

KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATICS

T120B169 App Development for Smart Mobile Systems

*FitMax*

|  |
| --- |
| *IFZm-1, Ernestas Kuprys:* |
| Date: *2024.05.27* |

Kaunas, 2024

Tables of Contents

[Description of Your app 3](#_Toc167726872)

[Functionality of your app 4](#_Toc167726873)

[List of functions 14](#_Toc167726874)

[Solution 15](#_Toc167726875)

[Task #1. Create a database to store user data. 15](#_Toc167726876)

[Task #2. Create a functional sing up system that checks if fields are properly filed. 18](#_Toc167726877)

[Task #3. Add a functioning login system that stores user session. 20](#_Toc167726878)

[Task #4. (Defense) Add a tracker for every time a user logs in inside the profile tab. 22](#_Toc167726879)

[Task #5. Store physical activity and plan information. 25](#_Toc167726880)

[Task #6. Create a questionnaire that stores user preferences. 28](#_Toc167726881)

[Task #7. Display daily tasks based off selected plan. 31](#_Toc167726882)

[Task #8. Add functioning toolbar to activities. 34](#_Toc167726883)

[Task #9. (Defense) Let user mark completed activities. 35](#_Toc167726884)

[Task #10. Display a working calendar that lets users go back and forwards by months. 39](#_Toc167726885)

[Task #11. Add a daily step counter. 42](#_Toc167726886)

[Task #12. Add a circular progress bar to show step count progress. 43](#_Toc167726887)

[Task #13. Let the user see past complete and incomplete activities/steps by clicking on calendar cells. 44](#_Toc167726888)

[Task #14. Display weekly calorie loss statistics as a graph. 45](#_Toc167726889)

[Task #15. Let the user rest daily activity and step count progress. 47](#_Toc167726890)

[Task #16. Let the user change their user data (weight, plan, step count). 49](#_Toc167726891)

[Task #17. Let the user permanently delete their account. 51](#_Toc167726892)

[Task #18. Remove the need for users to sign in if they have not logged out from their previous session. 53](#_Toc167726893)

[Task #19. (Defense) Display upcoming activities in calendar view. 54](#_Toc167726894)

[Reference list 55](#_Toc167726895)

# **Description of Your app**

1. What type is your application/game?
   * *Fitness Tracker App - allows users to track their daily physical activities, set fitness goals, and monitor their progress over time:*
2. Description.
   * **User Interface Design:**
     + Design a user-friendly interface with screens for tracking different types of physical activities, such as running, cycling, or weightlifting.
     + Include screens for setting fitness goals, viewing activity history, and monitoring progress.
   * **Data Storage:**
     + Use SQLite database or Room Persistence Library to store and retrieve activity data, user profiles, and fitness goals persistently.
     + Implement functions to save and update activity data, user profiles, and fitness goals.
   * **User Input Handling:**
     + Implement features for users to input their daily physical activities, including duration, distance, intensity, and calories burned.
     + Include options for users to manually enter activity data or sync data from external fitness tracking devices or apps.
   * **Graphical Data Visualization:**
     + Use charts or graphs to visually represent users' activity data, progress towards fitness goals, and trends over time.
     + Implement features for users to view their activity history, track changes in performance, and identify areas for improvement.
   * **Goal Setting and Monitoring:**
     + Allow users to set personalized fitness goals, such as daily step count, weekly running distance, or monthly weightlifting targets.
     + Implement features for users to track their progress towards fitness goals, receive notifications or reminders to stay on track, and adjust goals as needed.
   * **Social Sharing and Community Features:**
     + Include social sharing functionality to allow users to share their fitness achievements, progress updates, and workout routines with friends or on social media platforms.
     + Implement community features such as leaderboards, challenges, or virtual fitness groups to encourage interaction and motivation among users.
   * **Additional Features:**
     + Integrate with external APIs or services to provide additional features such as weather forecasts for outdoor activities, nutritional information for calorie tracking, or workout recommendations based on user preferences.
     + Implement features for users to track other health metrics such as sleep quality, heart rate, or body measurements.

# **Functionality of your app**

1. Press “Start” in the main menu:

A screen shot of a phone

Description automatically generated

1. Click on “sign up” to create an account:

A screen shot of a phone

Description automatically generated

1. Fill in the form:

A screenshot of a phone

Description automatically generated

1. Log in with new account:

A screen shot of a phone

Description automatically generated

1. Fill out questionnaire:

A screenshot of a cell phone

Description automatically generated

1. Go to “Home” screen to see your daily tasks:

A screenshot of a phone

Description automatically generated

1. Add steps to input field and click the checkbox on any selected activities:

A screen shot of a phone

Description automatically generated

1. Go to the dashboard fragment and click on the current day to see completion progress:

A screenshot of a cell phone

Description automatically generated

1. Go to the profile fragment, and click on “Reset progress” and then confirm:

A screenshot of a phone

Description automatically generated

1. Click on “Edit user data” and set a different step count and save your change:

A screenshot of a phone

Description automatically generated

1. Using the toolbar at the top of the screen, go back to the home fragment to see updated and reset step counter:

A screenshot of a phone

Description automatically generated

## **List of functions**

1. Create a database to store user data.
2. Create a functional sing up system that checks if fields are properly filed.
3. Add a functioning login system that stores user session.
4. (Defense) Add a tracker for every time a user logs in inside the profile tab.
5. Store physical activity and plan information on app start.
6. Create a questionnaire that stores user preferences.
7. Display daily tasks based off selected plan.
8. Add functioning toolbar to activities.
9. (Defense) Let user mark completed activities.
10. Display a working calendar that lets users go back and forwards by months.
11. Add a daily step counter.
12. Add a circular progress bar to show step count progress.
13. Let the user see past complete and incomplete activities/steps by clicking on calendar cells.
14. Display weekly calorie loss statistics as a graph.
15. Let the user rest daily activity and step count progress.
16. Let the user change their user data (weight, plan, step count).
17. Let the user permanently delete their account.
18. Remove the need for users to sign in if they have not logged out from their previous session.
19. (Defense) Display upcoming activities in calendar view.

