wireframe de la page de création, édition d'une tâche

IMPORTER UN CALENDRIER EXTERNE

Cette page permet d'importer un calendrier externe à partir d'un fichier ICS[2] automatiquement scanné sur le stockage de l'appareil (ex. dans la carte SD).

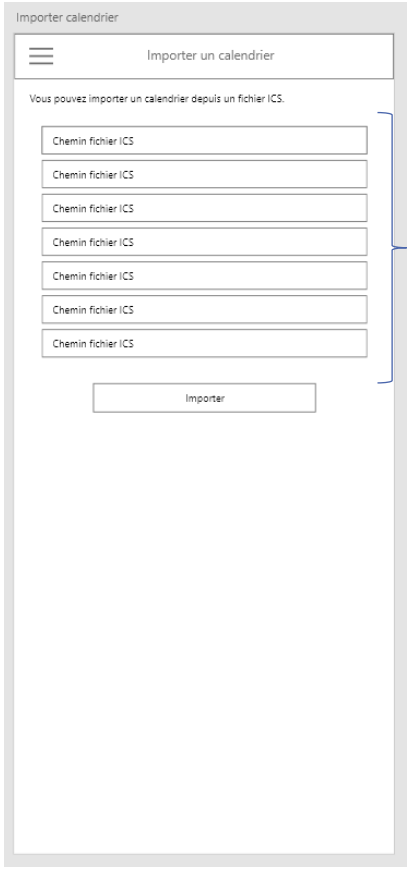


Illustration 7 : Wireframe de la page d'importation d'un calendrier

[2] Format iCalendar

Finalement, je n'ai rien appris de particulier pour cette partie, connaissant déjà les bases d'Adobe XD, c'était toutefois une étape nécessaire pour la bonne poursuite du projet.

MAQUETTE VISUELLE

Après avoir réalisé l'ergonomie et l'agencement de l'application, j'ai dû réaliser la maquette visuelle afin de pouvoir par la suite programmer rapidement et sans encombre l'interface utilisateur.

La maquette visuelle a été réalisée toujours sur Adobe XD.

CHARTE GRAPHIQUE

La charte graphique est très simpliste, je n'ai pas voulu m'embêter de faire un style très poussé, le projet étant porté sur l'obtention de nouvelles connaissances en informatique.

Le jeu de couleurs utilisé est dans les tons bleus :

Illustration 8 : Jeu de couleurs utilisé

Concernant les icônes, celles-ci proviennent de la bibliothèque d'icônes Font Awesome disponible gratuitement.

Enfin, la police utilisée est celle du système d'exploitation où se trouve l'application.

CONCEPTION DES COMPOSANTS

En premier, j'ai conçu les différents composants qui composeront les pages de l'application, par exemple la barre de navigation, ou l'entrée calendrier, etc.

Les composants peuvent avoir différents états (ex. déplié, plié, coché, pas coché, etc.) que vous verrez juste après.

Pour faire cela de façon efficace, c'est-à-dire, sans copier-coller plusieurs fois le même composant juste pour changer un petit détail, Adobe XD propose justement un système pratique pour gérer ce genre de cas que j'ai appris à cette occasion.

Globalement, il suffit simplement de créer un groupe d'éléments, et de créer plusieurs états, et pour chacun d'entre eux, faire la modification souhaitée.

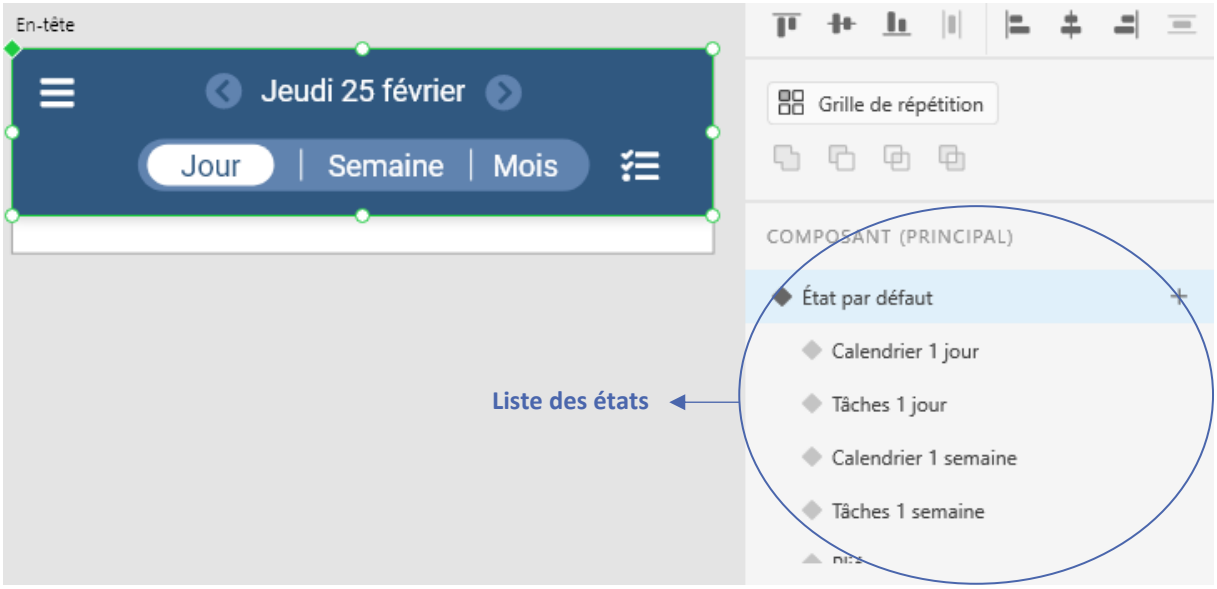


Illustration 9 : Design barre de navigation sur l'état par défaut

Ainsi, il suffit juste de copier ce composant dans plusieurs pages, et il suffit de changer l'état.

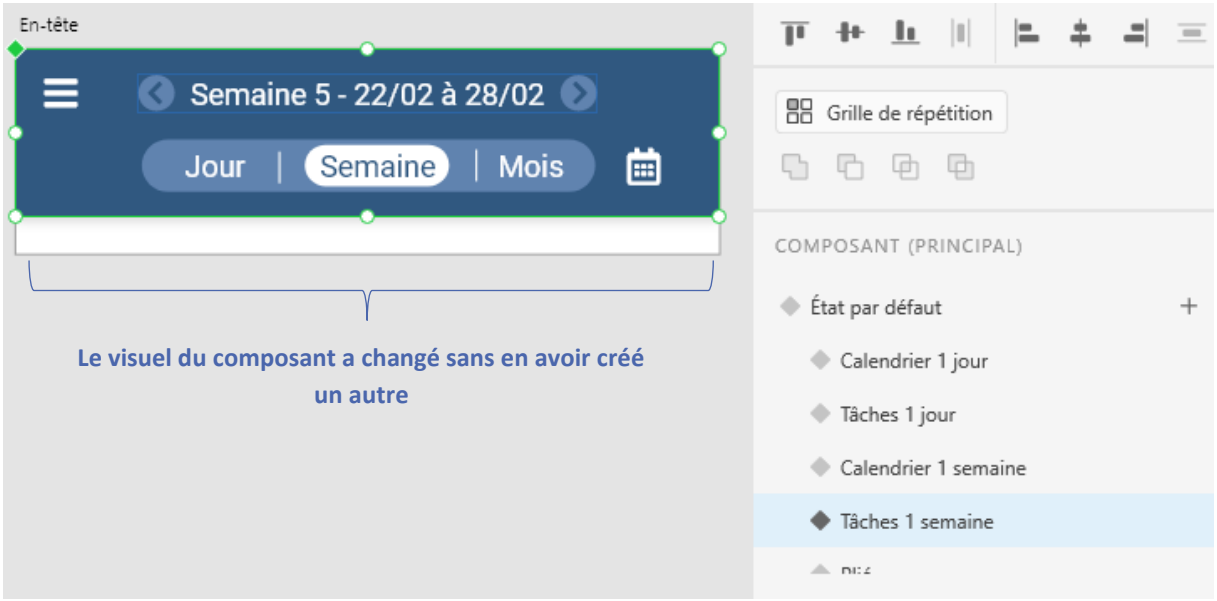


Illustration 10 : Design barre de navigation sur l'état liste de tâches vue journalière

CONCEPTION DES PAGES

Dans les prochaines pages, vous trouverez l'ensemble la maquette visuelle de chaque page.

