PRO8445

Projet informatique FIPA 1A de Télécom SudParis

Développement d'une application client & Installation d'un serveur de messagerie Matrix.org pour la FIPA

Membres :

Sydney ROBAUX
Romain VERRIER
Hoe Ziet WONG
Xingyu ZHU

Sommaire

I) Présentation du projet..... 3
II) Présentation de MATRIX..... 4
A) MATRIX Foundation.....4
B) Histoire.....4
C) Fédérations.....5
III) Cahier des charges..... 6
A) Cas d'usages..... 6
IV) Analyse fonctionnelle..... 8
A) Service MATRIX.....8
B) Base de données.....9
C) Application mobile..... 9
V) Architecture du projet.....10
A) Etudes comparatives de distribution LINUX pour serveur MATRIX.....10
B) Etudes comparatives de framework pour l'application mobile..... 13
C) Spécifications de la base des données.....15
VI) Installation du serveur.....17
A) Demande d'une VM..... 17
B) Installation du serveur Matrix.....17
C) Création d'un environnement de tests..... 19
VII) Développement d'application mobile.....21
A) Widget.....21
B) Connexion API..... 22
C) Token/Authentication.....27
D) Page login..... 28
E) Page chat..... 29
VIII) Conclusion.....30
IX) Sources.....31

I) Présentation du projet

Le projet d'installer un serveur de messagerie "Matrix.org" à Télécom SudParis s'inscrit dans le cadre de notre projet informatique. En effet, notre groupe à pour objectif de mettre en place un service de communication chiffré "end-to-end" pour les étudiants, anciens et professeurs de la Filière Ingénieur Par Apprentissage de Télécom SudParis. Il devra respecter des contraintes et permettre des cas d'usages, notamment ceux décrits dans la suite de ce rapport.

Pour résumer, afin de réaliser ce projet, nous avons pour objectif de:

- Nous approprier et étudier le concept de Matrix
- Mettre en place un serveur Matrix
- Développer une application cliente

II) Présentation de MATRIX

A) MATRIX Foundation

La fondation Matrix.org est une société communautaire à but non lucratif qui gère les normes de la communauté Matrix. Matrix a été créé pour répondre à plusieurs besoins utilisateurs. Tout commence avec la récupération du contrôle des communications de l'utilisateur par l'utilisateur. Cela est suivi par la sécurisation des communications via le chiffrement des données sur des hôtes divers et variés. Et enfin le rendu doit être gratuit et disponible à toute personne voulant en disposer.

Le résultat de la réponse à ces besoins est Matrix, un standard ouvert pour les communications décentralisées en temps réel à travers la communication IP. Les principaux cas d'utilisation sont les messageries instantanées, les communications pour l'IoT, les appels à travers IP (VoIP) ainsi que les API HTTP. D'un autre côté l'accès aux fonctionnalités de Matrix est fédéré, ce qui permet la gestion des identités sur le serveur.

Pouvoir travailler sur le standard Matrix apporte de plus amples connaissances sur la réalité de la communication à travers IP, sur la synchronisation de données décentralisées ainsi que sur la mise en application d'une fédération.

B) Histoire

En 2014, Amdocs, une société basée aux Etats-Unis spécialisée dans l'informatique a créé le projet Matrix qui s'appelait « Amdocs Unified Communications » lors de création.

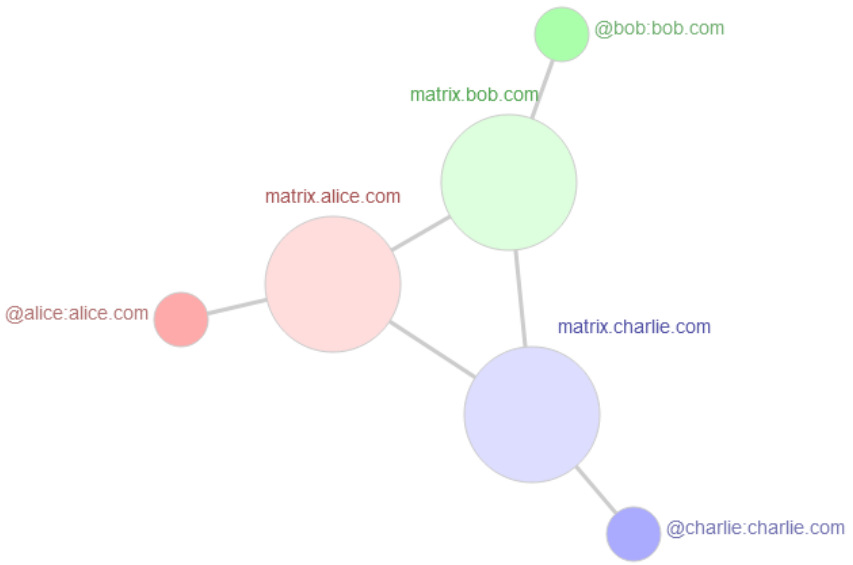
En 2015, l'entreprise a créé une filiale nommée « Vector Creations Limited » qui avait une vocation unique de développer ce projet, grâce au succès de ce protocole. Le développement du protocole s'est achevé en 2017 et il a été entièrement financé par Amdocs.

En 2017, Amdocs a mis fin au financement de ce projet et par conséquent, la filiale Vector Creations Limited devenait indépendante et continuait le projet.

En 2020, Element, le client de Matrix, a été créé et l'entreprise a aussi été renommée de la même manière. Element a remporté également le plus gros contrat au monde pour un service logiciel collaboratif dans la même année.

