Audit YouScribe

Ce rapport contient une analyse détaillée du cahier des charges de l'application YouScribe ainsi que l'estimation de temps et de complexité des fonctionnalités demandées.

Contexte

YouScribe permet de lire partout, et se présente comme une bibliothèque numérique de poche. Des applications iOS et Android existent, développées en C# avec Xamarin.

YouScribe a contacté Wyatt Studio pour son expertise en Flutter afin d'être accompagné dans la migration de leur base de code vers Flutter.

Analyse du Cahier des Charges

Step 1

Sign in with Apple

Package	Licence
https://pub.dev/packages/sign_in_with_apple	MIT
https://git.wyatt-studio.fr/Wyatt-FOSS/wyatt-packages/src/branch/master/packages/wyatt_authentication_bloc	GPL-3.0

⚠ L'intégration de Sign in avec Apple ne se limite pas à l'ajout de ce plugin à votre pubspec.yaml et à l'utilisation des fonctions de réception d'informations d'identification qu'il propose.

Il faut également configurer le back-end correctement.

Une fois que vous avez reçu les informations d'identification, elles doivent être vérifiées auprès des serveurs d'Apple, puis une nouvelle session doit être dérivée de ces informations dans votre système.

Votre serveur doit ensuite vérifier régulièrement la session auprès d'Apple via un refresh token obtenu lors de la validation initiale. Le serveur doit également révoquer la session dans votre système si l'autorisation a été retirée du côté d'Apple.

Wyatt Authentication permet d'encapsuler toute la logique d'authentification utilisateur en suivant l'architecture Wyatt et les blocs. (Voir partie suivante sur l'Authentification)

> Si une application propose l'authentification par Google, alors elle doit proposer l'authentification par Apple sur iPhone au risque de se faire refuser à la publication sur

> l'AppStore.

Complexité: faible

Temps de développement: 2j

Architecture

Chez Wyatt Studio nous conseillons la Clean Architecture couplée au state management en BLoC pattern et à l'injection de dépendances GetIt.

Cette architecture vise à créer des systèmes logiciels hautement maintenables, indépendants de la technologie et faciles à comprendre. Son objectif principal est de séparer les préoccupations en couches clairement définies et de maintenir les dépendances dans une seule direction, ce qui permet une plus grande flexibilité et facilité de maintenance.

En Flutter nous proposons généralement les couches suivantes:

1. **Couche de Données (Data)** :
   - Cette couche est responsable de l'accès aux sources de données, telles que les bases de données locales, les appels réseau ou les fichiers locaux.
   - Dans le cas de Flutter, cela pourrait signifier l'utilisation de packages pour gérer les requêtes réseau (comme Dio ou HTTP), les bases de données locales (comme SQLite ou Isaar), ou les sharedPreferences pour stocker des données locales.
   - La couche de données fournit une interface (contrat) définissant les opérations possibles, et qui facilitera l'injection de dépendances.
2. **Couche de Domaine (Domain)** :
   - Cette couche contient la logique métier centrale de l'application, indépendamment de tout détail d'implémentation technique.
   - Elle définit les entités métier, les règles de validation et de traitement des données, ainsi que les cas d'utilisation de l'application.
   - Les entités métier peuvent être des classes de données simples qui représentent des objets métier, telles que des utilisateurs, des livres, etc.
   - Les cas d'utilisation encapsulent la logique métier et interagissent avec la couche de données pour obtenir ou mettre à jour les données nécessaires.
3. **Couche de Présentation (Presentation)** :
   - Cette couche gère l'interface utilisateur de l'application.
   - Elle est responsable de la création et de la gestion de l'interface utilisateur en fonction des données fournies par la couche de domaine.
     - Cette UI réactive sera gérée par les **BLoC/Cubits**
   - Elle communique avec la couche de domaine pour obtenir les données nécessaires, puis affiche ces données à l'utilisateur.
   - Les widgets d'interface utilisateur peuvent également envoyer des actions (telles que des clics sur un bouton) à la couche de présentation, qui les transmet à la couche de domaine pour effectuer des opérations métier.

Chaque élément de chaque couche est géré via injection de dépendances, ce qui permet de centraliser l'instanciation de ces éléments et de switcher facilement de mode.