Illustration 11 : Design page calendrier, vue journalière

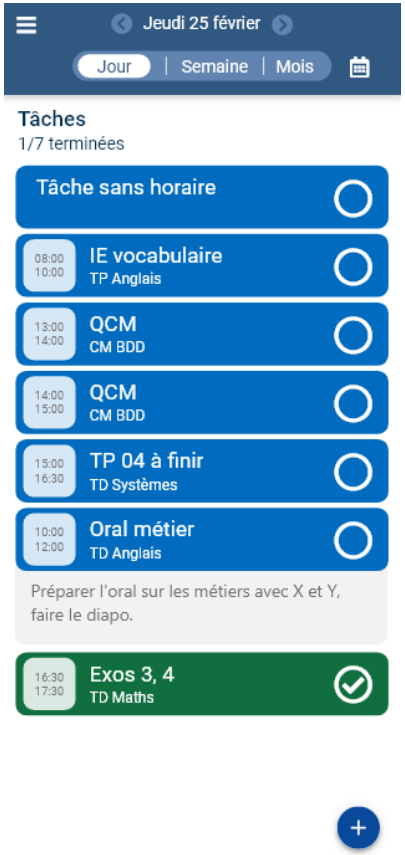


Illustration 12 : Design page tâches, vue journalière

Illustration 13 : Design page calendrier, vue hebdomadaire

Illustration 14 : Design page tâches, vue hebdomadaire

Bien que ce ne soit pas en rapport avec l'informatique, je suis quand même satisfait de m'avoir perfectionné dans l'utilisation d'Adobe XD afin d'être plus efficace et rapide, que j'utilise très régulièrement dans les projets.

PROGRAMMATION DU VISUEL DES PAGES

Maintenant que la partie conception est terminée, il ne reste plus qu'à programmer l'interface utilisateur en C# avec Xamarin. Dans cette section, je vais passer en revue tous les principes que j'ai appris tout en montrant des aperçus de l'application.

XAML[3]

Le framework Xamarin (Xamarin.Forms plus précisément), permet la programmation d'interface utilisateur par le biais du langage XAML. XAML est un langage à balise, extension du langage XML. Pour mieux illustrer, cela permet de créer des pages à la manière d'HTML.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:sys="clr-namespace:System;assembly=mscorlib"
             x:Class="PTS4.Views.EditTaskPage"
             Title="Créer/modifier une tâche">
    <ContentPage.Content>
        <ScrollView>
            <StackLayout Style="{StaticResource Page}">
                <Label Text="Date*" />
                <DatePicker x:Name="DatePicker" MinimumDate="{x:Static sys:DateTime.Now}" Date="{x:Binding StartTime, Mode=TwoWay}" DateSelected="DatePickerDateSelected" />

                <Label Text="Matière" />
                <Picker x:Name="CoursePicker" SelectedIndexChanged="CoursePickerSelectedIndexChanged">
                </Picker>

                <Label Text="Titre*"/>
                <Entry x:Name="TitleEntry" Text="{x:Binding Title, Mode=TwoWay}" Placeholder="DM Anglais" />

                <!-- Mots-clés pour le titre de la tâche -->
                <FlexLayout x:Name="TaskTagList" Direction="Row" JustifyContent="Center"  Wrap="Wrap">
                </FlexLayout>

                <Label Text="Description" Margin="0, 20, 0, 0" />
                <Editor x:Name="DescriptionEntry" AutoSize="TextChanges" Placeholder="Résumer le texte vu en cours" Text="{x:Binding Description, Mode=TwoWay}" />

                <Label Text="Répétition" />
                <Picker x:Name="RepeatPicker" SelectedIndexChanged="RepeatPickerSelectedIndexChanged">
                </Picker>

                <Button x:Name="ValidateButton" Text="Créer/Modifier" BackgroundColor="Green" TextColor="White" Margin="0, 20, 0, 0" Clicked="ValidateButtonClicked" />
                <Button x:Name="DeleteButton" Text="Supprimer" BackgroundColor="DarkRed" TextColor="White" Clicked="DeleteButtonClicked" />
            </StackLayout>
        </ScrollView>
    </ContentPage.Content>
</ContentPage>
```

Illustration 15 : Page d'édition d'une tâche codée avec XAML

Xamarin fournit des vues[4] de base, comme on peut le voir ici, par exemple :

- « Label » pour afficher du texte
- « Picker » pour une liste déroulante
- « Entry » pour un champ de saisie
- « Editor » pour un champ de saisie multilignes
- « DatePicker » pour choisir une date

Comme pour HTML, chaque type de vue contient des attributs que l'on modifier, tels que :

- « Name » pour définir le nom de la variable pour manipuler la vue dans le code C#
- « Margin » pour définir les marges de la vue
- « BackgroundColor » pour définir la couleur de fond

[3] eXtensible Application Markup Language
[4] Terme Xamarin désignant des éléments (ex. bouton, champ de texte, etc.)

DISPOSITIONS D'AFFICHAGE

Pour définir la mise en page, c'est-à-dire, comment les vues vont se disposer sur la page, Xamarin propose plusieurs dispositions, aussi appelées « conteneurs » :

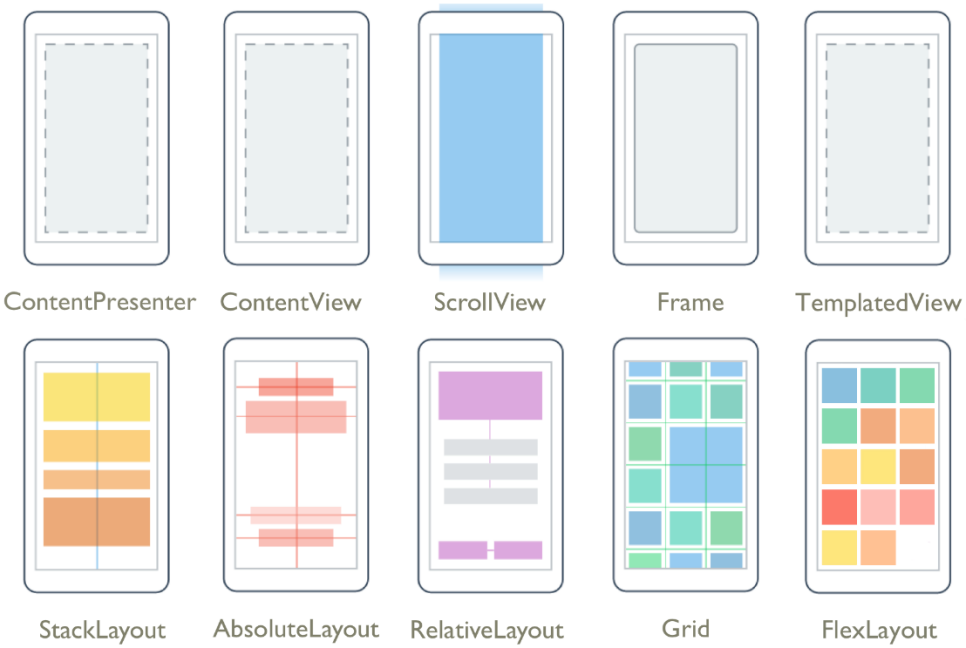


Illustration 16 : Dispositions d'affichage proposées par Xamarin

Ces dispositions peuvent contenir des éléments enfants comme des boutons qui seront organisés spécifiquement selon le type de disposition.

Prenons, par exemple, la page de création d'une tâche qui contient divers champs de texte, labels, et boutons. J'ai utilisé une disposition « StackLayout » qui permet que des éléments s'empilent les uns sous les autres :

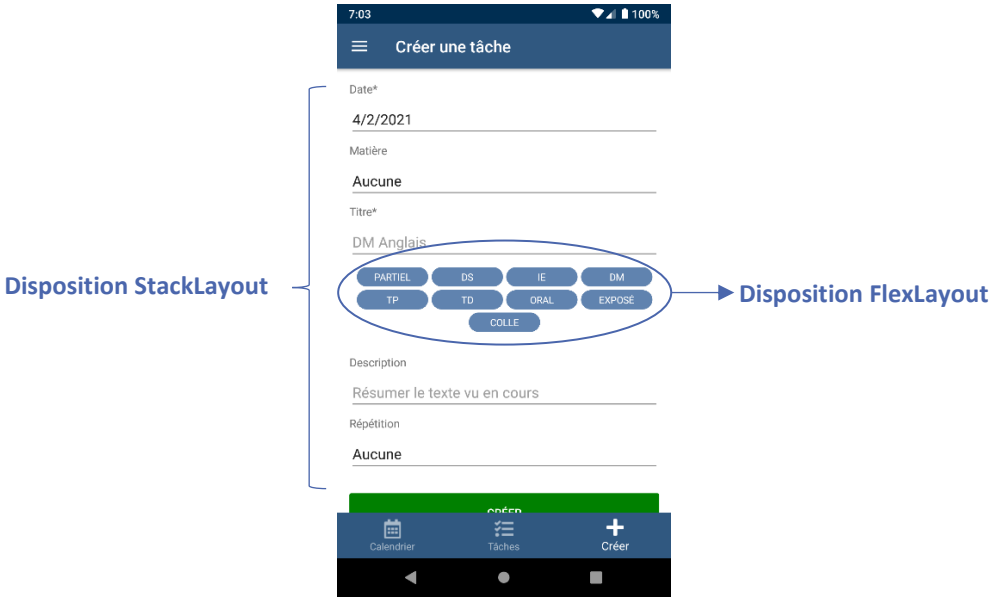


Illustration 17 : Page de création d'une tâche avec une disposition de pile

Regardez les boutons mots-clés, ceux-ci sont contenus dans une disposition flex qui permet d'agencer les éléments automatiquement sur toute la longueur de la page.

GESTIONNAIRE DE STYLES

Disposer les éléments sur la page est une chose, maintenant, il reste à modifier le visuel des éléments pour correspondre à la maquette visuelle.

Pour cela, Xamarin propose la possibilité de tout personnaliser avec XAML avec ce qu'il appelle le « gestionnaire de styles ».

NIVEAUX DE STYLES

Il y a plusieurs niveaux de styles :

- Les styles déclarés de **manière globale**, s'appliquent dans toutes les pages de l'application.
  Ceux-ci se définissent dans le fichier « App.xaml ».

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="PTS4.App">
    <!--
        Define global resources and styles here, that apply to all pages in your app.
    -->
    <Application.Resources>
        <ResourceDictionary>
            <!-- Styles généraux -->
            <Color x:Key="Primary">#305880</Color>
            <Color x:Key="Secondary">#6083AF</Color>
            <Color x:Key="Tertiary">#002E52</Color>

            <Style TargetType="ImageButton">
                <Setter Property="BackgroundColor" Value="#00000000" />
            </Style>

            <Style x:Key="Page" TargetType="StackLayout">
                <Setter Property="Margin" Value="15" />
            </Style>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

Constantes qui sont utilisables dans toutes les pages

Arrière-plan transparent pour tous les éléments « ImageButton »

Définir un style nommé « Page » qui s'applique sur les dispositions « StackLayout »

Illustration 18 : Styles globaux

Par exemple, ici, j'ai défini pour l'ensemble de l'application des constantes définissant les couleurs principales, cela évite d'écrire le code couleur à chaque endroit où je dois les utiliser.

De même, que j'ai défini des styles s'appliquant sur un certain type d'élément.

- Les styles déclarés au **niveau de la page/vue**, s'appliquent à toutes les vues de la page/vue.
  La portée est donc limitée à cette unique page/vue.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ContentView xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:entry="clr-namespace:PTS4.Views.Components.EntryParts"
             x:Class="PTS4.Views.Components.CalendarEntry">
    <ContentView.Resources>
        <Style TargetType="Label">
            <Setter Property="TextColor" Value="White" />
        </Style>
    </ContentView.Resources>

    <ContentView.Content>
        <Frame x:Name="CalendarEntryFrame" BackgroundColor="#006BBF" CornerRadius="10" HeightRequest="60" Padding="6">
            <Grid ColumnDefinitions="Auto, *, Auto">
                <Grid.GestureRecognizers>
                    <TapGestureRecognizer Tapped="OnTapped" />
                </Grid.GestureRecognizers>

                <entry:EntryScheduleBox Grid.Column="0" />
                <entry:EntryInfoBox Grid.Column="1" />
                <Label x:Name="TaskNumberLabel" Text="" Grid.Column="2" VerticalTextAlignment="End" Padding="6" FontSize="12" />
            </Grid>
        </Frame>
    </ContentView.Content>
</ContentView>
```

