IUT Nantes
Pôle Sciences et technologie
Nantes Université

Étudiant(e)s en 3ème année de BUT Informatique
Formation Initiale
Nantes Université - IUT de Joffre - BUT Informatique

R6.Real | Partie Maintenance Applicative

DOCTOwOLIB

Rapport de la SAE 601 - Maintenance Applicative

2024-2025

Encadrement :

- *BIGEON Emmanuel*
- *AIME Xavier*
- *ATTIOGBE Christian*
- *NACHOUKI Gilles*
- *RAOUL Guillaume*
- *ISMAIL Olfa*
- *MOTTU Jean-Marie*

Réalisé par :

- *MANSON Luna*
- *RODRIGUEZ Kévin*
- *OSSELIN Arthur*
- *COSSEC Alban*
- *MOREAU--THOMAS Nils*

Contexte du projet

Nous devions concevoir et développer une application mobile de suivi de traitements médicamenteux et de leurs possibles effets secondaires ou interactions risquées pour les patients. L'application doit permettre une gestion efficace du régime médicamenteux chez le patient, avec déclaration aux autorités des effets indésirables.

Our brief was to design and develop a mobile application for monitoring drug treatments and their possible side effects or risky interactions for patients. The application should enable effective management of the patient's medication regimen, with reporting of undesirable effects to the authorities.
This report is only in french as you could've guessed.

Les mots en **gras** suivis d'un astérisque* sont liées à une annexes
Les mots en **gras et bleus** sont des liens internes ou externes

Équipe antérieur

Lors de la première phase du projet, l'équipe était comprise de :

- *Louis VILLATE*, qui a travaillé sur la partie **script** et **back-end**
- *Lana HEYRENDT*, qui a travaillé principalement sur la partie script, et un peu sur la backend
- *Noémie VERNIER*, qui a travaillé principalement sur la backend, et un peu les script
- *Mathieu LEBRAS*, qui a travaillé sur l'application Android (**front-end**)
- *Romain PERROUQUET*, qui a travaillé sur l'application Android (front-end)
- *Luna MANSON*, qui a travaillé sur l'application Android et sa liaison avec la back-end

État initial

Fonctionnalités

Les fonctionnalitée prévue à l'origine pour l'application sont les suivante :

- Scan des ordonnances et traitement par **OCR**
- Rappel de prise de médicaments récurrent
- Enregistrement des prises des médicaments récurrent et ponctuel
- Minuteur pour la durée minimum entre les prises des médicaments à prise ponctuel
- Mise en évidence des contre-indication d'un médicament avec un déjà prévu, et catalogue des autres contre-indications
- Mise à disposition de donnée pour le médecin sur la prise de traitement, via un système de QR code a scanné pour récupéré un rapport qui ne transite jamais par un serveur
- Journal d'effet secondaire pour chaque médicament

La première phase du projet n'a pas permis la mise en "production" d'aucune des fonctionnalité prévu, même si beaucoup de travail préparatif à été réalisé, comme le traitement des contre-indications, le stockage des prises de médicaments, ainsi que la prise en charge des différents formats d'ordonnances et des formats de fichier possible (image et document pdf)

État Final

Fonctionnalités

Lors de la première phase, l'application n'avait pas beaucoup de fonctionnalités, et l'application lors de cette deuxième phase non plus. Cela s'explique par l'énorme besoin de reprendre le projet sur des bases de codes propres et solides. L'application ne contient donc que :

- Les rappels médicaux
- La gestion des informations utilisateur (tel que les alarmes, le poids, la taille...)

Cependant l'application possède désormais un Footer de navigation qui permettra une implémentation plus facile des futurs pages.

Répartition des tâches - ticket - réunion

Répartition des tâches

L'équipe de maintenance était donc constitué de :

- Arthur OSSELIN, qui a travaillé sur le **back-end**, les tests et la rédaction de la documentation
- Alban COSSEC, a travaillé sur la rédaction et l'application de la convention de nommage sur le **front-end**
- Luna Manson, nous a intégré au projet et a travaillé sur le **Jenkins** et le back-end
- Nils MOREAU–THOMAS, a travaillé sur la **refactorisation** du code, la rédaction de la documentation et l'organisation générale des dépots GitLab
- Kévin RODRIGUEZ, a travaillé sur la **refactorisation** du code, la résolution de bug ainsi que sur de l'amélioration des fonctionnalité de l'application

Gitlab et Gestion de tickets

Gitlab

Gitlab est l'outil que les propriétaires de l'application, lors de la première phase, on utilisé.
Dans le groupe il y avait 4 dépôts :

- application-kotlin, contenant l'application mobile
- script-bdpm-ansm, contenant un script qui créé la base de données de médicaments
- api, qui contient le serveur faisant le lien entre l'application et la base de données
- gitlab-profile, contenant toutes les tâches relatives à l'ensemble du projet, et qui explique également certains points, comme le contenu des autres dépôts.

