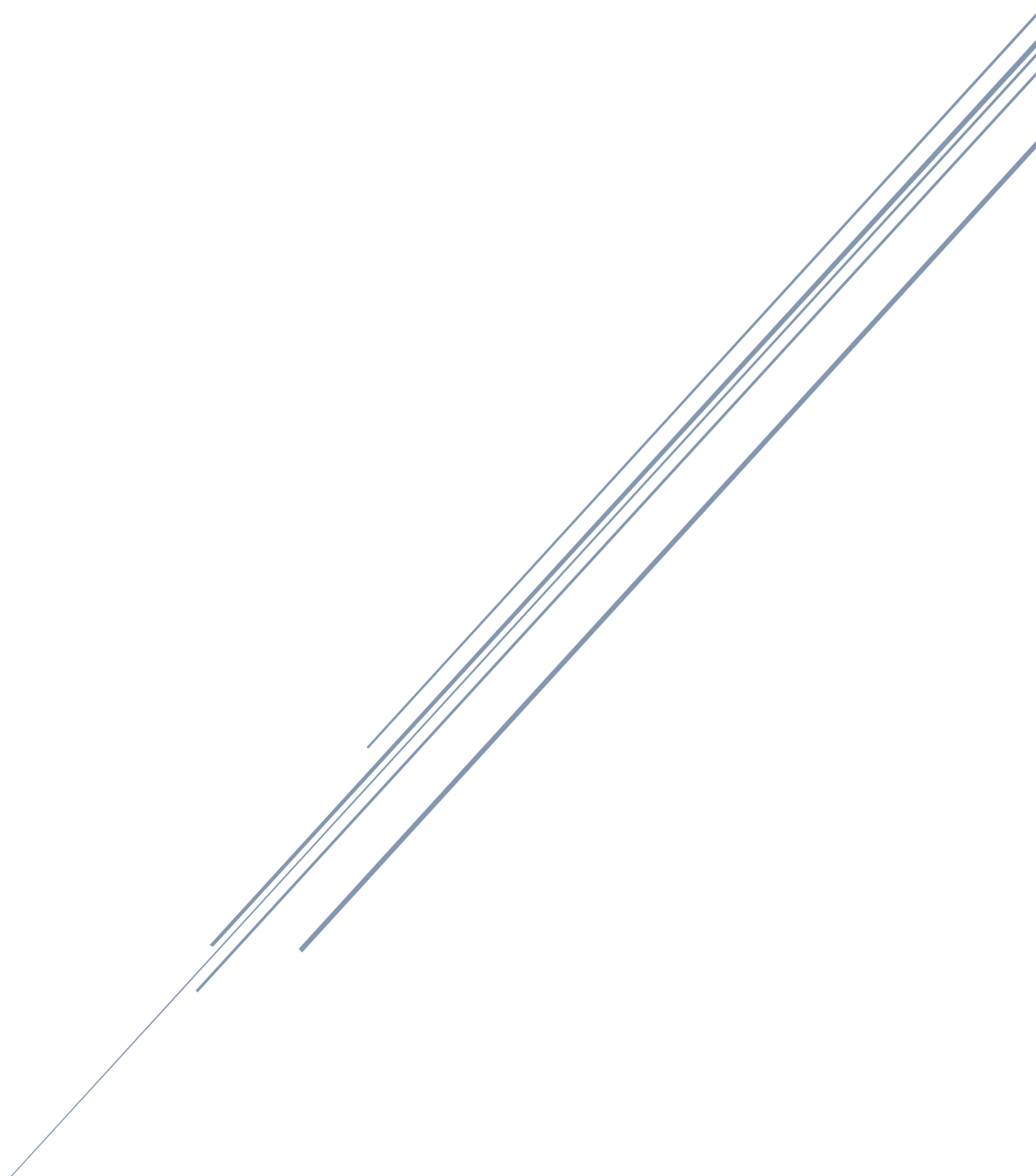


RAPPORT DE STAGE

Concepteur développeur d'application



Pop School
Stage du 14/12/2022 au 07/03/2023

Remerciements

Je remercie tout d'abord Madame REY Catherine qui m'a donné l'opportunité de prendre part à son projet ainsi que la confiance qu'elle m'a accordée.

Je remercie également la Pop School pour cette formation qualifiante qui m'a permis d'approfondir mes connaissances et qui je l'espère sera le tremplin, avec l'obtention du diplôme, d'une vie professionnelle riche et épanouissante.

Et pour finir Mégane avec qui l'entraide et la bienveillance étaient des valeurs communes

Sommaire

1. Introduction	P.1
2. Liste des compétences abordées	P.2
3. Résumé	P.3
4. L'entreprise	P.4
5. Présentation des outils utilisés	P.5
a. Logiciels	P.5
b. Langages	P.6
c. Dépendances	P.7
6. Présentation du projet	P.8
a. Cahier des charges	P.8
b. Maquette	P.9
c. UML	P.11
7. Création du projet	P.13
a. Types énumérés	P.15
b. Entités	P.16
c. Repositories	P.18
d. Services	P.19
e. Connexion	P.24
8. Tests	P.25
9. Interface graphique	P.29
a. Création des modèles de table	P.29
b. Création de l'interface Swing	P.33
10. Déploiement	P.58
11. Sécurité	P.62
12. Conclusion	P.64

1. Introduction

Je m'appelle Béline Baratte j'ai 34 ans, j'ai effectué divers métiers dans différents domaines comme la police, convoyeur de fond, médiatrice dans les transports en commun, etc....

Je n'ai jamais réellement pu et su m'épanouir dans chacun de ces métiers, et à chaque transition entre ces emplois je regardais les formations pour exercer des métiers dans l'informatique.

Ayant des factures à payer je ne pouvais me permettre, à ces moments-là, d'entamer une formation.

Alors que je travaillais de nuit dans un foyer de vie pour personnes en situation de handicap, j'ai commencé à apprendre toute seule les métiers dans l'informatique comme la modélisation 3D avec Blender, la conception de jeu vidéo avec Unity, la domotique avec des Arduino et des Raspberry jusqu'au langage de programmation avec C++, C#, Java.

Un jour, je suis allée faire un Escape Game et j'ai demandé à la direction s'il recherchait un animateur, à la place on m'a proposé un poste de développeur domoticien.

J'ai accepté et c'est comme ça que j'ai pu créer une salle d'escape entièrement domotisée ainsi que 3 applications dont une pour la mairie d'Hénin-Beaumont lors d'un salon de recrutement, une pour IKEA lors d'une soirée organisée dans leurs locaux et enfin une pour une salle d'Escape Game sur le thème de Very Bad Trip qui est aujourd'hui fonctionnelle et utilisée tous les jours par les joueurs.

Malheureusement avec le covid l'activité s'est arrêtée.

Cette expérience a été très enrichissante, j'ai pu mettre à profit mes connaissances acquises par moi-même et eu l'occasion de travailler en totale autonomie, chercher des solutions aux problèmes, etc....

C'est pour ça que j'ai souhaité faire cette formation afin de valider mes acquis mais aussi pouvoir constater si ma base était solide ainsi que de toujours me perfectionner et en apprendre davantage.

2. Liste des compétences abordées

N° Fiche AT	Activités types	N° Fiche CP	Compétences professionnelles
1	Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité	1	Maquetter une application
		2	Développer une interface utilisateur de type desktop
		3	Développer des composants d'accès aux données
		4	Développer la partie front-end d'une interface utilisateur web
		5	Développer la partie back-end d'une interface utilisateur web
2	Concevoir et développer la persistance des données en intégrant les recommandations de sécurité	6	Concevoir une base de données
		7	Mettre en place une base de données
		8	Développer des composants dans le langage d'une base de données
3	Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité	9	Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
		10	Concevoir une application
		11	Développer des composants métier
		12	Construire une application organisée en couches
		13	Développer une application mobile
		14	Préparer et exécuter les plans de tests d'une application
		15	Préparer et exécuter le déploiement d'une application

3. Résumé

Afin de valider les compétences pour la validation du titre de « Concepteur Développeur d'Application » j'ai effectué un projet lors de notre stage pour une micro-entreprise de coach de vie. Notre équipe est composée de deux apprenantes de la formation : Mégane Dominik et moi-même.

Dans un premier temps, notre cliente désire un site vitrine afin d'élargir sa visibilité sur le web ayant pour objectif de toucher un public plus large. De plus, il doit contenir les fonctionnalités suivantes :

- Un formulaire de contact
- Une prise de rendez-vous en ligne

Dans un second temps, notre cliente désire une application desktop qui aura les fonctionnalités suivantes :

- Ajout, suppression, modification d'un contact et d'un rendez-vous
- Affichage des données des contacts et des rendez-vous.
- Blocage de créneaux horaire réservé.

Pour développer ce projet nous avons utilisé le Framework « Spring » qui est basé sur le langage de programmation « Java » avec les dépendances :

- « thymeleaf »
- « Spring web »
- « Spring Data JPA »
- « Lombok »
- « Validation »

Côté front nous avons utilisé :

- « HTML »
- « CSS »
- « Bootstrap ».
- « Swing »

Pour la base de données nous avons utilisé une base de données relationnelle :

- « MariaDB ».

Il y a plusieurs axes d'amélioration à ce projet :

- Rendre davantage accessible l'ajout de post d'actualité par la cliente via l'application desktop, et ainsi lui faciliter les échanges avec sa clientèle.
- Rendre possible la réservation d'un même rendez-vous, à plusieurs participants, pour les réunions de groupe.

4. L'entreprise



J'ai effectué mon stage dans une micro-entreprise de coach de vie dont l'entrepreneuse individuelle s'appelle Catherine Rey.

Elle propose des services de coaching aux particuliers. Il s'agit d'un accompagnement personnalisé qui a pour but de développer le potentiel de chacun. La mise en place de stratégies de réussite pour atteindre des objectifs individuels et ciblés. Que cela soit dans le domaine personnel ou professionnel. Tels que :

- Développement de la confiance en soi
- Développement de l'estime de soi
- Accompagnement dans une réorientation professionnelle
- Aide à la préparation de passages d'examens

Elle propose également ses services aux entreprises qui cherchent à développer le potentiel de leurs employés. Tels que :

- Accompagnement dans un changement de poste
- Lors d'une restructuration permettre aux employés de retrouver un emploi rapidement
- Evaluation du potentiel de l'employé
- Aide au recrutement

Elle se met au service des personnes pour les faire évoluer, trouver leurs voies et s'épanouir.

En résumé, le coaching est l'art d'apprendre à « Apprendre à réussir ».

5. Présentation des outils

a. Logiciel :



Figma est une application web d'édition graphique. Elle permet le partage avec tous les membres de l'équipe. La réalisation est simple et rapide.



Strat UML est un outil de modélisation UML (Unified Modeling Language) qui permet de concevoir et de documenter des diagrammes de classes, de séquences, etc....



IntelliJ est un IDE (environnement de développement intégré). Il permet de créer, de déboguer et de déployer du code de manière efficace.



MariaDB est un système de gestion de bases de données relationnelles.



DBeaver est un outil de gestion de bases de données universelles qui permet aux utilisateurs de se connecter à plusieurs bases en même temps.



Looping est un logiciel de modélisation conceptuelle de données qui permet d'organiser et de structurer les données en diagrammes et représentations graphiques ordonnées.



Discord est un outil de communication. Il permet d'échanger et de partager très facilement.



Visual Studio Code est un éditeur de code source. Il permet de programmer en différent langage tel que Java, JavaScript, Html, CSS (etc.)

b. Langage :



Java est un langage de programmation orienté objet multiplateforme.



HTML est un langage de balisage d'hypertexte utilisé pour créer des pages web



CSS est un langage de feuilles de style en cascade utilisé pour décrire l'apparence d'une page web écrite en HTML.



JavaScript


JavaScript est un langage de programmation de script côté client utilisé pour ajouter des fonctionnalités interactives à des pages web



SQL est un langage de programmation utilisé pour gérer les données dans les systèmes de gestion de base de données relationnelles.

c. Dépendances

Les dépendances sont gérées grâce à Maven, dans le fichier « POM », qui automatise la construction, la documentation et la gestion des dépendances du projet. Elles peuvent être des bibliothèques, des Framework, etc... et sont nécessaires pour le bon fonctionnement d'un autre élément logiciel.

	Spring est un Framework open-source pour le développement d'applications Java. Il fournit un ensemble de composants et d'outils pour faciliter la création d'applications web, desktop et mobiles.
Swing	Swing est une bibliothèque graphique du langage Java. Elle permet le développement des applications graphiques et des interfaces utilisateurs.
Lombok	Lombok fournit des annotations pour simplifier la création de code. Il permet de générer automatiquement des getters, setters, constructeurs, toString, equals, hashCode et autres méthodes répétitives dans le code. Lombok est compatible avec la plupart des IDEs Java, y compris Eclipse, IntelliJ et NetBeans, et peut être utilisé avec les Frameworks Java tels que Spring et Hibernate.
Spring Data JPA	Spring Data JPA est un Framework de Spring qui utilise JPA (Java Persistence API) pour gérer les opérations de persistance de données avec la base de données. Il permet d'implémenter des opérations de base de données telles que la lecture, l'ajout, la mise à jour et la suppression sans écrire une implémentation spécifique pour chaque opération.
JCalendar	JCalendar est une bibliothèque de composants graphiques du langage Java pour la création de calendriers et des composants visuels pour la sélection de dates, tels que des calendriers, des sélecteurs de dates, des entrées de dates, etc.
thymeleaf	Thymeleaf est un moteur de modèle côté serveur pour les applications web basées sur le Framework Spring. Il est utilisé pour générer du contenu HTML, XML ou autre type de document à partir de modèles et de données.
Spring WEB	Spring Web est une partie du Framework de développement d'application Java Spring. Il est conçu pour simplifier le développement d'applications web et offre un large éventail de fonctionnalités.
Validation	La validation garantit la qualité et l'intégrité des données, ce qui peut aider à prévenir les erreurs et les problèmes de sécurité dans les applications web tels que : <ul style="list-style-type: none"> - Côté client une rétroaction immédiate aux utilisateurs sur les erreurs de saisie de données - Côté serveur il est utilisé pour sécuriser les données en les vérifiant une fois qu'elles sont soumises au serveur.
Bootstrap	Bootstrap est un Framework open source Front-end utilisant les langages HTML, CSS, JavaScript pour la création de site et d'application web responsive (s'adapte à tout type d'écran).

6. Présentation du projet

a. Cahier des charges

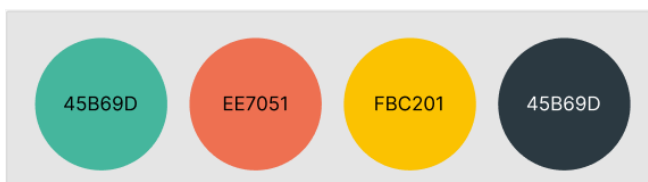
Le cahier des charges a été établi par rapport aux demandes du client.

Elle souhaite :

1. Un site vitrine lui faisant bénéficier d'une visibilité sur internet et ainsi attirer de nouveaux clients.
2. Un formulaire de contact lui permettant d'être en relation avec ces nouveaux clients et de pouvoir compléter son carnet d'adresse.
3. Pouvoir visualiser son carnet d'adresse et le gérer très facilement avec ces méthodes :
« Ajouter, supprimer et modifier un contact. »
4. Une prise de rendez-vous en ligne.
5. Pouvoir visualiser ses rendez-vous et les gérer très facilement avec ces méthodes :
« Ajouter, supprimer et modifier un rendez-vous. »
6. Et que la gestion de son carnet d'adresse et de ces rendez-vous doit être simple et rapide.

Elle nous a fourni son code couleur ainsi que son logo d'entreprise :

Code couleur



Logo



Grâce à ces éléments et étant deux à travailler sur ce projet nous avons décidé de couper le projet en deux parties.