Tous les labels contenus dans cette vue, ont une couleur de texte blanche

Illustration 19 : Styles au niveau de la vue entrée calendrier

Note : il est possible de créer ses propres vues, nous le verrons après dans la partie Création des composants.

GESTION DES ETATS GRACE AUX STYLES

Pour retranscrire les états créés dans les maquettes visuelles dans Xamarin, le gestionnaire de styles, permet de définir un style particulier pour un état donné d'une vue.

Prenons, un exemple pour ce soit plus parlant :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ContentView xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:entry="clr-namespace:PTS4.Views.Components.EntryParts"
             x:Class="PTS4.Views.Components.TaskEntry">
    <ContentView.Resources>
        <ResourceDictionary>
            <Style x:Key="TaskEntry" TargetType="Frame">
                <Setter Property="CornerRadius" Value="10" />
                <Setter Property="HeightRequest" Value="60" />
                <Setter Property="Padding" Value="6" />

                <Setter Property="VisualStateManager.VisualStateGroups">
                    <VisualStateGroupList>
                        <VisualStateGroup>
                            <VisualState Name="Normal">
                                <VisualState.Setters>
                                    <Setter Property="BackgroundColor" Value="#006BBF" />
                                </VisualState.Setters>
                            </VisualState>

                            <VisualState Name="Completed">
                                <VisualState.StateTriggers>
                                    <StateTrigger IsActive="{Binding Completed}" />
                                </VisualState.StateTriggers>

                                <VisualState.Setters>
                                    <Setter Property="BackgroundColor" Value="#106E3F" />
                                    <Setter TargetName="CompleteButton" Property="ImageButton.Source" Value="circle_filled.png" />
                                </VisualState.Setters>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateGroupList>
                </Setter>
            </Style>
```

Définition de la couleur de fond sur bleu, lorsque l'état « Normal » est actif

Définition de la couleur de fond sur vert, lorsque l'état « Completed » est actif

Illustration 20 : États de la vue d'une tâche

Illustration 22 : Composant entrée tâche, état normal

Illustration 21 : Composant entrée tâche, état complété

Ici, pour les états « Normal » et « Completed », la couleur de fond change, ainsi que l'icône utilisée pour la case à cocher.

GESTIONNAIRE D'EVENEMENTS

Lorsque l'utilisateur appuie sur un bouton, il faut déclencher une action spécifique. C'est le rôle du gestionnaire d'événements de Xamarin (basé sur celui du framework .NET).

Concrètement, il faut associer à la vue dont on veut gérer un événement externe (appuie, saisie de texte, sélection, etc.), une méthode de rappel (callback) a appelé.

```
    <ContentView.Content>
        <Frame x:Name="TaskEntryFrame" Style="{StaticResource TaskEntry}">
            <Grid ColumnDefinitions="Auto, *, Auto">
                <Grid.GestureRecognizers>
                    <TapGestureRecognizer Tapped="OnTapped" />
                </Grid.GestureRecognizers>

                <entry:EntryScheduleBox Grid.Column="0" />
                <entry:EntryInfoBox Grid.Column="1" />
                <ImageButton x:Name="CompleteButton" Source="circle_unfilled.png" Grid.Column="2" Clicked="OnCompleteButtonClicked" />
            </Grid>
        </Frame>
    </ContentView.Content>
</ContentView>
```

Illustration 23 : Association d'un callback à l'évènement « Clicked » du bouton de complétion d'une tâche

Lorsque le bouton de complétion de la tâche est appuyé, cela déclenche la méthode « OnCompleteButtonClicked » qui est déclaré dans la classe correspondante à la vue :

```
private void OnCompleteButtonClicked(object sender, EventArgs e)
{
    if (this.viewModel.Completed)
    {
        VisualStateManager.GoToState(this.TaskEntryFrame, "Normal");
        MessagingCenter.Send<TaskEntry>(this, "TaskUncomplete");
        MessagingCenter.Send<TaskEntry, String>(this, "TaskUncomplete", this.viewModel.Course);
    }
    else
    {
        VisualStateManager.GoToState(this.TaskEntryFrame, "Completed");
        MessagingCenter.Send<TaskEntry>(this, "TaskComplete");
        MessagingCenter.Send<TaskEntry, String>(this, "TaskComplete", this.viewModel.Course);
    }

    this.viewModel.Completed = !this.viewModel.Completed;
}
```

Illustration 24 : Méthode gérant l'évènement « Clicked » du bouton de complétion d'une tâche

Cette méthode change l'état de la vue de « Normal » à « Completed » notamment.

Note : à chaque fichier XAML, est associé un autre fichier C# contenant la classe correspondante.

GESTIONNAIRE DE NAVIGATION

Comme une application avec une unique page n'est en général pas très intéressante, elle en possède donc plusieurs, ce qui implique de gérer la navigation entre celles-ci.

Xamarin offre à la disposition du développeur un nouveau système de navigation basé sur les URI[5] nommé « Shell Navigation ». Il a pour objectif de mimer la navigation sur les sites Internet, avec les URL.

ROUTES

Par exemple, je peux définir une route (chemin d'accès à la page) pour chaque page :

- « /MainPage » pour la page principale
- « /ImportCalendarPage » pour la page d'importation d'un calendrier
- « /CalendarMainPage » pour la page de calendrier
- « /TaskMainPage » pour la page liste des tâches
- « /EditTaskPage » pour la page d'édition d'une tâche

Ainsi, si je souhaite naviguer vers la page d'édition d'une tâche à partir de la page principale, il suffit d'appeler une méthode en spécifiant la route « ImportCalendarPage » :

```
Shell.Current.GoToAsync("ImportCalendarPage");
```

Illustration 25 : Ligne de code pour naviguer vers une page en spécifiant sa route

QUERY PARAMETERS

Parfois, il est nécessaire de passer des informations à la page dont on veut naviguer vers, par exemple, l'identifiant d'une tâche que l'on veut éditer.

Pour cela, on peut spécifier dans l'URI destination, des « query parameters » :

