Université Cote d'Azur/FDS(UEH)

MongoDB : Gestion d'un cabinet médical

Etudiants :
Douilly RODELY
Djimy SURLIN
Pierre Rubens MILORME
Bob Charlemagne PIERRE

Professeur :
Gabriel MOPOLO-MOKE

29 septembre 2024

Table des matières

1 Choix du sujet

1.1 SUJET

Gestion d'un cabinet médical

1.2 Description du sujet

Cette application pour un cabinet médical permet la gestion des patients, des médecins, des rendez-vous, des consultations, des prescriptions, des examens et de la facturation. Les patients peuvent être enregistrés avec leurs détails personnels, tandis que les médecins sont répertoriés avec leur spécialité respective. Les rendez-vous entre patients et médecins sont programmés. Chaque consultation est consignée et associée à un patient, un médecin et une facture. Les prescriptions médicales sont enregistrées et liées à la consultation correspondante. Les détails des examens sont stockés et également liés à la consultation. Chaque consultation génère une facture avec le montant total à payer.

En résumé, cette application fournit un système complet pour gérer les opérations quotidiennes d'un cabinet médical, optimisant le suivi des patients et la gestion des consultations et des facturations.

2 MCD MERISE

2.1 Dictionnaire de données MERISE

Entité : PATIENT					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id_Patient#	Identifiant unique du patient	Entier	Long	Valeur unique, non nulle	Oui
Num_Sec_Social	Numéro de Sécurité Sociale du patient	Chaîne	Caractère variable, 50 caractères max	non nulle	Non
Nom	Nom de famille du patient	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
Sexe	Sexe du patient	Chaîne	Caractère, 4 caractères max	Non nulle	Non
Date_naissance	Date de naissance du patient	Date		Non nulle	Non
Poids	Poids du patient	Décimal	Numérique	Non nulle	Non
Hauteur	Poids du patient	Décimal	Numérique	Non nulle	Non
listTelephones	Chaîne	Caractère variable, 50 caractères max		Non nulle	Non
listPrenoms	Prénoms du patient	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
Adresse	Adresse du patient	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non

Email	Email du patient	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
Entité : ALLERGIE					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id_Allergie#	Identifiant allergie	Entier	Long	Valeur unique, oui nulle	Oui
nom_allergie	nom de l'allergie	Chaine	Caractère variable, 50 caractères max	Valeur unique, oui nulle	non
Entité : ANTECEDENTS_MEDICAUX					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id_ant_medic#	Identifiant	Entier	Long	Valeur unique, oui nulle	Oui
description	description	Chaine	Caractère variable, 50 caractères max	non nulle	non
date_antec_medic	date de l'allergie	Date		non nulle	non
Entité : FACTURE					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id_Facture#	Identifiant unique de la facture	Entier	Long	Valeur unique, non nulle	Oui
Montant_Total	Montant total de la facture	Décimal	Numérique	Non nulle	Non
Date_Facture	Date d'émission de la facture	Date		Non nulle	Non
Entité : EXAMEN					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id_Examen#	Identifiant unique de l'examen	Entier	Long	Valeur unique, non nulle	Oui

Details_Examen	Détails de l'examen	Chaîne	200 caractères max	Non nulle	Non
Date_Examen	Date de l'examen	Date		Non nulle	Non
Entité : PRESCRIPTION					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id_Prescription#	Identifiant unique de la prescription	Entier	Long	Valeur unique, non nulle	Oui
Details_Prescription	Détails de la prescription	Chaîne	200 caractères max	Non null	Non
Date_Prescription	Date de la prescription	Date		Non null	Non
Entité : CONSULTATION					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id_Consultation#	Identifiant unique de la consultation	Entier	Long	Valeur unique, non nulle	Oui
Raison	Raison de la consultation	Chaîne	Caractère variable, 200 caractères max	Non nulle	Non
Diagnostic	Diagnostic au terme de la consultation	Chaîne	Caractère variable, 300 caractères max	Non nulle	Non
Date_Consultation	Date de la consultation	Date		Non nulle	Non
Entité : MEDECIN					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id_Medecin#	Identifiant unique du médecin	Entier	Long	Valeur unique, non nulle	Oui
listTelephones	telephones du médecin	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non

Sexe	Sexe du médecin	Chaîne	Caractère, 4 caractères max	Non nulle	Non
Spécialité	Spécialité du médecin	Chaîne	Caractère, 4 caractères max	Non nulle	Non
Nom	Nom de famille du médecin	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
listPrenoms	Prénoms du médecin	Chaîne		Non nulle	Non
Adresse	Adresse du médecin	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
Email	Email du médecin	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
listTelephones	Numéros de téléphone du médecin	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
Date_naissance	Date de naissance du médecin	Date		Non nulle	Non
CV	CV du médecin	CLOB		Non nulle	Non

TABLE 1 – Dictionnaire de données MERISE

2.2 La description textuelle des associations

3 Descriptions textuelles des associations

Dans cette section sont décrites les associations entres les différentes entités. Une association permet de mettre en relation deux ou plusieurs entités :

3.1 Association « Effectuer » entre MEDECIN et CONSULTATION

Cette association indique que chaque consultation est effectuée par un médecin. Un médecin peut effectuer plusieurs consultations, mais une consultation est effectuée par un seul médecin à la fois.

3.2 Association « Inclure » entre FACTURE et CONSULTATION

Cette association représente le fait qu'une facture inclut une consultation. Chaque consultation peut être incluse dans une seule facture, et réciproquement une facture peut référencer une seule consultation.

3.3 Association « Passer » entre PATIENT et CONSULTATION

Cette association signifie qu'un patient passe une consultation. Chaque consultation est passée par un seul patient, mais un patient peut passer plusieurs consultations.

3.4 Association « Necessiter » entre CONSULTATION et EXAMEN

Cette association indique que chaque consultation peut nécessiter plusieurs examens. Chaque examen est associé à une seule consultation.

3.5 Association « Recevoir » entre PATIENT et FACTURE

Cette association représente le fait qu'un patient peut recevoir une facture. Chaque facture est destinée à un seul patient, mais un patient peut recevoir plusieurs factures.

3.6 Association « Contenir » entre CONSULTATION et PRESCRIPTION

Cette association signifie qu'une consultation peut contenir plusieurs prescriptions. Chaque prescription est associée à une seule consultation.

3.7 Association « Avoir_Rendez_Vous » entre PATIENT et MEDECIN

Cette association indique que chaque rendez-vous est entre un patient et un médecin. Chaque rendez-vous est pris par un patient avec un médecin spécifique.

