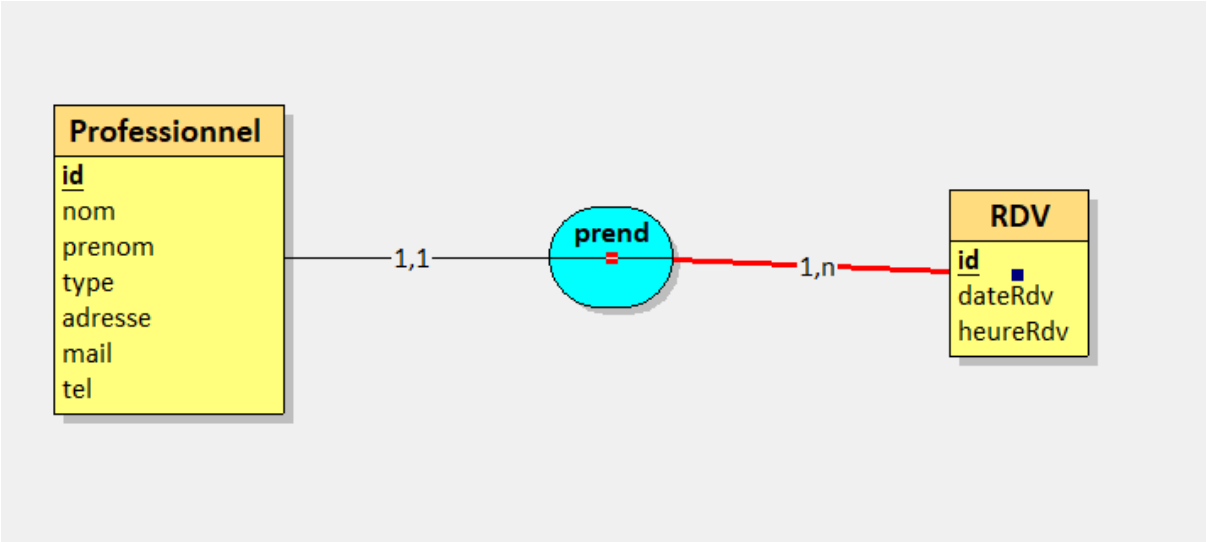
ALI WALID

BTS SIO 2

android studio

2024

MCD :



MLD :

```
CREATE TABLE RDV(
   id INT,
   dateRdv DATE,
   heureRdv VARCHAR(50),
   PRIMARY KEY(id)
);

CREATE TABLE Professionnel_(
   id INT,
   nom VARCHAR(50),
   prenom VARCHAR(50),
   type VARCHAR(50),
   adresse VARCHAR(50),
   mail VARCHAR(50),
   tel VARCHAR(50),
   id_1 INT NOT NULL,
   PRIMARY KEY(id),
   FOREIGN KEY(id_1) REFERENCES RDV(id)
);
```

Maquette :

Application de gestion

Enregistrement d'un professionnel

Nom :

Prénom :

Profession :

adresse :

Mail :

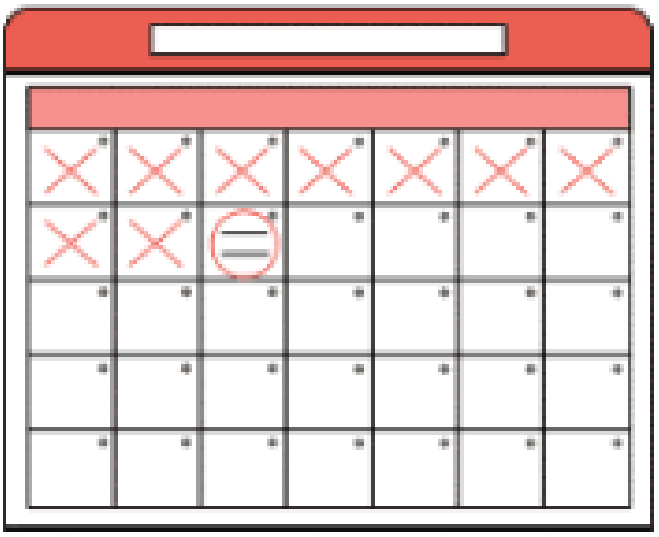
Tél :

Enregistrer

Retour

Prendre RDV

Professionnel :



heure :

Prendre RDV

Retour

Rechercher les professionnels

Adresse : 

Rechercher

Retour

Planning des RDV de la journéee

Selectionner une date

Afficher les rendez-vous

Retour

Cahier de Recette

1. Introduction

Nom de l'application : AG- Application de Gestion
Description : Application qui permet aux visiteurs médicaux de gérer leurs rendez-vous avec des professionnels de santé.
avec comme fonctionnalités pour enregistrer des professionnels, planifier des rendez-vous, afficher le planning d'une journée, et rechercher des professionnels par ville ou code postal.