Ma collègue s'occupe de la partie WEB (Coté utilisateurs) qui reprend les fonctionnalités suivantes :

- Site vitrine
- Prise de rendez-vous en ligne
- Formulaire de contact

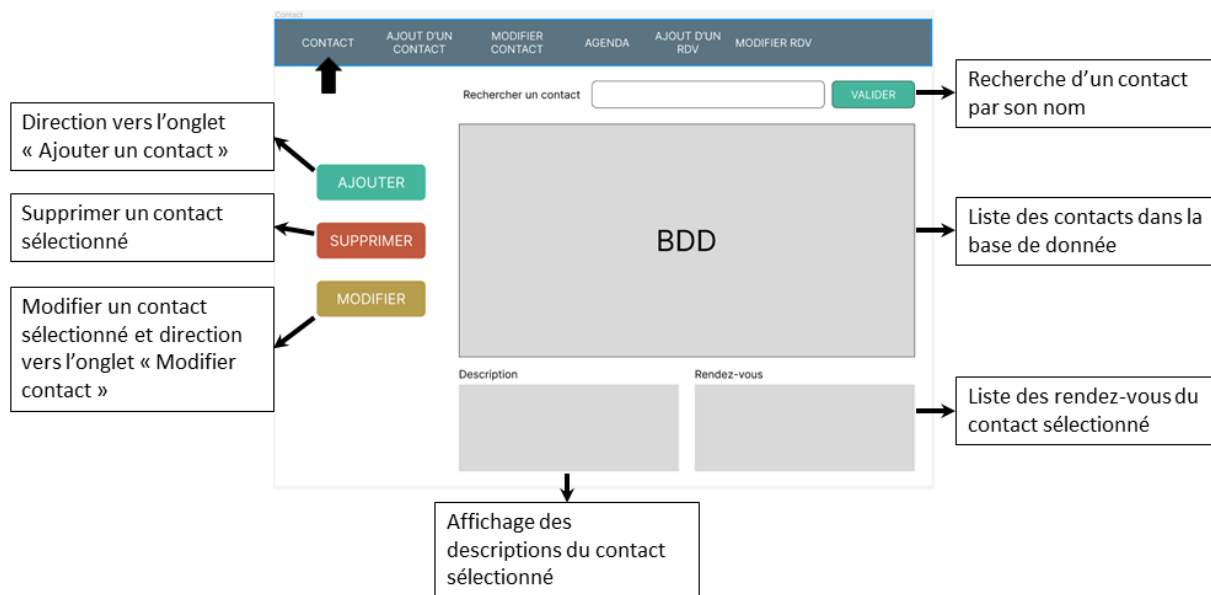
En ce qui me concerne je m'occupe de la partie Desktop (Coté Client) qui reprend les fonctionnalités suivantes :

- Visibilité des rendez-vous et contacts
- Ajout, suppression et modification d'un rendez-vous et d'un contact.

b. Maquette

Une fois les demandes de la cliente établies, j'ai commencé la réalisation de la maquette avec **FIGMA**. Cela m'a permis d'avoir une vision d'ensemble de mon projet et d'anticiper les problèmes ou incohérences éventuels. Mais aussi de faciliter la communication avec les différents protagonistes du projet. Ainsi cela clarifie les attentes et évite les malentendus et incompréhension.

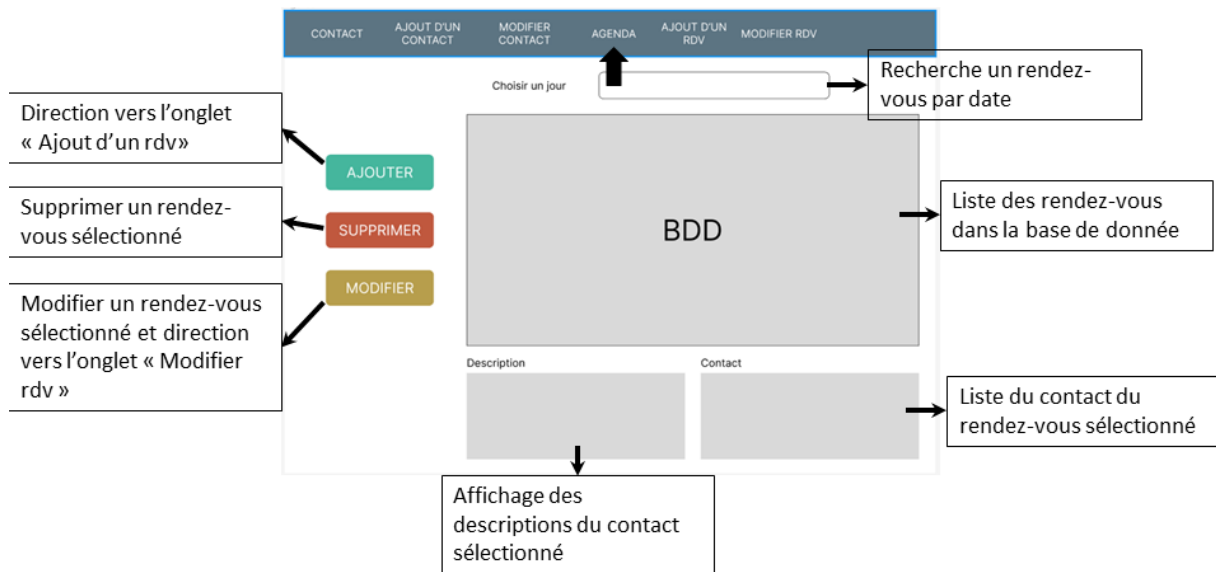
1. Contact :



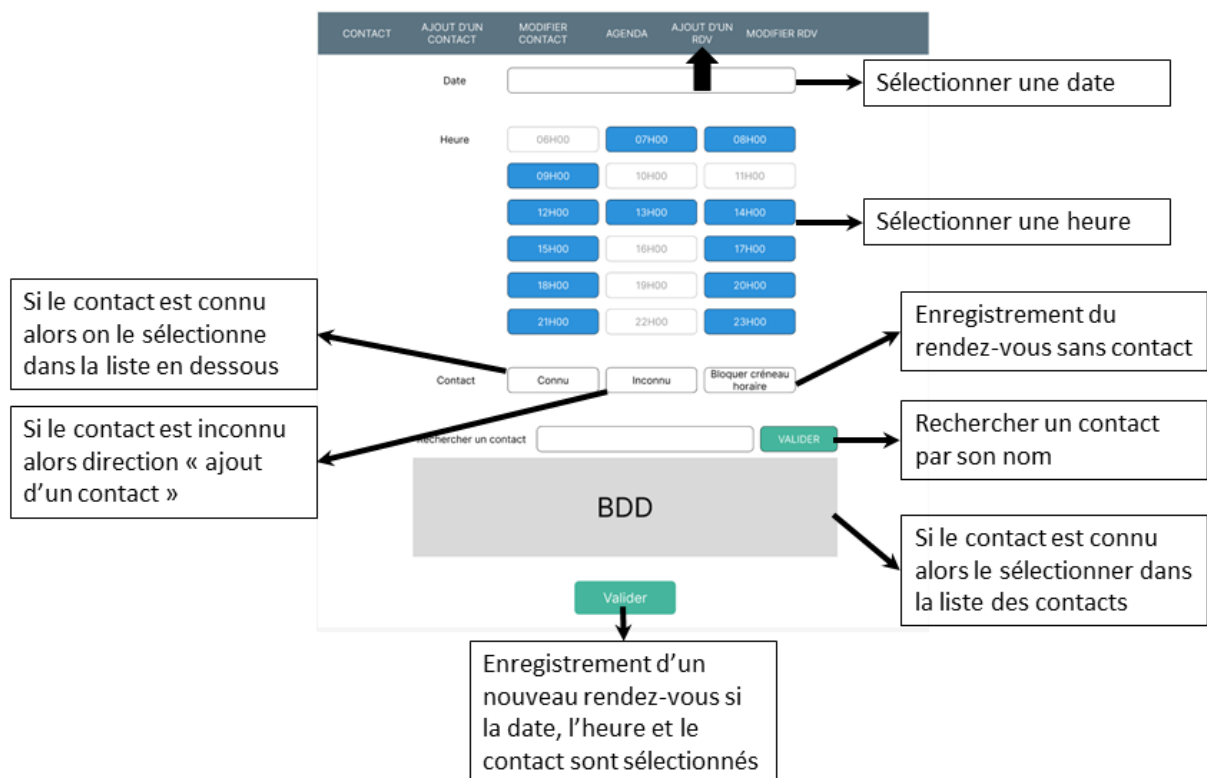
2. Ajout contact :



3. Agenda :



4. Ajout d'un rendez-vous :

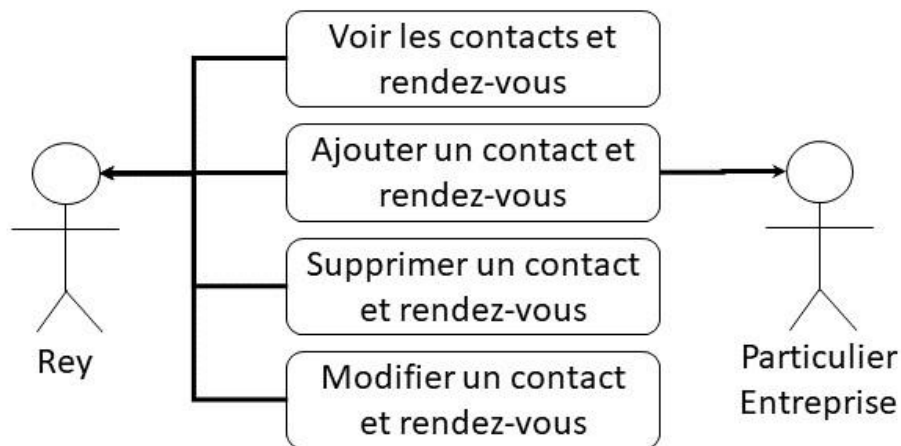


c. UML

Une fois la maquette terminée, j'ai effectué un diagramme de **cas d'utilisation** qui me permet de clarifier les besoins du système et des utilisateurs.

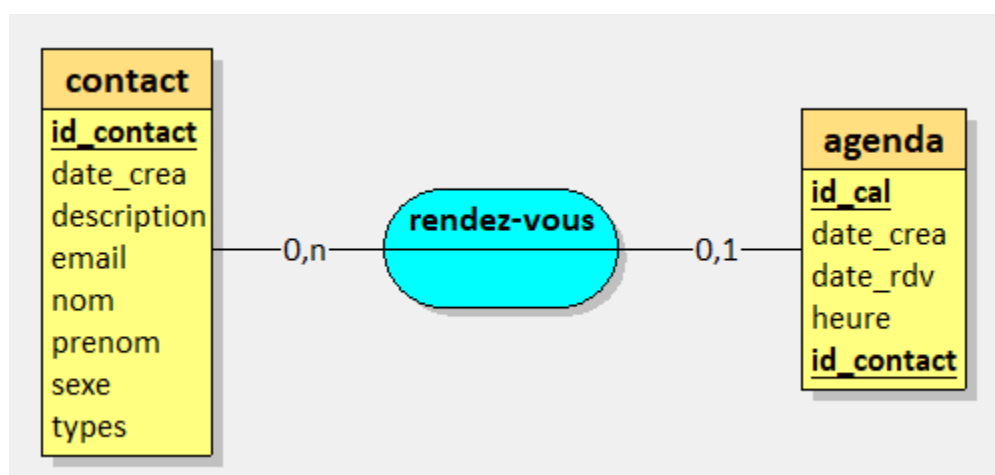
Cela me montre les fonctionnalités que l'utilisateur pourra effectuer.

Cas d'utilisation :



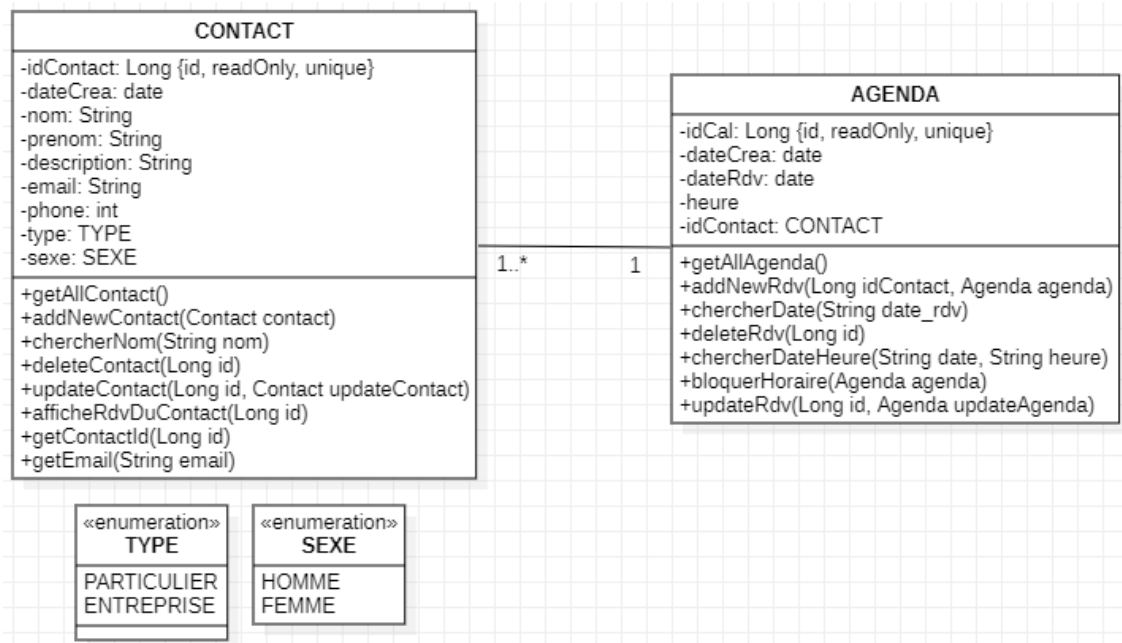
Afin de déterminer la façon dont mes données sont organisées et structurées je crée un :

MCD : Modèle Conceptuel de Données (Model Conceptual Data)

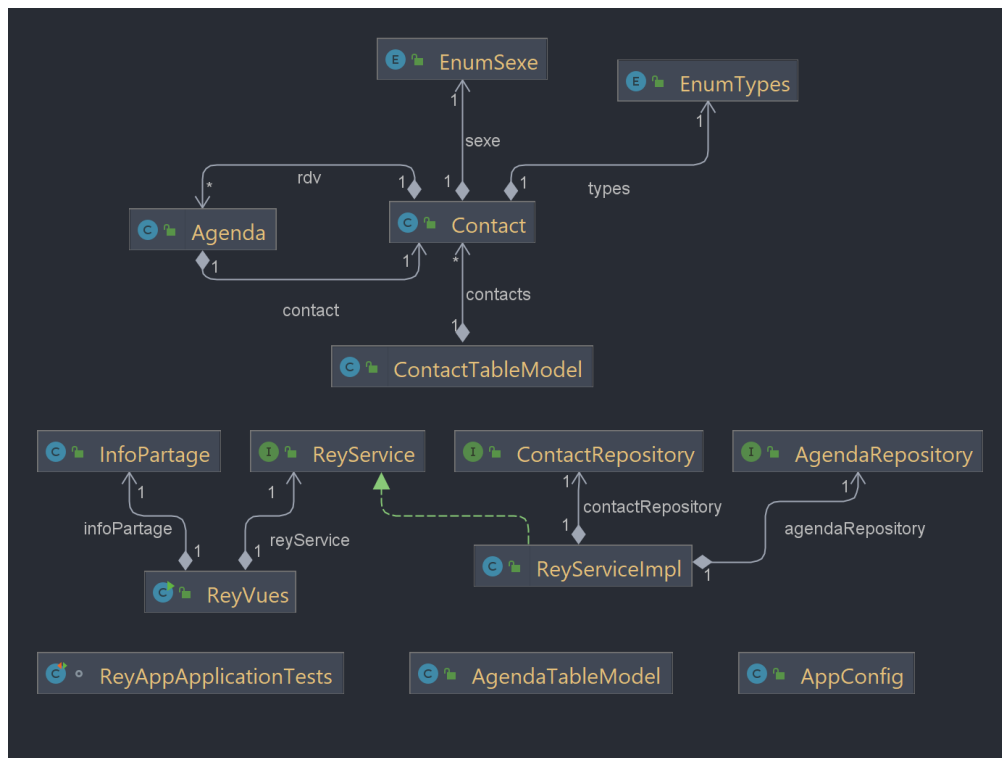


A ce niveau, je sais qu'il me faut deux tables « Contact » et « Agenda » dans ma base de données ainsi que deux types énumérés « Types » et « Sexe » et les méthodes qu'il me faudra pour le fonctionnement de mon application.