# **Solution**

## **Task #1. Create a database to store user data.**

The database was built using Room Persistence Library. Editing it was possible through the “App Inspection” tool. The User table is split into a User class, which stores the table’s model data, and a DAO class for storing related queries.

A screenshot of a computer program

Description automatically generated

Figure 1. Database layout

@Entity(indices = {@Index(value = {"email"}, unique = true)})  
public class User {  
 @PrimaryKey(autoGenerate = true)  
 private long id;  
  
 @NonNull  
 @ColumnInfo(name = "email")  
 private String email;  
  
 @NonNull  
 @ColumnInfo(name = "username")  
 private String username;  
  
 @NonNull  
 @ColumnInfo(name = "password")  
 private String password;

Figure 2 User class code

@Dao  
public interface UserDAO {  
 @Insert  
 void insert(User user);  
  
 @Query("DELETE FROM user")  
 void deleteAll();  
  
 @Query("SELECT \* FROM user ORDER BY username ASC")  
 List<User> getAllUsers();  
  
 @Query("SELECT COUNT(\*) FROM user WHERE email = :email\_string;")  
 boolean checkIfEmailAvailable(String email\_string);  
  
 @Query("SELECT id FROM user WHERE email = :email\_string AND password = :password\_string;")  
 long getIdByLogin(String email\_string, String password\_string);  
  
 @Query("SELECT username FROM user WHERE id = :id;")  
 String getUsernameById(Long id);  
}

Figure 3 UserDAO code

public class AppActivity extends Application {  
 private static AppDatabase *db*;  
  
 @Override  
 public void onCreate() {  
 super.onCreate();  
 *db* = Room.*databaseBuilder*(this, AppDatabase.class, "fitmax\_db")  
 .allowMainThreadQueries().build();  
 }  
  
 public static AppDatabase getDatabase() {  
 return *db*;  
 }  
}

Figure 4 AppActivity code

@Database(entities = {User.class, LoginDate.class}, version = 1)  
public abstract class AppDatabase extends RoomDatabase {  
 public abstract UserDAO userDAO();  
 public abstract LoginDateDAO loginDateDAO();  
}

Figure 5 AppDatabase code

## **Task #2. Create a functional sing up system that checks if fields are properly filed.**

The sign-up screen takes in 4 String values, if any of them are empty – an error is shown to the user. Before creating an account, a query is executed, checking whether the given email is taken or not. If it is, an appropriate error is shown. On successful creation, the user is sent back to the login screen.

A screen shot of a phone

Description automatically generated

Figure 6 Sign up page with invalid data

public class SignUp extends AppCompatActivity {  
 private AppDatabase db;  
 private Button signUpButton;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
  
 ActivitySignUpBinding binding;  
 super.onCreate(savedInstanceState);  
 binding = ActivitySignUpBinding.*inflate*(getLayoutInflater());  
 View view = binding.getRoot();  
 setContentView(view);  
  
 db = AppActivity.*getDatabase*();  
  
  
 // sign up function  
 binding.signUp.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
  
 boolean isValid = true;  
  
 // username ------------------------------------------------------------------------  
 String username\_string = binding.username.getText().toString().trim();  
 if (username\_string.isEmpty()) {  
 isValid = false;  
 binding.usernameContainer.setHelperText("Username invalid");  
 } else binding.usernameContainer.setHelperTextEnabled(false);  
  
 // email ---------------------------------------------------------------------------  
 String email\_string = binding.email.getText().toString().trim();  
 if (email\_string.isEmpty()){  
// || Patterns.EMAIL\_ADDRESS.matcher(email\_string).matches()) {  
 isValid = false;  
 binding.emailContainer.setHelperText("Email invalid");  
 } else if (db.userDAO().checkIfEmailAvailable(email\_string)) {  
 isValid = false;  
 binding.emailContainer.setHelperText("Email already in use");  
 } else binding.emailContainer.setHelperTextEnabled(false);  
  
 // password ------------------------------------------------------------------------  
 String password\_string = binding.password.getText().toString().trim();  
 if (password\_string.isEmpty()) {  
 isValid = false;  
 binding.passwordContainer.setHelperText("Password invalid");  
 } else binding.passwordContainer.setHelperTextEnabled(false);  
  
 // password confirm ----------------------------------------------------------------  
 String password\_confirm\_string = binding.passwordConfirm.getText().toString().trim();  
 if (password\_confirm\_string.isEmpty()) {  
 isValid = false;  
 binding.passwordConfirmContainer.setHelperText("Password invalid");  
 } else if (!password\_string.equals(password\_confirm\_string)) {  
 isValid = false;  
 binding.passwordConfirmContainer.setHelperText("Password does not match");  
 } else binding.passwordConfirmContainer.setHelperTextEnabled(false);  
  
 if (isValid){  
 User user = new User();  
 user.setUsername(username\_string);  
 user.setEmail(email\_string);  
 user.setPassword(password\_string);  
 db.userDAO().insert(user);  
  
 String message = "Account successfully created!";  
 Log.*v*("MMMM", message);  
 openLogin();  
 }  
 }  
  
 private boolean checkEmail(String email) {  
 return db.userDAO().checkIfEmailAvailable(email);  
 }  
 });  
 }  
  
 public void openLogin(){  
 Intent intent = new Intent(this, Login.class);  
 startActivity(intent);  
 }  
}

Figure 7 Sign up code

## **Task #3. Add a functioning login system that stores user session.**

Log in sessions were achieved by using SharedPreferences to store login information. First, the page checks if there is a matching password for the given email and password. If a match is found, an appropriate id is stored in SharedPreferences, and the user is sent to the main tab screen.

