|  |
| --- |
|  |
| Application MultiLingua Réalisation d'un prototype | |
|  |
|  |
| **Rédacteur : Philippe Berthelot** |
| **12/06/2017** |

|  |
| --- |
| Prototype d'application mobile pour la société MultiLingua |

# 

# Gestion du document

**Liste de diffusion**

Ce document s'adresse aux intervenants suivants:

|  |  |  |  |
| --- | --- | --- | --- |
| **Prénom Nom** | **Fonction** | **Adresse email** | **Téléphone** |
|  | Client |  |  |
|  | Chef de Projet |  |  |
|  | Développeur Front |  |  |
|  | Développeur Back |  |  |

**Versions du document**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Numéro de version** | **Date d'émission** | **Sections modifiées** | **Modifications principales** | **Observations** |
| Version 1 | 08/06/17 | Sans objet | Sans objet | Première version du proto |
| Version 2 | 12/06/17 |  | Ajout suivi résultats élèves + Homework |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Objectifs du document :**

Ce document sert de référence pour la réalisation du prototype MultiLingua.

Il permet de présenter les concepts de l'appli mobile MultiLingua et de transférer le code du prototype aux futurs réalisateurs.**Table des matières**

[Gestion du document 2](#_Toc485027529)

[Vision globale du projet 5](#_Toc485027530)

[Debrief client 5](#_Toc485027531)

[Cadrage du prototype 6](#_Toc485027532)

[Les différents utilisateurs de l'application 8](#_Toc485027533)

[Spécifications fonctionnelles du protype 9](#_Toc485027534)

[Fonctions directement accessibles à l'utilisateur 9](#_Toc485027535)

[Fonctions induites 9](#_Toc485027536)

[Fonctions liées au fonctionnement en réseau de l'appli 9](#_Toc485027537)

[Représentation des données de l'application 10](#_Toc485027538)

[Modélisation des données 10](#_Toc485027539)

[Stockage et gestion des données 11](#_Toc485027540)

[Fonctionnement du backend minimal pour l'enseignant 15](#_Toc485027541)

[Ce que fait le prototype - Description fonctionnelle 17](#_Toc485027542)

[Fonction Login 18](#_Toc485027543)

[Fonction Lesson et Homework 19](#_Toc485027544)

[Réalisation technique et programmation 28](#_Toc485027545)

[Choix de technologies 28](#_Toc485027546)

[Environnement de développement 31](#_Toc485027547)

[Architecture de l'application 33](#_Toc485027548)

[Processus d'initialisation et d'acquisition des données 34](#_Toc485027549)

[Gestion de la connexion Internet. 35](#_Toc485027550)

[Récupération des données dans la base de données Firebase 35](#_Toc485027551)

[Identification de la zone de données à interroger 35](#_Toc485027552)

[Stockage des identifiants Firebase 36](#_Toc485027553)

[Accès depuis l'application aux données stockées dans la base de données de Firebase 36](#_Toc485027554)

[Récupération des fichiers de données dans le Cloud 37](#_Toc485027555)

[Transfert de fichiers depuis une adresse externe avec la native Transfer de Ionic 2 37](#_Toc485027556)

[Gestion de la persistance des données dans l'application 37](#_Toc485027557)

[Utilisation du local storage 38](#_Toc485027558)

[Gestion de la déconnexion/reconnexion du réseau 38](#_Toc485027559)

[Gestion du login de l'élève 38](#_Toc485027560)

[Gestion de la composition des pages de leçons et d'exercices 38](#_Toc485027561)

[Récupération des données depuis Firebase 39](#_Toc485027562)

[Nommage des fichiers de données en local 40](#_Toc485027563)

[Construction de la vue des leçons 40](#_Toc485027564)

[Construction de la vue des exercices relatifs à une leçon 42](#_Toc485027565)

[Construction de la vue des exercices de révision leçon + leçon précédente 44](#_Toc485027566)

[Visualistion d'une série d'exercices à l'écran 45](#_Toc485027567)

[Les mécanismes d'injection de dépendance dans Angular 49](#_Toc485027568)

[Gestion du calendrier 51](#_Toc485027569)

[Gestion de l'apparition/disparition des pages et et de la navigation 56](#_Toc485027570)

[Gestion du transfert de données entre pages 58](#_Toc485027571)

[Gestion du suivi des résultats d'un élève et fonction de "homework" 59](#_Toc485027572)

[Enregistrement des résultats de l'élève 59](#_Toc485027573)

[Transfert du fichier vers le backend du professeur 61](#_Toc485027574)

[Gestion d'un travail personalisé donné par le professeur à l'élève 62](#_Toc485027575)

[Gestion de la présence du homework dans l'application 65](#_Toc485027576)

[Conclusions 67](#_Toc485027577)

[Glossaire 68](#_Toc485027578)

# Vision globale du projet

## Debrief client

Le client souhaite :

proposer à ses élèves une expérience unique consistant à :

1 - écouter ou lire une micro-leçon chaque jour,

2 - à valider sa compréhension à travers divers exercices très courts.

3 - Les exercices doivent porter sur la leçon du jour mais peuvent aussi porter sur la leçon d’il y a quelques jours, pour vérifier si elle a bien été retenue.

4 - En outre, l’application doit permettre de contacter son responsable de formation chez Multilingua

5 - Et de voir les prochaines dates de formation présentielle programmées.

6 - Un rappel doit être envoyé par l’application 1h avant la formation.

## Cadrage du prototype

Le prototype a pour but de montrer notre capacité à satisfaire la demande du client.

Il doit permettre de présenter une démonstration "vendeuse" qui donne envie de poursuivre le projet avec notre société.

Ce prototype n'a pas vocation a représenter l'ensemble des fonctionnalités de l'application finale, en particulier, il n'est pas demandé de concevoir un backend à ce stade du projet. Il n'est pas non plus demandé de fournir une base d'exercices réalistes, de simples exercices avec question réponse peuvent suffire.

Le prototype a principalement vocation à montrer notre capacité à réaliser des applications sur téléphone mobile. Les leçons et les exercices sont stockés dans des fichiers de texte. Il n'est pas question, à ce stade de prototypage, de proposer une base de donnée des leçons et des exercices.

Il nous a semblé judicieux de proposer deux façons de s'exercer après un cours de langue :

- proposer une leçon suivie d'une liste d'exercices spécifiques à cette leçon,

- proposer un mode "révision" avec des exercices de la leçon en cours et un certain nombre d'exercices tirés de la leçon précédente pour révision.

Le but est ici de montrer la faisabilité d'une présentation dynamique d'une série d'exercices tirés d'une base de données des exercices (ou ici d'un simple fichier des exercices) avec des critères plus ou moins complexes de choix de ces exercices.

**Une esquisse de backend**

Pour présenter au client un prototype réaliste et convainquant, le choix a été fait d'intégrer un backend minima,l réalisé avec peu d'effort de développement pour rester dans le cadre du prototypage rapide. Cela permet au prototype d'être alimenté en données pédagogique, en calendriers et en données de paramétrage d'une façon qui n'est pas codée "en dur" dans l'application. Il est ainsi possible d'illustrer la gestion du calendrier de l'enseignant avec un accès différentié de l'élève uniquement aux informations qui le concernent et non aux autres rendes-vous de l'enseignant.

**Fonctionnement de l'application sans connexion internet**

Le prototype illustre notre compréhension des spécificités du fonctionnement des applications sans réseau : lorsque la connexion internet n'est pas disponible, si les données ont été précédemment chargées lorsque le réseau était disponible, alors l'élève peut continuer à s'exercer. Il est informé qu'il est hors connexion. Il est de même informé d'une reprise de la connexion.

Cela suppose donc que les données de travail nécessaire à l'élève soient stockées sur son téléphone après avoir été transférées depuis le backend.

**Limites du prototype : suivi pédagogique des élèves**

Il n'est pas demandé de mettre en place une mémorisation et un suivi des résultats des élèves, l'application est a priori prévue comme un outil d'entrainement, mais pas d'évaluation ou de suivi pédagogique des élèves.

**Son et leçons enregistrées**

On n'a pas exploré la possibilité de permettre à l'élève d'écouter une leçon, il peut simplement la lire sur son écran. Mais dès lors que l'on sait charger des fichiers depuis le Cloud, ce qu'illustre bien cette applciation, on saurait de la même manière importer des fichiers son et les mettre à disposition de l'élève pour écoute.

**Evolutions à envisager et suggestions pour le client**

Pour une version vraiment opérationnelle de l'application, il nous semblerait utile :

1 - que le client puisse proposer des exercices personnalisés destinés à un élève ne particulier, dans une démarche pédagogique de coaching différencié.

2 - que le client puisse obtenir un retour sur le travail d'un élève en particulier, pour éventuellement ajuster ses prochains exercices à cet élève.

Une conception modulaire de l'application intégrant ces suggestions doit permettre ces évolutions ultérieures sans effort de développement additionnel significatif.

## Les différents utilisateurs de l'application

L'application est destinée aux élèves de MultiLingua apprenant l'anglais.

L'enseignant n'intervient pas directement sur l'application elle même, il est supposé fournir les données pédagogiques et les intégrer à la base de données des leçons et exercices.

Le prototype fonctionne pour un enseignant et plusieurs élèves.

# Spécifications fonctionnelles du protype

Cette partie du document fournit la liste des fonctions que l'application doit remplir quelle que soit la stratégie d'implémentation de ces fonctions.

## Fonctions directement accessibles à l'utilisateur

F 1- 1 Présenter une leçon à l'élève

F 1-1-1 Présenter la leçon en cours d'étude

F 1-1-2 Présenter la leçon précédente

F 1-1-3 Présenter la leçon suivante

F 1-2 Présenter une série d'exercices à l'élève

F 1-3 Présenter des exercices de révision d'une leçon

F 1-4 Présenter à l'élève le calendrier des prochains rendez-vous avec l'enseignant

F 1-4-1 Visualiser le calendrier

F 1-4-2 Permettre à l'élève d'entrer un rendez-vous dans son agenda personnel

F 1-5 Permettre à l'élève de contacter l'enseignant

F 1-6 Permettre à l'élève de s'identifier

Pour lire les données qui lui sont spécifiques, en particulier ses rendez-vous dans le calendrier.

## Fonctions induites

F 2 - 1 Lire les données des leçons en base de données

F 2- 2 Lire les données des exercices en base de données

F 2 - 3 Lire les données de paramétrage

## Fonctions liées au fonctionnement en réseau de l'appli

F 3 - 1 Gérer les déconnexions/reconnexions éventuelles du réseau et d'Internet

# Représentation des données de l'application

L'Application utilise des jeux de leçons et d'exercices fournis par le formateur pour permettre à l'élève de lire une leçon

## Modélisation des données

Les données manipulées par l'application sont :

1 - Les leçons et les exercices

2 - Les données relatives aux fichiers qui contiennent ces leçons et ces exercices

3 - Les données relatives à l'enseignant et au paramétrage de l'application

4 - Les données relatives

Une leçon est caractérisée par les attributs suivants :

id : number;

objective :string;

text :string

Un exercice est caractérisé par les attributs suivants :

lessonId : number;

exerciseName: string;

question: string;

possibleAnswers: [(string)];

rightAnswer: string;

explanation: string

Un enseignant est caractérisé par les attributs suivants :

firstName : string;

lastName: string;

eMail: string;

phoneNumber: string;

Un élève est caractérisé par les attributs :

name : string;

key: string;

key est une clé (chaîne de caractère) donnée par l'enseignant pour limiter l'accès des élèves aux données qui les concernent personnellement mais qui ne concernent pas les autres élèves.

