K.D. Polytechnic, Patan

(Government Polytechnic Institute, Affiliated to GTU)

Project Report On

"BLOOD DONATION APP"

Submitted By:
Name: Marediya Rehan Mahendihusen
Enrollment No.: 236310307055

Under the Guidance of:
Mr. Sagar Jasani
Designation: Project Manager

In Partial Fulfilment of the Requirements for the Award of the Diploma in
Computer Engineering

Academic Year: 2025–2026

Department of Computer Engineering
K.D. Polytechnic, Patan
Affiliated to Gujarat Technological University (GTU

Project Profile

Title : BLOOD-Donation Documentation

Abstract :

This Blood Donation app is designed to facilitate life-saving connections between blood donors and those in need. With an intuitive interface, users can effortlessly register as a blood donor or search for available donors by blood type and location. The app not only simplifies the process of finding suitable donors but also enables users to track their donation history and receive rewards for their contributions. Aimed at individuals eager to make a difference, this app serves as a crucial tool in the effort to save lives through blood donation. Optimized for mobile use, it ensures users have a seamless and impactful experience, encouraging a community of regular donors.

Key-Words :

- Call Support
- Location Services
- Internet Access
- Network Awareness
- User Onboarding
- Security Features

Modules :

- SplashActivity
- LoginActivity
- SignupActivity
- HomeActivity
- ProfileActivity
- ForgotPasswordActivity
- ChangePasswordActivity

- LocationActivity
- DonationHistoryActivity
- NotificationActivity
- SettingsActivit

Tech Stack :

- **Platform**: Android
- **Programming Language**: Java
- **Database**: MySQL
- **User Interface:** XML (for Android UI design)
- **Backend**: PHP (optional for server-side logic if applicable)
- **API**: RESTful APIs (optional for backend communication)
- **Version Control**: Git
- **Build Tool:** Gradle
- **Integrated Development Environment (IDE):** Android Studio

Front-End :

- XML

Back End :

- Java

Database :

- MySQL

Step 1:

Delving into Android and Its Versions:

Android, the widely-used mobile operating system created by Google, has undergone significant development since its debut in 2008 with Android 1.0, nicknamed "Astro." This OS has evolved through various iterations, each marked by a unique dessert name, following an alphabetical sequence. This charming tradition highlights the playful and community-focused aspect of Android's evolution.

By the time of the last update, the system has advanced to Android 12, which marks a substantial shift in design, enhanced privacy settings, and improved performance. Android continues to build on the solid

groundwork established by its earlier versions, integrating user and developer feedback to push technological boundaries further.

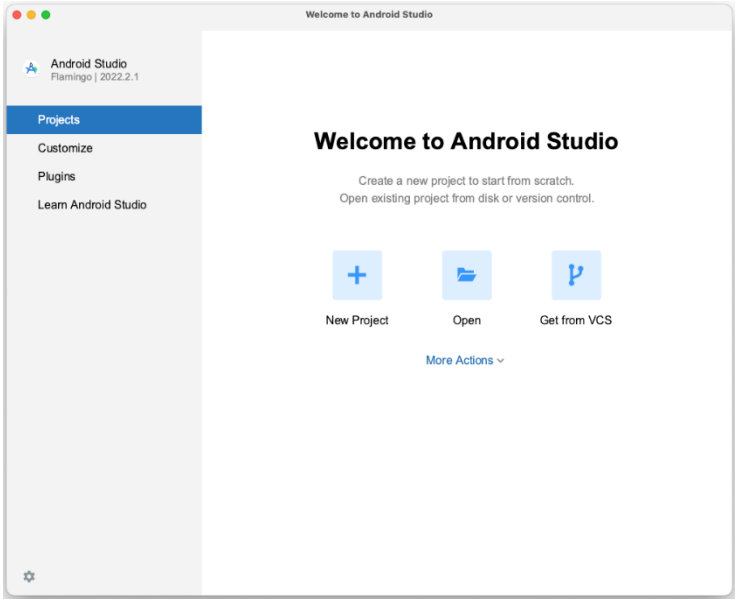
Android's transformative impact on mobile technology is evident as it adapts to the dynamic demands of global users. From the early days of Cupcake through to the innovations of KitKat, and up to the recent advancements in Pie, Android has consistently set the standard for what mobile operating systems can achieve, proving itself as a pivotal player in the tech arena. This section delves into the detailed evolution of Android, tracking its progression from its origins to its present status as a cornerstone of the mobile market.

Step 2:

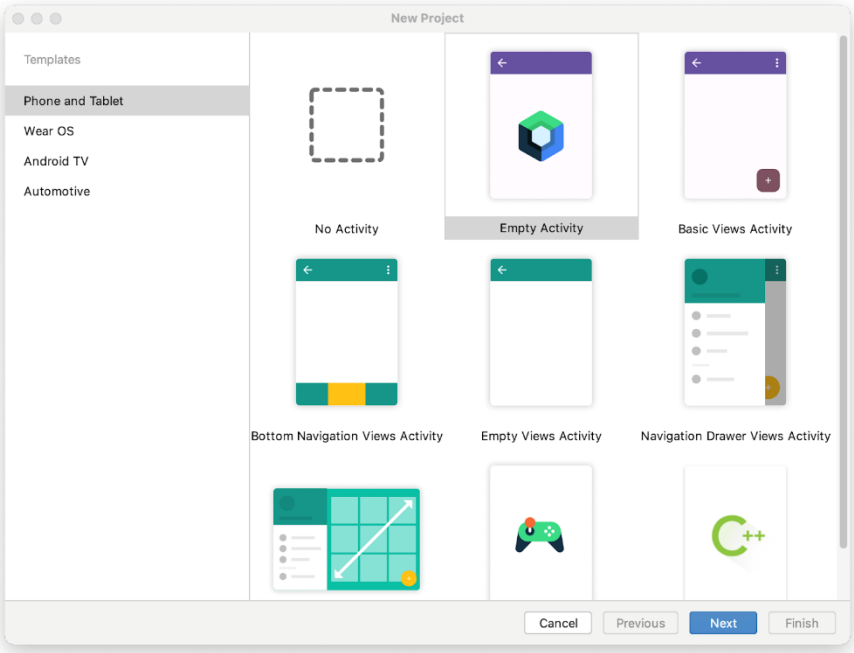
Create Project And Introduction About The Folders:

1. To initiate the utilization of Android Studio, simply double-click on its icon.

2. To create a fresh Android project, choose "New Project" from the welcome window within Android Studio.



3. Android Studio presents a variety of templates in the New Project dialog box.



In Android Studio, project templates serve as the fundamental framework for different application types. These templates define the structural layout and necessary files needed by Android Studio to build your project. Choosing a template provides pre-existing code to accelerate the development process of your project.

1. Ensure to choose the "Phone and Tablet" tab.

2. Choose the Empty Activity template, specifically crafted for commencing a straightforward project tailored for constructing a Compose app. This template showcases a solitary screen exhibiting the text "Hello Android!"

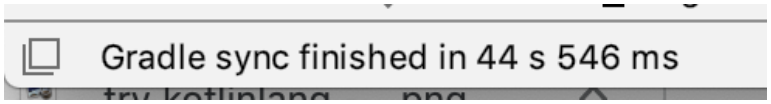
3. Click "Next" to proceed to the New Project dialog, where you can configure your project based on the following specifications:

- Enter "Greeting Card" in the Name field for your project.
- Keep the Package name field unchanged, as it determines the organization of files within the project's file structure. In this case, it will remain as com.example.greetingcard.
- Leave the Save location field unchanged, specifying the location where your project files are stored on your computer.

- API 24: Android 7.0 (Nougat)" from the Minimum SDK dropdown menu, indicating the minimum Android version required for your app to function.

- Click on the "Finish" button. This process may take a while, giving you a chance to relax and enjoy a cup of tea! As Android Studio initializes, you'll see a progress bar and messages updating you on the setup progress of your project. It might look something like this:

. A message similar to this one will inform you when the project setup is finished.



