Software Maintenance and Evolution - Mission 1

Francesco Nieri & Alexandre Verlaine
Group 2

October 2023

1 Lexicon

1.1 Feature model

1.1.1 Profil

Profil: Page réservée à l'utilisateur, où il peut modifier sa photo de profil, son statut.

Photo de profil: Photo introduite par l'utilisateur pour se représenter au sein de l'application.

Statut: Texte court introduit pour l'utilisateur pour se représenter ou représenter son humeur.

1.1.2 Chat

Individuel: Chat privé entre 2 personnes seulement.

Canal: Groupe de personnes réunies, mais dont seulement certains administrateurs peuvent envoyer des messages.

Groupe: Ensemble de personnes réunies qui peuvent tous envoyé des messages dans un chat.

1.1.3 Message

Message: Un élément de texte, d'image, de vidéo ou d'autre contenu envoyé dans une conversation.

Émoji: Des petites icônes et symboles utilisés pour exprimer des émotions dans les messages.

GIF: Une image animée utilisée pour ajouter de l'humour ou de l'expression aux conversations.

Lien: Chaîne de caractère qui renvoi vers un site internet.

Médias: Les photos, vidéos, fichiers audio et autres médias partagés dans une conversation.

Fichier: Un document ou un fichier envoyé dans une conversation.

1.1.4 Appel

Appel: Lance un appel, qui peut-être vidéo, vocal ou les deux.

Appel vidéo: Appel avec une vidéo en direct des personnes impliqué.e.s dans la conversations.

Appel vocal: Appel vocal en direct des personnes impliqué.e.s dans la conversations.

1.1.5 Contact

Contacts : Les personnes que vous avez ajoutées à votre liste de contacts et avec lesquelles vous pouvez communiquer.

Ajouter un contact: Ajouter un contact dans sa liste de contact.

Supprimer un contact: Enlever un contact de sa liste de contact.

1.1.6 Générale

Notification: Une alerte qui vous informe qu'un nouveau message ou une autre activité a eu lieu dans l'application.

Privacy: Une sécurité qui garantit que seul l'expéditeur et le destinataire peuvent lire le contenu du message.

1.1.7 Thème

Thème: Le thème choisi modifie l'application, son style d'affichage et sa couleur.

Thème sombre: Mode d'affichage à faible luminosité pour réduire la fatigue oculaire lors de l'utilisation nocturne de l'application.

Thème clair: Mode d'affichage typiquement avec un fond d'écran blanc.

1.1.8 État

État: Indique si une personne est connectée ou pas.

État en ligne: Indique qu'un contact est actuellement en ligne et disponible.

État déconnecté: Indique qu'un contact n'est pas actuellement connecté.

1.1.9 Restriction

Restriction Parental: Restriction destinée aux enfants en bas âge, elle limite l'activation de certaine feature.

Restriction Viellesse: Restriction destinée aux personnes trop agées, elle limite l'activation de certaine feature.

Restriction Voiture: Restriction destinée aux personnes qui sont en voiture, elle limite l'activation de certaine feature.

1.2 Context model

1.2.1 Age

Age: Age de la personne.

Jeune: Personne ayant moins de 18 ans.

Adulte: Personne agé entre 18 et 65 ans.

Vieux: Personne ayant plus de 65 ans.

1.2.2 Activité

Actif: Personne ayant ouvert l'application.

Inactif: Personne n'étant pas sur l'application.

