UNIVERSITÉ CLAUDE BERNARD LYON 1
UCBL LYON 1

DÉPARTEMENT INFORMATIQUE - MASTER 1

MIF01 : Gestion de projet et conception d'applications

RAPPORT

Application inspirée de *monespacesante.fr*

Présenté par:
Enzo Cecillon - p1805901
Alexis BONIS - p1805132

Table des matières

I. Présentation du sujet

Aujourd'hui, la société essaye de plus en plus de se diriger vers un monde 100% numérique.

En effet, la majorité de nos systèmes sont passés à l'ère du numérique, comme les banques, les impôts ou même les formulaires d'inscription dans certaines administrations.

Notre projet se concentrera donc sur une autre version du site https://www.monespacesante.fr/ qui permet à un patient de renseigner ses informations personnelles afin de créer un dossier de santé numérique, que les médecins et les patients pourront consulter et remplir. Seul le patient en question, ainsi que les médecins qu'il a consultés auront accès à ce dossier.

II. Architecture et Designs Patterns

2.1 Identification des besoins par la conception pilotée par le domaine

La conception pilotée par domaine est une approche majeure de conception de logiciels, se concentrant sur la modélisation de ces derniers.

Nous avons simulé l'interaction avec un client, et en particulier comment appréhender une application selon son point de vue, en comprenant la logique métier. Nous avons défini un vocabulaire commun qui nous a permis de transposer la logique métier de la manière la plus adaptée à notre code. Nous avons élaboré un modèle de domaine, en définissant des scénarios de cas d'utilisations pour chacun des acteurs, dans l'objectif d'identifier au mieux les besoins métier.

La première partie de ce rapport permet d'identifier deux acteurs de l'application: les patients et les professionnels de santé. Nous en avons déduit 8 cas d'utilisations principales que nous avons renseignées dans le schéma ci-dessous.

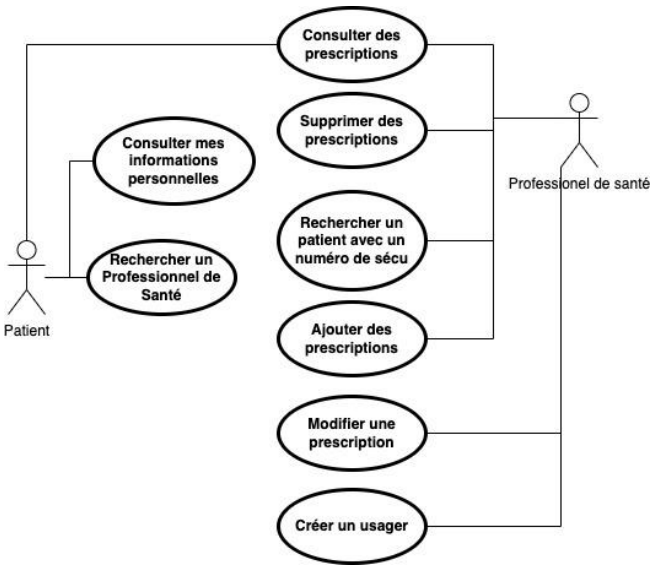


Schéma 1 - Cas d'utilisations des acteurs

2.2 Designs Pattern

2.2.1 Modèle MVC

Définition

Le modèle Modèle-Vue-Contrôleur (ou MVC) consiste à séparer un code mettant en place une interface graphique en trois parties, afin de permettre une meilleure maintenance et accessibilité de l'application.

Les trois parties de ce modèle de conception sont : le **Modèle**, qui contient la logique dite "métier" de l'application ainsi que les données, le **Contrôleur**, qui permet de traiter les actions de l'utilisateur et fait le lien entre le Modèle et la **Vue**, cette dernière contient uniquement l'affichage.

Il existe deux types d'architecture:

- Push-based, le Contrôleur envoie les informations à la Vue.
- Pull-based, la Vue interroge le Contrôleur pour avoir les informations stockées dans les Modèles.