Project Structure: After creating the new project, you'll find the project structure in the left pane of Android Studio. Here's a brief overview of the main folders you'll encounter:

- **app:** This directory contains the core module of your Android application. It includes configuration files specific to the application, resources like layout XML files and drawable images, as well as the source code.

- **Gradle Scripts:** The Gradle build scripts for your project are located in this directory. The build.gradle files define the prerequisites and settings necessary for building your application.

- **Manifests:** This directory contains the AndroidManifest.xml file, which provides crucial information to the Android system about your app, including details such as its package name, components, permissions, and other relevant information.

- **Java**: Your Java/Kotlin source code is found in the java folder, located within the app directory. This is where you'll write the logic for your application.

- **Res:** The "Res" folder, located within the app directory, contains various types of resources used in your application, including layouts, drawables, values, and more. This folder has several subfolders to organize these resources effectively.

- **Drawable:** Within the res folder, the Drawable subdirectory contains resources such as images, icons, and other graphical elements.

- **Layout:** Within the res folder, the Layout subdirectory contains XML files that define the UI layout structure of your application.

- **Values:** Within the res folder, the Values subdirectory holds XML files used for storing values such as strings, dimensions, colors, and styles used throughout your application**.**

Step 3:

Creating Splash Screen UI Layout for the Blood Donation Application:

The Splash Screen is the first point of interaction for users with the Blood Donation application, setting a critical first impression. This guide will help you design an engaging and visually appealing UI layout for the Splash Screen.

Layout Structure: The layout uses a RelativeLayout as the root, providing flexibility in positioning and scaling UI components. This layout is particularly effective for accommodating a range of screen sizes and orientations while keeping the design elements intact.

ImageView: An ImageView is centrally placed to prominently feature the app's logo, reinforcing the Blood Donation brand identity right at the start. The strategic central placement ensures the logo captures immediate user attention.

Background: The background of the Splash Screen is set to a vibrant red color, symbolizing vitality and urgency, aligning with the blood donation theme.

Animation and Styles: Implement subtle animations to enhance the visual appeal. An AlphaAnimation is used on the ImageView, fading in the logo from invisible to fully visible, creating a dynamic entrance effect. This animation not only captivates the user but also adds a professional touch to the app's presentation.

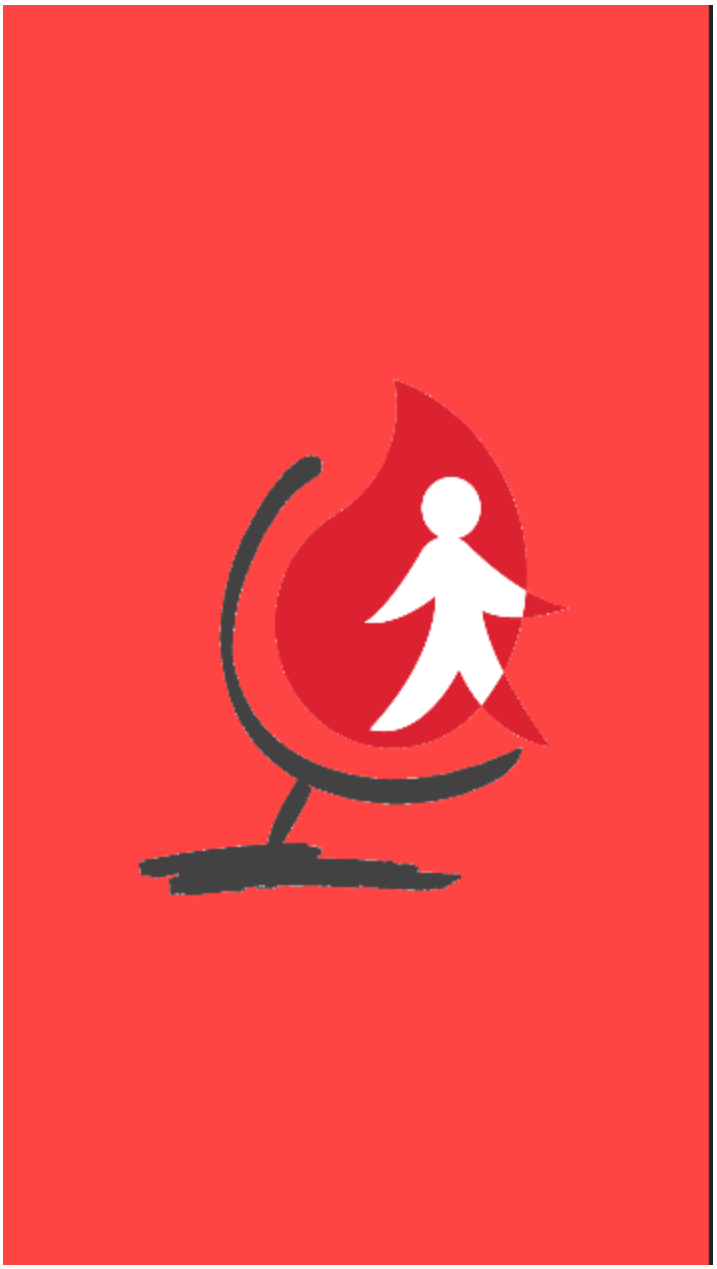
User Transition: After the animation completes, the application automatically transitions the user to the main interface based on their login status. This seamless transition ensures that users are smoothly guided from the Splash Screen to the primary functionalities of the app.

This thoughtfully designed Splash Screen not only elevates the aesthetic appeal of the Blood Donation app but also plays a pivotal role in transitioning users effectively from the introduction to the app's core features, enhancing the initial user experience.

XML code :

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    android:background="@android:color/holo_red_light"

    tools:context="blood.donate.SplashScreen">


    <ImageView
```

```
        android:id="@+id/splash_iv"
        android:layout_width="250dp"
        android:layout_height="300dp"
        android:layout_centerInParent="true"
        android:background="@drawable/logo" />