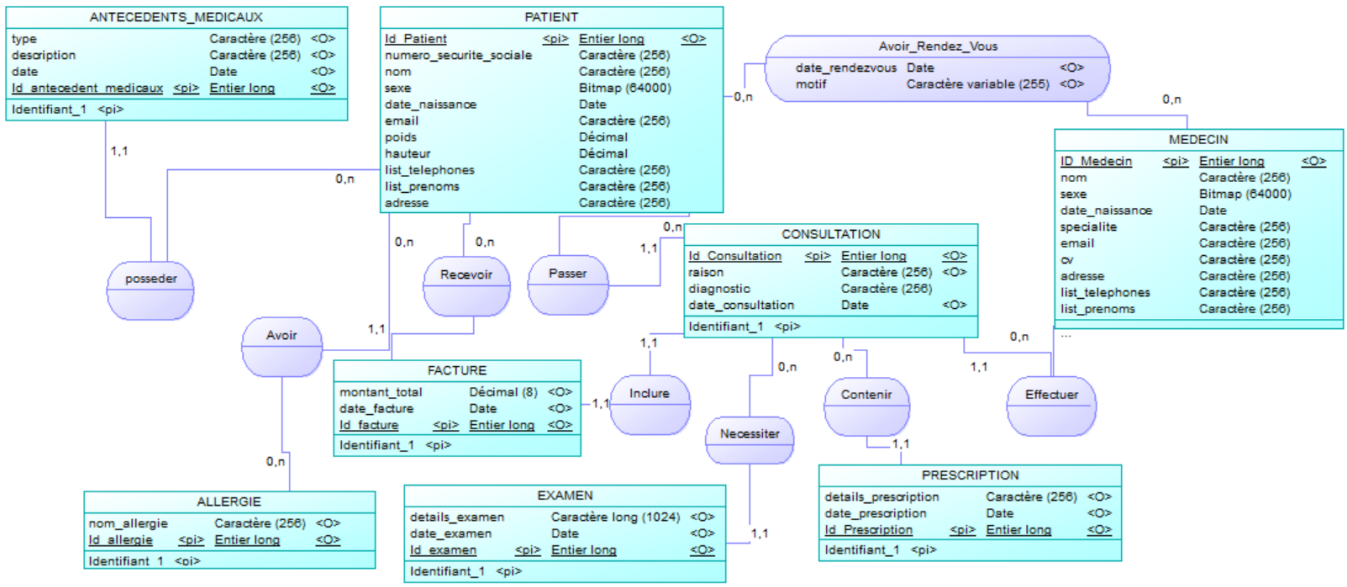
3.8 Association « Posséder » entre ANTECEDENTS_MEDICAUX et PATIENT

Cette association indique que chaque patient peut avoir plusieurs antécédents médicaux et un antécédent médical ne peut être lié qu'à un seul patient.

3.9 Association « Avoir » entre PATIENT et ALLERGIE

Cette association indique que chaque patient peut avoir plusieurs allergies et plusieurs patients peuvent avoir une même allergie.

3.10 La définition du Modèle Entité-Association MERISE



4 Conversion du MCD MERISE en des objets MONGODB et classes java

4.1 Spécification des Modèles de Documents

MongoDB est un système de base de données NoSQL orienté document, qui stocke les données sous forme de documents JSON/BSON. Les objets sont donc organisés en collections et chaque document peut être indépendant en termes de structure.
Dans notre cas de figure nous avons identifié deux collections (Medecin et Patient) suite à la conversion du MCD en objets MONGODB en respectant les préconisations et règles d'organisation des bases de données NOSQL et MONGODB en particulier, notamment en dénormalisant les données autant que possible.

4.1.1 Collection "Medecin" (document JSON/BSON)

```
{
  "_id": "ObjectId",
  "nom": "String",
  "sexe": "String",
  "date_naissance": "Date",
  "specialite": "String",
  "email": "String",
```

```
      "cv": "String",
      "list_telephones": ["String"],
      "list_prenoms": ["String"],
      "adresse": {
        "numero": "Number",
        "rue": "String",
        "code_postal": "Number",
        "ville": "String"
      },
      "list_rendez_vous": [
        {
          "patient": "ObjectId",
          "date_rendez_vous": "Date",
          "motif": "String"
        }
      ],
      "list_consultations": [
        {
          "patient": "ObjectId",
          "raison": "String",
          "diagnostic": "String",
          "date_consultation": "Date",
          "list_examens": [
            {
              "details_examen": "String",
              "date_examen": "Date"
            }
          ],
          "list_prescriptions": [
            {
              "details_prescription": "String",
              "date_prescription": "Date"
            }
          ]
        }
      ]
    }
```

Exemple de document de médecin

```
{
  "_id": "66f0820f580fe26567ae4f8d",
  "nom": "Kertzmann",
  "sexe": "Other",
  "date_naissance": "1997-08-08",
  "specialite": "Gynecologist",
  "email": "susy_mohr@metz.test",
  "cv": "Qui doloremque debitis et autem placeat natus qui quia
     voluptatibus.",
  "list_telephones": [
    "409.780.8066",
```

```
        "740-880-4984"
      ],
      "list_prenoms": [
        "Beau",
        "Jesica"
      ],
      "adresse": {
        "numero": "7777",
        "rue": "Francesco Lights",
        "code_postal": "32176-8834",
        "ville": "Isabellashire"
      },
      "list_rendez_vous": [
        {
          "patient": "66f0820f580fe26567ae4f8e",
          "date_rendez_vous": "2024-10-04",
          "motif": "Quaerat doloremque nobis."
        }
      ],
      "list_consultations": [
        {
          "patient": "66f0820f580fe26567ae4f8f",
          "raison": "Magnam dolorum suscipit.",
          "diagnostic": "Et nihil pariatur minima quam.",
          "date_consultation": "2024-08-24",
          "list_examens": [
            {
              "details_examen": "Rerum in et similique voluptates.",
              "date_examen": "2024-09-07"
            }
          ],
          "list_prescriptions": [
            {
              "details_prescription": "Et et non facere non.",
              "date_prescription": "2024-09-14"
            }
          ]
        }
      ]
    }
```


NB : Le modèle n'est pas un schéma strict. Les documents ne doivent pas nécessairement respecter en tout point le modèle de document. Le document de médecin ci-dessus est présenté à titre indicatif.

4.1.2 Collection "Patient" (document JSON)


```
{
  "_id": "ObjectId",
  "numero_securite_sociale": "String",
```

```
  "nom": "String",
  "sexe": "String",
  "date_naissance": "Date",
  "email": "String",
  "poids": "Number",
  "hauteur": "Number",
  "list_telephones": ["String"],
  "list_prenoms": ["String"],
  "adresse": {
    "numero": "Number",
    "rue": "String",
    "code_postal": "Number",
    "ville": "String"
  },
  "list_rendez_vous": [
    {
      "medecin": "ObjectId",
      "date_rendez_vous": "Date",
      "motif": "String"
    }
  ],
  "list_consultations": [
    {
      "medecin": "ObjectId",
      "raison": "String",
      "diagnostic": "String",
      "date_consultation": "Date",
      "list_examens": [
        {
          "details_examen": "String",
          "date_examen": "Date"
        }
      ],
      "list_prescriptions": [
        {
          "details_prescription": "String",
          "date_prescription": "Date"
        }
      ],
      "facture": {
        "Montant_Total": "Number",
        "date_facture": "Date"
      }
    }
  ],
  "antecedents_medicaux": [
    {
      "type": "String",
      "description": "String",
      "date": "Date"
```

```
    }
  ],
  "allergies": ["String"]
}
```