1.2.3 Horaire

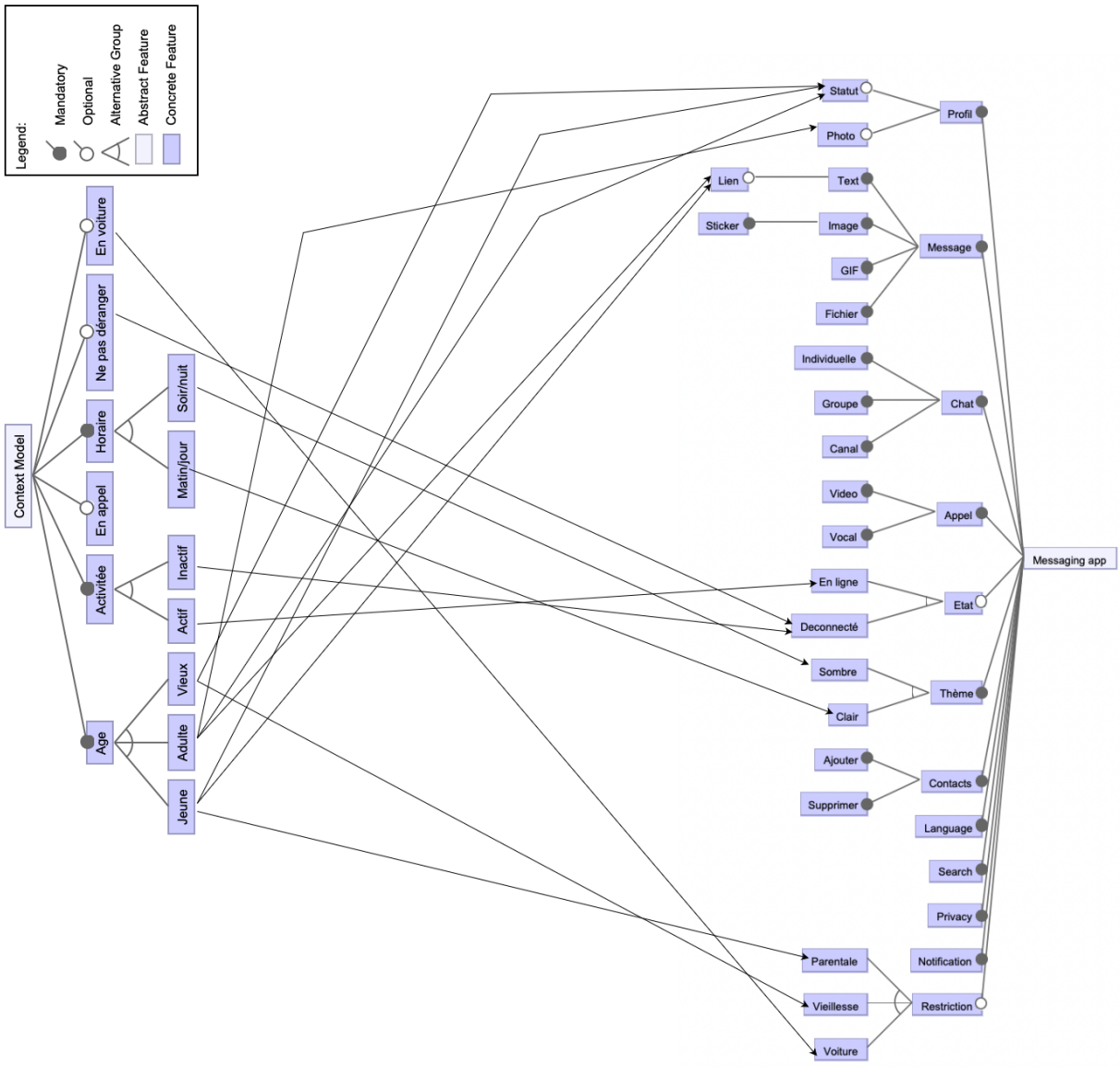
Horaire: Laps de temps durant la journée et la nuit. **Matin/jour**: Laps de temps durant la journée.

Soir/nuit: Laps de temps durant la nuit.