</RelativeLayout>
```



Step 4:

Implementing Login Functionality in the Blood Donation Application:

This guide will help you understand the creation and functionality of the `LoginFragment` used in the Blood Donation app. This fragment plays a crucial role in authenticating users and granting access to the app's main features.

Overview: The `LoginFragment` class extends `Fragment`, providing a user interface where users can enter their login credentials to access their account. The fragment is responsible for handling user authentication against stored credentials in a local SQLite database.

Key Components and Operations:

1. **UI Components**:
    - **EditText**: Two input fields, `edtemail` and `edtpass`, collect the user's email and password.
    - **Button**: The `btnlogin` initiates the login process when clicked.
2. **Database Interaction**:
    - The fragment initializes or opens an SQLite database named "Blood Donate" and ensures the 'register' table is created if it does not exist. This table stores user registration data, including email and password.
    - Upon clicking the login button, the app queries this table to check if the entered credentials match any stored records.
3. **Validation Checks**:
    - **Email Validation**: The email entered by the user is validated against a predefined pattern to ensure it's in a proper format. If not, a toast message informs the user of the invalid email.
    - **Field Completeness**: The app checks if the email and password fields are filled. If any field is empty, an error prompt is displayed.
4. **Successful Login**:
    - If the credentials are correct, the user's email is stored in `SharedPreferences` under `Constans.USERNAME`, which might be used for session management.
    - The user is then redirected to the `HomeActivity`, signifying a successful login.
5. **Error Handling**:
    - If no matching records are found in the database, a toast notifies the user of an unsuccessful login attempt, prompting them to check their credentials or register if they haven't yet.

Enhancing User Experience:

- Implementing more sophisticated error handling and validation feedback can significantly enhance the user experience. For instance, separating checks for non-existent users and incorrect passwords provides clearer guidance to users.
- Adding password visibility toggles and implementing more interactive, responsive validation could make the login process more user-friendly.

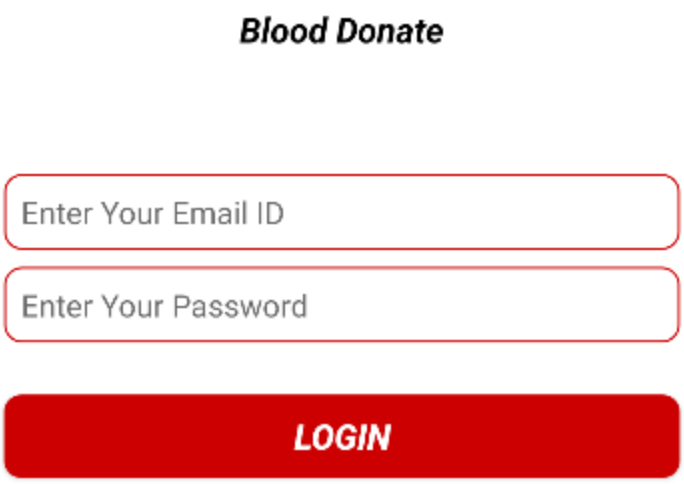
This fragment is integral to securing and personalizing the user experience in the Blood Donation app, ensuring that access is granted only to verified users and providing a seamless transition to the main content of the app.

XML :

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/white">
```

```
<TextView
    android:id="@+id/login_txt_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="30dp"
    android:text="Blood Donate"
    android:textColor="@android:color/black"
    android:textSize="20dp"
    android:textStyle="bold|italic" />

<EditText
    android:id="@+id/login_email"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/login_txt_title"
    android:layout_marginLeft="10dp"
    android:layout_marginTop="70dp"
    android:layout_marginRight="10dp"
    android:background="@drawable/custom_border"
    android:hint="@string/email"
    android:padding="10dp" />

<EditText
    android:id="@+id/login_pass"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/login_email"
    android:layout_marginLeft="10dp"
    android:layout_marginTop="10dp"
```

```
        android:layout_marginRight="10dp"
        android:background="@drawable/custom_border"
        android:hint="@string/pass"
        android:inputType="textPassword"
        android:padding="10dp" />

    <Button
        android:id="@+id/login_btn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/login_pass"
        android:layout_marginLeft="10dp"
        android:layout_marginTop="30dp"
        android:layout_marginRight="10dp"
        android:background="@drawable/custom_btn"
        android:text="Login"
        android:textColor="@android:color/white"
        android:textSize="20dp"
        android:textStyle="italic|bold" />

</RelativeLayout>
```



Step 5 :

Implementing Password Change Functionality in the Blood Donation Application:

This guide will walk you through the `ChangePassFragment` used in the Blood Donation app. This fragment is vital for allowing users to securely update their passwords.

Overview: The `ChangePassFragment` class extends `Fragment`, providing a user interface for changing passwords. It is essential for maintaining user account security by enabling users to update their credentials regularly or in case they suspect their password has been compromised.

Key Components and Operations:

1. **UI Components**:
    - **EditText**:
        - `edtoldpass`: Input field for the user's current password.
        - `edtnewpass`: Input field for the user's new password.
        - `edtnewcpass`: Confirmation input field for the new password.
    - **Button**:
        - `btnsave`: Triggers the password change process when clicked.
2. **Database Interaction**:
    - Initializes or opens the SQLite database "Blood Donate" and ensures the 'register' table exists for storing user data.
    - Queries the database to verify the old password and updates it if the new passwords entered match and are confirmed.
3. **Validation and Error Handling**:
    - **Password Match Check**: Validates that the new password and its confirmation match. If not, it displays a toast message "Password Is Not Match".
    - **Current Password Verification**: Checks if the entered current password is correct by querying the database. Incorrect entries prompt a toast "Old Password Is Wrong".
    - **Field Completeness**: Verifies that all fields are filled. If any field is empty, it sets an error message on the respective EditText.
    - **Database Update**: If all checks pass, it updates the user's password in the database. Success and failure of this operation are communicated through toast messages "Password Changed Successfully" or "Password Not Changed".

Enhancing User Experience:

- Providing real-time validation feedback as the user types can help catch errors early, improving the overall user experience.
- Implementing a strength meter for the new password could encourage users to choose stronger, more secure passwords.
- Adding a visibility toggle for password fields would make it easier for users to enter their passwords correctly.

This fragment plays a crucial role in enhancing security for users of the Blood Donation app by ensuring they can manage their passwords effectively, thus protecting their accounts from unauthorized access.

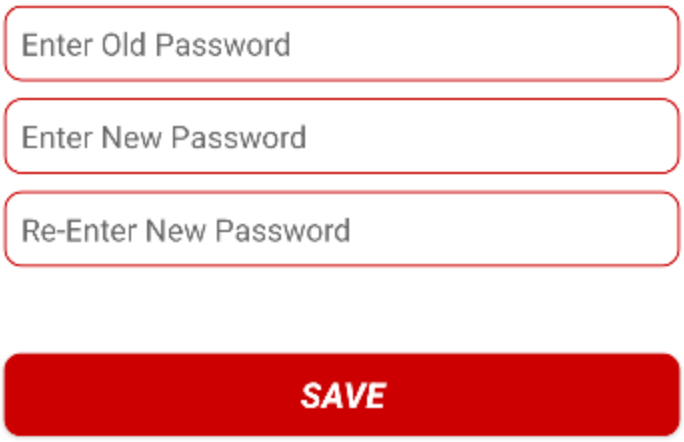
XML Code:

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

```
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="@android:color/white">

<EditText
    android:inputType="textPassword"
    android:id="@+id/change_pass_fragment_edt_old_pass"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="100dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:background="@drawable/custom_border"
    android:hint="Enter Old Password"
    android:padding="10dp" />

<EditText
    android:inputType="textPassword"
    android:id="@+id/change_pass_fragment_edt_new_pass"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/change_pass_fragment_edt_old_pass"
    android:layout_marginTop="10dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:background="@drawable/custom_border"
    android:hint="Enter New Password"
    android:padding="10dp" />

<EditText
```

```
        android:inputType="textPassword"
        android:id="@+id/change_pass_fragment_edt_cnew_pass"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/change_pass_fragment_edt_new_pass"
        android:layout_marginTop="10dp"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:background="@drawable/custom_border"
        android:hint="Re-Enter New Password"
        android:padding="10dp" />

    <Button
        android:id="@+id/change_pass_fragment_btn"
        android:layout_width="match_parent"
        android:background="@drawable/custom_btn"
        android:textColor="@android:color/white"
        android:textSize="20dp"
        android:textStyle="italic|bold"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_height="wrap_content"
        android:layout_below="@+id/change_pass_fragment_edt_cnew_pass"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="50dp"
        android:text="Save" />

</RelativeLayout>
```


Enter Old Password
Enter New Password
Re-Enter New Password
SAVE

Step 6 :

Implementing Forget Password Functionality in the Blood Donation Application:

This guide details the `ForgetPasswordFragment` used in the Blood Donation app. This fragment facilitates password recovery by using a security hint provided by the user during the registration process.

Overview: The `ForgetPasswordFragment` class extends `Fragment` and provides a user interface where users can enter a security hint to initiate password recovery. This is crucial for allowing users to regain access to their accounts if they forget their passwords.

Key Components and Operations:

1. **UI Components**:
    - **EditText**: `edthint` - where users enter their security hint.
    - **Button**: `btnnext` - users click this button to submit their hint and proceed with password recovery.
2. **Database Interaction**:
    - The fragment ensures the existence of the 'register' table in the SQLite database "Blood Donate" where user information, including the security hints, is stored.
    - Upon submission of a hint, the fragment queries this table to find a matching hint.
3. **Validation and Error Handling**:
    - **Hint Submission**: Checks if the hint field is empty. If it is, it prompts the user with an error message to enter the hint.
    - **Hint Verification**: Verifies if the entered hint matches any stored hints in the database. If a match is found, it transitions the user to the `ChangePass` activity to set a new password.
    - If no match is found, it displays a toast message "Hint Does Not Match!" indicating the failure to verify the user.