2. Objectifs de la Recette

Vérifier que toutes les fonctionnalités répondent aux spécifications initiales.
Identifier et corriger les éventuelles anomalies avant la livraison finale.
Tester l'application dans différents scénarios d'utilisation.

3. Environnement de Test

Version Android : Pixel 9 Pro API 35
Outils de développement : Android Studio, SQLite
Appareil de test : Émulateur Android
Base de données : SQLite intégrée à l'application.

Tests Unitaires

Vérifier que les méthodes des classes Java (notamment celles de gestion de la base de données) fonctionnent correctement.
Exemple :
Méthode insererRDV : vérifier qu'un rendez-vous est bien inséré dans la base.
Méthode getProfessionnelData : vérifier que les données sont correctement récupérées.

Tests de Performances

Tester le temps de réponse de l'application pour l'ajout, la modification et la récupération des données.

Tests d'Installation

Vérifier que l'application peut être installée et lancée correctement sur différents appareils Android.

Fonctionnalité	Description	Étapes de test	Résultat attendu	Résultat obtenu
Ajouter un professionnel	Ajouter un professionnel avec nom, prénom, type, adresse, mail, et tél	1. Ouvrir l'écran d'ajout 2. Remplir tous les champs 3. Cliquer sur "Ajouter"	Le professionnel est ajouté et visible dans la liste	Le professionnel a été ajouté avec succès.
Prendre un rendez-vous	Programmer un RDV avec date, heure et professionnel	1. Sélectionner une date dans le calendrier 2. Choisir un professionnel 3. Sélectionner une heure 4. Valider	Le RDV est enregistré et visible dans le planning	Le rendez-vous a été enregistré correctement et s'affiche dans le planning.
Afficher le planning quotidien	Afficher tous les RDV pour une date donnée	1. Ouvrir l'écran du planning 2. Sélectionner une date 3. Cliquer sur "Afficher les RDV"	La liste des RDV pour la date sélectionnée s'affiche	Les rendez-vous sont affichés pour la date sélectionnée.
Rechercher un professionnel	Rechercher des professionnels par ville ou code postal	1. Saisir une ville 2. Cliquer sur "Rechercher" 3. Répéter avec un code postal	La liste des professionnels correspondants s'affiche	Les recherches par ville et code postal retournent les résultats attendus.

Installation de l'application :