D'autre part, il existe plusieurs niveaux de MVC (pyramide hiérarchique) en fonction de ce que les contrôleurs peuvent gérer :

- Type 2, un Master Contrôleur responsable de la gestion des sous-contrôleurs.
- Type 1, ensemble de contrôleurs indépendants.

Cette répartition des responsabilités permet de valider le principe *Single Responsability* des principes SOLID, qui permettent d'avoir une application facilement maintenable et modifiable.

Pourquoi est-ce adapté à notre projet ?

Notre application a été sujette à beaucoup de changements lors de sa conception, et avec le modèle MVC, le débogage est grandement facilité car nous pouvons identifier plus facilement d'où vient le problème éventuel.

De plus, le MVC permet de faciliter l'implémentation de nouveaux modèles de conception en séparant les responsabilités des classes et en organisant le flot de contrôle dans l'application.

Nous avons choisi pour cette application d'utiliser le modèle MVC Push-based de type 2 dans le but de séparer complètement la vue du reste de l'application, ceci pourrait faciliter un portage de l'application vers différentes plateformes.

Enfin, nous avons mis l'accent sur le fonctionnement de l'application plutôt que son affichage, ce qui explique pourquoi nous avons favorisé les tests au détriment de l'implémentation de la vue.

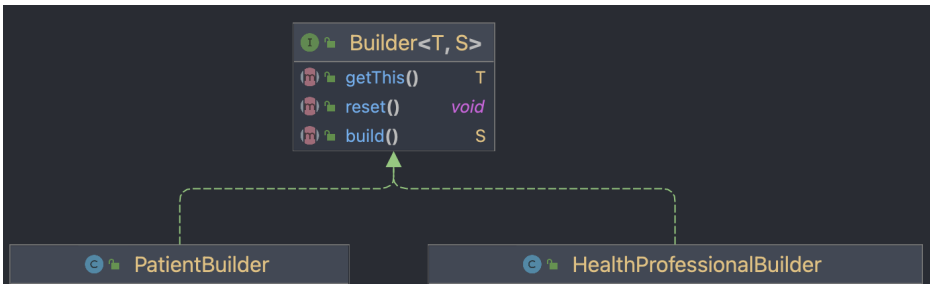
2.2.2 Builder

Définition

Le modèle Builder est un patron de conception qui permet de construire des objets complexes étape par étape. Il permet de produire différentes variations ou représentations d'un objet en utilisant le même code de construction.

Pourquoi est-ce adapté à notre projet ?

Nous avons la possibilité de construire des modèles de patient ou de professionnel de santé de façon totalement modulaire en enchaînant la construction d'un objet, attribut par attribut, sans avoir à définir des constructeurs pour chaque cas.



Il est possible d'encapsuler les étapes utilisées pour construire un objet en les mettant dans une classe séparée que l'on nomme Directeur. Nous avons choisi d'appliquer cette classe puisqu'elle se révèle idéale pour mettre en place des routines de construction et pouvoir les réutiliser ensuite dans notre application.

2.2.3 Data Access Object (DAO)

Définition

Le modèle Data Access Object (ou DAO) peut fonctionner de pair avec le modèle MVC. En effet, il consiste à manipuler des Modèles que l'on peut stocker dans différents types de gestionnaire de données (base de données, tableau … etc.). Il permet également d'interagir avec leur stockage à l'aide des opérations CRUD[1].

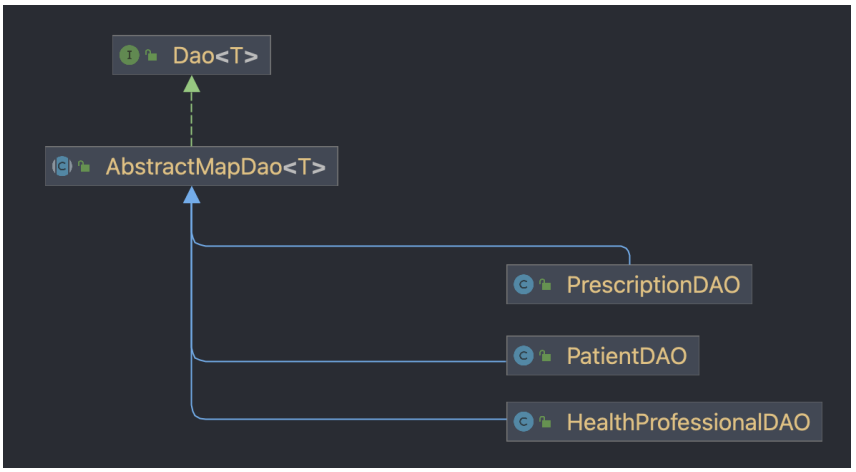
Pourquoi est-ce adapté à notre projet ?