Diagramme UML : Langage de Modélisation Unifié (Unified Modeling Language)

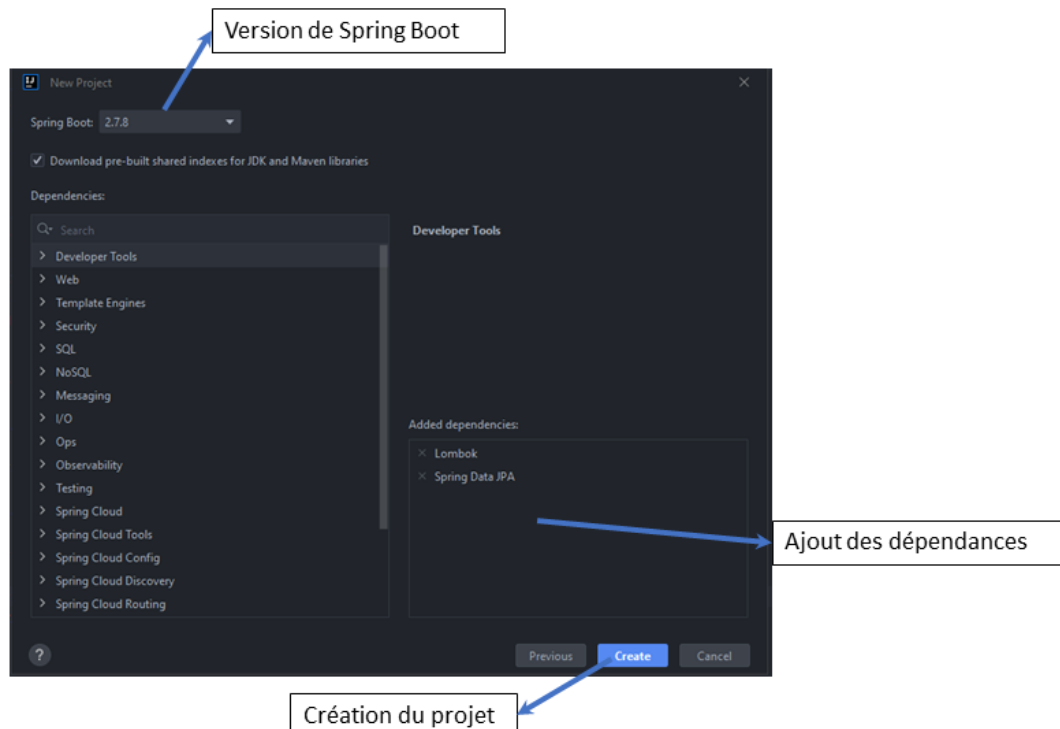
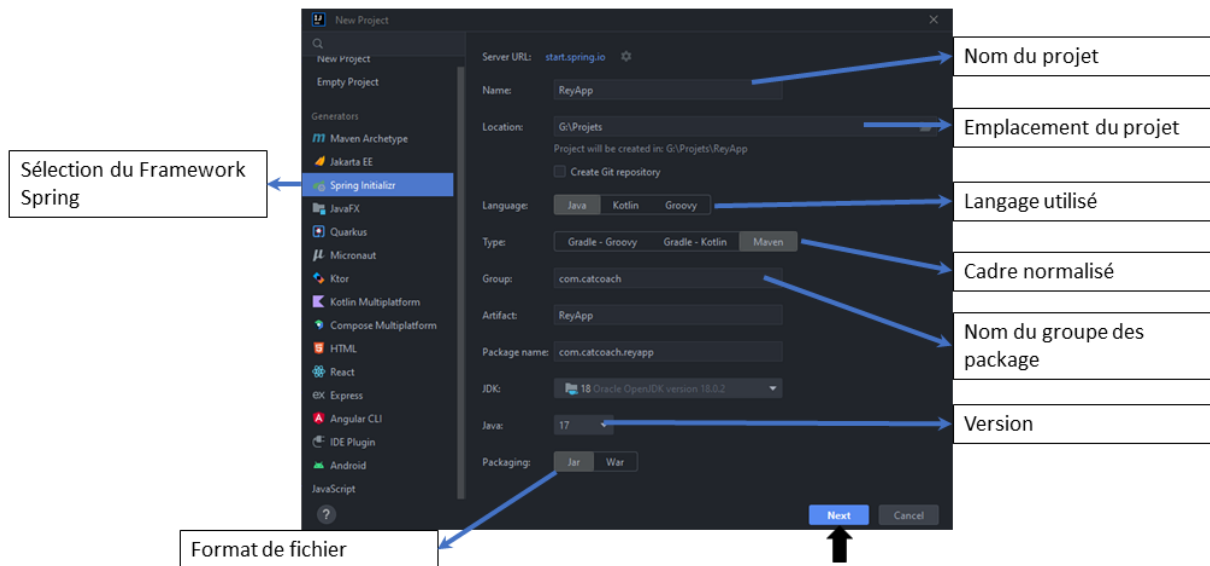


Afin de définir la structure et la relation entre mes classes je crée un **diagramme de classe** :



7. Création du projet

Un fois la maquette et les schémas effectués, qui me permettent d'avoir une vue d'ensemble de mon projet et de s'assurer que toutes les parties prenantes ont une idée claire de ce qu'il contient. Je commence alors la création de mon projet dans l'IDE IntelliJ.



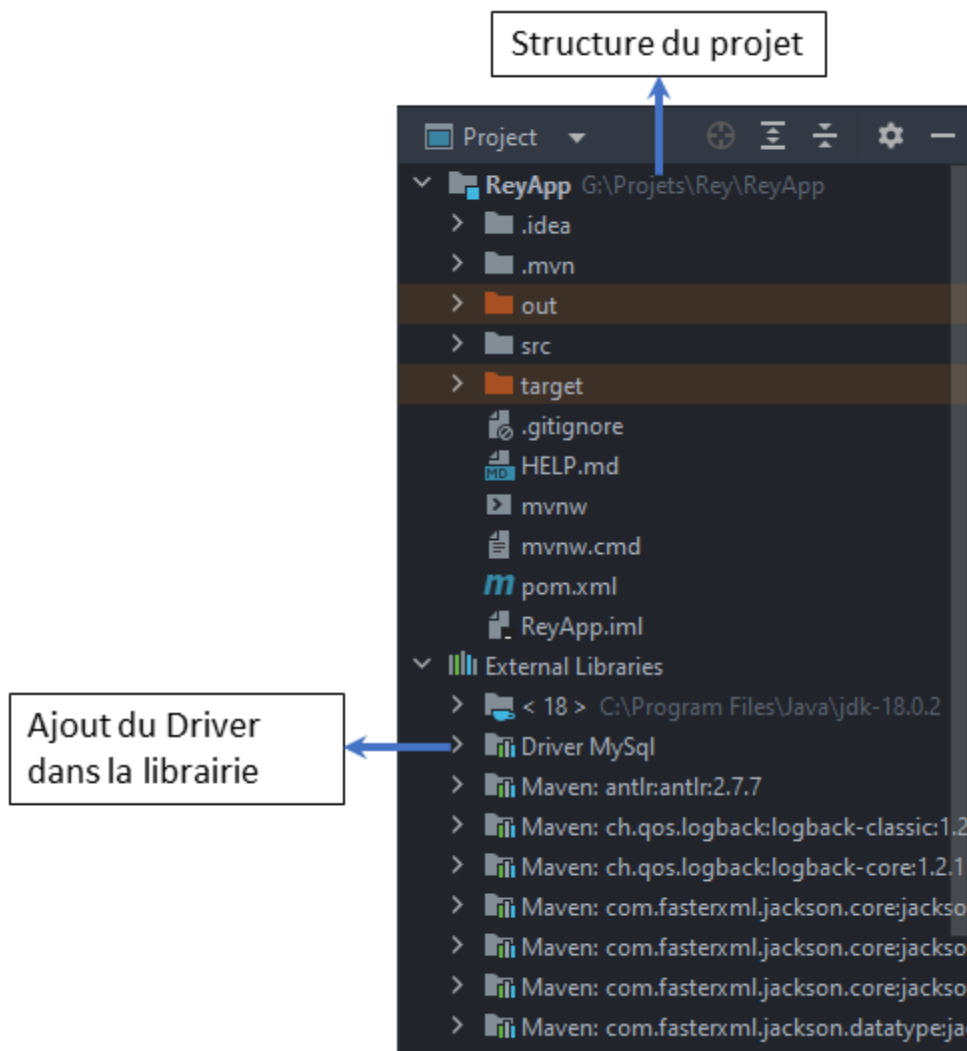
Une fois mon projet créé, je peux voir la structure de mon projet.

Il contiendra les différents Package qui me permettront d'organiser mon code en regroupant mes classes ou interfaces qui travaillent ensemble, mais aussi à maintenir une structure claire et facile à comprendre pour les autres développeurs qui travaillent sur le projet, ainsi que m'assurer de la sécurité de mes données.

Dans la librairie du projet, j'ajoute le Driver MySql qui me permettra par la suite de pouvoir communiquer avec ma base de données.

Le **Driver MySQL** est un pilote logiciel qui permet à une application Java de se connecter à une base de données MySQL.

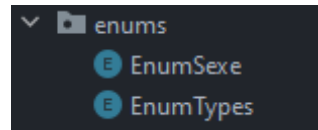
Il s'exécute sur le serveur de base de données et agit en tant que liaison pour envoyer des requêtes SQL et de recevoir des résultats de la base de données.



a. Types énumérés

Comme le schéma UML le montre, j'ai créé deux énumérations, un pour le « Types » et l'autre pour le « Sexe » des utilisateurs, cela me permet de prédéfinir la liste des constantes de valeur qui seront enregistrées dans ma base de données.

Je créer un package « enums » qui contient ces énumérations :



Ensuite je rentre mes données dans chacune d'entre elles :

EnumSexe :

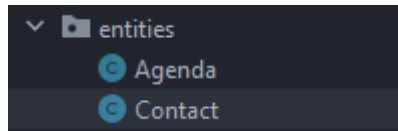
```
public enum EnumSexe {  
    3 usages  
    HOMME,  
    3 usages  
    FEMME  
}
```

EnumTypes :

```
public enum EnumTypes {  
    3 usages  
    PARTICULIER,  
    3 usages  
    ENTREPRISE  
}
```

b. Entités

Afin d'organiser, stocker et traiter de manière efficace les données de mon projet je créer un package « entities » qui contient mes objets métier « Agenda » et « Contact » dont les éléments seront persistés dans des champs et des colonnes de ma base de données ainsi que la relation entre mes entités qui sont définies en utilisant des clés étrangères.



Agenda :

```

64 usages
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
Lombok

@Entity    Annotation JPA
public class Agenda {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_cal")    Définition de ma clé primaire
    private Long idCal;

    @Temporal(TemporalType.DATE)
    @CreationTimestamp
    @Column(name = "date_crea")    Enregistrement de la date du jour automatique
    private Date dateCrea;

    @Column(name = "date_rdv")
    private String dateRdv;

    @Column(name = "heure")
    private String heure;

    1 usage    Plusieurs instances de cette entité peuvent être associées à une seule instance de l'autre entité
    @ManyToOne
    @JoinColumn(name = "id_contact", referencedColumnName = "id_contact")
    private Contact contact;
}

```

Contact :

```

55 usages
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor

@Entity
public class Contact {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_contact")
    private Long idContact;

    @Temporal(TemporalType.DATE)
    @Column(name = "date_crea")
    @CreationTimestamp
    private Date dateCrea;

    private String nom;
    private String prenom;
    private String description;

    @Column(name = "email", unique = true)
    private String email;

    private String phone;

    @Enumerated(EnumType.STRING)
    @Column(name = "types")
    private EnumTypes types;

    @Enumerated(EnumType.STRING)
    @Column(name = "sexe")
    private EnumSexe sexe;

    @OneToMany(mappedBy = "contact", fetch = FetchType.EAGER)
    private List<Agenda> rdv;
}

```

Lombok

Annotation JPA

Définition de ma clé primaire

Enregistrement de la date du jour automatique

L'adresse email doit être unique

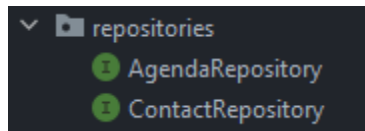
Enumération persistée en String

Un seule instance de cette entité peut être associée à plusieurs instances de l'autre entité

c. Repositories

Je crée un package « repositories » qui contient deux interfaces : « AgendaRepository » et « ContactRepository » qui me permettent de gérer l'accès aux données.

Ils fournissent une couche d'abstraction entre l'application et la base de données en permettant de cacher les détails complexes de la base de données.



AgendaRepository :

```
3 usages
@Repository
public interface AgendaRepository extends JpaRepository<Agenda, Long> {

    // CHERCHER UNE DATE
    1 usage
    List<Agenda> findByDateRdv(String date_rdv);

    // SUPPRIMER LES RENDEZ-VOUS ASSOCIE AU CONTACT SUPPRIME
    1 usage
    void deleteAgendaByContact_IdContact(long id);

    // CHERCHER UNE DATE ET UNE HEURE
    1 usage
    List<Agenda> findByDateRdvAndHeure(String date, String heure);
}
```

ContactRepository :

```
3 usages
@Repository
public interface ContactRepository extends JpaRepository<Contact, Long> {

    // CHERCHER UN CONTACT PAR SON NOM EN IGNORANT LA CASE
    1 usage
    List<Contact> findByNomContainingIgnoreCase(String nom);

    // CHERCHER SI ADRESSE MAIL EXISTE
    1 usage
    Contact findByEmail(String email);
}
```

d. Services

Pour séparer les couches de logique de l'application, je crée un package « Services » qui contient une interface « ReyService », qui définit les signature des méthodes qui sont implémentées dans la classe « ReyServiceImpl ».

ReyService :

```
4 usages 1 implementation
@Service
public interface ReyService {
    /* =====
        CONTACT
    =====*/
    // VOIR TOUS LES CONTACTS
    4 usages 1 implementation
    List<Contact> getAllContact();

    // CREER UN NOUVEAU CONTACT
    2 usages 1 implementation
    Contact addNewContact(Contact contact);

    // CHERCHER UN CONTACT PAR SON NOM
    1 usage 1 implementation
    List<Contact> chercherNom(String nom);

    // SUPPRIMER UN CONTACT
    1 usage 1 implementation
    void deleteContact(Long id);

    // MODIFIER UN CONTACT
    1 usage 1 implementation
    Contact updateContact(Long id, Contact updatedContact);

    // VOIR LES RENDEZ-VOUS DU CONTACT SELECTIONNE
    1 usage 1 implementation
    List<Agenda> afficheRdvDuContact(Long id);

    // SELECTIONNER UN CONTACT PAR SON ID
    2 usages 1 implementation
    Contact getContactId(Long id);

    // CHERCHER SI ADRESSE MAIL EXISTE
    1 usage 1 implementation
    Contact getEmail(String email);
}
```

```
1  /* =====  
2      AGENDA  
3  =====*/  
4  // VOIR TOUS LES RENDEZ-VOUS  
5  usages 1 implementation  
6  List<Agenda> getAllAgenda();  
7  
8  // CREER UN NOUVEAU RENDEZ-VOUS ATTRIBUER A UN CONTACT  
9  usages 1 implementation  
10 Agenda addNewRdv(Long idContact, Agenda agenda);  
11  
12 // CHERCHER UN DATE DANS LA BASE DE DONNEE  
13 usage 1 implementation  
14 List<Agenda> ChercheDate(String date_rdv);  
15  
16 // SUPPRIMER UN RENDEZ-VOUS  
17 usage 1 implementation  
18 void deleteRdv(Long id);  
19  
20 // CHERCHER UNE DATE ET UNE HEURE  
21 usages 1 implementation  
22 List<Agenda> chercherDateHeure(String date, String heure);  
23  
24 // BLOQUER LE CRENEAU HORAIRE  
25 usage 1 implementation  
26 Agenda bloquerHoraire(Agenda agenda);  
27  
28 // MODIFIER UN RENDEZ-VOUS SELECTIONNE  
29 usage 1 implementation  
30 Agenda updateRdv(Long id, Agenda updatedAgenda);  
31 }
```

Dans ma classe « ReyServiceImpl » j'appelle les méthodes créées précédemment pour séparer la responsabilité et améliorer la flexibilité de mon code.

Je crée un constructeur de mes repositories pour séparer la responsabilité de chaque couche de l'application.

ReyServiceImpl :

```
@Service
@Transactional
public class ReyServiceImpl implements ReyService{

    10 usages
    private ContactRepository contactRepository;
    10 usages
    private AgendaRepository agendaRepository;

    // CONSTRUCTEUR
    public ReyServiceImpl(ContactRepository contactRepository, AgendaRepository agendaRepository) {
        this.contactRepository = contactRepository;
        this.agendaRepository = agendaRepository;
    }

    /* =====
                                CONTACT
    =====*/
    // VOIR LA LISTE DES CONTACTS
    4 usages
    @Override
    public List<Contact> getAllContact() {
        return contactRepository.findAll();
    }

    // ENREGISTRER UN NOUVEAU CONTACT
    2 usages
    @Override
    public Contact addNewContact(Contact contact) {
        return contactRepository.save(contact);
    }

    // CHERCHER UN CONTACT PAR SON NOM
    1 usage
    @Override
    public List<Contact> chercherNom(String nom) {
        return contactRepository.findByNomContainingIgnoreCase(nom);
    }

    // SUPPRIMER UN CONTACT
    1 usage
    @Override
    public void deleteContact(Long id) {
        agendaRepository.deleteAgendaByContact_IdContact(id);
        contactRepository.deleteById(id);
    }
}
```



```
// MODIFIER UN CONTACT SELECTIONNE
1 usage
@Override
public Contact updateContact(Long id, Contact updatedContact) {
    Contact contact = contactRepository.findById(id).orElseThrow();
    contact.setNom(updatedContact.getNom());
    contact.setPrenom(updatedContact.getPrenom());
    contact.setDescription(updatedContact.getDescription());
    contact.setEmail(updatedContact.getEmail());
    contact.setPhone(updatedContact.getPhone());
    contact.setTypes(updatedContact.getTypes());
    contact.setSexe(updatedContact.getSexe());
    return contactRepository.save(contact);
}

// VOIR LES RENDEZ-VOUS DU CONTACT SELECTIONNE
1 usage
@Override
public List<Agenda> afficheRdvDuContact(Long id) {
    // Récupération du contact associé aux rendez-vous
    Contact contact = contactRepository.findById(id).orElse( other: null);
    // Récupération de la liste de rendez-vous du contact
    List<Agenda> listeRdv = new ArrayList<>();
    if (contact != null) {
        listeRdv = contact.getRdv();
    }
    return listeRdv;
}

// CHERCHER SI ADRESSE MAIL EXISTE
1 usage
@Override
public Contact getEmail(String email) {
    return contactRepository.findByEmail(email);
}
```

```

/* =====
AGENDA
===== */
// ENREGISTRER UN NOUVEAU RENDEZ-VOUS
2 usages
@Override
public Agenda addNewRdv(Long idContact, Agenda agenda) {return agendaRepository.save(agenda);}

// VOIR LA LISTE DES RENDEZ-VOUS
5 usages
@Override
public List<Agenda> getAllAgenda() {return agendaRepository.findAll();}

// CHERCHER UN DATE DANS LA BASE DE DONNEE
1 usage
@Override
public List<Agenda> ChercheDate(String date_rdv) {return agendaRepository.findByDateRdv(date_rdv); }

// SUPPRIMER UN RENDEZ-VOUS
1 usage
@Override
public void deleteRdv(Long id) { agendaRepository.deleteById(id); }

// CHERCHER UNE DATE ET UNE HEURE
2 usages
@Override
public List<Agenda> chercherDateHeure(String date, String heure) {
    return agendaRepository.findByDateRdvAndHeure(date, heure);
}

// BLOQUER LE CRENEAU HORAIRE
1 usage
@Override
public Agenda bloquerHoraire(Agenda agenda) { return agendaRepository.save(agenda);}

// MODIFIER UN RENDEZ-VOUS SELECTIONNE
1 usage
@Override
public Agenda updateRdv(Long id, Agenda updatedAgenda) {
    Agenda agenda = agendaRepository.findById(id).orElseThrow();
    agenda.setDateRdv(updatedAgenda.getDateRdv());
    agenda.setHeure(updatedAgenda.getHeure());
    return agendaRepository.save(agenda);
}

// SELECTIONNER UN CONTACT PAR SON ID
2 usages
@Override
public Contact getContactId(Long id) { return contactRepository.findById(id).orElse( other: null); }
}

```

e. Connexion

Pour pouvoir me connecter à ma base de données, je dois fournir certaines informations qui me permettent de créer un conteneur de beans qui est fondamentale à l'architecture du projet.