On génère l’APK : app-debug.apk
puis on double clique pour l’installer

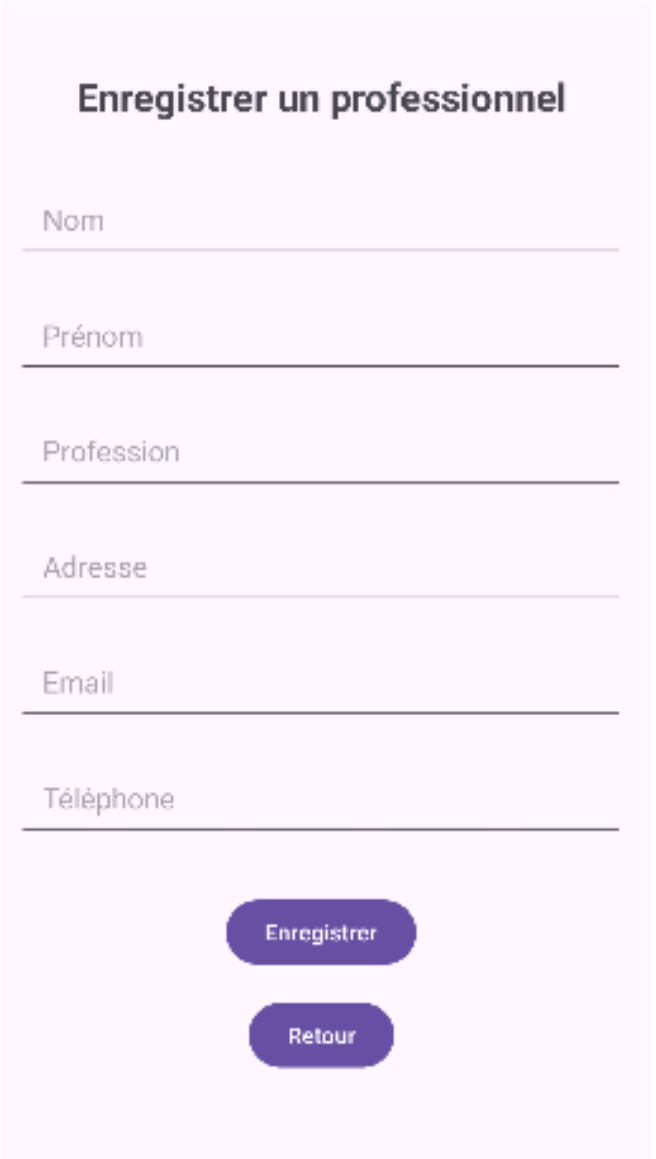
INTERFACE + CODE

Menu

```java
/**
 * Classe représentant l'écran principal de l'application.
 * Cette activité permet de naviguer vers les différentes sections de l'application.
 */
public class MainActivity extends AppCompatActivity {

    /**
     * Méthode appelée lors de la création de l'activité.
     * Initialise l'interface utilisateur et gère les barres système (par exemple, la barre de statut).
     *
     * @param savedInstanceState État sauvegardé de l'activité (si applicable).
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(R.layout.activity_main);

        // Ajuste les marges pour tenir compte des barres système.
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });
    }

    /**
     * Méthode appelée lorsqu'on clique sur le bouton "Enregistrer".
     * Redirige vers l'écran permettant d'enregistrer un professionnel.
     *
     * @param view La vue qui a déclenché cet événement.
     */
    public void enregistrer(View view) {
        Intent intent2 = new Intent( packageContext: this, enregistrer.class);
        startActivity(intent2);
    }
```

```java
/**
 * Méthode appelée lorsqu'on clique sur le bouton "Prendre RDV".
 * Redirige vers l'écran permettant de prendre un rendez-vous.
 *
 * @param view La vue qui a déclenché cet événement.
 */
public void prendreRdv(View view) {
    Intent intent2 = new Intent( packageContext: this, prendreRdv.class);
    startActivity(intent2);
}

/**
 * Méthode appelée lorsqu'on clique sur le bouton "Afficher Professionnels".
 * Redirige vers l'écran permettant de rechercher des professionnels.
 *
 * @param view La vue qui a déclenché cet événement.
 */
public void AfficherP(View view) {
    Intent intent2 = new Intent( packageContext: this, Rechercher_professionnels.class);
    startActivity(intent2);
}

/**
 * Méthode appelée lorsqu'on clique sur le bouton "Planning".
 * Redirige vers l'écran permettant d'afficher le planning des rendez-vous.
 *
 * @param view La vue qui a déclenché cet événement.
 */
public void planning(View view) {
