ECF - été 2024

SOIGNEMOI

Centre Hospitalier

Rédigé par Eric Terrisson - 2024 - Bachelor CDA, option iOS

Sommaire

1. Fichiers de création de base de données et intégration de données

La création de la base de données ainsi que l'intégration des données est directement gérer par le backend du projet, à savoir, l'API Rest développée avec le framework Symfony. Les explications de création de la base de données et de l'intégration des données de test sont expliquées dans le fichier ReadMe.md du repository "soignemoi-api".

2. Transaction SQL

Le fichier transaction.sql est accessible dans le repository "soignemoi-api".

Objectif de cette transaction :

Cette transaction a pour but la création d'un nouvel utilisateur (patient) et la réservation d'un séjour.

- Etape 1: création d'un USER
- Etape 2: création d'un PATIENT lié au USER
- Etape 3 : création d'un STAY lié au PATIENT

3. Manuel d’utilisation

Ce manuel à destination du correcteur présente l’ensemble des parcours réalisables sur les versions déployées des différentes applications.

Application Web

Compte patient de test
email: j.dujardin@test.com
mot de passe: password

En tant que visiteur

Réserver un séjour
On peut accéder au module de réservation par 2 entrées. Soit par “Séjour” dans le menu header, soit par le bouton “Planifiez votre séjour” dans le corps de la page d’accueil.
Suivre les étapes de réservation jusqu’à la validation. Au moment de la validation, les visiteurs sont redirigés vers le formulaire de login / création du compte.

Créer un compte
Sur la page de login, cliquez sur le bouton “S’enregistrer” pour afficher le formulaire de création de compte. Compléter le formulaire. Après la confirmation de la création du compte, il est possible de se connecter directement.

En tant qu’utilisateur authentifié

Se connecter
On accède au formulaire de connexion par le bouton “Se connecter” du header.

Consulter son profil
Quand un utilisateur est authentifié, un nouveau bouton “Profil” dans le header permet d'accéder directement aux informations personnelles de l’utilisateur.
Toutes les données sont groupées sur la même page.

Application mobile

Compte docteur de test
email: a.lim@soignemoi.com
mot de passe: password

Se connecter
La première page de l'app propose de se connecter. Complétez le formulaire avec le compte de test et validez.

Voir les visites du jour
Le tableau de bord présente les visites planifiez pour la journée. Au clic, on peut accéder directement à la fiche du client.

Accès à une fiche client
Soit un cliquant directement sur le patient dans le tableau présentant les visites du jour.
Soit en cliquant sur le menu "Patients" et en effectuant une recherche sur le nom du patient (3 caractères min).
Tapez "duj" dans la zone de recherche et validez pour voir les résultats.

Ajouter une prescription
En fin de liste des prescriptions, cliquez sur le bouton "Ajouter". Une modal s'ouvre présentant un formulaire à compléter. Cliquez sur enregistrer une fois le formulaire complété ou balayer vers le bas pour fermer ce formulaire.

Ajouter un avis
En fin de liste des avis, cliquez sur le bouton "Ajouter". Une modal s'ouvre présentant un formulaire à compléter. Cliquez sur enregistrer une fois le formulaire complété ou balayer vers le bas pour fermer ce formulaire.

Modifier une prescription
Sur la fiche d'un patient, appuyez sur une prescription existante afin d'en afficher le détail. La date de fin de validité est alors modifiable. Changez la date et cliquez sur le bouton d'enregistrement.

Application bureautique

Compte staff de test
email: j.dridi@soignemoi.com
mot de passe: password

Se connecter
La première page de l'app propose de se connecter. Complétez le formulaire avec le compte de test et validez.

Gestion des admissions et sorties
Accédez à la section de gestions des flux par 2 moyens :

- en cliquant sur "Admissions" dans le menu fixe
- en cliquant sur le widget Entrées/Sortie du dashboard

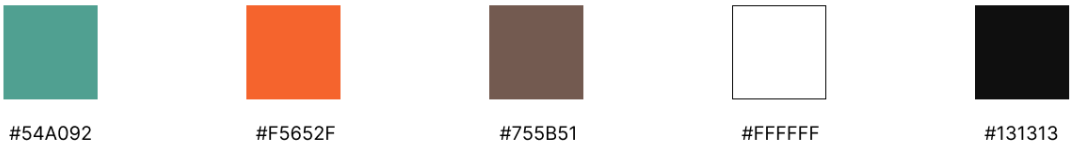
Cette section présente les entrées et sorties prévues pour la journée en cours. Les options de filtrages en haut permettent de choisir d'afficher uniquement les entrées ou les sorties. Un filtrage par service est également disponible. Un clic sur l'un des résultats permet d'accéder à la fiche détaillée du patient.

4. Charte graphique

Les éléments graphiques de ce projet ont été réalisés avec l'outil en ligne **Figma**.
Ce lien permet de visualiser directement les différents éléments (charte, wireframes, maquettes) :

https://www.figma.com/file/NDeiVvdIE4v2gIL8MUAdOI/SoigneMoi?type=design&node-id=0%3A1&mode=design&t=pljF4upphhKBG2kf-1