A screen shot of a phone

Description automatically generated

Figure 8 Login page

public class Login extends AppCompatActivity {  
 private AppDatabase db;  
 private Button loginButton;  
 private Button signupButton;  
 private TextView email\_field;  
 private TextView password\_field;  
 SharedPreferences sharedPrefs;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_login*);  
  
 db = AppActivity.*getDatabase*();  
 loginButton = findViewById(R.id.*log\_in*);  
 signupButton = findViewById(R.id.*sing\_up\_link*);  
 email\_field = findViewById(R.id.*email*);  
 password\_field = findViewById(R.id.*password*);  
  
  
 db.userDAO().getAllUsers();  
  
 loginButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 String email = email\_field.getText().toString().trim();  
 String password = password\_field.getText().toString().trim();  
 long id = db.userDAO().getIdByLogin(email, password);  
  
 // in case user doesn't exist  
 if (id <= 0)  
 return;  
  
 // store user id for session  
 sharedPrefs = getSharedPreferences("user", Context.*MODE\_PRIVATE*);  
 SharedPreferences.Editor editor = sharedPrefs.edit();  
 editor.putLong("user", id);  
 editor.apply();  
 openTabActivity();  
 }  
 });  
 signupButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 openSingUp();  
 }  
 });  
 }  
  
 public void openTabActivity() {  
 Intent intent = new Intent(this, TabScreen.class);  
 startActivity(intent);  
 }  
  
 public void openSingUp() {  
 Intent intent = new Intent(this, SignUp.class);  
 startActivity(intent);  
 }  
}

Figure 9 Login page code

## **Task #4. (Defense) Add a tracker for every time a user logs in inside the profile tab.**

For loogin information storage, a LoginDate table was created. Before the main tab activity switch happens in the login screen, the LoginDate table is updated. Using SharedPreferences, login dates are accessible in the profile screen through an SQL query.

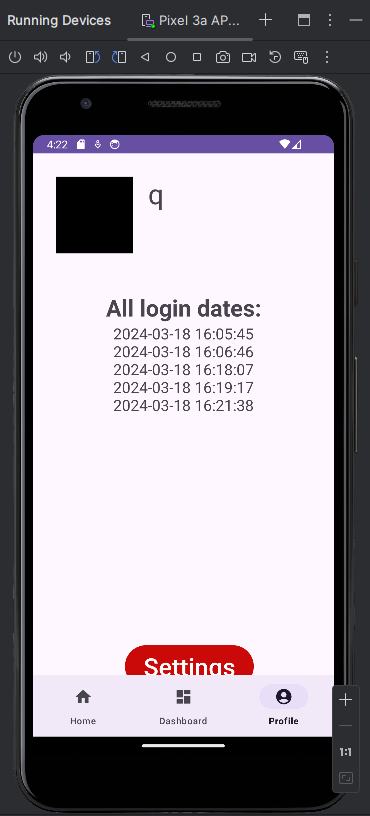


Figure 10 Login date tracker

// date setup  
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
String currentDateAndTime = sdf.format(new Date());  
LoginDate loginDate = new LoginDate();  
loginDate.setLogin\_date(String.*valueOf*(currentDateAndTime));  
loginDate.setId\_user(id);  
  
db.loginDateDAO().insert(loginDate);

Figure 11 Added date insert to login page

@Entity(foreignKeys = @ForeignKey(entity = User.class,  
 parentColumns = "id",  
 childColumns = "id\_user",  
 onDelete = ForeignKey.*CASCADE*))  
  
public class LoginDate {  
 @PrimaryKey(autoGenerate = true)  
 private long id\_login;  
  
 @NonNull  
 private long id\_user;  
  
 @NonNull  
 @ColumnInfo(name = "login\_date")  
 private String login\_date;  
  
 public long getId\_login() {  
 return id\_login;  
 }  
  
 public long getId\_user() {  
 return id\_user;  
 }  
  
 @NonNull  
 public String getLogin\_date() {  
 return login\_date;  
 }  
  
 public void setId\_login(long id\_login) {  
 this.id\_login = id\_login;  
 }  
  
 public void setId\_user(long id\_user) {  
 this.id\_user = id\_user;  
 }  
  
 public void setLogin\_date(@NonNull String login\_date) {  
 this.login\_date = login\_date;  
 }  
}

Figure 12 LoginDate model code

## **Task #5.** **Store physical activity and plan information.**

For this task 3 more database tables were added: PhysicalActivity, Plan and PlansFromActivities. The table “PlansFromActivities” simulates a many to many relationship between PhysicalActivity and Plan. These tables are later filled at program start by using a separate class “BaseData” to store data [1, 2, 3] query in string.

enum ActivityType {  
 *Core*,  
 *Holistic*,  
 *Lower*,  
 *Upper*}  
  
@Entity  
public class PhysicalActivity {  
  
 @PrimaryKey(autoGenerate = true)  
 private long id\_activity;  
  