## Stockage et gestion des données

Les données nécessaires au fonctionnement du prototype sont supposées être mises à disposition par l'enseignant avec :

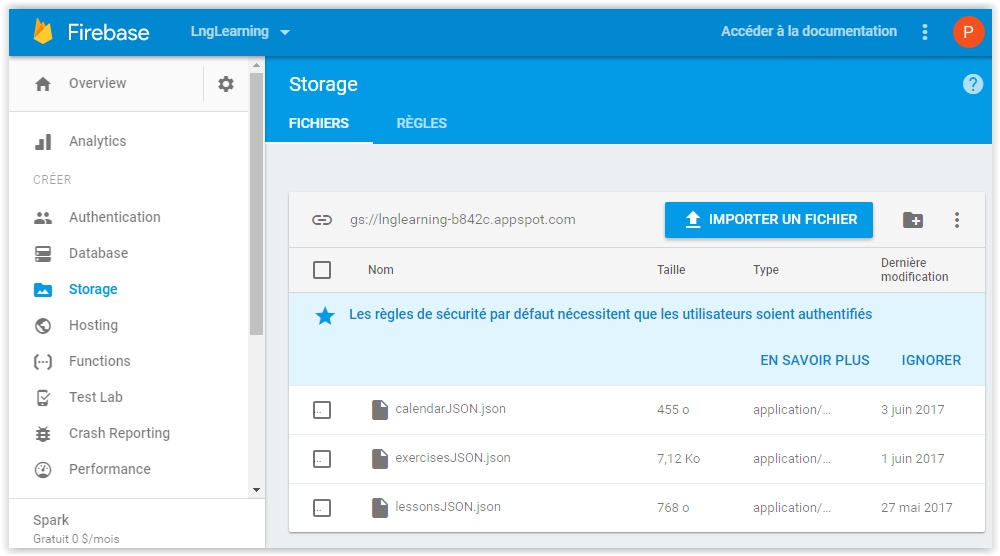
- un fichier lessonsJSON qui contient les textes des leçons,

- un fichier exercisesJSON qui contient les textes des exercices,

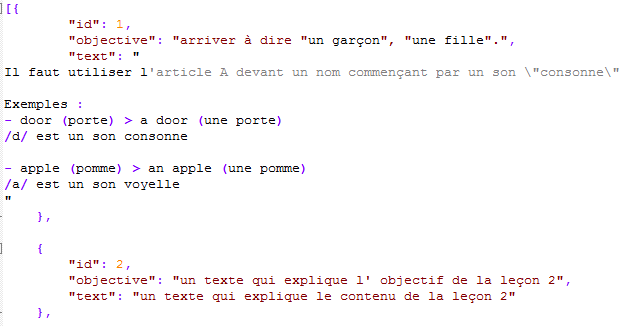
- un calendrier calendarJSON qui contient le calendrier de l'enseignant.

Ces données sont stockées dans des fichiers textes au format JSON et mises à disposition pour l'application sur un Storage de Firebase.

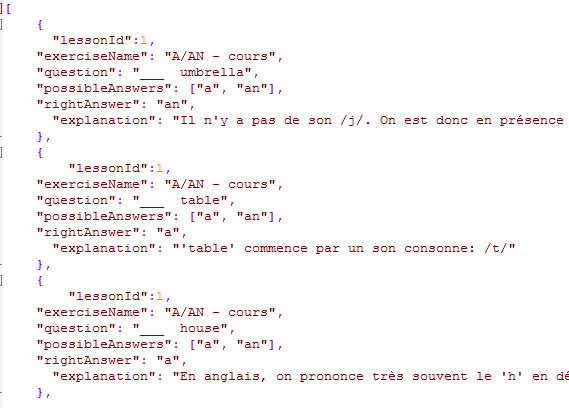
Ces fichiers sont chargées sur le portable e l'élève au démarrage de l'application. Il peut ainsi, ensuite, continuer à travailler off-line s'il n'a plus la connexion avec internet.



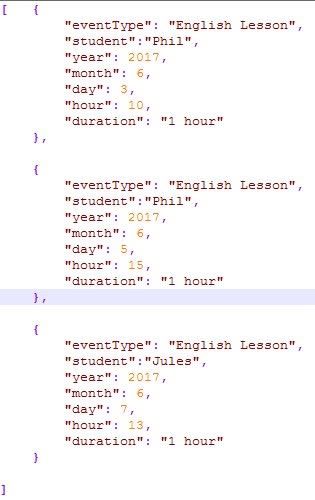
Contenu type du fichier des leçons :

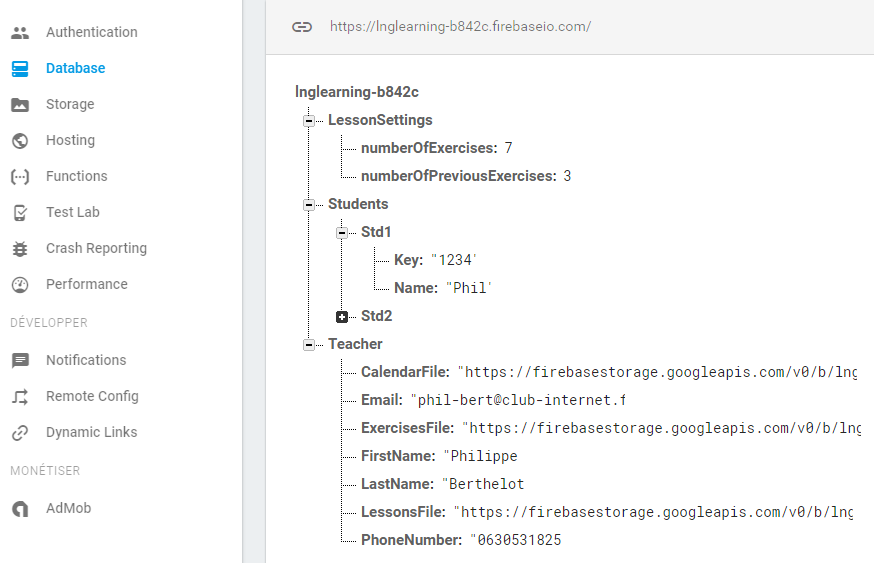


Contenu type du fichier des exercices



Contenu type du calendrier

Données de paramétrage stockées en base de données Firebase



Les champs CalendarFile, ExerciseFile et LessonsFile contiennent les liens de téléchargement générés par Firebase.

La table Students contient les identifiants des élèves pour filtrer les rendez-vous dans le calendrier personalisé de chaque élève.

## Fonctionnement du backend minimal pour l'enseignant

L'enseignant peut se connecter au compte Firebase de l'application pour :

- renseigner ses coordonnées, nom, prénom, téléphone, Email ;

- paramétrer le nombre d'exercises à proposer pour une leçon et le nombre d'exercices de révision à proposer à l'élève dans une cession d'exercices.

- donner les liens de téléchargement des fichiers ressources, ne tout premier lieu de son calendrier des leçons dès lors qu'il a été réactualisé.

Pour mettre à jour les données de l'application, en particulier le calendrier, l'enseignant :

1 - modifie les fichiers de données sous un éditeur de texte sur sa machine ;

2 - vérifie la compatibilité de son fichier avec le format JSON par exemple sur des sites https://jsonlint.com/ ou https://jsonformatter.curiousconcept.com/

**(le prototype n'effectue pas de validation de format JSON, il suppose a priori que le fichier fourni est au format JSON et a été préalablement vérifié).**

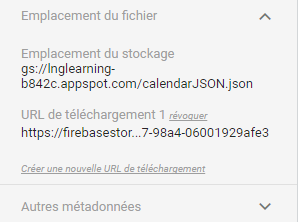
3 - se connecte au compte Firebase de l'application, sous l'onglet Storage ;

4 - il charge la nouvelle version du fichier de données qu'il vient de modifier ;

5 - il clique sur le nom du fichier qu'il vient de charger pour ouvrir la fenêtre suivante :



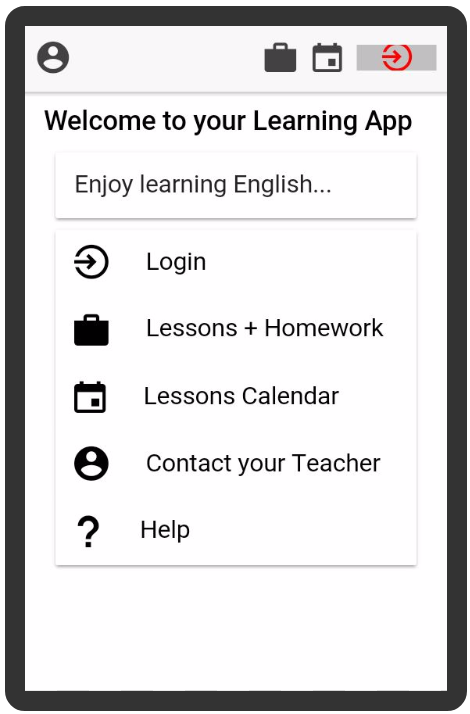
En cliquant sur Emplacement du fichier, il ouvre la fenêtre ci-dessous :



En cliquant sur URL de téléchargement, il mémorise dans son clipboard le lien à recopier dans la base de données. Il revient alors sur la fenêtre de la base de données en cliquant sur l'onglet *Database* à gauche de l'écran. Il peut alors coller le lien de téléchargement depuis le clipboard dans le champ qui contient le lien de téléchargement du fichier, par exemple CalendarFile, si c'est le fichier calendar qui vient d'être modifié.

# Ce que fait le prototype - Description fonctionnelle

Au démarrage de l'application, l'utilisateur dispose des options suivantes :



## Fonction Login

La fonctionnalité de login a été ajoutée à l'application pour montrer sa capacité à différencier le traitement des données adressées à un utilisateur particulier. En effet, le calendrier de l'enseignant concerne tous ses élèves, mais un élève en particulier n'est pas concerné par les dates des autres élèves.

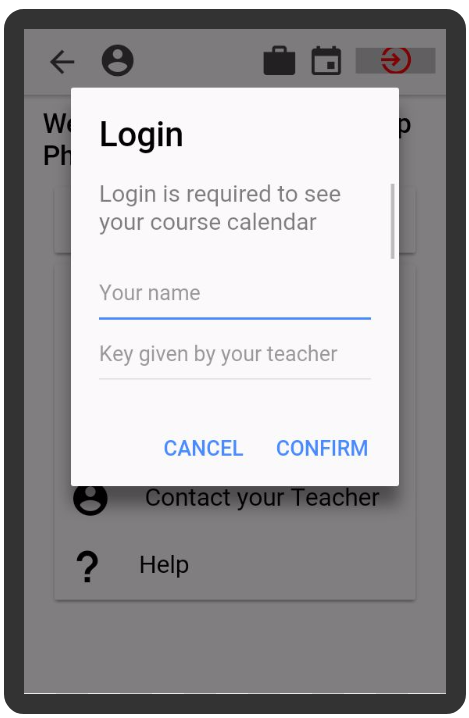
Le login ici mis en oeuvre n'implique donc pas un mot de passe et le but n'est pas non plus de collecter des données relatives à l'élève, mais simplement d'authentifier l'élève à l'aide d'une clé, en fait une chaîne de caractères, fournie par l'enseignant pour autoriser un élève en particulier à accéder à ses propres dates de cours sans l'informer de l'ensemble des dates de l'enseignant.

Cette fonction est accessible par les icônes :

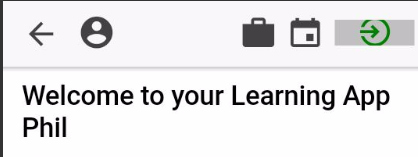




Permet d'ouvrir une fenêtre de saisie des identifiants de l'élève

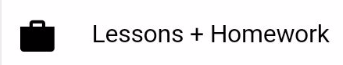


Une fois l'élève identifié, l'icône du menu de navigation change de couleur, devient verte si le login est correct, et le nom de l'élève s'affiche alors dans le titre de la page :



## Fonction Lesson et Homework

Cette fonction est accessible par les icônes :

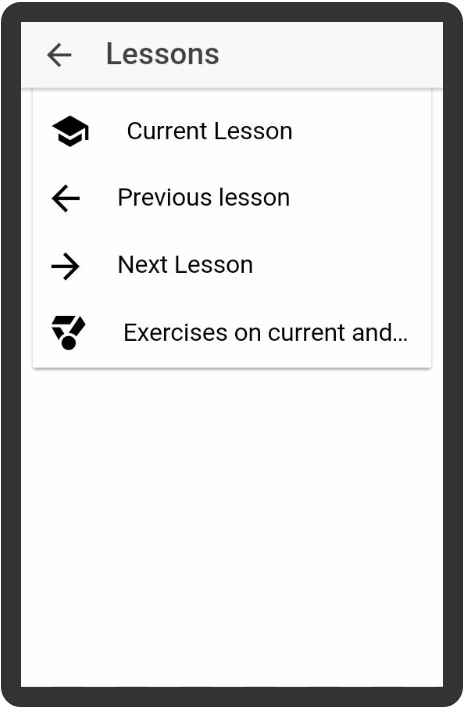


et

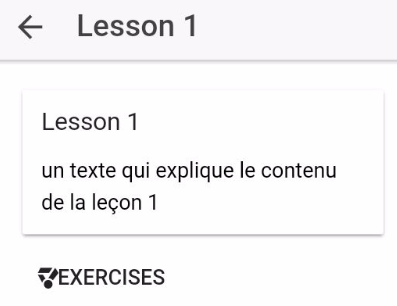


dans la barre de navigation

Elle ouvre la page d'accès aux leçons :

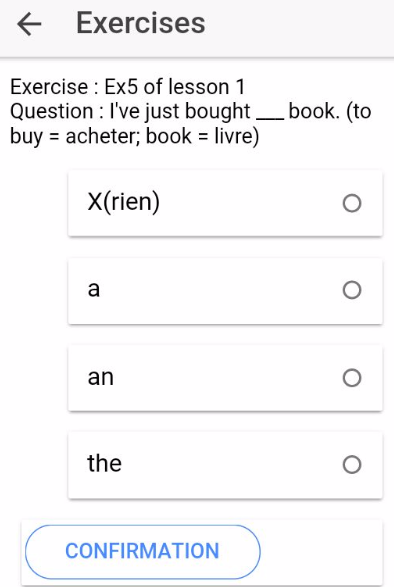


Cliquer sur Current Lessons permet alors de visualiser la micro leçon et d'accéder aux exercices associés :

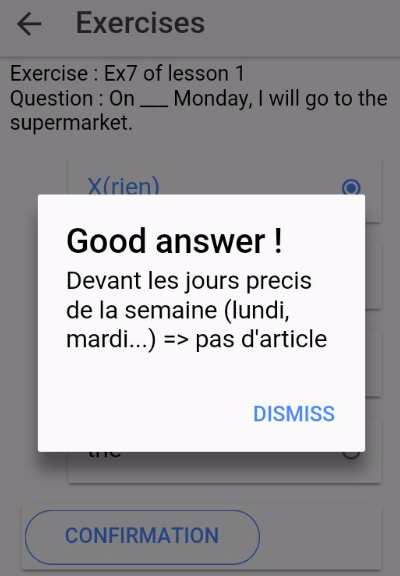


Cliquer sur EXERCICES ouvre la fenêtre suivante, dans laquelle l'élève peut s'exercer .