    Intent intent2 = new Intent( packageContext: this, planning.class);
    startActivity(intent2);
}
```

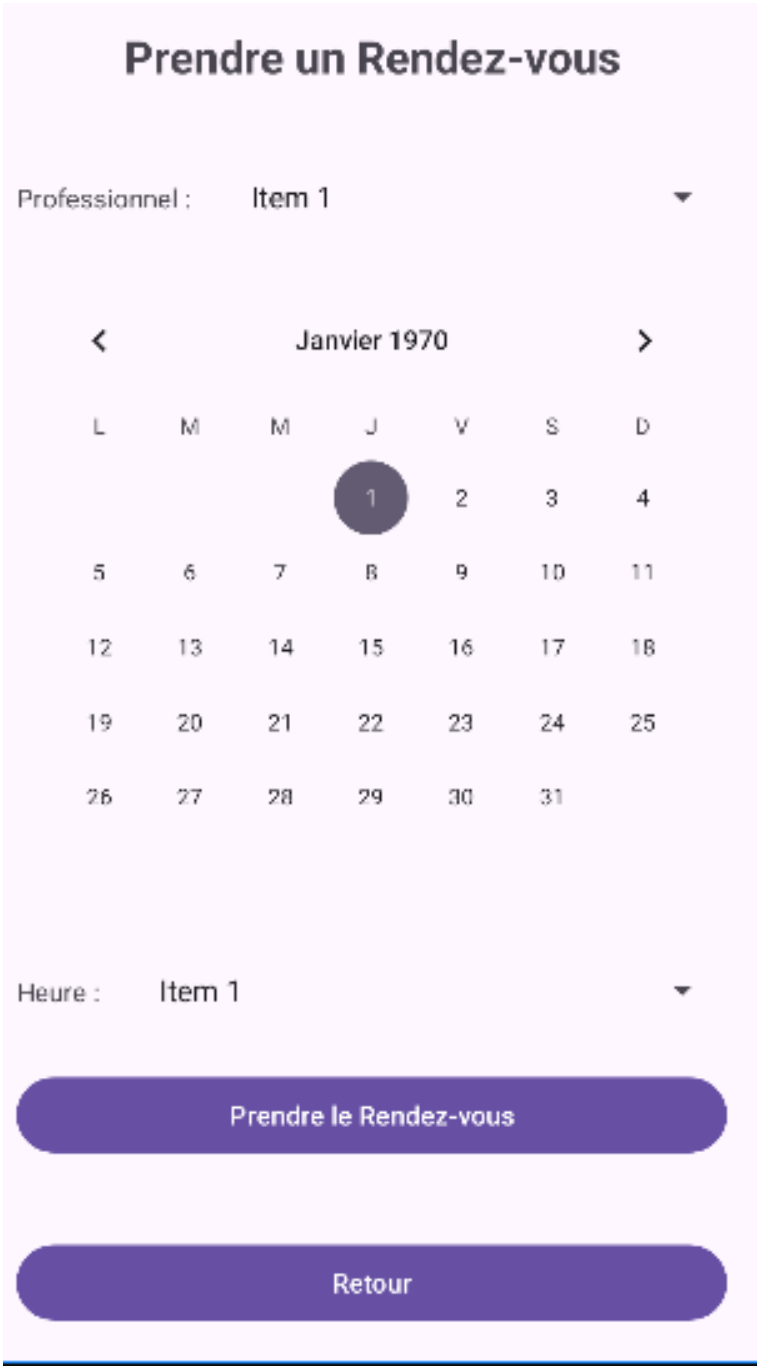
Enregistrer



```java
/**
 * Classe représentant l'écran permettant d'enregistrer un professionnel.
 */
public class enregistrer extends AppCompatActivity {

    /** Instance de la base de données. */
    BD db;

    /** Champs pour saisir les informations du professionnel. */
    private EditText nom, prenom, adresse, mail, tel, types;

    /**
     * Méthode appelée lors de la création de l'activité.
     * Initialise les composants de l'interface utilisateur et la base de données.
     *
     * @param savedInstanceState État sauvegardé de l'activité (si applicable).
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_enregistrer);

        // Ajuster les marges pour tenir compte des barres système.
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });

        // Initialisation de la base de données.
        db = new BD( context: this);

        // Initialisation des champs de saisie.
        nom = findViewById(R.id.nom);
        prenom = findViewById(R.id.prenom);
        types = findViewById(R.id.types);
        adresse = findViewById(R.id.adresse);
        mail = findViewById(R.id.mail);
        tel = findViewById(R.id.tel);
    }
```

```java
/**
 * Méthode appelée lorsqu'on clique sur le bouton "Enregistrer".
 * Insère les informations du professionnel dans la base de données.
 *
 * @param view La vue qui a déclenché cet événement.
 */
public void InsertionInfo(View view) {
    db.insertProfessionnel(
            nom.getText().toString(),
            prenom.getText().toString(),
            types.getText().toString(),
            adresse.getText().toString(),
            mail.getText().toString(),
            tel.getText().toString()
    );
}

/**
 * Méthode appelée lorsqu'on clique sur le bouton "Retour".
 * Redirige vers l'écran principal.
 *
 * @param view La vue qui a déclenché cet événement.
 */
public void Retour(View view) {
    Intent intent2 = new Intent( packageContext: this, MainActivity.class);
    startActivity(intent2);
}
```

