Groupe 2
Sprint 1

Introduction

Dans ce premier audit, nous allons présenter les différentes tâches effectuées au cours du sprint1.
Il correspond au scénario S1 et à l'incrément BD I1.

Dans un premier temps, nous verrons l'organisation du planning de travail. Puis nous ferons des zooms relatifs aux différentes tâches réalisées :

- Base de donnée
- Modèle
- IHM

Enfin, nous finirons par une démonstration de l'interface web fonctionnelle obtenue à la fin du sprint1.

Membres du groupe

- Celia Kezmane
- Nicolas Martinez
- Robin Miquel
- Matthieu Lehugeur

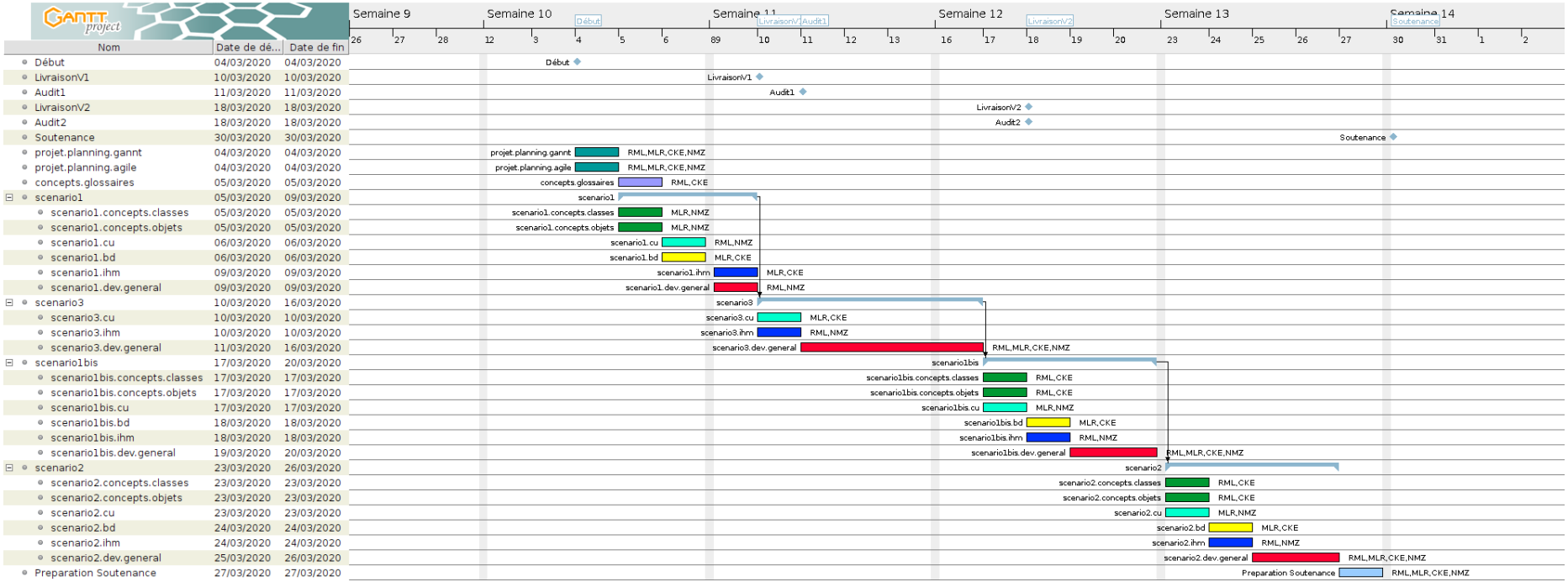
Description du problème

Un directeur de théâtre désire informatiser son système de réservation de places.