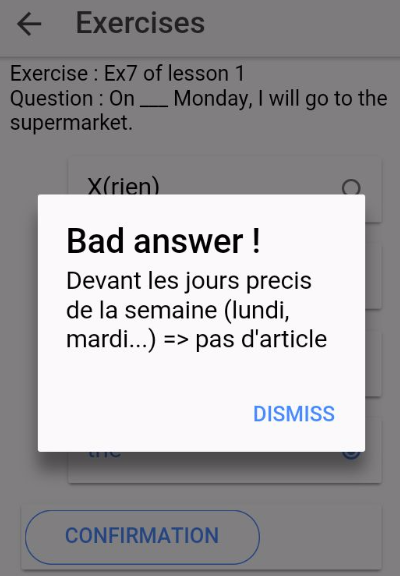
Les exercices portent ici exclusivmement sur la leçon en cours d'étude, une zone de révision de la leçon en cours et de la précédente étant par ailleurs à disposition de l'élève sous le titre "Exercises on current and last lessons" dans la page "Lessons".



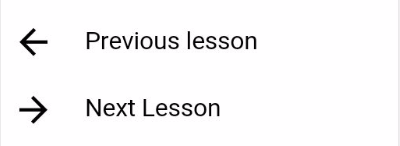
Une fois qu'il a confimé sa réponse, il reçoit le résultat :



ou

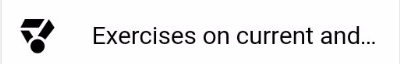


Les fonctions

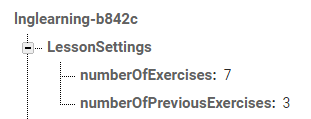


permettent de naviguer d'une leçon à l'autre. La leçon "en cours d'étude" devient alors la leçon qui vient d'être selectionnée. Elle est aussitôt mémorisée dans le stockage local du portable pour être rappellée en tant que "Current lesson" lorsque l'élève se reconnectera ultérieurement.

La fonction



permet une révision de la leçon précédente : la liste des exercices proposés est constitué d'exercices de la leçon en cours d'étude et d'exercices de la leçon précédente. Le nombre d'exercices par séance de révision et le nombre d'exercices tirés de la leçon passée sont paramétrées par le formateur dans les champs de la table LessonSettings dans Firebase.



numberOfExercises definit le nombre total d'exercices dans la séquence et numberOfPreviousExercises le nombre d'exercices de la leçon précédente à insérer dans la séquence.

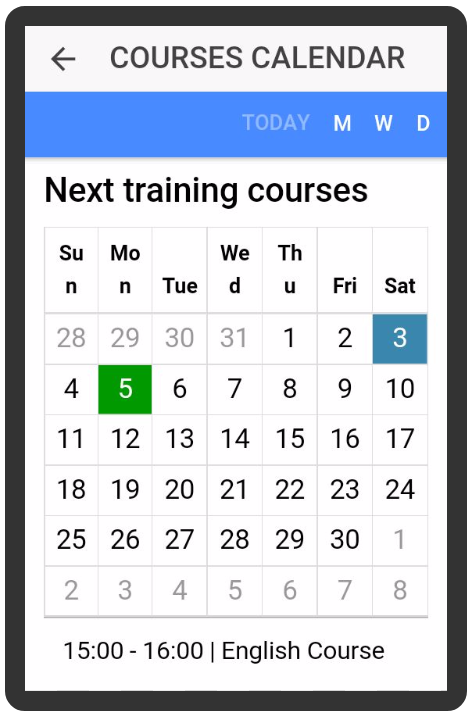
La fonction Lessons Calendar accessible par les icônes :



et dans la barre de menu :



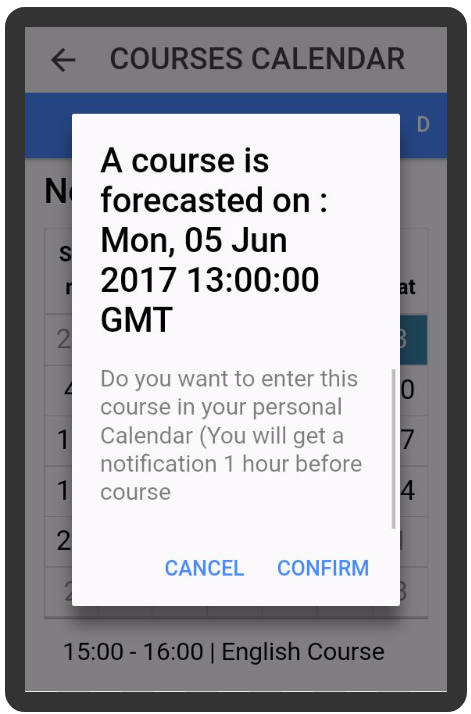
ouvre le calendrier des prochaines leçons de l'élève.



Il est alors possible de cliquer sur la zone :



qui ouvre un menu destiné à permettre à l'élève d'entrer la date de cours dans son agenda personnel.

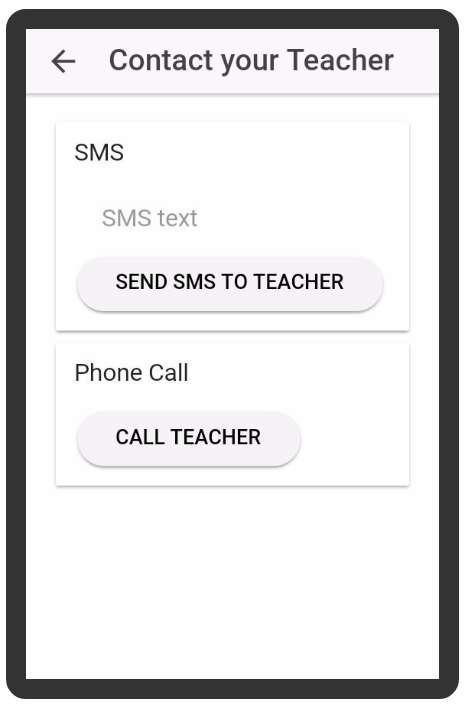


Cliquer sur Confirm ouvre le calendrier de l'élève sur son portable et lui envoie une première notification pour lui signifier que la date a été enregistrée et qu'il recevra une nouvelle notification une heure avant le cours.

La fonction Contact appeléé par :



ouvre une fenêtre qui permet d'appeler directement l'enseignant.



Les coordonnées utilisées pour appeler l'enseignant sont celles qu'il a entrées en base de données dans Firebase, dans la table Teacher. Elles sont réactualisées en temps réel au cas où l'enseignant les modifieraient depuis le backend.



# Réalisation technique et programmation

## Choix de technologies

**Ionic 2**

Le choix a été fait de développer le prototype avec un outil de dévelopement cross-plateform basé sur les paradigmes de la programmation Web, avec des interfaces homme-machine en HTLM et CSS.

L'environment retenu est Ionic 2, la dernière version stable du framework Ionic.

Ionic 2 présente des différences significatives vis à vis de la version précédente du framework Ionic : utilisation de typescript à la place de javascript et de Angular 2 à la place de Angular JS.

On peut considérer ces évolutions comme une avancée vers un cadre de développement orienté Objet plus proche des standards des langages de programmation courants (Java, Python etc.) avec un modèle de type MVC (Model/View/Controler) qui permet une organisation rigoureuse du développement avec une réelle indépendance des interfaces homme/machine par rapport au traitement des données.

Ce framework, dans sa version 2, est significativement plus facile à apprendre que ses prédécesseurs.

Ionic 2 offre une riche bibliothèque de fonctionalités dite 'natives' qui permettent d'interagir avec le téléphone. Ces "natives" masquent totalement la couche de communication avec les fonctionnalités spécifiques au portable, gérée, en fait, par le framework Cordova, devenu totalement transparent pour le développeur de l'application.

Dans les paradigmes Ionic 2, on distingue la

Les demandes fonctionnelles du cient peuvent être toutes satisfaites sans difficulté avec cette bibiliothèque, en particulier :

- l'interaction de l'application avec le calendrier de l'élève, qui se gère avec la primitive Calendar (https://ionicframework.com/docs/native/file/)

- l'interaction de l'application avec les outils de communication spécifiques d'un téléphone, tels que l'emission d'un appel téléphonique ou l'envoi d'un SMS (https://ionicframework.com/docs/native/call-number/ , https://ionicframework.com/docs/native/sms/, https://ionicframework.com/docs/native/local-notifications/ )

- la persistance des données, le fait de stocker la dernière leçon étudéieé pour redémarrer directement sur cette leçon à la prochaine cession, (https://ionicframework.com/docs/native/native-storage/)

- les transferts de fichiers depuis le cloud, qui se gèrenet facilement grâce aux "primitives " File et FileTransfer (https://ionicframework.com/docs/native/file/)

Un avantage majeur, dans notre contexte réside dans l'accessibilité du framework à un débutant qui évolue dans le cadre des technologies du Web.

Rappel du cadre général de développement du framework Ionic :

**Limitations relatives à Ionic 2**

La technologie n'est pas stable, elle évolue très rapidement, et la correction de nombreux bugs exige la fréquentation assidue des forums Ionic (https://forum.ionicframework.com/) et Stack Overflow (https://stackoverflow.com/).

Une Application est composée de très nombreux modules, dont on ne gère pas toujours les versions qui se chargent au démarrage d'une application et qui peuvent être instables ou présenter des bugs, ainsi que l'on en a fait l'expérience à plusieurs reprises.

Il ne nous est pas possible de comparer les performances entre une application native et une application cross-platform, mais on peut craindre une dégradation des performances du fait de la multiplication des couches logicielles de l'application.

**Le choix de Firebase**

Pour stocker des données et réaliser des fonctionnalités minimales de backend, Firebase nous a semblé un bon choix pour les raisons suivantes :

- gratuité pour un volume de données et de transactions limité ;

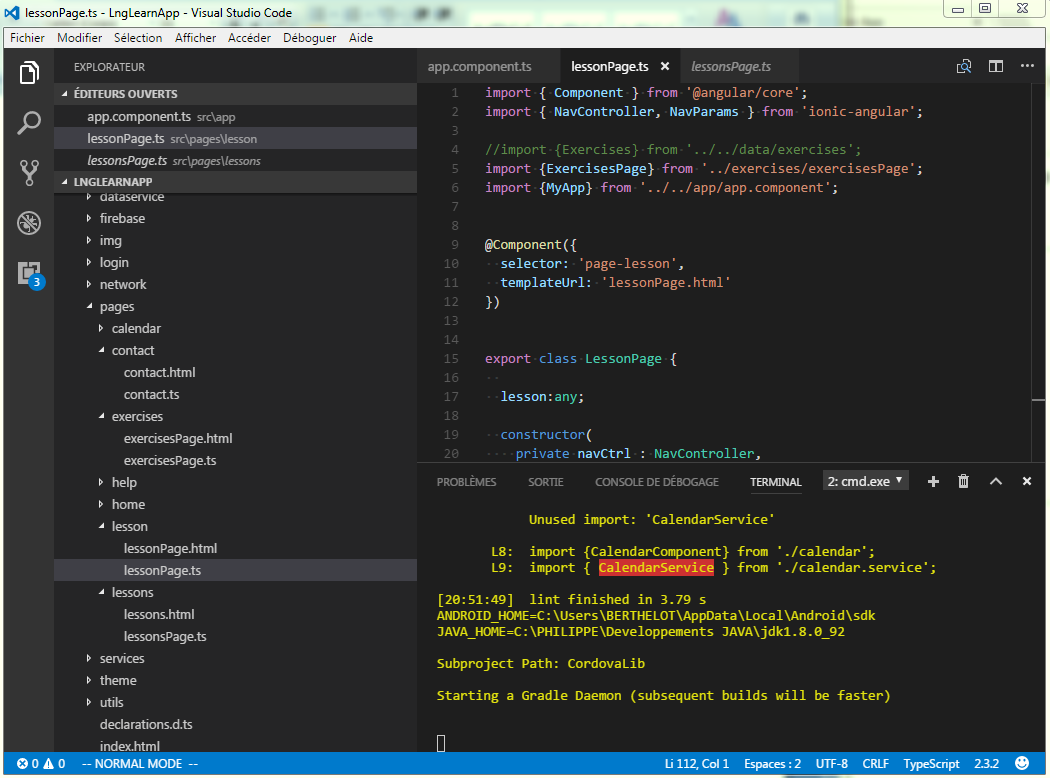
- facilité d'utilisation pour un débutant, y compris en bases de données ; il n'est en particulier pas nécessaire de maîtriser SQL ou autre langage d'interrogation de bases de données ;

- faible confidentialité des données : le recours à une solution Cloud gérée par un tiers qui dispose de fait d'un accès aux données du client serait critique si les données étaient fortement confidentielles, mais dans le cadre de ce projet, ce n'est pas le cas.

