

# Rapport Technique

## Plateforme de Discussion PHP/XML

### Équipe du projet

Mariama Baldé (INFORMATIQUE)

Elhadji Saloum Cissé (TÉLÉCOMS ET RÉSEAUX)

Mouhamed Lamine Faye (INFORMATIQUE)

Cheikh Ahmed Tidiane Thiadoun (INFORMATIQUE)

### Soumis à

Prof. Ibrahima Fall

15 juillet 2025

### Résumé

Ce document constitue le rapport technique d'une application de messagerie instantanée. Développée en PHP, l'application utilise exclusivement des fichiers XML pour la persistance des données, conformément aux exigences du projet. L'objectif de ce rapport est de documenter de manière détaillée le processus de conception et de développement logiciel. L'analyse couvre l'architecture technique, la conception orientée objet, le modèle de données XML, ainsi que les mécanismes de validation mis en œuvre.

## Table des matières

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                        | <b>3</b>  |
| <b>2</b> | <b>Cahier des charges fonctionnel</b>      | <b>4</b>  |
| <b>3</b> | <b>Processus de développement logiciel</b> | <b>5</b>  |
| <b>4</b> | <b>Conception orientée objet</b>           | <b>6</b>  |
| <b>5</b> | <b>Mapping Objet-XML</b>                   | <b>7</b>  |
| <b>6</b> | <b>Définition des DTD</b>                  | <b>9</b>  |
| <b>7</b> | <b>Architecture technique</b>              | <b>10</b> |
| <b>8</b> | <b>Tests et validation</b>                 | <b>12</b> |
| <b>9</b> | <b>Bilan technique et perspectives</b>     | <b>13</b> |
|          | <b>Annexes</b>                             | <b>14</b> |
| <b>A</b> | <b>DTD (Document Type Definitions)</b>     | <b>14</b> |
| <b>B</b> | <b>Diagrammes UML</b>                      | <b>19</b> |
| <b>C</b> | <b>Vues de l'application</b>               | <b>20</b> |
|          | <b>Bibliographie</b>                       | <b>24</b> |

# 1 Introduction

Le présent document constitue le rapport technique final pour le projet de développement d'une plateforme de discussion en ligne. Ce projet s'inscrit dans le cadre de l'unité d'enseignement sur les technologies XML et a pour objectif principal la mise en œuvre d'une application web dynamique où la persistance des données est entièrement gérée par des fichiers XML, sans recours à un système de gestion de base de données traditionnel.

L'application, développée en PHP, simule une messagerie instantanée permettant aux utilisateurs de s'inscrire, de se connecter, d'échanger des messages privés, de créer et de rejoindre des groupes de discussion. L'un des enjeux majeurs du projet réside dans la conception d'un modèle de données XML robuste et la validation de sa structure via des DTD (Document Type Definitions), garantissant ainsi l'intégrité et la cohérence des informations stockées.

Ce rapport a pour vocation de détailler l'ensemble du processus de développement, depuis l'analyse des besoins fonctionnels jusqu'à l'implémentation technique. Il présente l'architecture logicielle retenue, la conception orientée objet des composants PHP, le mapping entre les objets et la structure XML, ainsi que les défis techniques rencontrés et les solutions apportées. L'objectif est de fournir une vision claire et complète du travail réalisé, tout en justifiant les choix de conception et de technologie.

## 2 Cahier des charges fonctionnel

L'application doit répondre à un ensemble de besoins fonctionnels qui définissent son périmètre et ses capacités. Ces fonctionnalités sont présentées dans le tableau ci-dessous, accompagnées d'une brève description. Chaque fonctionnalité majeure est associée à un diagramme de cas d'utilisation, disponible en annexe, qui modélise les interactions entre les acteurs (utilisateurs) et le système.

| Cas d'utilisation             | Description  |
|-------------------------------|--|
| Gestion de l'authentification | L'utilisateur doit pouvoir se créer un compte, se connecter et se déconnecter. La sécurité des mots de passe doit être assurée par un mécanisme de hachage.              |
| Gestion du profil             | L'utilisateur doit pouvoir consulter et mettre à jour les informations de son profil (nom, avatar, biographie, etc.).  |
| Messagerie privée             | Un utilisateur connecté doit pouvoir envoyer et recevoir des messages textuels à un autre utilisateur. L'historique de la conversation doit être conservé et affiché.    |
| Gestion des groupes           | Un utilisateur doit pouvoir créer un groupe de discussion, inviter d'autres membres, et y envoyer des messages. Les administrateurs du groupe peuvent gérer les membres. |
| Recherche                     | L'application doit fournir une fonctionnalité de recherche permettant de retrouver des utilisateurs ou des messages.   |
| Validation des données        | Toutes les données stockées doivent être validées par une DTD pour garantir leur intégrité structurelle.   |

TABLE 1 – Tableau récapitulatif des cas d'utilisation.