C) Fédérations

Une fédération fait référence à un ensemble de fournisseurs de services informatiques et réseaux en privilégiant des normes de fonctionnements de manière collective. La fédération permet la communication entre deux services communications qui sont déployés indépendamment de l'un à l'autre. Par exemple, nous pouvons envoyer un courriel depuis un serveur Google (gmail.com) vers un serveur Microsoft (hotmail.com) grâce à la fédération des deux serveurs. Matrix propose la fédération ouverte qui permettra à tous de bénéficier de son écosystème par le déploiement des "homeserver". Un utilisateur est donc connecté uniquement à un serveur, qui est connecté aux plusieurs autres serveurs fédérés. Matrix est aussi un réseau décentralisé comme le fonctionnement d'Internet, et il est constitué des homeserver déployés par des utilisateurs. La décentralisation garantit ainsi une meilleure sécurité et la liberté dans la gestion des données sur le serveur déployé.



III) Cahier des charges

A) Cas d'usages

1) Besoin : Communications entre les utilisateurs

   Requirements : Chiffré de bout en bout, possibilité d'envoyer les données de tous les formats, latence faible, open-source

   Scénario : La plupart des plateformes existantes sont basées sur l'utilisation d'un serveur centralisé, avec Matrix, nous appuyons sur la décentralisation des serveurs pour pouvoir mieux gérer les données (point de vue sécurité etc.). Chaque utilisateur va disposer de son propre serveur, qui sera capable d'échanger avec les autres serveurs de Matrix.

2) Besoin : Utilisation d'une application mobile de messagerie (Client) propriétaire à la FIPA.

   Requirements : Recevoir et envoyer des données (messages, images, vidéos, etc.) depuis et vers diverses plateformes de communications

   Scénario : Les utilisateurs téléchargent et installent une application de messagerie simple, propriétaire et développée par les étudiants du groupe de projet. L'application doit contenir un canal de classe avec tous les étudiants et des canaux de communication entre deux élèves.

3) Besoin : Centralisation locale

   Requirements : Créer un serveur central local.

   Scénario : Dans une entreprise, chaque utilisateur de Matrix
   peut communiquer directement et envoyer des informations diffusées en mettant en place un serveur collectif d'entreprise (non ouvert sur l'extérieur).

4) Besoin : Modération des contenus

   Requirements : Assigner une personne/un administrateur pour modérer, contrôler ou censurer certains contenus en cas de besoin

   Scénario : Le serveur et l'application cliente doivent être modérés un ou des administrateurs. Les contenus inadaptés ne seront pas autorisés sur cette plateforme. Tous les contenus qui incitent la violence, haine, discrimination ou la pornographie seront bloqués.

   Le service Matrix doit :

   - Ne pas être accessible à des utilisateurs externes à Télécom SudParis.
   - Ne pas autoriser la création d'un compte sans vérifier que la demande ne provient pas d'un bot.
   - Permettre la création exceptionnelle d'un compte externe.

IV) Analyse fonctionnelle

A) Service MATRIX

1) Prérequis matériel et numérique

   Le service Matrix que nous souhaitons implémenter est appelé “on-premise” dans la mesure où il nécessite d’être déployé sur un serveur, de préférence avec un OS Linux. Un Virtual Private Server (VPS) loué chez un fournisseur cloud ou un serveur associé à l’école peuvent être notre solution d’hébergement.

   Le serveur doit être dimensionné pour supporter la charge d’une utilisation quotidienne par quelques élèves jusqu’à la fin de notre cursus à Télécom SudParis.

   De plus, il est nécessaire d’acquérir un nom de domaine pour le nommage et la communication avec le serveur Matrix.

2) Accès à la plateforme

   La création des comptes doit être limitée à une whitelist d’étudiants et de professionnels au sein de Télécom SudParis. Lors de la création effective du compte, le couple login et mot de passe doit être transmis à l’utilisateur via un email sur son adresse du domaine “telecom-sudparis.eu”.

3) Modération

   Les comptes administrateurs doivent avoir la capacité de bannir un utilisateur d’une “room” et de supprimer les messages problématiques

.

B) Base de données

A l'installation de Synapse, une base de données SQLite est installée cependant de nombreuses applications utilisent des bases de données différentes pour des raisons de gestion ou de performances.

Nous souhaitons donc mettre en place également une base de données de notre choix.

Cette étape nous permettrait notamment d'avoir l'accès privilégié à des données qui sont présentes et qui circulent sur notre service. Grâce à cela nous aurons une meilleure accessibilité sur les données, ce qui permettra de mieux répondre aux besoins de nos clients. En parallèle, cela permet aussi de séparer l'administration du service et de la base de données en deux instances indépendantes.

C) Application mobile

Après l'implémentation du service MATRIX, nous souhaitons créer une application client sur mobile.

Cette application aura les fonctionnalités suivantes :

- Télécharger automatiquement les messages (textes) depuis le serveur.
- Envoyer les messages (textes) depuis l'application.
- Télécharger automatiquement les médias de différents formats (Image, Vidéo, Voix, etc).
- Envoyer les médias de différents formats (Image, Vidéo, Voix, etc).
- Recevoir les notifications sur les messages/médias entrants.

Cette application aura une interface graphique intuitive et facile à utiliser.

V) Architecture du projet

A) Etudes comparatives de distribution LINUX pour serveur MATRIX

Le serveur Matrix est manipulé sur le système Linux. Donc pour déterminer le distributeur de Linux qui peut fournir les meilleures performances, nous allons comparer plusieurs distributeurs Linux en des points importants.