Il est important de noter que dans les paramètres, pour chaque dépôt, les merge requests ne peuvent être fusionnées uniquement si le **Jenkins** est un succès, cependant faute de pipeline correcte, l'option a été temporairement désactivée.

Gestion de versions

Nous avions définis qu'une version de l'application était de la forme suivante : X.Y.A

- X correspondant à la version "majeure" de l'application, par exemple dès que l'application a suffisamment évolué pour aller à la version majeure suivante
- Y correspondant à la version mineure de l'application, par exemple dès qu'une fonctionnalité est ajouté elle augmente
- A correspondant à la version "insignifiante", par exemple dès qu'un léger changement est fait, elle augmente

Ticket (Issues)

Dans ce projet nous avons beaucoup utilisé le système d'**issues** de gitlab pour structurer notre travail, chaque **ticket** correspond à un problème à résoudre, donc à une tâche à faire. Chaque problème peut être assigné à un niveau d'impact qui va définir à quel point la tâche doit être traitée rapidement ainsi qu'une catégorie de maintenance dont la tâche appartient(bug, amélioration, ...). On essayait le plus possible de donner une estimation du temps que la tâche allait prendre et de quand elle se terminerait.
Les commentaires et descriptions de chaque tickets décrivaient clairement les objectifs du tickets créer.

Demandes de fusion (Merge Requests)

Les **merge requests** ont été créées après que le code sur une **issue** ait été jugé comme une réponse suffisante au problème, la merge request était marquée en tant que brouillon au cas où il y a des choses à ajouter. Une fois que la demande n'est plus en brouillon, s'ensuit alors la **code review**.

Revue de code (Code Review)

Le code était donc étudié en détail par le(s) relecteur(s) et il y avait des commentaires sur les points à corriger/ à questionner. Après le temps des débats et des rectifications, la demande de fusion était approuvée une fois ces problèmes et interrogations commentées étaient résolus.
Puis la fusion a été faite. Le ticket étant associé à cette la merge request est ensuite placé dans les **issues** fermées.

Réunions

Afin de s'organiser et de se retrouver nous avons pratiqué les **méthodes agiles**, en faisant des points hebdomadaires où l'on se retrouvait pour échanger sur les démarches à suivre, ce qui était important à faire rapidement, s'expliquer des méthodes et bonnes pratiques tel que les conventions de nommages, de dialoguer sur ce qui avait été fait entre plusieurs réunions, l'impact potentiel sur le développement d'autres membre du groupe, de la gestion de tickets et .
Par exemple, Alban a effectué la tâche de renommer l'ensemble du code afin de respecter notre convention de nommage, ce qui allait nécessairement impacter les autres développeurs qui s'employaient à ajouter des fonctionnalités.
De plus, nous avons eu beaucoup de briefs sur les bonnes pratiques du gitlab que nous avons développé précédemment.

Voici un exemple de compte-rendu de réunion : **exemple de meeting hebdomadaire.**

Communication

Pour la communication entre les membres de l'équipe, nous avons utilisé un groupe Discord, les tickets des **issues** Gitlab et les réunions mentionnées ci-dessus.

Maintenance Corrective

Correction de bugs TODO: réécrire correctement

Des étudiants de troisième année étaient tâchés de trouver des bugs dans notre application, ce qu'ils ont admirablement fait.
De notre côté, nous avons aussi rédigé quelques rapport de bug.
Voici une liste non exhaustive de certains bugs relevés :

- **Issue 25 : Impossibilité de mettre un poids à virgule**
- **Issue 39 : Bug empêchant le lancement de l'application**
- **Issue 19** : **On ne peut pas sélectionner plusieurs allergies**

Améliorations et Préventions

A l'aide de issues laissés par les autres étudiants et au fur et à mesure de la compréhension du code existant, nous rédigions des suggestions d'amélioration. Voici quelques exemples.