Prendre RDV


Prendre un Rendez-vous
Professionnel :
Item 1
Janvier 1970
L M M J V S D
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
Heure :
Item 1
Prendre le Rendez-vous
Retour

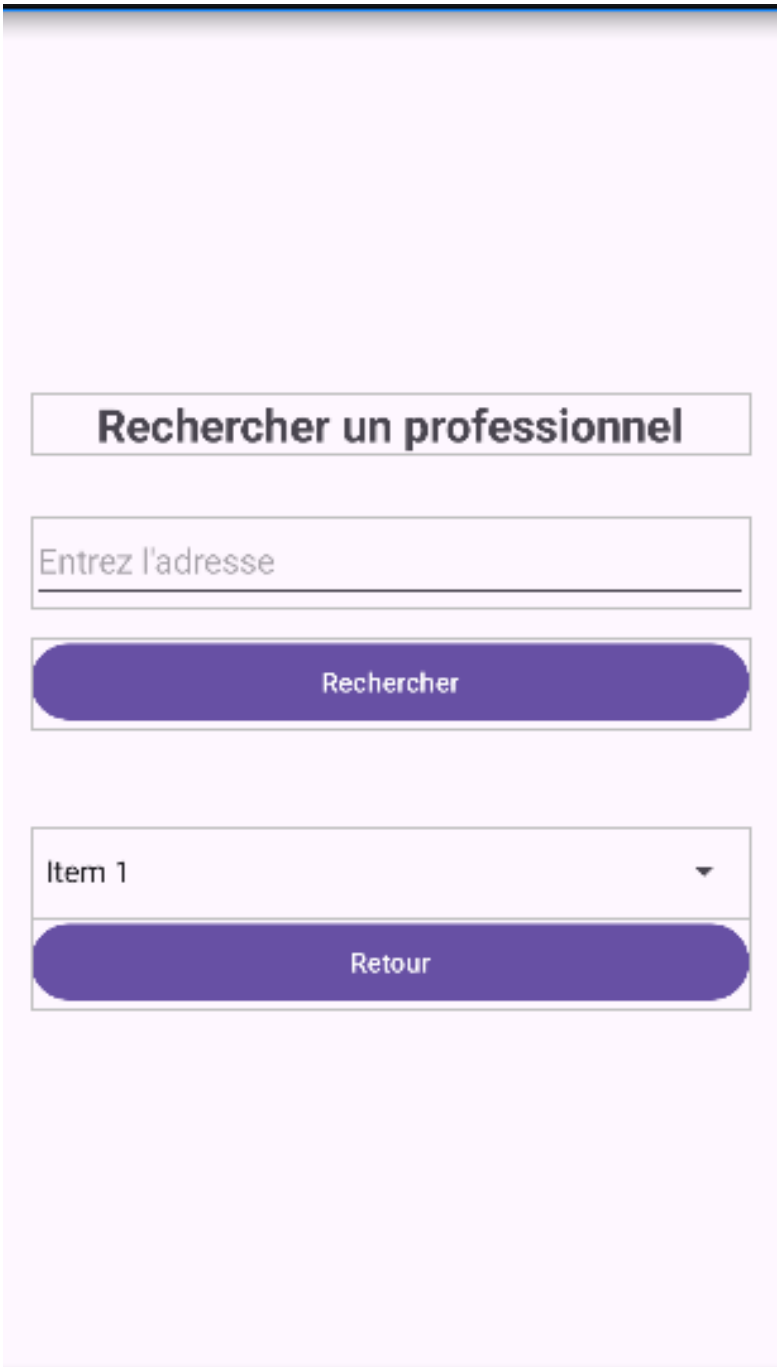
```java
/**
 * Classe représentant l'écran permettant de prendre un rendez-vous.
 * Permet de sélectionner une date, une heure et un professionnel pour prendre un rendez-vous.
 */
public class prendreRdv extends AppCompatActivity {

    private BD db;
    private CalendarView calendarView;
    private TextView test;
    private String[] HeureRdv = {"8h", "8h30", "9h", "9h30", "10h", "10h30", "11h", "11h30", "13h30", "14h", "14h30", "15h", "15h30", "16h", "16h30"};
    private Spinner spinHeure, spinPro;
    private int HeureSelect;
    private String selectDate, selectPro;

    /**
     * Méthode appelée lors de la création de l'activité.
     * Initialise les composants de l'interface utilisateur et configure les écouteurs pour les sélecteurs.
     *
     * @param savedInstanceState État sauvegardé de l'activité (si applicable).
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_prendre_rdv);

        db = new BD( context: this);

        spinHeure = findViewById(R.id.heureRdv);
        spinPro = findViewById(R.id.pro);
        calendarView = findViewById(R.id.calendarView);
        test = findViewById(R.id.test);

        // Initialisation de l'adaptateur pour le Spinner Heure
        ArrayAdapter<String> heure = new ArrayAdapter<>( context: this, android.R.layout.simple_spinner_item, HeureRdv);
        heure.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        spinHeure.setAdapter(heure);
```

```java
// Écouteur pour le Spinner Heure
spinHeure.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        HeureSelect = position;
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        // Rien à faire
    }
});

// Écouteur pour le calendrier
calendarView.setOnDateChangeListener((view, year, month, dayOfMonth) -> {
    selectDate = String.format("%04d-%02d-%02d", year, month + 1, dayOfMonth); // Format : yyyy-MM-dd
});

// Écouteur pour le Spinner Professionnel
spinPro.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        selectPro = (String) parent.getItemAtPosition(position);
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        // Rien à faire
    }
});

// Chargement des professionnels à partir de la base de données
getProfessionnelData();
```

```java
/**
 * Récupère les données des professionnels à partir de la base de données et les affiche dans le Spinner.
 */
public void getProfessionnelData() {
    try {
        Cursor data = db.getProfessionnelData();
        ArrayList<String> proListe = new ArrayList<>();
        while (data.moveToNext()) {
            String info = data.getString( columnIndex: 1) + " " + data.getString( columnIndex: 2) + " " + data.getString( columnIndex: 3); // Nom + Prénom
            proListe.add(info);
        }
        data.close();

        // Remplissage du Spinner avec la liste des professionnels
        ArrayAdapter<String> listePro = new ArrayAdapter<>( context: this, android.R.layout.simple_spinner_item, proListe);
        listePro.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        spinPro.setAdapter(listePro);
    } catch (Exception e) {
        // Gestion des erreurs
    }
}

/**
 * Insère un rendez-vous dans la base de données.
 *
 * @param date La date du rendez-vous.
 * @param heure L'heure du rendez-vous.
 * @param pro Le professionnel associé au rendez-vous.
 */
private void insererRDV(String date, String heure, String pro) {
    try {
        db.insererRDV(date, heure, pro);
    } catch (Exception e) {
        // Gestion des erreurs
    }
}
```

```java
/**
 * Méthode appelée lors du clic sur le bouton "Prendre RDV".
 * Vérifie les informations saisies et insère le rendez-vous dans la base de données si tout est valide.
 *
 * @param view La vue qui a déclenché cet événement.
 */
public void prendreRDV(View view) {
    try {
        if (selectDate != null && !selectDate.isEmpty() && HeureSelect >= 0 && selectPro != null && !selectPro.isEmpty()) {
            String heureR = HeureRdv[HeureSelect];

            insererRDV(selectDate, heureR, selectPro);
            test.setText("Rendez-vous pris avec succès pour le " + selectDate + " à " + heureR + " avec " + selectPro);
        } else {
            test.setText("Sélectionnez tous les champs.");
        }
    } catch (Exception e) {
        // Gestion des erreurs
    }
}

/**
 * Méthode pour revenir à l'écran principal.
 *
 * @param view La vue qui a déclenché cet événement.
 */
public void Retour(View view) {
    Intent intent2 = new Intent( packageContext: this, MainActivity.class);
    startActivity(intent2);
}
```

