Nicolas CHUSSEAU : s5a04a
Pacôme CAILLETEAU : s5a03a
Lilian DELHOMMEAU : s5a02a

Connection en ssh à la base de données : `ssh identifiant@172.26.82.44`

Pour chaque personne :
Création des 3 répertoires : sp_mongod, ss1_mongod et ss2_mongod :

```
mkdir -p sp_mongod/data/db
touch sp_mongod/mongod.log
touch sp_mongod/mongod.conf
mkdir -p ss1_mongod/data/db
touch ss1_mongod/mongod.log
touch ss1_mongod/mongod.conf
mkdir -p ss2_mongod/data/db
touch ss2_mongod/mongod.log
touch ss2_mongod/mongod.conf
```

mongod.config
mongod.log
rep data/db
numéro de port différents dans tous les .conf

le .conf

```
# mongod.conf

# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/

# Where and how to store data.
storage:
  dbPath: /home/s5a04a/sp_mongod/data/db
  journal:
    enabled: true
#  engine:
#  mmapv1:
#  wiredTiger:

# where to write logging data.
systemLog:
  destination: file
```

```
  logAppend: true
  path: /home/s5a04a/sp_mongod/mongod.log

# network interfaces
net:
  port: 27036
  bindIp:  0.0.0.0


# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo

#security:

#operationProfiling:

replication:
   replSetName: rs04a

sharding:
  clusterRole: shardsvr

## Enterprise-Only Options:

#auditLog:

#snmp:
```

Lancer les 3 serveurs (pour chaque personne du groupe) :

```
mongod --config ./sp_mongod/mongod.conf
```

Lancer un client mongo (pour chaque personne du groupe) :

```
mongo --host localhost --port 27036
```

Ajouter les serveurs secondaires (pour chaque personne du groupe) :

```
rs.initiate()
rs.add("172.26.82.44:27037")
rs.add("172.26.82.44:27038")
```

Vérification de l'ajout des serveurs

```
rs.status()
```

Créer un utilisateur :

```
db.createUser({"user":"root","pwd":"s5a03a","roles":[{"role":"userAdminAnyDatabase","db":"admin"},{"role":"readWriteAnyDatabase","db":"admin"}]})
```

Update du nom du serveur de mongo-s1 vers 172.26.82.44 ATTENTION À CHANGER LE N° DE PORT:

```
cfg = rs.conf();
// Find the member you want to change and update its host field
for (var i = 0; i < cfg.members.length; i++) { if (cfg.members[i].host === "mongo-s1:27078") {cfg.members[i].host = "172.26.82.44:27078"; }}
rs.reconfig(cfg);
```

Une fois par groupe
Créer un serveur de configuration (1 par groupe) :
Le serveur de configuration utilisé est celui de s5a04a.

```
mkdir -p confsrv/data/db
touch confsrv/mongod.log
touch confsrv/mongod.conf
```

le .conf :

```
# mongod.conf

# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/

# Where and how to store data.
storage:
  dbPath: /home/s5a04a/confsrv/data/db
  journal:
    enabled: true
#  engine:
#  mmapv1:
#  wiredTiger:

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /home/s5a04a/confsrv/mongod.log
```

```
# network interfaces
net:
  port: 27110
  bindIp:  0.0.0.0


# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo

#security:

#operationProfiling:

replication:
   replSetName: conf04a

sharding:
   clusterRole: configsvr

## Enterprise-Only Options:

#auditLog:

#snmp:
```

Ajouter des lignes :

```
sharding:
  clusterRole: shardsvr
```

dans tous les mongo (sp ss1 et ss2).

