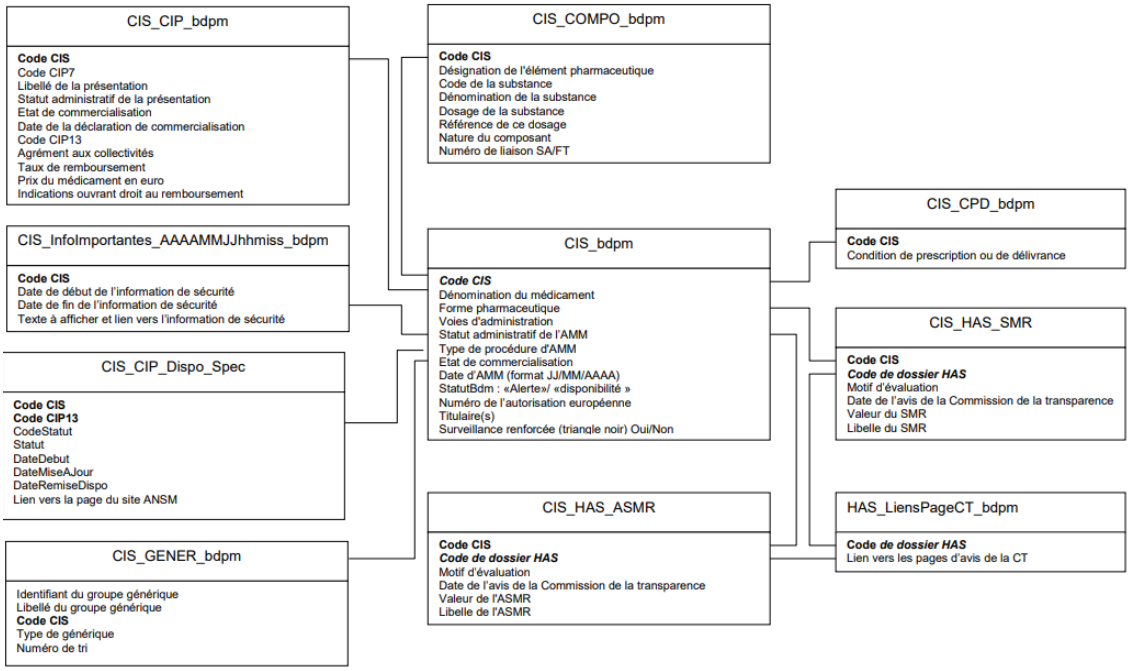
Romain Gouraud
Mathys Meunier
Nathan Ferry

SAE semestre 5 :

Nouveaux paradigmes de base de données

Traitement des données :

Les données du gouvernement n'étant pas du tout formatées pour être utilisées dans une base de données, il nous a fallu les traiter afin de les rendre exploitables. Pour ce faire, un programme écrit en Go se charge d'ordonner les données fournies.



Voici le schéma fourni par le gouvernement et par souci de simplicité, nous avons décidé de garder celui-ci. Les collections dans la base de données mongoDB sont donc les mêmes que les fichiers fournis.

Ainsi, voici comment les données sont traitées (exemple pour le fichier CIS_BDPM) :

```go
func LoadCis(path string) *[]CisData {  1 usage
    file, err := os.Open(path)
    if err != nil { log.Fatal(err) }
    reader := transform.NewReader(file, charmap.Windows1252.NewDecoder())
    scanner := bufio.NewScanner(reader)

    var dataList []CisData

    for scanner.Scan() {
        data := strings.Split(scanner.Text(),  sep: "\t")
        dataList = append(dataList, NewCisData(data))
    }

    return &dataList
}
```

Chaque fichier est chargé indépendamment par la fonction associée et traité afin de séparer correctement les données.
(Le nom des fonctions relatives à un fichier suit une convention simple : Nom de la fonction + fichier concerné)