Charte graphique



Desktop / Web font styles

HEADER TITLE

SECTION TITLE

Body highlight text

Body text

Best contrasts

13.0 / 1

6.3 / 1

6.0 / 1

Mobile font styles

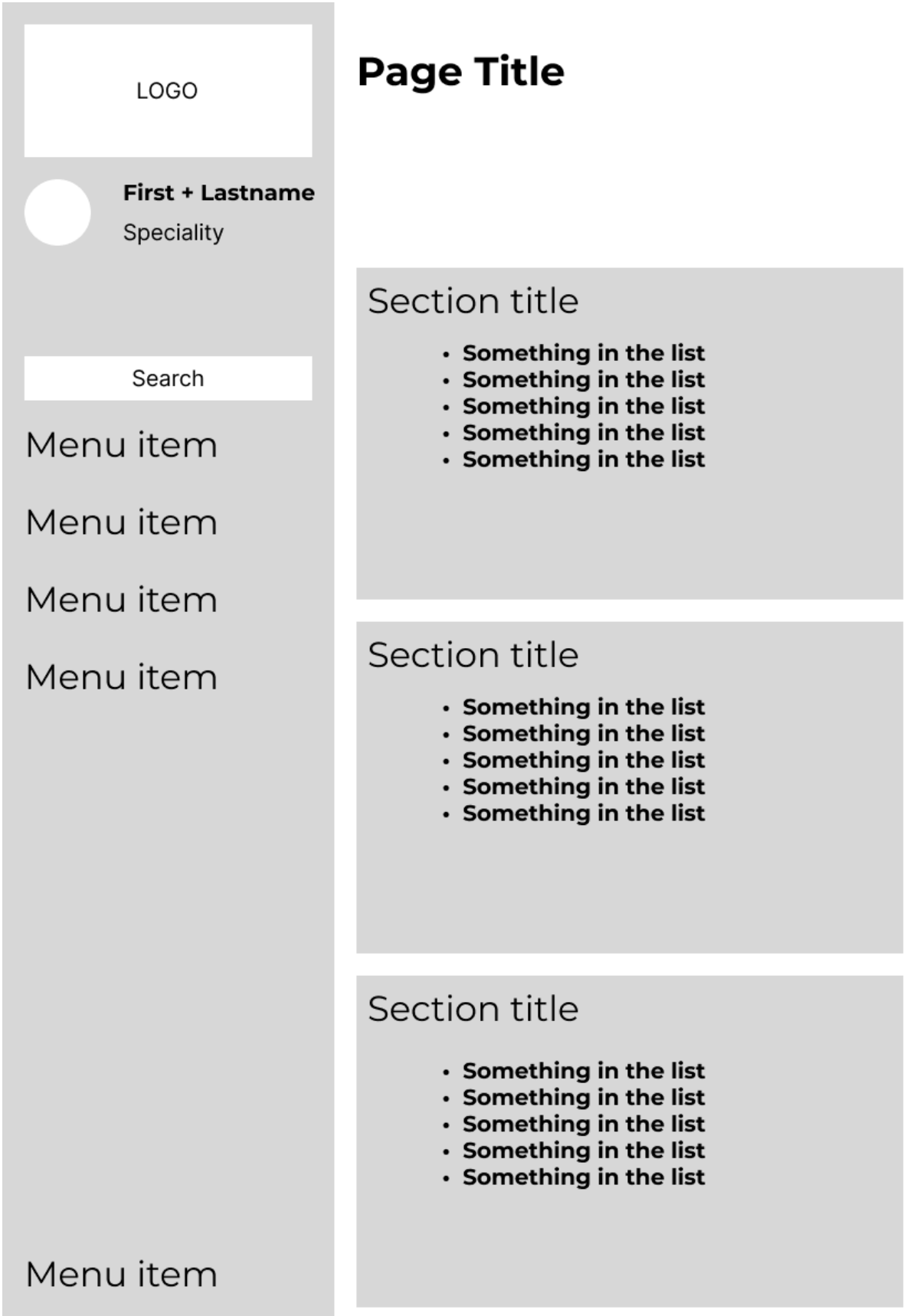
HEADER TITLE

SECTION TITLE