 @NonNull  
 @ColumnInfo(name = "activity\_name")  
 private String activity\_name;  
  
 @NonNull  
 @ColumnInfo(name = "duration")  
 private String duration;  
  
 @NonNull  
 @ColumnInfo(name = "activity\_type")  
 private ActivityType type;  
  
 @NonNull  
 @ColumnInfo(name = "met")  
 private Float met;  
  
 // methods -------------------------------------------------------------------------------------  
 public long getId\_activity() {  
 return id\_activity;  
 }  
  
 @NonNull  
 public String getActivity\_name() {  
 return activity\_name;  
 }  
  
 @NonNull  
 public String getDuration() {  
 return duration;  
 }  
  
 @NonNull  
 public ActivityType getType() {  
 return type;  
 }  
  
 @NonNull  
 public Float getMet() {  
 return met;  
 }  
  
 public void setId\_activity(long id\_activity) {  
 this.id\_activity = id\_activity;  
 }  
  
 public void setActivity\_name(@NonNull String activity\_name) {  
 this.activity\_name = activity\_name;  
 }  
  
 public void setDuration(@NonNull String duration) {  
 this.duration = duration;  
 }  
  
 public void setType(@NonNull ActivityType type) {  
 this.type = type;  
 }  
  
 public void setMet(@NonNull Float met) {  
 this.met = met;  
 }  
}

Figure 13 PhysicalActivity table code fragment

package com.example.fitmax.Database;  
  
import androidx.annotation.NonNull;  
import androidx.room.ColumnInfo;  
import androidx.room.Entity;  
import androidx.room.PrimaryKey;  
  
@Entity  
public class Plan {  
  
 @PrimaryKey(autoGenerate = true)  
 private long id\_plan;  
  
 @NonNull  
 @ColumnInfo(name = "plan\_name")  
 private String plan\_name;  
  
 public long getId\_plan() {  
 return id\_plan;  
 }  
  
 @NonNull  
 public String getPlan\_name() {  
 return plan\_name;  
 }  
  
 public void setId\_plan(long id\_plan) {  
 this.id\_plan = id\_plan;  
 }  
  
 public void setPlan\_name(@NonNull String plan\_name) {  
 this.plan\_name = plan\_name;  
 }  
}

Figure 14 Plan table code fragment

package com.example.fitmax.Database;  
  
import androidx.room.ColumnInfo;  
import androidx.room.Entity;  
import androidx.room.ForeignKey;  
import androidx.room.Index;  
import androidx.room.PrimaryKey;  
  
enum Weekday {  
 *Monday*,  
 *Tuesday*,  
 *Wednesday*,  
 *Thursday*,  
 *Friday*,  
 *Saturday*,  
 *Sunday*}  
  
@Entity(foreignKeys = {  
 @ForeignKey(entity = PhysicalActivity.class,  
 parentColumns = "id\_activity",  
 childColumns = "id\_activity",  
 onDelete = ForeignKey.*CASCADE*),  
 @ForeignKey(entity = Plan.class,  
 parentColumns = "id\_plan",  
 childColumns = "id\_plan",  
 onDelete = ForeignKey.*CASCADE*)},  
  
 indices = {  
 @Index(value = {"id\_activity"}),  
 @Index(value = {"id\_plan"})  
 }  
)  
  
public class PlansFromActivities {  
  
 @PrimaryKey(autoGenerate = true)  
 private long id\_pfa;  
  
 private long id\_plan;  
  
 private long id\_activity;  
  
 @ColumnInfo(name = "weekday")  
 private Weekday weekday;  
  
 // methods -------------------------------------------------------------------------------------  
  
  
 public void setId\_pfa(long id\_pfa) {  
 this.id\_pfa = id\_pfa;  
 }  
  
 public void setId\_plan(long id\_plan) {  
 this.id\_plan = id\_plan;  
 }  
  
 public void setId\_activity(long id\_activity) {  
 this.id\_activity = id\_activity;  
 }  
  
 public void setWeekday(Weekday weekday) {  
 this.weekday = weekday;  
 }  
  
 public long getId\_pfa() {  
 return id\_pfa;  
 }  
  
 public long getId\_plan() {  
 return id\_plan;  
 }  
  
 public long getId\_activity() {  
 return id\_activity;  
 }  
  
 public Weekday getWeekday() {  
 return weekday;  
 }  
}

Figure 15 PlansFromActivities table code fragment

try {  
 SupportSQLiteDatabase sqLiteDatabase = *db*.getOpenHelper().getWritableDatabase();  
 String[] queries = BaseData.*base\_data*.split(";");  
 for (String query : queries) {  
 if (!query.trim().isEmpty()) {  
 sqLiteDatabase.execSQL(query);  
 }  
 }  
} catch (SQLException e) {  
 System.*out*.println("Error executing query: " + e.getMessage());  
 e.printStackTrace();  
}

Figure 16 Added code to store initial database data in „AppActivity“ class

## **Task #6.** **Create a questionnaire that stores user preferences.**

Upon finishing account creation, you are sent to a questionnaire that adds additional info about the user. This data is stored in the “User” table, which has been extended. From there you can input your weight and choose a desirable activity plan from a spinner.