Enhancements for Better Security and User Experience:

- **Security Enhancements**: Implementing additional verification steps or using multi-factor authentication can enhance security during password recovery.
- **User Interface Improvements**: Offering hints or suggestions based on partial hint entries could improve user experience and aid in memory recall.
- **Error Feedback**: Providing more specific feedback about why a hint might not match (e.g., hint format incorrect or hint not recognized) could make the process more user-friendly.

This fragment is essential for ensuring that users can recover access to their accounts securely and efficiently, leveraging previously provided personal security data. It plays a vital role in maintaining user trust and operational integrity for the Blood Donation app.

XML Code:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:background="@android:color/white"

   android:layout_width="match_parent"
```

```
android:layout_height="match_parent">

<TextView
    android:id="@+id/forget_pass_txt"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="50dp"
    android:text="Blood Donate"
    android:textColor="@android:color/black"
    android:textSize="20dp"
    android:textStyle="bold|italic" />

<EditText
    android:id="@+id/forget_pass_edt_hint"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/forget_pass_txt"
    android:layout_marginLeft="10dp"
    android:background="@drawable/custom_border"
    android:padding="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="50dp"
    android:hint="@string/hint" />

<Button
    android:id="@+id/forget_pass_btn"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/forget_pass_edt_hint"
    android:layout_marginTop="40dp"
```

```
        android:background="@drawable/custom_btn"
        android:text="Next"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:textColor="@android:color/white"
        android:textSize="20dp"
        android:textStyle="bold|italic" />

</RelativeLayout>
```

Blood Donate

Enter Your Nick Name

NEXT

Java Code:

```
package blood.donate.Fragment;


import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import blood.donate.Activity.ChangePass;
import blood.donate.Constans;
import blood.donate.R;

import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.fragment.app.Fragment;

/**
 * A simple {@link Fragment} subclass.
 */
public class ForgetPasswordFragment extends Fragment {
```

```
    EditText edthint;
    Button btnnext;
    SQLiteDatabase db;
    SharedPreferences sp;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View view = inflater.inflate(R.layout.fragment_forget_password, container, false);
        sp = getActivity().getSharedPreferences(Constans.PREF,Context.MODE_PRIVATE);
        db = getActivity().openOrCreateDatabase("Blood Donate", Context.MODE_PRIVATE, null);
        String tableQuery = "create table if not exists register(id integer primary key autoincrement, name
text,email text,pass text,cno number,dob text,hint text,bloodgroup text)";
        db.execSQL(tableQuery);


        edthint = (EditText) view.findViewById(R.id.forget_pass_edt_hint);
        btnnext = (Button) view.findViewById(R.id.forget_pass_btn);



        btnnext.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(edthint.getText().toString().equals("")){
                    edthint.setError("Enter Hint");
                }
                else{

                    Cursor c = db.rawQuery("select * from register where hint='"
                            + edthint.getText().toString() + "'", null);
```

```
            if (c.getCount()>0) {
                sp.edit().putString(Constans.HINT,edthint.getText().toString()).commit();
                startActivity(new Intent(getActivity(), ChangePass.class));

            }
            else{
                Toast.makeText(getActivity(), "Hint Does Not Match!", Toast.LENGTH_SHORT).show();
            }
        }
    }
});

  return view;
  }
}
```

Step 7:

Implementing Registration Functionality in the Blood Donation Application:

The guide below provides an overview of the `SignupFragment` used in the Blood Donation app. This fragment is critical for capturing new user data and securely storing it within the app.

Overview: The `SignupFragment` extends `Fragment` and offers a user interface for new users to register. It collects essential information such as name, email, password, contact number, birth date, security hint, and blood group.

Key Components and Operations:

1. **UI Components**:

    - **EditTexts**: Collect various user inputs like name, email, password, contact number, birth date, and hint.
    - **Spinner**: Allows users to select their blood type from a predefined list.
    - **RadioGroup with RadioButtons**: Enables users to select their gender.
    - **Button**: `btnsignup` is used to trigger the registration process after all fields are filled.
2. **Database Interaction**:
    - The fragment creates or opens the SQLite database "Blood Donate" and ensures the 'register' table exists.
    - It inserts the new user's data into the database after validation checks pass.
3. **Validation and Error Handling**:
    - **Email Validation**: Checks if the email is entered and matches a predefined pattern. If not, it shows a toast message "Invalid Email".
    - **Password Matching**: Ensures the password and confirmation password are the same. If they don't match, a custom message dialog box shows "Password Does Not Match".
    - **Mandatory Fields**: Verifies that all fields are filled out. If any field is left empty, it sets an error on the respective field.
4. **User Feedback**:
    - Gender selection feedback is provided immediately with a toast message showing the selected gender.
    - Upon successful registration, the app redirects the user to the `HomeActivity`, and user data is saved into `SharedPreferences`.

Enhancements for Better User Experience:

- **Password Strength Meter**: Could be added to guide users in creating a stronger password.
- **Dynamic Validation Feedback**: Providing real-time feedback as users fill out the form can prevent errors at the submission point.
- **User Interface Improvements**: Enhancing the visual layout and interactive elements like custom dropdowns for blood types and date pickers for birth dates.

This fragment plays a crucial role in the Blood Donation app by ensuring user data is captured accurately and stored securely, facilitating a smooth onboarding process for new users.

XML Code :

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:background="@android:color/white">
```

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">


    <TextView
        android:id="@+id/signup_txt_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="30dp"
        android:text="Blood Donate"
        android:textColor="@android:color/black"
        android:textSize="20dp"
        android:textStyle="bold|italic" />


    <EditText
        android:id="@+id/signup_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/signup_txt_title"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_marginTop="30dp"
        android:background="@drawable/custom_border"
        android:hint="@string/name"
        android:padding="10dp" />
```

```
<EditText
    android:id="@+id/signup_email"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/signup_name"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="10dp"
    android:background="@drawable/custom_border"
    android:hint="@string/email"
    android:padding="10dp" />


<EditText
    android:inputType="textPassword"
    android:id="@+id/signup_pass"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/signup_email"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="10dp"
    android:background="@drawable/custom_border"
    android:hint="@string/pass"
    android:padding="10dp" />


<EditText
    android:inputType="textPassword"
    android:id="@+id/signup_confirm_pass"
    android:layout_width="match_parent"
```

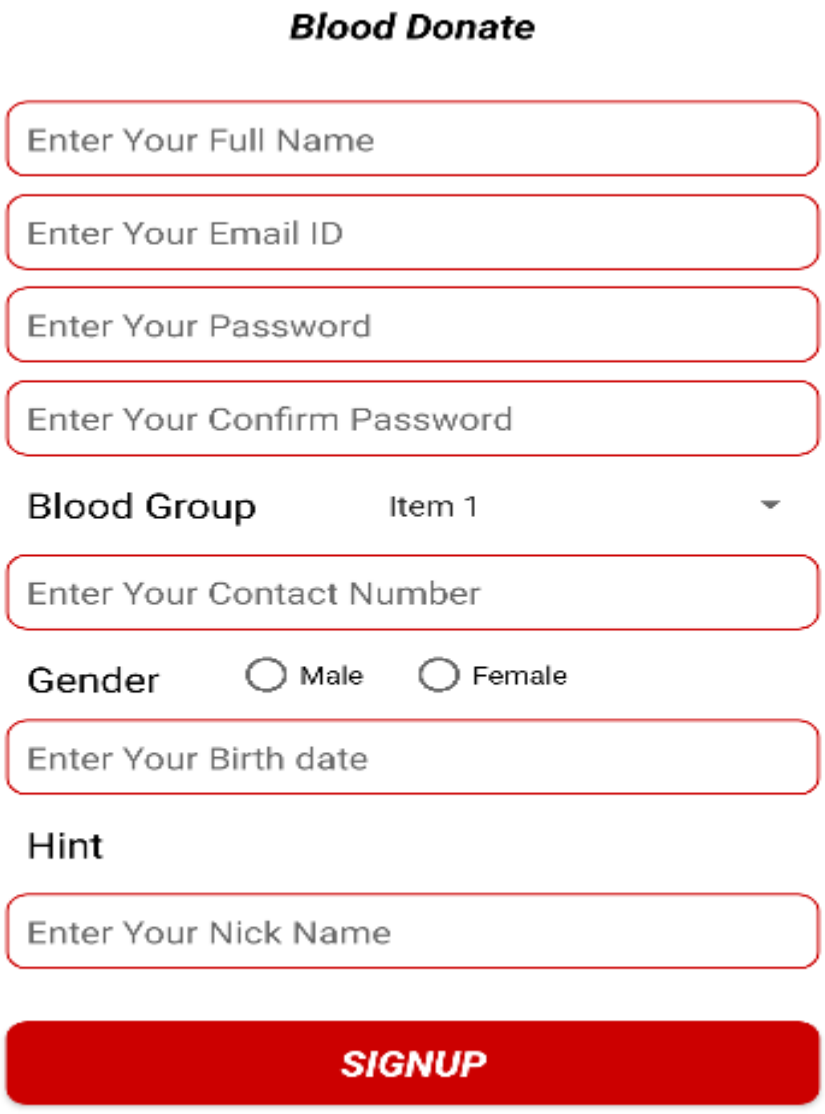
```
    android:layout_height="wrap_content"
    android:layout_below="@+id/signup_pass"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="10dp"
    android:background="@drawable/custom_border"
    android:hint="@string/cpass"
    android:padding="10dp" />