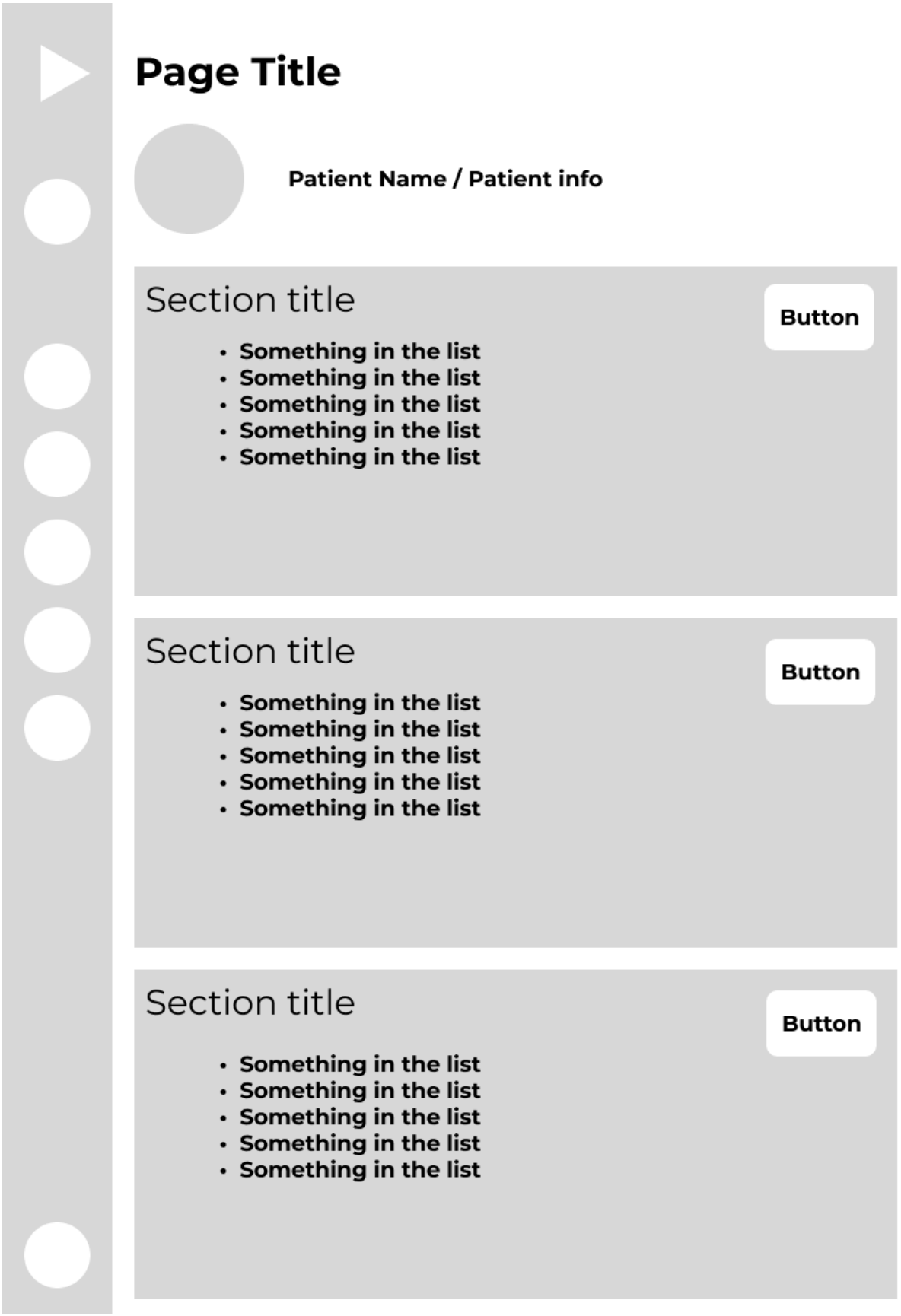
Body highlight text

Body text

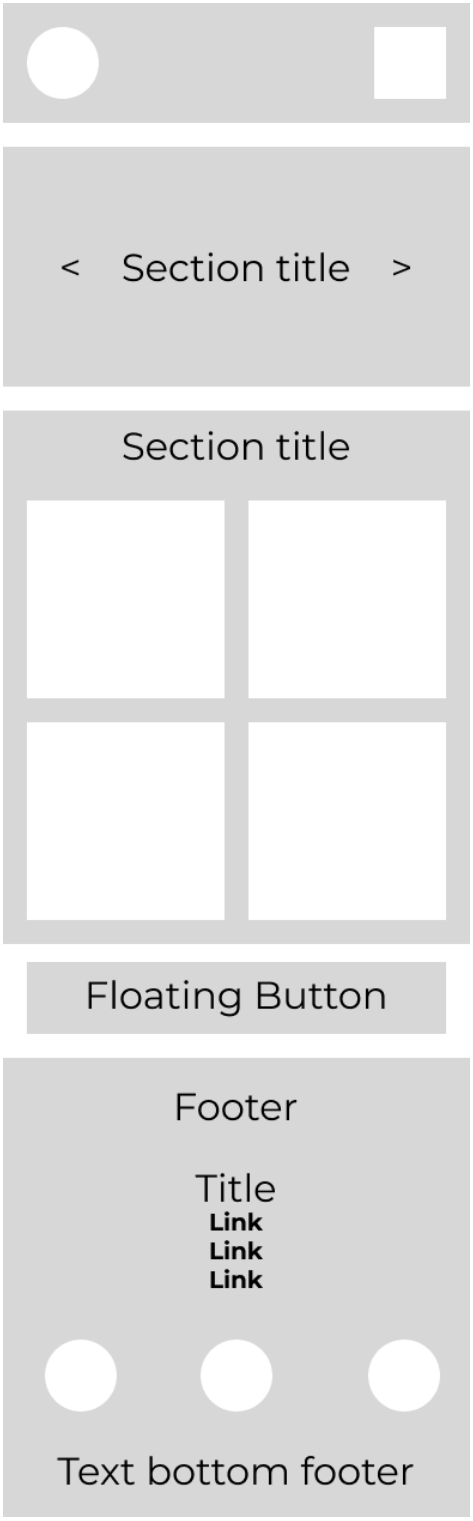
Wireframe / Mobile App - Dashboard page



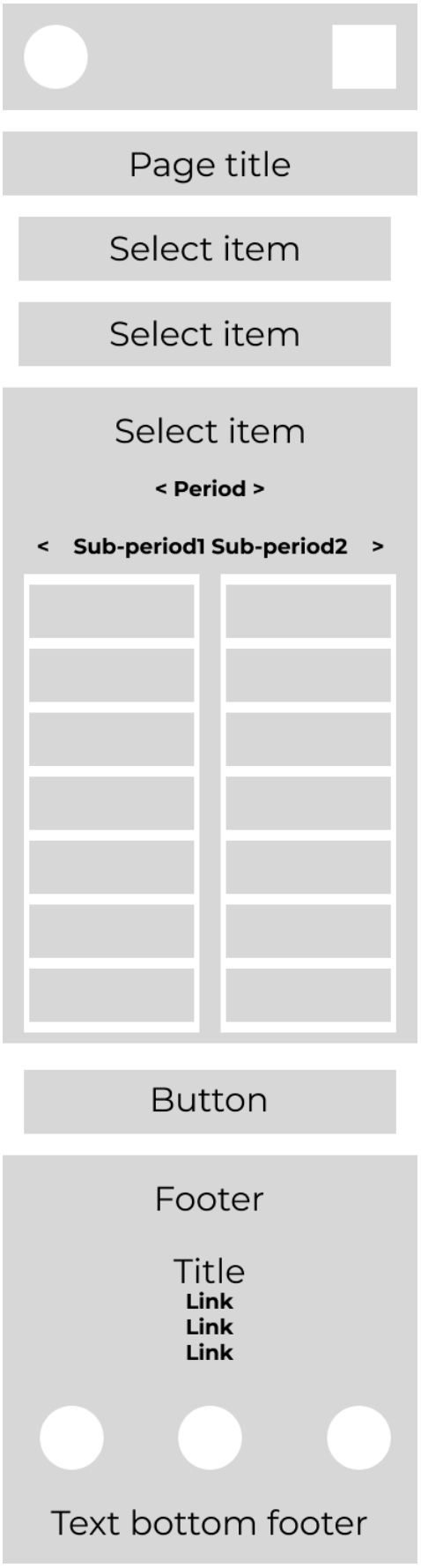
Wireframe / Mobile App - Patient details page