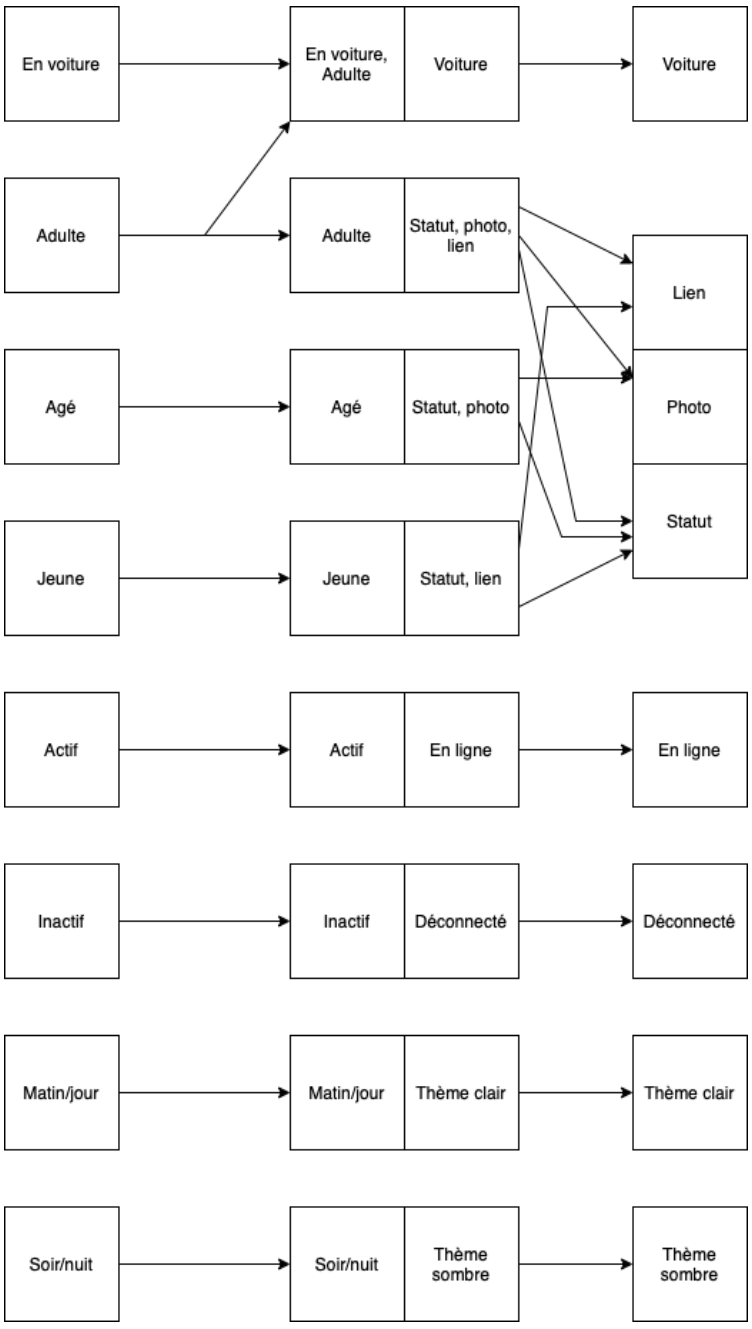
1.2.4 Other

Ne pas déranger: Mode actif qui réduit le nombre de feature. **En voiture**: Lorsque l'utilisateur est en voiture.

2 Context-feature system



3 Context-feature mapping model



4 Class diagram

Le diagramme présenté ci-dessus représente l'architecture du système de notre messagerie. Il est composé de deux principaux composants : les contrôleurs (AbstractController) et les vues (AbstractView). (Le diagramme tel que celui présenté dans les délivrables est mit en annexe du à sa densité)

Les **contrôleurs** sont responsables de la logique du système. Ils gèrent les interactions entre les utilisateurs et le système, ainsi que les opérations sur les données. Le contrôleur est représenté par une classe abstraite dans le diagramme qui est étendue par les sous classes qui définissent les différents composants du contrôleur.

Les **vues** sont responsables de l'affichage des informations aux utilisateurs. Elles fournissent une interface graphique ou textuelle pour interagir avec le système. Elles sont représentées par des classes concrètes directement liée au contrôleur afin de représenter chaque vue que nous avons dans le CLI.