Si on n'arrive plus à lancer les serveurs : supprimer tout ce qu'il y a dans db/

Lancer le serveur de conf :

```
mongod --config ./mongod.conf
```

Se connecter dessus en client :

```
mongo --port 27110
```

Le configurer en PRIMARY :

```
rs.initiate()
```

Lancer cette commande avec un port libre après le --port :

```
mongos --configdb conf04a/localhost:27110 --bind_ip 0.0.0.0 --port 27666
```

Se connecter en client mongo :

```
mongo --host localhost --port 27666
```

Depuis le client :

Ajouter les serveurs primaires :

```
sh.addShard("rs04a/172.26.82.44:27036")
sh.addShard("rs03a/172.26.82.44:27030")
sh.addShard("rs02a/172.26.82.44:27024")
```

Vérifier que c'est bien ajouté :

```
sh.status()
```

Créer la base :

```
sh.enableSharding("dbmedicalink")
sh.shardCollection("dbmedicalink.test", {"name": "hashed"})
db.test.insertMany([
  { id: 1, name: "Objet 1" },
  { id: 2, name: "Objet 2" },
  { id: 3, name: "Objet 3" },
  { id: 4, name: "Objet 4" },
  { id: 5, name: "Objet 5" },
```

```
  { id: 6, name: "Objet 6" },
  { id: 7, name: "Objet 7" }
])
```

Créer un user :

```
db.createUser({"user":"root","pwd":"s5a04a","roles":[{"role":"userAdminAnyDatabase","db":"admin"},{"role":"readWriteAnyDatabase","db":"admin"}]})
```

On fait en sorte que la base de données soit toujours accessible même si on ferme les terminaux.

Pour chaque membre du groupe pour sp, ss1, ss2, conf et le rooter :
Créer le dossier nécessaire :

```
mkdir -p  .config/systemd/user
```

Créer le fichier (pour chaque serveur pour) :

```
nano .config/systemd/user/s5a04a_sp.service
```

Mettre ça dedans (Le rooter est différent car on veut lancer une commande mongos et pas mongod) :

```
[Unit]
Description=Le serveur MongoDB principal de S5A04A.
After=network-online.target
Wants=network-online.target

[Service]
ExecStart=mongod --config /home/s5a04a/sp_mongod/mongod.conf
WorkingDirectory=/home/s5a04a/sp_mongod
StandardOutput=inherit
StandardError=inherit
Restart=always
RestartSec=10

[Install]
WantedBy=default.target
```

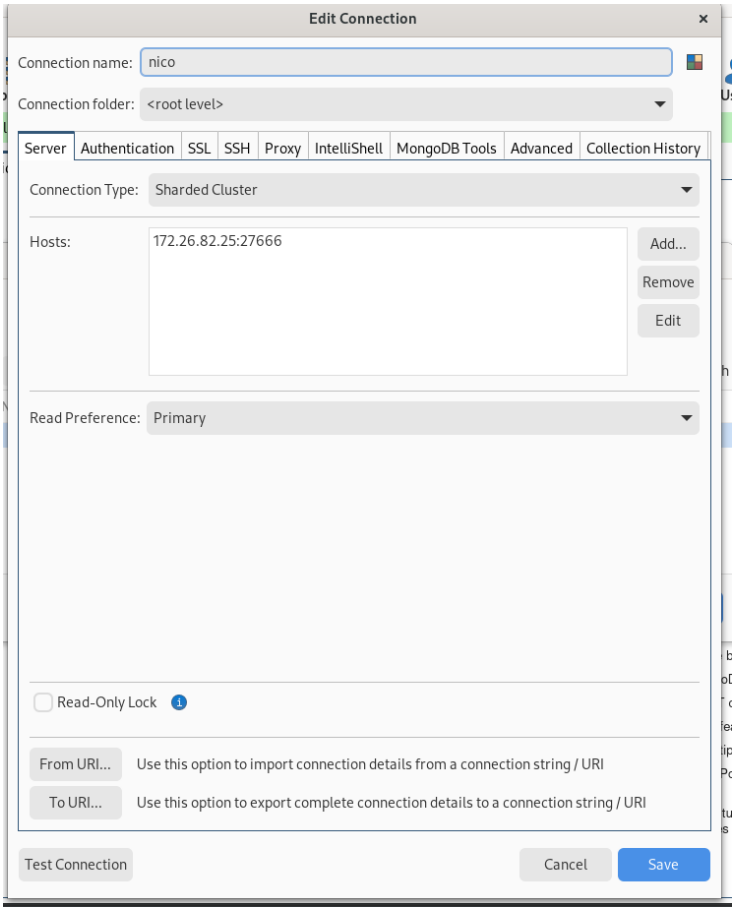
Faire les commandes suivante :