- **Issue 38 : Problème de structuration**
- **Issue 47 : Problème de Maintenabilité Critique : Complexités Cognitives**
- **Issue 11 : README manquant**
- **Issue 1 : Implémenter l'OCR**

Documentation créée ou corrigée

Nous avons, lors de cette maintenance, écrit plusieurs documents afin de faciliter :

- la compréhension, avec un **README.md**
- la sécurité, avec un **LICENSE.md**
- la contribution du projet, avec un **CONTRIBUTING.md**

Vous pouvez consulter les différents documents ici :

Dans le dépôt de l'API

- **README.md**
- **CONTRIBUTING.md**
- **LICENSE.md**

Dans le dépôt de l'application mobile

- **README.md**
- **CONTRIBUTING.md**
- **LICENSE.md**

La licence choisie est la MIT, nous l'avons choisie en nous basant sur **choosealicense.com** avec l'accord du groupe de la première phase (soit les propriétaires originaux du projet).
Cette licence stipule que le projet peut être utilisé à des fins personnelles, commerciales et peut être distribué et modifié. Cela permet à l'application d'être développée par une autre équipe tout en conservant les droits d'auteurs dessus.

Maintenance Préventive

Lors de la première revue de code de chaque membre de notre équipe avec les outils à notre disponibilité sur l'application qui nous avait été donnée de travailler, un constat assez évident nous apparaissait, une grosse partie de maintenance préventive devait être effectuée avant de commencer la partie évolutive de notre maintenance. Elle s'est agencée de cette manière.

SonarQube

SonarQube est un outil qui est utilisé pour inspecter le code source des logiciels et applications en développement et détecter des bugs, vulnérabilités de sécurités, instances de code dupliqué et autres anomalies pouvant nuire à la qualité du code source, et ainsi au fonctionnement de l'application qui en résulte. Elle existe aussi pour les IDE de Jetbrain et permet de voir les problèmes liés au code en direct [Figure 1.1]. Les problèmes sont triés par 4 impacts différents, rouge pour haut impact, orange pour moyen, jaune pour faible et bleu pour info.

Figure 1.1 : Rapport de problèmes SonarQube

Les problèmes de maintenance à l'exception des labellisés info et de **complexité cognitive** ont été réglés ou étaient de faux positifs.

Nombres Magiques (Magic Numbers)

Il y avait beaucoup de **Nombres Magiques** dans le code, ce qui s'explique par le fait qu'il y ait de nombreux composants graphiques dans l'application construit avec des valeurs numériques. On peut noter les valeurs flottantes (`60f`), des valeurs entières (`1500`) ou en pixels indépendants de la densité (`100.dp`). Cela valait aussi pour les couleurs (`Color(0xFFF9ABB8)`) et les mises en forme du texte en pixels évolutifs (`16.sp`). Toutes ces valeurs ont été nommées et placées dans le dossier `theme` dans des fichiers leur correspondant.

Code mort (Dead Code)

Il existait pas mal de **code mort** dans l'ensemble de l'application, des composants/méthodes utilisés nulle part avec aucune explication, des parties du code n'ayant pas d'intérêt à garder.
Après concertation de l'équipe et questionnement des anciens collaborateurs du projet, la plupart d'entre elles ont été supprimées ou même remplacées.

Code inaccessible (Unreachable Code)

Il n'y avait pas de **code inaccessible** à recenser.

Code commenté (Commented Code)

Il y avait beaucoup de code commenté, notamment des anciennes fonctions non utilisées par les anciens collaborateurs du projet, voici une **issue** qui montre ce problème. La grande majorité du code commenté a été supprimé ou intégré.

Factorisation (Refactoring)

Nous avons identifié beaucoup de code dupliqué dans le projet, pour les enlever nous avons notamment décidé de créer des composants qui permettrait de couvrir différentes duplication, parfois un changement de réflexion sur la manière d'exécution du code à permis de réduire ces duplications.

Grâce à l'utilisation de SonarQube, certaines complexités cyclomatiques trop volumineuses ont pu être détectées, pour les réduire, parfois faisant pair avec de la réduction de duplication, créer des composants et penser autrement permettait de les faire réduire.

Et lorsque le code ne permettait pas totalement d'être compris directement, des commentaires ont été ajoutés, pour les fonctions et certaines parties du code permettant de l'éclaircir.

Pour à la fois être clair et avoir une façon commune de coder, nous avons décidé de suivre des conventions, que ce soit des conventions quant à la hiérarchisation des fichiers, nommage des noms des variables et des fichiers, et de documentation des composants. (elles sont présentent dans le fichier Contributing.md)

Tests

API

Il n'y avait pas de tests dans l'API, ce qui a été corrigé dans les branches **mr-api-routing-adding-tests** et **mr-add-mongo-dao**. Nous avons également mis en place des fichiers de logs afin d'avoir une trace de la sortie des tests. Dans les tests de connexion, nous avons utilisé un **mock** pour simuler une base de données Mongo.