Une modélisation visuelle via des diagrammes de cas d'utilisation UML permettrait d'illustrer plus en détail les interactions entre les acteurs (utilisateur, administrateur) et le système pour chacun de ces points.

Un flux utilisateur typique, comme **l'envoi d'un message privé**, se décompose comme suit :

1. L'utilisateur authentifié accède à sa liste de contacts.
2. Il sélectionne un contact, ce qui ouvre une interface de conversation.
3. Il saisit son message dans un champ de texte et le soumet.
4. Le système, via la classe 'Message', instancie un nouvel objet message contenant l'ID de l'expéditeur, l'ID du destinataire, le contenu textuel et un horodatage ('timestamp').
5. Avant la sauvegarde, la classe 'Database' charge le fichier 'messages.xml', y ajoute

le nouveau nœud ‘<message>’, puis valide l’intégralité du document modifié contre ‘messages.dtd’.

6. Si la validation réussit, le fichier est sauvegardé ; sinon, une erreur est retournée.

### 3 Processus de développement logiciel

Pour ce projet, une **approche de développement itérative et incrémentale**, inspirée des méthodologies Agiles, a été adoptée. Plutôt qu’un long cycle en V, le projet a été construit par blocs fonctionnels successifs.

Le plan de développement a été structuré en plusieurs itérations (ou sprints) :

- **Itération 1 : Socle de l’application.** Mise en place de l’architecture (contrôleur frontal), de la classe ‘Database’ pour l’accès aux données XML, et développement du module de gestion des utilisateurs (‘User.php’) avec inscription et connexion. L’objectif est d’avoir un utilisateur capable de s’authentifier.
- **Itération 2 : Messagerie privée.** Développement de la classe ‘Message.php’ et des interfaces associées pour permettre l’échange de messages entre deux utilisateurs. Création du ‘messages.dtd’.
- **Itération 3 : Gestion des groupes.** Développement de la classe ‘Group.php’ pour la création de groupes et la gestion des membres. Adaptation de la classe ‘Message’ pour gérer les messages de groupe. Création du ‘groups.dtd’.
- **Itération 4 : Fonctionnalités sociales.** Ajout de la gestion des contacts (‘Contact.php’), des profils utilisateurs, et des paramètres.

Chaque itération suit un mini-cycle Analyse-Conception-Implémentation-Validation, permettant de livrer une fonctionnalité complète et testable à chaque étape.

- **Analyse :** Définition des entités métier (‘User’, ‘Message’, ‘Group’, ‘Contact’) et de leurs interactions.
- **Conception :** Modélisation orientée objet avec des diagrammes de classes UML pour représenter ces entités et leurs relations. Conception du schéma de données XML (DTD).
- **Implémentation :** Développement en PHP 8+, en s’appuyant sur l’extension SimpleXML pour la manipulation des données XML. L’interface est réalisée en HTML, CSS (Bootstrap) et JavaScript.
- **Validation :** Tests fonctionnels manuels, validation systématique des données XML via les DTD avant chaque écriture, et mise en place d’un embryon de tests unitaires.

## 4 Conception orientée objet

L'application repose sur une conception orientée objet robuste où chaque classe encapsule une responsabilité métier unique, respectant ainsi le **principe de responsabilité unique (SRP)**. Cette approche favorise la modularité, la maintenabilité et la testabilité du code.

- **User.php** : Gère tout ce qui concerne l'utilisateur (authentification, profil, session).
- **Message.php** : Responsable de l'envoi, la réception et la gestion des messages.
- **Group.php** : Gère la logique des groupes de discussion.
- **Contact.php** : Gère la liste de contacts d'un utilisateur.
- **Database.php** : Agit comme une couche d'abstraction pour l'accès aux données (Data Access Layer), centralisant la lecture, l'écriture et la validation des fichiers XML.

La classe 'Database' agit comme une **couche d'accès aux données (Data Access Layer)**, offrant un service de persistance aux autres classes métier. Ces dernières ne manipulent jamais directement les fichiers XML, mais délèguent ces opérations à 'Database', ce qui constitue une excellente pratique de conception.

