Normes de Codage

Auteurs: Équipe AP4
Date de création: 03/11/2024
Dernière mise à jour: 03/11/2024

Sommaire

Architecture et organisation du code

Structure du projet

Le projet est organisé en trois modules Maven distincts, suivant une architecture modulaire :

- **chat-common** : Contient les modèles, DTOs, énumérations et autres classes partagées entre le client et le serveur
- **chat-client** : Application cliente JavaFX avec interface utilisateur
- **chat-server** : Serveur Spring Boot pour la gestion des connexions WebSocket et la logique métier

Architecture client

L'application cliente suit le modèle **MVC (Modèle-Vue-Contrôleur)** :

- **Modèle** : Représente les données et la logique métier (classes dans les packages `models`, `dto`)
- **Vue** : Interface utilisateur définie dans les fichiers FXML
- **Contrôleur** : Gère les interactions entre le modèle et la vue (classes dans le package `controllers`)

En plus de cette séparation, l'application utilise :

- **Services** : Encapsulent la logique métier et l'accès aux ressources
- **DAO (Data Access Objects)** : Gèrent l'accès à la base de données
- **Scènes** : Encapsulent les différentes interfaces utilisateur

Architecture serveur

Le serveur est basé sur Spring Boot et utilise :

- **Controllers** : Gèrent les points d'entrée WebSocket et REST
- **Services** : Contiennent la logique métier
- **Listeners** : Réagissent aux événements WebSocket
- **Configuration** : Définit le comportement de l'application

Conventions de nommage

Packages

- Noms en minuscules
- Notation en arborescence (`com.ap4.client.controllers`)
- Organisation par fonctionnalité ou couche technique

Classes

- Notation **PascalCase** (première lettre de chaque mot en majuscule)
- Noms significatifs reflétant la responsabilité de la classe
- Exemples : `UserController`, `ChatService`, `MessageDAO`

Interfaces

- Préfixe "I" pour les interfaces (ex: `IUserService`)
- Notation PascalCase
- Décrivent un comportement ou un contrat

Méthodes

- Notation **camelCase** (première lettre en minuscule, puis majuscule pour chaque nouveau mot)
- Verbes d'action décrivant ce que fait la méthode
- Exemples : `getUserById()`, `sendMessage()`, `createNewChat()`

Variables

- Notation camelCase
- Noms descriptifs évitant les abréviations obscures
- Exemples : `userName`, `messageContent`, `isConnected`

Constantes

- Notation **SNAKE_CASE** en majuscules
- Mots séparés par des underscores
- Exemples : `MAX_MESSAGE_LENGTH`, `DEFAULT_PORT`

Fichiers FXML

- Notation camelCase
- Décrivent l'écran ou le composant qu'ils représentent
- Exemples : `login.fxml`, `chatListCell.fxml`

Formatage et style

Indentation et espacement

- Indentation de 4 espaces (pas de tabulations)
- Limite de 120 caractères par ligne

- Espacement autour des opérateurs (`x + y` plutôt que `x+y`)
- Espacement après les virgules (`method(a, b, c)` plutôt que `method(a,b,c)`)

Accolades

- Style Java standard :

```
if (condition) {
    // Code
} else {
    // Code
}
```

Imports

- Pas d'imports avec wildcard (`import java.util.*`)
- Imports organisés par groupe (Java standard, bibliothèques externes, packages internes)
- Élimination des imports non utilisés

Documentation du code

Javadoc

- Documentation Javadoc pour toutes les classes publiques et protégées
- Documentation des méthodes publiques avec description, paramètres et valeurs de retour
- Exemple :

```
/**
 * Envoie un message à un destinataire spécifié.
 *
 * @param sender L'expéditeur du message
 * @param recipient Le destinataire du message
 * @param content Le contenu du message
 * @return Le message créé avec son ID généré
 * @throws MessageTooLongException Si le contenu du message dépasse la
longueur maximale
 */
public Message sendMessage(User sender, User recipient, String content)
throws MessageTooLongException {
    // Implémentation
}
```

Commentaires

- Commentaires significatifs expliquant le "pourquoi" et non le "comment"
- Commentaires pour le code complexe ou non intuitif
- TODO et FIXME clairement marqués pour les améliorations futures

Bonnes pratiques et principes

Principes SOLID

- **S** (Single Responsibility) : Une classe ne doit avoir qu'une seule responsabilité
- **O** (Open/Closed) : Les entités doivent être ouvertes à l'extension mais fermées à la modification
- **L** (Liskov Substitution) : Les objets d'une classe dérivée doivent pouvoir remplacer les objets de la classe de base
- **I** (Interface Segregation) : Préférer plusieurs interfaces spécifiques plutôt qu'une interface générique
- **D** (Dependency Inversion) : Dépendre des abstractions, pas des implémentations concrètes

Autres principes

- **DRY** (Don't Repeat Yourself) : Éviter la duplication de code
- **KISS** (Keep It Simple, Stupid) : Privilégier la simplicité et la lisibilité
- **Fail-Fast** : Détecter et signaler les erreurs dès que possible

Gestion des erreurs

- Utilisation appropriée des exceptions pour les cas d'erreur
- Exceptions métier personnalisées pour les cas spécifiques
- Journalisation cohérente des erreurs avec Log4j

Tests

Types de tests

- **Tests unitaires** : Pour les composants individuels
- **Tests d'intégration** : Pour vérifier l'interaction entre composants

Conventions pour les tests

- Classes de test nommées avec le suffixe "Test" (ex: `UserServiceTest`)
- Organisation des tests selon la méthodologie AAA (Arrange, Act, Assert)
- Un test par comportement ou scénario

Couverture de code

- Objectif de couverture de code d'au moins 70%
- Utilisation de JaCoCo pour mesurer la couverture
- Attention particulière aux chemins d'erreur et cas limites