Exemple de document de patient

```
{
  "_id": "66f08628c97d6838ff4b3b48",
  "numero_securite_sociale": "847-37-8324",
  "nom": "Abernathy",
  "sexe": "O",
  "date_naissance": "2009-08-12",
  "email": "matthew_hand@donnelly.test",
  "poids": 77.64,
  "hauteur": 1.66,
  "list_telephones": [
    "357-824-3744",
    "(365) 603-4456"
  ],
  "list_prenoms": [
    "Tom",
    "Jamar"
  ],
  "adresse": {
    "numero": "2945",
    "rue": "Walter Mills",
    "code_postal": "28820",
    "ville": "Friesenport"
  },
  "list_rendez_vous": [
    {
      "medecin": "66f08628c97d6838ff4b3b49",
      "date_rendez_vous": "2024-10-02",
      "motif": "Qui culpa delectus."
    }
  ],
  "list_consultations": [
    {
      "medecin": "66f08628c97d6838ff4b3b4a",
      "raison": "Et qui labore.",
      "diagnostic": "Porro totam ut ut possimus.",
      "date_consultation": "2024-09-21",
      "list_examens": [
        {
          "details_examen": "Laboriosam quo vitae quam quasi.",
          "date_examen": "2024-09-11"
        }
      ],
      "list_prescriptions": [
        {
```

```
            "details_prescription": "Tenetur sunt quae ex iusto.",
            "date_prescription": "2024-09-21"
          }
        ],
        "facture": {
          "Montant_Total": 329.15,
          "date_facture": "2024-09-16"
        }
      },
      {
        "medecin": "66f08628c97d6838ff4b3b4b",
        "raison": "Exercitationem animi ex.",
        "diagnostic": "Perspiciatis sequi placeat dolor amet.",
        "date_consultation": "2024-09-20",
        "list_examens": [
          {
            "details_examen": "Est eius excepturi fuga tenetur.",
            "date_examen": "2024-09-11"
          }
        ],
        "list_prescriptions": [
          {
            "details_prescription": "Quis aut dolores quia rerum.",
            "date_prescription": "2024-09-15"
          }
        ],
        "facture": {
          "Montant_Total": 480.4,
          "date_facture": "2024-09-19"
        }
      }
    ],
    "antecedents_medicaux": [
      {
        "type": "Chronic",
        "description": "Ducimus repellat eos labore quis.",
        "date": "2023-08-05"
      },
      {
        "type": "Acute",
        "description": "Eum dolorem labore et occaecati.",
        "date": "2024-06-21"
      }
    ],
    "allergies": [
      "eligendi",
      "ab"
    ]
  }
```

NB : Le modèle n'est pas un schéma strict. Les documents ne doivent pas nécessairement respecter en tout point le modèle de document. Le document de patient ci-dessus est présenté à titre indicatif.