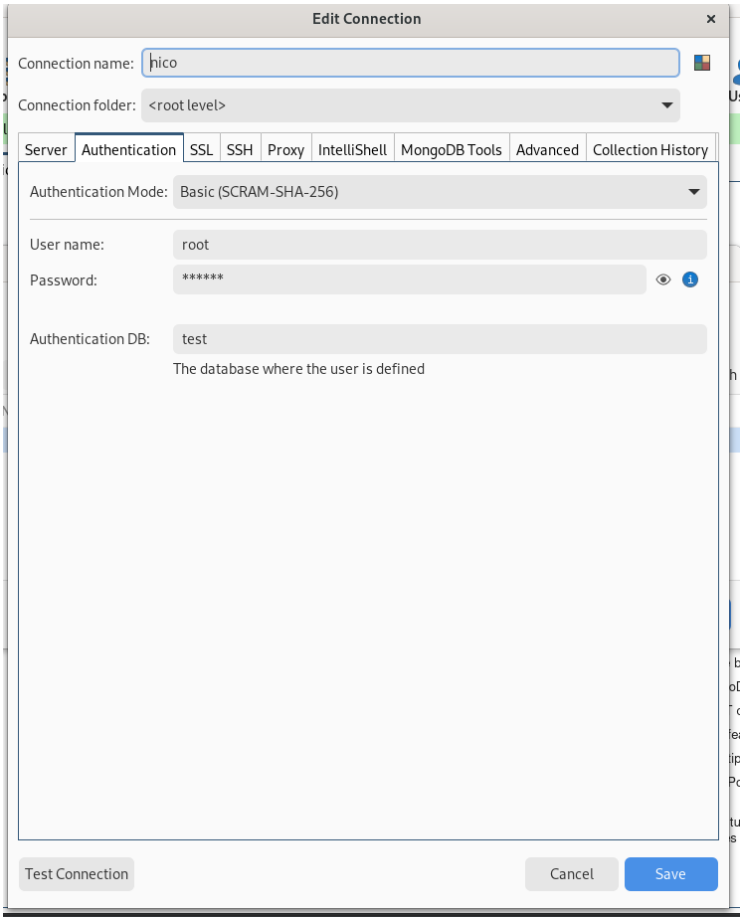
```
systemctl --user enable s5a04a_sp.service
systemctl --user start s5a04a_sp.service
systemctl --user status s5a04a_sp.service
```