Nous utilisons trois collections distinctes, une pour les patients, une pour les professionnels de santé, et une autre pour les prescriptions.

Les DAOs, nous permettent de dialoguer avec ces collections, afin de stocker nos Modèles, et d'effectuer les différentes opérations CRUD.

Ils nous permettent également de gagner en modularité en fonction de la manière dont on veut faire persister les données.

Ceux-ci permettent également de faciliter les tests sur la gestion des Modèles métier, sans utiliser leurs Modèles associés.



2.2.4 Data Transfert Object (DTO)

Définition

Le modèle Data Transfert Object (ou DTO), peut être un modèle complémentaire à celui des DAOs. Les DTOs permettent de dé-corréler les Modèles métier (patient, professionnel santé, prescription) des représentations qui sont échangées dans les requêtes et les réponses. Par ailleurs, cela permet d'ajouter une couche de sécurité en limitant les informations exposées lors des appels/fonctions.

[1] Create, Read, Update, Delete.

Pourquoi est-ce adapté à notre projet ?

Premièrement, l'ajout des DTOs apportent une meilleure modularité et permettent de regrouper dans un seul objet, plusieurs données qui nécessitent de nombreux appels vers un serveur. Ce pattern facilite grandement un portage vers une version web.

Deuxièmement, nous manipulons plusieurs données personnelles qu'il faut montrer au minimum. Les DTOs vont pouvoir limiter l'exposition des données sensibles dans l'application et les appels directs aux méthodes de leurs modèles associés.

III. Ethique

Dans notre projet, nous traitons donc les données personnelles des patients, sur lesquelles notre application repose. Cependant, certaines se posent : comment est-ce que nous protégeons les données des patients ? Quels sont les risques vis-à-vis des données personnelles ? De même, un sujet clé de notre époque, quels sont les dangers pour l'environnement ?

3.1 Règlement Général sur la Protection des Données (RGPD)

En effet, le traitement d'informations personnelles nous impose de respecter le RGPD (Réglement Général sur la Protection des Données). Ce règlement impose aux entreprises de poursuivre trois grands objectifs[2] :

- Renforcer les droits des personnes;
- Responsabiliser les acteurs traitant des données;
- Crédibiliser la régulation.

Dans notre projet nous ne respectons pas le RGPD car nous n'avons pas mis en place un système de consentement avec la patient et le professionnel de santé. N'importe quel professionnel de santé peut avoir accès aux données du patient et modifier/ajouter/supprimer ses prescriptions.

Dans une situation réelle, il serait imposé à l'entreprise de respecter le RGPD en mettant en place, par exemple, un système permettant de laisser l'accès aux données d'un patient seulement aux personnels de santé dont le patient a autorisé l'accès.

3.2 Secret médical

Comme énoncé dans la partie précédente, notre projet permet à n'importe quel professionnel de santé de modifier/ajouter/supprimer l'ordonnance de n'importe quel patient. Ce n'est donc également pas conforme au secret médical, qui impose aux professionnels de santé de garder le dossier du patient "secret" en ne divulguant les objets de la consultation seulement qu'aux personnes autorisées par le patient.

[2] *Source : Règlement européen sur la protection des données : ce qui change pour les professionnels | CNIL*

Dans le cadre de *monespacesante.fr*, le secret médical est maintenu, car le dossier d'un patient n'est visible que par les professionnels de santé autorisés.