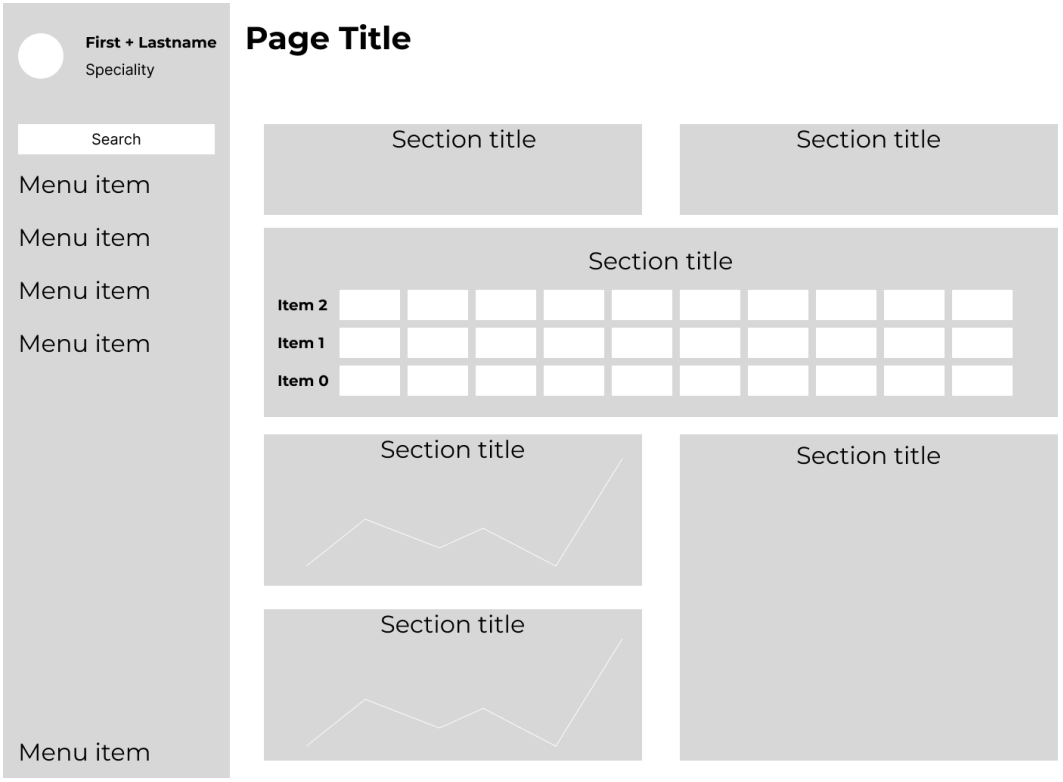
Wireframe / Web App - Home page



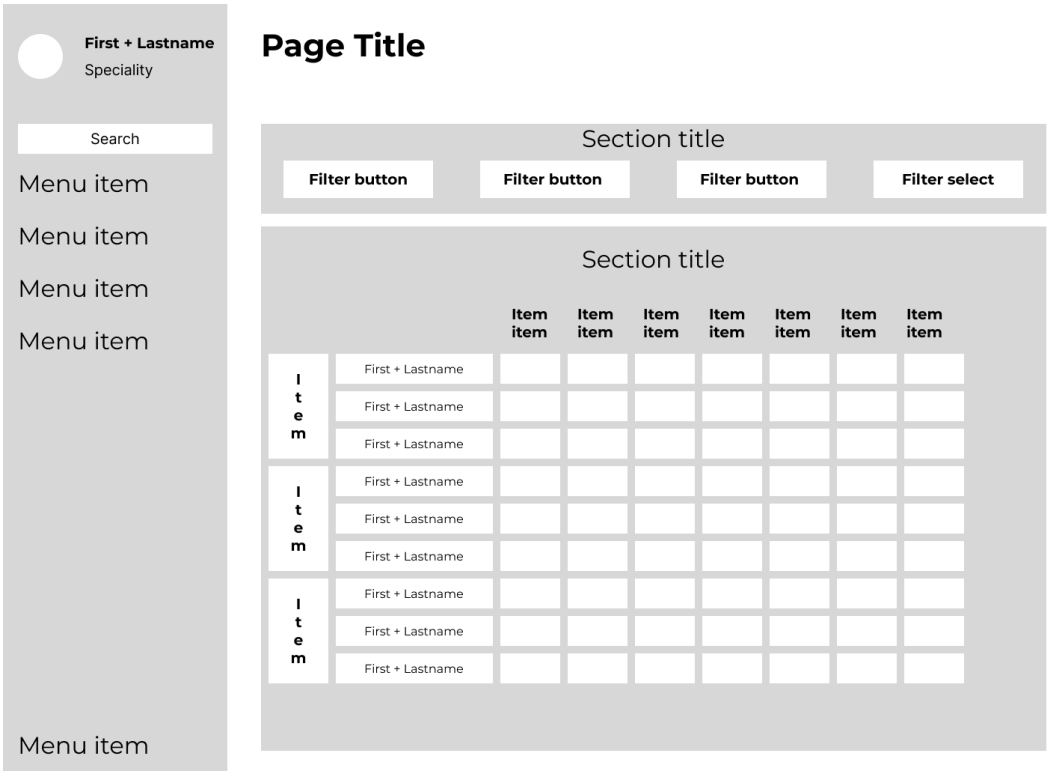
Wireframe / Web App - Stay page



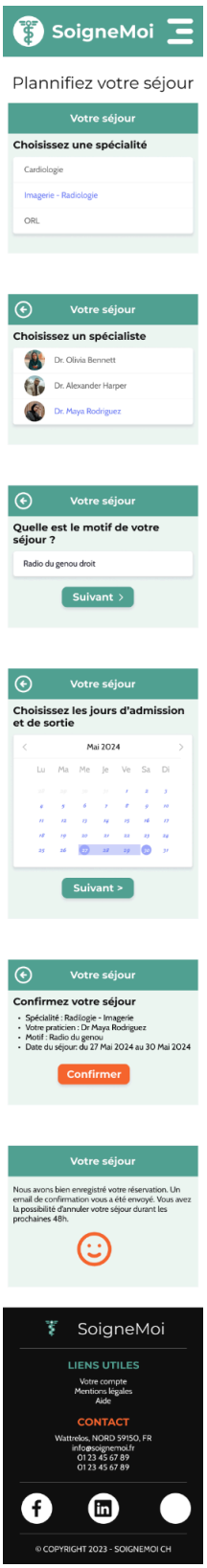
Wireframe / Desktop App - dashboard page