A screenshot of a phone

Description automatically generated

Figure 17 Questionnaire activity

// date setup  
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
String currentDateAndTime = sdf.format(new Date());  
LoginDate loginDate = new LoginDate();  
loginDate.setLogin\_date(String.*valueOf*(currentDateAndTime));  
loginDate.setId\_user(id);  
  
db.loginDateDAO().insert(loginDate);

Figure 18 Added date insert to login page

@Entity(foreignKeys = @ForeignKey(entity = User.class,  
 parentColumns = "id",  
 childColumns = "id\_user",  
 onDelete = ForeignKey.*CASCADE*))  
  
public class LoginDate {  
 @PrimaryKey(autoGenerate = true)  
 private long id\_login;  
  
 @NonNull  
 private long id\_user;  
  
 @NonNull  
 @ColumnInfo(name = "login\_date")  
 private String login\_date;  
  
 public long getId\_login() {  
 return id\_login;  
 }  
  
 public long getId\_user() {  
 return id\_user;  
 }  
  
 @NonNull  
 public String getLogin\_date() {  
 return login\_date;  
 }  
  
 public void setId\_login(long id\_login) {  
 this.id\_login = id\_login;  
 }  
  
 public void setId\_user(long id\_user) {  
 this.id\_user = id\_user;  
 }  
  
 public void setLogin\_date(@NonNull String login\_date) {  
 this.login\_date = login\_date;  
 }  
}

Figure 19 LoginDate model code

public class Questionnaire extends AppCompatActivity {  
 private AppDatabase db;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 // binding ---------------------------------------------------------------------------------  
 QuestionnaireBinding binding;  
 binding = QuestionnaireBinding.*inflate*(getLayoutInflater());  
 View view = binding.getRoot();  
 setContentView(view);  
  
 db = AppActivity.*getDatabase*();  
 // tool bar --------------------------------------------------------------------------------  
 setSupportActionBar(binding.toolbar.toolbar);  
 Objects.*requireNonNull*(getSupportActionBar()).setDisplayHomeAsUpEnabled(true);  
 setTitle(getTitle());  
 // -----------------------------------------------------------------------------------------  
  
 List<Plan> plans = db.planDAO().getAll();  
 populateSpinner(binding.planSpinner, plans);  
  
  
 // sign up function  
 binding.confirmButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
  
 Float weight = Float.*parseFloat*(binding.weight.getText().toString().trim());  
 if (weight.isNaN()) {  
 binding.weightContainer.setHelperText("Weight required");  
 return;  
 } else if (weight <= 0) {  
 binding.weightContainer.setHelperText("Weight must be positive number");  
 return;  
 } else binding.weightContainer.setHelperTextEnabled(false);  
  
  
 long id\_user = getIntent().getLongExtra("id\_user", -1);  
 if (id\_user < 0) {  
 Toast.*makeText*(getApplicationContext(),  
 "Error, id: " + id\_user + " is incorrect",  
 Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
  
 String plan\_string = binding.planSpinner.getSelectedItem().toString();  
 Plan selectedPlan = GetPlanFromName(plan\_string, plans);  
 db.userDAO().FinishQuestionnaire(id\_user, weight, selectedPlan.getId\_plan());  
  
 Toast.*makeText*(getApplicationContext(), "Account updated successfully",  
 Toast.*LENGTH\_SHORT*).show();  
  
 SessionManager.*storeLoginSession*(getApplicationContext(), id\_user);  
 SessionManager.*GoToMain*(view.getContext());  
 }  
 });  
 }  
  
 private Plan GetPlanFromName(String name, List<Plan> list) {  
 Plan foundPlan = null;  
 for (Plan plan : list) {  
 if (plan.getPlan\_name() == name)  
 foundPlan = plan;  
 }  
 return foundPlan;  
 }  
  
 private void populateSpinner(Spinner spinner, List<Plan> list) {  
  
 List<String> spinner\_list = new ArrayList<>();  
 for (Plan plan : list) {  
 spinner\_list.add(plan.getPlan\_name());  
 }  
 ArrayAdapter<String> adapter = new ArrayAdapter<>(this,  
 R.layout.*s\_plan\_layout*, spinner\_list);  
  
 adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);  
 spinner.setAdapter(adapter);  
 }  
}

Figure 20 Questionnaire activity code

## **Task #7. Display daily tasks based off selected plan.**

To display user activities based on the current weekday, a custom query was created in “User” table. Query results were displayed on the “Home fragment”.

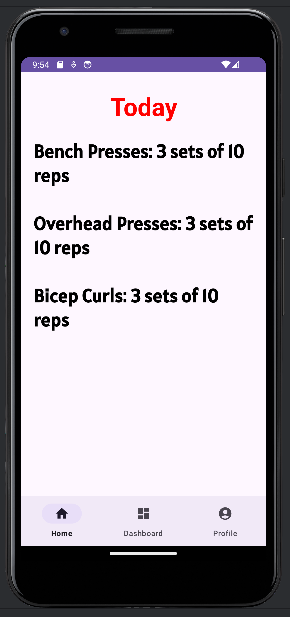


Figure 21 Displayed tasks in home fragment

@Query("SELECT PhysicalActivity.id\_activity, " +  
 "PhysicalActivity.activity\_name, " +  
 "PhysicalActivity.duration, " +  
 "PhysicalActivity.activity\_type, " +  
 "PhysicalActivity.met " +  
 "FROM Plan " +  
 "JOIN PlansFromActivities on plan.id\_plan = PlansFromActivities.id\_plan " +  
 "JOIN PhysicalActivity on PhysicalActivity.id\_activity = PlansFromActivities.id\_activity " +  
 "JOIN User on user.id\_plan = PlansFromActivities.id\_plan " +  
 "WHERE User.id\_user = :id\_user " +  
 "AND PlansFromActivities.weekday = CASE strftime('%w', 'now')\n" +  
 " WHEN '0' THEN 'Sunday'\n" +  
 " WHEN '1' THEN 'Monday'\n" +  
 " WHEN '2' THEN 'Tuesday'\n" +  
 " WHEN '3' THEN 'Wednesday'\n" +  
 " WHEN '4' THEN 'Thursday'\n" +  
 " WHEN '5' THEN 'Friday'\n" +  
 " ELSE 'Saturday'\n" +  
 " END;\n")  
List<PhysicalActivity> GetTodaysActivities(long id\_user);