## Environnement de développement

Le propos n'est pas ici de détailler la procédure d'installation des différentes utilitaires logiciels nécessaire au développement, qui se retrouve facilement sur les nombreux tutoriaux consacrés à l'installation de Ionic sur Internet.

Le cadre utilisé pour le développement du projet est l'IDE Visuel Studio Code, qui présente l'avantage d'être gratuit et suffisamment performant pour travailler efficacement à la réalisation de ce prototype.



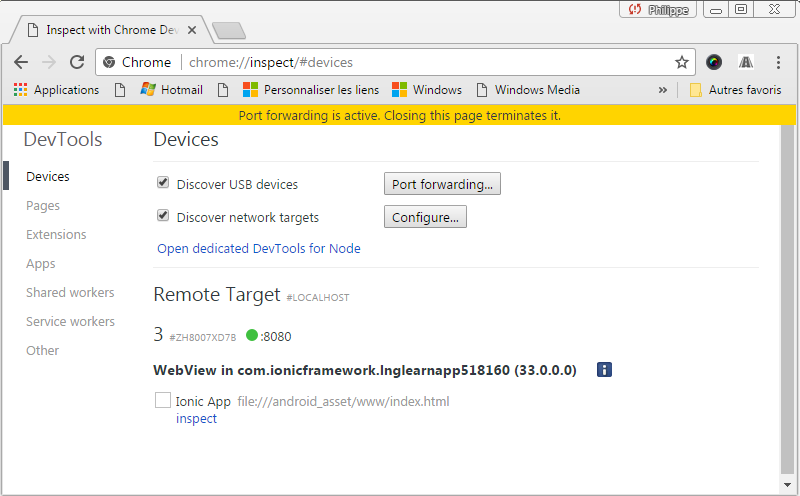
Pour tester l'application, plusieurs possibilités s'offrent au développeur :

- test dans le cadre d'un Web Browser

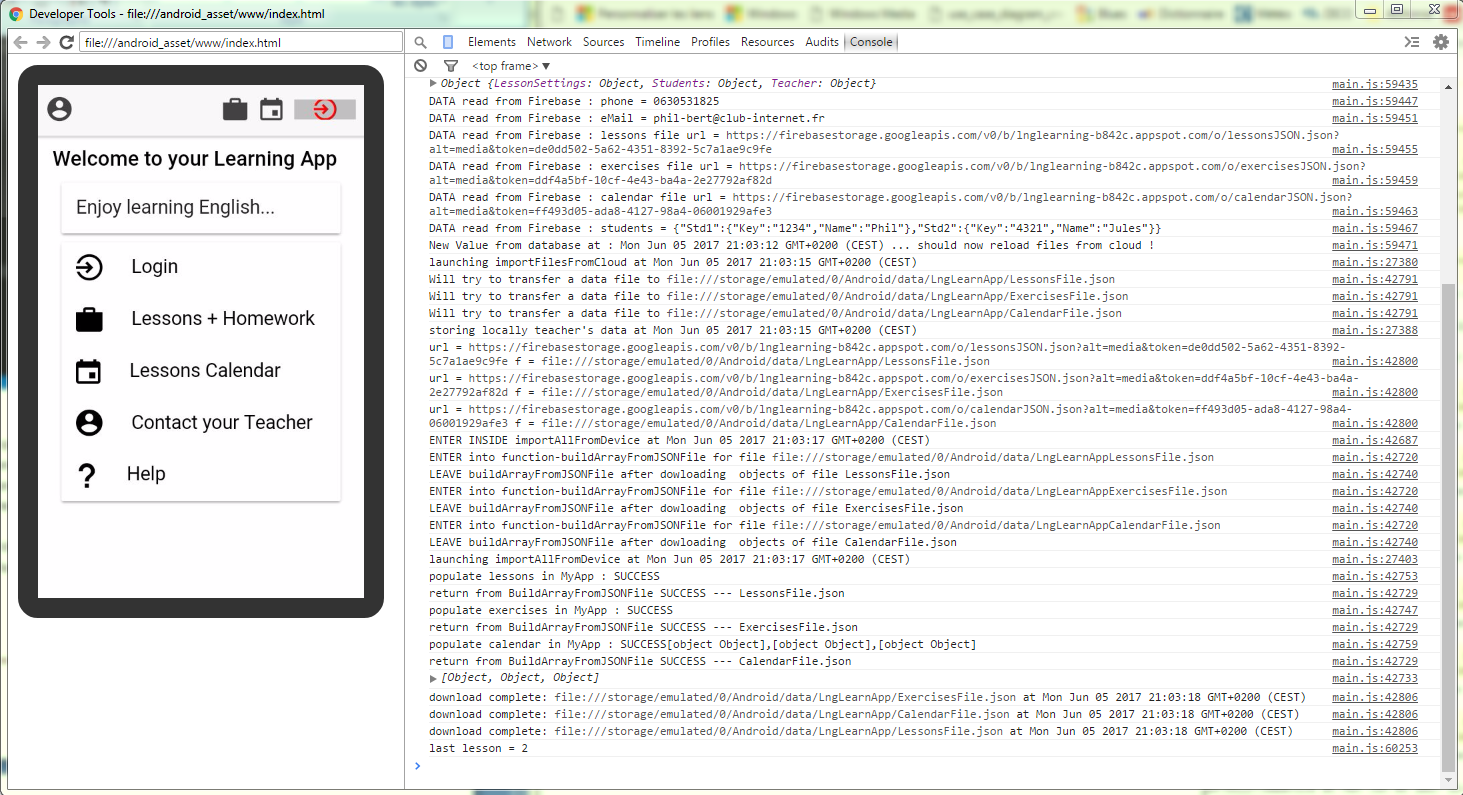
- test sur un émulateur de téléphone portable

- test directement sur un téléphone portable.

Après avoir testé les différentes solutions, on a finalement opté pour un test direct sur le téléphone portable en connexion USB avec l'ordinateur de développement ; une technique tout à fait praticable du point de vue des performances, et efficace lorsqu'elle est réalisé avec l'outil de test de chrome.

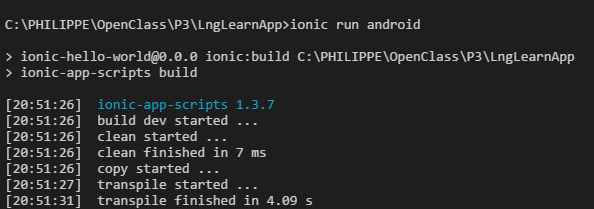


Cet outil permet d'ouvrir une vue de fonctionnement de l'application avec une console qui donne ensuite la possibilité de lire les messages envoyés par l'application (par la commande console.log('...') ) pour tester son fonctionnement.



Il suffit alors, une fois le téléphone connecté à l'ordinateur de développement, de lancer la compilation de l'application avec la commande : ionic run android.

(Limitation : on n'a testé l'application que sur un téléphone Android).



## Architecture de l'application

Les framework Ionic et Angular imposent une architecture modulaire de l'application.

Les principaux modules peuvent être regroupées en trois familles principales :

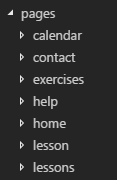
- le module app : le composant principal de l'application qui contient toutes les tâches effectuées au démarrage de l'application.

- les modules pages : qui concernent la gestion de tout ce qui est présenté à l'utilisateur final, la gestion des interfaces homme/machine et de données affichées dans ces interfaces.

- les modules utilitaires qui remplissent des services sans être directement visibles pour l'utlisateur.

**Modules gérant l'affichage d'informations à l'écran**

Les modules PAGES concernent donc les vues affichées à l'écran lors des navigations de l'utilisateur.



- le module home contient tout ce qui concerne la page d'accueil de l'application.

- les modules lessons, lesson, exercises, permettent l'affichage des leçons et des exercices.

- le module calendar gère le calendrier des cours à venir.

- le module contact gère les contacts possibles avec formateur depuis le téléphone.

**Modules utilitaires**

- module de gestion des données :

Le module data-services gère l'importation des données dans l'application depuis les fichiers de données stockés sur le téléphone. Ce module gère aussi l'importation des fichiers vers le téléphone depuis le Cloud.

Le module firebase gère la connexion avec le Cloud et la récupérationdes données et des fichiers depuis les utilitaires de l'application Firebase.

## Processus d'initialisation et d'acquisition des données

L'initialisation de l'application est effectuée dans le module app, par le programme app.component.ts. La logique de l'initialisation est résumée par le diagramme ci-dessous.

Si l'on dispose d'une connexion au lancement de l'application, alors on recharge les données depuis Firebase, prenant ainsi en compte les modifications apportées par l'enseignant ; sinon, si des fichiers de données sont déjà présents sur le téléphone, on recharge les données à partir de ces fichiers. Si ni la connexion Internet, ni les fichiers en local ne sont disponibles, l'élève devra attendre la reconnexion avec le réseau.

## Gestion de la connexion Internet.

Le déconnexion et la reconnexion au réseau et à Internet sont gérées en temps réel tout au long de la cession d'utilisation de l'application par la primitive **Network** (https://ionicframework.com/docs/native/network/ ) et les deux méthodes associées à la classe Network de Ionic 2:

- network.onDisconnect().subscribe pour surveiller les déconnexions éventuelles;

- network.onConnect().subscribe pour surveiller la reprise de la connexion;

## Récupération des données dans la base de données Firebase

L'interface entre Ionic et Firebase est gérée de façon transparente pour l'utilisateur avec le module firebase fourni avec les modules de node.

### Identification de la zone de données à interroger

Les informations de configuration sont obtenues depuis le site Firebase et sont appelées lors de l'invocation de la méthode : initializeApp(FirebaseConfig)

Ces données permettent l'identification de la base et sa connexion à l'application. Elles sont fournies par Firebase en cliquant sur Ajouter Firebase à votre application Web dans la fenêtre d'accueil d'un projet Firebase : 

### Stockage des identifiants Firebase

Une fois recopiés dans Firebase, les identifiants du projet Firebase sont ensuite stockésdans la classe FirebaseConfig :

export const firebaseConfig = {

apiKey: "AIzaSyDYS9GRE19PhVrqv6zIl9icr6J68bqxvcg",

authDomain: "lnglearning-b842c.firebaseapp.com",

databaseURL: "https://lnglearning-b842c.firebaseio.com",

projectId: "lnglearning-b842c",

storageBucket: "lnglearning-b842c.appspot.com",

messagingSenderId: "544188967179"

}

### Accès depuis l'application aux données stockées dans la base de données de Firebase

1 - la connexion à une base de données est gérée par le module database qu'il faut en premier lieu importer dans la classe qui va servir d'interface avec Firebase (FirebaseInterface) :

import { database ,initializeApp } from 'firebase';

la connexion à la base de données est effectuée par le module database que l'on invoque ainsi :

var root = database().ref();

la récupération des données est ensuite réalisée par

root.on('value', function(snap) {

var dataFirebase :any = snap.val();

puis l'accès aux données s'effectue à l'aide de la variable dataFirebase de la façon suivante :

par exemple, pour accéder au numéro de téléphone de l'enseignant :

dataFirebase['Teacher'].PhoneNumber

ou pour charger la tabledes élèves :

dataFirebase['Students']

## Récupération des fichiers de données dans le Cloud

Cette récupération est effectuée par le module dataservice en utilisant la native FileTransfer de Ionic2 .