Wireframe / Desktop App - flux page

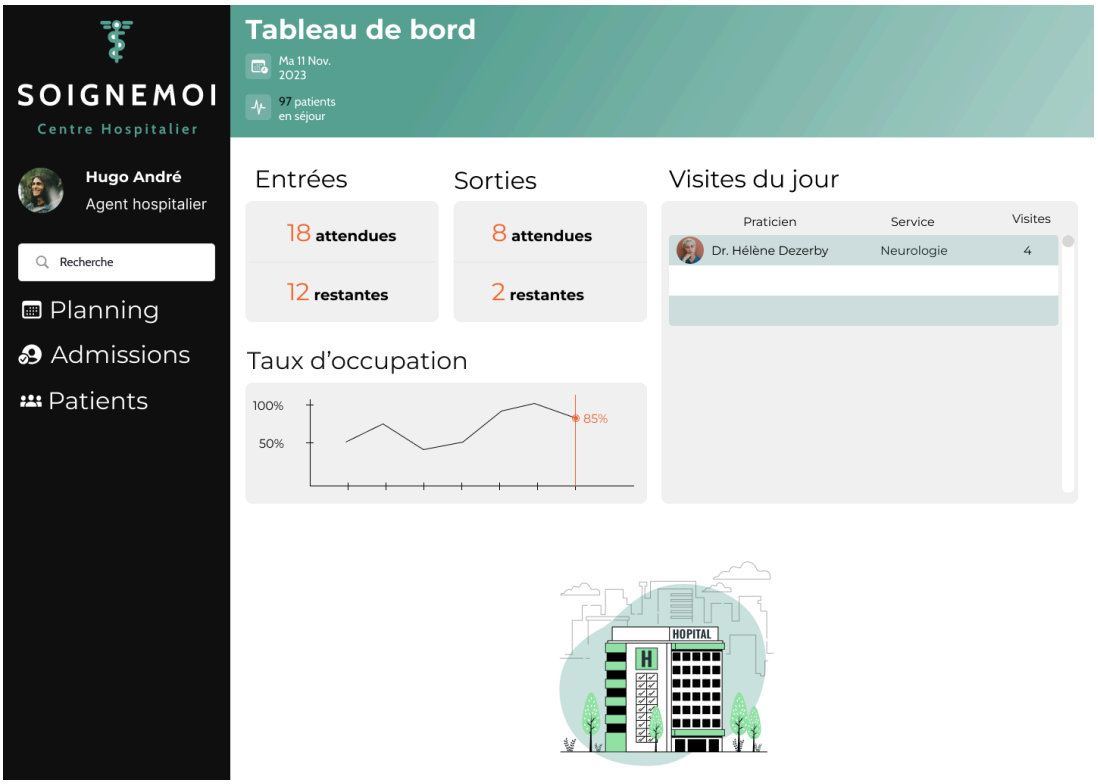


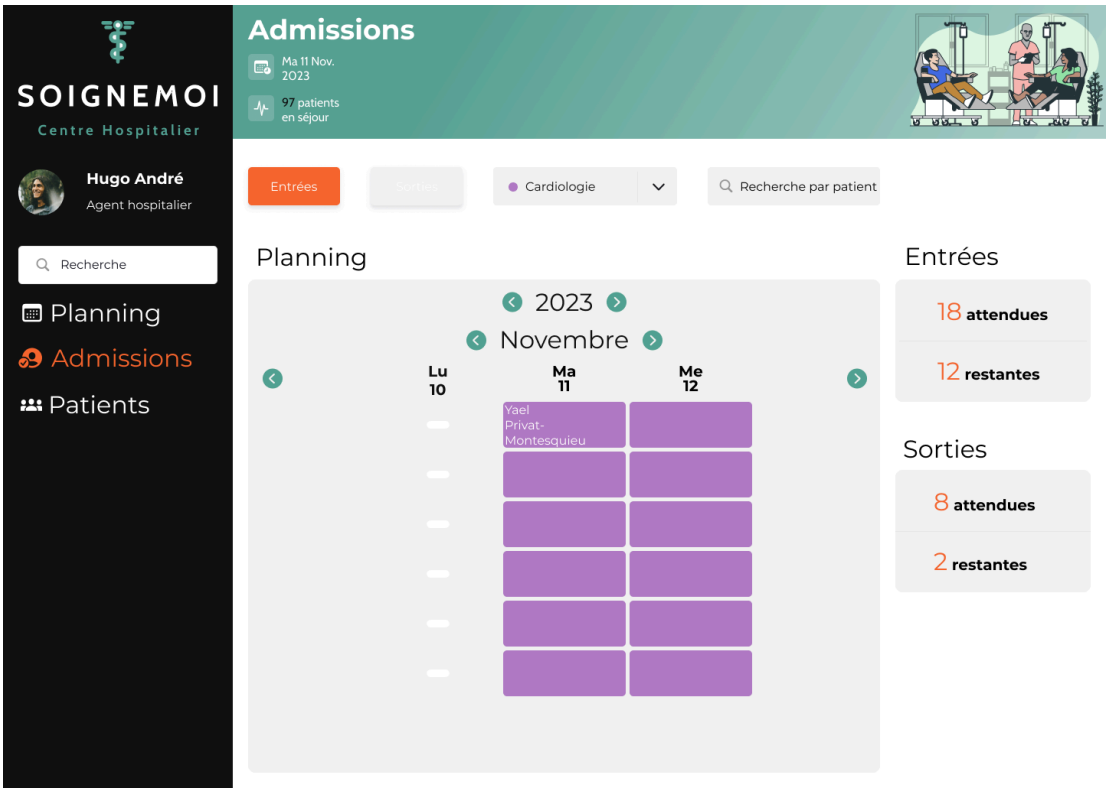
Mockup / web App



Mockup / Mobile App

Mockup / Desktop App





5. Gestion de projet

La gestion de ce projet a été réalisée selon la méthodologie en cascade.
L'utilisation d'une des méthodes Agile étant peu adapté à ce projet car :

- Un cahier des charges a été rédigé préalablement
- Le rôle de Product Owner ne peut pas être attribué.