1) Avantages et Inconvénients des distributeurs

Distribution	**Avantages**	**Inconvénients**	**Utilisation recommandée**
Ubuntu	Facile à utiliser, grande communauté de développeurs, adaptée aux débutants	Peut être considérée comme un peu lourde en termes de ressources système	Ordinateurs personnels, ordinateurs portables, serveurs
Debian	Stable, sécurisée, fiable, grande bibliothèque de logiciels	Peut nécessiter des connaissances avancées pour la configuration	Serveurs, utilisateurs avancés
Fedora	Mises à jour fréquentes, fonctionnalités innovantes	Peut être moins stable que d'autres distributions, fonctionnalités expérimentales	Ordinateurs personnels, ordinateurs portables
CentOS	Stabilité, sécurité, compatibilité avec les logiciels RHEL	Peut manquer de fonctionnalités plus récentes, mises à jour moins fréquentes	Serveurs, environnements d'entreprise
Arch Linux	Personnalisable, flexibilité, système de gestion de paquets avancé	Peut nécessiter des connaissances avancées pour la configuration, peut être moins stable que d'autres distributions	Utilisateurs avancés

2) Installation de serveur Matrix sur les différents distributeurs

Distributeur Linux	Ressources matérielles	Facilité de configuration	Facilité d'installation	Facilité de maintenance	Mises à jour	Performance
Ubuntu	Exigence matérielle modeste	Facile	Facile	Facile	Automatique	Pour un petit nombre d'utilisateurs
Debian	Exigence matérielle modeste	Difficile	Facile	Facile	Automatique	Pour un petit nombre d'utilisateurs
CentOS/ Fedora	Pour les serveurs Matrix nécessitant plus de ressources	Difficile	Facile	Facile	Automatique	Pour un nombre plus important d'utilisateurs (Entreprise)
Arch Linux	Exigence matérielle modeste	Le plus complexe	Plus complexe	Plus complexe	Manuelle	Pour personnaliser et configurer le serveur

3) Notre choix de distributeur

Il y a plusieurs distributions Linux qui peuvent être utilisées pour configurer un serveur Matrix. Le choix du distributeur Linux pour le serveur Matrix dépend du nombre d'utilisateurs, des ressources matérielles et des compétences techniques disponibles. Sur la base des trois raisons ci-dessus, nous avons décidé de choisir Ubuntu.

La raison pour laquelle Ubuntu est la plus recommandée pour le serveur Matrix est qu'elle est facile à utiliser et dispose d'une grande communauté de développeurs, ce qui signifie qu'il y a une grande quantité de documentation et de support en ligne disponibles pour aider à la configuration du serveur. De plus, Ubuntu est livrée avec des outils de configuration et de gestion des paquets avancés, ce qui facilite l'installation des dépendances et des logiciels nécessaires pour faire fonctionner Matrix. D'ailleurs, Ubuntu a des exigences matérielles modestes et peut être installé sur des systèmes avec des spécifications limitées. Enfin, pour un petit nombre d'utilisateurs (notre cas), les distributions Debian et Ubuntu peuvent être une bonne option, car elles offrent une installation facile à l'aide de paquets pré-configurés et ne nécessitent pas de ressources matérielles importantes. Mais Debian nécessite des connaissances techniques plus avancées pour la configuration. Nous recommandons donc d'utiliser Ubuntu car nous sommes nouveaux dans la configuration de serveurs.

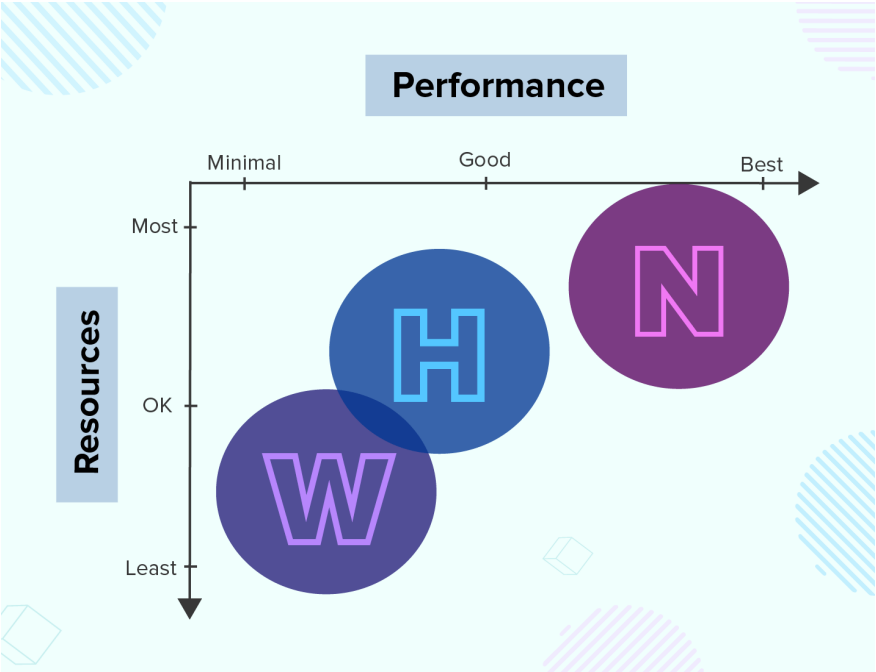
B) Etudes comparatives de framework pour l'application mobile

Quelle technologie pour l'application mobile ? (Native, TechnoWeb, Hybride)

Il existe 3 différents types de framework pour développer une application mobile notamment Native, Techno Web et Hybride

Voici un tableau comparatif des ces frameworks selon différents critères

	Native	**TechnoWeb**	**Hybride**
Performance (Réactivité)	>TechnoWeb >Hybride	<Native <Hybride	>TechnoWeb <Native
Multi-plateforme	Non	Oui	Oui
Facilité à coder	Basse	Haute	Moyenne
Dépendance	App Store/Google Play	Navigateur web & connexion réseau	App Store/Google Play
Mémoire de stockage nécessaire	Haute	0	Basse
Exemple	Java, Kotlin, Swift	HTML5, CSS5	Flutter, ReactActive



Legendes : N - Native, H - Hybride, W - TechnoWeb

Quel Software Development Kit (SDK) ? (Flutter, AndroidStudio, NativeScript, ReactNative)

	Flutter	**ReactNative**
Langage utilisé	Dart	JavaScript
Facilité d'utilisation	Moins complexe	Moins complexe
Documentation disponible	Très bien documenté	Très bien documenté
Multiplateforme	Oui	Oui
Exemple d'utilisation	Google Play, eBay, BMW	Facebook, Instagram, Airbnb, Uber

Après avoir réalisé ces études de comparaison entre les différentes technologies, nous avons décidé d'utiliser technologie de type hybride car il s'agit d'une solution qui a le compromis entre la performance et les ressources nécessaires. Cette technologie est aussi le framework le plus utilisé sur le marché aujourd'hui.