Rechercher Professionnels

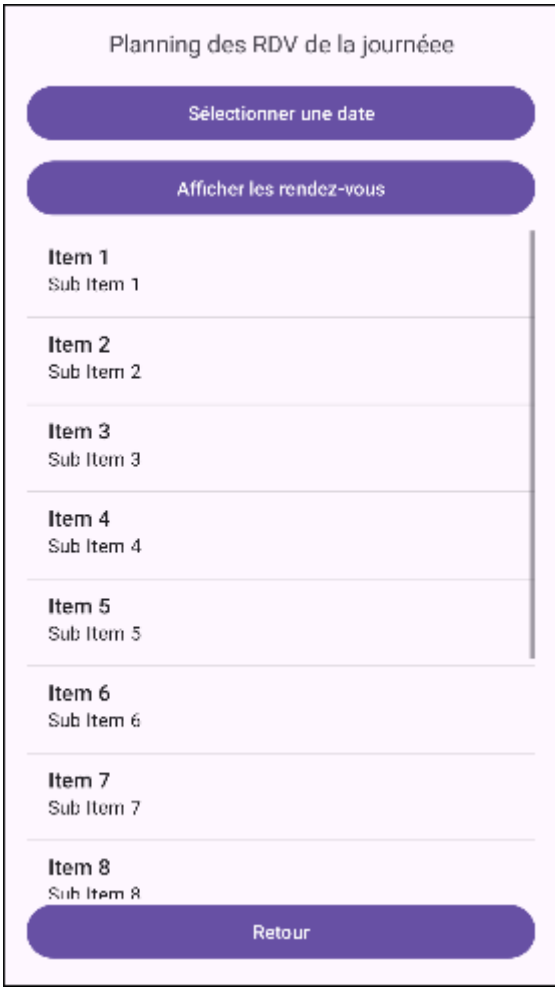


```java
/**
 * Classe représentant l'écran permettant de rechercher des professionnels par adresse.
 * Permet à l'utilisateur de saisir une adresse et de trouver les professionnels correspondants.
 */
public class Rechercher_professionnels extends AppCompatActivity {

    private BD db;
    private Spinner listePro;
    private EditText adresse;
    private Button rechercher;

    /**
     * Méthode appelée lors de la création de l'activité.
     * Initialise les composants de l'interface utilisateur et configure le bouton de recherche.
     *
     * @param savedInstanceState État sauvegardé de l'activité (si applicable).
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_rechercher_professionnels);

        db = new BD( context: this);
        listePro = findViewById(R.id.listePro);
        adresse = findViewById(R.id.adresse);
        rechercher = findViewById(R.id.rechercher);

        // Configure le bouton de recherche
        rechercher.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                rechercherProfessionnels();
            }
        });
    }
```

```java
/**
 * Recherche les professionnels dans la base de données en fonction de l'adresse saisie.
 * Affiche les résultats dans un Spinner. Si aucun professionnel n'est trouvé, un message est affiché.
 */
private void rechercherProfessionnels() {
    String query = adresse.getText().toString().trim(); // Récupère l'adresse saisie par l'utilisateur

    // Si l'adresse n'est pas vide
    if (!query.isEmpty()) {
        // Effectue la recherche des professionnels en fonction de l'adresse
        Cursor cursor = db.getListePro(query);
        ArrayList<String> resultats = new ArrayList<>();

        // Ajoute les résultats à la liste
        while (cursor.moveToNext()) {
            String nom = cursor.getString( columnIndex: 0); // Récupère le nom du professionnel
            String prenom = cursor.getString( columnIndex: 1); // Récupère le prénom du professionnel
            resultats.add(nom + " " + prenom); // Ajoute le nom et prénom dans la liste
        }

        // Si aucun résultat n'est trouvé
        if (resultats.isEmpty()) {
            resultats.add("Aucun professionnel trouvé");
        }
        cursor.close();

        // Met à jour le Spinner avec les résultats trouvés
        ArrayAdapter<String> adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_spinner_item, resultats);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        listePro.setAdapter(adapter);
    }
}
```

```java
/**
 * Méthode pour revenir à l'écran principal.
 *
 * @param view La vue qui a déclenché cet événement.
 */
public void Retour(View view) {
    Intent intent2 = new Intent( packageContext: this, MainActivity.class);
    startActivity(intent2);
}
```