Les documents ont été rédigés en s'adressant directement au client "SoigneMoi".

Les documents constituants la gestion de projet sont consultables en annexe :

5a - Cahier des charges - Reprises des éléments du sujet sous forme d'un cahier des charges professionnel

5b - Recommandations techniques - Réponse au cahier des charges par le détail des recommandations pour la réalisations des différents éléments du projet

Tableau de gestion de projet - Tableau Kanban de planification pour la phase de conception et de test.

https://trello.com/b/91Ttd68c/soignemoi

6. documentation technique

Pour ce projet, j'ai décidé d'utiliser la stratégie d'architecture à 3 niveaux : frontend, backend et base de données.
Cette architecture à trois niveaux favorise la modularité, la maintenance et la collaboration entre les équipes de développement. Chaque couche peut être développée, testée et déployée indépendamment, ce qui accélère le processus de développement et permet des mises à jour plus agiles.

Web App

Réflexion initiale
Les scénarios décrivent une app web avec une présentation classique d'entreprise enrichie d'un module de réservation dynamique. Des solutions techniques disponibles pour développer ce projet, utiliser la bibliothèque ReactJS me paraît la plus appropriée. Moins complet,et donc moins lourd, qu'un framework comme

Angular, ReactJS permet de créer un projet avec une structure très légère. Les dépendances, nombreuses et documentées permettent également de gagner en rapidité sur le temps de développement comparé à du développement natif (routage, requêtes http). Les dépendances limitées à celle que l'on utilise permettent de limiter les risques de sécurité liés à des sources externes.

Mobile App

Réflexion initiale

La phase préliminaire de ma réflexion a été orientée par les besoins de l'utilisateur final du produit, à savoir un médecin travaillant au sein d'un établissement hospitalier. Dans ce contexte, il est essentiel que l'application puisse être utilisée sur un matériel fiable, permettant une saisie rapide et une visualisation instantanée des données. Pour répondre à ces impératifs, j'ai décidé de développer spécifiquement l'application pour une tablette, plus spécifiquement pour les iPad, reconnues comme les tablettes les plus fiables du marché.

Ainsi, mon choix s'est orienté vers le développement natif en utilisant le langage Swift, accompagné du framework SwiftUI. Cette combinaison assure une expérience utilisateur fluide et optimisée, en exploitant pleinement les fonctionnalités offertes par les appareils iPad. La programmation en natif garantit également une intégration harmonieuse avec le système d'exploitation iOS, maximisant la stabilité, la performance et la réactivité de l'application médicale. En optant pour cette approche, nous nous assurons que notre produit répond non seulement aux exigences fonctionnelles, mais également aux standards élevés de fiabilité et d'efficacité requis dans le milieu hospitalier.

Desktop App

Réflexion initiale

Étant donné l'absence de nécessité d'accéder au disque local ou à des périphériques externes, il est envisageable d'opter pour des technologies web. Mon choix s'est finalement porté sur ReactJS, associé à Electron pour l'emballage. Cette combinaison offre la possibilité de réutiliser des composants de l'application web, tout en permettant un déploiement efficace sur différentes plateformes. En sélectionnant ReactJS avec Electron, nous nous assurons non seulement une cohérence avec l'écosystème web, mais également une flexibilité pour une diffusion optimale de notre application de bureau sur divers environnements.

APIRest

Réflexion initiale
Dans le domaine des API Rest, mes compétences se limitent à Express et Symfony. En tenant compte de ma familiarité avec le langage PHP, ainsi que de la réputation de fiabilité et de performances de Symfony, j'ai décidé de privilégier ce dernier. L'utilisation de Doctrine simplifie significativement la gestion des relations avec la base de données, contribuant ainsi à une mise en œuvre efficace de l'API. De plus, la réputation de Symfony en matière de sécurité renforce la confiance dans la protection des données et des communications, offrant ainsi un environnement robuste pour le développement et l'utilisation de l'API.

Environnement de travail

Je travaille sur un système d'exploitation **MacOS**, choisi pour sa stabilité, sa sécurité et son interface utilisateur conviviale.

Mon éditeur de texte principal est **Visual Studio Code**, un outil polyvalent qui combine légèreté et puissance. Il offre une gamme étendue d'extensions, facilitant l'intégration de différents langages de programmation, et son interface intuitive accélère le développement de code efficace.

Pour le développement d'applications iOS, j'utilise **XCode**, l'environnement de développement intégré (IDE) d'Apple. Grâce à sa suite complète d'outils, XCode simplifie la création, le débogage et le déploiement d'applications sur les appareils Apple, offrant une expérience de développement native optimale.

Lorsqu'il s'agit de tester et de valider mes API, **Postman** est un outil essentiel dans mon arsenal. Sa capacité à simplifier les requêtes HTTP, à automatiser les tests et à documenter les API facilite grandement le processus de développement, garantissant la robustesse des services que je crée.

La gestion de version est cruciale dans mon workflow, et c'est pourquoi **Github** est au cœur de mes opérations. Cette plateforme de développement collaboratif me permet de suivre les modifications, de travailler avec d'autres développeurs et de garantir l'intégrité du code source tout au long du cycle de vie du projet.

La navigation web et le débogage sont également essentiels, et c'est pourquoi **Chrome** est mon navigateur de prédilection. Ses outils de développement intégrés facilitent l'inspection du code, le suivi des performances et le débogage des applications web.

Enfin, pour la conception d'interfaces utilisateur, **SF Symbols**, la bibliothèque de symboles d'Apple, offre une collection exhaustive d'icônes prêtes à l'emploi, contribuant à une conception visuelle cohérente et esthétiquement agréable dans mes projets iOS.