Nous avons également choisi Flutter au détriment de ReactNative pour quelques raisons :

- Framework Flutter est moins complexe et facile à maîtriser et à installer
- Langage Dart utilisé par Flutter nous permettrait de découvrir et apprendre ce langage
- Nous avons également un membre de l'équipe qui est expérimenté avec ce framework

C) Spécifications de la base des données

Le service Matrix est installé avec une base de données SQLite cependant il est fortement recommandé de créer une base de données autre et de la lier avec le service Matrix. Afin de déterminer la base de données à utiliser nous allons comparer plusieurs Systèmes de Gestion de Bases de Données (SGBD) sur des points importants.

1) La sécurité

	Native network encryption	**Brute-force protection**	**Patch access**
MySQL	Yes(SSL)	No	Partial
MariaDB	Yes (SSL)	No	Yes(documented)
Oracle	Yes	Yes	Unknown
SQLite	No	No	Partial
PostgreSQL	Yes	Yes	Yes(documented)

Au niveau de la sécurité des SGBD génériques on remarque que le SQLite installé par défaut est très faible et ne comporte même pas de chiffrement des données sur le réseau. À côté on remarque des bases de données comme PostgreSQL, Oracle et MariaDB possédant plus de sécurité.

2) Les possibilitées d'actions

	Intersect	**Outer joins**	**Merge joins**	**Parallel query**
MySQL	No	Yes	No	No
MariaDB	Yes	Yes	No	No
Oracle	Yes	Yes	Yes	Yes
SQLite	Yes	LEFT only	No	No
PostgreSQL	Yes	Yes	Yes	Yes

Au niveau des commandes SQL possibles on remarque que toutes ne sont pas réalisables sur les différents SGBD. Cependant le plus important est le Parallel query permettant à notre base de données d'être plus performante. On peut remarquer que le SGBD SQLite ne le prend pas en charge alors que PostgreSQL et Oracle le peuvent en plus des autres commandes.

VI) Installation du serveur

A) Demande d'une VM

Le serveur Matrix devant être disponible pour tous les étudiants de la FIPA nous pousse à faire tourner le serveur sur le réseau de l'école. En même temps que cette demande nous demandons un dns pour la machine ce qui permet de ne pas avoir à connaître l'IP.

Pour réaliser cette demande nous avons été redirigés vers M.PROCACCIA qui a créé un ticket sur le service de ticketing de l'école. Après une relance de notre part ainsi qu'une relance de M.RANC nous avons pu récupérer une VM. La VM est accessible en ssh, en voici les caractéristiques:

- Nom de domaine: rs2m-matrix-vps.telecom-sudparis.eu
- Distribution : Ubuntu 22.04
- Firewall machine : uniquement ssh en IPv4 et IPv6
- Comptes :
    - root : pour le service informatique
    - matrix : pour le groupe projet et avec les droits sudoers
- IP machine : 157.159.52.55

B) Installation du serveur Matrix

Afin de mieux intégrer le fonctionnement de Matrix nous utilisons l'installation par synapse et non celle en conteneur Docker toute prête. Après de nombreux essais en local en suivant la procédure définie sur la documentation Matrix et sur plusieurs sites nous avons pu comprendre et avoir une procédure d'installation qui évite les erreurs obtenues en suivant uniquement la documentation. Cette procédure est la suivante :

- sudo apt update
    - Mise à jour des paquets actuels
- sudo apt install matrix-synapse-py3
    - Installation du serveur matrix, le serveur se lancera automatiquement
- sudo nano /etc/matrix-synapse/homeserver.yaml
    - Fichier de configuration global du serveur

Lors des installations en local nous avons eu des difficultés à bien sécuriser le serveur, dans le meilleur des cas nous avons un certificat auto-signé et dans le pire des cas le serveur ne tournait plus. Donc pour avoir un serveur au moins fonctionnel nous essayons de rester simple dans la configuration. Les lignes commentées étaient nécessaires à la compréhension ou on du être modifiées.

```
pid_file: "/var/run/matrix-synapse.pid"
listeners:
 - port: 8080 #port 8080 != 8008(préconfiguration)
   tls: false
   type: http #pas en https pour rester simple
   x_forwarded: true
   bind_addresses: ['0.0.0.0'] #Toutes les adresses de la machine
   resources:
     - names: [client, federation]
       compress: false
database:
 name: sqlite3 #Base de donnée préconfigurée
 args:
   database: /var/lib/matrix-synapse/homeserver.db
log_config: "/etc/matrix-synapse/log.yaml"
media_store_path: /var/lib/matrix-synapse/media
signing_key_path: "/etc/matrix-synapse/homeserver.signing.key"
suppress_key_server_warning: true #Permet de ne pas avertir du manque de
certificat
trusted_key_servers:
 - server_name: "matrix.org"
macaroon_secret_key: 'azertyazerty' #Permet de créer les jetons
d'authentification (ligne complètement ajoutée)
server:
   server_name: "rs2m-matrix-vps.telecom-sudparis.eu" #Permet de définir le nom du
serveur
registration_enabled: true #Permet la création de comptes(ligne ajoutée)
registration_shared_secret:"nb3utUkXEtS+dE4lPiEL8cWmE/M4zd8nRVfTpQOiT4jQj2zY0kI0aU
6cLRYeuVVV" #Vérifie l'authenticité des requêtes d'inscription (ligne
ajoutée)
```

Comme on peut le voir, des lignes ont été paramétrées pour correspondre à nos attentes. Comme au début on ne savait pas si l'IP de notre serveur était tournante on a lié l'adresse à toutes les adresses du serveur dont on sait que au moins une d'entre elles est liée au dns. Ensuite afin d'avoir un serveur rapidement disponible pour les tests nous n'avons pas changé la base de données. Et enfin nous avons les lignes de fin qui permettent de créer et de s'authentifier avec des utilisateurs.