```
Shell.Current.GoToAsync($"//EditTaskPage?task={this.viewModel.Guid}");
```

Illustration 26 : Navigation vers la page d'édition d'une tâche en passant son identifiant en paramètre d'URI

Ici, on navigue vers la page d'édition d'une tâche, et en plus, on précise un paramètre nommé « task » ayant pour valeur l'identifiant de la tâche que l'on souhaite éditer.

[5] Uniform Resource Identifier

```
[QueryProperty(nameof(TaskGuid), "task")]
[QueryProperty(nameof(EventGuid), "event")]
6 références
public partial class EditTaskPage : ContentPage
{
    private EditTaskViewModel viewModel;

    /// <summary>
    /// Indiquer si la vue est en train de charger les données d'une tâche existante
    /// </summary>
    private bool isLoading;

    /// <summary>
    /// GUID de la tâche en cours d'édition<br/>
    /// null si création d'une nouvelle tâche
    /// </summary>
    5 références
    public String TaskGuid { get; set; }

    /// <summary>
    /// GUID de l'entrée calendrier source<br/>
    /// null si création d'une nouvelle tâche à partir de rien
    /// </summary>
    4 références
    public String EventGuid { get; set; }
```

On indique que les propriétés « TaskGuid » et « EventGuid » de cette classe reçoivent les query parameters « task » et « event »

Illustration 27 : Classe de la page d'édition d'une tâche montrant les propriétés recevant les « query parameters »

Le query parameter « task » transmis juste avant, est stocké dans la propriété « TaskGuid », ainsi nous pouvons l'utiliser pour récupérer les données de la tâche, et les afficher dans les champs de la page.

CREATION DE COMPOSANTS

Avec Xamarin, il est possible de créer des pages. Il est aussi possible de créer ses propres vues en plus de celles par défaut. Cela a plusieurs avantages, cela permet de réutiliser un ensemble d'éléments dans plusieurs pages très facilement, donc de factoriser le code. Cela permet aussi de réduire considérablement la taille du code, afin d'éviter d'avoir 500 lignes du même fichier. Par ailleurs, cela permet de respecter la responsabilité unique du principe SOLID en programmation objet.

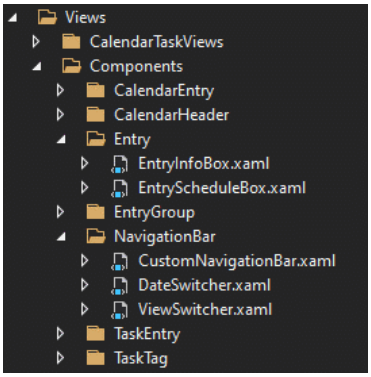


Illustration 28 : Les différents composants constituants les pages

Comme vous pouvez l'apercevoir, j'ai divisé mon interface utilisateur en plusieurs morceaux :

Illustration 29 : Composant « NavigationBar »

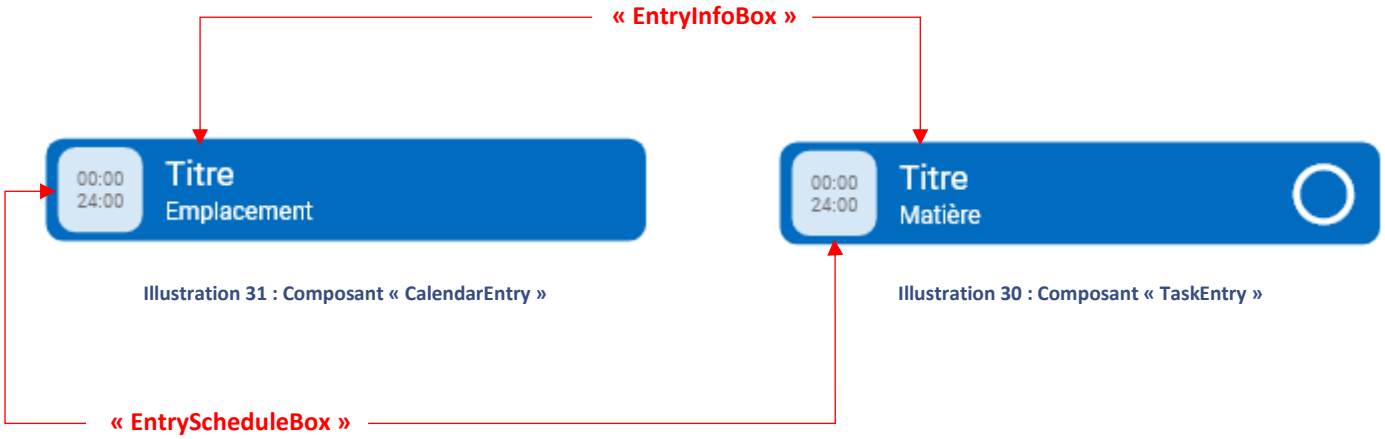


Illustration 31 : Composant « CalendarEntry »

Illustration 30 : Composant « TaskEntry »

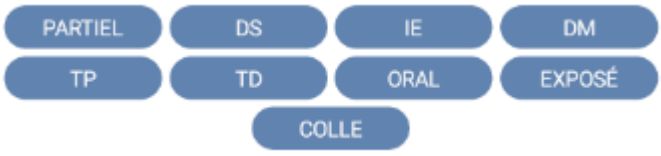


Illustration 33 : Composant « TaskTag »

Illustration 32 : Composant « CalendarHeader »

Ensuite, il est possible d'insérer ces vues personnalisées dans des pages ou d'autres vues :

```
    <ContentView.Content>
        <Frame x:Name="CalendarEntryFrame" BackgroundColor="#006BBF" CornerRadius="10" HeightRequest="60" Padding="6">
            <Grid ColumnDefinitions="Auto, *, Auto">
                <Grid.GestureRecognizers>
                    <TapGestureRecognizer Tapped="OnTapped" />
                </Grid.GestureRecognizers>

                <entry:EntryScheduleBox Grid.Column="0" />
                <entry:EntryInfoBox Grid.Column="1" />
                <Label x:Name="TaskNumberLabel" Text="" Grid.Column="2" VerticalTextAlignment="End" Padding="6" FontSize="12" />
            </Grid>
        </Frame>
    </ContentView.Content>
</ContentView>
```

Illustration 34 : Insertion des vues « EntryScheduleBox » et « EntryInfoBox » dans la vue « CalendarEntry »

Notez, qu'il est possible de créer ses propres propriétés dans les classes correspondantes aux vues, pour ajouter des attributs, à l'instar des attributs « Text » du label… Toutefois, je n'ai très peu utilisé cette fonctionnalité.

PARTICULARITES ANDROID, IOS

Je me suis heurté à un problème dû à la différence de fonctionnement entre Android et iOS pour gérer certaines apparences de vues. Je peux penser notamment au sélecteur de date :

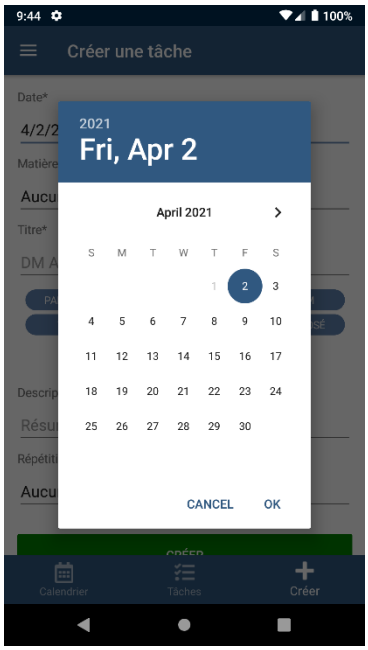


Illustration 36 : Sélecteur de date sur Android

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
  <style name="MainTheme" parent="MainTheme.Base">
    <!-- As of Xamarin.Forms 4.6 the theme has moved into the Forms binary -->
    <!-- If you want to override anything you can do that here. -->
    <!-- Underneath are a couple of entries to get you started. -->

    <!-- Set theme colors from https://aka.ms/material-colors -->
    <!-- colorPrimary is used for the default action bar background -->
    <item name="colorPrimary">#305880</item>
    <!-- colorPrimaryDark is used for the status bar -->
    <item name="colorPrimaryDark">#002E52</item>
    <!-- colorAccent is used as the default value for colorControlActivated
         which is used to tint widgets -->
    <item name="colorAccent">#6083AF</item>

    <!-- Couleurs ListView -->
    <item name="android:colorPressedHighlight">@color/listViewHighlighted</item>
    <item name="android:colorLongPressedHighlight">@color/listViewHighlighted</item>
    <item name="android:colorFocusedHighlight">@color/listViewSelected</item>
    <item name="android:colorActivatedHighlight">@color/listViewSelected</item>
    <item name="android:activatedBackgroundIndicator">@color/listViewSelected</item>

    <!-- Couleurs DatePicker -->
    <item name="android:datePickerDialogTheme">@style/AppCompatDialogStyle</item>
  </style>

  <color name="listViewSelected">#305880</color>
  <color name="listViewHighlighted">#305880</color>

  <style name="AppCompatDialogStyle" parent="Theme.AppCompat.Light.Dialog">
    <item name="colorAccent">#305880</item>
  </style>
</resources>
```

Illustration 35 : Fichier de styles spécifique à Android

Il n'était pas possible de modifier l'apparence avec Xamarin seul, il a plutôt fallu que je modifie le style présent dans le fichier de styles spécifique aux applications Android, de même pour le fichier spécifique à iOS.