Dans un premier temps, il souhaite qu'un client puisse accéder au site et faire apparaître les programmations disponibles en fonction des différents filtres (Horaires, Type de spectacle, Public cible)

Planning prévisionnel

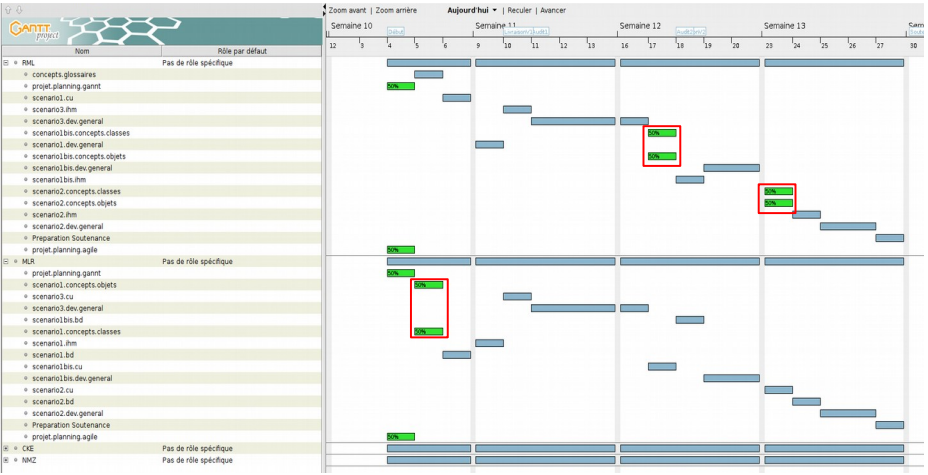
planning.gantt

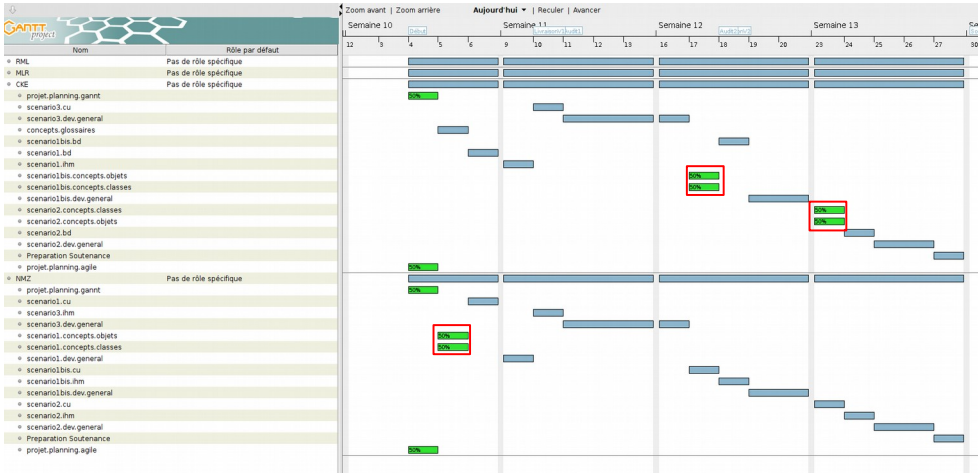


4 scénarios: 1, 3, 1bis et 2

Pour le scénario 1: 72h de travail prévues au total

Planning prévisionnel

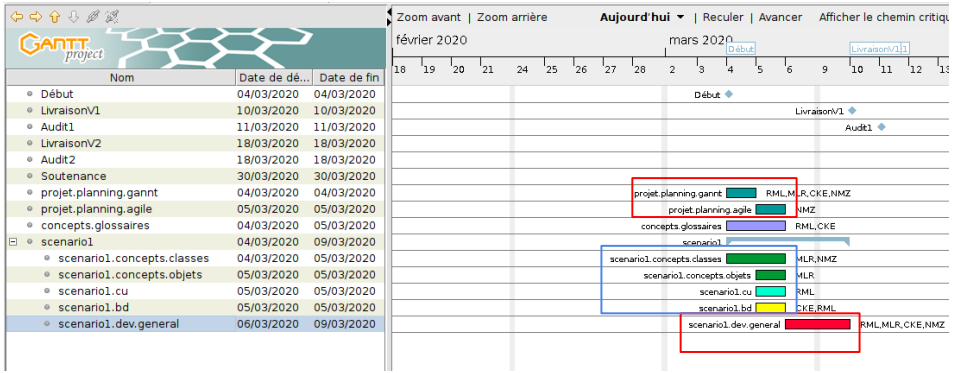




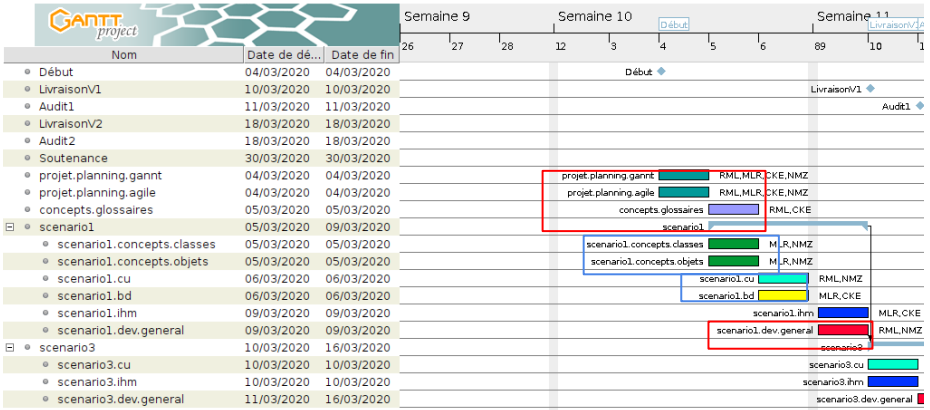