Le plus intriguant est le choix du changement de port. Au début le port 8008 était fonctionnel et notre serveur tournait dessus. Cependant un week-end le serveur s'est redémarré et n'était plus fonctionnel. En changeant le port il fonctionnait donc nous avons changé le port.

Une hypothèse à ce souci est que le serveur a redémarré sans libérer le port. Si cela finit par se reproduire il sera important de se pencher dessus avant que tous les ports de la machine ne soient bloqués.

Après avoir configuré le serveur, il est temps de le rendre disponible. Pour cela il suffit juste d'autoriser le port sur le firewall via la commande suivante.

- sudo ufw allow 8080

Après tout ça le serveur Matrix est accessible sur le réseau de l'école sous le nom rs2m-matrix-vps.telecom-sudparis.eu:8080. Et il est possible d'en vérifier la disponibilité sur le lien suivant http://rs2m-matrix-vps.telecom-sudparis.eu:8080/ toujours en étant sur le réseau de l'école.

C) Création d'un environnement de tests

Afin de tester un peu le serveur il a été choisi de créer des utilisateurs ce qui est le principal pour commencer. Une fonction bash a été créée permettant de créer un utilisateur en prenant en paramètre un nom d'utilisateur et un mot de passe.

```
#!/bin/bash
# Fonction pour créer un utilisateur
create_matrix_user() {
    username=$1
    password=$2
    register_command="register_new_matrix_user -c homeserver.yaml
http://rs2m-matrix-vps.telecom-sudparis.eu:8080/ $username $password
--no-admin"
    $register_command
}
```

Cette fonction pourra être utilisée en étant mise dans une boucle prenant en paramètre une liste d'adresses mail qui récupérera le nom et initialiser le mot de passe avec un mot de passe "générique".

VII) Développement d'application mobile

A) Widget

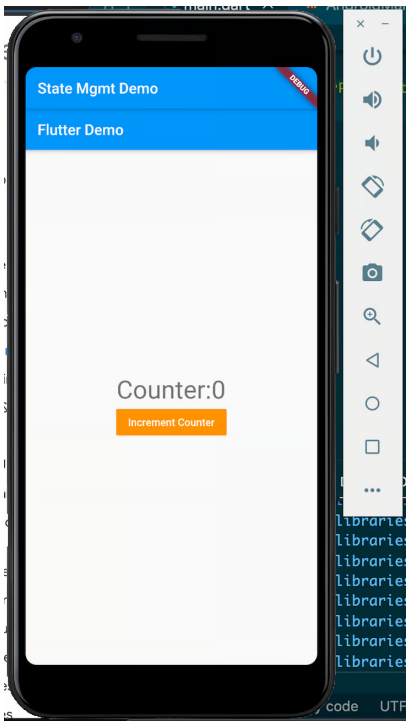
Dans l'application mobile que nous concevons, l'interface utilisateur (UI) est composée principalement des Widgets de flutter. Les widgets de flutter sont développés en utilisant les différents frameworks, tout en s'inspirant de React. Les widgets définissent essentiellement l'interface présentée aux utilisateurs. Il existe deux types de widgets dans flutter :

1) Widget type stateless

   Un widget de type stateless est statique et ne change pas en cas d'interaction avec l'utilisateur. Quelques exemples sont Icon, IconButton, et Text. Les widgets stateful sont des sous-classes de StatelessWidget.

2) Widget type stateful

   Un widget stateful est dynamique : par exemple, il peut changer son apparence en réponse aux événements déclenchés par les interactions de l'utilisateur ou lorsqu'il reçoit des données. Les exemples de widgets stateful incluent Checkbox, Radio, Slider, InkWell, Form et TextField. Les widgets stateful sont des sous-classes de StatefulWidget.



Dans cet exemple, le Text "Counter:0" est donc de type stateless, et le TextButton est de type stateful car en appuyant sur ce bouton, l'utilisateur déclenche un événement.

Dans notre application, nous avons utilisé différents widgets de chaque type. Les widgets les plus utilisés sont notamment :

- Text : afficher les textes sur l'application
- Container : segmenter les différentes parties sur l'interface et gérer l'espacement des parties
- TextButton : un bouton avec du texte qui permet déclencher une action
- TextField : un champs qui permet à l'utilisateur de saisir un texte
- Icon : afficher une icône
- IconButton : afficher une icône dynamique/cliquable

B) Connexion API

L'application interagit avec le serveur Matrix en utilisant un API développé par Matrix.org. Client Server API sert d'une passerelle qui permet aux utilisateurs de s'authentifier auprès de serveur, envoyer et récupérer les messages. L'API communique avec le serveur en envoyant les requêtes HTTP comme GET et POST via l'outil de ligne de commande CURL. On utilise la commande CURL parce que CURL est une puissante commande pour le transfert de données avec des serveurs Web. Il prend en charge plusieurs protocoles, notamment HTTP que l'on utilise dans le projet, et peut effectuer diverses opérations réseau, telles que l'envoi de demandes, la réception de réponses, etc.