4.2 Spécification des classes et des méthodes JAVA

4.2.1 Méthodes CRUD

1. Définition des Classes :

```java
// Classe Medecin
package org.example.entity;

import org.bson.types.ObjectId;

import java.util.Date;
import java.util.List;

public class Medecin {
    private ObjectId _id;
    private String nom;
    private String sexe;
    private Date dateNaissance;
    private String specialite;
    private String email;
    private String cv;
    private List<String> listTelephones;
    private List<String> listPrenoms;
    private Adresse adresse;
    private List<RendezVous> listRendezVous;
    private List<Consultation> listConsultations;

    public ObjectId getId() {
        return _id;
    }

    public void setId(ObjectId id) {
        this._id = id;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String getSexe() {
        return sexe;
    }
```

```
42
43      public void setSexe(String sexe) {
44          this.sexe = sexe;
45      }
46
47      public Date getDateNaissance() {
48          return dateNaissance;
49      }
50
51      public void setDateNaissance(Date dateNaissance) {
52          this.dateNaissance = dateNaissance;
53      }
54
55      public String getSpecialite() {
56          return specialite;
57      }
58
59      public void setSpecialite(String specialite) {
60          this.specialite = specialite;
61      }
62
63      public String getEmail() {
64          return email;
65      }
66
67      public void setEmail(String email) {
68          this.email = email;
69      }
70
71      public String getCv(){
72          return this.cv;
73      }
74
75      public void setCv(String cv){
76          this.cv = cv;
77      }
78
79      public List<String> getListTelephones() {
80          return listTelephones;
81      }
82
83      public void setListTelephones(List<String> listTelephones
            ) {
84          this.listTelephones = listTelephones;
85      }
86
87      public List<String> getListPrenoms() {
88          return listPrenoms;
89      }
90
```

```
91      public void setListPrenoms(List<String> listPrenoms) {
92          this.listPrenoms = listPrenoms;
93      }
94
95      public Adresse getAdresse() {
96          return adresse;
97      }
98
99      public void setAdresse(Adresse adresse) {
100         this.adresse = adresse;
101     }
102
103     public List<RendezVous> getListRendezVous() {
104         return listRendezVous;
105     }
106
107     public void setListRendezVous(List<RendezVous>
            listRendezVous) {
108         this.listRendezVous = listRendezVous;
109     }
110
111     public List<Consultation> getListConsultations() {
112         return listConsultations;
113     }
114
115     public void setListConsultations(List<Consultation>
            listConsultations) {
116         this.listConsultations = listConsultations;
117     }
118
119     public String toString(){
120         return "Medecin => nom : " + this.nom + ", sexe : " +
                this.sexe + ", specialite : " + this.specialite +
                ", email : " + this.email;
121     }
122 }
123
124 // Classe Patient
125 package org.example.entity;
126
127 import org.bson.types.ObjectId;
128
129 import java.util.Date;
130 import java.util.List;
131
132 public class Patient {
133     private ObjectId _id;
134     private String numeroSecuriteSociale;
135     private String nom;
136     private String sexe;
```

```
137     private Date dateNaissance;
138     private String email;
139     private double poids;
140     private double hauteur;
141     private List<String> listTelephones;
142     private List<String> listPrenoms;
143     private Adresse adresse;
144     private List<RendezVous> listRendezVous;
145     private List<Consultation> listConsultations;
146     private List<AntecedentMedical> antecedentsMedicaux;
147     private List<String> allergies;
148 
149     public ObjectId getId() {
150         return _id;
151     }
152 
153     public void setId(ObjectId id) {
154         this._id = id;
155     }
156 
157     public String getNumeroSecuriteSociale() {
158         return numeroSecuriteSociale;
159     }
160 
161     public void setNumeroSecuriteSociale(String
            numeroSecuriteSociale) {
162         this.numeroSecuriteSociale = numeroSecuriteSociale;
163     }
164 
165     public String getNom() {
166         return nom;
167     }
168 
169     public void setNom(String nom) {
170         this.nom = nom;
171     }
172 
173     public String getSexe() {
174         return sexe;
175     }
176 
177     public void setSexe(String sexe) {
178         this.sexe = sexe;
179     }
180 
181     public Date getDateNaissance() {
182         return dateNaissance;
183     }
184 
185     public void setDateNaissance(Date dateNaissance) {
```

```
186         this.dateNaissance = dateNaissance;
187     }
188 
189     public String getEmail() {
190         return email;
191     }
192 
193     public void setEmail(String email) {
194         this.email = email;
195     }
196 
197     public double getPoids() {
198         return poids;
199     }
200 
201     public void setPoids(double poids) {
202         this.poids = poids;
203     }
204 
205     public double getHauteur() {
206         return hauteur;
207     }
208 
209     public void setHauteur(double hauteur) {
210         this.hauteur = hauteur;
211     }
212 
213     public List<String> getListTelephones() {
214         return listTelephones;
215     }
216 
217     public void setListTelephones(List<String> listTelephones
        ) {
218         this.listTelephones = listTelephones;
219     }
220 
221     public List<String> getListPrenoms() {
222         return listPrenoms;
223     }
224 
225     public void setListPrenoms(List<String> listPrenoms) {
226         this.listPrenoms = listPrenoms;
227     }
228 
229     public Adresse getAdresse() {
230         return adresse;
231     }
232 
233     public void setAdresse(Adresse adresse) {
234         this.adresse = adresse;
```

```
235     }
236
237     public List<RendezVous> getListRendezVous() {
238         return listRendezVous;
239     }
240
241     public void setListRendezVous(List<RendezVous>
            listRendezVous) {
242         this.listRendezVous = listRendezVous;
243     }
244
245     public List<Consultation> getListConsultations() {
246         return listConsultations;
247     }
248
249     public void setListConsultations(List<Consultation>
            listConsultations) {
250         this.listConsultations = listConsultations;
251     }
252
253     public List<AntecedentMedical> getAntecedentsMedicaux() {
254         return antecedentsMedicaux;
255     }
256
257     public void setAntecedentsMedicaux(List<AntecedentMedical
            > antecedentsMedicaux) {
258         this.antecedentsMedicaux = antecedentsMedicaux;
259     }
260
261     public List<String> getAllergies() {
262         return allergies;
263     }
264
265     public void setAllergies(List<String> allergies) {
266         this.allergies = allergies;
267     }
268
269     public String toString(){
270         return this.nom + this.sexe + this.
                numeroSecuriteSociale + this.email;
271     }
272 }
273
274 // Classe Examen
275 package org.example.entity;
276
277 import java.util.Date;
278
279 public class Examen {
280     private String detailsExamen;
```

```
281     private Date dateExamen;
282 
283     public String getDetailsExamen() {
284         return detailsExamen;
285     }
286 
287     public void setDetailsExamen(String detailsExamen) {
288         this.detailsExamen = detailsExamen;
289     }
290 
291     public Date getDateExamen() {
292         return dateExamen;
293     }
294 
295     public void setDateExamen(Date dateExamen) {
296         this.dateExamen = dateExamen;
297     }
298 }
299 
300 // Classe Prescription
301 package org.example.entity;
302 
303 import java.util.Date;
304 
305 public class Prescription {
306     private String detailsPrescription;
307     private Date datePrescription;
308 
309     public String getDetailsPrescription() {
310         return detailsPrescription;
311     }
312 
313     public void setDetailsPrescription(String
        detailsPrescription) {
314         this.detailsPrescription = detailsPrescription;
315     }
316 
317     public Date getDatePrescription() {
318         return datePrescription;
319     }
320 
321     public void setDatePrescription(Date datePrescription) {
322         this.datePrescription = datePrescription;
323     }
324 }
325 
326 // Classe Facture
327 package org.example.entity;
328 
329 import java.util.Date;
```

```
330
331 public class Facture {
332     private double montantTotal;
333     private Date dateFacture;
334
335     public double getMontantTotal() {
336         return montantTotal;
337     }
338
339     public void setMontantTotal(double montantTotal) {
340         this.montantTotal = montantTotal;
341     }
342
343     public Date getDateFacture() {
344         return dateFacture;
345     }
346
347     public void setDateFacture(Date dateFacture) {
348         this.dateFacture = dateFacture;
349     }
350 }
351
352 // Classe AntecedentMedical
353 package org.example.entity;
354
355 import java.util.Date;
356
357 public class AntecedentMedical {
358     private String type;
359     private String description;
360     private Date date;
361
362     public String getType() {
363         return type;
364     }
365
366     public void setType(String type) {
367         this.type = type;
368     }
369
370     public String getDescription() {
371         return description;
372     }
373
374     public void setDescription(String description) {
375         this.description = description;
376     }
377
378     public Date getDate() {
379         return date;
```

```
380         }
381
382         public void setDate(Date date) {
383             this.date = date;
384         }
385     }
386
387     // Classe Adresse
388     package org.example.entity;
389
390     public class Adresse {
391         private int numero;
392         private String rue;
393         private int codePostal;
394         private String ville;
395
396         public int getNumero() {
397             return numero;
398         }
399
400         public void setNumero(int numero) {
401             this.numero = numero;
402         }
403
404         public String getRue() {
405             return rue;
406         }
407
408         public void setRue(String rue) {
409             this.rue = rue;
410         }
411
412         public int getCodePostal() {
413             return codePostal;
414         }
415
416         public void setCodePostal(int codePostal) {
417             this.codePostal = codePostal;
418         }
419
420         public String getVille() {
421             return ville;
422         }
423
424         public void setVille(String ville) {
425             this.ville = ville;
426         }
427     }
428
429     // Classe RendezVous
```

```java
package org.example.entity;

import java.util.Date;

public class RendezVous {
    private Medecin medecin;
    private Patient patient;
    private Date dateRendezVous;
    private String motif;

    public Medecin getMedecin() {
        return medecin;
    }

    public void setMedecin(Medecin medecin) {
        this.medecin = medecin;
    }

    public Patient getPatient() {
        return patient;
    }

    public void setPatient(Patient patient) {
        this.patient = patient;
    }

    public Date getDateRendezVous() {
        return dateRendezVous;
    }

    public void setDateRendezVous(Date dateRendezVous) {
        this.dateRendezVous = dateRendezVous;
    }

    public String getMotif() {
        return motif;
    }

    public void setMotif(String motif) {
        this.motif = motif;
    }
}

// Classe Consultation
package org.example.entity;

import java.util.Date;
import java.util.List;

public class Consultation {
```

```
480     private Medecin medecin;
481     private Patient patient;
482     private String raison;
483     private String diagnostic;
484     private Date dateConsultation;
485     private List<Examen> listExamens;
486     private List<Prescription> listPrescriptions;
487     private Facture facture;
488 
489     public Medecin getMedecin() {
490         return medecin;
491     }
492 
493     public void setMedecin(Medecin medecin) {
494         this.medecin = medecin;
495     }
496 
497     public Patient getPatient() {
498         return patient;
499     }
500 
501     public void setPatient(Patient patient) {
502         this.patient = patient;
503     }
504 
505     public String getRaison() {
506         return raison;
507     }
508 
509     public void setRaison(String raison) {
510         this.raison = raison;
511     }
512 
513     public String getDiagnostic() {
514         return diagnostic;
515     }
516 
517     public void setDiagnostic(String diagnostic) {
518         this.diagnostic = diagnostic;
519     }
520 
521     public Date getDateConsultation() {
522         return dateConsultation;
523     }
524 
525     public void setDateConsultation(Date dateConsultation) {
526         this.dateConsultation = dateConsultation;
527     }
528 
529     public List<Examen> getListExamens() {
```

```
530             return listExamens;
531         }
532
533         public void setListExamens(List<Examen> listExamens) {
534             this.listExamens = listExamens;
535         }
536
537         public List<Prescription> getListPrescriptions() {
538             return listPrescriptions;
539         }
540
541         public void setListPrescriptions(List<Prescription>
                listPrescriptions) {
542             this.listPrescriptions = listPrescriptions;
543         }
544
545         public Facture getFacture() {
546             return facture;
547         }
548
549         public void setFacture(Facture facture) {
550             this.facture = facture;
551         }
552     }
```