Modèle conceptuel de données

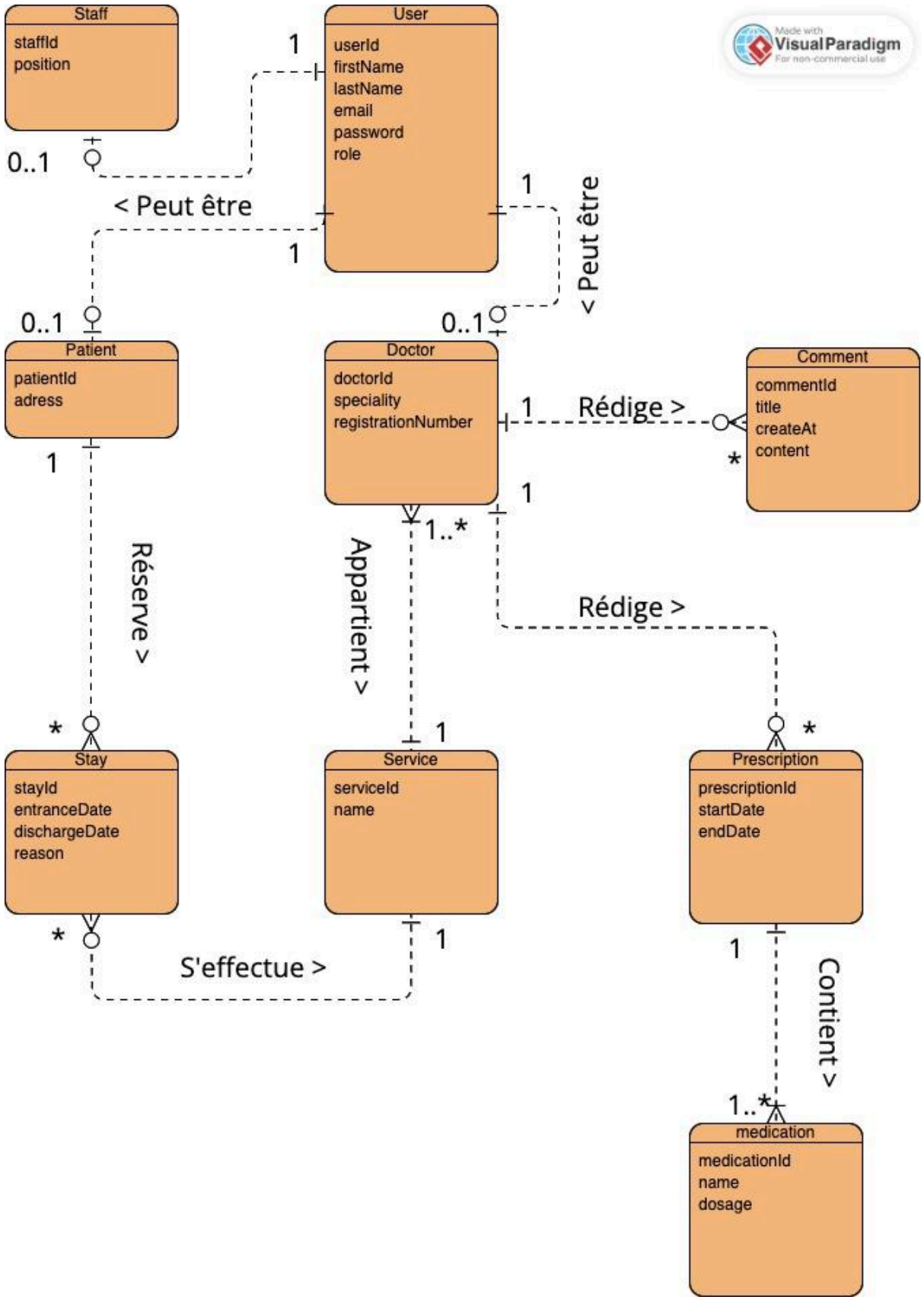


Diagramme d’utilisation et diagramme de séquence

Les diagrammes utilisateurs et quelques diagrammes de séquences sont consultables dans l’annexe **6a - Diagrammes**.

7. Plan de test

Un environnement de test complet a été réalisé pour l’application mobile.

La cible

La cible des tests est une application mobile iOS, spécifiquement développée pour iPad OS. Cette application est développée en utilisant le langage natif et Swift et le framework SwiftUI. Cette application permet essentiellement de récupérer, d’afficher et de créer des données au travers de ces échanges avec une API.

Plan des tests

Tests unitaires, tests d’intégration, tests UI, tests de performance, tests de compatibilité, tests de sécurité, tests de qualité et tests d’acceptation.

Test Unitaires

Objectif : Les tests portent sur les fonctionnalités essentielles de l’application, dont

- Login - Api
- Récupération du profil utilisateur - Api
- Création d’une prescription - Api
- Création d’un commentaire - Api
- Recherche du nom d’un patient - Api
- Ajout d’un token à l’utilisateur actif - ActiveUser
- Formatage des dates iso8601full - Extension

Environnement : Les tests unitaires sont inclus dans les sources du projet. Ils ont été réalisés en utilisant le framework XCTest.

Méthode : Pour que les tests de l'api se fassent indépendamment de la disponibilité réelle du service, il a été fait recours à un MOCKER. Cette technique permet d'émuler une session URL par un composant pour lequel on spécifie un type de retour. Cette technique permet de valider que les retours sont conformes à ceux attendus par les différentes fonctions.

Résultats : **100% validé**

Test d'Intégration

Objectifs : Vérifier comment les différents composants de l'application interagissent entre eux. Comme l'architecture de ce projet est de type MVVM, il convient donc de tester les ViewModels qui font la coordination entre les données appelées sur l'API et les interfaces graphiques.

Environnement : Les tests unitaires sont inclus dans les sources du projet. Ils ont été réalisés en utilisant le framework XCTest. Ces XCTests nécessitent que le serveur APIRest soit en fonction et avoir chargé les fixtures dans Symfony afin d'avoir des données sur lesquelles jouer. Il faut également, dans l'app mobile, que l'URL de base soit définie ainsi qu'avoir renseigner les variables d'identification avec un compte test (docteur) dans **Product\Scheme\Edit Scheme -> Test - Onglet Arguments**

Méthode : Ces tests sont rédigés à l'aide de XCTest. Pour les démarrer,il suffit de lancer les tests du projet dans XCode.