Application Mobile

Il existait déjà des exemples de tests comme des tests instrumentalisés. Nous avons continué d'écrire des tests en utilisant l'IDE qui permettait de générer certains tests bien que c'était insuffisant, en utilisant notamment des **dummies** et des mocks pour préparer des valeurs de tests utilisées en ce moment et qui vont être utilisées à l'avenir. Il reste encore beaucoup de tests à produire.

Maintenance Adaptative

La partie maintenance adaptative ne nous concerne pas sur cette application puisque l'environnement a déjà bien été définis lors du semestre 5, il n'y avait donc pas de choses à mettre en place, si ce n'est un conteneur docker pour fixer l'environnement a une certaine version (ce qui est toujours prévu).

Maintenance Evolutive

Devops | Jenkins

Il y avait auparavant une **pipeline Jenkins** qui existait pour l'application mobile, mais qui n'a pas pu être reproduite à cause d'une migration du système hôte qui à causer des soucis à la réinstallation du **SDK Android**, il a donc été choisi de ne pas se concentrer dessus au vu de la quantité de travail requise pour amener à un produit fonctionnel

Évolution de la navigation

La navigation existante sur l'application fonctionnait correctement. Néanmoins, afin d'avoir une page pour le scan d'ordonnance et une autre pour les effets secondaires il était nécessaire d'avoir un Footer pour avoir une navigation plus simple entre les pages.
Cependant l'application actuelle ne permettait pas de conserver un composant en l'état au travers des différentes activités.
Voici donc comment l'application était structuré en terme de navigation :

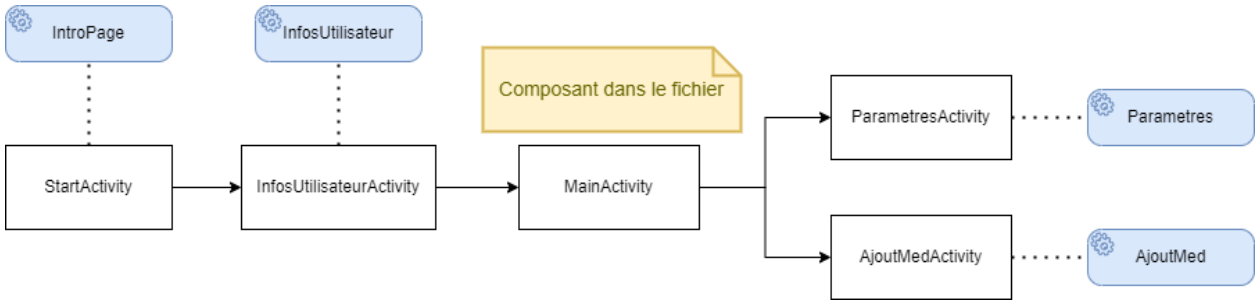

Figure 2.1 : La navigation d'antan

Nous pouvons voir qu'il y a uniquement un composant par activité (hormis MainActivity qui est directement le contenu de l'activité). Cela veut dire que s'il fallait mettre un footer, celui-ci serait recréé à chaque activité, pour chaque pages en l'occurrence.

L'idée était donc de n'avoir qu'une activité principale dont le contenu changeait selon la page souhaitée, cela permettait donc d'avoir un seul et unique Footer qui reste en l'état.

La nouvelle navigation prendra alors cette forme là :