2. Méthode pour insérer un médecin :

```
1       public void insertOneMedecin(Medecin medecin) {
2           Document doc = new Document("nom", medecin.getNom())
3                   .append("sexe", medecin.getSexe())
4                   .append("date_naissance", medecin.
                        getDateNaissance())
5                   .append("specialite", medecin.getSpecialite()
                        )
6                   .append("email", medecin.getEmail());
7           medecinCollection.insertOne(doc);
8       }
```

3. Méthodes pour insérer plusieurs médecins :

```
1       public void insertMedecins(List<Medecin> medecins) {
2           List<Document> documents = new ArrayList<>();
3           for (Medecin medecin : medecins) {
4               Document doc = new Document("nom", medecin.getNom
                    ())
5                       .append("sexe", medecin.getSexe())
6                       .append("date_naissance", medecin.
                            getDateNaissance())
7                       .append("specialite", medecin.
                            getSpecialite())
```

```
                    .append("email", medecin.getEmail());
            documents.add(doc);
        }
        medecinCollection.insertMany(documents);
    }
```

4. Méthode pour lire un medecin par son id

```
public Medecin getMedecinById(String id) {
    MongoCollection<Document> medecinCollection = database.
       getCollection("Medecin");
    Document query = new Document("_id", new ObjectId(id));
    Document result = medecinCollection.find(query).first();
    if (result != null) {
        return new Medecin(result.getString("nom"), result.
           getString("specialite"));
    }
    return null;
}
```

5. Méthode pour mettre à jour un medecin

```
public void updateMedecin(String id, String email) {
    MongoCollection<Document> medecinCollection = database.
       getCollection("Medecin");
    medecinCollection.updateOne(Filters.eq("_id", new
       ObjectId(id)), Updates.set("email", email));
}
```

6. Méthode pour supprimer un medecin

```
public void deleteMedecin(String id) {
    MongoCollection<Document> medecinCollection = database.
       getCollection("Medecin");
    medecinCollection.deleteOne(Filters.eq("_id", new
       ObjectId(id)));
}
```

4.2.2 Indexation secondaire

1. Création d'index secondaire sur le champ "nom"

```
medecinCollection.createIndex(Indexes.ascending("nom"));
```

4.2.3 Méthode de Consultation (Jointure, Groupement)

1. Groupement des medecins par specialité

```java
public void groupMedecinsBySpecialite() {
    MongoCollection<Document> medecinCollection = database.
       getCollection("Medecin");
    AggregateIterable<Document> groupResult =
       medecinCollection.aggregate(Arrays.asList(
        Aggregates.group("$specialite", Accumulators.sum("
           total", 1))
    ));

    for (Document doc : groupResult) {
        System.out.println(doc.toJson());
    }
}
```

5 Compléments sur le moteur NoSql MONGODB

5.1 Modèles de données supportés

5.1.1 Modèle de données : Document (JSON/BSON)

MongoDB stocke les données sous forme de documents JSON (ou BSON, une version binaire de JSON). Chaque document est un ensemble de paires clé-valeur. Cela permet de gérer des structures complexes, des tableaux et des sous-documents imbriqués. Le modèle est flexible, sans schéma strict (schema-less).

5.1.2 Usage

Convient aux cas où la structure des données peut varier ou être modifiée fréquemment.

5.2 Procédure d'installation du moteur et des utilitaires

5.2.1 Installation

Relativement simple. MongoDB offre des packages pour les systèmes d'exploitation populaires (Linux, Windows, macOS). Il suffit d'installer MongoDB et de démarrer le service MongoDB ("mongod").

5.2.2 Installation (Windows)

Voici un guide étape par étape pour installer MongoDB sur un système Windows :

1. Téléchargement de `MongoDB` :
   (a) Accédez à la page de téléchargement de MongoDB Community Edition.
   (b) Choisissez la version de MongoDB compatible avec votre système (en général, Windows 64-bit).
   (c) Téléchargez l'installeur `.msi` pour Windows.
2. Installation de `MongoDB` :
   (a) Ouvrez le fichier `.msi` téléchargé pour lancer l'installation.

(b) Suivez les étapes de l'assistant d'installation :

i. Sélectionnez l'option `Complete` pour installer `MongoDB` avec les options par défaut.

ii. Cochez l'option `Install MongoDB as a Service` afin que `MongoDB` démarre automatiquement au démarrage de Windows.

iii. Vous pouvez également installer `MongoDB Compass`, l'interface graphique de `MongoDB` (facultatif mais recommandé pour gérer vos bases de données).

3. Configurer MongoDB

(a) Après l'installation, vous devez configurer `MongoDB` :

i. Créer le répertoire de données : MongoDB utilise le dossier `C:\data\db`

Par défaut pour stocker ses bases de données. Vous devez créer ce répertoire manuellement.

i. Ouvrez l'Explorateur de fichiers.

ii. Allez dans le disque `C:\`.

iii. Créez un dossier nommé `data`.

iv. À l'intérieur de `data`, créez un sous-dossier nommé `db` : `C:\data\db`

4. Vérifier que MongoDB est bien installé : Ouvrez un terminal (invité de commandes) et tapez la commande suivante pour vérifier que MongoDB est correctement installé :

```
mongod --version
```

5. Lancer MongoDB : MongoDB devrait s'exécuter en tant que service, mais vous pouvez aussi le démarrer manuellement.

— Via l'invite de commandes : Tapez cette commande `mongod` pour démarrer MongoDB. MongoDB sera lancé et écoutera par défaut sur le port `27017`.

— Accéder à MongoDB avec le shell. Ouvrez un nouvel onglet dans l'invite de commandes et tapez : `mongo`. Cela lancera le shell MongoDB, où vous pourrez interagir avec votre base de données.

— Utiliser MongoDB Compass (facultatif) Si vous avez installé MongoDB Compass, vous pouvez l'utiliser pour interagir avec MongoDB via une interface graphique en lançant l'application.

Si `MongoDB` a été installé en tant que service, vous pouvez démarrer, arrêter ou redémarrer `MongoDB` directement depuis le gestionnaire de services de Windows :

(a) Ouvrez "Services" (tapez `services.msc` dans l'Invite de commandes).