Résultats : **100% validé**

Tests UI (Interface Utilisateur)

Objectif : Ces tests ont pour but d'automatiser la **vérification des scénarios décrits par la user story 6** en simulant le comportement d'un utilisateur.
Les 3 scénarios testés sont : rédaction d'une prescription, rédaction d'un avis et modification d'une prescription.

Environnement : les UITests nécessitent que le serveur APIRest soit en fonction et avoir chargé les fixtures dans Symfony afin d'avoir des données sur lesquelles jouer. Il faut également, dans l'app mobile, que l'URL de base soit définie ainsi qu'avoir renseigner les variables d'identification avec un compte test (docteur) dans **Product\Scheme\Edit Scheme -> Test - Onglet Arguments**

Méthode : Les UITests sont intégrés au projet, ils sont également rédigés à l'aide de XCTest. Pour les démarrer,il suffit de lancer les tests du projet dans XCode.

Résultats : **100% validé**

Test de compatibilité

L'application a été développée pour une compatibilité maximale avec la version 16.1 de iOS. Des tests sur simulateur montrent qu'elle est également compatible avec la version 16.0 mais pas avec les versions précédentes. Concernant les dernières versions iOS (17.2), elles n'ont pu être testées faute d'avoir accès au matériel nécessaire.
A faire : tester l'app sur du matériel plus récent.

Résultats: **0% validé**

Tests de Sécurité

Les tests de sécurité ont été effectués en se basant sur le Top 10 des vulnérabilités fourni par l'OWASP. Pour chaque type de vulnérabilité est donné une définition et **les bonnes pratiques mises en place dans l'application** pour répondre à cette vulnérabilité.

Injection SQL
Vulnérabilités : injection de code SQL dans les formulaires.
Bonnes pratiques : Données sont sérialisées avec une bibliothèque sécurisée (JSONSerialization); Utilisation de structures CODABLE pour la sérialisation JSON afin de valider les données; Utilisation du protocole https; Messages d'erreurs non exhaustifs; Limitation de la taille des champs de texte.

Broken Authentication
Vulnérabilités : authentification faible
Bonne pratique : authentification des requêtes par JWT; utilisation du protocole https.

Cryptographic failures
Vulnérabilités : Données sont exposées car le chiffrement est faible, inexistant ou
Bonne pratique : utilisation du protocole https.

Insecure design
Vulnérabilités : Conception inappropriée ou lacune architecturale
Bonne pratique : architecture en 3 couches, intégration de la sécurité dans le design même de la conception de l'app sans surcouche.

Security misconfiguration
Vulnérabilités : Le principe de "Security misconfiguration" fait référence aux vulnérabilités qui se produisent lorsque les composants d'une application sont configurés de manière incorrecte ou non sécurisée. Ces vulnérabilités résultent souvent de la négligence des bonnes pratiques de sécurité lors de la configuration des différents éléments d'une application.
Bonnes pratiques : vérifier les fichiers de configuration; suppression compte et mot de passe par défaut; uniquement privilèges nécessaires (principe du moindre privilège).

Vulnerable and outdated components
Vulnérabilités : Le principe des "Vulnerable and outdated components" se réfère aux vulnérabilités qui surviennent lorsque des composants logiciels utilisés dans une application sont non pris en charge, obsolètes ou vulnérables à des exploits connus.
Bonnes pratiques : limiter l'utilisation de composants externes à ceux qui sont absolument nécessaires, parfaitement maîtrisés et connus; Veille à effectuer sur les vulnérabilités de la version d' iOS (içi 16.1).

Identification and authentication failures
Vulnérabilités : Les échecs d'identification et d'authentification surviennent lorsque les fonctions liées à l'identité d'un utilisateur, à l'authentification ou à la gestion de session ne sont pas mises en œuvre correctement ou suffisamment protégées.
Bonne pratique : mécanisme d'authentification fort; Temporisation des demandes de connexion.

A faire : Gestions des erreurs d'authentification.

Software and data integrity failures
Vulnérabilités : Les échecs d'intégrité logicielle et de données surviennent lorsque le code et l'infrastructure ne protègent pas contre les violations d'intégrité. L'utilisation de plugins, bibliothèques ou modules logiciels provenant de sources non fiables, de dépôts ou de réseaux de diffusion de contenu (CDN) peut introduire le potentiel d'accès non autorisé, de code malveillant ou de compromission du système par des attaquants.
Bonnes pratiques : Sources des composants fiables et légitimes, vérifier l'authenticité des composants par leur signature, mécanismes de mise à jour utilisant des protocoles sécurisés de transmission et de vérification de l'authenticité.

Security logging and monitoring failures
Vulnérabilités : Les échecs de journalisation (logging) et de surveillance (monitoring) de sécurité sont souvent à la base de presque chaque incident majeur de sécurité. Les attaquants comptent sur une surveillance insuffisante et une réponse lente pour établir une présence dans votre application et atteindre leurs objectifs tout en restant non détectés.
Bonnes pratiques : aucunes

A faire : journalisation, surveillance, alertes proactives sur l'API.

Server-side request forgery
Vulnérabilités : Les failles de type Server-side Request Forgery (SSRF) surviennent lorsqu'une application web ne valide pas correctement l'URL fournie par l'utilisateur lorsqu'elle récupère une ressource distante. Cela permet à un attaquant de contraindre ou de forcer l'application à envoyer une requête forgée vers une destination inattendue, même lorsque l'application est protégée derrière un pare-feu, un réseau privé virtuel (VPN) ou une autre forme de liste de contrôle d'accès réseau (ACL).
Bonnes pratiques : Une app iOS effectuant des appels réseaux définis vers une API ne semble pas vulnérable aux attaques SSRF.

Résultats : **50% validé**