Pas de rôles prédéfinis afin d'instaurer une rotation pour les différentes tâches

Planning prévisionnel vs planning effectif

planning.effectif
planning.gantt
suivis



effectif



prévisionnel

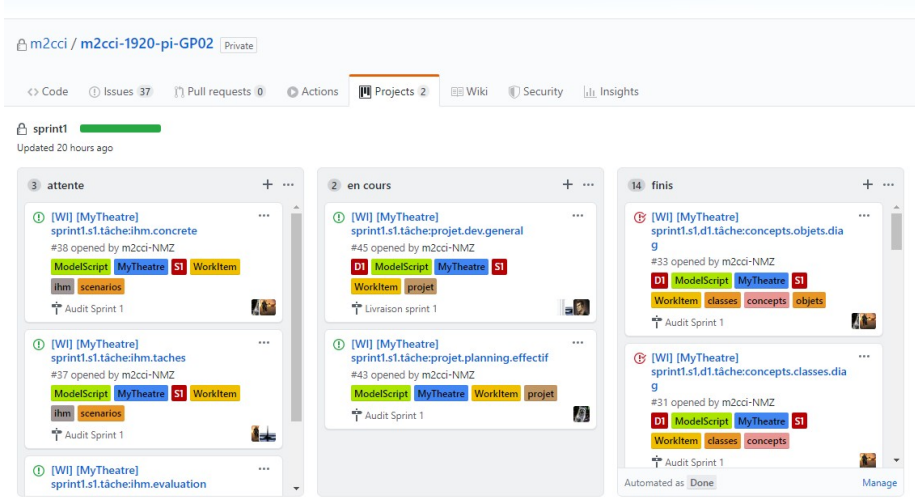
Sous-estimation des tâches glossaire, planning, dev, bd

Sur-estimation de la tâche cu, classes

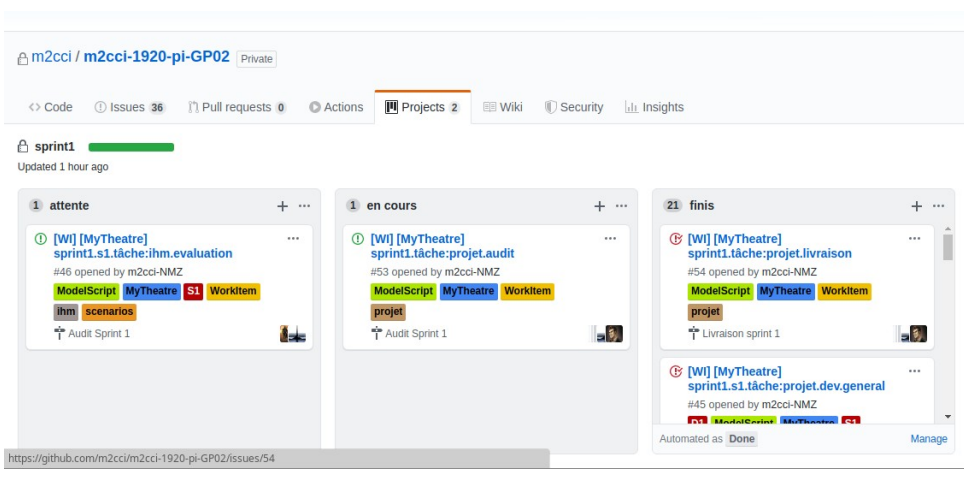
Au total, 60h de travail effectif sur 72h prévues: 8h d'empêchements pour cause de connexion réseau et 4h de cours sur jsp