Les relations entre les classes métier sont principalement des associations. Par exemple, un 'User' est associé à plusieurs 'Message's (en tant qu'expéditeur) et peut être membre de plusieurs 'Group's. La relation entre 'User' et son 'Profile' est une **composition** : le profil n'existe pas sans l'utilisateur. De même, un 'Group' est composé de 'Member's.

Le diagramme de classes ci-dessous (Figure 1) illustre ces relations. La notation utilisée est standard UML : les lignes pleines avec une flèche représentent une association dirigée, et les lignes pointillées une dépendance (par exemple, une classe en utilise une autre sans la stocker comme attribut).

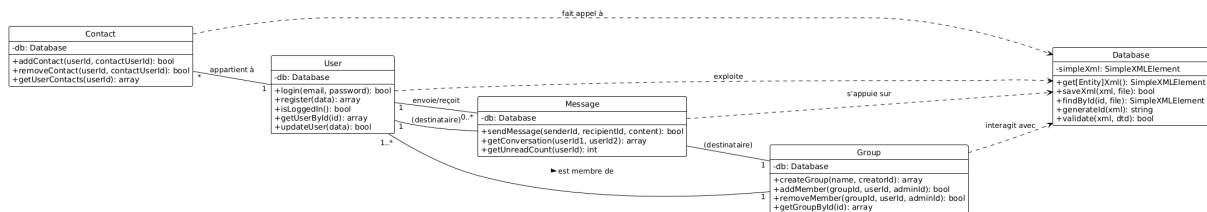


FIGURE 1 – Diagramme de classes de l'application.

## 5 Mapping Objet-XML

Le passage du modèle objet (classes PHP) au modèle de données hiérarchique (XML) est une étape fondamentale de la conception. Ce mapping objet-XML a été implémenté en suivant un ensemble de règles de transformation directes et intuitives :

- Une instance de classe est mappée à un élément XML principal (e.g., un objet ‘User’ devient un élément ‘<user>’).
- Les attributs de la classe (propriétés scalaires) deviennent des sous-éléments XML (e.g., la propriété ‘username’ devient ‘<username> ... </username>’). Les relations de composition deviennent ‘dans’ ‘<user>’).
- Les collections d’objets (listes) sont mappées à une série d’éléments XML (e.g., une liste de membres dans un groupe devient ‘<members><member.../><member.../></members>’).

### Exemple de correspondance : Classe ‘User’ et DTD

La classe ‘User’ en PHP contient des propriétés comme ‘id’, ‘username’, ‘email’, etc. Cette structure est directement traduite dans le ‘users.dtd’.

```

1 class User {
2     private string $id;
3     private string $username;
4     private string $email;
5     // ...
6 }
```

Listing 1 – Extrait simplifié de la classe User

Cette classe correspond à la structure XML validée par la DTD suivante :

```

1 <!ELEMENT user (username, email, ...) >
2 <!ATTLIST user id ID #REQUIRED>
3 <!ELEMENT username (#PCDATA)>
```

Listing 2 – Extrait de users.dtd

Plusieurs défis techniques émergent de ce mapping :

- **Perte du typage fort** : XML ne dispose pas d’un système de types aussi riche que PHP. Toutes les données issues d’un nœud textuel sont des chaînes de caractères (‘PCDATA’). La couche applicative doit donc systématiquement effectuer des conversions de type (e.g., ‘(bool)’, ‘(int)’) et des validations sémantiques (e.g., ‘filter\_var(email, FILTER\_VALIDATE\_EMAIL)’).
- **Gestion des relations** : Contrairement aux S...
- **Choix entre élément et attribut** : Une décision de conception récurrente a été de déterminer si une donnée devait être un sous-élément ou un attribut. La convention observée est que les métadonnées, les identifiants et les énumérations (‘id’, ‘role’, ‘status’) sont des attributs pour leur concision, tandis que les données de contenu (‘username’, ‘content’) sont des éléments.

Le tableau suivant synthétise ce mapping pour l'entité 'User' :

| Classe PHP             | Élément XML                | Règle DTD                             |
|------------------------|----------------------------|---------------------------------------|
| 'User' (objet)         | '<user>'                   | 'ELEMENT user (username, email, ...)' |
| 'id' (propriété)       | '<user id="...">'          | 'ATTLIST user id ID REQUIRED'         |
| 'username' (propriété) | '<username>...</username>' | 'ELEMENT username (PCDATA)'           |
| 'email' (propriété)    | '<email>...</email>'       | 'ELEMENT email (PCDATA)'              |

TABLE 2 – Exemple de mapping Objet-XML-DTD pour un utilisateur.