(b) Recherchez le service "MongoDB".

(c) Cliquez avec le bouton droit pour démarrer, arrêter ou redémarrer le service.

5.2.3 Installation (Linux)

```
bash > sudo apt-get install -y mongodb & systemctl start mongodb
```

5.2.4 Installation via Docker

1. Installer *Docker Desktop*.
2. Dans un terminal ou invité de commandes : `docker run -name mongodb -p 27017:27017 -d mongodb/mongodb-community-server:latest`

5.2.5 Utilitaires

1. Outils de ligne de commande comme "mongo", "mongodump", et "mongoimport".
   — Mongo fait référence au client interactif utilisé pour interagir avec une base de données `MongoDB`.
      — Vous pouvez l'utiliser pour exécuter des commandes MongoDB, écrire des requêtes, gérer des collections, etc.
      — Commande typique pour se connecter à une base MongoDB : `mongo` Si vous voulez spécifier une base de données particulière : `mongo <nom_de_la_base>`
   — Mongodump est un outil utilisé pour sauvegarder une base de données MongoDB. Il crée un *snapshot* des collections sous forme de fichiers BSON (Binary JSON).
      — Cela est utile pour les sauvegardes ou pour déplacer des données d'un environnement à un autre. Commande de base pour sauvegarder une base de données : `mongodump -db <nom_de_la_base>` Vous pouvez aussi spécifier un répertoire de sortie : `mongodump -db <nom_de_la_base> -out /chemin/vers/sauvegarde`
   — Mongoimport permet de restaurer ou importer des données dans MongoDB depuis des fichiers au format JSON, CSV ou TSV. Commande de base pour importer un fichier JSON dans une collection MongoDB :
   `mongoimport --db <nom\_de\_la\_base> --collection <nom\_collection> --file <chemin\_vers\_le\_fichier.json>`
   . Pour des fichiers CSV, vous pouvez ajouter l'option `-type csv` :

```
mongoimport --db <nom\_de\_la\_base> --collection
<nom\_de\_la\_collection> --type csv --file
<chemin\_vers\_le\_fichier.csv> --headerline
```

   Ces trois outils sont essentiels pour la gestion, la sauvegarde et la restauration des données dans MongoDB.
2. MongoDB Compass offre une interface utilisateur graphique pour interagir avec les données.

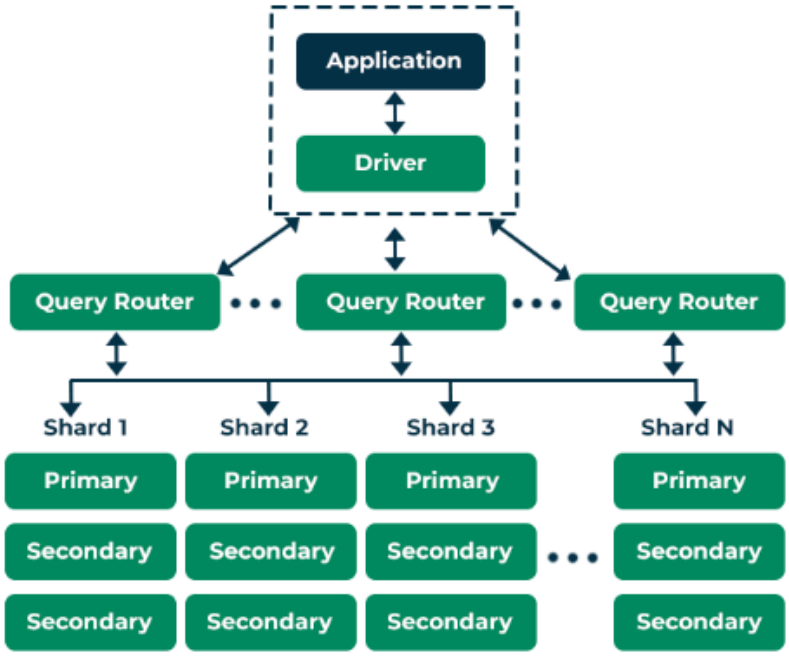
5.3 Architecture du moteur NoSQL (schéma)

5.3.1 Architecture

MongoDB suit une architecture maître-esclave (ou primaire-secondaire) dans un ensemble de *replicas* ou dans une partition. Les données sont distribuées en partitions (sharding) pour faciliter la scalabilité horizontale. Les partitions sont réparties entre plusieurs serveurs.

5.3.2 Schéma

— Sharding pour le partitionnement.
— Réplication pour la haute disponibilité.
— Chaque ensemble de replicas a un replica *primaire* ou maître qui gère les écritures.



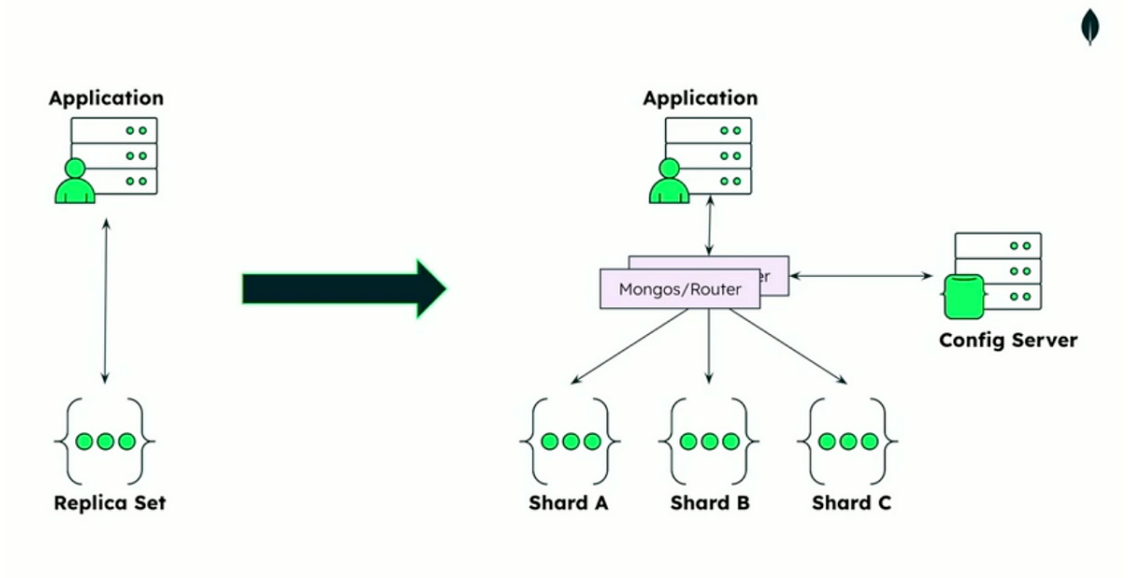
5.4 Méthode de partitionnement

5.4.1 Sharding

MongoDB utilise un mécanisme de *sharding* pour partitionner horizontalement les données. Les documents sont distribués entre différents partitions en fonction de la clé de sharding (clé de partitionnement).

5.4.2 Schéma

— Clé de partitionnement choisie pour diviser les documents.
— Chaque partition stocke une portion des données.