Planning


Planning des RDV de la journéee
Sélectionner une date
Afficher les rendez-vous
Item 1
Sub Item 1
Item 2
Sub Item 2
Item 3
Sub Item 3
Item 4
Sub Item 4
Item 5
Sub Item 5
Item 6
Sub Item 6
Item 7
Sub Item 7
Item 8
Retour

```java
/**
 * Classe représentant l'écran permettant de visualiser les rendez-vous pour une date donnée.
 * L'utilisateur peut sélectionner une date via un calendrier et afficher les rendez-vous associés.
 */
public class planning extends AppCompatActivity {

    private ListView listViewPlanning;   // Liste des rendez-vous à afficher
    private Button btnDate, btnAfficherRdv;  // Boutons pour sélectionner la date et afficher les RDV
    private BD db;  // Objet pour accéder à la base de données
    private ArrayAdapter<String> adapter; // Adaptateur pour afficher la liste des rendez-vous
    private String selectedDate;  // Date sélectionnée par l'utilisateur

    /**
     * Méthode appelée lors de la création de l'activité.
     * Initialise les composants de l'interface utilisateur et les événements des boutons.
     *
     * @param savedInstanceState État sauvegardé de l'activité (si applicable).
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_planning);

        // Initialisation des vues
        listViewPlanning = findViewById(R.id.listViewPlanning);
        btnDate = findViewById(R.id.btnDate);
        btnAfficherRdv = findViewById(R.id.btnAfficherRdv);
        db = new BD( context: this);

        // Événement pour le bouton "Sélectionner la date"
        btnDate.setOnClickListener(v -> showDatePickerDialog());

        // Événement pour le bouton "Afficher les rendez-vous"
        btnAfficherRdv.setOnClickListener(v -> afficherRDV(selectedDate));
    }
```

```java
/**
 * Méthode pour afficher le sélecteur de date via un dialogue.
 * Permet à l'utilisateur de choisir une date pour afficher les rendez-vous associés.
 */
private void showDatePickerDialog() {
    Calendar calendar = Calendar.getInstance();
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    // Ouvre un dialogue pour sélectionner une date
    new DatePickerDialog( context: this, (view, selectedYear, selectedMonth, selectedDay) -> {
        // Formate la date choisie en "YYYY-MM-DD"
        selectedDate = String.format(Locale.getDefault(), format: "%04d-%02d-%02d",
                ...args: selectedYear, selectedMonth + 1, selectedDay);
    }, year, month, day).show();
}
```

```java
/**
 * Méthode pour afficher les rendez-vous pour la date sélectionnée.
 * Effectue une requête sur la base de données et affiche les rendez-vous dans la liste.
 *
 * @param date La date pour laquelle les rendez-vous doivent être affichés.
 */
private void afficherRDV(String date) {
    String[] colonnes = {BD.dateRdv, BD.heureRdv, BD.proRdv}; // Colonnes à récupérer
    String condition = BD.dateRdv + " = ?"; // Filtre par date
    String[] dateR = {date};  // Paramètre de la date sélectionnée
    String heureR = BD.heureRdv + " ASC";  // Tri des résultats par heure

    // Effectue la requête dans la base de données
    Cursor cursor = db.rechercher(BD.rdvTable, colonnes, condition, dateR, heureR);
    List<String> rdvListe = new ArrayList<>();  // Liste des rendez-vous

    // Ajoute chaque rendez-vous à la liste
    if (cursor != null) {
        while (cursor.moveToNext()) {
            String rdvInfo = "Professionnel : " + cursor.getString(cursor.getColumnIndexOrThrow(BD.proRdv)) +
                    " - Heure : " + cursor.getString(cursor.getColumnIndexOrThrow(BD.heureRdv));
            rdvListe.add(rdvInfo);
        }
        cursor.close();
    }

    // Si aucun rendez-vous n'est trouvé, affiche un message
    if (rdvListe.isEmpty()) {
        rdvListe.add("Aucun rendez-vous pour cette date.");
    }

    // Met à jour l'adaptateur de la liste avec les résultats
    adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1, rdvListe);
    listViewPlanning.setAdapter(adapter);
}
```

```java
/**
 * Méthode pour revenir à l'écran principal.
 *
 * @param view La vue qui a déclenché cet événement.
 */
public void retour(View view) {
    startActivity(new Intent( packageContext: this, MainActivity.class));  // Lance l'écran principal
}
```