Il gère la création, la configuration et l'injection des objets dans l'application.

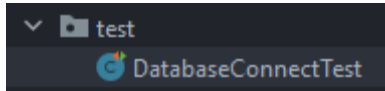
```
2 usages
@Configuration Configuration pour les conteneurs de beans
@ComponentScan("com.catcoatch.reyapp") Scan du package pour trouver les services, les repositories et les contrôleurs pour les
inclure dans le conteneur de beans
public class AppConfig {
    1 usage
    @Bean Créer un objet pour chaque méthode annotée
    public DataSource dataSource() {
        return DataSourceBuilder.create()
            .url("jdbc:mysql://localhost:3306/reyTest")
            .username("root") Instance de connexion à la base de données
            .password("ErodRyuk09Loop")
            .driverClassName("com.mysql.jdbc.Driver")
            .build();
    }
    1 usage
    @Bean Instance de gestion des entités et les transactions avec la base de données
    public EntityManagerFactory entityManagerFactory() {
        HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
        vendorAdapter.setGenerateDdl(true);

        LocalContainerEntityManagerFactoryBean factory = new LocalContainerEntityManagerFactoryBean();
        factory.setJpaVendorAdapter(vendorAdapter);
        factory.setPackagesToScan("com.catcoatch.reyapp.entities");
        factory.setDataSource(dataSource());
        factory.afterPropertiesSet();

        return factory.getObject();
    }
    @Bean Instance de transactions avec la base de données
    public PlatformTransactionManager transactionManager() {
        JpaTransactionManager txManager = new JpaTransactionManager();
        txManager.setEntityManagerFactory(entityManagerFactory());
        return txManager;
    }
}
```

8. Tests

Je crée un package « test » pour effectuer tous les tests des méthodes de mon application afin de vérifier leurs bons fonctionnements.



Test base de données :

```

19  @RunWith(SpringRunner.class)
20  @ContextConfiguration(classes = {AppConfig.class})
21  public class DatabaseConnectTest {
22
23      1 usage
24      @Autowired
25      private AppConfig appConfig;
26
27      @Autowired
28      private ReyService reyService;
29
30      // TEST CONNEXION BASE DE DONNEES
31      @Test
32      public void testConnection() throws SQLException {
33          try {
34              Connection connection = appConfig.dataSource().getConnection();
35              System.out.println("Connection réussie ! La base de données est connectée.");
36              assertNotNull(connection);
37          } catch (SQLException e) {
38              System.out.println("La connexion a échoué : " + e.getMessage());
39          }
40      }

```

```

11:06:15.628 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - Before test method: context [DefaultTestContext@3ce16309 testClass = DatabaseConnectTest]
11:06:15.637 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate - Retrieved ApplicationContext [1083962448] from cache with key [[MergedContextConfiguration@6ab
11:06:15.637 [main] DEBUG org.springframework.test.context.cache - Spring test ApplicationContext cache statistics: [DefaultContextCache@2ed84be9 size = 1, maxSize = 32, parentContextCount = 0, hitCo
11:06:15.637 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate - Retrieved ApplicationContext [1083962448] from cache with key [[MergedContextConfiguration@6ab
11:06:15.637 [main] DEBUG org.springframework.test.context.cache - Spring test ApplicationContext cache statistics: [DefaultContextCache@2ed84be9 size = 1, maxSize = 32, parentContextCount = 0, hitCo
11:06:15.812 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate - Retrieved ApplicationContext [1083962448] from cache with key [[MergedContextConfiguration@6ab
11:06:15.812 [main] DEBUG org.springframework.test.context.cache - Spring test ApplicationContext cache statistics: [DefaultContextCache@2ed84be9 size = 1, maxSize = 32, parentContextCount = 0, hitCo
Connection réussie ! La base de données est connectée.
11:06:15.814 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate - Retrieved ApplicationContext [1083962448] from cache with key [[MergedContextConfiguration@6ab
11:06:15.814 [main] DEBUG org.springframework.test.context.cache - Spring test ApplicationContext cache statistics: [DefaultContextCache@2ed84be9 size = 1, maxSize = 32, parentContextCount = 0, hitCo
11:06:15.816 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate - Retrieved ApplicationContext [1083962448] from cache with key [[MergedContextConfiguration@6ab
11:06:15.816 [main] DEBUG org.springframework.test.context.cache - Spring test ApplicationContext cache statistics: [DefaultContextCache@2ed84be9 size = 1, maxSize = 32, parentContextCount = 0, hitCo
11:06:15.816 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate - Retrieved ApplicationContext [1083962448] from cache with key [[MergedContextConfiguration@6ab
11:06:15.816 [main] DEBUG org.springframework.test.context.cache - Spring test ApplicationContext cache statistics: [DefaultContextCache@2ed84be9 size = 1, maxSize = 32, parentContextCount = 0, hitCo
11:06:15.816 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate - Retrieved ApplicationContext [1083962448] from cache with key [[MergedContextConfiguration@6ab
11:06:15.816 [main] DEBUG org.springframework.test.context.cache - Spring test ApplicationContext cache statistics: [DefaultContextCache@2ed84be9 size = 1, maxSize = 32, parentContextCount = 0, hitCo
11:06:15.818 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate - Retrieved ApplicationContext [1083962448] from cache with key [[MergedContextConfiguration@6ab
11:06:15.818 [main] DEBUG org.springframework.test.context.cache - Spring test ApplicationContext cache statistics: [DefaultContextCache@2ed84be9 size = 1, maxSize = 32, parentContextCount = 0, hitCo
11:06:15.819 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate - Retrieved ApplicationContext [1083962448] from cache with key [[MergedContextConfiguration@6ab
11:06:15.819 [main] DEBUG org.springframework.test.context.cache - Spring test ApplicationContext cache statistics: [DefaultContextCache@2ed84be9 size = 1, maxSize = 32, parentContextCount = 0, hitCo
11:06:15.819 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - After test method: context [DefaultTestContext@3ce16309 testClass = DatabaseConnectTest]

```

Listes des rendez-vous :

```
61 // VOIR LES RENDEZ-VOUS
62 @Test
63 public void testAfficheListeRdv() throws SQLException{
64     List<Agenda> list = reyService.getAllAgenda();
65     assertNotNull(list);
66 }
```

10 :39 :22.330 [main] DEBUG org.hibernate.internal.util.EntityPrinter - Listing entities:

10 :39:22.330 [main] DEBUG org.hibernate.internal.util.EntityPrinter -
com.catcoatch.reyapp.entities.Contact{types=PARTICULIER, idContact=2, phone=0602030103,
rdv=[com.catcoatch.reyapp.entities.Agenda#7], description=Je cherche des conseils pour réussir ma
carrière professionnel., dateCrea=2023-02-03, sexe=HOMME, nom=Petit, prenom=Pierre,
email=pierre@gmail.com}

10 :39:22.330 [main] DEBUG org.hibernate.internal.util.EntityPrinter -
com.catcoatch.reyapp.entities.Agenda{dateRdv=23-02-2023, heure=16H00, idCal=7,
contact=com.catcoatch.reyapp.entities.Contact#2, dateCrea=2023-02-03}

10:39:22.330 [main] DEBUG org.hibernate.internal.util.EntityPrinter -
com.catcoatch.reyapp.entities.Agenda{dateRdv=23-02-2023, heure=19H00, idCal=4, contact=null,
dateCrea=2023-02-03}

Ajout d'un contact :

```
37 // AJOUT D'UN CONTACT
38 @Test
39 public void ajoutContact() throws SQLException{
40     Contact contact = new Contact();
41     Contact result = reyService.addNewContact(contact);
42     assertNotNull(result);
43 }
```

10:39:22.352 [main] DEBUG org.hibernate.internal.util.EntityPrinter - Listing entities:

10:39:22.352 [main] DEBUG org.hibernate.internal.util.EntityPrinter -
com.catcoatch.reyapp.entities.Contact{types=null, idContact=13, phone=null, rdv=null,
description=null, dateCrea=Thu Feb 09 10:39:22 CET 2023, sexe=null, nom=null, prenom=null,
email=null}

Listes des contacts :

```
54      // VOIR LES CONTACTS
55      @Test
56      public void testAfficheListeContact() throws SQLException{
57          List<Contact> list = reyService.getAllContact();
58          assertNotNull(list);
59      }
```

10:39:22.471 [main] DEBUG org.hibernate.internal.util.EntityPrinter - Listing entities:

10:39:22.471 [main] DEBUG org.hibernate.internal.util.EntityPrinter -
com.catcoatch.reyapp.entities.Contact{types=PARTICULIER, idContact=2, phone=0602030103,
rdv=[com.catcoatch.reyapp.entities.Agenda#7], description=Je cherche des conseils pour réussir ma
carrière professionnel., dateCrea=2023-02-03, sexe=HOMME, nom=Petit, prenom=Pierre,
email=pierre@gmail.com}

10:39:22.471 [main] DEBUG org.hibernate.internal.util.EntityPrinter -
com.catcoatch.reyapp.entities.Contact{types=PARTICULIER, idContact=3, phone=0720103022, rdv=[],
description=J'ai besoin d'aide afin de m'améliorer dans ma recherche d'emploi., dateCrea=2023-02-
03, sexe=FEMME, nom=Gibon, prenom=Flore, email=gibon@gmail.com}

10:39:22.471 [main] DEBUG org.hibernate.internal.util.EntityPrinter -
com.catcoatch.reyapp.entities.Contact{types=null, idContact=13, phone=null, rdv=[],
description=null, dateCrea=2023-02-09, sexe=null, nom=null, prenom=null, email=null}

10:39:22.471 [main] DEBUG org.hibernate.internal.util.EntityPrinter -
com.catcoatch.reyapp.entities.Agenda{dateRdv=23-02-2023, heure=16H00, idCal=7,
contact=com.catcoatch.reyapp.entities.Contact#2, dateCrea=2023-02-03}

Ajout d'un rendez-vous :

```
45      // AJOUT D'UN RENDEZ-VOUS
46      @Test
47      public void ajoutRdv() {
48          Long idContact = 1L;
49          Agenda agenda = new Agenda();
50          Agenda result = reyService.addNewRdv(idContact, agenda);
51          assertNotNull(result);
52      }
```

10:39:22.481 [main] DEBUG org.hibernate.internal.util.EntityPrinter - Listing entities:

10:39:22.481 [main] DEBUG org.hibernate.internal.util.EntityPrinter -
com.catcoatch.reyapp.entities.Agenda{dateRdv=null, heure=null, idCal=8, contact=null,
dateCrea=Thu Feb 09 10:39:22 CET 2023}

Chercher un contact par son nom :

```
68      // VOIR UN CONTACT PAR SON NOM
69      @Test
70      public void nomContact() throws SQLException{
71          String nom = "Gibon";
72          List<Contact> list = reyService.chercherNom(nom);
73          assertNotNull(list);
74      }
75  }
```

10:52:26.956 [main] DEBUG org.hibernate.internal.util.EntityPrinter - Listing entities:

10:52:26.956 [main] DEBUG org.hibernate.internal.util.EntityPrinter -
com.catcoatch.reyapp.entities.Contact{types=PARTICULIER, idContact=3, phone=0720103022, rdv=[],
description=J'ai besoin d'aide afin de m'améliorer dans ma recherche d'emploi., dateCrea=2023-02-
03, sexe=FEMME, nom=Gibon, prenom=Flore, email=gibon@gmail.com}

9. Interface graphique

a. Création des modèles de table

Pour pouvoir voir la liste de mes contacts et mes rendez-vous, je dois créer des modèles des tables de ma base de données en déterminant mes colonnes et à quoi elles correspondent dans ma base.

AgendaTableModel :

```

22 usages
public class AgendaTableModel extends AbstractTableModel {

    7 usages
    private List<Agenda> agenda;

    2 usages
    private String[] columnNames = {"Date", "Heure"};

    5 usages
    public AgendaTableModel(List<Agenda> listAgenda) {
        this.agenda = listAgenda;
    }

    @Override
    public int getRowCount() {
        return agenda.size();
    }

    @Override
    public int getColumnCount() {return columnNames.length;}

    @Override
    public String getColumnName(int columnIndex) { return columnNames[columnIndex]; }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        Agenda columnAgenda = agenda.get(rowIndex);
        switch (columnIndex) {
            case 0: return columnAgenda.getDateRdv();
            case 1: return columnAgenda.getHeure();
            default: return null;
        }
    }

    2 usages
    public void setAgenda(List<Agenda> agenda) {
        this.agenda = agenda;
        fireTableDataChanged();
    }

```



```
// RECUPERE LA LISTE DES RENDEZ-VOUS
4 usages
public List<Agenda> getAgenda() {
    return agenda;
}

// RECUPERE L ID DU RENDEZ-VOUS DANS LA TABLE A PARTIR DE SON INDEX
2 usages
public Long getIdRdv(int row) {
    Agenda agenda1 = agenda.get(row);
    return agenda1.getIdCal();
}

// MODIFIER UN RENDEZ-VOUS SELECTIONNE
1 usage
public void updateRow(int ligneSelectionnee, Agenda agendas) {
    agenda.set(ligneSelectionnee, agendas);
    fireTableRowsUpdated(ligneSelectionnee, ligneSelectionnee);
}
}
```

ContactTableModel :

```

28 usages
public class ContactTableModel extends AbstractTableModel {

    10 usages
    private List<Contact> contacts;

    2 usages
    private String[] columnNames = {"Nom", "Prenom", "Email", "Téléphone", "Type", "Sexe"};

    4 usages
    public ContactTableModel(List<Contact> contacts) {this.contacts = contacts; }

    @Override
    public int getRowCount() { return contacts.size();}

    @Override
    public int getColumnCount() {return columnNames.length;}

    @Override
    public String getColumnName(int columnIndex) { return columnNames[columnIndex]; }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        Contact contact = contacts.get(rowIndex);
        switch (columnIndex) {
            case 0: return contact.getNom();
            case 1: return contact.getPrenom();
            case 2: return contact.getEmail();
            case 3: return contact.getPhone();
            case 4: return contact.getTypes();
            case 5: return contact.getSexe();
            default: return null;
        }
    }
}

// RECUPERE L ID D UN CONTACT DANS LA TABLE A PARTIR DE SON INDEX
5 usages
public Long getId_Contact(int row) {
    Contact contact = contacts.get(row);
    return contact.getIdContact();
}
    
```

```
2 usages
public void setContacts(List<Contact> contacts) {
    this.contacts = contacts;
    fireTableDataChanged();
}

// RECUPERER LA LISTE DES CONTACTS DE LA LIGNE SELECTIONNEE
1 usage
public Contact getContact(int ligneSelectionnee) {
    return contacts.get(ligneSelectionnee);
}

// RECUPERER LA LISTE DES CONTACTS
4 usages
public List<Contact> getContacts() {
    return contacts;
}

// MODIFIER UN CONTACT SELECTIONNE
1 usage
public void updateRow(int ligneSelectionnee, Contact contact) {
    contacts.set(ligneSelectionnee, contact);
    fireTableRowsUpdated(ligneSelectionnee, ligneSelectionnee);
}

// RECUPERER LA DESCRIPTION D UN CONTACT DANS LA TABLE A PARTIR DE SON INDEX
2 usages
public String getDescription(int row) {
    Contact contact = contacts.get(row);
    return contact.getDescription();
}

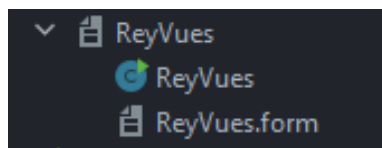
// VOIR LE CONTACT DU RENDEZ-VOUS SELECTIONNE
1 usage
public void setContact(Contact contact){
    this.contacts = Collections.singletonList(contact);
    fireTableDataChanged();
}
}
```

b. Création de l'interface Swing

Pour la création de mon interface graphique, je crée un package qui contient les vues (les pages) de mon projet.