Planning Github

planning.effectif
planning.agile



Mi parcours (06/03)



Fin de sprint (10/03)

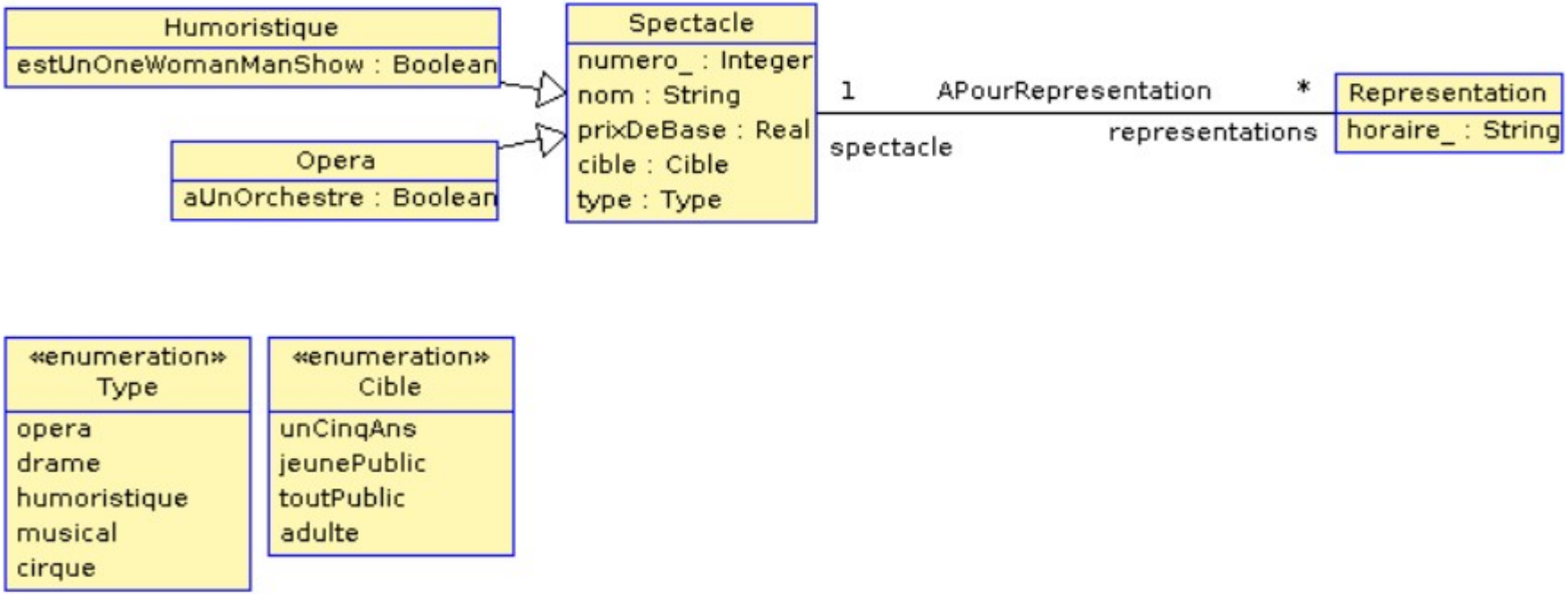
Résultat de la rétrospective

- Ce qui n’a pas marché
  - Utilisation de ganttProject très lourde
- Ce qui a marché
  - Planification
  - Répartition diverse des tâches afin que tout le monde soit confronté aux différentes tâches
- Ce qu’il serait bien d’essayer
  - Création de branches git