En utilisant cet API nous avons la possibilité de (en dessous, c'est juste un exemple, on ne va pas utiliser ce compte "toto" dans la partie de l'application) :

1) Créer des utilisateurs et se connecter

   Créer un nouvel utilisateur "toto" en fournissant le nom de domaine de serveur Matrix et la port utilisée. Dans la création on définit le nom du compte et son mot de passe :

```
matrix@rs2m-matrix-vps:/etc/matrix-synapse$ register_new_matrix_user -c homeserver.yaml http://rs2m-matrix-vps.te
lecom-sudparis.eu:8080/
New user localpart [matrix]: toto
Password:
Confirm password:
Make admin [no]: y
Sending registration request...
Success!
```

Le nouveau compte “toto” est créé comme un administrateur.

Pour se connecter sur le compte “toto”, on envoie une requête POST au domaine /client/r0/login du serveur. On a besoin juste de fournir le nom de l’utilisateur et son mot de passe et de bien indiquer le nom du domaine de notre serveur Matrix :

```
matrix@rs2m-matrix-vps:/etc/matrix-synapse$ curl -XPOST -d '{"type":"m.login.password", "user":"toto", "password"
:"12345"}' "http://rs2m-matrix-vps.telecom-sudparis.eu:8080/_matrix/client/r0/login"
{"user_id":"@toto:rs2m-matrix-vps.telecom-sudparis.eu","access_token":"syt_dG90bw_yHkHaVNWBWYkiyBlBjyV_37NaUu","h
ome_server":"rs2m-matrix-vps.telecom-sudparis.eu","device_id":"YWGCWNQKWQ"}matrix@rs2m-matrix-vps:/etc/matrix-syn
```

Si la connexion est établie, on peut récupérer son access_token et son user_id.

Token récupéré : syt_dG90bw_yHkHaVNWBWYkiyBlBjyV_37NaUu
user_id : @toto:rs2m-matrix-vps.telecom-sudparis.eu

2) Créer une room

Afin que les utilisateurs puissent envoyer et recevoir des message entre eux, on doit créer la room et mettre les utilisateurs dans la même room.

Pour créer la room, on envoie la requête POST au domaine /client/r0/createRoom du serveur en fournissant le token de créateur. Et dans la commande, on doit définir l’autorisation et le nom de la room :

```
matrix@rs2m-matrix-vps:/etc/matrix-synapse$ curl -XPOST -H 'Authorization: Bearer syt_dG90bw_yHkHaVNWBWYkiyBlBjyV
_37NaUu' -H 'Content-Type: application/json' -d '{"preset": "public_chat", "name": "FIPA-toto"}' 'http://rs2m-mat
rix-vps.telecom-sudparis.eu:8080/_matrix/client/r0/createRoom'
{"room_id":"!ShfNWPSqDVUhkhGjtX:rs2m-matrix-vps.telecom-sudparis.eu"}matrix@rs2m-matrix-vps:/etc/matrix-synapse$
```

Dans l’exemple ci-dessus, la room est en “public_chat”. Cela signifie que n’importe qui peut rejoindre dans la room et tous les participants même s’ils ne sont pas le créateur ou le propriétaire peuvent envoyer les invitations aux autres utilisateurs.

Si la création de la room est réussie, on peut récupérer le **room_id : !ShfNWPSqDVUhkhGjtX:rs2m-matrix-vps.telecom-sudparis.eu**

En utilisant la commande suivante, on peut lister les rooms créés et savoir leurs informations : curl --header "Authorization: Bearer

syt_dG90bw_yHkHaVNWBWYkiyBlBjyV_37NaUu" -X GET http://rs2m-matrix-vps.telecom-sudparis.eu:8080/_synapse/admin/v1/rooms

Voici, les informations de notre nouveau room **(room_id : !ShfNWPSqDVUhkhGjtX:rs2m-matrix-vps.telecom-sudparis.eu) :**

{"room_id":"!ShfNWPSqDVUhkhGjtX:rs2m-matrix-vps.telecom-sudparis.eu","name":"FIPA-toto","canonical_alias":null,"joined_members":1,"joined_local_members":1,"version":"10","creator":"@toto:rs2m-matrix-vps.telecom-sudparis.eu","encryption":null,"federatable":true,"public":false,"join_rules":"public","guest_access":null,"history_visibility":"shared","state_events":6,"room_type":null}]

3) Inviter un utilisateur dans une room

Avant l'invitation, on a déjà créé un autre utilisateur "azerty", "toto" peut inviter (ajouter) "azerty" dans la room qu'on a créé dans la précédente étape dans 2 étapes :

1. Comme un invitant "toto" doit lancer une invitation à "azerty" (invité) en fournissant access_token d'invitant (token de "toto") et user_id d'invité (user_id de "azerty") au domaine 'invite' de notre serveur Matrix.

```
matrix@rs2m-matrix-vps:/etc/matrix-synapse$ curl -X POST -H "Authorization: Bearer syt_dG90bw_IMQGtmIwrrECTImptwcf_3bSRpL" -H "Con
tent-Type: application/json" -d '{"user_id":"@azerty:rs2m-matrix-vps.telecom-sudparis.eu"}' "http://rs2m-matrix-vps.telecom-sudpar
is.eu:8080/ matrix/client/r0/rooms/%21ShfNWPSqDVUhkhGjtX:rs2m-matrix-vps.telecom-sudparis.eu/invite"
```