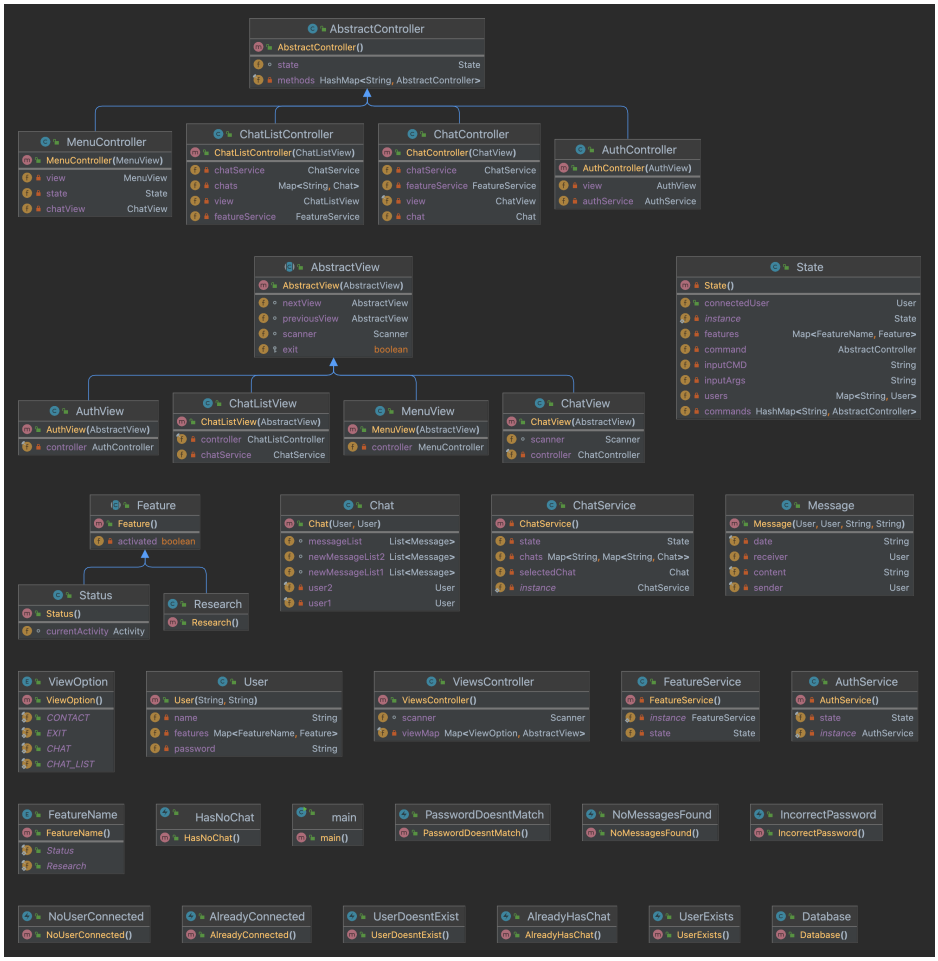
La structure est divisée en plusieurs couches :

- La couche d'abstraction contient les classes abstraites AbstractController et AbstractView. Ces classes fournissent une interface commune pour les contrôleurs et les vues.
- La couche de contrôle contient les classes concrètes ChatListController, ChatController et AuthController. Ces classes implémentent la logique dite "métier" du système.
- La couche de vue contient les classes concrètes ChatListView, ChatView et AuthView. Ces classes fournissent "l'interface graphique" (le CLI) du système.

Plus précisément les fonctions sont définies tel quel:

- **AbstractController** est une classe abstraite qui représente un contrôleur. Elle fournit une interface commune pour tous les contrôleurs.
- **ChatListController** est une classe concrète qui représente le contrôleur de la liste de chats. Elle est responsable du code logique de la liste des chats à l'utilisateur.
- **ChatController** est une classe concrète qui représente le contrôleur d'un chat. Elle est responsable du code logique qui gère les messages d'un chat à l'utilisateur.
- **AuthController** est une classe concrète qui représente le contrôleur d'authentification. Elle est responsable de l'authentification des utilisateurs.
- **AbstractView** est une classe abstraite qui représente une vue. Elle fournit une interface commune pour toutes les vues.

- **ChatListView** est une classe concrète qui représente la vue de la liste de chats. Elle fournit une interface graphique pour afficher la liste des chats à l'utilisateur ainsi que le menu d'action de ce contexte.
- **ChatView** est une classe concrète qui représente la vue d'un chat. Elle fournit une interface graphique pour afficher les messages d'un chat à l'utilisateur.
- **AuthView** est une classe concrète qui représente la vue d'authentification. Elle fournit une interface graphique pour authentifier les utilisateurs.



5 Activation/Deactivation des features

L'activation/deactivation des features se fait dans le main menu, pour l'instant, l'utilisateur est invité à activer les features qu'il veut ajouter.

Nous avons une classe abstraite Feature qui correspond à une feature générique, chaque feature implémentée héritera donc de cette classe pour avoir une classe commune à toutes les features. Chaque classe User contient une liste de ses propres features, qui peuvent etre ou pas activés.