## 6 Définition des DTD

La validation des données est une pierre angulaire du projet, assurée par un ensemble de DTD. Chaque DTD agit comme un contrat formel définissant la grammaire du document XML associé. Elle spécifie les éléments autorisés, leur ordre (séquence ‘a, b, c’), leur cardinalité (zéro ou un ‘*i*’, un ou plusieurs ‘+’, zéro ou plusieurs ‘\*’) et les attributs possibles, y compris leur type (‘CDATA’, ‘ID’, etc.) et leur caractère obligatoire (‘REQUIRED’, ‘IMPLIED’).

Les choix de structure dans les DTD sont pragmatiques. Par exemple, dans ‘groups.dtd’, l’élément ‘<member>’ est déclaré ‘EMPTY’ car toutes ses informations (‘*user<sub>i</sub>d*’, ‘*role*’) sont stockées dans

La validation est mise en œuvre de manière programmatique dans la classe ‘Database’. Avant chaque opération d’écriture, le contenu du fichier XML est chargé dans un objet ‘DOMDocument’, et la méthode ‘*dom->validate()*’ est appelée. Cette méthode vérifie la conformité du document à la DTD. *!DOCTYPE > ‘.Cela prévient la corruption des fichiers de données par des écritures invalides.*

Par exemple, si une tentative d’écriture omettait la balise ‘<username>’ obligatoire pour un utilisateur, la méthode ‘*dom->validate()*’ retournerait ‘false’ et lancerait une erreur, empêchant

Il est à noter que si les DTD sont efficaces pour la validation structurelle, elles présentent des limites, notamment l’absence de typage de données précis (tout est chaîne de caractères). Une migration vers **XML Schema (XSD)** aurait permis de définir des contraintes plus fines (e.g., un ‘timestamp’ doit être une date valide, un ‘id’ doit suivre un certain pattern).

## 7 Architecture technique

L'application est bâtie sur une architecture multi-couches classique, qui sépare la présentation, la logique métier et l'accès aux données. Cette séparation des préoccupations est essentielle pour la maintenabilité et l'évolutivité du système.

L'interface utilisateur a été développée avec le framework front-end **Bootstrap 5** pour assurer un design responsive et moderne, et utilise la bibliothèque **FontAwesome** pour les icônes. Les interactions dynamiques sont gérées via des requêtes AJAX simples avec du JavaScript natif.

Un diagramme de composants UML permettrait de visualiser l'architecture physique, en montrant le client (navigateur), le serveur web Apache exécutant le code PHP, et la couche de persistance constituée des fichiers XML.

### Arborescence des dossiers

```
/
|-- assets/ (CSS, JS, images)
|-- config/ (config.php)
|-- data/ (users.xml, messages.xml, ...)
|-- includes/ (classes PHP: User, Message, ...)
|-- pages/ (vues: dashboard.php, login.php, ...)
|-- schemas/ (users.dtd, messages.dtd, ...)
|-- tests/ (fichiers de test)
|-- index.php (Contrôleur frontal)
|-- Dockerfile
```

### Architecture logique

- **Couche de présentation (Frontend)** : Gérée par les fichiers dans 'pages/' et 'assets/'. C'est ce que l'utilisateur voit et avec quoi il interagit.
- **Couche de logique métier (Backend)** : Implémentée par les classes PHP dans 'includes/'. Elle contient les règles et les processus de l'application.
- **Couche d'accès aux données** : Centralisée dans la classe 'Database.php', elle gère toutes les interactions avec les fichiers XML dans 'data/'.
- **Contrôleur frontal ('index.php')** : L'application implémente le patron de conception **Front Controller**. Toutes les requêtes HTTP sont redirigées (via la configuration du serveur web, e.g., '.htaccess') vers 'index.php'. Ce script unique a plusieurs responsabilités : initialiser l'environnement (chargement de la configuration, démarrage des sessions), instancier les classes de service, et router la requête vers le script de page approprié dans le dossier 'pages/' en fonction des paramètres GET.

Il gère également la logique de protection des pages, n'autorisant l'accès qu'aux utilisateurs authentifiés.

## 8 Tests et validation

La stratégie de validation du projet est multi-niveaux. Des tests unitaires ont été menés pour valider le comportement de chaque méthode critique des classes métier. Le tableau ci-dessous présente quelques exemples de cas de test.

| Méthode testée             | Cas d'entrée             | Résultat attendu                 |
|----------------------------|--------------------------|----------------------------------|
| 'User : :login()'          | Identifiants valides     | Retourne 'true', session ouverte |
| 'User : :login()'          | Mot de passe incorrect   | Retourne 'false', pas de session |
| 'Message : :sendMessage()' | Contenu non vide         | Message sauvegardé en XML        |
| 'Database : :validate()'   | Fichier XML conforme DTD | Retourne 'true'                  |

TABLE 3 – Exemples de tests unitaires.