Exemple:

- BookmarksDataSource (abstract)
  - BookmarksDataSourceImpl → implémentation réelle utilisant SQLite
  - BookmarksDataSourceMock → implémentation mockée

Dans le fichier d'injection de dépendance, on choisit d'injecter BookmarksDataSource qui sera utilisé tel quel dans toute l'application (peu importe de l'implémentation derrière) et l'on peut facilement switcher entre implémentation réelle / mockée à la compilation par exemple.

Aussi la (dé)sérialisation des modèles peut être générée automatiquement.

Package	**Licence**
https://pub.dev/packages/flutter_bloc	**MIT**
https://pub.dev/packages/get_it	MIT
https://pub.dev/packages/json_serializable	BSD-3-Clause
https://git.wyatt-studio.fr/Wyatt-FOSS/wyatt-packages/src/branch/master/packages/wyatt_architecture	GPL-3.0

> L'application YouScribe contient une 50aine d'entités (dé)sérialisables.
> Aussi, une documentation et la spécification de l'API contenant les entrées/sorties de celle ci peuvent grandement accélérer le processus et faciliter les tests !

Complexité: faible

Temps de développement: 4j

Flavors

Une des problématiques de YouScribe est de pouvoir développer une application en marque blanche.

Flutter propose nativement des outils prévus à cet effet:
https://docs.flutter.dev/deployment/flavors

Les "flavors" permettent de **configurer** l'application à la compilation.

Ensuite il existe aussi les targets: https://docs.flutter.dev/testing/build-modes

En utilisant `--target` il est possible de changer le point d'entrée de l'application et ainsi la démarrer dans un certain mode.

Package	Licence
https://pub.dev/packages/flutter_flavorizr	MIT

Flavorizr permet d'automatiser et centraliser les configurations de flavors dans un même fichier.

> Les 2 options peuvent (et sont souvent) être utilisés en même temps.
> → Les flavors vont ainsi permettre de configurer différents bundleId, différents fichiers manifest pour android, différentes signatures de release ou, encore, différentes environnement Firebase.
> → La target va, quant à elle, permettre de lancer l'application dans différents mode, pour injecter les data sources réelles, ou les data sources mockées par exemple.

À cela se rajoute la possibilité de passer des variable à la compilation avec `--dart-define` ou `--dart-define-from-file` est ainsi gagner en granularité de personnalisation.

Complexité: faible

Temps de développement: 1j (pour configurer YouScribe et Maktabati)

Local Datasource

Certains éléments ont besoin d'être sauvegardés en local pour le fonctionnement de l'application (les marques page par exemple).

Différentes options s'offrent à vous,

Package	Licence
https://pub.dev/packages/sqflite	BSD-2-Clause
https://pub.dev/packages/hive	Apache-2.0

> → Le gros avantage de **Hive** c'est que la sérialisation des modèles implémentée précédemment fonctionnera out of the box, il est ainsi possible d'implémenter facilement un système de cache et d'y stocker les mêmes modèles que ceux de l'API.
> → Aussi les benchmarks de Hive montre de bien meilleures performances que sqflite

Complexité: dépendant de la solution choisie:

- sqflite: modérée

- hive: faible

Temps de développement: dépendant de la solution choisie:

- implémentation sqflite: 4 jours
- implémentation Hive: 2 jours

Sending/Receiving information at startup

Au démarrage de l'application une synchronisation avec le serveur est effectuée.

Cette tâche est déléguée à un repository (couche domaine) connecté aux datasources locales et distantes.

Pour le client HTTP

Package	Licence
https://pub.dev/packages/dio	MIT

> Une datasource générique permettant de faire des appels à l'API de façon authentifiée (avec un access token) est à prévoir.

Complexité: modérée

Temps de développement: 2 jours

Aussi, si la synchronisation doit être faite, de manière asynchrone et totalement autonome (même si l'utilisateur quitte l'application directement après l'avoir ouverte) il peut être judicieux d'utiliser un outil tel que workmanager afin de planifier une tâche en arrière plan

Package	Licence
https://pub.dev/packages/workmanager	MIT

Complexité: élevée

Temps de développement: 3 jours

Robust analytics/logging system

Un système de remontée d'évènements est à prévoir dans l'application.

Actuellement Firebase Analytics est utilisé, et ce service peut également être intégré à l'application Flutter.