Figure 22 Weekday activity detection query

public class HomeFragment extends Fragment {  
  
 private AppDatabase db;  
 private FHomeBinding binding;  
  
 public View onCreateView(@NonNull LayoutInflater inflater,  
 ViewGroup container, Bundle savedInstanceState) {  
 HomeViewModel homeViewModel =  
 new ViewModelProvider(this).get(HomeViewModel.class);  
  
 binding = FHomeBinding.*inflate*(inflater, container, false);  
 View root = binding.getRoot();  
  
// final TextView textView = binding.planDetails;  
  
  
 db = AppActivity.*getDatabase*();  
 binding.planDetails.setText("");  
  
 long id\_user = SessionManager.*getLoginSession*(getContext());  
 List<PhysicalActivity> list = db.userDAO().GetTodaysActivities(id\_user);  
 String lines = "";  
 for (PhysicalActivity activity : list) {  
 lines += activity.getActivity\_name() + ": " + activity.getDuration() + "\n\n";  
 }  
 binding.planDetails.setText(lines);  
  
  
// homeViewModel.getText().observe(getViewLifecycleOwner(), textView::setText);  
 return root;  
 }  
  
 @Override  
 public void onDestroyView() {  
 super.onDestroyView();  
 binding = null;  
 }  
}

Figure 23 Home fragment code

## **Task #8. Add functioning toolbar to activities.**

Toolbar functionality was achieved by creating a custom toolbar layout and inserting it in the desired activity. For the toolbar back button and current text display to work properly, appropriate activity labels and parent activities must be set in AndroidManifest.xml.

A screenshot of a computer

Description automatically generated

Figure 24 Appbar in login screen

<?xml version="1.0" encoding="utf-8"?>  
<androidx.appcompat.widget.Toolbar xmlns:android="http://schemas.android.com/apk/res/android"  
 android:id="@+id/toolbar"  
 android:layout\_width="match\_parent"  
 android:layout\_height="?attr/actionBarSize"  
 android:background="@color/toolbar"  
 android:elevation="4dp"  
 android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">  
  
 <TextView  
 android:id="@+id/toolbar\_title"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"  
 android:textColor="@android:color/white"  
 android:textSize="20sp"  
 android:textStyle="bold" />  
  
</androidx.appcompat.widget.Toolbar>

Figure 25 Toolbar xml code

// tool bar --------------------------------------------------------------------------------  
setSupportActionBar(binding.toolbar.toolbar);  
Objects.*requireNonNull*(getSupportActionBar()).setDisplayHomeAsUpEnabled(true);  
setTitle(getTitle());  
// -----------------------------------------------------------------------------------------

Figure 26 Toolbar code fragment

## **Task #9. (Defense) Let user mark completed activities.**

A checklist was implemented with the use of CompletedActivities table. In home fragment, linear layout, checkbox items were added for each activity along with an appropriate OnClickListener. Upon clicking, the selected activity is marked as completed.

A cell phone with a white screen

Description automatically generated

Figure 27 Activity checklist

@Entity(foreignKeys = {  
 @ForeignKey(entity = PhysicalActivity.class,  
 parentColumns = "id\_activity",  
 childColumns = "id\_activity",  
 onDelete = ForeignKey.*CASCADE*),  
 @ForeignKey(entity = User.class,  
 parentColumns = "id\_user",  
 childColumns = "id\_user",  
 onDelete = ForeignKey.*CASCADE*)},  
  
 indices = {  
 @Index(value = {"id\_activity"}),  
 @Index(value = {"id\_user"})  
 }  
)  
public class CompletedActivities {  
  
 @PrimaryKey(autoGenerate = true)  
 private long id\_completed\_activity;  
  
 private long id\_activity;  
  
 private long id\_user;  
  
 @NonNull  
 @ColumnInfo(name = "completion\_date")  
 private String completion\_date;  
  
 @NonNull  
 @ColumnInfo(name = "completed")  
 private boolean completed;  
  
  
 public long getId\_completed\_activity() {  
 return id\_completed\_activity;  
 }  
  
 public void setId\_completed\_activity(long id\_completed\_activity) {  
 this.id\_completed\_activity = id\_completed\_activity;  
 }  
  
 public long getId\_activity() {  
 return id\_activity;  
 }  
  
 public void setId\_activity(long id\_activity) {  
 this.id\_activity = id\_activity;  
 }  
  
 public long getId\_user() {  
 return id\_user;  
 }  
  
 public void setId\_user(long id\_user) {  
 this.id\_user = id\_user;  
 }  
  
 @NonNull  
 public String getCompletion\_date() {  
 return completion\_date;  
 }  
  
 public void setCompletion\_date(@NonNull String completion\_date) {  
 this.completion\_date = completion\_date;  
 }  
  
 public boolean isCompleted() {  
 return completed;  
 }  
  
 public void setCompleted(boolean completed) {  
 this.completed = completed;  
 }  
}