Un exemple de test d'assertion en pseudo-code PHPUnit ressemblerait à ceci :

```
public function testLoginWithValidCredentials() {  
    $user = new User();  
    $this->assertTrue($user->login('testuser', 'password123'));  
}
```

## 9 Bilan technique et perspectives

### Avantages et inconvénients du stockage XML

L'utilisation de XML comme système de persistance principal est une décision de conception forte avec des conséquences importantes.

| Avantages                                    | Inconvénients                             |
|--|---|
| Format lisible par l'humain                  | Verbosité du format                       |
| Interopérabilité et standardisation (W3C)    | Performance des requêtes sur gros volumes |
| Validation de structure via DTD/XSD          | Absence de gestion des transactions       |
| Bon support par les langages (PHP SimpleXML) | Gestion complexe des accès concurrents    |

TABLE 4 – Bilan de l'utilisation de XML pour la persistance.

### Perspectives d'amélioration

Malgré ses limites dans ce contexte, le projet constitue une excellente base. Pour le faire évoluer vers une application de production, plusieurs pistes sont envisageables :

- **Migration vers une base de données XML native** : Des systèmes comme BaseX ou eXist-db conserveraient le modèle de données XML tout en offrant des performances optimisées, l'indexation, et un langage de requêtage puissant (XQuery).
- **Passage à une base de données NoSQL** : Une base de données orientée document comme MongoDB serait une transition naturelle, car son modèle (collections de documents JSON/BSON) est conceptuellement proche de la structure XML du projet.
- **Amélioration de la sécurité** : Mettre en place des mesures plus robustes comme des Content Security Policies (CSP), des jetons CSRF, et une gestion des sessions plus fine.

## A DTD (Document Type Definitions)

### users.dtd

```

1 <!-- DTD pour le fichier des utilisateurs (users.xml) -->
2 <!DOCTYPE users [
3     <!-- L' élément racine <users> peut contenir plusieurs éléments <
4     user> -->
5     <!ELEMENT users (user*)>
6
7     <!-- Structure d'un utilisateur. 'last_login' est optionnel ('?') --
8     >
9     <!ELEMENT user (username, email, password_hash, profile, settings,
10    created_at, last_login?)>
11
12     <!-- Attributs pour l' élément <user> -->
13     <!ATTLIST user
14         id CDATA #REQUIRED -- Identifiant
15         unique textuel
16         status (online|offline|away|busy) "offline" -- Statut de
17         pr sence, 'offline' par d faut
18         role (user|admin|moderator) "user" -- R le de l'
19         utilisateur, 'user' par d faut
20         is_active (true|false) "true" -- Compte actif ou
21         non
22         is_verified (true|false) "false" -- Compte v rifi
23         ou non
24     >
25
26     <!-- Éléments de base de l'utilisateur -->
27     <!ELEMENT username (#PCDATA)>
28     <!ELEMENT email (#PCDATA)>
29     <!ELEMENT password_hash (#PCDATA)>
30     <!ELEMENT created_at (#PCDATA)>
31     <!ELEMENT last_login (#PCDATA)>
32
33     <!-- Sous-structure pour le profil, 'avatar' et 'bio' sont
34     optionnels -->
35     <!ELEMENT profile (first_name, last_name, avatar?, bio?)>
36     <!ELEMENT first_name (#PCDATA)>
37     <!ELEMENT last_name (#PCDATA)>
38     <!ELEMENT avatar (#PCDATA)>
39     <!ELEMENT bio (#PCDATA)>
40
41     <!-- Sous-structure pour les paramètres de l'utilisateur -->
42     <!ELEMENT settings (notifications, theme, language, privacy_level)>
43     <!ELEMENT notifications (#PCDATA)>

```

```

35 <!-- ELEMENT theme (#PCDATA)>
36 <!-- ELEMENT language (#PCDATA)>
37 <!-- ELEMENT privacy_level (#PCDATA)>
38 ]>