Dans mon package je crée un nouveau « GUI Form » qui me sert d'interface visuelle pour que l'utilisateur puisse interagir avec le programme.



La classe « ReyVues » contient les méthodes pour faire fonctionner mon application.

Le fichier « ReyVues.form » contient les éléments graphiques.

Je commence par tester le lancement d'une fenêtre vide

Pour pouvoir lancer la fenêtre, je crée une méthode main () dans ma classe « ReyVues » qui sera le point d'entrée de mon application :

Cette méthode comprend :

1. La connexion à ma base de données au lancement de la fenêtre.
2. Création d'une nouvelle fenêtre avec un titre.
3. Ajuste la taille de la fenêtre en fonction du contenu
4. Récupère la taille de l'écran
5. Stock la hauteur de l'écran
6. Stock la largeur de l'écran
7. Définit la taille de la fenêtre à la moitié de la largeur et de la hauteur de l'écran
8. Centrer la fenêtre
9. Rendre la fenêtre visible
10. Déterminé l'élément principal que contient la fenêtre
11. Fermer la fenêtre

```

4 usages
@Component
public class ReyVues extends JFrame{

    1 usage
    private ReyService reyService;

    2 usages
    private JPanel rootPanel;

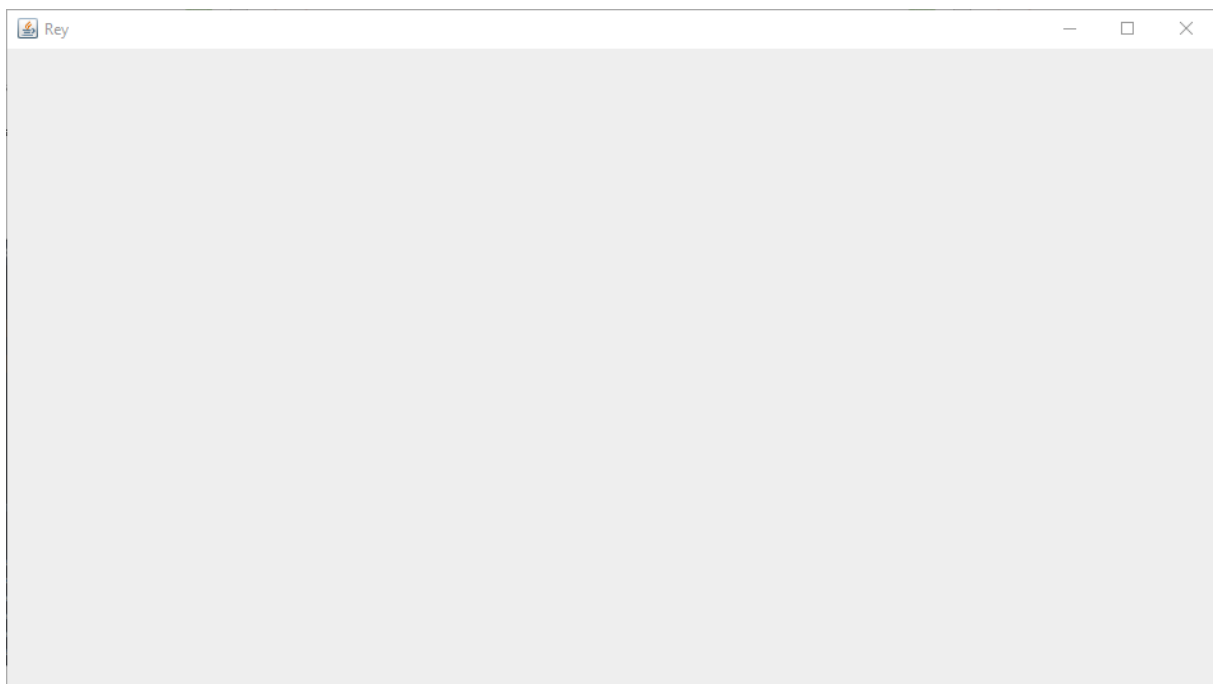
    public ReyVues(ReyService reyService) { this.reyService = reyService; }

    public static void main(String[] args) {
        // CONNEXION A LA BASE DE DONNEES
        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
        ReyVues app = context.getBean(ReyVues.class);

        JFrame frame = new JFrame( title: "Rey");
        frame.pack();
        Dimension tailleEcran = Toolkit.getDefaultToolkit().getScreenSize();
        int height = tailleEcran.height;
        int width = tailleEcran.width;
        frame.setSize( width: width/2, height: height/2);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
        frame.setContentPane(app.rootPanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

Le test ouvrira une fenêtre vide qui sera centrée à l'écran avec mon titre en haut de ma fenêtre.



Ensuite, je place mes éléments graphiques dans « ReyVues.form ».

Une fois mes éléments placés, je crée les différentes méthodes de fonctionnement et événements sur mes boutons :

```
@Component
public class ReyVues extends JFrame {

    private ReyService reyService;

    private InfoPartage infoPartage;

    // Listes des éléments graphique enlevé pour une meilleure lisibilité
    private JPanel rootPanel;
    private JTabbedPane tabbedMenu;
    //

    JDateChooser dateChooser = new JDateChooser();
    JDateChooser dateChoix = new JDateChooser();
    JDateChooser dateChoixModifRdv = new JDateChooser();
    private ButtonGroup GroupTypes = new ButtonGroup();
    private ButtonGroup GroupSexe = new ButtonGroup();

    // CONSTRUCTEUR
    public ReyVues(ReyService reyService, InfoPartage infoPartage) {
        this.reyService = reyService;
        this.infoPartage = infoPartage;

        // ACTION EVENT

        // Gestion des différents onglets
        tabbedMenu.addChangeListener(e -> {
            int selectedIndex = tabbedMenu.getSelectedIndex();
            updateTable(selectedIndex);
        });
    }
}
```

```
=====
CONTACT
=====

// Direction vers l'onglet "Ajout d'un contact" du bouton "Ajouter" de
l'onglet "Contact"
btnAjoutContact.addActionListener(e -> {
    // Obtenez l'index de l'onglet actuel
    int currentIndex = tabbedMenu.getSelectedIndex();
    // Calculez l'index de l'onglet suivant
    int nextIndex = (currentIndex + 1) % tabbedMenu.getTabCount();
    // Sélectionnez l'onglet suivant
    tabbedMenu.setSelectedIndex(nextIndex);
});

// Direction vers l'onglet "Modifier contact" d'un contact sélectionné dans
la table
btnModifierContact.addActionListener(e -> {
    int ligneSelectionnee = tableContactContact.getSelectedRow();
    ContactTableModel tableModel = (ContactTableModel)
        tableContactContact.getModel();
    if (ligneSelectionnee == -1) {
        JOptionPane.showMessageDialog(tabbedMenu, "Veuillez sélectionner un
        contact à modifier.");
    } else {
        Contact contact = tableModel.getContact(ligneSelectionnee);
        labelId.setText(contact.getIdContact().toString());
        textNomModif.setText(contact.getNom());
        textPrenomModif.setText(contact.getPrenom());
        editorDescrModif.setText(contact.getDescription());
        textEmailModif.setText(contact.getEmail());
        textPhoneModif.setText(contact.getPhone());

        if (contact.getTypes() == EnumTypes.PARTICULIER) {
            radioBtnParticulierModif.setSelected(true);
            panelRadioSexeModif.setVisible(true);
            textPrenomModif.setVisible(true);
            labelPrenomModif.setVisible(true);
        } else if (contact.getTypes() == EnumTypes.ENTREPRISE) {
            radioBtnEntrepriseModif.setSelected(true);
            panelRadioSexeModif.setVisible(false);
            textPrenomModif.setVisible(false);
            labelPrenomModif.setVisible(false);
        }
        if (contact.getSexe() == EnumSexe.FEMME) {
            radioBtnFemmeModif.setSelected(true);
        } else if (contact.getSexe() == EnumSexe.HOMME) {
            radioBtnHommeModif.setSelected(true);
        }
        // Obtenez l'index de l'onglet actuel
        int currentIndex = tabbedMenu.getSelectedIndex();
        // Calculez l'index de l'onglet suivant
        int nextIndex = (currentIndex + 2) % tabbedMenu.getTabCount();
        // Sélectionnez l'onglet suivant
        tabbedMenu.setSelectedIndex(nextIndex);
    }
});
```

```
// Supprimer le contact sélectionné dans la table depuis son id dans la BDD
btnSupContact.addActionListener(e -> {supprimerContact();});

// Création d'un nouveau contact
btnSaveContact.addActionListener(e -> { saveContact(); });

// Chercher un contact avec son nom
btnValideRecherche.addActionListener(e -> { chercherContact();});

// Modifier un contact sélectionné
btnModifier.addActionListener(e -> { modifierContact();});

// Voir la description du contact sélectionné
tableContactContact.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        voirDescription();
    }
});

// Événement au clique du radio bouton entreprise
radioBtnEntreprise.addActionListener(e -> {
    panelRadioSexe.setVisible(false);
    textPrenom.setVisible(false);
    labelPrenom.setVisible(false);
});

// Événement au clique du radio bouton particulier
radioBtnParticulier.addActionListener(e -> {
    panelRadioSexe.setVisible(true);
    textPrenom.setVisible(true);
    labelPrenom.setVisible(true);
});

// Événement au clique du radio bouton entreprise
radioBtnEntrepriseModif.addActionListener(e -> {
    panelRadioSexeModif.setVisible(false);
    textPrenomModif.setVisible(false);
    labelPrenomModif.setVisible(false);
});

// Événement au clique du radio bouton particulier
radioBtnParticulierModif.addActionListener(e -> {
    panelRadioSexeModif.setVisible(true);
    textPrenomModif.setVisible(true);
    labelPrenomModif.setVisible(true);
});
```



```
=====
                                AGENDA
=====

// Événement aux cliques du bouton pour se diriger vers l'onglet "Ajout
d'un rendez-vous"
btnAjouterRdv.addActionListener(e -> {
    // Obtenez l'index de l'onglet actuel
    int currentIndex = tabbedMenu.getSelectedIndex();
    // Calculez l'index de l'onglet suivant
    int nextIndex = (currentIndex + 1) % tabbedMenu.getTabCount();
    // Sélectionnez l'onglet suivant
    tabbedMenu.setSelectedIndex(nextIndex);
});

// Création d'un nouveau rdv
btnSaveRdv.addActionListener(e -> { saveRdv(); });

// Supprimer un rendez-vous
btnSupRdv.addActionListener(e -> { supprimerRdv(); });

// Événement au clique du bouton pour faire apparaître la liste des
contacts
btnConnu.addActionListener(e -> { panelTableContactRdv.setVisible(true); });

// Événement au clique du bouton pour se diriger vers l'onglet "ajouter
contact"
btnInconnu.addActionListener(e -> {
    panelTableContactRdv.setVisible(false);
    // Obtenez l'index de l'onglet actuel
    int currentIndex = tabbedMenu.getSelectedIndex();
    // Calculez l'index de l'onglet suivant
    int nextIndex = (currentIndex - 3) % tabbedMenu.getTabCount();
    // Sélectionnez l'onglet suivant
    tabbedMenu.setSelectedIndex(nextIndex);

    // HEURE
    String heureRdv = labelHeure.getText();
    this.infoPartage.setHeureRdv(heureRdv);
    String heureContact = this.infoPartage.getHeureRdv();
    labelHeureRdv.setText(heureContact);

    // DATE
    String dateRdv = labelDate.getText();
    this.infoPartage.setDateRdv(dateRdv);
    String dateContact = this.infoPartage.getDateRdv();
    labelDateRdv.setText(dateContact);

    panelRdv.setVisible(true);
});
```

```
// Evénement au choix d'une date sélectionné mise à jour des tableaux
Agenda et Contact
dateChoix.addPropertyChangeListener(new PropertyChangeListener() {
    @Override
    public void propertyChange(PropertyChangeEvent evt) {

        Date selectedDate = dateChoix.getDate();
        if(selectedDate == null) {
            labDateChoix.setText("");
            return;
        }
        DateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
        String formattedDate = dateFormat.format(selectedDate);
        labDateChoix.setText(formattedDate);

        List<Agenda> agenda = reyService.ChercheDate(formattedDate);

        List<Agenda> validAgenda = new ArrayList<>();
        for (Agenda rdv : agenda) {
            Contact contact = rdv.getContact();
            if (contact != null) {
                validAgenda.add(rdv);
            }
        }

        AgendaTableModel tableModelAgenda = (AgendaTableModel)
            tableRdvAgenda.getModel();
        tableModelAgenda.setAgenda(validAgenda);
        tableModelAgenda.fireTableDataChanged();

        List<Contact> contacts = new ArrayList<>();

        // Parcourez la liste des rendez-vous et ajoutez chaque contact
        // unique à la liste des contacts
        for(Agenda rdv : agenda) {
            Contact contact = rdv.getContact();
            if(contact != null && !contacts.contains(contact)) {
                contacts.add(contact);
            }
        }
        // Mettre à jour le modèle de tableau avec les données des contacts
        ContactTableModel tableModelContact = (ContactTableModel)
            tableContactRdv.getModel();
        tableModelContact.setContacts(contacts);
        tableModelContact.fireTableDataChanged();
        textDescrRdv.setText("");
    }
});
```

```
// Evènement pour mettre la date dans le label
dateChooser.addPropertyChangeListener(new PropertyChangeListener() {
    @Override
    public void propertyChange(PropertyChangeEvent evt) {
        if (evt.getPropertyName().equals("date")) {
            // Récupérez la date sélectionnée avec le JDateChooser
            Date date = dateChooser.getDate();
            if(date == null) {
                labelDate.setText("");
                return;
            }
            // Formatez la date en utilisant le format par défaut
            SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-
                yyyy");
            String formattedDate = dateFormat.format(date);

            // Mise à jour du label avec la date formatée
            labelDate.setText(formattedDate);
        }
        labelHeure.setText("");
    }
});

// Evènement à l'affichage de la date les boutons se désactive si l'heure
// est déjà prise
labelDate.addPropertyChangeListener(new PropertyChangeListener() {
    @Override
    public void propertyChange(PropertyChangeEvent evt) {
        String date = labelDate.getText();
        if (date.isEmpty()){
            panelHeure.setVisible(false);
            panelBtnHeure.setVisible(false);
        }else {
            panelHeure.setVisible(true);
            panelBtnHeure.setVisible(true);
            panelInfo.setVisible(true);
        }
        choixHeure();
    }
});

// Evènement a l'affichage de l'heure les boutons pour sélectionner le
// contact s'affiche
labelHeure.addPropertyChangeListener(new PropertyChangeListener() {
    @Override
    public void propertyChange(PropertyChangeEvent evt) {
        String heure = labelHeure.getText();
        if (heure.isEmpty()){
            panelLabelContact.setVisible(false);
            panelBtnContact.setVisible(false);
        }else {
            panelLabelContact.setVisible(true);
            panelBtnContact.setVisible(true);
        }
    }
});
```

```
// Modifier la date
dateChoixModifRdv.addPropertyChangeListener(new PropertyChangeListener() {
    @Override
    public void propertyChange(PropertyChangeEvent evt) {
        if (evt.getPropertyName().equals("date")) {
            // Récupérez la date sélectionnée avec le JDateChooser
            Date date = dateChoixModifRdv.getDate();
            if (date == null) {
                LabelDateModif.setText("");
                return;
            }
            // Formatez la date en utilisant le format par défaut
            SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-
                yyyy");
            String formattedDate = dateFormat.format(date);

            // Mise à jour du label avec la date formatée
            LabelDateModif.setText(formattedDate);
        }
        LabelHeureModif.setText("");
    }
});

// Sélectionner un contact existant dans la base
tableAjoutRdvContact.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        rdvContact();
    }
});

// Evénement du choix du rendez-vous pour voir le contact
tableRdvAgenda.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        rdvAgenda();
    }
});

// Evénement pour afficher la description du contact du rendez-vous
sélectionné
tableContactRdv.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {afficheDescrContactRdv();}
});

// Evénement pour bloquer le créneau horaire de ce jour
btnBloquerRdv.addActionListener(e -> {bloqueCreneau(); });

// Modifier le rendez-vous sélectionné
BtnModifRdv.addActionListener(e -> {modifierRdv();});
```

```
// Direction vers l'onglet "Modifier rendez-vous"
btnAgendaModifRdv.addActionListener(e -> {
    int ligneSelectionnee = tableRdvAgenda.getSelectedRow();
    AgendaTableModel tableModel = (AgendaTableModel)
        tableRdvAgenda.getModel();
    if (ligneSelectionnee == -1) {
        JOptionPane.showMessageDialog(tabbedMenu, "Veuillez sélectionner un
            rendez-vous à modifier.");
    } else {
        Agenda agenda = tableModel.getAgenda().get(ligneSelectionnee);
        labelIdRdv.setText(agenda.getIdCal().toString());
        labelNomModif.setText(agenda.getContact().getNom());
        labelPrenomModif.setText(agenda.getContact().getPrenom());
        labelEmailModif.setText(agenda.getContact().getEmail());
        labelPhoneModif.setText(agenda.getContact().getPhone());