3.3 Risques de piratage

N'importe quel logiciel informatique n'est pas fiable à 100%, que ce soit dans ses fonctionnalités, mais surtout au regard de sa sécurité.

Notre projet, et même *monespacesante.fr*, traite des informations médicales, qui sont directement liées à la santé des personnes et qui constituent donc des informations hautement confidentielles. Hors, le risque 0 de piratage n'existe pas, et, dans le pire des cas, les informations personnelles peuvent alors être dérobées et ensuite diffusées.

Notre projet ne possède pas de charte ou de politique de confidentialité, mais dans*monsespacesante.fr*, le consentement du patient est supposé accepté lors de la création d'un compte, ce qui couvre l'entreprise d'un problème de ce type. Mais il reste légitime de se demander si ce n'est pas risqué d'utiliser une plateforme de ce type pour des informations aussi confidentielles que la santé.

3.4 Pollution

La préservation de l'environnement est un enjeu très important actuellement. Cependant, le stockage de données sur des serveurs et l'utilisation même d'une application logicielle polluent énormément.

L'utilisation d'objets informatiques constituent d'ores et déjà 1180 kg de CO2/an sur 12 092 kg de C02/an en 2016, ce qui représente environ 9,76% de leur empreinte carbone par an[3]. L'ajout d'une application utilisée quotidiennement augmenterait donc l'empreinte carbone.

Pour un projet de notre envergure, où nous ne stockons pas de données sur un serveur, les dégâts sont négligeables, mais pour une application comme *monespacesante.fr*, il faudrait également grandement optimiser la taille des données sur le serveur.

En effet, l'application a été conçue dans le but que la totalité de la population française l'utilise, ce qui implique une taille de données extrêmement élevée au total et cela représenterait donc un grand facteur de pollution, à la fois en raison de son utilisation fréquente, mais aussi compte tenu de sa grande taille de données.

De plus, l'idée de la Souveraineté Numérique au niveau européen pourrait introduire de nouvelles alternatives aux services de cloud Américain comme AWS, Google … etc. Ces nouveaux services de cloud seront moins efficaces que la concurrence américaine, ce qui impliquerait un impact écologique plus important. En contrepartie, la sécurité des données sera augmentée car l'union européenne pourrait superviser l'utilisation des données, ce qui n'est pas le cas aujourd'hui avec les géants américains.

Ce qui peut nous amener à réfléchir sur notre choix de priorité: soit favoriser l'écologie au dépend de la protection des données, ou bien favoriser la sécurité en négligeant un peu plus l'écologie.

---

[3] *Source : L'empreinte carbone des français, un sujet tabou ? – Ravijen*

IV. Tests

Le test se traduit par l'évaluation d'un système ou d'un composant par des moyens automatiques ou manuels, afin de vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus.[4] Il existe différentes catégories de tests comme par exemple l'intégration, l'unitaire ou le bout en bout … etc.

Certaines méthodes peuvent permettre d'optimiser le code et le temps de développement, tout en garantissant le bon fonctionnement de l'application. Pour notre projet ,nous avons choisis de réaliser des tests unitaires ainsi que la mise en pratique de la méthode TDD (**T**est **D**riven **D**evelopment) qui nous a permis de valider le bon fonctionnement de chaque partie de notre code.

1. Tests Unitaires

Définition

En programmation informatique, le test unitaire (ou « T.U. », ou « U.T. » en anglais) est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel, ou d'une portion d'un programme (appelée « unité » ou « module »)[5].

Un test unitaire permet aussi de démontrer l'utilisation des différentes méthodes à travers son nom et ses instructions.

Ce que cela a apporté à notre projet

Les tests unitaires ont participé à l'amélioration de la qualité de notre code, notamment en facilitant l'exercice de refactorisation de l'application en modèle MVC. En effet, nous avons pu visualiser directement les méthodes qui ne s'accordaient plus avec le nouveau modèle de structuration du code.

Ils nous ont donc permis de trouver en amont des bugs, qui auraient pu nous poser problème plus tard lors de l'intégration des fonctionnalités. Par extension, le coût total de bugs trouvés pendant la phase de développement est largement amorti, grâce aux tests unitaires.