### Transfert de fichiers depuis une adresse externe avec la native Transfer de Ionic 2

on doit en premier lieu importer la native Transfer du module ionic/native :

import { Transfer, FileUploadOptions, TransferObject }

from '@ionic-native/transfer';

on peut alors utiliser la native en créant un objet destiné à gérer le transfert :

var fileTransfer: TransferObject = this.transfer.create();

puis réaliser un transfert de fichier depuis une URL vers le fichier f local renseigné avec son URL locale complète externe avec la commande :

fileTransfer.download(url, f, true)

(on peut ou non autoriser le remplacement d'un fichier déjà existant par le fichier téléchargé).

(procédure d'installation décrite sur : https://ionicframework.com/docs/native/network/ )

## Gestion de la persistance des données dans l'application

Les données sont rendues persistantes dans l'application, pour rester disponibles d'une session à une autre :

- soit en utilisant des fichiers JSON stockés dans une zone de données du téléphone portable,

-soit en utilisant le local storage géré par Ionic.

Les interactions avec les fichiers stockés sur le téléhone portable sont gérées par la native File de Ionic 2 qui fournit tous les utilitaires nécessaires à l'interaction avec des fichiers.

(https://ionicframework.com/docs/native/network/)

Cependant, dans ce prototype, on ne va pas écrire des données dans un fichier en local, ce qui serait une solution possible pour rendre des données persitantes, mais on utilise le "local storage".

### Utilisation du local storage

Il faut en premier lieu importer le module qui gère le local storage dans Ionic :

import { Storage } from '@ionic/storage';

puis l'injecter dans le constructeur de la classe qui va l'utiliser :

constructor(

private storage : Storage,

on peut alors l'utiliser, par exemple pour stocker le numéro de téléphone de l'enseignant :

this.storage.set('teacherPhoneNumber', MyApp.teacherPhoneNumber).then (()=>{});

et pour relire l'information :

this.storage.get('teacherPhoneNumber')

## Gestion de la déconnexion/reconnexion du réseau

C'est le module network et plus particulièrement le programme network.ts qui gére ces connexions et deconnexions du réseau.

Ces connexions et déconnexions sont géres par les méthodes :

this.network.onDisconnect().subscribe

et

this.network.onConnect().subscribe

qui permettent de réagir automatiquement à des événements de connexion et déconnexion du réseau.

## Gestion du login de l'élève

Cette fonctionnalité est implémentée dans le module :



par la classe :

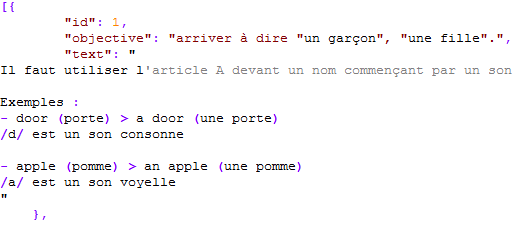
export class StudentLogin

qui fournit la méthode :

studentLogin(navCtrl:NavController){

## Gestion de la composition des pages de leçons et d'exercices

Les leçons sont, initialement, déposées sur la zone storage du projet sur Firebase, en format JSON, par exemple.



Lors de l'initialisation de l'application, le fichier des leçons est chargé sur le téléphone portable.

L'ensemble des fonctionnalités liées à la remontée depuis Firebase et à la mise à disposition des données dans l'application est rassemblé dans le module :



qui met à disposition de l'applciation la classe

export class DataService

dans laquelle se trouvent les méthodes liées au chargement des données.

### Récupération des données depuis Firebase

la méthode :

loadFileFromCloud(fileName, url)

va récupérer les fichiers depuis le storage de Firebase.

le transfert est assuré par la commande :

fileTransfer.download(url, f, true)

qui assure le transfert depuis l'URL d'un fichier sur Firebase Storage vers le fichier en local désigné par f, avec autorisation d'écrasment des versions précédentes du fichier.

Une fois la récupération des fichiers de données effectuées, et une fois ces fichiers stockés en local, il faut les lire et transformer les données au format JSON en objets gérés par Angular.

C'est ce qu'effectue la méthode

importAllFromDevice ()

qui appelle, pour les leçons, la méthode

buildArrayFromJSONFile(fullPath, ExternalFilesConfig.lessonsFileName, 'lessons');

ou pour les exercices :

buildArrayFromJSONFile(fullPath, ExternalFilesConfig.exercisesFileName,'exercises')

Cette méthode appelle à son tour la méthode

populateMyAppFiles (target, JSON.parse(res) )

après avoir lu le contenu du fichier texte par

file.readAsText(filePath, fileName)

La méthode populateAppFiles va assurer le transfert des objets construits par le parsing du texte en JSON dans une variable globale de l'application, en l'occurence :

MyApp.lessonsAll = obj

et

MyApp.exercisesAll = obj

### Nommage des fichiers de données en local

Il est nécessaire de choisir une zone de stockage des données en local, et le nommage de cette zone va dépendre du système d'exploitation.

Les noms des fichiers en local et les noms de répertoire sont définis dans la constante ExternalFilesConfig du module



export const ExternalFilesConfig =

{

lessonsFileName : "LessonsFile.json",

exercisesFileName : "ExercisesFile.json",

calendarFileName: "CalendarFile.json",

pathToDir : {"android" : "Android/data/" , "ios" :"Ios/data/" , "windows" : "Windows/data", "fireOS" : "FireOS/data/" },

// these data path has been tested on android only

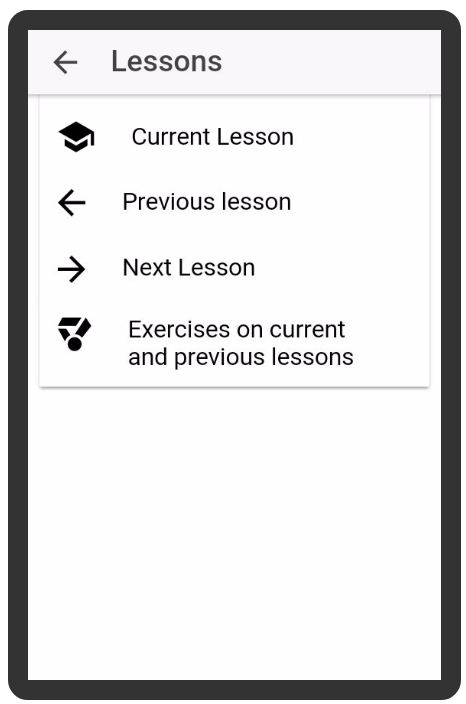
dir:"LngLearnApp"

}

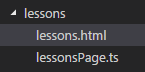
### Construction de la vue des leçons

Interface homme/Machine

La vue **Lessons** permet l'accès à la leçon en cours d'étude, la leçon précédente ou la leçon suivante. Elle donne accès à des exercices de révision de la leçon en cours et de la leçon précédente ( à la différence de la vue spécifique à une leçon, qui permettra l'accès à des exercices spécifiques à la leçon étudée ).



Cette vue est gérée par le module



dont la classe

export class LessonsPage

fournit la méthode :

displayLesson(lessonNumber)

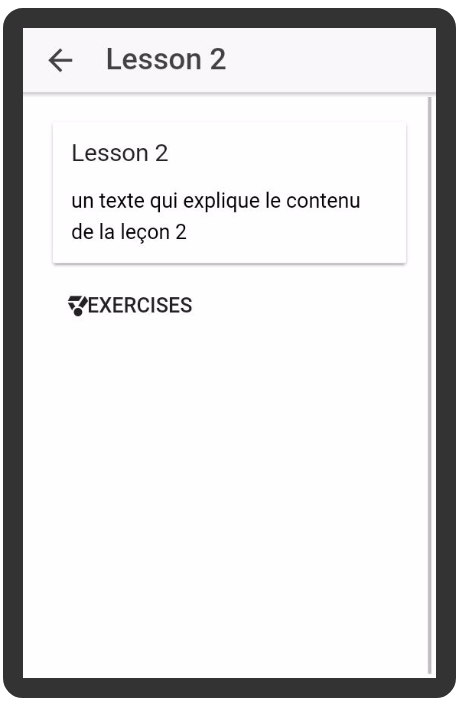
Cette méthode va rechercher dans la variable globale des leçons la leçon spécifiée par lessonNumber, à l'aide de la commande :

var lesson = MyApp.lessonsAll.find(it => it.id === this.lastLesson);

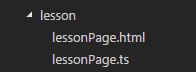
Cette leçon est alors apssée à la vue spécifique à une leçon :

this.navCtrl.push(LessonPage, lesson);

Cette commande ouvre la vue spécifique à une leçon :



qui elle-même donne accès à une page d'exercices, et dont le contrôle est géré par le module :



dont la classe

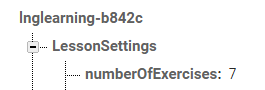
export class LessonPage

apelle la constitution du jeu d'exercises à montrer quand l'élève clique sur EXERCISES.

displayThisLessonExercises

### Construction de la vue des exercices relatifs à une leçon

Le nombre d'exercices à présenter à l'élève dans une session de travail est fixé par le formateur dans ses données Firebase, avec le paramètre numberOfExercises . (Il peut être modifé en temps réel et la modification est automatiquement appliqué à l'élève alors qu'il est en train d'utiliser l'application)



La méthode précédente va sélectionner des exercices en relation avec la leçon en cours d'étude et les place dans le tableau

ExercisesToDisplay

displayThisLessonExercises(lessonId){

var ExercisesTabCurrentLesson = [] ;

var ExercisesToDisplay =[];

// Récupération des exercices de la leçon courrante

for (let exo of MyApp.exercisesAll ) {

if ( exo.lessonId === lessonId){ExercisesTabCurrentLesson.push(exo);};

};

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

var imax = Math.min (ExercisesTabCurrentLesson.length, MyApp.numberOfExercisesPerLesson);

var i = 0;

var indice = 0

while (i < imax) {

indice = Math.floor(Math.random() \* ExercisesTabCurrentLesson.length );

ExercisesToDisplay.push ( ExercisesTabCurrentLesson [indice]);

ExercisesTabCurrentLesson.splice(indice, 1);

i++;

};

this.navCtrl.push(ExercisesPage, ExercisesToDisplay);

}

Dans un premier temps, la méthode va rechercher tous les exercices relatifs à la leçon, ce que fait la commande :

if ( exo.lessonId === lessonId){ExercisesTabCurrentLesson.push(exo);};};

qui vient peupler la table ExercicesTabCurrentLesson avec les exercices de la leçon en cours.

Pour une leçon donnée, il peut y avoir moins d'exercices disponibles dans le fichier des exercices que ce qui est demandé par le formateur avec le paramètre numberOfExercises, d'où le bornage du nombre d'exercices parla variable imax qui gère ce cas de figure.

Dans un second temps, des exercices sont tirés au sort dans cette liste.

while (i < imax) {

indice = Math.floor(Math.random() \* ExercisesTabCurrentLesson.length );

ExercisesToDisplay.push ( ExercisesTabCurrentLesson [indice]);

ExercisesTabCurrentLesson.splice(indice, 1);

i++;

};

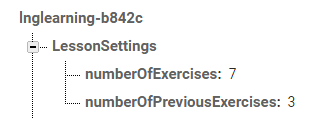
La table ExercisesTabCurrentLesson est en fait la table des exercices encore disponibles pour sélection et affichage ; elle se vide au fur et à mesure que l'on tire au sort des exercices à montrer, qui vont, eux, s'insérer dans la table ExercisesToDisplay.

Un indice est donc tiré au sort dans les limites du nombre d'exercices qui restent dans la table ExercisesTabCurrentLesson. L'exercice correspondant à cet indice est placé dans la table des ExercisesToDisplay par la commande **push** et cet exercice est retiré de la table ExercisesTabCurrentLesson des exercices encore disponibles par la commande **splice**.

### Construction de la vue des exercices de révision leçon + leçon précédente

Le principe de construction de la liste des exercices avec révision de la leçon précédente est sensiblement le même que précédemment.

Le nombre d'exercices de la leçon précédente est déterminé par le paramètre numberOfPreviousExercises fixé par le formateur dans Firebase



Dans ce cas, c'est la classe

export class LessonsPage

qui gère la vue correspondante, tout particulièrement avec sa méthode

displayExercises(lessonId)

adaptée à la double sources d'exercices.