Chaque ligne des fichiers est représentée par une structure dont les champs sont ceux donnés par le schéma vu plus haut.

```go
type CisData struct {  5 usages
    CIS                  int       `bson:"_id"`
    MedName              string    `bson:"medName"`
    PharmaForm           string    `bson:"pharmaForm"`
    AdministrationRoutes []string  `bson:"administrationRoutes"`
    MA                   string    `bson:"marketingAuth"`
    MAType               string    `bson:"marketingAuthType"`
    SellingState         string    `bson:"sellingState"`
    MADate               time.Time `bson:"marketingAuthDate"`
    BdmState             string    `bson:"bdmState"`
    EuAuthNumber         string    `bson:"EuAuthNumber"`
    Owner                []string  `bson:"owners"`
    ReinforcedSurveillance bool    `bson:"reinforcedSurveillance"`
}
```

```go
func InsertManyCis(database *mongo.Database, wg *sync.WaitGroup) {  1 usage
    defer wg.Done()

    ctx := context.TODO()

    collection := database.Collection( name: "cis")

    if err := collection.Drop(ctx); err != nil {
        log.Println( v...: "Error while dropping collection cis :\n", err)
        return
    }

    data := *LoadCis( path: "medicineData/CIS_bdpm.txt")
    documents := make([]interface{}, len(data))
    for i, _ := range data {
        documents[i] = data[i]
    }

    if _, err := collection.InsertMany(ctx, documents); err != nil {
        log.Println( v...: "Error while inserting collection cis :\n", err)
        return
    }
}
```

Ainsi, il est très simple d'insérer nos données, la librairie mongoDB se chargeant de convertir les structures afin de les stocker.

La base de données MongoDB :

Le serveur mis à disposition par l'IUT n'étant pas suffisant pour les besoins de ce projet, nous avons décidé d'utiliser un service tiers (DigitalOcean) afin d'héberger notre serveur.

Comme mentionné plus haut, les données traitées sont ensuite stockées dans mongoDB.
Nous avons mis en place 3 replicaSets de 3 serveurs afin d'assurer la bonne redondance des données :

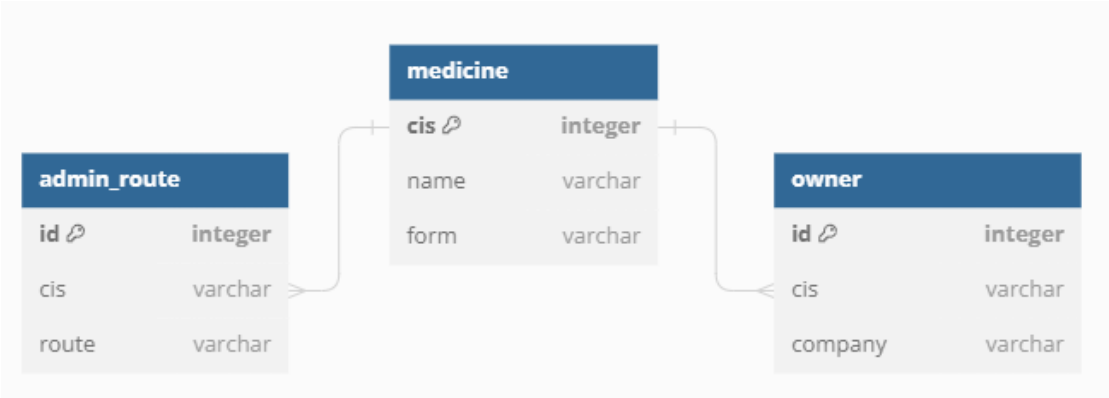
Nous avons ici l'exemple de l'un des trois replicaSet, où l'on distingue 3 membres, 1 primary et 2 secondary.

Nous avons également transformé chaque instance mongod en service afin de pouvoir s'assurer de leur fonctionnement en toutes situations et qu'ils se relancent automatiquement à chaque redémarrage.

```
[Unit]
Description=First database of the first replicaSet.
After=network-online.target
Wants=network-online.target
[Service]
ExecStart=mongod --config /root/mongoDBs/repSet1/mDB1/mongod.conf
WorkingDirectory=/root/mongoDBs/repSet1/mDB1
StandardOutput=inherit
StandardError=inherit
Restart=always
RestartSec=10
[Install]
WantedBy=default.target
```

Nous avons également mis en place un sharding entre ces replicSets afin d'assurer une meilleure redondance des données.

La base de données local :



Voici le schéma de la base de données en local. Nous avons cherché à ne prendre que les informations essentielles à la vérification des médicaments présents sur l'ordonnance. De ce fait, nous ne retrouvons que les informations relatives aux médicaments, aux moyens d'administrations et des propriétaires des médicaments.