<TextView
    android:id="@+id/signup_blood_grp_txt"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/signup_confirm_pass"
    android:layout_marginLeft="15dp"
    android:layout_marginTop="10dp"
    android:padding="5dp"
    android:text="Blood Group"
    android:textColor="@android:color/black"
    android:textSize="20dp" />


<Spinner
    android:id="@+id/signup_sp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/signup_blood_grp_txt"
    android:layout_alignParentRight="true"
    android:layout_alignTop="@+id/signup_blood_grp_txt"
    android:layout_marginLeft="50dp"
```

```
    android:layout_marginRight="10dp"
    android:layout_toRightOf="@+id/signup_blood_grp_txt" />


<EditText
    android:id="@+id/signup_contact_nomber"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/signup_blood_grp_txt"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="10dp"
    android:background="@drawable/custom_border"
    android:hint="@string/cno"
    android:inputType="phone"
    android:maxLength="10"
    android:padding="10dp" />


<TextView
    android:id="@+id/signup_gender_txt"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/signup_contact_nomber"
    android:layout_marginLeft="15dp"
    android:layout_marginTop="10dp"
    android:padding="5dp"
    android:text="Gender"
    android:textColor="@android:color/black"
    android:textSize="20dp" />
```

```
<RadioGroup
    android:id="@+id/signup_rg"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/signup_gender_txt"
    android:layout_marginLeft="30dp"
    android:layout_toRightOf="@+id/signup_gender_txt"
    android:orientation="horizontal">


    <RadioButton
        android:id="@+id/signup_male_rb"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Male" />


    <RadioButton
        android:id="@+id/signup_female_rb"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="20dp"
        android:text="Female" />


</RadioGroup>
```

```
<EditText
    android:id="@+id/signup_date_of_birth"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/signup_rg"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="10dp"
    android:background="@drawable/custom_border"
    android:hint="@string/date"
    android:padding="10dp" />


<TextView
    android:id="@+id/signup_hint_txt"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/signup_date_of_birth"
    android:layout_marginLeft="15dp"
    android:layout_marginTop="10dp"
    android:padding="5dp"
    android:text="Hint"
    android:textColor="@android:color/black"
    android:textSize="20dp" />


<EditText
    android:id="@+id/signup_hint"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/signup_hint_txt"
```

```
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_marginTop="10dp"
        android:background="@drawable/custom_border"
        android:hint="@string/hint"
        android:padding="10dp" />



    <Button
        android:id="@+id/signup_btn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/signup_hint"
        android:layout_marginBottom="20dp"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_marginTop="30dp"
        android:background="@drawable/custom_btn"
        android:text="signup"
        android:textColor="@android:color/white"
        android:textSize="20dp"
        android:textStyle="italic|bold" />



  </RelativeLayout>
</ScrollView>
```


Blood Donate
Enter Your Full Name
Enter Your Email ID
Enter Your Password
Enter Your Confirm Password
Blood Group
Item 1
Enter Your Contact Number
Gender
Male
Female
Enter Your Birth date
Hint
Enter Your Nick Name
SIGNUP

Java Code:

```
package blood.donate.Fragment;


import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import blood.donate.Activity.HomeActivity;
import blood.donate.Constans;
import blood.donate.MyMessage;
import blood.donate.R;

import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Spinner;
import android.widget.Toast;

import androidx.fragment.app.Fragment;

/**
 * A simple {@link Fragment} subclass.
 */
```

```
public class SignupFragment extends Fragment {


    EditText edtname,edtemail,edtpass,edtcpass,edtcno,edtbdate,edthint;

    Spinner sp_signup;

    RadioGroup rg_signup;

    RadioButton rb_male,rb_female;

    Button btnsignup;

    SharedPreferences sp;



    ArrayAdapter sp_adapter;

    String sp_str[] = {"A Positive","A Negative","B Positive","B Negative","AB Positive","AB Negative","O Positive","O Negative"};

    String emailPattern = "[a-zA-Z0-9._-]+@[a-z]+\\.+[a-z]+";

    SQLiteDatabase db;

    Cursor c;

    MyMessage message;

    String email;

    String sBloodGroup;


    @Override

    public View onCreateView(LayoutInflater inflater, ViewGroup container,

                             Bundle savedInstanceState) {

        // Inflate the layout for this fragment

        View view = inflater.inflate(R.layout.fragment_signup, container, false);

        sp = getActivity().getSharedPreferences(Constans.PREF, Context.MODE_PRIVATE);


        db = getActivity().openOrCreateDatabase("Blood Donate", Context.MODE_PRIVATE, null);

        String tableQuery = "create table if not exists register(id integer primary key autoincrement, name text,email text,pass text,cno number,dob text,hint text,bloodgroup text)";

        db.execSQL(tableQuery);
```

```
edtname = (EditText)view.findViewById(R.id.signup_name);
edtemail  = (EditText)view.findViewById(R.id.signup_email);
edtpass  = (EditText)view.findViewById(R.id.signup_pass);
edtcpass  = (EditText)view.findViewById(R.id.signup_confirm_pass);
edtcno  = (EditText)view.findViewById(R.id.signup_contact_nomber);
edtbdate  = (EditText)view.findViewById(R.id.signup_date_of_birth);
edthint = (EditText) view.findViewById(R.id.signup_hint);

sp_signup = (Spinner) view.findViewById(R.id.signup_sp);

rg_signup = (RadioGroup) view.findViewById(R.id.signup_rg);

rb_male = (RadioButton) view.findViewById(R.id.signup_male_rb);
rb_female = (RadioButton) view.findViewById(R.id.signup_female_rb);

btnsignup = (Button) view.findViewById(R.id.signup_btn);

sp_adapter = new ArrayAdapter(getActivity(),android.R.layout.simple_list_item_1,sp_str);
sp_adapter.setDropDownViewResource(android.R.layout.simple_list_item_checked);
sp_signup.setAdapter(sp_adapter);

sp_signup.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
   @Override
   public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {
      sBloodGroup = sp_str[i];
   }

   @Override
   public void onNothingSelected(AdapterView<?> adapterView) {
```

```
    }
});

rg_signup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        if(rb_male.isChecked()){
            Toast.makeText(getActivity(), rb_male.getText(), Toast.LENGTH_SHORT).show();