Figure 28 CompletedAtivities table class

public class HomeFragment extends Fragment {  
  
 private AppDatabase db;  
 private FHomeBinding binding;  
  
 public View onCreateView(@NonNull LayoutInflater inflater,  
 ViewGroup container, Bundle savedInstanceState) {  
 HomeViewModel homeViewModel =  
 new ViewModelProvider(this).get(HomeViewModel.class);  
  
 binding = FHomeBinding.*inflate*(inflater, container, false);  
 View root = binding.getRoot();  
  
 db = AppActivity.*getDatabase*();  
  
 long id\_user = SessionManager.*getLoginSession*(getContext());  
 List<PhysicalActivity> list = db.userDAO().GetTodaysActivities(id\_user);  
 for (PhysicalActivity activity : list) {  
  
 CheckBox checkBox = new CheckBox(getContext());  
 checkBox.setTextSize(20);  
 checkBox.setText(activity.getActivity\_name() + ": " + activity.getDuration());  
 binding.planContainer.addView(checkBox);  
 checkBox.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
  
 if (checkBox.isChecked()) {  
 CompletedActivities ca = new CompletedActivities();  
 ca.setCompleted(true);  
 ca.setId\_activity(activity.getId\_activity());  
 ca.setId\_user(id\_user);  
  
 SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");  
 String currentDate = sdf.format(new Date());  
  
 ca.setCompletion\_date(currentDate);  
  
 db.completedActivitiesDAO().insert(ca);  
 checkBox.setEnabled(false);  
 }  
 }  
 });  
 }  
  
 return root;  
 }  
  
 @Override  
 public void onDestroyView() {  
 super.onDestroyView();  
 binding = null;  
 }  
}

Figure 29 Home fragment checklist implementation

## **Task #10. Display a working calendar that lets users go back and forwards by months.**

A working calendar implementation was achieved with the use of an Adapter and a ViewHolder classes. The view holder describes the individual cell logic, while the adapter creates sets the data to a RecycleView.

A screenshot of a cell phone

Description automatically generated

Figure 30 Calendar display

public class CalendarAdapter extends RecyclerView.Adapter<CalendarViewHolder> {  
  
 private final ArrayList<String> daysOfMonth;  
 private final OnItemListener onItemListener;  
 private final LocalDate selectedDate;  
  
 public CalendarAdapter(ArrayList<String> daysOfMonth, OnItemListener onItemListener, LocalDate selectedDate) {  
 this.daysOfMonth = daysOfMonth;  
 this.onItemListener = onItemListener;  
 this.selectedDate = selectedDate;  
 }  
  
 @NonNull  
 @Override  
 public CalendarViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  
  
 // instantiates day cell  
 LayoutInflater inflater = LayoutInflater.*from*(parent.getContext());  
 View view = inflater.inflate(R.layout.*calendar\_cell*, parent, false);  
 ViewGroup.LayoutParams layoutParams = view.getLayoutParams();  
 layoutParams.height = (int) (parent.getHeight() / 6);  
 return new CalendarViewHolder(view, onItemListener);  
 }  
  
 @Override  
 public void onBindViewHolder(@NonNull CalendarViewHolder holder, int position) {  
  
 holder.dayOfMonth.setText(daysOfMonth.get(position));  
  
 if (daysOfMonth.get(position).isEmpty()) {  
 return;  
 }  
  
 int day = Integer.*parseInt*(daysOfMonth.get(position));  
 if (selectedDate.withDayOfMonth(day).isEqual(LocalDate.*now*()))  
 holder.dayOfMonth.setBackgroundResource(R.drawable.*calendar\_highlight*);  
 }  
  
 @Override  
 public int getItemCount() {  
 return daysOfMonth.size();  
 }  
  
 public interface OnItemListener {  
 void onItemClick(int position, String dayText);  
 }  
}

Figure 31 Calendar adapter code fragment

public class CalendarViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {  
  
 public final TextView dayOfMonth;  
 private final CalendarAdapter.OnItemListener onItemListener;  
  
 public CalendarViewHolder(@NonNull View itemView, CalendarAdapter.OnItemListener onItemListener) {  
 super(itemView);  
 dayOfMonth = itemView.findViewById(R.id.*day\_text*);  
 this.onItemListener = onItemListener;  
 itemView.setOnClickListener(this);  
 }  
  
 @Override  
 public void onClick(View v) {  
 onItemListener.onItemClick(getAdapterPosition(), (String) dayOfMonth.getText());  
 }  
}

Figure 32 Calendar view holder fragment

private void setMonthView(LocalDate date) {  
 // moth - year display  
 binding.monthYearText.setText(getMonthYearString(date));  
 // day container  
 ArrayList<String> days = getDayList(date);  
  
 // instantiation  
 CalendarAdapter adapter = new CalendarAdapter(days, this, selectedDate);  
 RecyclerView.LayoutManager layoutManager = new GridLayoutManager(getContext(), 7);  
 binding.calendarContainer.setLayoutManager(layoutManager);  
 binding.calendarContainer.setAdapter(adapter);  
}  
  
private ArrayList<String> getDayList(LocalDate date) {  
  
 ArrayList<String> days = new ArrayList<String>();  
 YearMonth month = YearMonth.*from*(date);  
 int daysInMonth = month.lengthOfMonth();  
 LocalDate firstOfMonth = date.withDayOfMonth(1);  
 int firstDayOfWeek = firstOfMonth.getDayOfWeek().getValue();  
  
 for (int i = 1; i <= 42; i++)  
 if (i < firstDayOfWeek || i > daysInMonth + firstDayOfWeek - 1)  
 days.add("");  
 else  
 days.add(String.*valueOf*(i - firstDayOfWeek + 1));  
 return days;  
}

Figure 33 Calendar creation logic

## **Task #11. Add a daily step counter.**

A daily step counter was added using CompletedSteps and StepHistory tables. Completed steps of days are stored as entries in CompletedSteps and the StepHistory table is used to keep track of changes to user preferences. A similar layout was also applied to plans for future use.