Serveur n° : 172.28.82.25 id_user : s5aX ; mp : s5aX

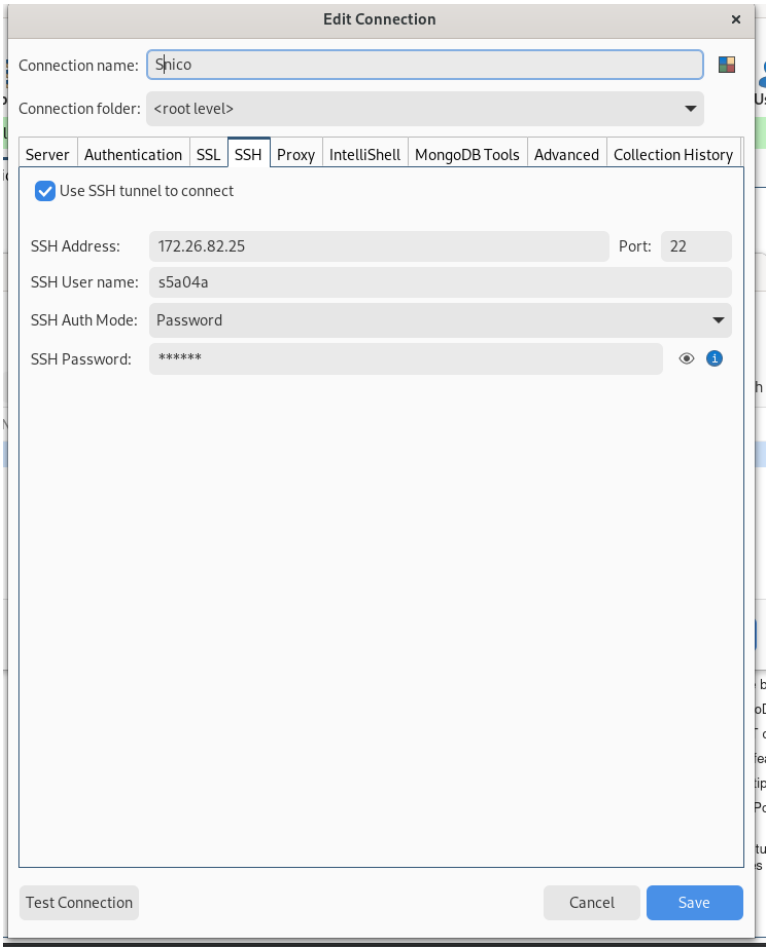
Binôme	id_user	N° port SP	N° port SS1	N° Port SS2	Sharding
	s5a01a	27018	27019	27020	
	s5a01b	27021	27022	27023	
	s5a02a	27024	27025	27026	
	s5a02b	270217	27028	27029	
	s5a03a	27030	27031	27032	
	s5a03b	27033	27034	27035	
	s5a04a	27036	27037	27038	
	s5a04b	27039	27040	27041	
	s5a05a	27042	27043	27044	
	s5a05b	27045	27046	27047	
	s5a06a	27048	27049	27050	
	s5a06b	27051	27052	27053	
	s5a07a	27054	27055	27056	
	s5a07b	27057	27058	27059	
	s5a08a	27060	27061	27062	
	s5a08b	27063	27064	27065	

Connexion à la bd via studio 3T sur le port du routeur :





mot de passe = mdp du user créer


Edit Connection
Connection name:
Shico
Connection folder:
<root level>
Server
Authentication
SSL
SSH
Proxy
IntelliShell
MongoDB Tools
Advanced
Collection History
Use SSH tunnel to connect
SSH Address:
172.26.82.25
Port:
22
SSH User name:
s5a04a
SSH Auth Mode:
Password
SSH Password:
******
Test Connection
Cancel
Save

Ça marche :)

Base de données en local

Pour la base de données local, nous avons choisis d'utiliser la bibliothèque androïd Room. Nous avons découpé notre gestion de la abse de données en 4 parties.
La première partie est la création de la base de données en singleton (car on ne veut qu'une seule connexion à la abse de données).

```
@Database(entities = [User::class, Medoc::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {

    E214406A
    abstract fun userDao(): UserDao
    E211556C
    abstract fun medocDao(): MedocDao


    E214406A +1
    companion object : SingletonHolder<AppDatabase, Context>({ it: Context
        Room.databaseBuilder(it.applicationContext, AppDatabase::class.java, name: "test.db").build()
    })
```

La deuxième partie est la création des entités de la base de donnes. Les entités vont se traduire en table dans la base de données.

```
@Entity
data class User(
    @PrimaryKey val uuid: String,
    @ColumnInfo(name = "statut") val statut: String?,
    @ColumnInfo(name = "nom") val nom: String?,
    @ColumnInfo(name = "prenom") val prenom: String?,
    @ColumnInfo(name = "dateDeNaissance") val dateDeNaissance: String?,
    @ColumnInfo(name = "email") val email: String?,
    @ColumnInfo(name = "password") var password: String?,
    @ColumnInfo(name = "isConnected") var isConnected: Boolean?,
)
```

Ainsi, l'entité User présenté au dessus deviendra une table user qui aura comme attributs tous les champs de la classe. De plus uuid sera une clé primaire.