Dans un premier temps, on va charger une liste de tous les exercices possibles pour le leçon n et la leço n-1 :

for (let exo of MyApp.exercisesAll ) {

if (exo.lessonId === lessonId){

ExercisesTabCurrentLesson.push(exo);

};

if (lessonId > 1 && exo.lessonId === (lessonId - 1) ){

ExercisesTabPreviousLesson.push(exo);

};

};

puis, si la leçon courante n'est pas la première, on procède comme précédemment, mais une première fois pour la leçon courante avec selection de

MyApp.numberOfExercisesPerLesson - MyApp.numberOfPreviousLessonExercise

exercices de la leçon courante et selction de

MyApp.numberOfPreviousLessonExercises

exercices de la leçon précédente.

Ce qui conduit à l'algorithme suivant pour la sélection au hasard d'exercices dans la base des exercices fournis par l'enseignant :

if ( lessonId > 1) {

var imax2 = Math.min (ExercisesTabPreviousLesson.length,MyApp.numberOfPreviousLessonExercises);

// imax2 est le nombre maxi d'exercices de la leçon précédente qu'il est possible de proposer

if(Debug){console.log('Constitution de la liste des exercices => imax1 = ' , [imax1] , ' imax2 = ', [imax2])};

while (i < imax1) {

indice = Math.floor(Math.random() \* ExercisesTabCurrentLesson.length);

// (on tire un exercice au hasard dans la liste des exercices)

ExercisesToDisplay.push (ExercisesTabCurrentLesson [indice]);

ExercisesTabCurrentLesson.splice(indice, 1);

i++;

}

i = 0;

while (i < imax2) {

indice = Math.floor(Math.random() \* ExercisesTabPreviousLesson.length);

// (on tire un exercice au hasard dans la liste des exercices)

ExercisesToDisplay.push ( ExercisesTabPreviousLesson [indice]);

ExercisesTabPreviousLesson.splice(indice, 1);

i++;

}

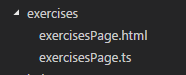
}

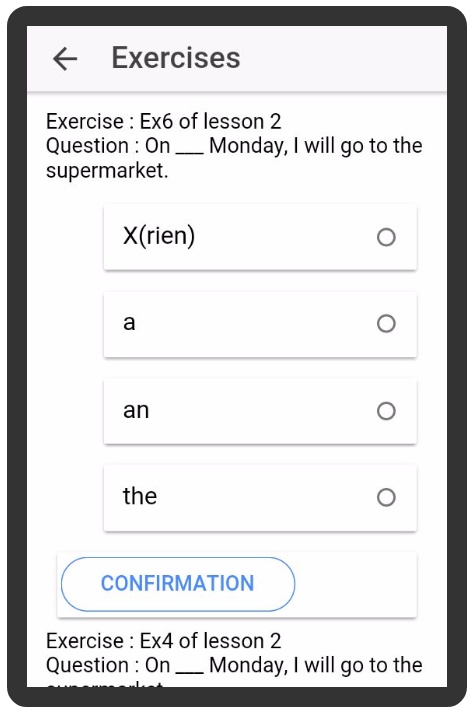
### Visualistion d'une série d'exercices à l'écran

La vue spécifique à une série d'exercices est appelée par la commande :

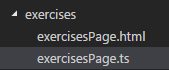
this.navCtrl.push(ExercisesPage, ExercisesToDisplay);

qui envoie la table des exercices à montrer en paramètre de la vue gérée par la classe ExercisesPage.





L'affichage d'une série d'exercices est géré par le module exercises :



La composition d'une vue d'exercices est assurée par l'utilisation de la directive ngfor fournie par Angular. Cette directive permet de boucler sur les exercices présents dans le tableau des exercices nommé ici **exercices** et la deuxième boucle portant sur les réponses possibles à un exercice donné permet de constituer les réponses possibles du QCM.

Puis la directive ngModel permet de faire passer à la classe controlant la vue la réponse à une question sous le nom de "**answ**".

<div \*ngFor="let exercise of exercises">

<div> Exercise : {{exercise.exerciseName}} </div>

<div> Question : {{exercise.question}} </div>

<ul \*ngFor = "let answer of exercise.possibleAnswers">

<ion-card>

<ion-list radio-group [(ngModel)]="answ">

<ion-item>

<ion-label>{{answer}}</ion-label>

<ion-radio value="{{answer}}"></ion-radio>

</ion-item>

</ion-list>

</ion-card>

</ul>

<ion-card>

<button ion-button round outline (click)="respondToAnswer(exercise,answ)">Confirmation</button>

</ion-card>

</div>

Du côté du controleur de cette vue, qui est gérée par la classe :

export class ExercisesPage

on peut constater, en parcourant le code de la classe ExercisesPage, que les exercices qui viennent remplir le tableau exercises de la vue sont passés dans le constructeur par la commande :

constructor(

params: NavParams,

private alertCtrl : AlertController)

{

this.exercises = params.data;

}

En fait, les exercices ont été envoyés à cette classe par l'intermédiaire des NavParams depuis l'appelant, en l'occurence la classe :

export class LessonPage

qui une fois la liste des exercices constituées, a appelé la vue des exercices par la commande :

this.navCtrl.push(ExercisesPage, ExercisesToDisplay);

commande qui a donc envoyé le tableau ExercisesToDisplay via les NavParam jusque vers le controleur de la vue d'exercices, un mécanisme typique d'Angular.

#### Fonctionnement de la sélection d'une réponse dans les radio buttons.

Les choix possibles du QCM sont, du point de vue de la structure Objet de la classe Exercise, rangés dans un tableau des "possibleAnswers", ce tableau étant un attribut de l'objet exercice tiré de la table des exercices.

Quand on clique sur le rond d'un radio button, le paramètre answ prend la valeur définie par le paramètre value ="{{answer}}" , soit donc une réponse possible au QCM retiré de la tablde des possibleAnswers.

Cette syntaxe, avec usage des signes {{ .. }} , est typiqued'Angular et connue sur le nom de *data binding* ; elle permet de relier les informations montrées par une vue aux méthodes responsables de leur gestion dans le controleur de la vue.

Quand on clique sur un bouton, le paramètre answ prend donc la valeur correspondant à la possibleAnswer associée au bouton.

Quand l'utilisateur clique sur le bouton de Confirmation, il appelle la méthode

respondToAnswer(exercise,answ) fournie par la classe qui controle cette vue, soit la classe ExercisesPage.

En réponse à cet appel, on lance donc la procédure

respondToAnswer(exercise : object, answer : string ){

var exService = new ExerciseServices(this.alertCtrl,<Exercise> exercise);

exService.manageAnswer(answer);

}

Cette procédure se charge ensuite d'appeler les services associés à la classe ExerciseServices, qui gère tous les services associés à la classe Exercise. Ce sont ces services qui vont gérer l'affichage de la réponse et toute autre action associée à une réponse de l'élève, par exemple l'enregistrement de la réponse pour des statistiques éventuelles, une extension possible du prototype.

Ces services sont exposés ci-dessous en tant qu'exemple de l'injection de dépendance, un processus d'Angular qui rend possible un modèle objet de l'application aux standards de la programmation orienté objet.

## Les mécanismes d'injection de dépendance dans Angular

Ces mécanismes sont un peu déroutants pour un débutant, mais permettent une conception objet "propre" d'une application, avec une réelle indépendance des objets entre eux. Une modification des services associés à une classe ne va pas impacter les classes qui utilisent cette classe et ne va pas impliquer une modification du code associé à ces classes utilisatrices. Il s'agit donc d'un mécanismes fondamental de la conception objet dans un projet utilisant Angular.

On en donne ici un exemple, pour illustrer sa mise en oeuvre dans les services associés aux exercices .

On a rendu autonome la gestion des exercices, en particuler la réponse à des questions. Cette fonctionalité n'est pas gérée au niveau du controleur de la vue, mais au niveau d'une classe indépendante qui se charge uniquement de l'ensemble des services associés aux exercices.

Ainsi, on met à disposition des autres classes exploitant des exercices un module de services génériques associés aux exercices :

export class ExerciseServices

Ce module fournit les services :

manageAnswer(answer)

Pour gérer la réponse à une question.

isAnswerCorrect

Vérifie la conformité de la réponse de l'élève à la réponse attendue.

displayAnswer

Envoie une alerte indiquant si la réponse est juste ou fausse et l'explication associée.

displayNoAnswer

Envoie une alerte si l'élève à cliqué sur Confirmer par erreur avant choisir une réponse.

Cette classe des ExerciseServices est à disposition de toute autre classe qui aurait, par exemple, à gérer des réponses à un questionnaire. Pour l'utiliser, il suffit de lui fournir le jeu d'informations demandées par son constructeur dans un objet de la classe Exercise :

export class ExerciseServices {

lessonId : number;

exerciseName: string;

question: string;

possibleAnswers: [(string)];

rightAnswer: string;

explanation: string;

constructor (

private alertCtrl: AlertController,

private exPattern: Exercise

)

{

this.lessonId = exPattern.lessonId;

this.exerciseName = exPattern.exerciseName;

this.question = exPattern.question;

this.possibleAnswers = exPattern.possibleAnswers;

this.rightAnswer = exPattern.rightAnswer;

this.explanation= exPattern.explanation;

}

Un utilisateur pourra "consommer" ce service de la façon suivante : dès lors qu'il aura reçu ou créé et renseigné un objet "exercise" de la classe Exercise et reçu une réponse "answer" à une des questions de l'exercice, il pourra créer un service associé à son objet exercise, puis invoquer les méthodes associées à ce service.

var exService = new ExerciseServices(this.alertCtrl,<Exercise> exercise);

exService.manageAnswer(answer);

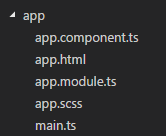
La pratique de l'injection de dépendance implique l'observation rigoureuse de règles propres à Angular :

1 - le service doit être déclaré comme injectable avec le "décorateur" correspondant de Angular:

import { Injectable } from '@angular/core';

@Injectable()

2 - il doit être déclaré comme provider dans les providers de app.module.ts



providers: [

...

ExerciseServices,

...

]

Ensuite le service est injectable soit à la suite d'une création d'une nouvelle instance par la commande

var exService = new ExerciseServices ...

soit par injection dans le constructeur de la classe qui va consommer le service

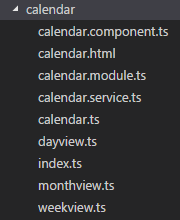
constructor(

private exServ : ExerciseServices

## Gestion du calendrier

Le calendrier est réalisé essentiellement à l'aide d'un plugin récupéré sur Internet puis customisé.

Il est géré par les modules du répertoire calendar.



La partie active du calendrier est gérée dans le module calendar.component.ts avec la classe CalendarPage qui contient les méthodes permettant de gérer les événements à inclure dans le calendrier et la réponse à des actions de l'utilisateur.

Le principe global de la gestion du calendrier est le suivant :

1 ) L'enseignant défini ses dates dans un fichier calendar au format JSON qu'il va placer sur la zone storage de firebase **sans oublier de recopier le lien d'accès dans le champ** :



Ainsi, l'application peut copier le fichier depuis la zone Storage de Firebase vers le téléphone portable lors de l'initialisation de l'application, ce qui sera effectué par la commande :

loadFileFromCloud(ExternalFilesConfig.calendarFileName, MyApp.calendarFileURL);

exécutée depuis la classe DataSercice du module data.service.ts .



le transfert est effectué par la commande :

fileTransfer.download(url, f, true)

à laquelle il suffit de donner l'URL de l'origine (url), le chemin d'accès complet au fichier de destination, ici f, avec l'autorisation de remplacer ou non un fichier existant, ici mise à true pour écraser le calendrier précédent.

A l'initialisation de l'application, le transfert des données depuis le fichier Calendar JSON vers la variable globale de l'application MyApp.Calendar est effectuée par la séquence suivante.

createCalendarEvents()

constitue le calendrier interne à l'application par la méthode

buildArrayFromJSONFile(onDeviceDirectory,fileName,target)

Cette méthode va lire le contenu du fichier JSON spécifié par les paramètres onDeviceDirectory et fileName à l'aide de la commande

file.readAsText(filePath, fileName)

qui va charger vers l'application le contenu textuel du fichier JSON.

Le résultat de la lecture est géré dans une promesse Angular dont la structure complète est la suivante :

this.file.readAsText(filePath, fileName)

.then(

(res) => {

this.populateMyAppFiles (target, JSON.parse(res) ) // Will send data into the array of objects designed by target

},

(err) => {console.log ('reading file = ' + filePath + '/' + fileName + ' ; error = ' + JSON.stringify(err))}

);

Le résultat de la promesse, res, est transformé en Objet par la commande :

JSON.parse(res)

Cet objet est alors recopié dans la variable globale spécifiée par le paramètre target.

Dans la méthode **populateMyAppFiles** :

populateMyAppFiles (target, obj)

va recopier l'objet obj dans la variable globale désignée par le paramètre target.