concept.classes
S1, S3, I1
classes.cl1

Zoom : Base de données

Diagramme de classe



Héritage par référence

bd.relations
S1, S3, I1
relations.res

```
relation LesSpectacles
        transformation
                from R_Class(Spectacle)

        columns
                numeroSpe_ : Integer
                nomSpe : String
                prixDeBaseSpe : Real
                cibleSpe : Cible
                typeSpe : Type

        constraints
                key numeroSpe_
                numeroSpe_ > 0
```

```
relation LesOperas
        transformation
                from R_Reference(Opera)

        columns
                numeroSpe_ : Integer
                aUnOrchestreOpe : Boolean

        constraints
                key numeroSpe_
```

```
relation LesRepresentations
        transformation
                from R_Class (Representation)
                from R_OneToMany (Spectacle)

        columns
                horaireRep_: Date
                numeroSpe : Integer
                //placesDispoRep_d : Integer
                //tauxReducRep : Real

        constraints
                key dateRep_
```

```
constraints
        LesOperas[numeroSpe_] C= LesSpectacles[numeroSpe_]
        LesRepresentations[numeroSpe] C= LesSpectacles[numeroSpe_]
```

Création des tables

bd.sql.schema
S1, S3, I1
schema.sql

```sql
CREATE TABLE LesSpectacles(
        numeroSpe INTEGER,
        nomSpe VARCHAR(100) NOT NULL,
        prixDeBaseSpe REAL,
        cibleSpe VARCHAR(11),
        typeSpe VARCHAR(12),

        CONSTRAINT PK_Spe
                PRIMARY KEY (numeroSpe),
        CONSTRAINT CK_Spe_cibleSpe
                CHECK (cibleSpe in ("unCinqAns", "jeunePublic", "toutPublic","adulte")),
        CONSTRAINT CK_Spe_typeSpe
                CHECK (typeSpe in ("opera", "drame", "humoristique", "musical", "cirque")),
        CONSTRAINT CK_Spe_numeroSpe
                CHECK ( 0 < numeroSpe),
        CONSTRAINT CK_Spe_prixDeBaseSpe
                CHECK ( 0 < prixDeBaseSpe)
);
```