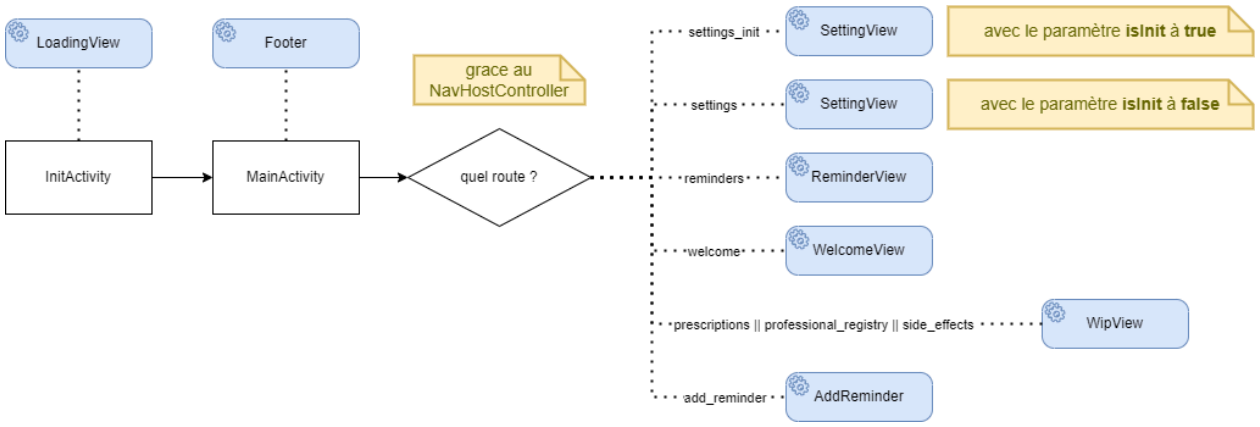

Figure 2.2 : La nouvelle navigation

Nous constatons alors qu'il y a une activité nommée `InitActivity` qui est utilisée pour charger des données pour ensuite lancer l'activité principale de l'application, qui contient donc l'entièreté des pages de l'application.

Cela a été rendu possible grâce au `NavHostController` de Jetpack Compose qui permet d'avoir un routeur ou les routes sont définies par un nom et le composable qui est lié.

Factorisation des composants

Les composants génériques (Button, Dropdown, TextField...) existants était soit, dans les cas des Button, centralisés sur un seul fichier soit, dans le cas des TextField et Dropdown, répété partout dans le code.

Les boutons

Pour les boutons cela était problématique car si le fichier présente une erreur, tous les boutons en seront impactés. De plus, certains boutons avaient une action définie, parfois selon le texte de celui-ci.
Afin de garantir un bon fonctionnement de chaque boutons, indépendamment, il était donc nécessaire de les réorganiser avec, pour chaque bouton, un fichier qui lui est propre.
Un extrait du code est disponible ici :
Extrait de code des boutons avec les actions déterminés par le texte

Et le diagramme résumant la structure des boutons au début :

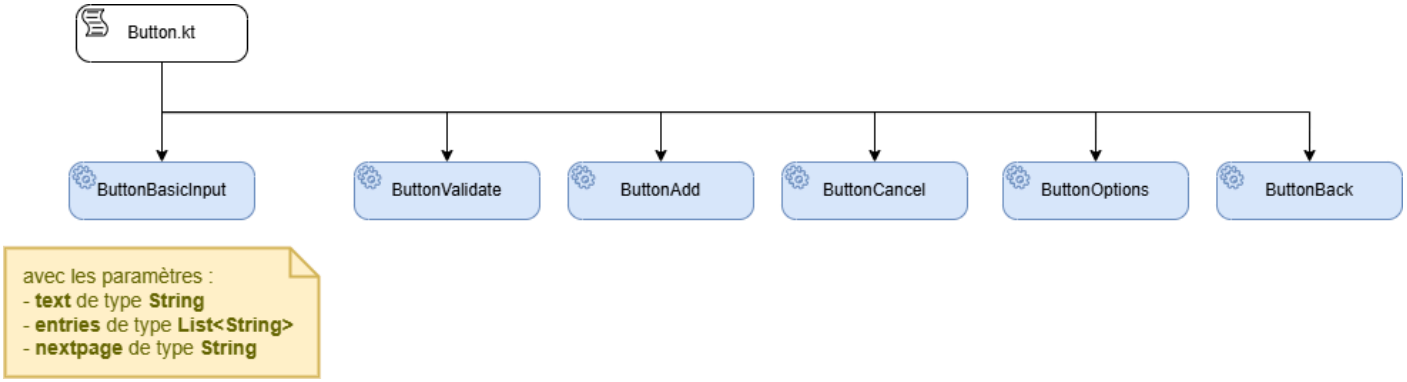

Figure 2.3 : La structure des boutons avant

Nous remarquons donc ce qui a été énoncé plus tôt : un seul fichier contenant tout avec des actions prédéfinies par moment et parfois selon le texte du bouton.

La nouvelle organisation des boutons prendra donc cette forme là :