La section de cette méthode gérant spécifiquement le calendrier est la suivante :

if (target === 'calendar') { MyApp.calendar = obj}

A partir de ce moment, l'application dispose, dans la variable globale MyApp.Calendar, de l'ensemble des rendez-vous de l'enseignant.

Pour générer le calendrier spécifique à un élève, la méthode

createCalendarEvents()

que l'on trouve dans le module



fait le tri des événements qui ne concernent que l'élève dont le nom est contenu dans la variable globale :

MyApp.studentName

le tri s'effectue dans la boucle suivante sur le contenu de MyApp.calendar.

for (var i = 0; i < MyApp.calendar.length; i++) {

if( MyApp.studentIsLogged && MyApp.calendar[i].student === MyApp.studentName ) {

eventType = MyApp.calendar[i].eventType;

startYear = MyApp.calendar[i].year;

startMonth = MyApp.calendar[i].month;

startDay = MyApp.calendar[i].day;

startHour = MyApp.calendar[i].hour;

endDay = startDay;

startTime;

endTime;

startTime = new Date(startYear , (startMonth-1), startDay, startHour );

endTime = new Date(startYear , (startMonth-1), startDay, (startHour+1) ) ;

events.push({

title: 'English Course',

startTime: startTime,

endTime: endTime,

allDay: false

})

}

La commande events.push remplit la table des événements gérées par le module calendar.

La partie interface homme/machine du calendrier est gérée essentiellement par la directive calendar :

<calendar [eventSource]="eventSource"

[calendarMode]="calendar.mode"

[currentDate]="calendar.currentDate"

(onCurrentDateChanged)="onCurrentDateChanged($event)"

(onEventSelected)="onEventSelected($event)"

(onTitleChanged)="onViewTitleChanged($event)"

(onTimeSelected)="onTimeSelected($event)"

step="25">

</calendar>

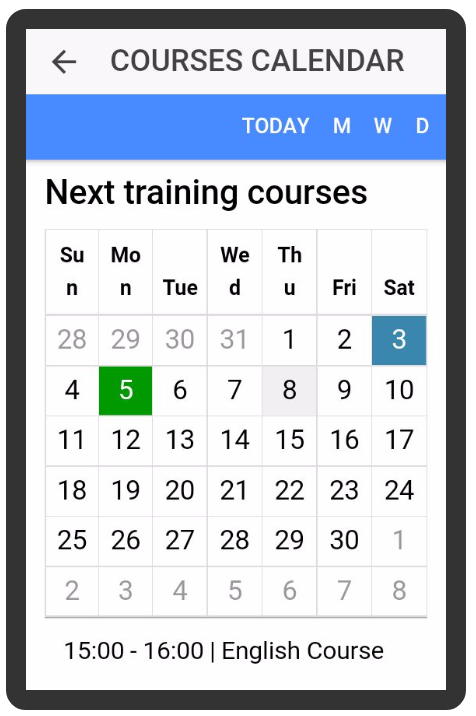
La réaction à des actions de l'utilisateur est gérée par la méthode

onEventSelected(event)

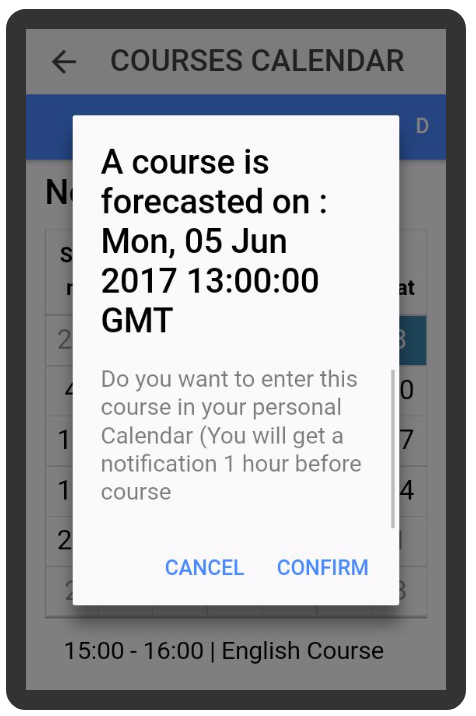
du module



Cette méthode permet de réagir à une pression de l'utilisateur sur l'événement notifié en bas de page, comme ici l'événement : 15:00 - 16:00 : English Course



pour obtenir la réponse :



Si l'élève confirme, la date est entrée dans le calendrier personnel de l'élève, une première notification est envoyée aussitôt sur son portable pour l'informer de la prise en compte du rendez-vous, et une notification devant intervenir une heure avant le rendez-vous est automatiquement générée.

Dans la méthode :

onEventSelected(event)

l'interaction avec le calendrier de l'élève est gérée par la native Calendar de Ionic:

createEventInteractively('Your English Lesson', 'home', 'notes', event.startTime, event.endTime);

(documentation : https://ionicframework.com/docs/native/calendar/)

et les notifications sont gérées parla native LocalNotifications, par exemple pour une notification différée, une heure avant le cours :

localNotifications.schedule({

text: 'Your English Course starts in one hour',

at: new Date( event.startTime - 3600),

led: 'FF0000',

sound: (MyApp.platformIs === 'android')? 'file://sound.mp3': 'file://beep.caf',

});

(documentation : https://ionicframework.com/docs/native/local-notifications/)

## Gestion de l'apparition/disparition des pages et et de la navigation

L'appel d'une vue depuis une méthode est géré, dans Angular, par le module NavController.

Ainsi, par exemple, le controle de lapremière vue proposée à l'utilisateur, la vue home, gère l'ouverture des différentes vues de l'application :

1 - en important NavController

import { NavController, NavParams } from 'ionic-angular';

2 - en appelant les différentes vues par la méthode push fournie par la classe NavController.

showLessons() {this.navCtrl.push(LessonsPage)}

showCoursesCalendar() {this.navCtrl.push(CalendarPage)}

showContactForm(){this.navCtrl.push(ContactPage)}

showHelp(){this.navCtrl.push(Help)}

showResults() {this.navCtrl.push(ResultsPage)}

studentLogin(){this.login.studentLogin(this.navCtrl)}

La dernière commande constitue un exemple intéressant d'une navigation en cascade :

on fait appel au service StudentLogin défini par ailleurs. Ce service est d'abord injecté dans la classe HomePage :

export class HomePage {

...

constructor(

private navCtrl : NavController,

private navParams : NavParams,

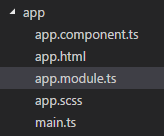
private login : StudentLogin

puis le service studentLogin fourni par la classe StudentLogin est appelé avec la syntaxe

studentLogin(){this.login.studentLogin(this.navCtrl)}

et le passage du paramètre navCtrl permet la continuité de la navigation entre les pages.

Toute page doit être préalablement insérée dans lesdeclarations et les entryComponents de app.module.ts



declarations: [

MyApp,

HomePage,

LessonPage,

LessonsPage,

ExercisesPage,

CalendarPage,

ContactPage,

ResultsPage,

Help

],

entryComponents: [

MyApp,

HomePage,

LessonPage,

LessonsPage,

ExercisesPage,

CalendarPage,

ContactPage,

ResultsPage,

Help

],

## Gestion du transfert de données entre pages

D'une façon générale, dans le paradigme d'Angular, le transfert de données entre pages est géré par le composant NavParams qui constitue une sorte de tampon des données à transférer d'un composant à un autre.

Il faut préalablement importer ce module dans la définition d'une classe si celle-ci à vocation à envoyer des données dans une autre vue ou à en recevoir d'un autre composant.

import { NavController, NavParams } from 'ionic-angular';

ensuite, dans le cas par exemple du transfert de la table des exercices à montrer depuis la classe qui constitue cette table vers la classe qui gère sa présentation à l'écran et les commandes associées à cette vue, la commande suivante assure le transfert des données vers la cible. (Elle appelle la vue du composant ExercisesPage

this.navCtrl.push(ExercisesPage, ExercisesToDisplay);

On récupère les données envoyées de la façon suivante, par exemple, ici, dans la classe ExercisesPage, pour le transfert du contenu d'une leçon par l'intermédiaire des navParams :

constructor(

private navCtrl : NavController,

private navParams: NavParams

)

{ this.lesson = navParams.data }

Les navParams.data ici récupérés correspondent au tableau ExercisesToDisplay qui a été passé "en tant que NavParam" lors de la commande push qui a appelé la vue.

## Gestion du suivi des résultats d'un élève et fonction de "homework"

Il nous semble intéressant de pouvoir suggérer au client quelques extensions à sa demande initiale, en particulier lui montrer qu'avec notre application, un suivi personalisé des résultats des élèves est possible et surtout, qu'il est possible d'envoyer des exercices personalisés pour un élève donné en fonction de ses résultats aux tests antérieurs, ce qui permet un véritable coaching de l'élève.

Cela suppose d'être capable :

1 - de garder une trace des résultats de l'élève aux différents exercices ;

2 - de transmettre ces résultats à l'enseignant ;

3- de permettre à l'enseignant de disposer d'une zone du backend où il peut charger un fichier de travail à effectuer (homework) spécifiquement pour un élève.

### Enregistrement des résultats de l'élève

Une varaible globale a été rajoutée dans le module app.components :

static studentResults : any = new Array;

La classe results.services a été créée pour fournir toutes les fonctions nécessaires à la gestion des résultats de l'élève :

Lorsque l'élève clique sur le bouton de confirmation de sa réponse à un exercice,

dans le fichier = exercisesPage

<button ion-button round outline (click)="respondToAnswer(exercise,answ)">Confirmation</button>

la méthode qui gère la réponse appelle le service qui va prendre en charge la réponse avec , en paramètres, l'exercice auquel l'élève a répondu et la réponse qu'il a donnée.

... class ExercisesPage ...

respondToAnswer(exercise : object, answer : string ){

var exService = new ExerciseServices(this.alertCtrl,<Exercise> exercise);

exService.manageAnswer(answer);

la méthode manageAnswer va prendre en charge la mémorisation de la réponse en appelant la méthode logResult

... class ExerciseServices...

manageAnswer(answer){

var ok : boolean = this.isAnswerCorrect(answer);

this.displayAnswer(ok);

this.logResult(answer,ok);

}

la méthode logResult créé un objet qui contient les paramètres de l'exercice ainsi que la réponse de l'élève ; c'est cet objet de la classe ExerciseResults qui va être inséré dans la variable globale qui contient les résultats. La méthode logResult appelle à son tour la méthode qui va insérer le résultat dans la variable globale qui contient le tableau des résultats:

... class ExerciseServices...

logResult(answer : string , ok:boolean){

// This will add result to the App static object holding all exercises results

var result = new ExerciseResults (this.exPattern, answer, ok);

result.insertIntoStudentResults();

}

Lors de sa création, une instance de la classe ExerciseServices reçoit en argument un exercice qu'elle recopie dans cette nouvelle instance :

constructor (

private alertCtrl: AlertController,

private exPattern: Exercise

)

{

this.lessonId = exPattern.lessonId;

this.exerciseName = exPattern.exerciseName;

this.question = exPattern.question;

this.possibleAnswers = exPattern.possibleAnswers;

this.rightAnswer = exPattern.rightAnswer;

this.explanation= exPattern.explanation;

}

C'est ensuite un ExerciseResults qui est inséré dans la variable globale MyApp.studentResults.

class ExerciseResults...

insertIntoStudentResults() {

MyApp.studentResults.push(this);

}

Une fois le tableau des StudentResults constitué au fil des exercises, il devient possible de traiter les résultats de l'élève globalement. La classe ExerciseResults fournit pour cela une méthode qui va écrire le compte tendu (log) du travail de l'élève, la méthode :

writeResultsAsPlainTextToBeFiled()

une méthode qui gère le stockage du fichier des résultats sur le téléphone de l'élève pour toute exploitation ultérieure éventuelle :

storeResultsOnDevice

et surtout la méthode qui permet de transférer le fichier des résultats vers le backend du professeur :

uploadResultsOnFirebase

**Remarque : reset du fichier des résultats en cas de changement d'élève**

Au cas où l'élève se délogue et qu'un autre élève prend sa suite, par exemple dans le contexte d'une utilisation familiale de l'application, il est nécessaire d'effacer les résultats de l'élève précédent de la mémoire des résultats, ce qui est effectué par la méthode :

... class ResultServices ...

resetStudentResults(){

MyApp.studentResults = new Array();

}

### Transfert du fichier vers le backend du professeur

Le transfert vers Firebase est assuré par la méthode **putString** qui permet d'envoyer un fichier sous la forme d'une chaîne de caractères.