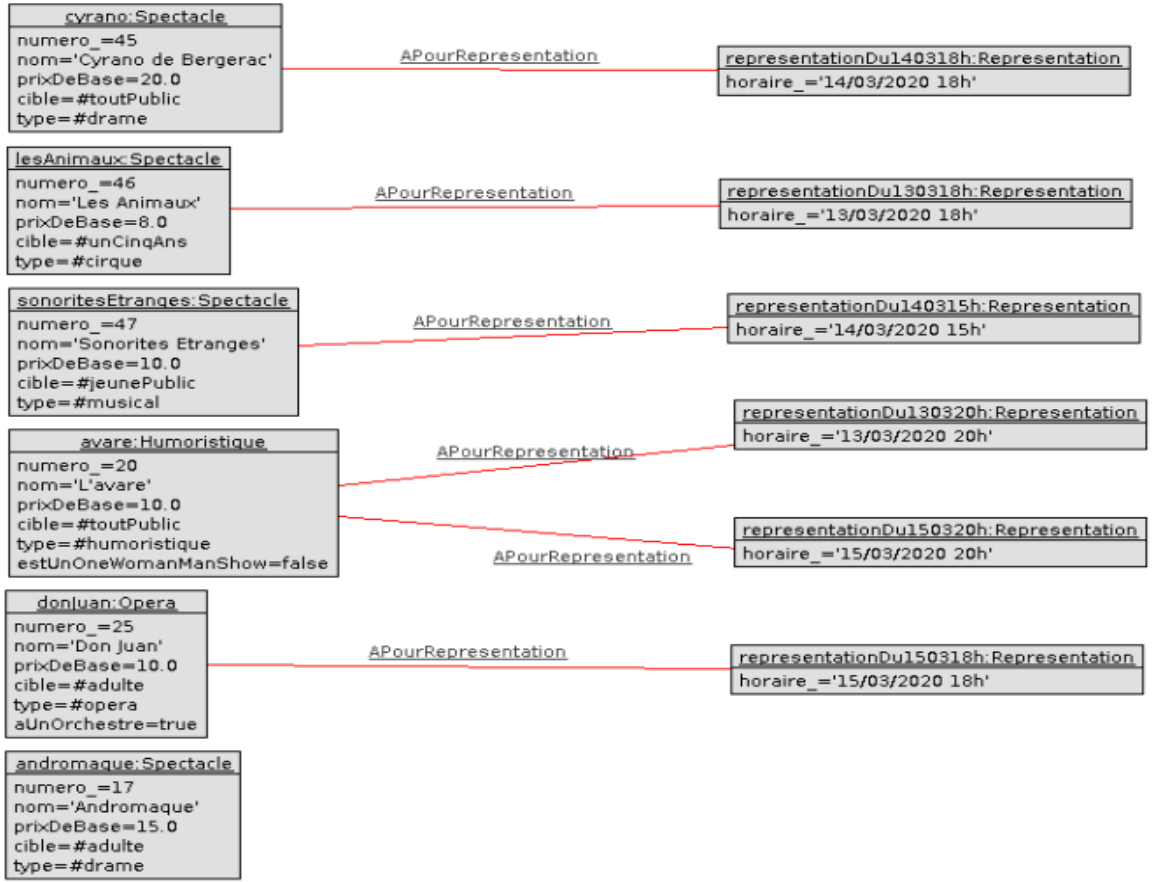
Traduction des types spéciaux

```sql
CREATE TABLE LesRepresentations(
        horaireRep VARCHAR(14),
        numeroSpe INTEGER,
        CONSTRAINT PK_Rep
                PRIMARY KEY (horaireRep),
        CONSTRAINT FK_Rep_numeroSpe
                FOREIGN KEY (numeroSpe) REFERENCES LesSpectacles(numeroSpe)
);
```

Foreign key pour le One to Many

Diagramme d’objets

concept.objets
S1, S3, I1
o1.ob1



Tester la BD : Jeu de données négatives et vérifications supplémentaires

concept.objets
S1, S3, I1
jddn1.sql

- LesSpectacles
  - 2 `Spectacle`s avec le même numeroSpe
  - numeroSpe ou prixDeBase negatifs
  - Valeur cibleSpe ou typeSpe incorrecte

- LesHumoristiques et LesOperas
  - numeroSpe pas dans LesSpectacles
  - Booléens incorrectes (ni 0 ni 1)

- LesRepresentations
  - 2 `Representation`s avec la même horaireRep
  - numeroSpe pas dans LesSpectacles

- Tests supplémentaires
  Création d'un fichier verif.sql qui s'execute à la fin de la création de la BD et qui vérifie le respect de contraintes supplémentaires : Ici, on vérifie seulement que les `Spectacle`s de type opéra ou humoristique sont bien aussi dans les tables LesOperas et LesHumoristiques

Requête à la BD

dev.general
S1, S3, I1
ProgDAO.java

```
String queryRep = "SELECT S.numeroSpe, nomSpe, prixDeBaseSpe, cibleSpe, typeSpe, estUnOneWomanManShowHum, aUnOrchestreOpe, horaireRep \n"
        + "FROM LesSpectacles S LEFT OUTER JOIN LesOperas O ON S.numeroSpe = O.numeroSpe \n"
        + "LEFT OUTER JOIN LesHumoristiques H ON S.numeroSpe = H.numeroSpe \n"
        + "JOIN LesRepresentations R ON R.numeroSpe = S.numeroSpe \n"
        + "WHERE horaireRep>=? AND horaireRep<=? AND cibleSpe=? AND typeSpe=?"
        + "ORDER BY horaireRep ;";
```

```
// Création des objets
Date horaire = horaireFormatter.parse(horaireRep);
Spectacle s;
switch(type){
    case "opera":
        int aUnOrchestreOpe = rs.getInt("aUnOrchestreOpe");
        boolean aUnOrchestre = (aUnOrchestreOpe == 1);
        s = new Opera(numero, nom, prixDeBase, cible, type, aUnOrchestre);
        break;
    case "humoristique":
        int estUnOneWomanManShowHum = rs.getInt("estUnOneWomanManShowHum");
        boolean estUnOneWomanManShow = (estUnOneWomanManShowHum == 1);
        s = new Humoristique(numero, nom, prixDeBase, cible, type, estUnOneWomanManShow);
        break;
    default:
        s = new Spectacle(numero, nom, prixDeBase, cible, type);
}
Representation rep = new Representation(horaire, s);
representations.add(rep) ;
```