CONCLUSION

Nous voilà enfin terminés pour cette phase de développement de l'interface utilisateur. Au départ, je ne connaissais pas du tout Xamarin, et je ne le cache pas que les débuts ont été assez compliqués, au point de me demander si Xamarin était nul. Finalement, grâce à la documentation Microsoft très bien expliquée concernant Xamarin.Forms : https://docs.microsoft.com/fr-fr/xamarin/xamarin-forms/, j'ai pu m'en sortir en apprenant tous les concepts inhérents au bon développement d'une interface utilisateur sous ce framework.

J'ai appris :

- Langage XAML
- Création des styles, leurs différents niveaux, et la gestion des états
- Gestion des évènements
- Gestion de la navigation, avec passage de données
- Création de nouvelles pages, vues
- Réagir face à certaines particularités Android et iOS

DEVELOPPEMENT DES FONCTIONNALITES

Maintenant que nous avons passé le gros morceau qu'est l'interface utilisateur, attaquons-nous au développement des fonctionnalités qui n'est plus du domaine de Xamarin, mais du framework .NET[6].

Celui-ci, je le maîtrisais déjà un peu avant d'avoir débuter le projet avec les cours de Programmation objet à l'IUT.

FORMAT ICALENDAR

L'application doit pouvoir offrir la fonctionnalité d'importer un calendrier externe (ex. provenant de Zimbra). Après quelques recherches, j'ai pu identifier le format standard qui est iCalendar.

iCalendar est un format de données inscrit dans le standard RFC 5545. Il est le format le plus utilisé dans les applications calendrier (ex. Zimbra, iCal, Windows Calendar, etc.), d'où le fait que beaucoup d'applications offrent la possibilité d'exporter son calendrier.

Je vais donc profiter de ce format pour l'utiliser dans mon application.

BIBLIOTHEQUE ICAL.NET

Pour ne pas réinventer la roue, et de toute façon, je n'aurais pas eu assez de temps, je vais utiliser une bibliothèque C# pour facilement manipuler les fichiers iCalendar et en extraire les données.

J'ai donc trouvé une bibliothèque adéquate dans le gestionnaire de paquets NuGet[7], appelée « Ical.Net ».

PRESENTATION GLOBALE

La bibliothèque permet de lire un fichier iCalendar qui retourne une instance de type « Calendar ». Cette instance propose de nombreuses méthodes, mais celle qui m'intéresse, c'est celle qui permet de récupérer les événements (entrées calendrier) du calendrier importé.

Un événement est constitué de plusieurs attributs (non exhaustif) :

- Titre
- Localisation
- Date de début
- Date de fin

ADAPTATEUR

Le problème étant que les instances des événements calendrier que me retourne la bibliothèque ne correspondent pas à mes classes (qui seront décrites juste après cette partie).

Pour cela, la solution type est d'utiliser le patron de conception « Adapteur » qui permet de convertir l'interface[8] d'une bibliothèque en l'interface qui nous intéresse.

J'ai donc réalisé cet adaptateur, afin de convertir les instances de la bibliothèque.

[6] Framework Microsoft pour le développement logiciel
[7] Outil pour le framework .NET pour facilement gérer les paquets
[8] Ensemble des classes, méthodes dans ce cas

CLASSES DU MODELE

Dans cette partie, je vais décrire les classes que j'ai créées pour la couche métier de mon application. Celles-ci vont contenir les données concernant le calendrier et les tâches, ainsi que les méthodes permettant de manipuler ces données (ajout, modification, suppression notamment).

CLASSE « UTILS »

Cette classe offre des méthodes statiques de conversions de date :

- À partir d'une date, retourner la date du début de la semaine
- À partir d'une date, retourner la date de la fin de la semaine

CLASSE « COURSE »

Cette classe correspond à une matière, elle possède comme attributs :

- Titre de la matière

CLASSE « COURSES »

Cette classe stocke toutes les matières existantes dans le calendrier. Elle utilise le patron de conception « Multiton » qui permet de n'avoir qu'une unique instance de chaque « Course » afin d'éviter les doublons.

Elle permet de récupérer l'instance d'une matière en spécifiant son titre.

CLASSE « CALENDAR ENTRY »

Cette classe correspond à un évènement calendrier, elle possède comme attributs :

- Identificateur unique, pour identifier l'instance pour la passer entre les pages avec les « query parameters »
- Matière, permet de préciser la matière à laquelle se réfère l'évènement calendrier
- Localisation, en général c'est la salle
- Date de début
- Date de fin

CLASSE « CALENDAR ENTRIES »

Cette classe gère une collection d'instances « CalendarEntry » et permet d'effectuer des requêtes sur celles-ci, par exemple :

- Retourner les événements à une date spécifique
- Retourner les événements d'une semaine spécifique
- Retourner les matières d'une date spécifique (prend les matières des événements à cette date)
- Retourner un événement par son identifiant unique
- Ajouter, supprimer un événement de la collection
- Retourner tous les événements du calendrier

C'est le centre névralgique de la gestion des évènements calendrier dans l'application.

LINQ

Les requêtes sur les évènements calendrier sont effectuées grâce à l'API[9] LINQ fournit par le framework .NET. Elle permet de d'effectuer des requêtes sur des collections d'instances avec une syntaxe proche de SQL.

Par exemple, voici une requête qui retourne tous les évènements calendrier (from), de la collection (in), où la date de début et de fin sont égales à celle spécifiée « date » (where), ordonnés par date de début croissante (orderby) :

```
/// <summary>
/// Retourner tous les entrées calendrier d'un jour spécifique
/// </summary>
/// <param name="date">Date de la journée</param>
/// <returns>Entrées calendrier du jour</returns>
2 références
public IEnumerable<CalendarEntry> GetEntriesOnDay(DateTime date)
{
    IEnumerable<CalendarEntry> entries =
        from entry in this.entries
        where (entry.StartTime.Date.Equals(date.Date) && entry.EndTime.Date.Equals(date.Date))
        orderby entry.StartTime
        select entry;

    return entries;
}
```

Illustration 37 : Requête LINQ pour retourner tous les évènements calendrier à une date spécifique

C'est donc un moyen très pratique d'effectuer des requêtes facilement sans passer par des boucles.

CLASSE « TASK ENTRY »

Cette classe correspond à une tâche, elle possède comme attributs :

- Identificateur unique
- Matière, facultatif, permet de préciser la matière à laquelle se réfère la tâche
- Titre de la tâche
- Description, facultatif
- Date de début
- Date de fin
- Si complétée ou non
- Mode de répétition : sans répétition, à chaque fois que la matière apparaît, à chaque certain jour

CLASSE « TASK ENTRIES »

Cette classe gère une collection d'instances « TaskEntry » et permet d'effectuer des requêtes avec LINQ sur celles-ci, par exemple :

- Retourner les tâches à une date spécifique
- Retourner les tâches d'une semaine spécifique
- Retourner une tâche par son identifiant unique
- Ajouter, supprimer une tâche de la collection
- Retourner toutes les tâches

[9] Application Programming Interface

C’est le centre névralgique de la gestion des tâches dans l’application.

CONCLUSION

Cette phase de développement des fonctionnalités est plus courte, toutefois j’y ai quand même appris certaines choses :

- J’ai pu me rafraîchir la mémoire sur le langage C# et le framework .NET
- J’ai pu apprendre à utiliser l’API LINQ
- J’ai pu apprendre l’existante du format iCalendar

LIAISON COUCHE METIER, INTERFACE UTILISATEUR

L'application à ce stade possède une interface utilisateur complète, ainsi que les fonctionnalités, il ne reste donc plus qu'à associer les deux ensembles pour obtenir l'application finale.

C'est donc du ressort de Xamarin que de lier les deux parties.

MODELE-VUE-VUE MODELE

La convention Xamarin pour lier les données à l'interface utilisateur est le patron d'architecture « Modèle-vue-vue modèle ». Je vous en avais déjà un peu parler dans l'introduction.

CLASSES « VUE MODELE »

Dans cette partie, je vais décrire les classes « vue modèle » que j'ai créée. Elles serviront d'intermédiaires entre les classes interface utilisateurs et les classes du modèle.

Toutes les classes vue modèle effectuent un mappage (accesseurs et mutateurs) des attributs des classes modèle qui leur sont associées.

CLASSE « CALENDAR ENTRY VIEW MODEL »

Cette classe fait l'intermédiaire entre :

- Vue « Calendar Entry », composant évènement calendrier
- Modèle « Calendar Entry »

Elle effectue un simple formatage de la date en format français et sous forme « HH:mm ».

CLASSE « TASK ENTRY VIEW MODEL »

Cette classe fait l'intermédiaire entre :

- Vue « Task Entry », composant tâche
- Modèle « Task Entry »

Elle effectue un simple formatage de la date en format français et sous forme « HH:mm ».