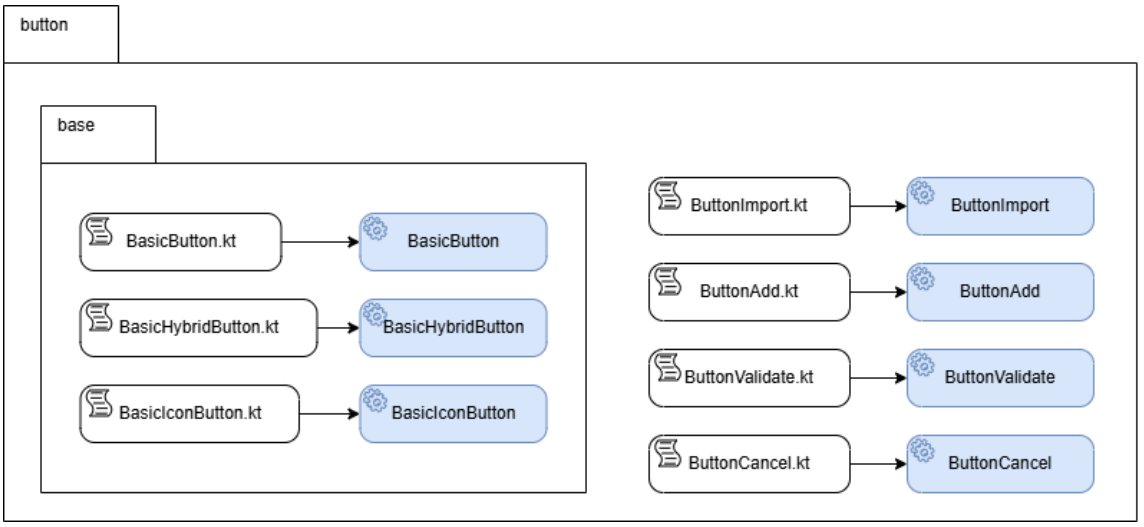

Figure 2.4 : La structure des boutons après

Nous remarquons qu'il y a alors un bouton dit "de base" et des variantes comme avant. Mais là, il y a un composant par fichier.

Les autres composants

Pour les Dropdown et les TextField, il n'y avait donc tout simplement pas de composant de base, le même code était copier-coller partout, voici donc la nouvelle structure, qui elle contient des composants de base :

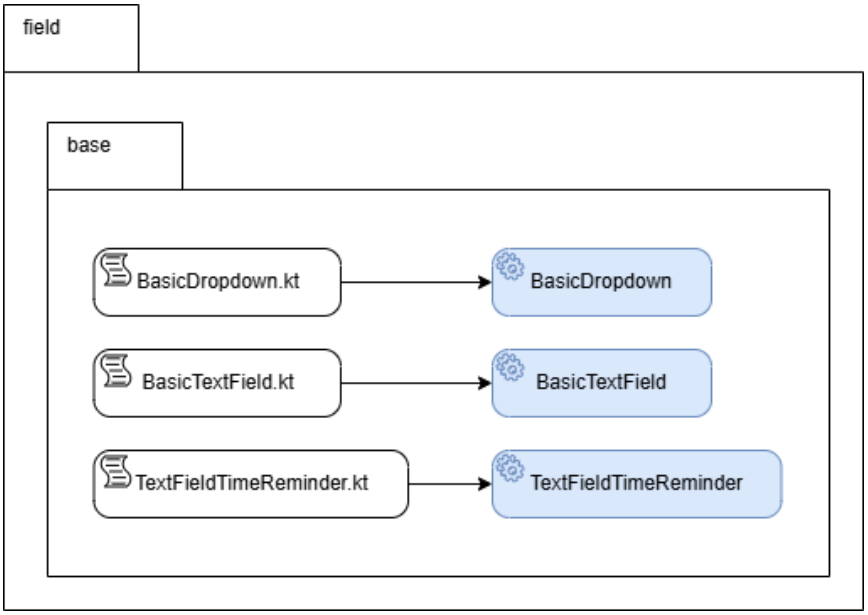

Figure 2.5 : La structure des autres inputs après

Conclusion

Pour conclure notre travail sur ce projet, nous avons récupéré une application fonctionnelle mais malheureusement chaotique et de gros soucis de lisibilité et de structure. Ce qui nous a permis de faire une grande refonte de l'application pour partir sur des bases saines. Malheureusement, nous n'avons pas pu développer de nouvelles fonctionnalités majeures. Mais il valait mieux une application fonctionnelle et agréable à maintenir qu'une qui est désorganisée avec des fonctionnalités en plus mais avec zéro maintenabilité.