Package	Licence
https://pub.dev/packages/firebase_core	BSD-3-Clause
https://pub.dev/packages/firebase_analytics	BSD-3-Clause

Aussi, la flutterfire_cli permet de mettre en place et connecter facilement son application à son environnement Firebase.

> Les **flavors** peuvent être utilisées pour connecter différentes versions de l'application à différents environnements Firebase (liés à l'application par son applicationId ou bundleId)

Pour logguer les évènements dans la console en local (sans passer par un `print` ou un `debugPrint`) le mieux est d'utiliser un logger.

Package	Licence
https://pub.dev/packages/logging	BSD-3-Clause

Le niveau de logging peut être modifié en fonction de la flavor, de la target ou du mode d'exécution (debug, ou release)

Complexité: faible

Temps de développement: 1 jour

Package	Licence
https://pub.dev/packages/sentry_flutter	MIT
https://pub.dev/packages/firebase_crashlytics	BSD-3-Clause

Il est aussi possible d'intégrer un outil de signalement d'erreur.

→ Sentry, solution payante clé en main

→ Crashlytics, compatible avec Analytics, et retourne les erreurs avec un contexte lié aux Analytics. Gratuit.

UIKit

Afin de styliser l'application, un package (ou plusieurs packages) d'UI Kit peuvent être développés afin de respecter les règles d'UI des marques.

Package	Licence
https://git.wyatt-studio.fr/Wyatt-FOSS/wyatt-packages/src/branch/master/packages/wyatt_ui_components	GPL-3.0

Wyatt UI Components permet de développer rapidement des UI Kits basés sur le concept de components. Un exemple d'implémentation est le Wyatt UIKit.

Afin d'optimiser l'arbre de widgets, ce plugin permet d'inscrire dans un registre les différentes instances de composants et des les utiliser à différents endroits dans l'application.

Cet UI Kit peut également contenir les couleurs, espacements, constantes relatives à l'UI, les typographies, les assets globales (AppIcon, icons, font...) de l'application.

L'application en marque blanche peut ainsi injecter un ui kit différent en fonction de la target. Les implémentations d'UI Kit suivants un contrat abstrait l'app en marque blanche pourra appeler un Bouton sans se soucier de son apparence.

Package	Licence
https://pub.dev/packages/flutter_gen	MIT

FlutterGen permet de générer des assets type safe pour son application afin de ne pas avoir à utiliser le chemin d'accès en dur dans son code, et passer de cette implémentation dangereuse:

```
Widget build(BuildContext context) {
  return Image.asset('assets/images/profile.jpeg');
}
```

à cette implémentation:

```
Widget build(BuildContext context) {
  return Assets.images.profile.image();
}
```

> Le site : https://www.fluttericon.com/ permet de générer des fonts utilisables dans Flutter à partir de SVG afin de générer des collections d'icons utilisables dans Flutter.
> Sinon, https://pub.dev/packages?q=icons permet de trouver dans les packages exposants des icons/des collections d'icons, déjà faites.

Complexité: modérée

Temps de développement: 8 jours

Step 2

HTML Parsing

La lib officielle Dart html permet de parser du html. À partir des descriptions il sera ainsi possible de définir un mapping balise html → widget Flutter.

Package	Licence
https://pub.dev/packages/html	unknown

Complexité: élevée

Temps de développement: 5 jours

Read progression algorithm

Un algorithme de calcul de progression de lecture du contenu doit être implémenté

→ Code C# à adapter, et à retrouver dans `ReaderStatService.cs`

> Il est possible qu'il faille adapter le lecteur ePub afin de prendre en compte les chapitres.

Complexité: faible

Temps de développement: 2 jours

Infinite Scroll

Dans les résultats de recherche, par exemple, un scroll infini est nécessaire. En paginant les résultats et les chargeant par étape à la fin du scroll d'une `ListView` il est possible d'obtenir le résultat souhaité.

Possibilité d'utiliser un package ou une **implémentation maison**

Package	Licence
https://pub.dev/packages/infinite_scroll_pagination	MIT

> L'API doit retourner des résultats paginés.

Complexité: faible

Temps de développement: 1 jour

Step 3

File System Datasource

Afin de télécharger les produits et de les gérer en local (déchiffrer, ouvrir un epub etc..) il est nécessaire de pouvoir travailler dans le file system du smartphone.