```

Listing 3 – schemas/users.dtd

**messages.dtd**

```

1 <!-- DTD pour le fichier des messages (messages.xml) -->
2 <!-- DOCTYPE messages [
3   <!-- L' lment racine <messages> contient une liste de <message>
   -->
4   <!-- ELEMENT messages (message*)>
5
6   <!-- Structure d'un message. 'attachments' et 'is_read' sont
   optionnels -->
7   <!-- ELEMENT message (sender_id, recipient_type, recipient_id, content,
   timestamp, attachments?, is_read?)>
8
9   <!-- Attributs pour l' lment <message> -->
10  <!-- ATTLIST message
11   id CDATA #REQUIRED -- Identifiant unique du
   message
12   type (text|file|image|system) "text" -- Type de message, '
   text' par d faut
13   is_edited (true|false) "false" -- Indique si le message
   a t dit
14   is_deleted (true|false) "false" -- Indique si le message
   a t supprim (soft delete)
15   >
16
17   <!-- lments de base du message -->
18   <!-- ELEMENT sender_id (#PCDATA)>
19   <!-- ELEMENT recipient_type (user|group)> -- Le destinataire est
   soit un 'user', soit un 'group'
20   <!-- ELEMENT recipient_id (#PCDATA)>
21   <!-- ELEMENT content (#PCDATA)>
22   <!-- ELEMENT timestamp (#PCDATA)>
23   <!-- ELEMENT is_read (#PCDATA)>
24
25   <!-- Structure pour les pi ces jointes, peut contenir plusieurs <
   file> -->
26   <!-- ELEMENT attachments (file*)>
27   <!-- ELEMENT file (#PCDATA)> -- Contient le chemin ou
   l'URL du fichier
28

```

```

29 <!-- Attributs pour une pi ce jointe <file> -->
30 <!ATTLIST file
31     name CDATA #REQUIRED           -- Nom du fichier
32     size CDATA #IMPLIED           -- Taille du fichier (
optionnel)
33     type CDATA #IMPLIED           -- Type MIME du fichier
(optionnel)
34 >
35 ]>

```

Listing 4 – schemas/messages.dtd

## groups.dtd

```

1 <!-- DTD pour le fichier des groupes (groups.xml) -->
2 <!DOCTYPE groups [
3     <!-- L' lment racine <groups> contient une liste de <group> -->
4     <!ELEMENT groups (group*)>
5
6     <!-- Structure d'un groupe. 'settings' est optionnel -->
7     <!ELEMENT group (name, description, created_by, created_at, members,
settings?)>
8
9     <!-- Attributs pour l' lment <group> -->
10    <!ATTLIST group
11        id CDATA #REQUIRED           -- Identifiant unique du
groupe
12        is_active (true|false) "true" -- Indique si le groupe
est actif
13    >
14
15    <!-- lments de base du groupe -->
16    <!ELEMENT name (#PCDATA)>
17    <!ELEMENT description (#PCDATA)>
18    <!ELEMENT created_by (#PCDATA)>           -- ID de l'utilisateur
cr ateur
19    <!ELEMENT created_at (#PCDATA)>
20
21    <!-- Liste des membres du groupe -->
22    <!ELEMENT members (member*)>
23    <!ELEMENT member EMPTY>           -- L' lment membre
est vide, ses infos sont dans les attributs
24
25    <!-- Attributs pour un <member> -->
26    <!ATTLIST member
27        user_id CDATA #REQUIRED           -- ID de l'utilisateur
membre

```



```

28     role (admin|moderator|member) "member" -- R le dans le groupe,
    'member' par d faut
29     joined_at CDATA #IMPLIED -- Date d'adh sion (
optionnel)
30 >
31
32 <!-- Param tres du groupe, 'max_members' est optionnel -->
33 <!ELEMENT settings (privacy, notifications, max_members?)>
34 <!ELEMENT privacy (public|private|secret)>
35 <!ELEMENT notifications (true|false)>
36 <!ELEMENT max_members (#PCDATA)>
37 ]>

```

Listing 5 – schemas/groups.dtd

## Annexe B : Exemples de Données XML

*Cette annexe présente des extraits des fichiers de données XML pour illustrer la structure et le type de données gérées par l'application.*

### Exemple de users.xml

```

1 <!-- Fichier contenant les informations des utilisateurs -->
2 <users>
3   <!-- Exemple d'un utilisateur avec le r le 'admin' -->
4   <user id="1" status="online" role="admin" is_active="true" is_verified
    ="true">
5     <username>admin</username>
6     <email>admin@messaging.com</email>
7     <password_hash>$2y$10$...</password_hash> <!-- Mot de passe hach
    -->
8     <profile>
9       <first_name>Administrateur</first_name>
10      <last_name>Syst me</last_name>
11      <avatar>admin_avatar.jpg</avatar>
12    </profile>
13    <created_at>2024-01-01T00:00:00</created_at>
14    <last_login>2025-07-15T01:35:42</last_login>
15  </user>
16  <!-- Exemple d'un utilisateur standard -->
17  <user id="2" status="offline" role="user" is_active="true" is_verified
    ="true">
18    <username>mariama</username>
19    <email>mariama@messaging.com</email>
20    <password_hash>$2y$10$...</password_hash>
21    <profile>