Définitions

Script : Un script est un fichier contenant une série d'instructions exécutables automatiquement par un interpréteur, utilisé pour automatiser des tâches ou exécuter des programmes sans intervention manuelle.
Back-End : Le back-end désigne la partie d'une application qui gère la logique métier, le traitement des données et la communication avec la base de données, fonctionnant côté serveur et invisible pour l'utilisateur final.
Front-End : Le front-end désigne la partie visible d'une application avec laquelle l'utilisateur interagit, comprenant l'interface graphique, la mise en page et les éléments dynamiques, ici développée en Kotlin avec Android Studio.
OCR (Optical Character Recognition) : L'OCR est une technologie permettant de convertir une image contenant du texte en un format numérique exploitable, facilitant ainsi la recherche, l'édition et l'analyse du contenu textuel.
Ticket (Issues) : Un ticket, ou "issue", est une demande ou un rapport détaillant un bug, une erreur, ou une fonctionnalité à améliorer dans un projet. Il est utilisé dans les systèmes de gestion de projet (ici gitlab) pour suivre et résoudre les problèmes rencontrés durant le développement.
Demande de fusion (Merge request) : Une demande de fusion est une proposition de modification d'un code source dans un système de gestion de version. Elle permet de réviser, discuter et intégrer les changements d'une branche dans la branche principale d'un projet avant de les fusionner définitivement.
Revue de code (Code Review) : La revue de code est un processus où les développeurs examinent le code écrit par leurs collègues pour détecter des erreurs, améliorer la qualité du code, vérifier la conformité aux bonnes pratiques et assurer la cohérence du projet avant l'intégration des modifications.
Méthodes Agiles : Les méthodes agiles sont un ensemble de pratiques de gestion de projet qui privilégient la flexibilité, l'adaptation et la collaboration. Elles reposent sur des cycles de développement courts (itérations), des retours fréquents des utilisateurs et une amélioration continue pour répondre efficacement aux besoins changeants du projet.
README.md : Un fichier README.md est un document généralement inclus dans un projet logiciel, écrit en Markdown, qui fournit des informations essentielles sur le projet. Il décrit son objectif, son installation, son utilisation, ainsi que des détails importants pour les contributeurs et utilisateurs. Des conseils pour le rédiger sont disponibles sur **makeareadme**.
CONTRIBUTING.md : Un fichier CONTRIBUTING.md est un document inclus dans un projet logiciel qui guide les contributeurs sur la manière de participer au projet. Il explique les bonnes pratiques, les règles de soumission de code, les étapes pour signaler des problèmes, ainsi que les attentes en matière de style et de comportement. En voici un **exemple.**
LICENSE.md : Un fichier LICENSE.md est un document qui décrit les conditions légales régissant l'utilisation, la distribution et la modification du code source d'un projet. Il précise le type de licence sous laquelle le projet est publié (par exemple, MIT, GPL, Apache), définissant ainsi les droits et obligations des utilisateurs et contributeurs.
Complexité cognitive : La complexité cognitive est une mesure de la facilité avec laquelle un être humain est capable de le comprendre. La version la plus consensuelle (surtout parce qu'elle est intégrée à leur outil) est celle de SonarSource. Elle a été proposée en 2016 et se sert d'un compteur qui est incrémenté lorsque des

instructions et structures de contrôles particuliers sont rencontrées. En général, il est conseillé qu'elle ne dépasse pas 15.

Maintenance Corrective : La maintenance logicielle corrective est la forme type et classique de maintenance (pour les logiciels et tout le reste d'ailleurs). La maintenance corrective est nécessaire lorsqu'un logiciel rencontre un problème, notamment lorsque surviennent des défaillances et des erreurs. Celles-ci peuvent avoir un impact considérable sur la fonctionnalité du logiciel en général et doivent donc être corrigées le plus rapidement possible.

Maintenance Préventive : La maintenance logicielle préventive consiste à se projeter dans l'avenir afin que votre logiciel puisse continuer à fonctionner comme souhaité le plus longtemps possible.

SonarQube : SonarQube est un outil complet d'analyse de la qualité du code qui combine des outils d'analyse statique et dynamique pour examiner le code source et générer des rapports sur la qualité du code de votre projet . Il identifie les bugs dans le code en analysant le projet et informe les développeurs afin qu'ils les corrigent.

Factorisation (Refactoring) : La factorisation de code consiste à restructurer et mutualiser des portions de code redondantes afin d'améliorer la maintenabilité, la lisibilité et la réutilisabilité, sans en modifier le comportement fonctionnel.

Nombres Magiques (Magic Numbers) : Les nombres magiques sont un de ces cas :

- Une valeur unique avec une signification inexpliquée ou des occurrences multiples qui pourrait (de préférence) être remplacée par une constante nommée.
- Une valeur constante numérique ou textuelle utilisée pour identifier un format de fichier ou un protocole (pour les fichiers, voir Liste des signatures de fichiers).
- Une valeur unique distinctive qui a peu de chances d'être confondue avec d'autres significations (par exemple, les identifiants uniques universels - UUID).