Classe BD

```java
public class BD extends SQLiteOpenHelper {

    public static final String ProjetA = "ProjetA.db";
    public static final String professionnel = "professionnel_table";
    public static final String idP = "ID";
    public static final String nom = "NOM";
    public static final String prenom = "PRENOM";
    public static final String types = "TYPES";
    public static final String adresse = "ADRESSES";
    public static final String mail = "MAIL";
    public static final String tel = "TEL";

    public static final String rdvTable = "rdv_table";
    public static final String idRdv = "IDRDV";
    public static final String dateRdv = "DATERDV";
    public static final String heureRdv = "HEURERDV";
    public static final String proRdv = "PRORDV";

    /**
     * Constructeur de la classe BD.
     *
     * @param context Le contexte dans lequel la base de données est utilisée.
     */
    public BD(Context context) {
        super(context, ProjetA, factory: null, version: 1);
    }

    /**
     * Crée les tables "professionnel_table" et "rdv_table" dans la base de données.
     *
     * @param db La base de données SQLite.
     */
    @Override
```

```java
public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE " + professionnel + " (" +
            idP + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            nom + " TEXT, " +
            prenom + " TEXT, " +
            types + " TEXT, " +
            adresse + " TEXT, " +
            mail + " TEXT, " +
            tel + " TEXT)");

    db.execSQL("CREATE TABLE " + rdvTable + " (" +
            idRdv + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            dateRdv + " TEXT, " +
            heureRdv + " TEXT, " +
            proRdv + " TEXT)");
}

/**
 * Met à jour la base de données en supprimant les tables existantes et en recréant les tables.
 *
 * @param db La base de données SQLite.
 * @param oldVersion La version ancienne de la base de données.
 * @param newVersion La nouvelle version de la base de données.
 */
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + professionnel);
    db.execSQL("DROP TABLE IF EXISTS " + rdvTable);
    onCreate(db);
}
```

```java
/**
 * Insère un professionnel dans la table "professionnel_table".
 *
 * @param nom Le nom du professionnel.
 * @param prenom Le prénom du professionnel.
 * @param types Le type de professionnel.
 * @param adresse L'adresse du professionnel.
 * @param mail L'email du professionnel.
 * @param tel Le numéro de téléphone du professionnel.
 */
public void insertProfessionnel(String nom, String prenom, String types, String adresse, String mail, String tel) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues cv = new ContentValues();
    cv.put(BD.nom, nom);
    cv.put(BD.prenom, prenom);
    cv.put(BD.types, types);
    cv.put(BD.adresse, adresse);
    cv.put(BD.mail, mail);
    cv.put(BD.tel, tel);
    db.insert(professionnel, nullColumnHack: null, cv);
    db.close();
}
```

```java
/**
 * Insère un rendez-vous dans la table "rdv_table" après avoir vérifié qu'un rendez-vous similaire n'existe pas déjà.
 *
 * @param dateRdv La date du rendez-vous.
 * @param heureRdv L'heure du rendez-vous.
 * @param proRdv Le professionnel associé au rendez-vous.
 */
public void insererRDV(String dateRdv, String heureRdv, String proRdv) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues cv2 = new ContentValues();
    cv2.put(BD.dateRdv, dateRdv);
    cv2.put(BD.heureRdv, heureRdv);
    cv2.put(BD.proRdv, proRdv);

    String selection = BD.dateRdv + " = ? AND " + BD.heureRdv + " = ? AND " + BD.proRdv + " = ?";
    String[] selectionArgs = {dateRdv, heureRdv, proRdv};

    Cursor cursor = db.query(rdvTable, columns: null, selection, selectionArgs, groupBy: null, having: null, orderBy: null);

    if (cursor.getCount() == 0) {
        db.insert(rdvTable, nullColumnHack: null, cv2);
    }

    cursor.close();
    db.close();
}

/**
 * Récupère toutes les données des professionnels.
 *
 * @return Un curseur contenant toutes les données de la table des professionnels.
 */
public Cursor getProfessionnelData() {
    SQLiteDatabase db = this.getReadableDatabase();
    return db.rawQuery( sql: "SELECT * FROM " + professionnel, selectionArgs: null);
}
```

```java
/**
 * Recherche les professionnels ayant une adresse spécifique.
 *
 * @param query L'adresse à rechercher.
 * @return Un curseur contenant les professionnels correspondant à l'adresse.
 */
public Cursor getListePro(String query) {
    SQLiteDatabase db = this.getReadableDatabase();
    return db.rawQuery(
            sql: "SELECT " + nom + ", " + prenom + " FROM " + professionnel + " WHERE " + adresse + " = ?",
            new String[]{query});
}

/**
 * Recherche des données dans une table spécifiée avec des critères donnés.
 *
 * @param table Le nom de la table à interroger.
 * @param colonnes Les colonnes à récupérer.
 * @param condition La condition WHERE pour la recherche.
 * @param dateR Les arguments pour la condition WHERE.
 * @param heureR La colonne de tri des résultats.
 * @return Un curseur contenant les résultats de la requête.
 */
public Cursor rechercher(String table, String[] colonnes, String condition, String[] dateR, String heureR) {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = null;
    try {
        String query = "SELECT " + String.join( delimiter: ", ", colonnes) +
                " FROM " + table +
                " WHERE " + condition +
                " ORDER BY " + heureR;

        cursor = db.rawQuery(query, dateR);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return cursor;
}
```