2. Se connecter sur le compte "azerty" (invité) et lancer l'approbation de l'invitation (une requête POST) vers le domaine "join" du serveur en fournissant son access_token. Dès que l'invitation est fini, on lance une requête GET pour chercher dans le domaine 'join_room' du serveur pour vérifier si 'azerty' joint dans la room.

```
matrix@rs2m-matrix-vps:/etc/matrix-synapse$ curl -X POST -H "Authorization: Bearer syt_YXplcnR5_iHMzEdOLrpKCZAJHGNdD_2QX1xi" -H "C
ontent-Type: application/json" -d '{}' "http://rs2m-matrix-vps.telecom-sudparis.eu:8080/_matrix/client/r0/rooms/%21ShfNWPSqDVUhkhG
jtX:rs2m-matrix-vps.telecom-sudparis.eu/join"
{"room_id":"!ShfNWPSqDVUhkhGjtX:rs2m-matrix-vps.telecom-sudparis.eu"}matrix@rs2m-matrix-vps:/etc/matrix-synapse$ curl -X GET -H "A
uthorization: Bearer syt_dG90bw_IMQGtmIwrrECTImptwcf_3bSRpL" "http://rs2m-matrix-vps.telecom-sudparis.eu:8080/_synapse/admin/v1/us
ers/@azerty:rs2m-matrix-vps.telecom-sudparis.eu/joined_rooms"qDVUhkhGjtX:rs2m-matrix-vps.telecom-sudparis.eu/join"
{"joined_rooms":["!vdSInBRduAlqgIggQP:rs2m-matrix-vps.telecom-sudparis.eu","!xgmUwQQAicNxHlxKdr:rs2m-matrix-vps.telecom-sudparis.e
u","!FzsTjhcIoAJFTZanAl:rs2m-matrix-vps.telecom-sudparis.eu","!nrjnVTDEmkmQFPnNwq:rs2m-matrix-vps.telecom-sudparis.eu","!tkFOWvIXw
NJEBoFILx:rs2m-matrix-vps.telecom-sudparis.eu","!aOrbSFlvLROKqKepgZ:rs2m-matrix-vps.telecom-sudparis.eu","!YsOhpuHkYlibvpHpzr:rs2m
-matrix-vps.telecom-sudparis.eu","!ShfNWPSqDVUhkhGjtX:rs2m-matrix-vps.telecom-sudparis.eu","!kQMtmHEssEJbQTQllb:rs2m-matrix-vps.te
lecom-sudparis.eu","!DOfwQefJpJVEBzfEdG:rs2m-matrix-vps.telecom-sudparis.eu","!fzPjYntZakpHkGIYnz:rs2m-matrix-vps.telecom-sudparis
.eu","!EQxrFpjVMLGrLoFgPV:rs2m-matrix-vps.telecom-sudparis.eu"],"total":12}matrix@rs2m-matrix-vps:/etc/matrix-synapse$
```

Voilà, "azerty" est dans le room de "toto".

4) Rejoindre une room

Avant la jointure, on a déjà créé une room pour “azerty”, et son room_id est %21fzPjYntZakpHkGIYnz:rs2m-matrix-vps.telecom-sudparis.eu.

Maintenant pour que “Toto” puisse rejoindre dans cette room par lui même, “toto” doit lancer une requête POST au domaine “join” du serveur et en fournissant son access_token. Et dans le domaine “join”, il faut bien indiquer le room_id dans lequel on va rejoindre.

```
matrix@rs2m-matrix-vps:/etc/matrix-synapse$ curl -XPOST -d '{}' "http://rs2m-matrix-vps.telecom-sudparis.eu:8080/
_matrix/client/r0/join/%21fzPjYntZakpHkGIYnz:rs2m-matrix-vps.telecom-sudparis.eu?access_token=syt_dG90bw_yHkHaVNW
BWYkiyBlBjyV_37NaUu"
{"room_id":"!fzPjYntZakpHkGIYnz:rs2m-matrix-vps.telecom-sudparis.eu"}matrix@rs2m-matrix-vps:/etc/matrix-synapse$
```

Pour lister les rooms où “toto” joint, on va lancer une requête GET vers le domaine “joined_rooms” de “toto” et en fournissant son access_token et son user_id :

```
matrix@rs2m-matrix-vps:/etc/matrix-synapse$ curl -X GET -H "Authorization: Bearer syt_dG90bw_yHkHaVNWBWYkiyBlBjyV
_37NaUu" "http://rs2m-matrix-vps.telecom-sudparis.eu:8080/_synapse/admin/v1/users/@toto:rs2m-matrix-vps.telecom-s
udparis.eu/joined_rooms"
{"joined_rooms":["!fzPjYntZakpHkGIYnz:rs2m-matrix-vps.telecom-sudparis.eu","!xgmUwQQAicNxHlxKdr:rs2m-matrix-vps.t
elecom-sudparis.eu","!ShfNWPSqDVUhkhGjtX:rs2m-matrix-vps.telecom-sudparis.eu"],"total":3}matrix@rs2m-matrix-vps:/
```

On a réussi à joindre le room de “azerty”.

Maintenant l’utilisateur “azerty” et utilisateur “toto” sont dans la même room, ils peuvent commencer à communiquer via API.

5) Envoyer un message dans une room

L’envoi d’un message dans une room est possible avec API Client.

Dans cet exemple, nous envoyons un message depuis le compte azerty :

```
curl -XPOST -d '{"msgtype":"m.text", "body":"hello, je suis azerty"}'
"http://rs2m-matrix-vps.telecom-sudparis.eu:8080/_matrix/client/r0/rooms/%21xg
mUwQQAicNxHlxKdr:rs2m-matrix-vps.telecom-sudparis.eu/send/m.room.messa
ge?access_token=syt_YXplcnR5_nUnnqqDAFBzUlEcAEbLu_4afqa5"
```

Pour envoyer un message, l'utilisateur doit fournir son token d'accès et écrire le message dans le champ "body" d'objet JSON envoyé. L'identifiant de room concerné doit également être renseigné dans l'url

Un envoi réussi est donc suivi par la réception l'identifiant de cet événement :

{"event_id":"$_sCuLs2GNVGLwa5I4Gw5FlZmBmVnHtY9apvzTfzlmpI"}

6) Récupérer les messages envoyés dans une room

Pour récupérer un message dans une room, nous utilisons une commande tel qu' affiché dans l'exemple ci-dessous :

```
curl -X GET -H "Authorization: Bearer
syt_YXplcnR5_nUnnqqDAFBzUlEcAEbLu_4afqa5"
"http://rs2m-matrix-vps.telecom-sudparis.eu:8080/_matrix/client/r0/rooms/%21xg
mUwQQAicNxHlxKdr:rs2m-matrix-vps.telecom-sudparis.eu/messages?dir=b&li
mit=1&type=m.text
```

Comme pour l'envoi de message, la requête doit commencer par le token d'accès d'utilisateur et l'identifiant de room doit aussi être indiqué dans l'url. La valeur "limit=1" à la fin de l'url nous permet de récupérer le dernier et un seul message.