Zoom: Partie modèles de l’architecture MVC

4 classes:

- Representation
  - Modélise une `Representation`, contient un pointeur vers la classe Spectacle
- Spectacle
  - Modélise un `Spectacle`
- Humoristique
  - Modélise un `Humoristique`, hérite de la classe Spectacle
- Opera
  - Modélise un `Opera`, hérite de la classe Spectacle

Exemple de code pour la classe Représentation

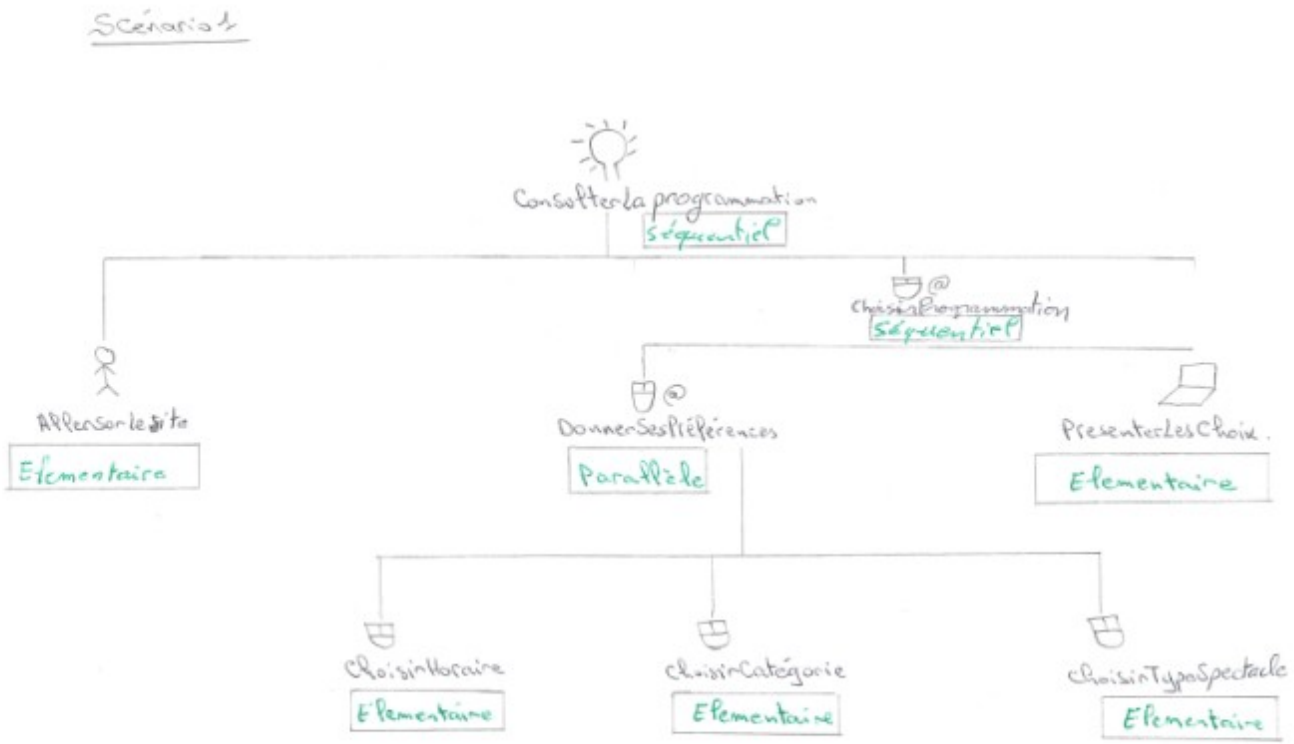
```java
public class Representation {
    private final Date horaire;
    private final Spectacle spe;

    public Representation(Date date, Spectacle spe) {
        this.horaire = date;
        this.spe = spe;
    }

    /** Renvoie la date du spectacle ...5 lines */
    public Date getHoraire() {
        return this.horaire;
    }

    /** Renvoie l'objet spectacle associé a sa representation ...5 lines */
    public Spectacle getSpectacle() {
        return this.spe;
    }

    @Override
    public String toString() {
        SimpleDateFormat formatDate = new SimpleDateFormat("dd/MM/yyyy HHh");

        return "Representation : " + "Horraire =" + formatDate.format(horaire) + "\n" + spe  + "\n";
    }
}
```