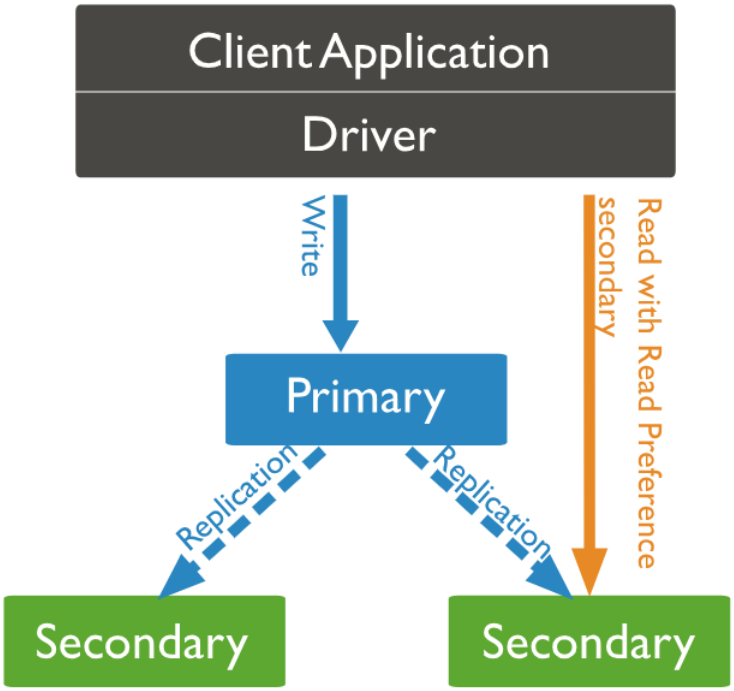
5.5 Méthode de réplication

5.5.1 Réplication

MongoDB utilise des *replicas* pour la haute disponibilité. Un ensemble de replicas contient un replica primaire et plusieurs secondaires. Le primaire gère les écritures, et les réplicas (secondaires) répliquent les données.

5.5.2 Schéma

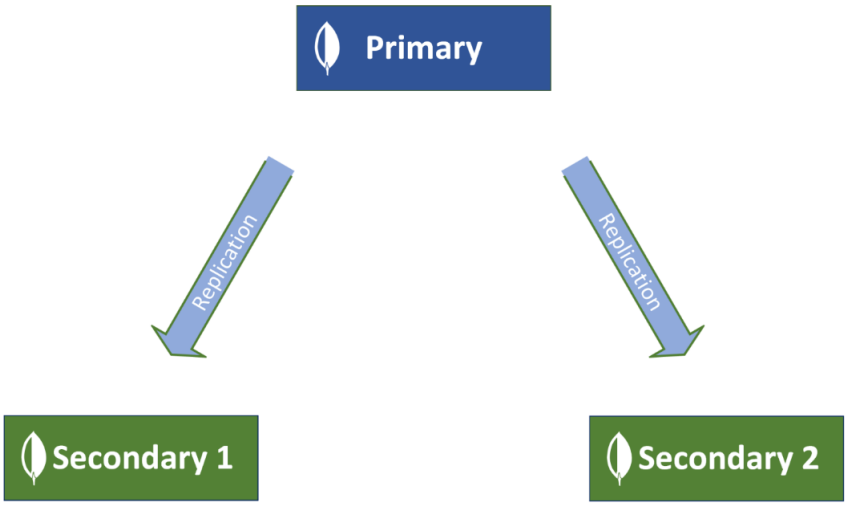
— Primaire pour les écritures.
— Secondaires pour la redondance et les lectures.



5.6 Montée en charge

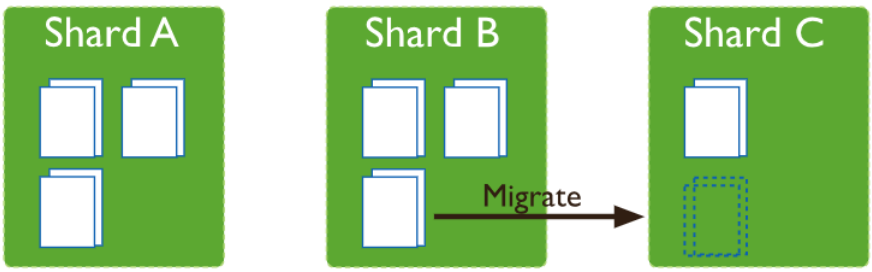
5.6.1 Montée en charge verticale

MongoDB permet une montée en charge verticale grâce au mécanisme de réplication sur plusieurs serveurs dans une même partition. En ajoutant de nouveaux serveurs ou replicas, les données peuvent être répliquées de manière synchrone par défaut pour mieux répartir la charge. L'ajout de nouveaux esclaves ou replicas dans une partition pour étendre la base de données verticalement permet de faciliter les opérations de lecture garantir et améliorer la disponibilité.



5.6.2 Montée en charge horizontale

MongoDB permet une montée en charge horizontale grâce au partitionnement sur plusieurs serveurs. En ajoutant de nouveaux serveurs, les partitions peuvent être redistribuées pour mieux répartir la charge. L'ajout de partitions pour étendre la base de données horizontalement permet de faciliter les opérations d'écriture.

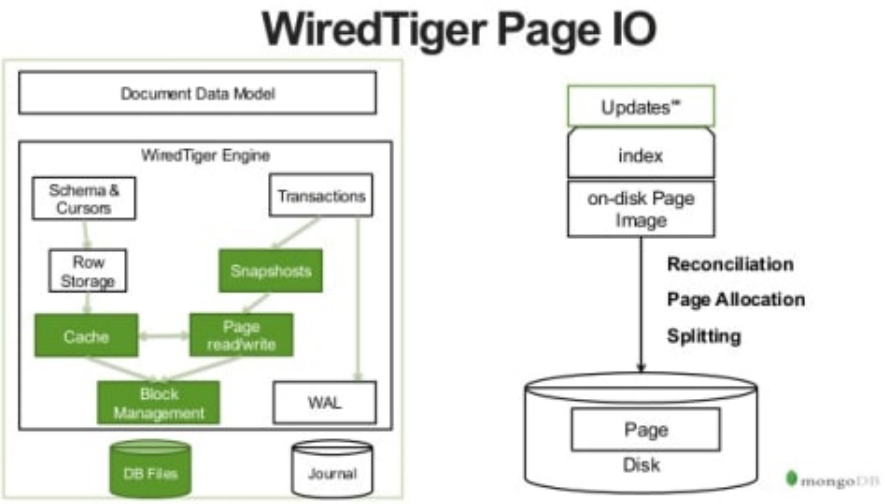


5.7 Gestion du cache mémoire

MongoDB utilise le cache de *mappage mémoire*. Le moteur *WiredTiger* utilise la mémoire disponible pour les lectures, mais laisse au système d'exploitation la gestion précise de l'utilisation de la mémoire.

5.7.1 Schéma

WiredTiger, mémoire tampon pour les écritures et mappage des pages de disque dans la mémoire.