        }
        else{
            Toast.makeText(getActivity(), rb_female.getText(), Toast.LENGTH_SHORT).show();
        }
    }
});
message = new MyMessage(getActivity());

btnsignup.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Validation();

        email = edtemail.getText().toString();
    }

    private void Validation() {

        if(edtemail.getText().toString().equals("")){
            edtemail.setError("Enter Email ID");
        }
```

```
String email = edtemail.getText().toString();
if (email.matches(emailPattern)) {
} else {
   Toast.makeText(getActivity(), "In Valid Email", Toast.LENGTH_SHORT).show();
   return;
}

if(edtpass.getText().toString().equals("")){
   edtpass.setError("Enter Your Password");
}

String pwd=edtpass.getText().toString();
String pwd1=edtcpass.getText().toString();
if(pwd.equals(pwd1)){
   //Toast.makeText(getActivity(), "match", Toast.LENGTH_SHORT).show();
}else {
   message.myMsg("Oops!","Password Does Not Match");
     //Toast.makeText(getActivity(), "not match", Toast.LENGTH_SHORT).show();
return;
}

if(edtname.getText().toString().equals("")){
   edtname.setError("Enter Name");
}
if(edtbdate.getText().toString().equals("")){
   edtbdate.setError("Enter Birth Date");
}
if(edtcno.getText().toString().equals("")){
   edtcno.setError("Enter Contact Number");
}
if(edthint.getText().toString().equals("")){
```

```
                edthint.setError("Enter Hint");
            }
            else{
                String insert_Query = "insert into register values (?, '"+ edtname.getText().toString() + "','" + edtemail.getText().toString() + "','" + edtpass.getText().toString() + "','" + edtcno.getText().toString() + "','" + edtbdate.getText().toString() + "','" + edthint.getText().toString() + "','"+sBloodGroup+"')";
                db.execSQL(insert_Query);


                Intent i = new Intent(getActivity(), HomeActivity.class);


                String email_sp=edtemail.getText().toString();
                sp.edit().putString(Constans.USERNAME,email_sp).commit();
                startActivity(i);
                clear();
            }
        }
    });
    return view;
  }


  private void clear() {
    edtname.setText("");
    edtemail.setText("");
    edtpass.setText("");
    edtcpass.setText("");
    edtcno.setText("");
    edtbdate.setText("");
    edthint.setText("");
  }
}
```

Step 8 :

Implementing View Profile Functionality in the Blood Donation Application:

The guide below outlines the `ViewProfileFragment` used in the Blood Donation app. This fragment is essential for displaying the logged-in user's profile information, providing a clear view of their stored details.

Overview: The `ViewProfileFragment` class extends `Fragment` and offers a user interface that allows users to view their stored profile data, including name, email, password, contact number, birth date, and security hint.

Key Components and Operations:

1. **UI Components**:
    - **TextViews**: Used to display the user's profile data (name, email, password, contact number, birth date, and hint).
2. **Database Interaction**:
    - The fragment initializes or opens the SQLite database "Blood Donate" and ensures the 'register' table exists.
    - It retrieves and displays the profile information from the database.
3. **Data Retrieval**:
    - A SQL query is executed to fetch all entries from the 'register' table.
    - If the cursor finds data, it reads and sets the profile information to the respective TextViews.
    - If no data is found, a toast message "data not found.." is displayed, indicating the absence of stored user data.

Considerations and Enhancements:

- **Security Concerns**: Displaying sensitive information such as the password directly in the app might pose security risks. It's generally advisable to either not display passwords or to show them in a masked format.
- **Error Handling**: More robust error handling can be implemented to manage database errors or cases where the database might be inaccessible.

- **User Experience**: Enhancing the layout with better visual separation and labeling of the profile data could improve readability and user interaction.
- **Data Privacy**: Consider adding functionality that allows users to choose which details they want to display for enhanced privacy.

This fragment plays a crucial role in the Blood Donation app by enabling users to verify their stored information, ensuring transparency and trust in the app's data handling practices.

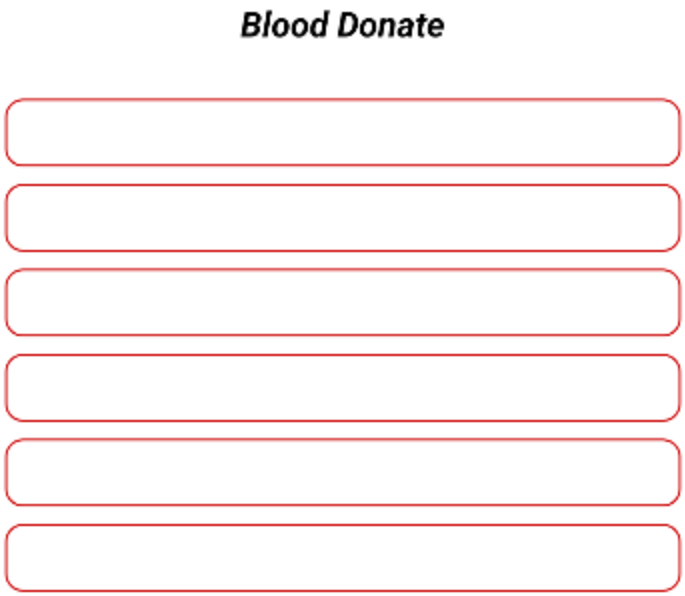
Java Code:

```
package blood.donate.Fragment;

import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import android.widget.Toast;

import androidx.fragment.app.Fragment;

import blood.donate.R;

/**
 * A simple {@link Fragment} subclass.
 */
public class ViewProfileFragment extends Fragment {
```

```
    TextView txtname,txtemail,txtpass,txtcno,txtbdate,txthint;


    private SQLiteDatabase db;


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View view = inflater.inflate(R.layout.fragment_view_profile, container, false);


        db = getActivity().openOrCreateDatabase("Blood Donate", Context.MODE_PRIVATE, null);
        String tableQuery = "create table if not exists register(id integer primary key autoincrement, name
text,email text,pass text,cno number,dob text,hint text,bloodgroup text)";
        db.execSQL(tableQuery);


        txtname = (TextView) view.findViewById(R.id.view_profile_name);
        txtemail = (TextView)  view.findViewById(R.id.view_profile_email);
        txtpass = (TextView)  view.findViewById(R.id.view_profile_pass);
        txtbdate = (TextView)  view.findViewById(R.id.view_profile_date_of_birth);
        txtcno = (TextView)  view.findViewById(R.id.view_profile_contact_nomber);
        txthint = (TextView)  view.findViewById(R.id.view_profile_hint);


        String QueryDisplay = "select * from register";
        Cursor c = db.rawQuery(QueryDisplay, null);
        StringBuilder sb = new StringBuilder();
        if (c.moveToFirst()) {
            do {
                String name = c.getString(0);
                String email = c.getString(1);
                String pass = c.getString(2);
                String bdate = c.getString(3);
```

```
            String cno = c.getString(4);
            String hint = c.getString(5);
            txtname.setText(name);
            txtemail.setText(email);
            txtpass.setText(pass);
            txtbdate.setText(bdate);
            txtcno.setText(cno);
            txthint.setText(hint);
        } while (c.moveToNext());
        //Toast.makeText(getActivity(), "" + sb, Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(getActivity(), "data not found..", Toast.LENGTH_SHORT).show();
    }

return view;
}

/*protected void openDatabase() {
    db = getActivity().openOrCreateDatabase("SagarDB", Context.MODE_PRIVATE, null);
}*/

/*private void showRecords() {
}

protected void moveNext() {
    if (!c.isLast())
        c.moveToNext();
    showRecords();
}

protected void movePrev() {
```

```
        if (!c.isFirst())
            c.moveToPrevious();
        showRecords();
    }
*/