```

```

22     <first_name>mariama</first_name>
23     <last_name/>
24 </profile>
25     <created_at>2024-01-01T00:00:00</created_at>
26     <last_login>2025-07-15T03:41:13</last_login>
27 </user>
28 </users>

```

Listing 6 – Extrait de data/users.xml

### Exemple de messages.xml

```

1 <!-- Fichier contenant les messages      changs      -->
2 <messages>
3   <!-- Message de type 'file' envoy      un groupe -->
4   <message id="4" type="file" is_edited="false" is_deleted="false">
5     <sender_id>1</sender_id>
6     <recipient_type>group</recipient_type>
7     <recipient_id>1</recipient_id>
8     <content>Document de pr sentation du projet</content>
9     <timestamp>2024-01-15T09:00:00</timestamp>
10    <attachments>
11      <file name="presentation.pdf" size="2048576" type="application/pdf
12      ">presentation.pdf</file>
13    </attachments>
14    <is_read>true</is_read>
15  </message>
16  <!-- Message texte simple envoye a un groupe -->
17  <message id="12" type="text" is_edited="false" is_deleted="false">
18    <sender_id>1</sender_id>
19    <recipient_type>group</recipient_type>
20    <recipient_id>1</recipient_id>
21    <content>bonjour ca va</content>
22    <timestamp>2025-07-13T18:21:03</timestamp>
23    <is_read>false</is_read>
24  </message>
25 </messages>

```

Listing 7 – Extrait de data/messages.xml

### Exemple de groups.xml

```

1 <!-- Fichier contenant les groupes de discussion -->
2 <groups>
3   <!-- Exemple d'un groupe de projet prive -->
4   <group id="1" is_active="true">
5     <name>Equipe Projet DSS</name>

```

```

6      <description>Groupe de travail pour le projet de plateforme de
7      discussion</description>
8      <created_by>1</created_by>
9      <created_at>2024-01-10T09:00:00</created_at>
10     <members>
11       <!-- Le createur est admin du groupe -->
12       <member user_id="1" role="admin" joined_at="2024-01-10T09:00:00"/>
13     </members>
14     <settings>
15       <privacy>private</privacy>
16       <notifications>true</notifications>
17       <max_members>20</max_members>
18     </settings>
19 </group>
</groups>

```

Listing 8 – Extrait de data/groups.xml

## B Diagrammes UML

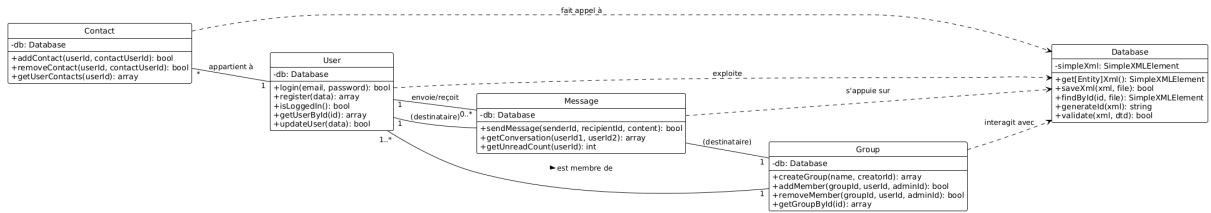


FIGURE 2 – Diagramme de classes de l'application.

## C Vues de l'application




FIGURE 3 – Page d'accueil de l'application.

The screenshot displays the login interface of a web application titled 'Plateforme de Discussion'. The page features a green header bar with the site's name and navigation links for 'Accueil' and 'Inscription'. A central white card contains the login form, which includes a blue icon of a right-pointing arrow inside a bracket, the heading 'Connexion', and the instruction 'Connectez-vous à votre compte'. The form has two input fields: 'Nom d'utilisateur' with a user icon and 'Mot de passe' with a lock icon and a toggle for visibility. A green 'Se connecter' button with the same arrow-in-bracket icon is positioned below the fields. A link for 'S'inscrire' is provided for users without an account. The footer contains the site name, a descriptive tagline, a copyright notice for 2025, and the project name 'Projet DSS XML'.