6 Génération automatique des données

Dans cette section nous allons générer des données pour nos deux collections *Medecin* et *Patient*. Pour cela nous utilisons le package `gem` du langage `Ruby`. Ci-dessous la procédure d'installation de `Ruby` et `Gem` :

1. Installation de Ruby
   - macOS : Ruby est généralement déjà pré-installé.
   - Windows : Ruby est téléchargeable a l'adresse `https://rubyinstaller.org/`.
   - Linux (Ubuntu) : `sudo apt install ruby`.
2. Ouvrir un Terminal/Command Prompt :
   - macOS/Linux : Open a terminal.
   - Windows : Open Command Prompt or PowerShell.
3. `gem install faker`
4. `gem install mongo`
5. `gem list` (vérification de l'installation)

6.1 Script de génération de 1000 médecins

```
require 'faker'
require 'json'
require 'date'
require 'mongo'

def generate_random_document
  {
    "_id" => BSON::ObjectId.new,
```

```
9           "nom" => Faker::Name.last_name,
10          "sexe" => ["Male", "Female", "Other"].sample,
11          "date_naissance" => Faker::Date.birthday(min_age: 25,
                max_age: 65),
12          "specialite" => ["Cardiologist", "Dermatologist", "
                Pediatrician", "Orthopedic Surgeon", "Neurologist", "
                Gynecologist"].sample,
13          "email" => Faker::Internet.email,
14          "cv" => Faker::Lorem.sentence(word_count: 10),
15          "list_telephones" => [Faker::PhoneNumber.phone_number,
                Faker::PhoneNumber.phone_number],
16          "list_prenoms" => [Faker::Name.first_name, Faker::Name.
                first_name],
17          "adresse" => {
18            "numero" => Faker::Address.building_number,
19            "rue" => Faker::Address.street_name,
20            "code_postal" => Faker::Address.zip_code,
21            "ville" => Faker::Address.city
22          },
23          "list_rendez_vous" => [
24            {
25              "patient" => BSON::ObjectId.new,
26              "date_rendez_vous" => Faker::Date.forward(days: 23),
27              "motif" => Faker::Lorem.sentence(word_count: 3)
28            }
29          ],
30          "list_consultations" => [
31            {
32              "patient" => BSON::ObjectId.new,
33              "raison" => Faker::Lorem.sentence(word_count: 3),
34              "diagnostic" => Faker::Lorem.sentence(word_count: 5),
35              "date_consultation" => Faker::Date.backward(days: 30),
36              "list_examens" => [
37                {
38                  "details_examen" => Faker::Lorem.sentence(
                        word_count: 5),
39                  "date_examen" => Faker::Date.backward(days: 15)
40                }
41              ],
42              "list_prescriptions" => [
43                {
44                  "details_prescription" => Faker::Lorem.sentence(
                        word_count: 5),
45                  "date_prescription" => Faker::Date.backward(days:
                        10)
46                }
47              ]
48            }
49          ]
50        }
```

```
51      end
52
53      documents = Array.new(1000) { generate_random_document }
54
55      File.open("medecins_data.json", "w") do |f|
56        f.write(JSON.pretty_generate(documents))
57      end
```

6.2 Script de génération de 1000 patients

```
1       require 'mongo'
2       require 'faker'
3       require 'json'
4       require 'date'
5
6       def generate_random_patient
7         {
8           "_id" => BSON::ObjectId.new,
9           "numero_securite_sociale" => Faker::IdNumber.valid,
10          "nom" => Faker::Name.last_name,
11          "sexe" => ["M", "F", "O"].sample,
12          "date_naissance" => Faker::Date.birthday(min_age: 0,
              max_age: 100),
13          "email" => Faker::Internet.email,
14          "poids" => Faker::Number.between(from: 45.0, to: 120.0).
              round(2),
15          "hauteur" => Faker::Number.between(from: 1.5, to: 2.0).
              round(2),
16          "list_telephones" => [Faker::PhoneNumber.phone_number,
              Faker::PhoneNumber.phone_number],
17          "list_prenoms" => [Faker::Name.first_name, Faker::Name.
              first_name],
18          "adresse" => {
19            "numero" => Faker::Address.building_number,
20            "rue" => Faker::Address.street_name,
21            "code_postal" => Faker::Address.zip_code,
22            "ville" => Faker::Address.city
23          },
24          "list_rendez_vous" => [
25            {
26              "medecin" => BSON::ObjectId.new,
27              "date_rendez_vous" => Faker::Date.forward(days: 23),
28              "motif" => Faker::Lorem.sentence(word_count: 3)
29            }
30          ],
31          "list_consultations" => generate_consultations(Faker),
32          "antecedents_medicaux" => generate_antecedents_medicaux(
              Faker),
33          "allergies" => [Faker::Lorem.word, Faker::Lorem.word]
```

```
34        }
35      end
36
37      def generate_consultations(faker)
38        consultations_list = []
39
40        2.times do
41          examens_list = [
42            {
43              "details_examen" => Faker::Lorem.sentence(word_count:
                    5),
44              "date_examen" => Faker::Date.backward(days: 15)
45            }
46          ]
47
48          prescriptions_list = [
49            {
50              "details_prescription" => Faker::Lorem.sentence(
                    word_count: 5),
51              "date_prescription" => Faker::Date.backward(days: 10)
52            }
53          ]
54
55          facture = {
56            "Montant_Total" => faker::Number.between(from: 50.0, to:
                  500.0).round(2),
57            "date_facture" => Faker::Date.backward(days: 7)
58          }
59
60          consultations_list << {
61            "medecin" => BSON::ObjectId.new,
62            "raison" => Faker::Lorem.sentence(word_count: 3),
63            "diagnostic" => Faker::Lorem.sentence(word_count: 5),
64            "date_consultation" => Faker::Date.backward(days: 30),
65            "list_examens" => examens_list,
66            "list_prescriptions" => prescriptions_list,
67            "facture" => facture
68          }
69        end
70
71        consultations_list
72      end
73
74      def generate_antecedents_medicaux(faker)
75        [
76          {
77            "type" => "Chronic",
78            "description" => Faker::Lorem.sentence(word_count: 5),
79            "date" => Faker::Date.backward(days: 500)
80          },
```

```ruby
        {
          "type" => "Acute",
          "description" => Faker::Lorem.sentence(word_count: 5),
          "date" => Faker::Date.backward(days: 300)
        }
      ]
    end

    patients = Array.new(1000) { generate_random_patient }

    File.open("patients_data.json", "w") do |f|
      f.write(JSON.pretty_generate(patients))
    end
```

6.3 Génération des données dans des fichiers JSON

6.3.1 Données des médecins

1. Mettre le script pour la génération des données pour les médecins dans un fichier, par exemple `medecins_script.rb`
2. `ruby medecins_script.rb`
3. le fichier `medecins_data.json` a été généré et contient les données des médecins au format *JSON*.

6.3.2 Données des patients

1. Mettre le script pour la génération des données pour les patients dans un fichier, par exemple `patients_script.rb`
2. `ruby patients_script.rb`
3. le fichier `patients_data.json` a été généré et contient les données des patients au format *JSON*.