La troisième partie est un doa servant à appeler la base de données pour récupérer les données tout en les renvoyant en tant qu'objet manipulable.

```kotlin
@Dao
interface UserDao {
    E214406A
    @Query("SELECT * FROM user")
    fun getAll(): List<User>

    E214406A
    @Query("SELECT * FROM user WHERE uuid IN (:uuid)")
    fun getById(uuid: String): List<User>

    E214406A +1
    @Query("SELECT * FROM user WHERE isConnected IN (:isConnected)")
    fun getByConnected(isConnected: Boolean = true): List<User>

    E214406A
    @Insert
    fun insertAll(vararg users: User)

    E214406A
    @Delete
    fun delete(user: User)

    E214406A
    @Update
    fun update(user: User)
```

La quatrième partie est une partie optionnel que j'ai choisis de mettre pour avoir une couche d'abstration et gérer mes potentiels erreurs.

```kotlin
class UserRepository(private val userDao: UserDao) {

    E214406A
    fun getAllUsers(): List<User> {
        return try {
            userDao.getAll()
        } catch (e: Exception) {
            emptyList()
        }
    }

    E214406A
    fun getOneUserById(uuid: String): List<User> {
        return try {
            userDao.getById(uuid)
        } catch (e: Exception) {
            emptyList()
        }
    }
```

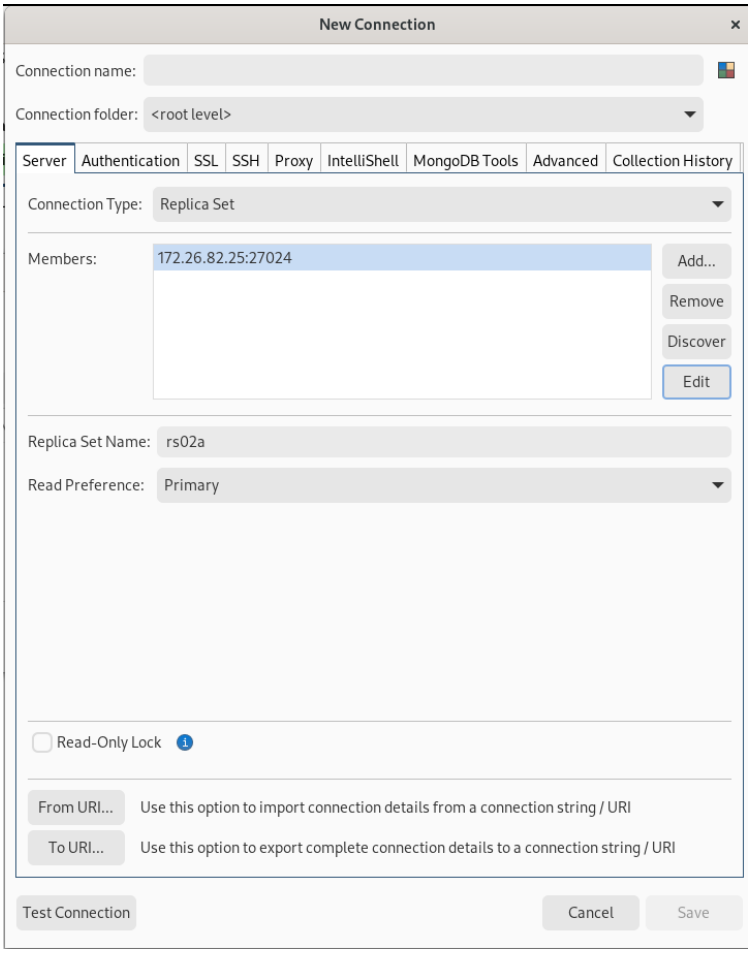
```kotlin
fun insertUser(user: User): Pair<Boolean, String> {
    val hashedPassword :String = hashPassword(user.password!!)
    user.password = hashedPassword
    return try {
        userDao.insertAll(user)
        Pair(true, "Success")
    } catch (e: SQLiteConstraintException) {
        Pair(false, "User already exists")
    } catch (e: SQLiteException) {
        Pair(false, "Database Error : ${e.message}")
    } catch (e: Exception) {
        Pair(false, "Unknown Error : ${e.message}")
    }
}
```