Le serveur retourne donc un objet json qui contient le message :

{"type":"m.room.message","room_id":"!xgmUwQQAicNxHlxKdr:rs2m-matrix-v
ps.telecom-sudparis.eu","sender":"@test:rs2m-matrix-vps.telecom-sudparis.eu","c
ontent":{"msgtype":"m.text","body":"hello, je suis
azerty"},"origin_server_ts":1687651426532,"unsigned":{"age":98774},"event_id"
:"$MdX9iUQ7XpP4hvDy46PY_swRULsh_piLNrxyGFDPr5E","user_id":"@test:
rs2m-matrix-vps.telecom-sudparis.eu","age":98774}],"start":"t10-65_0_0_0_0_0_
0_0_0_0","end":"t10-64_0_0_0_0_0_0_0_0_0"}

C) Token/Authentication

Après le lancement de l'application, le processus d'authentification de l'utilisateur aura lieu en premier avant que l'utilisateur puisse accéder à la messagerie.

Dans un premier temps, l'application vérifie si un token d'utilisateur a déjà existé dans sa mémoire. Si oui, l'étape d'authentification ne sera plus nécessaire pour cet utilisateur. Dans le cas contraire, l'application va afficher la page de login en lui demandant son login et mot de passe de compte Matrix. Les informations saisies seront donc envoyées au serveur via une commande cURL de format :

```
curl -XPOST -d '{"username":"example", "password":"wordpass", "auth": {"type":"m.login.dummy"}}'
"https://localhost:8448/_matrix/client/r0/register"
```

Cette commande permet notamment la récupération de token d'utilisateur qui nous servira dans les autres requêtes HTTP.

Le token récupéré sera par la suite stocké en mémoire locale. Pour stocker ce token, nous faisons l'appel à un package de flutter qui s'appelle "flutter_secure_storage". Ce package permet le stockage des informations d'utilisateur de façon sécurisée. En Android, flutter_secure_storage utilise encryptedSharedPreference pour stocker les données de manière sécurisée. encryptedSharedPreference est une méthode de stockage sécurisé des préférences partagées, où les clés et les valeurs sont chiffrées. Ce mécanisme utilise le chiffrement AES pour générer une clé secrète qui est elle-même chiffrée avec RSA, puis stockée dans le KeyStore.

Exemple de méthode pour sauvegarder une information :

```
Future setUserName(String username) async {
  await storage.write(key: _keyUserName, value: username);
}
```

Exemple de méthode pour récupérer une information :

```
Future<String> _getUsername() async {
  return await storage.read(key: _keyUserName) ?? "";
}
```

D) Page login

Comme expliqué précédemment, la première partie de l'application est la page de login.

Le mécanisme de l'authentification se déroule dans l'ordre ci-dessous :

1) Au lancement, cette interface sera affichée. En cliquant sur le bouton "Continuer", l'utilisateur sera redirigé vers la page de chat si le token d'utilisateur est déjà présent dans la mémoire.

2) S'il n'existe pas de token dans la mémoire de l'application, les champs pour saisir le login et le mot de passe seront affichés. Les informations seront utilisées ensuite pour récupérer le token depuis le serveur et l'enregistrer en local. A la prochaine connexion, cet utilisateur n'aura plus besoin donc de s'authentifier auprès de serveur.

E) Page chat

Cette page est la partie principale de l'application. Sur cette page, un utilisateur peut afficher une conversation dans un room et aussi d'envoyer les messages.

A la chargement de la page, l'application va récupérer les messages dans le room depuis le serveur en faisant appel à API Client. Ce processus sera répété dans une boucle par l'application toutes les 10 secondes afin de pouvoir récupérer et afficher les messages récents.

En bas de cette page, l'utilisateur a également la possibilité d'envoyer les messages en saisissant dans le champ du texte et en cliquant sur le bouton. Les messages sont aussi envoyés grâce à API Client.

VIII) Conclusion

Grâce à ce projet informatique, nous avons pu déployer un serveur Matrix. Cette expérience nous a permis notamment de comprendre le fonctionnement d'un serveur Matrix, le principe de fédération et décentralisation et ainsi que la configuration de ce serveur. La partie du développement application mobile a été aussi une expérience très enrichissante car nous avons dû concevoir une application de zéro, qui nécessite non seulement une connaissance approfondie du fonctionnement de Matrix, mais aussi la maîtrise de programmation; le framework Flutter et le langage Dart dans notre cas.

A la fin de ce projet, nous avons pu aussi identifier les améliorations possibles comme l'ouverture de flux depuis l'extérieur afin que le serveur soit accessible depuis l'extérieur et ainsi que l'amélioration d'application.

IX) Sources

https://docs.flutter.dev/ui/interactive#:~:text=A%20widget%20is%20either%20stateful,are%20examples%20of%20stateless%20widgets.

https://docs.flutter.dev/ui/widgets-intro#:~:text=Flutter%20widgets%20are%20built%20using,their%20current%20configuration%20and%20state.

https://pub.dev/packages/flutter_secure_storage

https://matrix.org/docs/legacy/client-server-api/

https://matrix-org.github.io/synapse/v1.41/admin_api/rooms.html