Plateforme de Discussion


Accueil Inscription





**Connexion**


Connectez-vous à votre compte

Nom d'utilisateur



Mot de passe

 Se connecter

Pas encore de compte ? [S'inscrire](#)

Plateforme de Discussion

Une plateforme de discussion moderne développée avec PHP et XML

© 2025 Plateforme de Discussion. Tous droits réservés.  
Projet DSS XML

FIGURE 4 – Page de connexion.

Plateforme de Discussion

Tableau de bord

Messages

Groupes

Contacts

Administrateur

Bonjour, Administrateur !

Bienvenue sur votre tableau de bord. Vous avez 0 message(s) non lu(s).

3

Conversations

1

Groupes

3

Contacts

0

Non lus

Conversations récentes

mariama

hello

15/07 01:50

Équipe Projet DSS

Document de présentation du projet

15/01 09:00

Voir toutes les conversations

Actions rapides

Nouveau message

+ Créer un groupe

+ Ajouter un contact

Modifier le profil

Mes groupes

Équipe Projet DSS

1 membres

Voir tous les groupes

Activité récente

Connexion réussie

Vous vous êtes connecté avec succès

15/07/2025 18:41

Dernière connexion

Votre dernière connexion

15/07/2025 18:41

Plateforme de Discussion

Une plateforme de discussion moderne développée avec PHP et XML.

© 2024 Plateforme de Discussion. Tous droits réservés.

Projet DSS XML - Institut de Formation

FIGURE 5 – Tableau de bord principal après connexion.

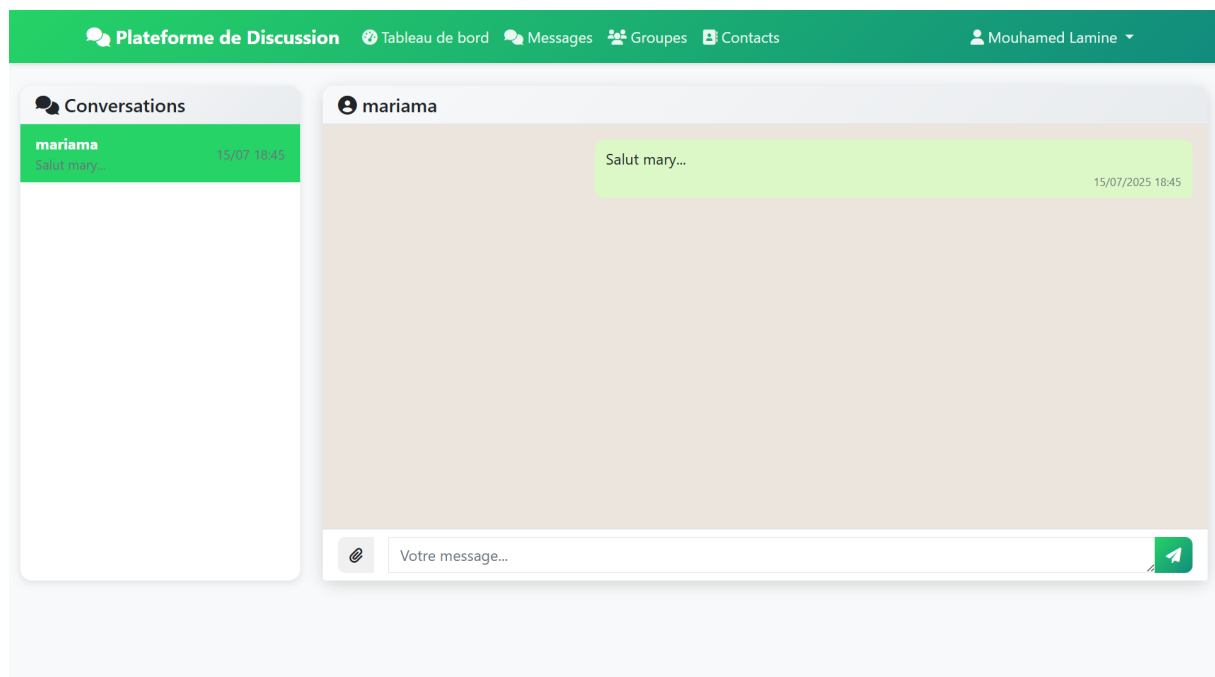


FIGURE 6 – Interface de la messagerie.

## Bibliographie

## Références

- [1] World Wide Web Consortium (W3C). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. <https://www.w3.org/TR/xml/>
- [2] The PHP Group. *SimpleXML Documentation*. <https://www.php.net/manual/en/book.simplexml.php>