}
```



Step 9 :

Implementing Main Activity with Navigation Drawer in the Blood Donation Application:

This guide will walk you through the `MainActivity` of the Blood Donation app, which is central to navigating through various parts of the application using a navigation drawer.

Overview: The `MainActivity` class extends `AppCompatActivity` and serves as the hub for navigating to different fragments such as `LoginFragment`, `SignupFragment`, `ForgetPasswordFragment`, and `GuestReceiveFragment` via a navigation drawer.

Key Components and Operations:

1. **Toolbar and Drawer Layout**:
    - **Toolbar**: Acts as the app bar at the top of the screen, providing a familiar and consistent user interface element across Android apps.
    - **DrawerLayout**: A layout that acts as a parent container for the application, allowing the navigation drawer to slide in and out.
2. **ActionBarDrawerToggle**:
    - This toggle handles the opening and closing of the navigation drawer when the user taps on the hamburger icon on the toolbar.
3. **NavigationView**:
    - Manages the items within the navigation drawer, each corresponding to different sections of the app. It uses a listener to handle item selection events.
4. **Fragment Management**:
    - Based on the navigation item selected, the `MainActivity` dynamically replaces the content area with the appropriate fragment using `FragmentManager`.
5. **Back Press Handling**:
    - Customizes the back button behavior. If the navigation drawer is open, it closes the drawer. If the user is on a fragment other than the main one and presses the back button, the fragment manager handles the back navigation, otherwise, it prompts the user with an exit confirmation dialog.

Code Snippets:

Navigation Drawer Toggle:
java
Copy code
```
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
        this, drawer, toolbar, R.string.navigation_drawer_open,
R.string.navigation_drawer_close);
drawer.setDrawerListener(toggle);
toggle.syncState();
```

-

Handling Navigation Item Selections:
java
Copy code
```
public boolean onNavigationItemSelected(MenuItem item) {
    int id = item.getItemId();
```

```
    Fragment fm = null;
    switch (id) {
        case R.id.nav_login:
            fm = new LoginFragment();
            break;
        case R.id.nav_signup:
            fm = new SignupFragment();
            break;
        case R.id.nav_forget_password:
            fm = new ForgetPasswordFragment();
            break;
        case R.id.nav_guest_receiver:
            fm = new GuestReceiveFragment();
            break;
        default:
            break;
    }
    if (fm != null) {

getSupportFragmentManager().beginTransaction().replace(R.id.main_content,
fm).addToBackStack(null).commit();
    }
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}
```

-

Exit Confirmation Dialog:
java
Copy code

```
new AlertDialog.Builder(this)
    .setTitle("Blood Donate")
    .setMessage("Are you sure you want to exit?")
    .setNegativeButton("No", null)
    .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface arg0, int arg1) {
            MainActivity.super.onBackPressed();
        }
    }).create().show();
```

-

Enhancements for Better User Experience:

- **Visual Feedback**: Providing visual feedback (like color change or icon animation) when selecting items can enhance the user's interaction with the navigation drawer.
- **Fragment Transition Animations**: Adding animations to fragment transitions can make the navigation experience more fluid and engaging.

This `MainActivity` setup is crucial for a cohesive user experience in the Blood Donation app, enabling efficient navigation across different functionalities within the app.

Java Code:

package blood.donate.Activity;

import android.app.AlertDialog;

import android.content.DialogInterface;

import android.content.Intent;

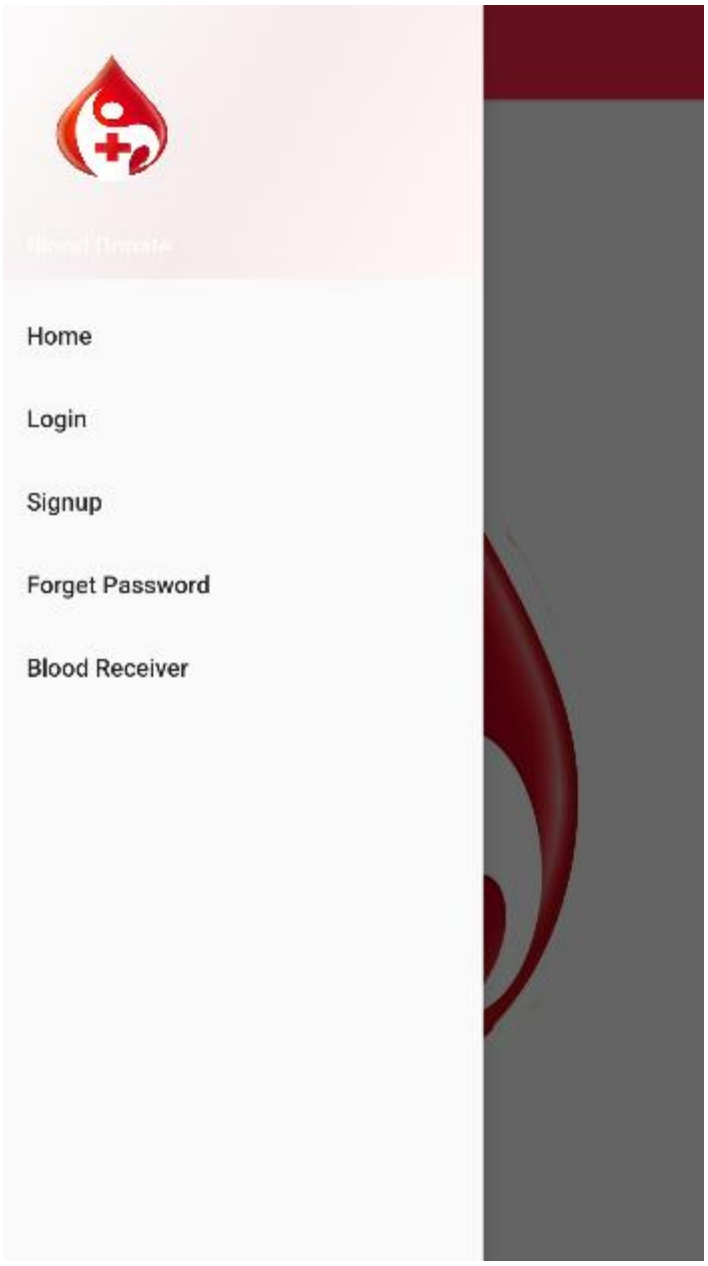
```
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

import androidx.appcompat.app.ActionBarDrawerToggle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;
import androidx.core.view.GravityCompat;
import androidx.drawerlayout.widget.DrawerLayout;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;

import com.google.android.material.navigation.NavigationView;

import blood.donate.Fragment.ForgetPasswordFragment;
import blood.donate.Fragment.LoginFragment;
import blood.donate.GuestReceiveFragment;
import blood.donate.Fragment.SignupFragment;
import blood.donate.R;

public class MainActivity extends AppCompatActivity
        implements NavigationView.OnNavigationItemSelectedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
```

```
    ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
            this, drawer, toolbar, R.string.navigation_drawer_open, R.string.navigation_drawer_close);
    drawer.setDrawerListener(toggle);
    toggle.syncState();

    NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
    navigationView.setNavigationItemSelectedListener(this);
}

@Override
public void onBackPressed() {
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
    } else {
        //super.onBackPressed();

        FragmentManager manager = getSupportFragmentManager();
        if (manager.getBackStackEntryCount() > 1 ) {
            // If there are back-stack entries, leave the FragmentActivity
            // implementation take care of them.
            manager.popBackStack();

        } else {
            // Otherwise, ask user if he wants to leave :)
            new AlertDialog.Builder(this)
                    .setTitle("Blood Donate")
                    .setMessage("Are you sure you want to exit?")
                    .setNegativeButton("No", null)
                    .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
```

```
                    public void onClick(DialogInterface arg0, int arg1) {
                        MainActivity.super.onBackPressed();
                    }
                }).create().show();
        }
    }


}

/*@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}*/
```

```
    @SuppressWarnings("StatementWithEmptyBody")
    @Override
    public boolean onNavigationItemSelected(MenuItem item) {
        // Handle navigation view item clicks here.
        int id = item.getItemId();

        if (id == R.id.nav_home) {
            // Handle the camera action
            startActivity(new Intent(MainActivity.this,MainActivity.class));
        } else if (id == R.id.nav_login) {