CLASSE « NAVIGATION BAR VIEW MODEL »

Cette classe est associée à la :

- Vue « Navigation Bar », composant barre de navigation

Elle gère les actions suivantes :

- Stocker la date en cours de visionnage
- Stocker la temporalité en cours de visionnage (journalière, hebdomadaire)
- Formater la date en format français
- Passer au jour, semaine suivant(e) selon la temporalité actuelle
- Reculer au jour, semaine précédent(e) selon la temporalité actuelle

CLASSE « EDIT TASK VIEW MODEL »

Cette classe est associée à la :

- Page « Edit Task », page d'édition d'une tâche

Elle gère les actions suivantes :

- Importer les données d'une tâche à modifier, pour que la page puisse remplir ses champs
- Importer les données d'un évènement calendrier
- Vérifier que tous les champs du formulaire sont bien valides
- Créer une nouvelle tâche à partir du formulaire
- Modifier une tâche existante à partir du formulaire
- Supprimer une tâche existante
- Obtenir la liste des matières à la date sélectionnée

CLASSE « IMPORT CALENDAR VIEW MODEL »

Cette classe est associée à la :

- Page « Import Calendar », page d'importation d'un calendrier externe

Elle gère les actions suivantes :

- Scanner le stockage de l'appareil à la recherche de fichiers iCalendar
- Stocker la liste des fichiers iCalendar trouvés

CLASSE « MAIN VIEW MODEL »

Cette classe fait l'intermédiaire entre :

- La page parente à toutes les autres pages de l'application
- Modèle « Calendar Entries »
- Modèle « Task Entries »

Elle gère les actions suivantes :

- Charger un calendrier
- Réinitialiser le calendrier
- Réinitialiser les tâches
- Requêter les évènements et tâches

BINDINGS

Les différents labels, champs, etc. de mon application doivent être reliés avec les données auxquelles ils sont liés. Pour cela, Xamarin propose un mécanisme de « bindings » qui permet de synchroniser une vue à une propriété d'une instance d'une classe.

BINDING CONTEXT

Le « binding context », est l'instance du modèle auquel est reliée la page/vue. Par exemple, la vue d'une entrée tâche a pour binding context, l'instance de la tâche associée.

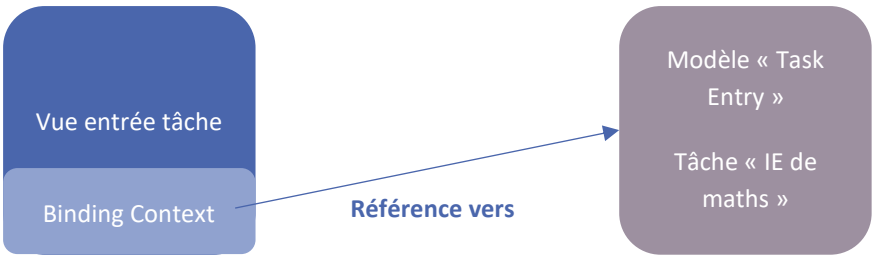


Schéma 2 : Binding Context

BINDINGS

Une fois le binding context établi, il suffit d'indiquer aux champs concernés à quelle propriété de l'instance référencée, ils doivent lire et modifier.

Lorsque la vue est affichée, ou est actualisée, celle-ci récupère la valeur en appelant la propriété qui lui est associée.

Pareillement, lorsque la valeur de la vue est modifiée par l'utilisateur (ex. un champ de saisie), cette valeur sera transmise à l'instance référencée qui se mettra à jour.

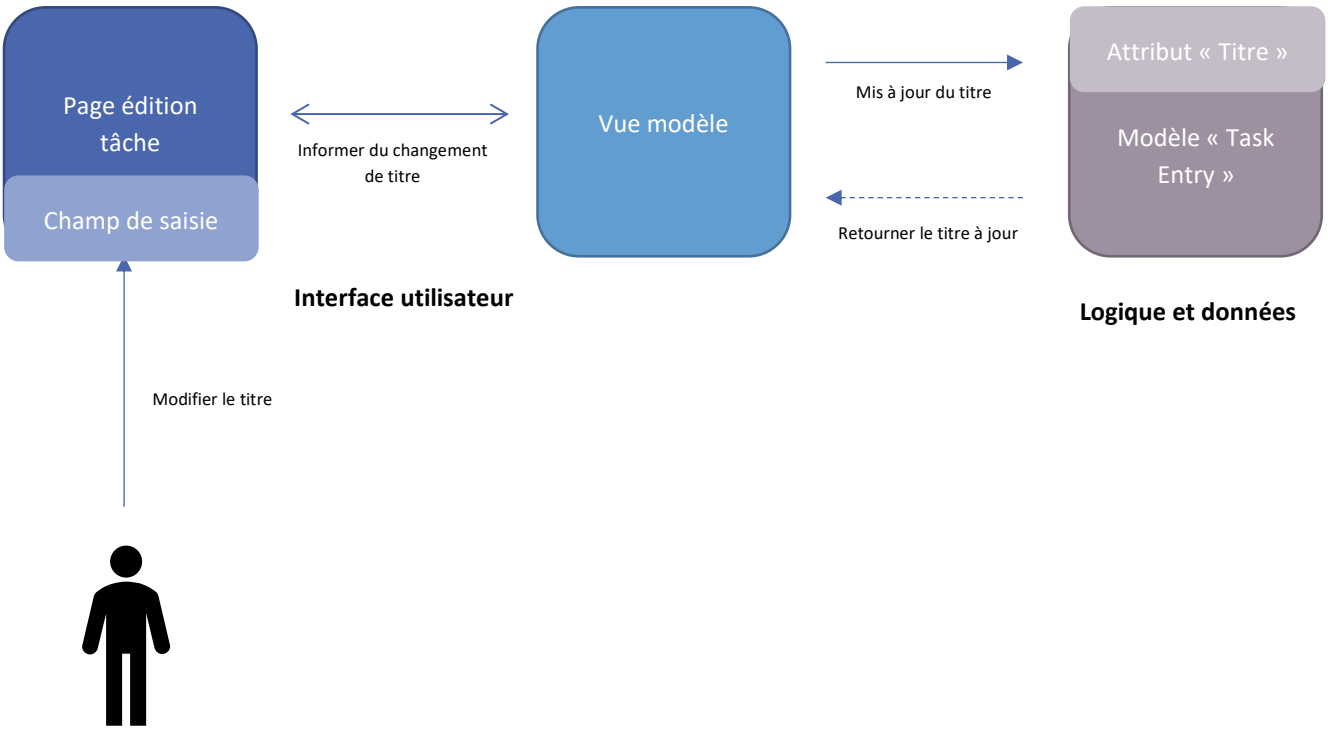


Schéma 3 : Exemple de binding entre la page d'édition d'une tâche et une instance de la tâche en cours d'édition

Dans l'idéal, il faut spécifier en tant que binding context, la vue modèle pour servir d'intermédiaire et bien isoler l'interface utilisateur et le modèle.

MESSAGING CENTER

J'ai souvent été confronté au problème que je devais informer d'autres pages et vues concernant quelque chose, par exemple, lorsque la temporalité a été changée, ou bien quand une tâche a été complétée.

Je me suis renseigné, et j'ai vu que Xamarin proposait un système à cette problématique, c'est le « Messaging Center ».

C'est un système qui permet à des pages et vues d'envoyer et de recevoir des messages.

ABONNES : CEUX QUI REÇOIVENT

Les vues peuvent s'abonner à d'autres vues, ainsi qu'à un certain type spécifique de message. En s'abonnant, il faut spécifier une action à effectuer lors de la réception du message, par exemple, appeler une méthode que l'on souhaite.

Par exemple, la page principale est abonnée à la barre de navigation pour être informée du changement de date ou de temporalité. Si elle reçoit un message, elle actualisera le calendrier ou la liste des tâches en fonction.

La page principale est abonnée aux messages :

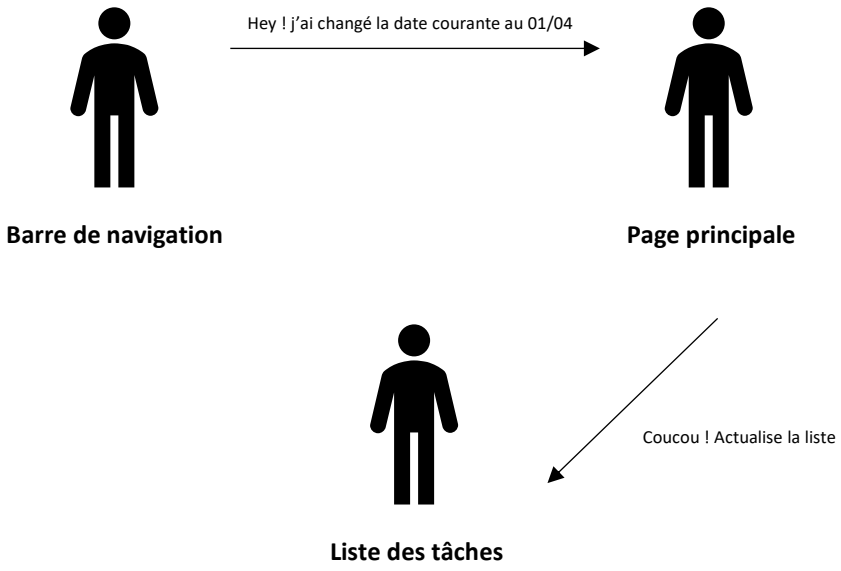
- Changement de date
- Changement de temporalité
- Réinitialisation du calendrier
- Réinitialisation des tâches
- Importer d'un calendrier

EXPEDITEURS : CEUX QUI ENVOIENT

Les vues peuvent envoyer des messages d'un certain type à d'autres vues. Il est possible de transmettre des paramètres. Par exemple, lorsqu'un calendrier est importé depuis la page d'importation, cette dernière envoie un message à la page principale avec en paramètre le chemin du fichier iCalendar à ouvrir.