        // Obtenez l'index de l'onglet actuel
        int currentIndex = tabbedMenu.getSelectedIndex();
        // Calculez l'index de l'onglet suivant
        int nextIndex = (currentIndex + 2) % tabbedMenu.getTabCount();
        // Sélectionnez l'onglet suivant
        tabbedMenu.setSelectedIndex(nextIndex);
    }
});

// Modification du label de la date
labelDateModif.addPropertyChangeListener(new PropertyChangeListener() {
    @Override
    public void propertyChange(PropertyChangeEvent evt) {
        choixHeureModif();
    }
});

// Événement au clique du bouton pour insérer l'heure dans un label
h1.addActionListener(e -> { labelHeure.setText("06H00"); });
h2.addActionListener(e -> { labelHeure.setText("07H00"); });
h3.addActionListener(e -> { labelHeure.setText("08H00"); });
h4.addActionListener(e -> { labelHeure.setText("09H00"); });
h5.addActionListener(e -> { labelHeure.setText("10H00"); });
h6.addActionListener(e -> { labelHeure.setText("11H00"); });
h7.addActionListener(e -> { labelHeure.setText("12H00"); });
h8.addActionListener(e -> { labelHeure.setText("13H00"); });
h9.addActionListener(e -> { labelHeure.setText("14H00"); });
h10.addActionListener(e -> { labelHeure.setText("15H00"); });
h11.addActionListener(e -> { labelHeure.setText("16H00"); });
h12.addActionListener(e -> { labelHeure.setText("17H00"); });
h13.addActionListener(e -> { labelHeure.setText("18H00"); });
h14.addActionListener(e -> { labelHeure.setText("19H00"); });
h15.addActionListener(e -> { labelHeure.setText("20H00"); });
h16.addActionListener(e -> { labelHeure.setText("21H00"); });
h17.addActionListener(e -> { labelHeure.setText("22H00"); });
h18.addActionListener(e -> { labelHeure.setText("23H00"); });
```

```
// Evénement au clique du bouton pour modifier l'heure dans un label
m1.addActionListener(e -> { LabelHeureModif.setText("06H00"); });
m2.addActionListener(e -> { LabelHeureModif.setText("07H00"); });
m3.addActionListener(e -> { LabelHeureModif.setText("08H00"); });
m4.addActionListener(e -> { LabelHeureModif.setText("09H00"); });
m5.addActionListener(e -> { LabelHeureModif.setText("10H00"); });
m6.addActionListener(e -> { LabelHeureModif.setText("11H00"); });
m7.addActionListener(e -> { LabelHeureModif.setText("12H00"); });
m8.addActionListener(e -> { LabelHeureModif.setText("13H00"); });
m9.addActionListener(e -> { LabelHeureModif.setText("14H00"); });
m10.addActionListener(e -> { LabelHeureModif.setText("15H00"); });
m11.addActionListener(e -> { LabelHeureModif.setText("16H00"); });
m12.addActionListener(e -> { LabelHeureModif.setText("17H00"); });
m13.addActionListener(e -> { LabelHeureModif.setText("18H00"); });
m14.addActionListener(e -> { LabelHeureModif.setText("19H00"); });
m15.addActionListener(e -> { LabelHeureModif.setText("20H00"); });
m16.addActionListener(e -> { LabelHeureModif.setText("21H00"); });
m17.addActionListener(e -> { LabelHeureModif.setText("22H00"); });
m18.addActionListener(e -> { LabelHeureModif.setText("23H00"); });
}

=====
GESTION DES ONGLETS DU MENU
=====

private void updateTable(int selectedIndex) {
    switch (selectedIndex) {
        case 0 -> {
            labelIdRdv.setText("");
            labelId.setText("");
            textChercher.setText("");
            List<Contact> contactContact = reyService.getAllContact();
            ContactTableModel contactModelContact = new
                ContactTableModel(contactContact);
            tableContactContact.setModel(contactModelContact);
            labelDescrContact.setText("");
            List<Agenda> agendasContact = reyService.getAllAgenda();
            AgendaTableModel agendaModelAgenda = new
                AgendaTableModel(agendasContact);
            tableRdvContact.setModel(agendaModelAgenda);
        }
        case 1 -> {
            labelIdRdv.setText("");
            labelId.setText("");
            panelRdv.setVisible(false);
            panelRadioSexe.setVisible(true);
            labelId.setText("");
            GroupTypes.add(radioBtnEntreprise);
            GroupTypes.add(radioBtnParticulier);
            GroupTypes.clearSelection();
            GroupSexe.add(radioBtnFemme);
            GroupSexe.add(radioBtnHomme);
            GroupSexe.clearSelection();
            textNom.setText(null);
            textPrenom.setText("");
            textPhone.setText("");
            editorDescr.setText("");
            textEmail.setText("");
        }
    }
}
```

```

case 2 -> {
    labelIdRdv.setText("");
    if (labelId.getText().isEmpty()) {
        JOptionPane.showMessageDialog(tabbedMenu,
            "Veuillez sélectionner un contact à modifier.");
        // Obtenez l'index de l'onglet actuel
        int currentIndex = tabbedMenu.getSelectedIndex();
        // Calculez l'index de l'onglet suivant
        int nextIndex = (currentIndex - 2) % tabbedMenu.getTabCount();
        // Sélectionnez l'onglet suivant
        tabbedMenu.setSelectedIndex(nextIndex);
    }
}
case 3 -> {
    labelIdRdv.setText("");
    labelId.setText("");
    dateChoix.setDateFormatString("dd-MM-yyyy");
    panelDateChoix.add(dateChoix);
    dateChoix.setDate(null);
    List<Contact> contactsAgenda = reyService.getAllContact();
    ContactTableModel contactModel = new
        ContactTableModel(contactsAgenda);
    tableContactRdv.setModel(contactModel);
    labelDescrContact.setText("");
    List<Agenda> agendasAgenda = reyService.getAllAgenda();
    AgendaTableModel agendaModel = new AgendaTableModel(agendasAgenda);
    tableRdvAgenda.setModel(agendaModel);
    textDescrRdv.setText("");
}
case 4 -> {
    labelIdRdv.setText("");
    labelId.setText("");
    dateChooser.setDateFormatString("dd-MM-yyyy");
    panelDateChoose.add(dateChooser);
    labelDate.setText("");
    labelHeure.setText("");
    labelContact.setText("");
    panelInfo.setVisible(false);
    panelTableContactRdv.setVisible(false);
    dateChooser.setDate(null);
}
case 5 -> {
    labelId.setText("");
    if (labelIdRdv.getText().isEmpty()) {
        JOptionPane.showMessageDialog(tabbedMenu,
            "Veuillez sélectionner un rendez-vous à modifier.");
        // Obtenez l'index de l'onglet actuel
        int currentIndex = tabbedMenu.getSelectedIndex();
        // Calculez l'index de l'onglet suivant
        int nextIndex = (currentIndex - 2) % tabbedMenu.getTabCount();
        // Sélectionnez l'onglet suivant
        tabbedMenu.setSelectedIndex(nextIndex);
    }
    dateChoixModifRdv.setDateFormatString("dd-MM-yyyy");
    panelDateModif.add(dateChoixModifRdv);
    dateChoixModifRdv.setDate(null);
}
case 6 -> {
    labelIdRdv.setText("");
    labelId.setText("");
}
}
}

```

```
=====
CONTACT
=====

// ENREGISTRER UN NOUVEAU CONTACT
private void saveContact() {
    // Identifier les inputs du contact
    String nom = textNom.getText();
    String prenom = textPrenom.getText();
    String email = textEmail.getText();
    String phone = textPhone.getText();
    String description = editorDescr.getText();

    // Identifier les éléments du contact de la base de donnée
    Contact contact = new Contact();
    contact.setNom(nom);
    contact.setPrenom(prenom);
    contact.setEmail(email);
    contact.setPhone(phone);
    contact.setDescription(description);

    // Identifier les inputs de l'agenda
    String date = labelDateRdv.getText();
    String heure = labelHeureRdv.getText();

    // Identifier les éléments de l'agenda de la base de donnée
    Agenda agenda = new Agenda();
    agenda.setContact(contact);
    agenda.setDateRdv(date);
    agenda.setHeure(heure);

    // Vérifier si le bouton Entreprise est sélectionné et Vérifier si les
    // champs visibles ne sont pas vide
    if (radioBtnEntreprise.isSelected()) {
        contact.setTypes(EnumTypes.ENTREPRISE);
        if (nom.isEmpty() || email.isEmpty() || phone.isEmpty() ||
            description.isEmpty()) {
            JOptionPane.showMessageDialog(null, "Veuillez remplir tous les
                champs !", "Erreur", JOptionPane.ERROR_MESSAGE);
            return;
        }
    } else if (radioBtnParticulier.isSelected()) {
        contact.setTypes(EnumTypes.PARTICULIER);
        if (nom.isEmpty() || prenom.isEmpty() || email.isEmpty() ||
            phone.isEmpty() || description.isEmpty()) {
            JOptionPane.showMessageDialog(null, "Veuillez remplir tous les
                champs !", "Erreur", JOptionPane.ERROR_MESSAGE);
            return;
        }
    } else {
        // Afficher une alerte si aucun n'est sélectionné
        JOptionPane.showMessageDialog(this, "Veuillez sélectionner un type de
            contact.", "Erreur", JOptionPane.ERROR_MESSAGE);
        return;
    }

    // Vérifiez le format de l'adresse email
    String emailRegex = "^[a-zA-Z0-9_+&*-]+(?:\\." +
        "[a-zA-Z0-9_+&*-]+)*@" +
        "(?:[a-zA-Z0-9-]+\\.)+[a-z" +
        "A-Z]{2,7}$";
    Pattern emailPattern = Pattern.compile(emailRegex);
    if (!emailPattern.matcher(email).matches()) {
        JOptionPane.showMessageDialog(null, "L'adresse email est incorrecte !",
            "Erreur", JOptionPane.ERROR_MESSAGE);
        return;
    }
}
```



```
// Vérifiez le format du numéro de téléphone
String phoneRegex = "^\\(?([0-9]{3})\\)?[-.\\s]?([0-9]{3})[-.\\s]?([0-9]{4})$";
Pattern phonePattern = Pattern.compile(phoneRegex);
if (!phonePattern.matcher(phone).matches()) {
    JOptionPane.showMessageDialog(null, "Le numéro de téléphone est incorrect !", "Erreur", JOptionPane.ERROR_MESSAGE);
    return;
}

// Vérifier si les sexes sont visibles et Enregistrement du choix du sexe
if (panelRadioSexe.isVisible()){
    if (radioBtnFemme.isSelected()) {
        contact.setSexe(EnumSexe.FEMME);
    } else if (radioBtnHomme.isSelected()) {
        contact.setSexe(EnumSexe.HOMME);
    } else {
        // Afficher une alerte si aucun n'est sélectionné
        JOptionPane.showMessageDialog(this, "Veuillez sélectionner le sexe du contact.", "Erreur", JOptionPane.ERROR_MESSAGE);
        return;
    }
} else if (!panelRadioSexe.isVisible()){
    contact.setSexe(null);
}

// Vérifier si l'adresse email est déjà présente dans la base de donnée
Contact emailExist = reyService.getEmail(email);
if (emailExist != null){
    JOptionPane.showMessageDialog(null, "Cette adresse email est existe déjà !", "Erreur", JOptionPane.ERROR_MESSAGE);
    textEmail.setText("");
    return;
}

if (!panelRdv.isVisible()){
    // Enregistrement des éléments du contact dans la base de donnée
    reyService.addNewContact(contact);
} else if (panelRdv.isVisible()){
    // Enregistrement des éléments du contact et du rendez-vous dans la base de donnée
    reyService.addNewContact(contact);
    reyService.addNewRdv(contact.getIdContact(), agenda);
}

JOptionPane.showMessageDialog(this, "Contact sauvegardé avec succès.");

// Vider les composants
textNom.setText("");
textPrenom.setText("");
textEmail.setText("");
textPhone.setText("");
editorDescr.setText("");
panelRdv.setVisible(false);
GroupTypes.add(radioBtnEntreprise);
GroupTypes.add(radioBtnParticulier);
GroupTypes.clearSelection();
GroupSexe.add(radioBtnFemme);
GroupSexe.add(radioBtnHomme);
GroupSexe.clearSelection();
```

```
// Obtenez l'index de l'onglet actuel
int currentIndex = tabbedMenu.getSelectedIndex();
// Calculez l'index de l'onglet suivant
int nextIndex = (currentIndex - 1) % tabbedMenu.getTabCount();
// Sélectionnez l'onglet suivant
tabbedMenu.setSelectedIndex(nextIndex);

}

// CHERCHER UN CONTACT AVEC SON NOM
private void chercherContact() {
    String nom = textChercher.getText();
    List<Contact> contacts = reyService.chercherNom(nom);
    ContactTableModel tableModel = (ContactTableModel)
        tableContactContact.getModel();
    tableModel.setContacts(contacts);
    tableModel.fireTableDataChanged();
    labelDescrContact.setText("");
}

// SUPPRIMER UN CONTACT
private void supprimerContact() {
    int ligneSelectionnee = tableContactContact.getSelectedRow();
    ContactTableModel tableModel = (ContactTableModel)
        tableContactContact.getModel();
    if (ligneSelectionnee == -1) {
        JOptionPane.showMessageDialog(tabbedMenu, "Veuillez sélectionner un
            contact à supprimer.");
    } else {
        if (ligneSelectionnee >= 0 && ligneSelectionnee <
            tableModel.getContacts().size()) {
            Long id_contact = tableModel.getIdContact(ligneSelectionnee);
            reyService.deleteContact(id_contact);
            displayContacts();
            List<Agenda> agendasContact = reyService.getAllAgenda();
            AgendaTableModel agendaModelAgenda = new
                AgendaTableModel(agendasContact);
            tableRdvContact.setModel(agendaModelAgenda);
        }
        JOptionPane.showMessageDialog(this, "Contact supprimé avec succès.");
        labelDescrContact.setText("");
        tableRdvContact.repaint();
    }
}

// METTRE A JOUR LE TABLEAU DES CONTACTS
private void displayContacts() {
    List<Contact> contacts = reyService.getAllContact();
    ContactTableModel model = new ContactTableModel(contacts);
    tableContactContact.setModel(model);
    tableContactRdv.setModel(model);
    tableAjoutRdvContact.setModel(model);
}
}
```

```
// MODIFIER LE CONTACT SELECTIONNE
private void modifierContact() {
    // Identifier les inputs du contact
    String nom = textNomModif.getText();
    String prenom = textPrenomModif.getText();
    String email = textEmailModif.getText();
    String phone = textPhoneModif.getText();
    String description = editorDescrModif.getText();

    int ligneSelectionnee = tableContactContact.getSelectedRow();
    ContactTableModel tableModel = (ContactTableModel)
        tableContactContact.getModel();
    if (ligneSelectionnee == -1) {
        JOptionPane.showMessageDialog(this, "Veuillez sélectionner un contact à
            modifier.");
    } else {
        Long id = tableModel.getId_Contact(ligneSelectionnee);
        Contact updatedContact = new Contact();
        updatedContact.setNom(textNomModif.getText());
        updatedContact.setPrenom(textPrenomModif.getText());
        updatedContact.setDescription(editorDescrModif.getText());
        updatedContact.setEmail(textEmailModif.getText());
        updatedContact.setPhone(textPhoneModif.getText());

        // Vérifier si le bouton Entreprise est sélectionné et Vérifier si les
        // champs visibles ne sont pas vide
        if (radioBtnEntrepriseModif.isSelected()) {
            updatedContact.setTypes(EnumTypes.ENTREPRISE);
            if (nom.isEmpty() || email.isEmpty() || phone.isEmpty() ||
                description.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Veuillez remplir tous les
                    champs !", "Erreur", JOptionPane.ERROR_MESSAGE);
                return;
            }
        } else if (radioBtnParticulierModif.isSelected()) {
            updatedContact.setTypes(EnumTypes.PARTICULIER);
            if (nom.isEmpty() || prenom.isEmpty() || email.isEmpty() ||
                phone.isEmpty() || description.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Veuillez remplir tous les
                    champs !", "Erreur", JOptionPane.ERROR_MESSAGE);
                return;
            }
        } else {
            // Afficher une alerte si aucun n'est sélectionné
            JOptionPane.showMessageDialog(this, "Veuillez sélectionner un type
                de contact.", "Erreur", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Vérifiez le format de l'adresse email
        String emailRegex = "^[a-zA-Z0-9_+&*-]+(?:\\." +
            "[a-zA-Z0-9_+&*-]+)*@" +
            "(?:[a-zA-Z0-9-]+\\." +
            "[a-Z]{2,7})$";
        Pattern emailPattern = Pattern.compile(emailRegex);
        if (!emailPattern.matcher(email).matches()) {
            JOptionPane.showMessageDialog(null, "L'adresse email est incorrecte
                !", "Erreur", JOptionPane.ERROR_MESSAGE);
            return;
        }
    }
}
```