            Fragment fm = new LoginFragment();
            FragmentManager manager = getSupportFragmentManager();
manager.beginTransaction().replace(R.id.main_content,fm).addToBackStack("MainActivity").commit();

        } else if (id == R.id.nav_signup) {

            Fragment fm = new SignupFragment();
            FragmentManager manager = getSupportFragmentManager();
manager.beginTransaction().replace(R.id.main_content,fm).addToBackStack("MainActivity").commit();

        } else if (id == R.id.nav_forget_password) {

            Fragment fm = new ForgetPasswordFragment();
            FragmentManager manager = getSupportFragmentManager();
manager.beginTransaction().replace(R.id.main_content,fm).addToBackStack("MainActivity").commit();

        }
```

```
        else if (id == R.id.nav_guest_receiver) {

            Fragment fm = new GuestReceiveFragment();
            FragmentManager manager = getSupportFragmentManager();

manager.beginTransaction().replace(R.id.main_content,fm).addToBackStack("MainActivity").commit();

        }

        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
        drawer.closeDrawer(GravityCompat.START);
        return true;
    }
}
```



Step 10 :

Implementing Guest Blood Type Selection in the Blood Donation Application:

This guide describes the `GuestReceiveFragment` used in the Blood Donation app. This fragment facilitates the selection of a guest user's blood type and initiates navigation based on the selected type.

Overview: The `GuestReceiveFragment` extends `Fragment` and offers a user interface that allows guest users (non-registered) to select their blood type from a set of radio buttons. This selection is used to direct the guest to appropriate activities within the app that are specific to their blood type.

Key Components and Operations:

1. **UI Components**:
   - **RadioGroup**: Contains multiple `RadioButton` instances for each blood type.
   - **RadioButtons**: Individual buttons for each blood type (A+, A-, B+, B-, AB+, AB-, O+, O-).
2. **SharedPreferences**:
   - Used to store the selected blood type temporarily, which can be retrieved in subsequent activities or operations.
3. **Database Interaction**:
   - Ensures the 'register' table exists in the SQLite database "Blood Donate," although it seems not directly used in this fragment.
4. **Event Handling**:
   - The `OnCheckedChangeListener` for the `RadioGroup` captures the selection event, retrieves the text from the selected `RadioButton`, and stores it in `SharedPreferences`.
   - An intent is then used to navigate to an activity (`ApActivity`), presumably configured to handle different blood types.

Code Snippet Example:

Setting Up RadioGroup Listener:
java
Copy code

```
grp_rg.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {

    @Override

    public void onCheckedChanged(RadioGroup group, int checkedId) {

        RadioButton rb = view.findViewById(checkedId);

        sp.edit().putString(Constans.BLOOD_GROUP,
rb.getText().toString()).commit();

        startActivity(new Intent(getActivity(), ApActivity.class));

    }

});
```

-

Enhancements for Better Functionality and User Experience:

- **Dynamic Activity Navigation**: Instead of directing all selections to a single activity, consider mapping each blood type to a specific activity tailored to provide relevant information or services for that type.
- **Visual Feedback**: Provide immediate visual feedback upon selection, such as highlighting or a brief confirmation message, to reassure the user that the selection has been acknowledged.
- **Use of Shared Preferences**: While practical for passing simple data, consider using more robust state management or passing data directly through intents if sensitive or complex data structures are involved.

This fragment is crucial for enabling guest users to interact with the Blood Donation app by selecting their blood type, which helps tailor the app's functionality to meet their specific needs or conditions.

Java Code :

```
package blood.donate;


import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.RadioButton;
import android.widget.RadioGroup;

import androidx.fragment.app.Fragment;
```

```
/**
 * A simple {@link Fragment} subclass.
 */
public class GuestReceiveFragment extends Fragment {

    RadioGroup grp_rg;
    RadioButton grp_Apositive,grp_Anegative,grp_Bpositive,grp_Bnegative
            ,grp_Abpositive,grp_Abnegative,grp_Opositive,grp_Onegative;

    SharedPreferences sp;
    SQLiteDatabase db;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View view = inflater.inflate(R.layout.fragment_guest_receive, container, false);
        sp = getActivity().getSharedPreferences(Constans.PREF, Context.MODE_PRIVATE);

        db = getActivity().openOrCreateDatabase("Blood Donate", Context.MODE_PRIVATE, null);
        String tableQuery = "create table if not exists register(id integer primary key autoincrement, name
text,email text,pass text,cno number,dob text,hint text,bloodgroup text)";
        db.execSQL(tableQuery);

        grp_rg = (RadioGroup) view.findViewById(R.id.grp_rg);
        grp_Apositive = (RadioButton) view.findViewById(R.id.grp_APositive);
        grp_Anegative = (RadioButton) view.findViewById(R.id.grp_ANegative);
        grp_Bpositive = (RadioButton) view.findViewById(R.id.grp_BPositive);
        grp_Bnegative = (RadioButton) view.findViewById(R.id.grp_BNegative);
        grp_Abpositive = (RadioButton) view.findViewById(R.id.grp_ABPositive);
```

```
    grp_Abnegative = (RadioButton) view.findViewById(R.id.grp_ABNegative);
    grp_Opositive = (RadioButton) view.findViewById(R.id.grp_OPositive);
    grp_Onegative = (RadioButton) view.findViewById(R.id.grp_ONegative);

    grp_rg.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
      @Override
      public void onCheckedChanged(RadioGroup group, int checkedId) {
        RadioButton rb = view.findViewById(checkedId);
        sp.edit().putString(Constans.BLOOD_GROUP,rb.getText().toString()).commit();
        startActivity(new Intent(getActivity(), ApActivity.class));
        /*if(grp_Apositive.isChecked()){
          Toast.makeText(getActivity(), grp_Apositive.getText().toString(),
Toast.LENGTH_SHORT).show();
          startActivity(new Intent(getActivity(), ApActivity.class));
        }
        else if(grp_Anegative.isChecked()){
          Toast.makeText(getActivity(), grp_Anegative.getText().toString(),
Toast.LENGTH_SHORT).show();
          startActivity(new Intent(getActivity(), BloodANegataive.class));
        }
        else if(grp_Bpositive.isChecked()){
          Toast.makeText(getActivity(), grp_Bpositive.getText().toString(),
Toast.LENGTH_SHORT).show();
          startActivity(new Intent(getActivity(), BloodBPositive.class));
        }
        else if(grp_Bnegative.isChecked()){
          Toast.makeText(getActivity(), grp_Bnegative.getText().toString(),
Toast.LENGTH_SHORT).show();
          startActivity(new Intent(getActivity(), BloodBNegative.class));
        }
        else if(grp_Abpositive.isChecked()){
```

```
                Toast.makeText(getActivity(), grp_Abpositive.getText().toString(),
Toast.LENGTH_SHORT).show();

                startActivity(new Intent(getActivity(), BloodABPositive.class));

            }

            else if(grp_Abnegative.isChecked()){

                Toast.makeText(getActivity(), grp_Abnegative.getText().toString(),
Toast.LENGTH_SHORT).show();

                startActivity(new Intent(getActivity(), BloodABNegative.class));

            }

            else if(grp_Opositive.isChecked()){

                Toast.makeText(getActivity(), grp_Opositive.getText().toString(),
Toast.LENGTH_SHORT).show();

                startActivity(new Intent(getActivity(), BloodOPositive.class));

            }

            else if(grp_Onegative.isChecked()){

                Toast.makeText(getActivity(), grp_Opositive.getText().toString(),
Toast.LENGTH_SHORT).show();

                startActivity(new Intent(getActivity(), BloodONegative.class));

            }*/

        }

    });

    return view;

  }


}
```

- ◯ A Positive
- ◯ A Negative
- ◯ B Positive
- ◯ B Negative
- ◯ AB Positive
- ◯ AB Negative
- ◯ O Positive
- ◯ O Negative