Le processus de la deactivation est le suivant:

- Lorsque l'utilisateur souhaite activer ou désactiver une feature, il rentre un nombre en input qui correspond à une feature dans la class `menuView`.
- La vue envoie donc à son contrôleur que l'utilisateur souhaite activer/désactiver une feature, le contrôleur activera/désactivera donc la feature pour l'utilisateur connecté.
- La feature est maintenant activée/désactivée, à chaque fois que la vue doit afficher les choix possibles activées par une feature ou afficher une feature particulière, elle demande au contrôleur si elle est activée, dans ce cas, la vue affichera des informations supplémentaires, par exemple, si status est activé, la vue sait que lors du print d'un contact, elle doit ajouter son status à coté.
- Si elle est activé la view display la feature.

La feature "Research" est liée à un keybind (6) dans le choix des options pour un chat, si l'utilisateur appuie 6 lors du choix, rien ne va se passer, car le contrôleur vérifie que la feature est bien activée avant d'effectuer l'opération.

6 Principes d'architecture

L'architecture de ce système repose sur les principes suivants :

6.1 Maintenance et évolution

Comme défini plus haut, les classes abstraites fournissent les interfaces communes aux contrôleurs et aux vues, tandis que les classes concrètes `ChatListController`, `ChatController`, `AuthController`, `ChatListView`, `ChatView`, et `AuthView` implémentent ces interfaces. Cette séparation permet de faciliter la compréhension et la maintenance du code.

6.2 Division du système en couches

Vu que le système est divisé en trois couches : la couche d'abstraction, la couche de contrôle, et la couche de vue, cela permet de limiter les interactions entre

les différentes parties du système, ce qui facilite la modification ou l'extension d'une partie du système sans affecter les autres parties.

7 Avantages de cette architecture

Ces principes permettent de réaliser les avantages suivants :

7.1 Réduction de la duplication de code

Les classes concrètes peuvent hériter des méthodes et des propriétés des classes abstraites. Cela permet de réduire la duplication de code et d'améliorer la cohérence du code.

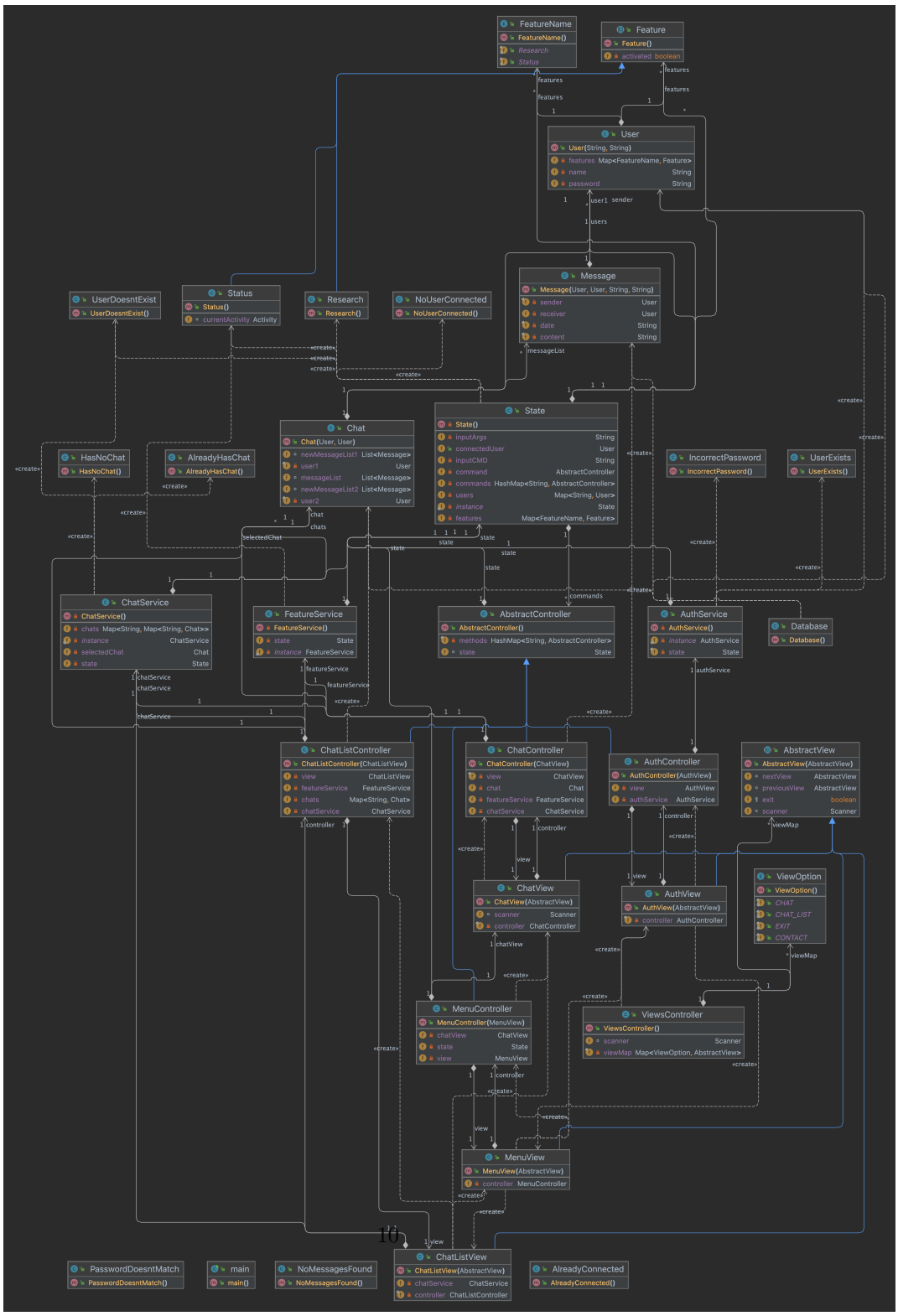
7.2 Facilité d'ajout de nouvelles fonctionnalités