Package	Licence
https://pub.dev/packages/path_provider	BSD-3-Clause
https://pub.dev/packages/path	BSD-3-Clause

Path Provider donne accès aux documents de l'application afin d'y stocker le contenu téléchargé. Path quant à lui permet la manipulation des chemins.

Aussi, avec la lib standard **dart:io** il est possible de manipuler les fichiers dans l'espace des documents attribué à l'application.

Complexité: faible

Temps de développement: 3 jours

Datasources and Password encryption

Il faut également implémenter des datasources distantes afin de récupérer les différents contenus chiffrés.

Package	Licence
https://pub.dev/packages/dio	MIT
https://github.com/hugo-pcl/native-crypto-flutter	MIT
https://pub.dev/packages/pointycastle	MIT
https://pub.dev/packages/path_provider	BSD-3-Clause
https://pub.dev/packages/archive	MIT

NativeCrypto est une implémentation native de certaines primitives cryptographiques, et notamment du HMAC SH256 et AES utilisés dans l'application.

PointyCastle propose une implémentation de la crypto à clé publique, et notamment RSA qui est utilisé dans l'application.

Une fois la clé secrète récupérée (le password) il est possible de déchiffrer le contenu d'un produit téléchargé au moment de le décompresser.

Archive permet de décompresser des fichiers zip, un cipher peut être configuré afin de déchiffrer le contenu.

Complexité: modérée

Temps de développement: 8 jours

PDF Reader

Afin de donner accès à un lecteur PDF natif à l'utilisateur, il peut être interessant de continuer à utiliser **PSPDFKIT.**

Le package Flutter existe et donne accès aux différentes implémentations natives.

→ https://pspdfkit.com/pdf-sdk/flutter/

Package	**Licence**
https://pub.dev/packages/pspdfkit_flutter	unknown

Complexité: modérée

Temps de développement: 2 jours

Audio Player

Afin de lire l'audio il faut répartir les tâches, et observer 2 responsabilités distinctes:

- lire un fichier audio dans l'application
- lire un fichier audio en background hors de l'application

Il faut aussi que cela fonctionne en bluetooth.

Package	**Licence**
https://pub.dev/packages/just_audio	Apache-2.0, MIT
https://pub.dev/packages/just_audio_background	MIT

https://pub.dev/packages/audio_service	MIT

Just Audio a la responsabilité de lire de l'audio, tandis que Just Audio Background ou Audio Service ont la responsabilité de lire de l'audio en fond.

> Just Audio Background étant très récent il est préférable de partir sur une solution plus stable et éprouvée comme Audio Service, bien que plus complexe.

Complexité: élevée

Temps de développement: 8 jours

E-Pub Viewer

Afin d'ouvrir des ePub il est possible d'utiliser la package html cité précédemment. Ce qui a été packagé dans le plugin suivant:

Package	**Licence**
https://pub.dev/packages/epub_view	MIT

> Ceci ouvre un widget flutter après avoir parsé le contenu e-pub, et non un viewer natif.
> Si un viewer natif est nécessaire il sera alors nécessaire de faire une **implémentation maison** basée sur des dépendances natives (java/kotlin et swift/objective-c)

Complexité: modérée (élevée si implémentation maison)

Temps de développement: 5 jours (15 jours si implémentation maison)

Step 4

Google Play & AppStore Subscriptions

Il est nécessaire de permettre le paiement des abonnements à YouScribe dans l'application. Pour cela nous pouvons utiliser les implémentations natives des différents magasins d'application de chaque OS.

Un package officiel de Flutter permet d'accéder aux API des magasins d'application dépendant de l'OS. Si l'application s'exécute sur Android le Google Play sera proposé, et sur iOS c'est l'AppStore.

Package	Licence
https://pub.dev/packages/in_app_purchase	BSD-3-Clause

Complexité: faible

Temps de développement: 2 jours

Step 5

Firebase Push Notifications

Afin d'engager au maximum les utilisateurs il est nécessaire de mettre en place un système de notifications push.

Firebase propose ce service, et propose de centraliser l'envoie de notification à Android (FCM) et Apple (APN).