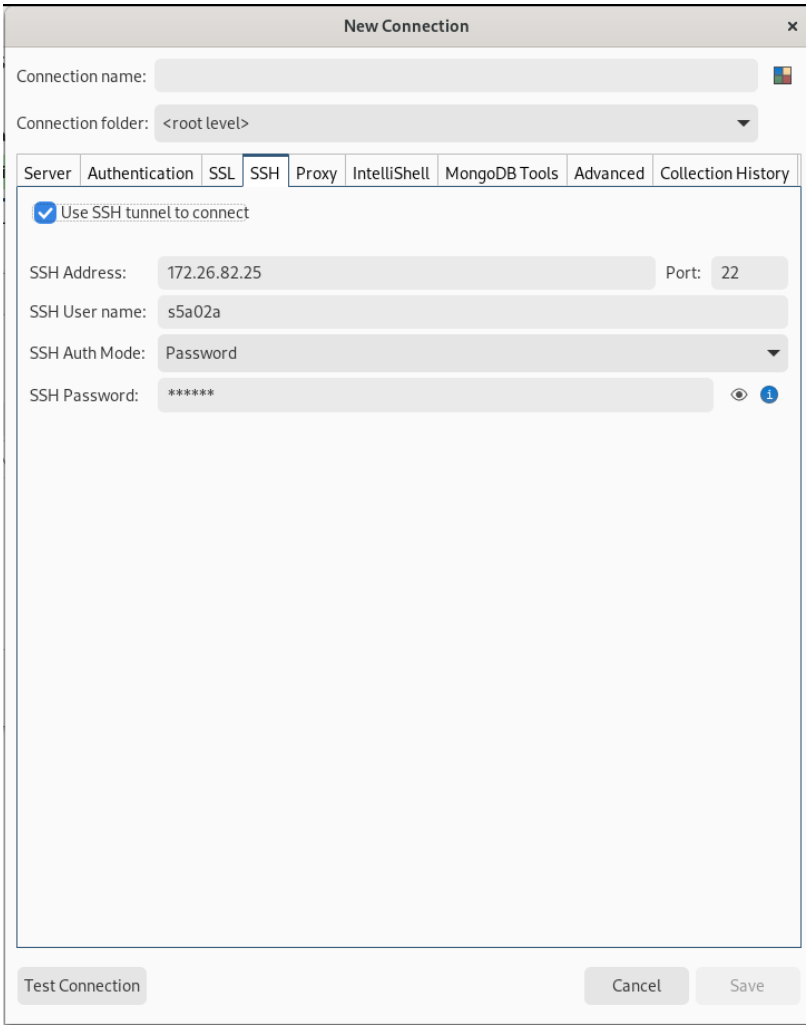
Nous avons plus de table que les tables présenté ici mais celles-ci n'étant pas encore développé, elles n'apparaîssent pas sur ce rapport.