... class ResultServices ...

uploadResultsOnFirebase(txt :string, fileName : string){

var storageRef = firebase.storage().ref();

if(!MyApp.studentIsLogged){return}

var fileRef = storageRef.child(ExternalFilesConfig.resultsFirebaseDirectory + MyApp.studentName + '/' + fileName);

setTimeout(

() => {

fileRef.putString (txt)

.then(

(snapshot) => {console.log('File Uploaded successfully on Firebase ! ' + snapshot) } ,

(err) => {console.log ('File transfer error = ' + err )}

)

.catch( e => {if(Debug) {console.log('Error transfering file', e )} } )

},1000)

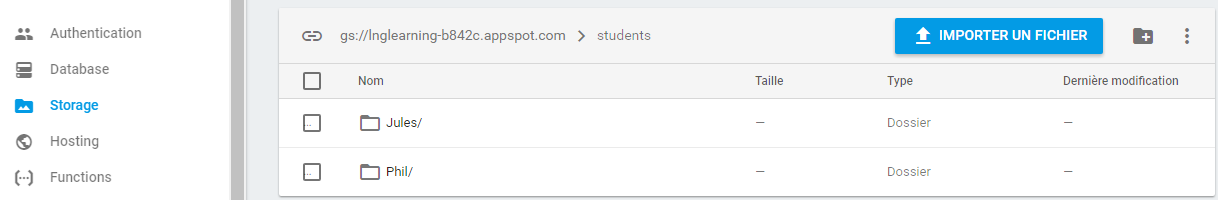
}

La méthode putString est fournie par le module firebase, importé dans la classe à l'aide de la commande :

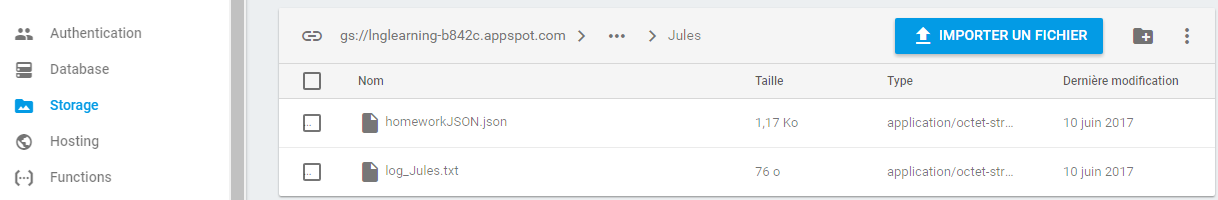
import \* as firebase from "firebase"

### Gestion d'un travail personalisé donné par le professeur à l'élève

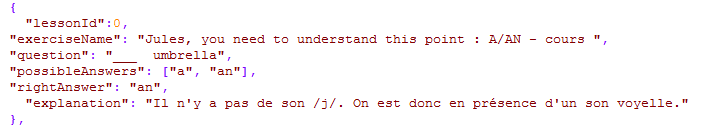
Dans Firebase, un répertoire des étudiants contient les sous-répertoires propres à chaque étudiant :



C'est dans ces répertoires que l'enseignant peut placer un fichier de travail spécifique à un élève :



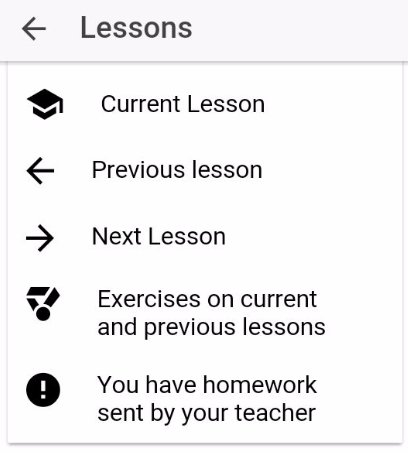
C'est aussi là qu'il peut retrouver le fichier des résultats obtenus par l'élève à sa dernière série d'exercices.

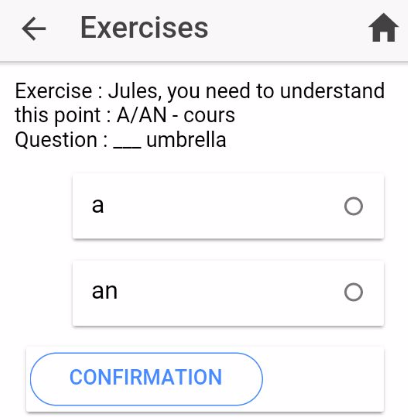
Si le professeur souhaite constituer un fichier d'exercises spécifiques à un élève, il édite ce fichier en format JSON : 

puis il le charge sur Firebase et renseigne ensuite les champs de l'élève dans la Database :

****

Quand il met homeWork à true, l'élève est informé qu'il a un travail spécifique à effectuer :





### Gestion de la présence du homework dans l'application

Affichage conditionnel dans la vue des leçons :

dans le fichier lessons.html, l'affichage du message homework et du bouton correspondant est géré par la directive \*ngIf :

<button ion-item \*ngIf="homework">

<ion-icon name="md-alert" item-left (click)="manageHomework()"></ion-icon>

You have homework <br/> sent by your teacher

</button>

C'est donc l'attribut homework de la classe lessonsPage qui va déterminer cet affichage.

La classe lessonsPage fournit la méthode :

... class LessonsPage ...

isThereHomeworkToBeDone(){

if (!MyApp.studentIsLogged) {

this.homework = false;

return

}

for ( var Std in MyApp.students) {

if(MyApp.students[Std].Name === MyApp.studentName && MyApp.students[Std].homeWork) {

this.homework = true;

return

}

}

}

Cette méthode est appelée par le constructeur de LessonsPage, et l'information sur la présence de homework est gérée par l'interface avec Firebase.

La méthode manageHomework est appelée lorsque l'élève clique sur le bouton relatif au homework :

manageHomework(){

var root = database().ref();

root.on('value', function(snap) {

var dataFirebase:any = snap.val();

MyApp.students = dataFirebase['Students'];

for (var Std in MyApp.students){

if(MyApp.students[Std].Name === MyApp.studentName && MyApp.students[Std].homeWork){

MyApp.homeworkFileURL = MyApp.students[Std].homeWorkFile

}

}

});

this.dataServ.loadFileFromCloud(ExternalFilesConfig.homeworkFileName, MyApp.homeworkFileURL);

setTimeout(

() => {

var platformPath : string = this.dataServ.getPlatformPath();

var fullPath = platformPath + ExternalFilesConfig.dir;

this.dataServ.buildArrayFromJSONFile(fullPath, ExternalFilesConfig.homeworkFileName, 'homework');

},

1000 );

setTimeout(

() => { this.displayHomework() },

2000 );

}

Cette méthode se charge de lire le fichier du homework par un appel à loadFileFromCloud, qui télécharge le fichier du homework depuis le Cloud sur le téléphone de l'élève, puis buildArrayFromJSONFile qui va lire le contenu du fichier transféré et le met à disposition de displayHomework qui visualise le travail à effectuer.

# Conclusions

Même s'il s'agit encore d'un prototype, les fonctionnalités illustrées sont assez réalistes pour donner une bonne idée de ce que serait une application opérationnelle. En l'état, elle peut être déjà utilisée par un enseignant unique.

Sa principale faiblesse est le caractère minimal du backend de Firebase, pour une réalisation en vraie grandeur, il faudrait concevoir un backend plus facile d'utilisation pour un utilisateur non informaticien.

Aucune évaluation de montée en charge n'a été effectuée, et il n'est pas possible, à ce stade, de dire au client à quel moment il faudrait passer à une solution payante sur Firebase, si l'utilisation de Firebase était retenue pour l'application finale.

Le développement de cette application a demandé approximativement deux mois de travail pour un débutant n'ayant auparavent pas même réalisé un projet en javascript et pas encore familier avec les technologies du web et les développements en langage objet. On ne peut donc qu'être séduit par la simplicité et l'efficacité du framework IONIC 2, en dépit de ses instabilités, de ses évolutions rapides, et de ses bugs. Les versions Ionic 2 et 3, utilisant Angular 2, sont beaucoup plus simples d'utilisation et surtout conceptuellement plus faciles à comprendre que la version 1 avec Angular JS. On se rapproche beaucoup plus des stadards de la programmation objet.

Les instabilités ou bugs du framework font l'objet d'un travail rapide et efficace de la part d'une communauté de développeurs très réactifs et visiblement très motivés pour assurer le succès du framework.

Enfin, le framework dispose d'une importante et dynamique communauté d'utilisateurs qui proposent des tutoriels et des formations de très bonne qualité permettant un apprentissage rapide.

(On a tout particulièrement exploité et apprécié les formations et tutoriels de Maximilian Schwarzmüller sur Angular et Ionic et de Vasco | Angular University sur le site Udemy)

L'utilisation de Firebase s'est avéré particulièrement pertinent pour réaliser des fonctionalités minimales de backend sans effort de programmation spécifique.

Il serait absolument falacieux de prétendre que l'on peut développer un projet en Ionic avec seulement quelques bases minimales en Angular. On peut sans doute réussir à faire une application qui fonctionne tant bien que mal, avec des fonctionnalités limitées, mais l'optimisation du code, la qualité de la conception objet, de la modularité de l'application, dépendent considérablement de la maîtrise d'Angular, en particulier des mécanismes d'injection de dépendance qui demandent un apprentissage important et ne sont pas particulièrement simples à maîtriser dans la pratique. A l'opposé, la partie strictement liée à Ionic est d'une simplicité étonnante, tant les interactions avec le téléphone sont bien encapsulées par les natives de Ionic2.

(Le prototype, dans sa forme présente ne peut pas être considéré comme entièrement satisfaisant : par exemple le choix de placer le process d'initialisation d'une classe dans un constructeur n'est pas conseillé dans les bonnes pratiques d'Angular, et le lancement de l'apllication reste lent).

Ceci-dit, la découverte d'Angular 2, une fois passés les premiers moments de découvertes et de déconvenues, est très enthousiasmante et donne envie d'en faire son cadre de travail pour de futurs projets sur le web, dès lors qu'il y a d'importants traitement d'informations à effectuer.

# Glossaire

**API**

Dans les paradigmes Ionic, les API désignent des composants HTML spécifiques que Ionic offre pour réaliser des interfaces Homme/Machine adaptées à l'écran d'un téléphone portable.

**Application**

L'ensemble des programmes qui mettent en oeuvre une famille de fonctionalités informatiques sur un téléphone portable sous la forme d'un programme exécutable unique pour l'utilisateur.

**Backend**

Ce terme désigne l'ensemble utilitaires informatiques à distance de l'utilisateur qui lui mettent des données à disposition via Internet.

**Cloud**

Lieu virtuel de stockage de données sur Internet. Le Cloud permet à l'utilisateur de mettre ses fichiers de données à disposition d'applications qui se connectent au serveur de données surlequel ces données sont déposées.

**Composant**

Dans le paradigme d'Angular, un composant peut être défini comme un constituant d'Interface Homme/Machine (IHM) avec une vue à insérer dans l'IHM de l'application, et une classe associée à cette vue. Toute classe d'Angular n'est pas nécessairement associée à une vue, elle peut aussi avoir un rôle fonctionnel sans lien avec les IHM. Un composant peut donc être considéré comme une portion d'interface Homme/Machine à intéger dans les vues gérées par l'application, avec une vue constituée de directives html (+éventuellement des "components Ionic") et un ensemble de méthodes qui gèrent les informations présentées par la vue et les actions autorisées à l'utilisateur à partir de cette vue.

**Directive**

Commande que l'on peut insérer à l'intérieur d'une balise HTML pour réaliser des actions programmatiques qui ne sont pas a priori gérées par le langage HTML. Par exemple :

<div \*ngFor="let exercise of exercises">

permet de balayer la liste des objets exercice contenus dans la table exercises et d'afficher dans la vue HTML correspondante des informations spécifiques à chaque objet de la classe exercice.

**Firebase**

Ensemble de services proposés par Google pour héberger des données et réaliser sans développement informatique des fonctionnalités de type backend vis à vis d'un site Internet ou d'une application mobile.

**Framework**

Cadre de travail pour le développement d'un logiciel.

**IDE**

Plateforme de travail qui pemet d'éditer, corriger, tester une application de façon à gagner en productivité par rapport à des éditeurs de texte non orientés développement informatique.

**IHM**

Interface Homme/Machine

**Ionic 2**

Environnement de développement ("Framework" en anglais) d'applications dites "crossplatform", c'est à dire développées indépendamment du système d'exploitation de la plateforme cible de l'application. Ionic gère la couche IHM de l'application et rend transparente la couche de gestion des interactions avec le matériel constitutif de la plateforme d'utilisation de l'application.

**Native**

Le terme désigne, dans les paradigmes de Ionic2, toutes les fonctions mises à disposition pour communiquer avec le téléphone et utiliser l'ensemble des fonctionnalités spécifiques à un appareil mobile.