RN Secure Storage :

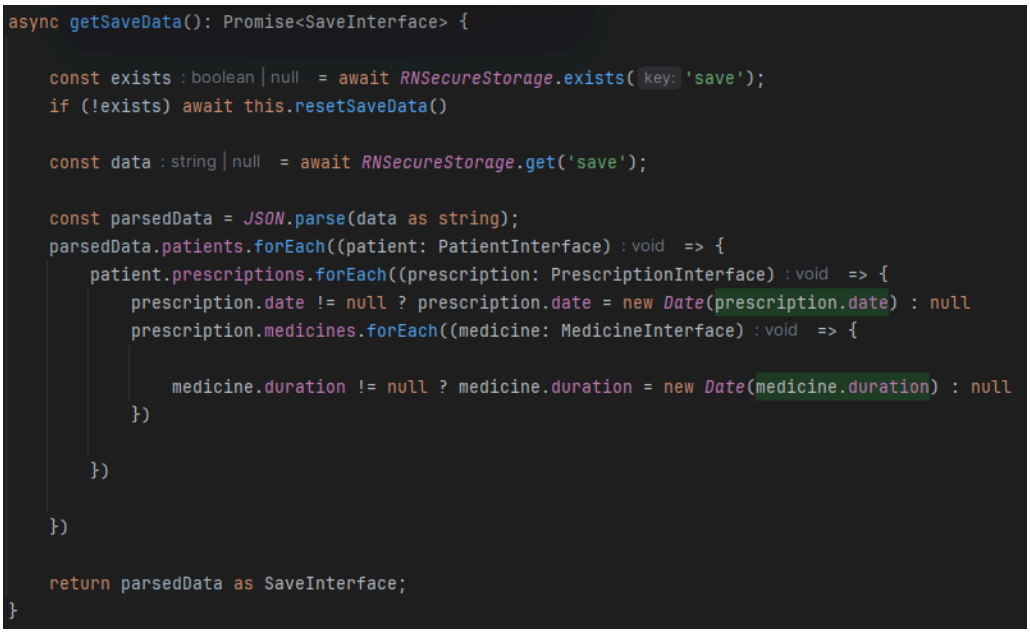
Afin de stocker les informations des utilisateurs, nous avons utilisé la librairie RN Secure Storage. Celle-ci permet un stockage local et simple des informations de l'utilisateur, tout en proposant une sécurité accrue grâce à son système de cryptage intégré.

```
import RNSecureStorage, { ACCESSIBLE } from 'rn-secure-storage';
import defaultSaveForTest from "@data/defaultSaveForTest.json";

5+ usages  Mathys MEUNIER +1
export default {
    async setSaveData(newSave: SaveInterface | ((oldSave: SaveInterface) => SaveInterface)) : Promise<void>  {
        typeof newSave !== "function"

        try {
            if (newSave instanceof Function) {
                const oldSave : SaveInterface  = await this.getSaveData();
                newSave = newSave(oldSave);
                RNSecureStorage.set( key: 'save', JSON.stringify(newSave),
                    options: { accessible: ACCESSIBLE.WHEN_UNLOCKED })
            } else {
                RNSecureStorage.set( key: 'save', JSON.stringify(newSave),
                    options: { accessible: ACCESSIBLE.WHEN_UNLOCKED })
            }
        } catch (error) {
            console.error("Erreur lors de la sauvegarde des données :", error);
        }
    },

    async resetSaveData() : Promise<void>  {
        RNSecureStorage
            .set( key: 'save', JSON.stringify( value: __DEV__ ? defaultSaveForTest : defaultSaveForTest),
                options: { accessible: ACCESSIBLE.WHEN_UNLOCKED })
    },
```



```
async getSaveData(): Promise<SaveInterface> {

    const exists : boolean | null  = await RNSecureStorage.exists( key: 'save');
    if (!exists) await this.resetSaveData()

    const data : string | null  = await RNSecureStorage.get('save');

    const parsedData = JSON.parse(data as string);
    parsedData.patients.forEach((patient: PatientInterface) : void  => {
        patient.prescriptions.forEach((prescription: PrescriptionInterface) : void  => {
            prescription.date != null ? prescription.date = new Date(prescription.date) : null
            prescription.medicines.forEach((medicine: MedicineInterface) : void  => {

                medicine.duration != null ? medicine.duration = new Date(medicine.duration) : null
            })

        })

    })

    return parsedData as SaveInterface;
}
```

Si la librairie fonctionne via un système de fichiers, elle agit comme une base de données dans la façon d'accéder aux données, ce qui rend son utilisation extrêmement simple tout en proposant les avantages d'une base de données telle que les contraintes d'intégrité.