Package	Licence
https://pub.dev/packages/firebase_core	BSD-3-Clause
https://pub.dev/packages/firebase_messaging	BSD-3-Clause
https://git.wyatt-studio.fr/Wyatt-FOSS/wyatt-packages/src/branch/master/packages/wyatt_cloud_messaging_bloc_firebase	GPL-3.0

Wyatt Cloud Messaging Bloc Firebase permet de wrapper les 2 plugins précédents dans l'architecture Wyatt en utilisant le bloc pattern et des usecases.

> Il est nécessaire d'avoir une icon spéciale (transparente/monochrome) pour les notifications Android. À configurer dans le fichier manifest.

Complexité: faible

Temps de développement: 1 jour

Timed tasks

Pour lancer des tâches régulières en arrière plan il est possible d'utiliser `workmanager`.

Package	Licence
https://pub.dev/packages/workmanager	MIT

Complexité: élevée

Temps de développement: 3 jours → déjà implémenté en partie dans "Sending/Receiving information at startup"

Implementing search and classification handling

Il est possible de query l'API pour effectuer une recherche et classifier du contenu.

> Cette fonctionnalité se base énormément sur le back-end, ce n'est pas la responsabilité de l'application Flutter que de chercher/classifier du contenu.

Package	Licence
https://pub.dev/packages/dio	MIT

Complexité: faible

Temps de développement: 1 jour

CI/CD

Chez Wyatt Studio nous utilisons et proposons Fastlane pour gérer nos pipelines de CI/CD.

Package	Licence
https://git.wyatt-studio.fr/Wyatt-FOSS/wyatt-fastlane-plugins	GPL-3.0

Ces scripts/plugins Ruby sont à utiliser avec Fastlane pour déployer rapidement l'application sur l'AppStore et le Play Store.

> De plus une gestion des secrets est à prévoir pour ne pas avoir de **clés secrètes** sur le dépot git !

Complexité: faible

Temps de développement: 4 jours

Step 6

Add play back controls on app's home page

Package	Licence
https://pub.dev/packages/just_audio_background	MIT
https://pub.dev/packages/audio_service	MIT

Comme discuté dans Audio Player, ces 2 plugins permettent de lancer des services d'audio en fond (natifs) et de les afficher sur le lockscreen (ou même CarPlay).

Complexité: élevée

Temps de développement: 8 jours → déjà implémenté en partie dans "Audio Player"

Good practices

Il est intéressant de rapidement mettre en place les bonnes pratiques de Flutter afin de développer sereinement une application maintenable dans le temps.

Router

Nous proposons d'utiliser GoRouter, une solution maintenant recommandée par la team Flutter.

→ https://pub.dev/packages/go_router

- Parsing du chemin d'accès et des paramètres de requête à l'aide d'une syntaxe de modèle (par exemple, "user/:id')
- Affichage de plusieurs écrans pour une destination (sous-itinéraires)
- Prise en charge de la redirection - vous pouvez rediriger l'utilisateur vers une URL différente en fonction de l'état de l'application, par exemple vers une connexion lorsque l'utilisateur n'est pas authentifié.
- Prise en charge de plusieurs navigateurs via ShellRoute - vous pouvez afficher un navigateur interne qui affiche ses propres pages en fonction de la route correspondante. Par exemple, pour afficher une BottomNavigationBar qui reste visible en bas de l'écran.
- Automatiquement compatible avec les deep links.

Ce package est maintenant supporté par la team Flutter.

Gestion d'erreur

Dans l'architecture Wyatt nous gérons toutes les erreurs à l'aide d'un objet `Result` qui permet d'encapsuler une valeur de succès ou une erreur à la manière de la programmation fonctionnelle. (Inspiré du result de la lib standard de Rust https://doc.rust-lang.org/std/result/)

Génération d'assets

En plus d'utiliser *flutter_gen* nous utilisons également:

- https://pub.dev/packages/flutter_native_splash
- https://pub.dev/packages/flutter_launcher_icons

L'un pour générer un splash screen **natif**, et l'autre pour générer les icons d'application dans tous les formats et pour toutes les plateformes à l'aide d'une seule image d'entrée.

Internationalization

Pour internationaliser nos applications nous utilisons la méthode officielle en passant par flutter_localization, intl et les fichiers ARB.

→ https://docs.flutter.dev/ui/accessibility-and-localization/internationalization