```
// Vérifiez le format du numéro de téléphone
String phoneRegex = "^\\(?([0-9]{3})\\)?[-\\.\\s]?([0-9]{3})[-\\.\\s]?([0-9]{4})$";
Pattern phonePattern = Pattern.compile(phoneRegex);
if (!phonePattern.matcher(phone).matches()) {
    JOptionPane.showMessageDialog(null, "Le numéro de téléphone est incorrect !", "Erreur", JOptionPane.ERROR_MESSAGE);
    return;
}

// Vérifier si les sexes sont visibles et Enregistrement du choix du sexe
if (panelRadioSexeModif.isVisible()){
    if (radioBtnFemmeModif.isSelected()) {
        updatedContact.setSexe(EnumSexe.FEMME);
    } else if (radioBtnHommeModif.isSelected()) {
        updatedContact.setSexe(EnumSexe.HOMME);
    } else {
        // Afficher une alerte si aucun n'est sélectionné
        JOptionPane.showMessageDialog(this, "Veuillez sélectionner le sexe du contact.", "Erreur", JOptionPane.ERROR_MESSAGE);
        return;
    }
} else if (!panelRadioSexeModif.isVisible()){
    updatedContact.setSexe(null);
}

// Sauvegarder les modifications du contact
Contact contact = reyService.updateContact(id, updatedContact);
// Mettre à jour la ligne de la table correspondant au contact modifié
tableModel.updateRow(ligneSelectionnee, contact);
// Afficher un message de confirmation
JOptionPane.showMessageDialog(this, "Contact modifié avec succès.");
labelId.setText("");
GroupTypes.add(radioBtnEntrepriseModif);
GroupTypes.add(radioBtnParticulierModif);
GroupTypes.clearSelection();
GroupSexe.add(radioBtnFemmeModif);
GroupSexe.add(radioBtnHommeModif);
GroupSexe.clearSelection();
textNomModif.setText("");
textPrenomModif.setText("");
textPhoneModif.setText("");
editorDescrModif.setText("");
textEmailModif.setText("");

// Obtenez l'index de l'onglet actuel
int currentIndex = tabbedMenu.getSelectedIndex();
// Calculez l'index de l'onglet suivant
int nextIndex = (currentIndex - 2) % tabbedMenu.getTabCount();
// Sélectionnez l'onglet suivant
tabbedMenu.setSelectedIndex(nextIndex);
}
}
```

```
// AFFICHER LA DESCRIPTION DU CONTACT SELECTION
private void voirDescription() {
    // Récupération de la ligne sélectionnée dans la table
    int ligneSelectionnee = tableContactContact.getSelectedRow();

    // Récupération du ContactTableModel associé à la table
    ContactTableModel tableModel = (ContactTableModel)
        tableContactContact.getModel();
    AgendaTableModel agendaTableModel = (AgendaTableModel)
        tableRdvContact.getModel();

    // Vérification que l'index est valide (compris entre 0 et la taille de la
        liste moins 1)
    if (ligneSelectionnee >= 0 && ligneSelectionnee <
        tableModel.getContacts().size()) {
        // Utilisation de la méthode getDescription du ContactTableModel pour
            récupérer la description
        String description = tableModel.getDescription(ligneSelectionnee);
        // Utilisation de la méthode setText pour afficher la description dans
            le label textDescription
        labelDescrContact.setText(description);

        Long id_contact = tableModel.getId_Contact(ligneSelectionnee);

        List<Agenda> agenda = reyService.afficheRdvDuContact(id_contact);

        agendaTableModel.setAgenda(agenda);

        tableRdvContact.repaint();
    }
}
```

```
=====
                                AGENDA
=====

// SAUVEGARDER UN RENDEZ VOUS
private void saveRdv() {
    String date = labelDate.getText();
    String heure = labelHeure.getText();
    String labContact = labelContact.getText();

    if (date.isEmpty()){
        JOptionPane.showMessageDialog(null, "Veuillez sélectionner un date !",
            "Erreur", JOptionPane.ERROR_MESSAGE);
        return;
    }
    if (heure.isEmpty()){
        JOptionPane.showMessageDialog(null, "Veuillez sélectionner une heure
        disponible !", "Erreur", JOptionPane.ERROR_MESSAGE);
        return;
    }
    if (labContact.isEmpty()){
        JOptionPane.showMessageDialog(null, "Veuillez sélectionner un contact
        dans vos contact ou enregistrer un nouveau contact !",
            "Erreur", JOptionPane.ERROR_MESSAGE);
        return;
    }
}

int ligneSelectionnee = tableAjoutRdvContact.getSelectedRow();
ContactTableModel tableModel = (ContactTableModel)
    tableAjoutRdvContact.getModel();
Long id_contact = tableModel.getId_Contact(ligneSelectionnee);
Contact contact = reyService.getIdContact(id_contact);

Agenda agenda = new Agenda();
agenda.setDateRdv(date);
agenda.setHeure(heure);
agenda.setContact(contact);

reyService.addNewRdv(contact.getIdContact(), agenda);

JOptionPane.showMessageDialog(this, "Rendez-vous enregistré avec succès.");

labelDate.setText("");
labelHeure.setText("");
labelContact.setText("");
panelInfo.setVisible(false);
panelTableContactRdv.setVisible(false);

// Obtenez l'index de l'onglet actuel
int currentIndex = tabbedMenu.getSelectedIndex();
// Calculez l'index de l'onglet suivant
int nextIndex = (currentIndex - 1) % tabbedMenu.getTabCount();
// Sélectionnez l'onglet suivant
tabbedMenu.setSelectedIndex(nextIndex);
}
```

```
// SUPPRIMER UN RENDEZ VOUS
private void supprimerRdv() {
    // Récupération de la ligne sélectionnée dans la table
    int ligneSelectionnee = tableRdvAgenda.getSelectedRow();
    // Récupération de AgendaTableModel associé à la table
    AgendaTableModel tableModel = (AgendaTableModel) tableRdvAgenda.getModel();

    if (ligneSelectionnee == -1) {
        JOptionPane.showMessageDialog(tabbedMenu, "Veuillez sélectionner un
            rendez-vous à supprimer.");
    } else {
        // Vérification que l'index est valide (compris entre 0 et la taille de
            la liste moins 1)
        if (ligneSelectionnee >= 0 && ligneSelectionnee <
            tableModel.getAgenda().size()) {
            // Utilisation de la méthode getIdRdv de AgendaTableModel pour
                récupérer l'ID du rdv
            Long id = tableModel.getIdRdv(ligneSelectionnee);
            // Utilisation de la méthode delete pour supprimer le rdv
            reyService.deleteRdv(id);
            // Mise à jour du tableau
            displayAgenda();
            textDescrRdv.setText("");
        }
        JOptionPane.showMessageDialog(this, "Rendez-vous supprimé avec
            succès.");
    }
}

// MISE A JOUR DES TABLEAUX DES AGENDAS
private void displayAgenda() {
    List<Agenda> agenda = reyService.getAllAgenda();
    AgendaTableModel model = new AgendaTableModel(agenda);
    tableRdvAgenda.setModel(model);
}
```

```
// BOUTON D'HEURE SAVE POUR DESACTIVE SI DEJA PRIS
private void setButtonState(JButton button, String heure) {
    String date = labelDate.getText();
    List<Agenda> agenda = reyService.chercherDateHeure(date, heure);
    boolean hasContact = false;
    boolean isTaken = true;
    Agenda rdv = null;
    for(Agenda a:agenda) {
        if(a.getContact() != null) {
            hasContact = true;
            rdv = a;
            break;
        }
    }
    if (agenda.isEmpty()){
        isTaken = false;
    }
    if (!hasContact) {
        if (isTaken){
            button.setEnabled(false);
            button.setBackground(Color.RED);
        }
        else{
            button.setEnabled(true);
            button.setBackground(new Color(45, 146, 220));
        }
    } else {
        if (isTaken){
            button.setEnabled(false);
            button.setBackground(Color.GRAY);
        }
        else{
            button.setEnabled(true);
            button.setBackground(Color.RED);
        }
    }
}

// INITIALISATION DES BOUTONS D'HEURE SAVE
private void choixHeure() {
    setButtonState(h1, "06H00");
    setButtonState(h2, "07H00");
    setButtonState(h3, "08H00");
    setButtonState(h4, "09H00");
    setButtonState(h5, "10H00");
    setButtonState(h6, "11H00");
    setButtonState(h7, "12H00");
    setButtonState(h8, "13H00");
    setButtonState(h9, "14H00");
    setButtonState(h10, "15H00");
    setButtonState(h11, "16H00");
    setButtonState(h12, "17H00");
    setButtonState(h13, "18H00");
    setButtonState(h14, "19H00");
    setButtonState(h15, "20H00");
    setButtonState(h16, "21H00");
    setButtonState(h17, "22H00");
    setButtonState(h18, "23H00");
}
}
```



```
// SELECTIONNER UN CONTACT PAR SON ID
private void rdvContact() {
    // Récupération de la ligne sélectionnée dans la table
    int ligneSelectionnee = tableAjoutRdvContact.getSelectedRow();
    // Récupération du ContactTableModel associé à la table
    ContactTableModel tableModel = (ContactTableModel)
        tableAjoutRdvContact.getModel();
    // Vérification que l'index est valide (compris entre 0 et la taille de la
        liste moins 1)
    if (ligneSelectionnee >= 0 && ligneSelectionnee <
        tableModel.getContacts().size()) {
        // Utilisation de la méthode getIdContact du ContactTableModel pour
            récupérer l'ID du contact
        Long id_contact = tableModel.getId_Contact(ligneSelectionnee);
        Contact contact = reyService.getContactId(id_contact);
        String nom = contact.getNom();
        labelContact.setText(nom);
    }
}

// VOIR LE CONTACT DU RENDEZ-VOUS SELECTIONNE
private void rdvAgenda() {
    // Récupération de la ligne sélectionnée dans la table
    int ligneSelectionnee = tableRdvAgenda.getSelectedRow();

    // Récupération du ContactTableModel associé à la table
    AgendaTableModel agendaTableModel = (AgendaTableModel)
        tableRdvAgenda.getModel();
    ContactTableModel tableModel = (ContactTableModel)
        tableContactRdv.getModel();

    // Vérification que l'index est valide (compris entre 0 et la taille de la
        liste moins 1)
    if (ligneSelectionnee >= 0 && ligneSelectionnee <
        agendaTableModel.getAgenda().size()) {
        // Récupération de l'ID de contact à partir de l'agenda
        Contact idContact =
            agendaTableModel.getAgenda().get(ligneSelectionnee).getContact();

        // Condition si le contact n'est pas null
        if(idContact != null) {
            // Mise à jour du modèle de table pour afficher le contact
                sélectionné
            tableModel.setContact(idContact);
            tableModel.fireTableDataChanged();
            // Effacer le texte
            textDescrRdv.setText("");
        } else {
            //Ajoutez votre logique pour gérer le cas où le contact est null
            textDescrRdv.setText("Créneau horaire bloqué");
            // Mise à jour du tableau contact
            displayContacts();
        }
    }
}
}
```

```
// AFFICHER LA DESCRIPTION DU CONTACT SELECTIONNE DU RDV
private void afficheDescrContactRdv(){
    // Récupération de la ligne sélectionnée dans la table
    int ligneSelectionnee = tableContactRdv.getSelectedRow();
    // Récupération du ContactTableModel associé à la table
    ContactTableModel tableModel = (ContactTableModel)
        tableContactRdv.getModel();
    // Vérification que l'index est valide (compris entre 0 et la taille de la
    // liste moins 1)
    if (ligneSelectionnee >= 0 && ligneSelectionnee <
        tableModel.getContacts().size()) {
        // Utilisation de la méthode getDescription du ContactTableModel pour
        // récupérer la description
        String description = tableModel.getDescription(ligneSelectionnee);
        // Utilisation de la méthode setText pour afficher la description dans
        // le label textDescription
        textDescrRdv.setText(description);
    }
}

// BLOQUER DES CRENEAUX SANS CONTACT
private void bloquerCreneau() {

    String date = labelDate.getText();
    String heure = labelHeure.getText();

    Agenda agenda = new Agenda();
    agenda.setDateRdv(date);
    agenda.setHeure(heure);

    reyService.bloquerHoraire(agenda);

    JOptionPane.showMessageDialog(this, "Créneau horaire bloqués avec
        succès.");
    // Obtenez l'index de l'onglet actuel
    int currentIndex = tabbedMenu.getSelectedIndex();
    // Calculez l'index de l'onglet suivant
    int nextIndex = (currentIndex - 1) % tabbedMenu.getTabCount();
    // Sélectionnez l'onglet suivant
    tabbedMenu.setSelectedIndex(nextIndex);
}

// MODIFIER LE RENDEZ-VOUS SELECTIONNE
private void modifierRdv(){
    int ligneSelectionnee = tableRdvAgenda.getSelectedRow();
    AgendaTableModel tableModel = (AgendaTableModel) tableRdvAgenda.getModel();
    Long id = tableModel.getIdRdv(ligneSelectionnee);
    Agenda updatedAgenda = new Agenda();
    updatedAgenda.setDateRdv(LabelDateModif.getText());
    updatedAgenda.setHeure(LabelHeureModif.getText());

    // Sauvegarder les modifications du contact
    Agenda agenda = reyService.updateRdv(id, updatedAgenda);
    // Mettre à jour la ligne de la table correspondant au contact modifié
    tableModel.updateRow(ligneSelectionnee, agenda);
    // Afficher un message de confirmation
    JOptionPane.showMessageDialog(this, "Rendez-vous modifié avec succès.");

    // Obtenez l'index de l'onglet actuel
    int currentIndex = tabbedMenu.getSelectedIndex();
    // Calculez l'index de l'onglet suivant
    int nextIndex = (currentIndex - 2) % tabbedMenu.getTabCount();
    // Sélectionnez l'onglet suivant
    tabbedMenu.setSelectedIndex(nextIndex);
}
```

```
// BOUTON D'HEURE MODIFIER POUR DESACTIVE SI DEJA PRIS
private void setButtonStateModif(JButton button, String heure) {
    String date = LabelDateModif.getText();
    List<Agenda> agenda = reyService.chercherDateHeure(date, heure);
    boolean hasContact = false;
    boolean isTaken = true;
    Agenda rdv = null;
    for(Agenda a:agenda) {
        if(a.getContact() != null) {
            hasContact = true;
            rdv = a;
            break;
        }
    }
    if (agenda.isEmpty()){
        isTaken = false;
    }
    if (!hasContact) {
        if (isTaken){
            button.setEnabled(false);
            button.setBackground(Color.RED);
        }
        else{
            button.setEnabled(true);
            button.setBackground(new Color(45, 146, 220));
        }
    } else {
        if (isTaken){
            button.setEnabled(false);
            button.setBackground(Color.GRAY);
        }
        else{
            button.setEnabled(true);
            button.setBackground(Color.RED);
        }
    }
}

// INITIALISATION DES BOUTONS D'HEURE MODIFIER
private void choixHeureModif(){
    setButtonStateModif(m1, "06H00");
    setButtonStateModif(m2, "07H00");
    setButtonStateModif(m3, "08H00");
    setButtonStateModif(m4, "09H00");
    setButtonStateModif(m5, "10H00");
    setButtonStateModif(m6, "11H00");
    setButtonStateModif(m7, "12H00");
    setButtonStateModif(m8, "13H00");
    setButtonStateModif(m9, "14H00");
    setButtonStateModif(m10, "15H00");
    setButtonStateModif(m11, "16H00");
    setButtonStateModif(m12, "17H00");
    setButtonStateModif(m13, "18H00");
    setButtonStateModif(m14, "19H00");
    setButtonStateModif(m15, "20H00");
    setButtonStateModif(m16, "21H00");
    setButtonStateModif(m17, "22H00");
    setButtonStateModif(m18, "23H00");
}
```

```
=====
                                MAIN
=====

public static void main(String[] args) {
    // CONNEXION A LA BASE DE DONNEES
    AnnotationConfigApplicationContext context = new
        AnnotationConfigApplicationContext(AppConfig.class);
    ReyVues app = context.getBean(ReyVues.class);

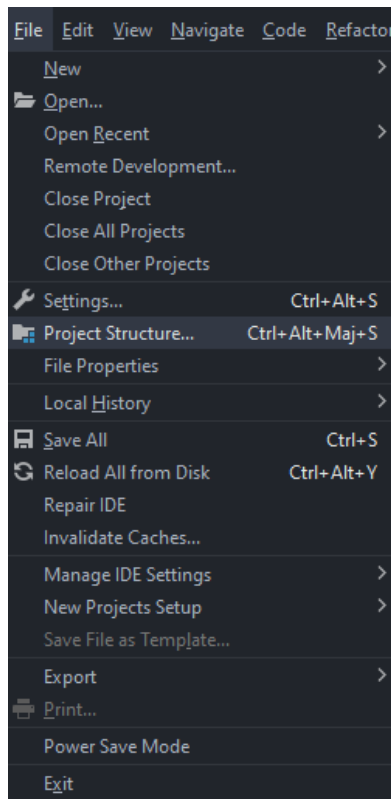
    // RECUPERER LA LISTE DES CONTACTS
    List<Contact> contacts = app.reyService.getAllContact();
    // INITIALISER LE MODEL DE TABLE AVEC LA LISTE DES CONTACTS
    ContactTableModel ContactTableModel = new ContactTableModel(contacts);
    app.tableContactContact.setModel(ContactTableModel);
    app.tableContactRdv.setModel(ContactTableModel);
    app.tableAjoutRdvContact.setModel(ContactTableModel);
    // RECUPERER LA LISTE DES RENDEZ-VOUS
    List<Agenda> agenda = app.reyService.getAllAgenda();
    // INITIALISER LE MODEL DE TABLE AVEC LA LISTE DES RENDEZ-VOUS
    AgendaTableModel agendaTableModel = new AgendaTableModel(agenda);
    app.tableRdvContact.setModel(agendaTableModel);
    app.tableRdvAgenda.setModel(agendaTableModel);