De nouvelles fonctionnalités peuvent être ajoutées en créant de nouvelles classes concrètes. Cela permet de faciliter l'ajout de nouvelles fonctionnalités sans affecter le code existant.

7.3 Amélioration de la lisibilité du code

Les classes abstraites fournissent une documentation implicite des fonctionnalités que doivent fournir les classes concrètes. Cela permet d'améliorer la lisibilité du code et de faciliter la compréhension du fonctionnement du système.

8 Annexe


FeatureName
FeatureName()
Research
Status
Feature
Feature()
activated boolean
features
User
User(String, String)
features Map<FeatureName, Feature>
name String
password String
user1 sender
users
Message
Message(User, User, String, String)
sender User
receiver User
date String
content String
messageList
UserDoesntExist
UserDoesntExist()
Status
Status()
currentActivity Activity
Research
Research()
NoUserConnected
NoUserConnected()
«create»
State
State()
inputArgs String
connectedUser User
inputCMD String
command AbstractController
commands HashMap<String, AbstractController>
users Map<String, User>
instance State
features Map<FeatureName, Feature>
Chat
Chat(User, User)
newMessageList1 List<Message>
user1 User
messageList List<Message>
newMessageList2 List<Message>
user2 User
chat
chats
selectedChat
HasNoChat
HasNoChat()
AlreadyHasChat
AlreadyHasChat()
IncorrectPassword
IncorrectPassword()
UserExists
UserExists()
state
commands
ChatService
ChatService()
chats Map<String, Map<String, Chat>>
instance ChatService
selectedChat Chat
state State
FeatureService
FeatureService()
state State
instance FeatureService
featureService
AbstractController
AbstractController()
methods HashMap<String, AbstractController>
state State
AuthService
AuthService()
instance AuthService
state State
authService
Database
Database()
ChatListController
ChatListController(ChatListView)
view ChatListView
featureService FeatureService
chats Map<String, Chat>
chatService ChatService
controller
ChatController
ChatController(ChatView)
view ChatView
chat Chat
featureService FeatureService
chatService ChatService
AuthController
AuthController(AuthView)
view AuthView
authService AuthService
AbstractView
AbstractView(AbstractView)
nextView AbstractView
previousView AbstractView
exit boolean
scanner Scanner
viewMap
ChatView
ChatView(AbstractView)
scanner Scanner
controller ChatController
chatView
AuthView
AuthView(AbstractView)
controller AuthController
ViewOption
ViewOption()
CHAT
CHAT_LIST
EXIT
CONTACT
MenuController
MenuController(MenuView)
chatView ChatView
state State
view MenuView
ViewsController
ViewsController()
scanner Scanner
viewMap Map<ViewOption, AbstractView>
MenuView
MenuView(AbstractView)
controller MenuController
ChatListView
ChatListView(AbstractView)
chatService ChatService
controller ChatListController
PasswordDoesntMatch
PasswordDoesntMatch()
main
main()
NoMessagesFound
NoMessagesFound()
AlreadyConnected
AlreadyConnected()