Enfin, nous avons pu identifier des cas particuliers d'utilisations qui n'auraient sûrement pas été pris en compte lors de l'écriture du code. Par exemple, lors de la création d'un patient, nous vérifions que le numéro de sécurité sociale est bien composé de nombres et de longueur 13.

---

[4] *IEEE (Standard Glossary of Software Engineering Terminology)*

[5] Source: *Test unitaire — Wikipédia*

2. TDD

Définition

Le TDD est une méthode de développement de logiciel qui consiste à concevoir un logiciel par petites étapes, de façon progressive, en écrivant, avant chaque partie du code source propre au logiciel, les tests correspondants et en remaniant le code continuellement.[6]

Ce que cela a apporté à notre projet

Nous avons utilisé le TDD pour la conception des DAOs notamment. En effet, nous avons d'abord commencé par écrire les tests afin de pleinement définir les différentes fonctions et contraintes que les DAOs devront satisfaire.

Cela nous a permis de gagner beaucoup de temps, notamment lors des phases de débogage, car nous pouvions nous concentrer sur une seule fonctionnalité à la fois et nous savions donc directement d'où venait le problème lorsqu'il y en avait un.

V. Conclusion

1. Ce que l'on a appris

La notion de Modèles de Conception nous a été introduite grâce à ce projet, nous avons pu constaté que ces modèles permettent de mieux structurer le code de façon compréhensible et logique, en fonction des problèmes de conception que l'on veut résoudre.

Les tests, bien que fastidieux, nous ont accompagnés pendant la phase de refactorisation, afin de vérifier la non-régression des fonctions déjà existantes dans le code original.

Nous avons aussi complètement dissocié la Vue du code métier, dans le but de ne pas écrire du code qui dépend de l'interface graphique de l'application. Cela a permis de nous concentrer essentiellement sur la logique métier et de créer une application modulaire, pouvant s'adapter plus facilement à des changements de structure.

De plus, nous avons opté pour une organisation basée sur le Cycle en V, où nous avons d'abord étudié les besoins métiers à travers la création de diagrammes. Puis, s'en est suivi l'écriture des tests, en accord avec les besoins. Cependant, cela nous a poussé à complètement recoder le projet, ce que nous n'avions pas prévu à l'origine. Ce temps perdu nous a grandement retardé sur le reste du projet. Pour éviter cela, nous aurions dû nous concentrer sur une fonctionnalité à la fois, plutôt que de tout faire en même temps et finir avec une application incomplète.

[6] *Source: Test driven development — Wikipédia*

2. Discussion

Peu avant la date limite du rendu, nous avons réalisé que le système de deux classes distinctes pour les patients et professionnels de santé, aurait pu être optimisé par l'intermédiaire de rôles.

En effet, nous aurions eu une classe utilisateur, avec des attributs généraux pour tous les types d'usagers, mais avec un attribut spécifique qui définirait leur rôle. Ces rôles pourraient posséder des droits uniques ou partagés, selon les cas.

Par exemple, un professionnel de santé et un patient pourraient tous deux consulter les prescriptions de ce dernier, tandis que seul le professionnel de santé pourrait ajouter une prescription pour le patient. Ce concept serait "facilement" applicable avec les modèles de conception Builder et Director.

Pour terminer, nous avons dû faire face à un problème de compatibilité avec l'application. Cette dernière ne pouvait plus être exécutée sous Mac suite à une mise à jour du paquet JavaFX (cf. fichier de log sur la forge).

Nous avons eu l'idée de créer une image Docker qui serait utilisée pour faire fonctionner un conteneur dans lequel on exécute l'application, ce qui permettrait de pallier au problème de compatibilité.

VI. Annexes

2. Règlement européen sur la protection des données : ce qui change pour les professionnels | CNIL

3. L'empreinte carbone des français, un sujet tabou ? – Ravijen

4. IEE - Tests

5. Test unitaire — Wikipédia

6. Test driven development — Wikipédia

7. Unit Test an Abstract Class