RESUME

Voici un schéma récapitulatif du fonctionnement du « Messaging Center » :



CONCLUSION

C’est terminé pour la phase de liaison entre l’interface utilisateur et la couche métier. C’était une phase assez compliquée, car je me suis souvent perdu en essayant de synchroniser tout le monde, mais j’ai pu améliorer ce problème grâce aux systèmes assez pratique offerts par Xamarin.

J’ai appris :

- Architecture « Modèle-vue-vue modèle », pour une meilleure séparation des couches et un code plus maintenable
- Bindings, un moyen très pratique pour synchroniser champs et données
- Messaging Center, très pratique pour faciliter la communication inter-vues

CONCLUSION

Nous voilà quasiment à la fin de ce long rapport. J'ai essayé d'expliquer la démarche que j'ai effectuée pour réaliser ce projet, tout en essayant d'expliquer les concepts que j'ai pu apprendre à maîtriser, j'espère avoir été suffisamment clair pour le lecteur ne connaissant pas Xamarin.

APPORTS DU PROJET

Il ne va pas sans dire que ce projet m'a apporté beaucoup. Cela m'a permis de faire mes premiers pas dans le développement d'application mobile.

Bien qu'au départ, je connaissais les bases du langage C# et du framework .NET, cela a été l'occasion de les revoir et de me perfectionner encore plus dessus.

Les débuts du projet ont été particulièrement difficiles, en raison de XAML, qui est un langage à balises et que je n'apprécie en général pas ce genre de langage. Mais à force de persévérance, et grâce à l'excellente documentation de Microsoft j'ai réussi à m'en sortir et créer une application mobile fonctionnelle à 80%.

Bien que je trouvais au départ Xamarin, assez contraignant, finalement, c'est après avoir bien compris les outils qu'il propose, que j'ai changé de point de vue, et au contraire, il est finalement assez pratique, notamment avec le système de bindings, du messaging center, et du gestionnaire de styles.

Globalement, ce projet m'a permis de faire un bon tour d'horizon des principales fonctionnalités proposés par Xamarin, et si à l'avenir, je devais faire une application mobile, je ne serais pas en difficulté.

Un de mes visions en informatique, est d'en apprendre le plus possible, et c'est un plaisir d'avoir gagné une nouvelle compétence grâce à ce projet, qui en plus pourra me servir dans ma vie professionnelle.

OBJECTIFS ATTEINTS, NON ATTEINTS

Voici la liste des objectifs atteints, ainsi que ceux non atteints, faute de temps.

Objectif	Atteint
Interface utilisateur	
Importer un calendrier externe	
Consulter son emploi du temps (journalier)	
Consulter son emploi du temps (hebdomadaire)	
Consulter son emploi du temps (mois)	
Couleur unique pour chaque matière	
Consulter ses tâches	
Créer, modifier, supprimer une tâche	
Compléter une tâche	
Trier les tâches par date croissante	
Ranger les tâches terminées à la fin	
Dérouler une tâche pour consulter la description	
En vue hebdomadaire, les jours passés sont masqués par défaut	
Modes de répétition des tâches	

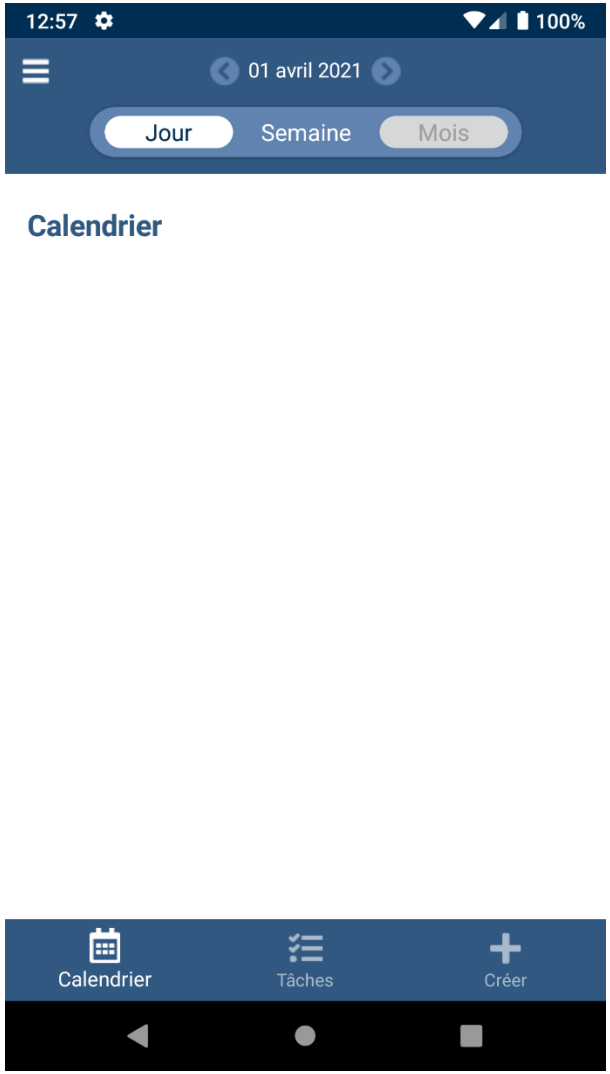
Mots-clés pour écrire le titre de la tâche	
Sauvegarder les données après fermeture de l'application	
Changer de date	
Dans la page de création d'une tâche, en choisissant une date, la liste des matières pour cette date s'affiche	
Réinitialiser le calendrier	
Réinitialiser les tâches	

BILAN PROSPECTIF

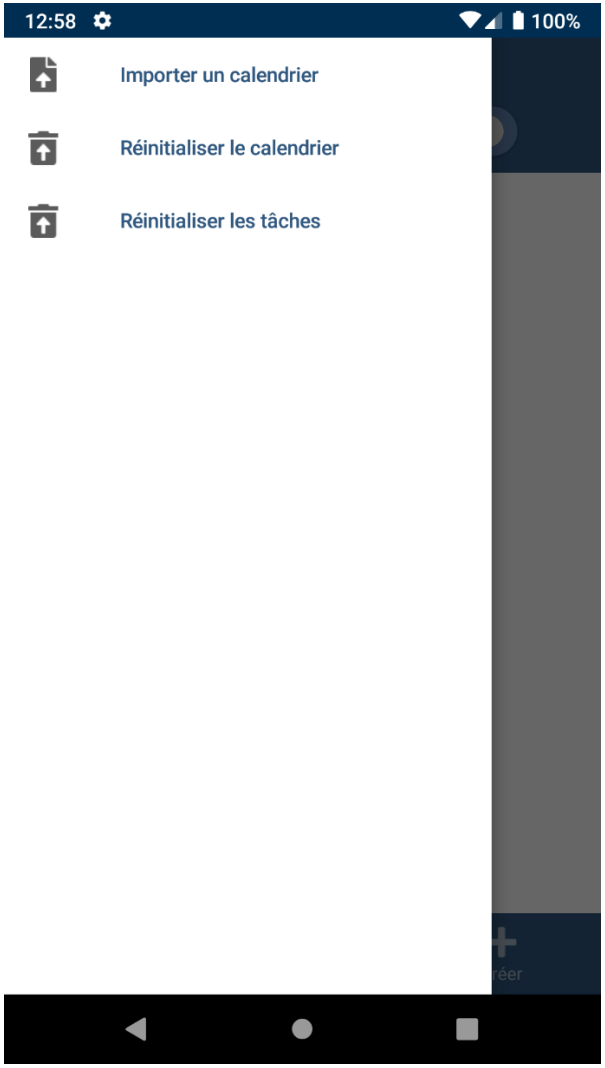
Globalement, je suis assez satisfait du résultat, il n'y aucun bug, l'interface est agréable à naviguer, et les fonctionnalités implémentées sont bien faites.