    JFrame frame = new JFrame("Rey");
    frame.pack();
    Dimension tailleEcran = Toolkit.getDefaultToolkit().getScreenSize();
    int height = tailleEcran.height;
    int width = tailleEcran.width;
    frame.setSize(width/2, height/2);
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
    frame.setContentPane(app.rootPanel);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

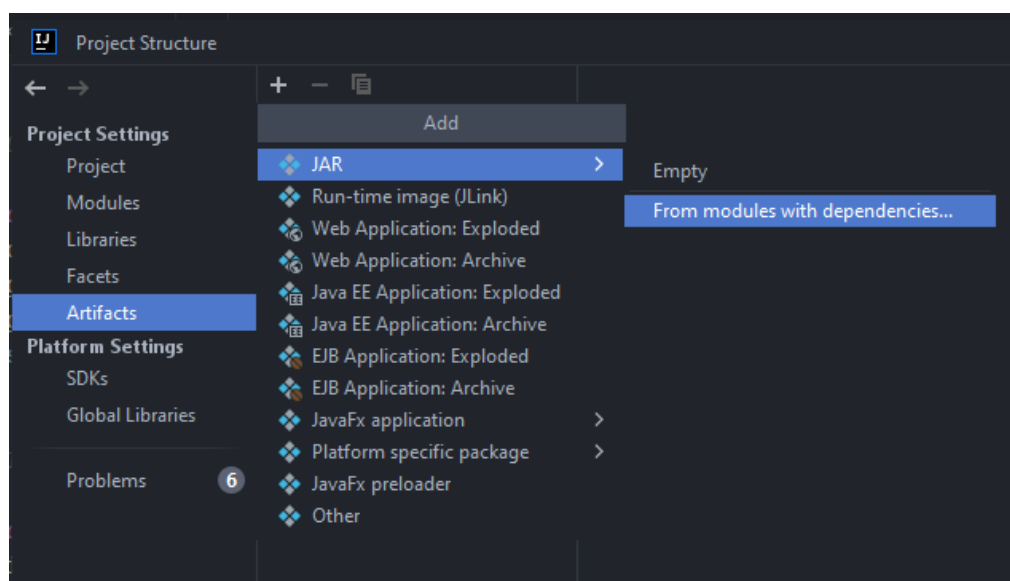
10. Déploiement

Pour effectuer le déploiement je commence, à l'aide d'IntelliJ, par créer un fichier JAR qui me servira d'exécutable.

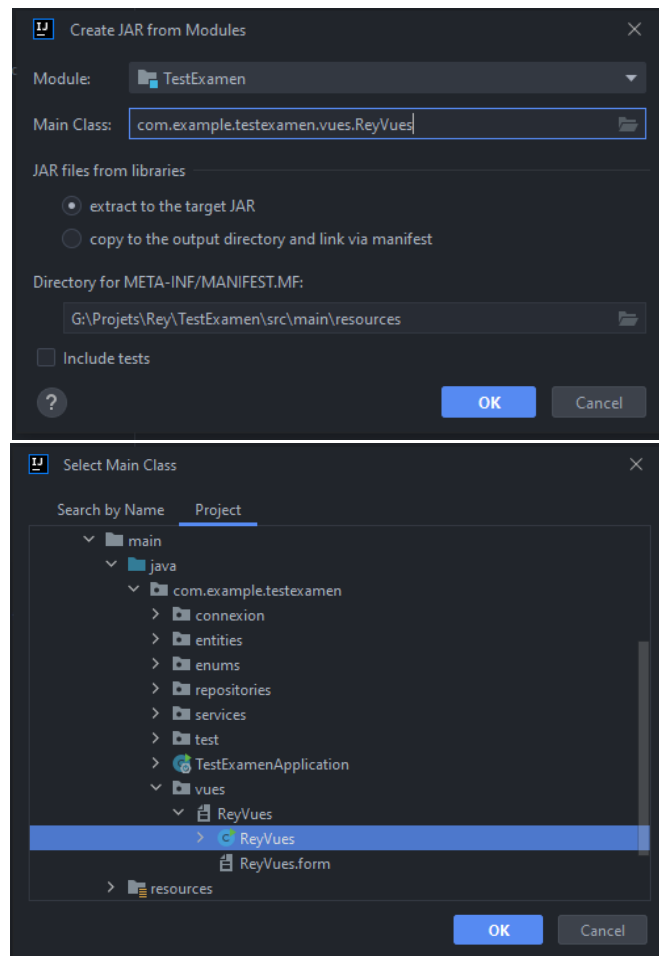
Pour cela je vais dans **File -> Project Structure...**



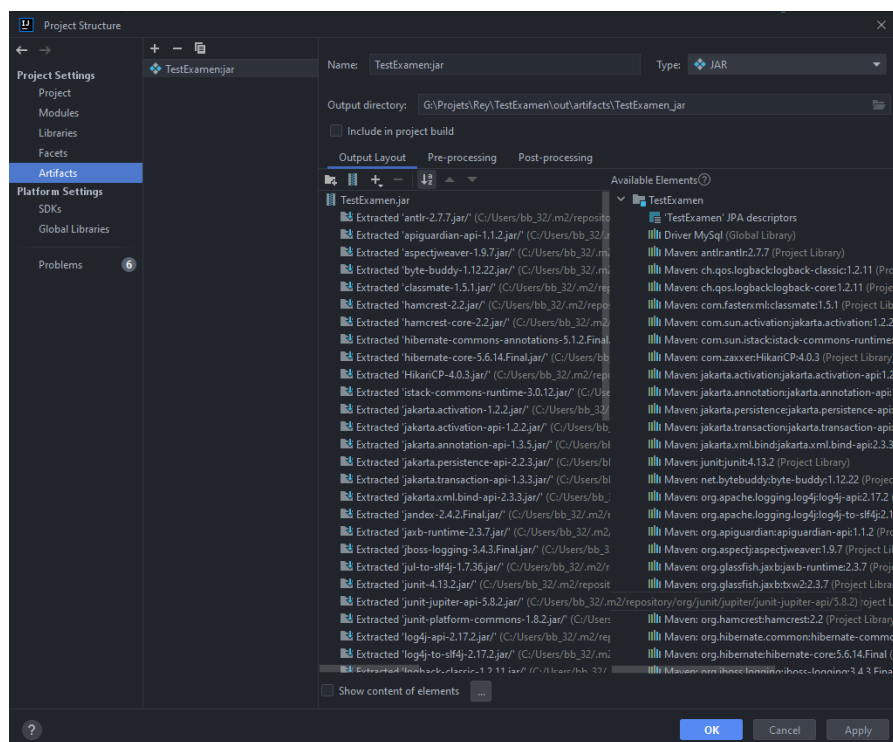
Ensuite dans **Artifacts** je clique sur le + et sélectionne **JAR -> From modules with dependencies...**



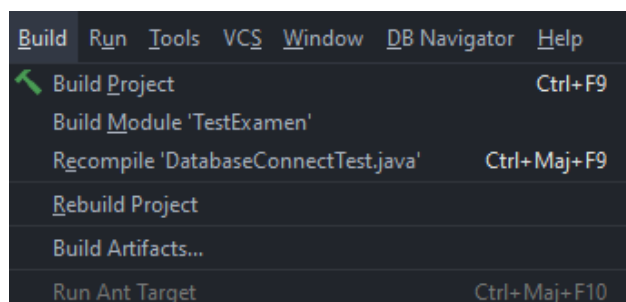
Ensuite, je sélectionne le point d'entrée de mon application c'est-à-dire là où elle se lance.



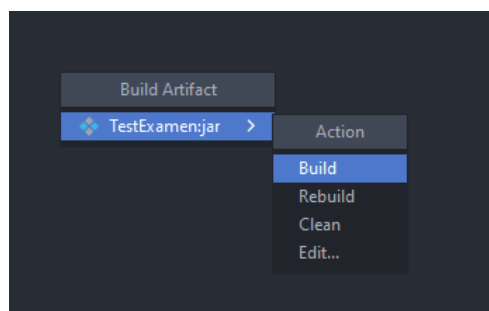
Ensuite je clique sur **Apply** et **OK**



Ensuite je vais sur **Build** -> Build Artifacts...




Dans **Action** je choisis **Build**



Il a créé un dossier **out** dans le dossier de mon projet

nom	modified	type	size
.idea	01/02/2023 09:56	Dossier de fichiers	
.mvn	01/02/2023 09:56	Dossier de fichiers	
out	03/02/2023 15:06	Dossier de fichiers	
src	01/02/2023 09:56	Dossier de fichiers	
target	01/02/2023 14:07	Dossier de fichiers	
.gitignore	01/02/2023 09:56	Fichier source Git I...	1 Ko
HELP.md	01/02/2023 09:56	Fichier source Mar...	2 Ko
mvnw	01/02/2023 09:56	Fichier	11 Ko
mvnw.cmd	01/02/2023 09:56	Script de comman...	7 Ko
pom.xml	09/02/2023 10:38	Document XML	3 Ko
ReyApp.iml	09/02/2023 10:39	Fichier IML	12 Ko

Dans ce dossier se trouve mon fichier JAR nouvellement créé.

 ReyApp.jar	03/02/2023 15:06	Executable Jar File	39 622 Ko
--	------------------	---------------------	-----------

Si je double clique dessus je constate que mon programme se lance correctement.

Rey

Contact

Ajout d'un contact

Modifier contact

Agenda

Ajout d'un rendez-vous

Modifier rendez-vous

Actualité

Rechercher un contact

Valider

Ajouter

Supprimer

Modifier

Nom	Prenom	Email	Téléphone	Type	Sexe
Petit	Pierre	pierre@gmail.com	0602030103	PARTICULIER	HOMME
Gibon	Flore	gibon@gmail.com	0720103022	PARTICULIER	FEMME

Description

Rendez-vous

Date	Heure
23-02-2023	19H00
23-02-2023	16H00

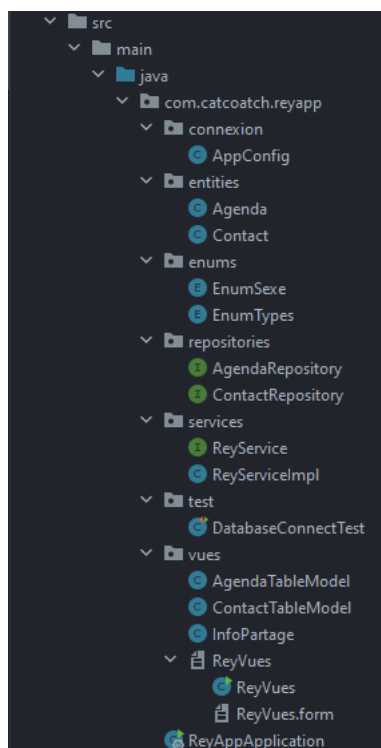
Par la suite il faudra mettre le fichier JAR sur un serveur qui faudra configurer, lancer, tester et distribuer.

11.Sécurité

La sécurité est une partie importante quand on crée un projet.

Il s'assure de la protection des données sensibles, il prévient des attaques et il garantit la disponibilité et la fiabilité du système.

Dans ce projet nous pouvons voir que la sécurité est :



Dans un premier temps : Assuré par la création des packages qui séparent les couches logiques et métiers. De plus en utilisant JPA il évite d'écrire les requêtes de base comme la création en utilisant 'save' comme nous pouvons le voir dans le package « repositories »

```
3 usages
@Repository
public interface ContactRepository extends JpaRepository<Contact, Long> {

    // CHERCHER UN CONTACT PAR SON NOM EN IGNORANT LA CASE
    1 usage
    List<Contact> findByNameContainingIgnoreCase(String nom);

    // CHERCHER SI ADRESSE MAIL EXISTE
    1 usage
    Contact findByEmail(String email);
}
```

Dans un deuxième temps : Le format de l'adresse email et du numéro de téléphone sont définis pour éviter l'entrée de données non conformes au format.

```
// Vérifiez le format de l'adresse email
String emailRegex = "^[a-zA-Z0-9_+&*~]+(?:\\.\\.+|"+
    "[a-zA-Z0-9_+&*~]+)*@" +
    "(?:[a-zA-Z0-9-]+\\.\\.)+[a-z]+"+
    "[A-Z]{2,7}$";
Pattern emailPattern = Pattern.compile(emailRegex);
if (!emailPattern.matcher(email).matches()) {
    JOptionPane.showMessageDialog( parentComponent: null,
        message: "L'adresse email est incorrecte !",
        title: "Erreur", JOptionPane.ERROR_MESSAGE);
    return;
}

// Vérifiez le format du numéro de téléphone
String phoneRegex = "^[\\(\\{[0-9]{3}\\}\\)?[-.\\s]?([0-9]{3})[-.\\s]?([0-9]{4})$";
Pattern phonePattern = Pattern.compile(phoneRegex);
if (!phonePattern.matcher(phone).matches()) {
    JOptionPane.showMessageDialog( parentComponent: null,
        message: "Le numéro de téléphone est incorrect !",
        title: "Erreur", JOptionPane.ERROR_MESSAGE);
    return;
}
```

Dans un troisième temps : Avant un nouvel envoi du formulaire, donc un enregistrement d'un nouveau contact, je m'assure que l'adresse mail n'existe pas déjà.

```
// Vérifiez le format de l'adresse email
String emailRegex = "[a-zA-Z0-9_+&*~]+(?:\\.\\.+|"+
    "[a-zA-Z0-9_+&*~]+)*@" +
    "(?:[a-zA-Z0-9-]+\\.\\.)+[a-z]+" +
    "A-Z]{2,7}$";
Pattern emailPattern = Pattern.compile(emailRegex);
if (!emailPattern.matcher(email).matches()) {
    JOptionPane.showMessageDialog( parentComponent: null,
        message: "L'adresse email est incorrecte !",
        title: "Erreur", JOptionPane.ERROR_MESSAGE);
    return;
}

// Vérifiez le format du numéro de téléphone
String phoneRegex = "^\\((?([0-9]{3})\\)?)?[-\\.\\s]?([0-9]{3})[-\\.\\s]?([0-9]{4})$";
Pattern phonePattern = Pattern.compile(phoneRegex);
if (!phonePattern.matcher(phone).matches()) {
    JOptionPane.showMessageDialog( parentComponent: null,
        message: "Le numéro de téléphone est incorrect !",
        title: "Erreur", JOptionPane.ERROR_MESSAGE);
    return;
}
```

```
@Column(name = "email", unique = true)
private String email;
```

Pour finir, je précise que certaines données ne doivent pas être vides ou nulles.

```
// Vérifier si le bouton Entreprise est sélectionné et Vérifier si les champs visibles ne sont pas vide
if (radioBtnEntreprise.isSelected()){
    contact.setTypes(EnumTypes.ENTREPRISE);
    if (nom.isEmpty() || email.isEmpty() || phone.isEmpty() || description.isEmpty()){
        JOptionPane.showMessageDialog( parentComponent: null, message: "Veuillez remplir tous les champs !",
            title: "Erreur", JOptionPane.ERROR_MESSAGE);
        return;
    }
} else if (radioBtnParticulier.isSelected()) {
    contact.setTypes(EnumTypes.PARTICULIER);
    if (nom.isEmpty() || prenom.isEmpty() || email.isEmpty() || phone.isEmpty() || description.isEmpty()){
        JOptionPane.showMessageDialog( parentComponent: null, message: "Veuillez remplir tous les champs !",
            title: "Erreur", JOptionPane.ERROR_MESSAGE);
        return;
    }
} else {
    // Afficher une alerte si aucun n'est sélectionné
    JOptionPane.showMessageDialog( parentComponent: this, message: "Veuillez sélectionner un type de contact.",
        title: "Erreur", JOptionPane.ERROR_MESSAGE);
    return;
}
```

12. Conclusion

Cette formation m'a permis d'en apprendre davantage sur la programmation, de combler certaines lacunes, mais aussi de me conforter dans l'idée que la base de mes connaissances étaient justes.

Ce qui me plaît le plus dans ce métier, c'est de toujours en apprendre plus, de chercher et trouver des solutions, d'être en constante évolution, mais aussi de pouvoir créer quelque chose et de le voir être utilisé par des personnes.

Je souhaite évoluer dans ce domaine pour :

- Trouver l'épanouissement professionnel dans cette branche.
- Pouvoir retrouver un poste comme j'ai déjà eu.
- Être face à de nouveaux défis.
- Multiplier les projets.

Et pour finir, si je devais améliorer ce projet je me pencherais sur la sécurité, le développement d'une application mobile, la possibilité de réserver un rendez-vous à plusieurs pour faire des réunions de groupe et enfin améliorer considérablement le design.