Tests unitaires sur les modèles

```java
public class RepresentationTest {

    private static Representation representation = null;

    public RepresentationTest() {
    }

    @Before
    public void setUp() {
        Date date = new Date(10,10,10,10,0);
        Spectacle spectacle= new Spectacle(10, "nom", 10.5, "comedie", "toutPublic");
        representation = new Representation(date, spectacle);
    }

    /**
     * Test of getDate method, of class Representation.
     */
    @Test
    public void testGetDate() {
        System.out.println("getDate");
        Date date = new Date(10,10,10,10,0);
        assertEquals(date, representation.getDate());
    }

    /**
     * Test of getSpectacle method, of class Representation.
     */
    @Test
    public void testGetSpectacle() {
        System.out.println("getSpectacle");
        Spectacle spectacle= new Spectacle(10, "nom", 10.5, "comedie", "toutPublic");
        assertEquals(spectacle, representation.getSpectacle());
    }

    /**
     * Test of toString method, of class Representation.
     */
    @Test
    public void testToString() {
        System.out.println("toString");
        Date date = new Date(10,10,10,10,0);
        SimpleDateFormat formatDate = new SimpleDateFormat("dd/MM/yyyy HHh");
        assertEquals("Representation : Horraire =" + formatDate.format(date) + "\nSpectacle : nom (comedie, toutPublic) à 10.5€\n", represen
    }

}
```

Zoom : IHM

Modèle de tâches



Maquette

ihm.concrete

Vue des `Representation`s

JSP et Respect des normes IHM

```
<tbody>
    <%
        SimpleDateFormat horaireFormatter = new SimpleDateFormat("dd/MM à HH");
        List<Representation> prog = (List<Representation>) request.getAttribute("progList");
        for (Representation r : prog) {
            Date date = r.getHoraire();
            String nom = r.getSpectacle().getNom();
            Double prixDeBase = r.getSpectacle().getPrixDeBase();
            String cible = r.getSpectacle().getCible();
            String type = r.getSpectacle().getType();
    %>

    <tr>
        <td><%=horaireFormatter.format(date)%>h</td>
        <td><%=nom%></td>
        <td><%=prixDeBase%> €</td>
        <td><%=cible%></td>
        <td><%=type%></td>
    </tr>
```

```
<h5>Dates</h5>
<%
    SimpleDateFormat navigateurJourFormatter = new SimpleDateFormat("yyyy-MM-dd");
    Date dateDebut = (Date) request.getAttribute("dateDebut");
    Date dateFin = (Date) request.getAttribute("dateFin");

    // Affichage des dates sélectionnés dans la nouvelle page
    String dateDebutForm;
    if (dateDebut == null){
        dateDebutForm = "2020-03-01";
    } else {
        dateDebutForm = navigateurJourFormatter.format(dateDebut);
    }
    String dateFinForm;
    if (dateFin == null) {
        dateFinForm = "2020-03-31";
    } else {
        dateFinForm = navigateurJourFormatter.format(dateFin);
    }
%>
<input type="date" name="dateDebut" value=<%=dateDebutForm%>> au
<input type="date" name="dateFin" value=<%=dateFinForm%>>
```

Perspectives au sprint2

Scénario S3 : Le directeur du théâtre souhaite pouvoir ajouter des `Representation`s et des `Spectacle`s dans le système

Scénario S1bis : Fonctionnalité de réservation de places