APPLICATION FINALE

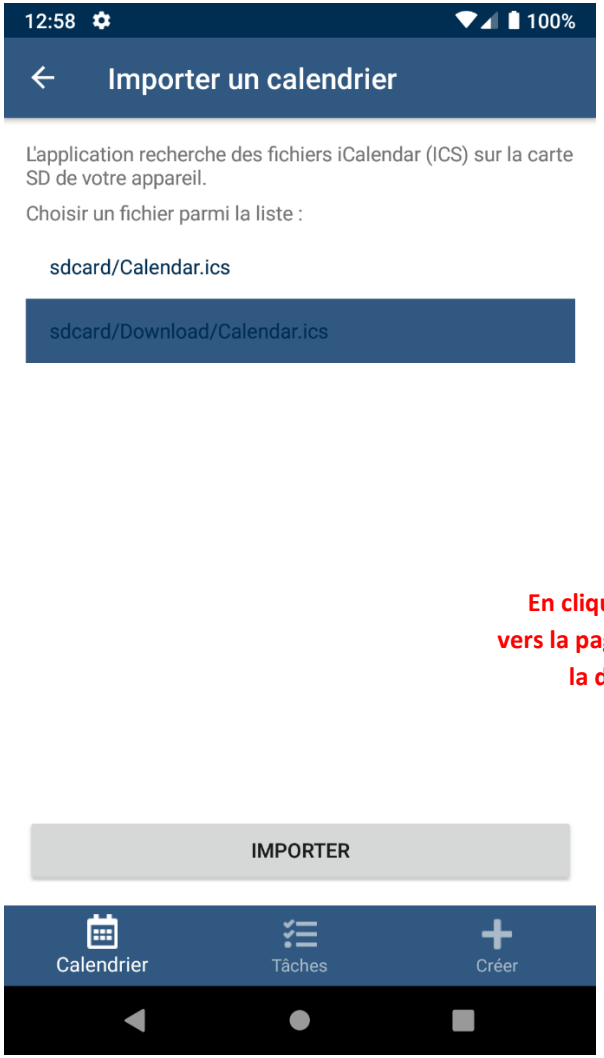
Sur les prochaines pages se trouvent des captures d'écran de l'application finale sur un appareil Android.



Capture d'écran 1 : Page d'accueil sans calendrier importé



Capture d'écran 2 : Menu latéral



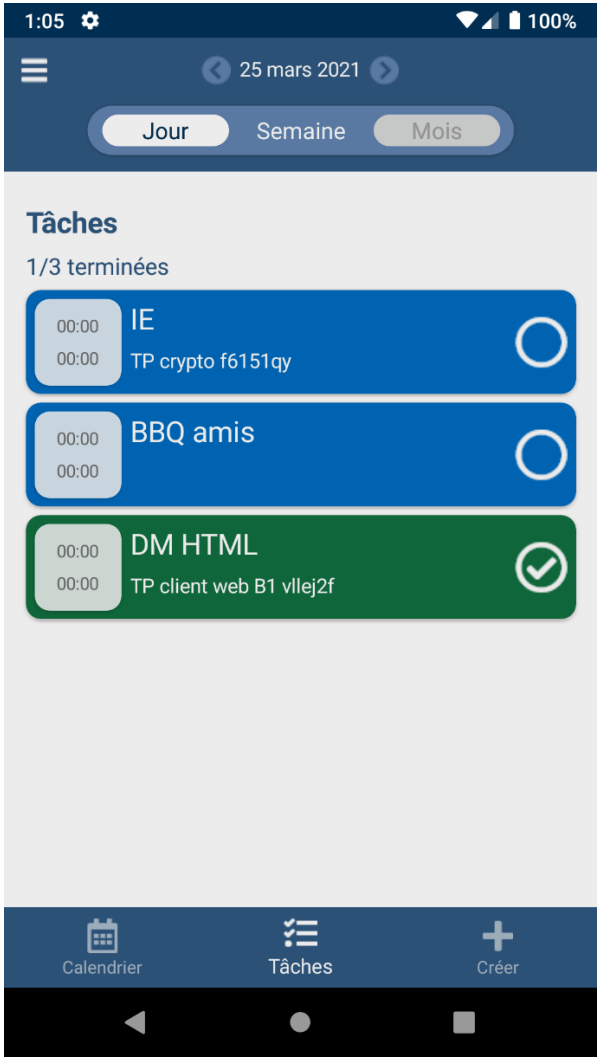
Capture d'écran 3 : Page d'importation d'un calendrier externe

En cliquant sur un cours, cela dirige vers la page de création de tâche avec la date et la matière préremplis

Capture d'écran 4 : Calendrier, vue journalière

Capture d'écran 5 : Calendrier, vue hebdomadaire

Les jours passés sont automatiquement masqués

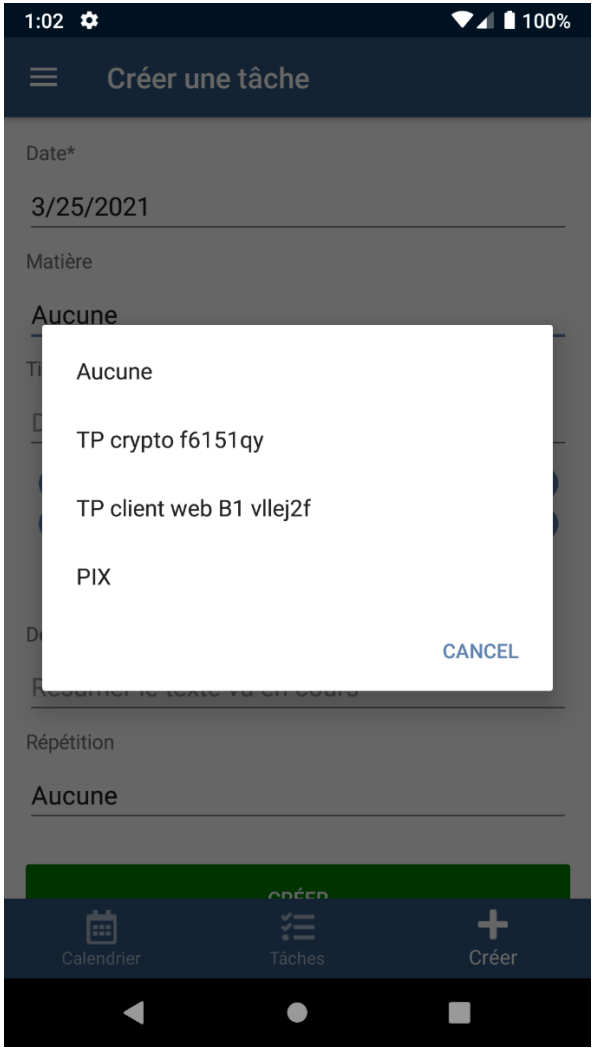


Capture d'écran 6 : Liste des tâches, vue journalière

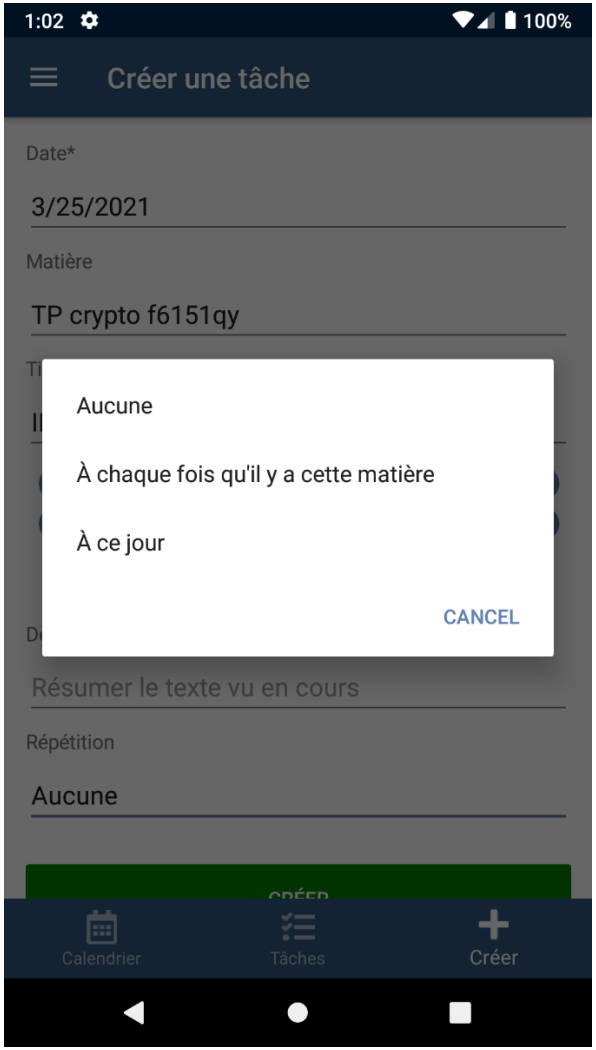
Capture d'écran 7 : Liste des tâches, vue hebdomadaire

Cliquer sur un mot-clé, l'écrit automatiquement dans le champ titre

Capture d'écran 8 : Page de création d'une tâche



Capture d'écran 9 : Les matières affichées sont celles dont l'étudiant a, à la date sélectionnée



Capture d'écran 10 : Les différents modes de répétitions

TABLE DES ILLUSTRATIONS

TABLE DES SCHEMAS