Code mort : Le code mort est du code existant qui n'est utilisé nul part.

```
fun dead_function(){
   //code mort car calculé mais pas utilisé
   a + b
}
```

Code Inaccessible : Le code inaccessible est un code qui ne pourra jamais être exécuté.

```
fun unreachable_code_function(){
   return x
   // pas exécutable car retourné avant
   a = b + c
}
```

Code commenté : Le code commenté est inutile en lui-même et rend plus difficile la lecture et la compréhension de celui-ci, en particulier sur des gros projets. Bien qu'il est parfois utile d'en avoir comme référence ou pour ne pas perdre du code qui serait peut-être utile à l'avenir.

```
/*fun commented_code_function(){
   return x
   // fonction commentée
   a = b + c
}*/
```

Dummy : Un dummy est un objet ou une valeur fictive utilisée dans les tests pour simuler des entrées ou des sorties sans avoir de logique ou de comportement réel. Contrairement aux mocks, les dummies ne sont pas destinés à simuler des interactions, mais simplement à remplir un rôle dans les tests (par exemple, fournir des valeurs de remplacement).
Mock : Un mock (ou objet simulé) est une version factice d'un objet ou d'une fonction utilisée lors des tests pour simuler le comportement d'un composant ou d'un service externe. Les mocks permettent de tester un code de manière isolée, sans dépendre de ressources externes réelles, telles que des bases de données ou des API.
Maintenance Adaptative : La maintenance logicielle adaptative est liée à l'évolution des technologies ainsi qu'aux politiques et règles concernant vos logiciels. Il s'agit notamment des modifications de système d'exploitation, du stockage dans le cloud, du matériel, etc. Lorsque ces changements sont effectués, votre logiciel doit s'adapter afin de répondre correctement aux nouvelles exigences et continuer à bien fonctionner.
Maintenance Évolutive : La maintenance évolutive, aussi appelée maintenance adaptative, fait référence à un type de maintenance logicielle visant à faire évoluer ou à adapter un système informatique pour répondre à de nouvelles exigences fonctionnelles ou techniques.
DevOps : DevOps est une pratique qui combine le développement logiciel et les opérations IT, visant à améliorer la collaboration, l'automatisation et l'efficacité dans le déploiement, la gestion et la mise à jour des applications.
Jenkins : Jenkins est un outil open-source d'intégration continue et de livraison continue (CI/CD) qui permet d'automatiser les processus de développement logiciel, comme les tests, la construction et le déploiement d'applications. Il aide à faciliter la collaboration entre les équipes de développement et d'opérations en intégrant des modifications fréquentes dans le code source et en assurant une livraison rapide et fiable des applications.
CI/CD : CI/CD (Intégration Continue / Livraison Continue) est un ensemble de pratiques qui visent à automatiser et améliorer les processus de développement et de déploiement. **L'intégration continue (CI)** consiste à intégrer fréquemment les modifications de code dans un dépôt partagé, avec des tests automatisés pour détecter rapidement les erreurs. **La livraison continue (CD)** vise à automatiser le déploiement des applications dans des environnements de production, garantissant ainsi une mise à jour rapide et fiable du logiciel.
SDK Android : Le SDK (Software Development Kit) Android est un ensemble d'outils et de bibliothèques permettant aux développeurs de créer, tester et déployer des applications Android sur différentes versions du système d'exploitation.

Annexes

Extrait de code des boutons avec les actions déterminés par le texte

```
@Composable
fun ButtonBasicInput(text : String, entries : List<String>, nextpage : String){
    val context = LocalContext.current
    Button(
        onClick = {
            if(text == "AJOUTER MÉDICAMENT"){
                App.listMed.add(entries)
            }

            if(text == "ENREGISTRER"){
                App.heures = mutableListOf(
                    entries[3],entries[4],
                    entries[5],entries[6],
                    entries[7],entries[8]
                )
            }

            if (nextpage == "AjoutMed") {
                context.startActivity(Intent(context, AjoutMedActivity::class.java))
            }
            if (nextpage ==  "InfosUtilisateur"){
                context.startActivity(Intent(context, InfosUtilisateurActivity::class.java))
            }
            if (nextpage == "Main"){
                context.startActivity(Intent(context, MainActivity::class.java))
            }
        },
        colors = ButtonDefaults.buttonColors(containerColor = Color(0xFFFF91A4)),
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 30.dp)
            .heightIn(51.dp, 60.dp),
        shape = RoundedCornerShape(22.dp),
    ) {
        Text(text, color = Color.White, fontSize = TextUnit(18f, TextUnitType.Sp))
    }
}
```