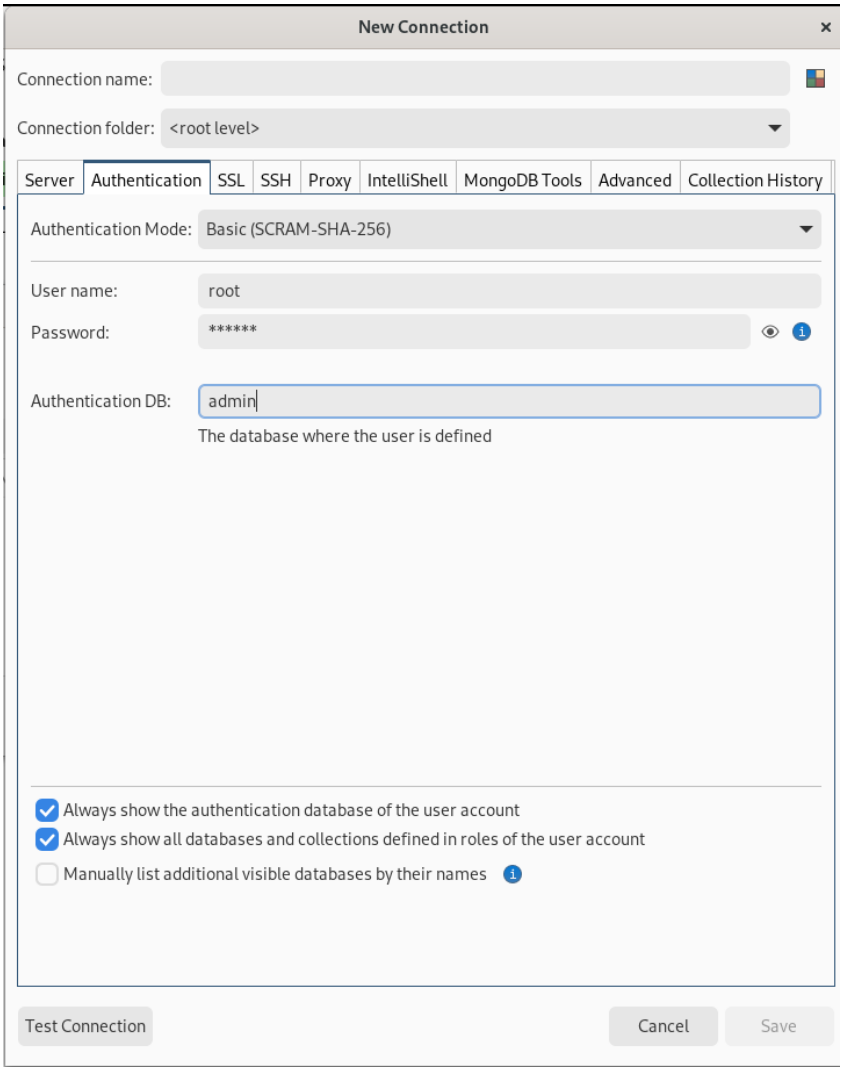
TODO : vérifier l'utilité de ça :

Dans Studio3T :
Server
- Connection Type : ReplicaSet
- Members : le serveur primaire
- Ajouter le ReplicaSetName

Authentication
- Authentication Mode : Basic ou Legacy (je sais plus)
- User name : root
- Pwd : s5a04a
- Authentication DB : test




New Connection
Connection name:
Connection folder:
<root level>
Server
Authentication
SSL
SSH
Proxy
IntelliShell
MongoDB Tools
Advanced
Collection History
Use SSH tunnel to connect
SSH Address:
172.26.82.25
Port:
22
SSH User name:
s5a02a
SSH Auth Mode:
Password
SSH Password:
******
Test Connection
Cancel
Save


New Connection
Connection name:
Connection folder:
<root level>
Server
Authentication
SSL
SSH
Proxy
IntelliShell
MongoDB Tools
Advanced
Collection History
Authentication Mode:
Basic (SCRAM-SHA-256)
User name:
root
Password:
******
Authentication DB:
admin
The database where the user is defined
Always show the authentication database of the user account
Always show all databases and collections defined in roles of the user account
Manually list additional visible databases by their names
Test Connection
Cancel
Save