A screenshot of a computer

Description automatically generated

Figure 34 Daily step counter

@Entity(foreignKeys = @ForeignKey(entity = User.class,  
 parentColumns = "id\_user",  
 childColumns = "id\_user",  
 onDelete = ForeignKey.*CASCADE*),  
 primaryKeys = {"id\_user", "completion\_date"})  
public class CompletedSteps {  
  
 private long id\_completed\_steps;  
  
 private int step\_count;  
  
 private long id\_user;  
  
 @NonNull  
 @ColumnInfo(name = "completion\_date")  
 private String completion\_date;  
  
 // methods -------------------------------------------------------------------------------------  
  
 public long getId\_completed\_steps() {  
 return id\_completed\_steps;  
 }  
  
 public void setId\_completed\_steps(long id\_completed\_steps) {  
 this.id\_completed\_steps = id\_completed\_steps;  
 }  
  
 public int getStep\_count() {  
 return step\_count;  
 }  
  
 public void setStep\_count(int step\_count) {  
 this.step\_count = step\_count;  
 }  
  
 public long getId\_user() {  
 return id\_user;  
 }  
  
 public void setId\_user(long id\_user) {  
 this.id\_user = id\_user;  
 }  
  
 @NonNull  
 public String getCompletion\_date() {  
 return completion\_date;  
 }  
  
 public void setCompletion\_date(@NonNull String completion\_date) {  
 this.completion\_date = completion\_date;  
 }  
}

Figure 35 Completed steps database table

private boolean addSteps() {  
 if (binding.stepInput.getText().toString().isEmpty())  
 return false;  
  
 long id\_user = SessionManager.*getLoginSession*(getContext());  
 int steps = Integer.*parseInt*(binding.stepInput.getText().toString());  
 String today = CommonMethods.*getToday*();  
  
 if (db.completedStepsDAO().userEntriesAdded(id\_user, today) > 0)  
 db.completedStepsDAO().addSteps(id\_user, today, steps);  
 else {  
 CompletedSteps completedSteps = new CompletedSteps();  
 completedSteps.setId\_user(id\_user);  
 completedSteps.setCompletion\_date(today);  
 completedSteps.setStep\_count(steps);  
 db.completedStepsDAO().insert(completedSteps);  
 }  
 return true;

Figure 36 Step addition code

## **Task #12. Add a circular progress bar to show step count progress.**

A circular progress bar base class was sourced from a GitHub repository [4]. It displays the percentage completion rate of the user’s daily step count.

A screen shot of a phone

Description automatically generated

Figure 37 Step count progress bar

<RelativeLayout  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content">  
  
 <com.mikhaellopez.circularprogressbar.CircularProgressBar  
 android:id="@+id/step\_progress"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="200dp"  
 app:cpb\_background\_progressbar\_color="@color/red"  
 app:cpb\_background\_progressbar\_width="25dp"  
 app:cpb\_progressbar\_color="#4CAF50"  
 app:cpb\_progressbar\_width="25dp" />  
  
 <TextView  
 android:id="@+id/step\_percentage"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_centerInParent="true"  
 android:fontFamily="sans-serif"  
 android:text="125%"  
 android:textSize="50sp"  
 android:textStyle="bold" />  
</RelativeLayout>

Figure 38 Progress bar xml code

private void setStepCountDisplay() {  
  
 db = AppActivity.*getDatabase*();  
 long id\_user = SessionManager.*getLoginSession*(getContext());  
  
 // progress bar  
 String today = CommonMethods.*getToday*();  
 int maxSteps = db.stepHistoryDAO().getUserSteps(id\_user, today);  
 int currentSteps = db.completedStepsDAO().getUserSteps(id\_user, today);  
 binding.stepProgress.setProgressMax(maxSteps);  
 binding.stepProgress.setProgressWithAnimation(currentSteps, 500L);  
  
 binding.stepsTotal.setText(currentSteps + "/" + maxSteps);  
 int percentage = (int) ((float) currentSteps / (float) maxSteps \* 100);  
 Log.*d*("EEEEEEE", "Percentage is: " + percentage);  
 binding.stepPercentage.setText(percentage + "%");  
}

Figure 39 Progress bar code fragment

## **Task #13. Let the user see past complete and incomplete activities/steps by clicking on calendar cells.**

To achieve a popup window functionality, a PopupWindow class was used. It was instantiated by implementing an onClickListener to the calendar adapter. On click the item displays that specific day’s completed activity and step counts.

A screenshot of a cell phone

Description automatically generated

Figure 40 Calendar popup

@Override  
public void onItemClick(int position, String dayText) {  
  
 if (dayText.isEmpty())  
 return;  
  
 long id\_user = SessionManager.*getLoginSession*(getContext());  
 User user = db.userDAO().getUser(id\_user);  
  
 LocalDate fullDate = selectedDate.withDayOfMonth(Integer.*parseInt*(dayText));  
 LocalDate creationDate = LocalDate.*parse*(user.getCreation\_date());  
  
 // return if selected date is before creation date  
 // or if it's after current day  
 if (fullDate.isBefore(creationDate) || fullDate.isAfter(LocalDate.*now*()))  
 return;  
  
 // get activity info -----------------------------------------------------------------------

// other logic ...

// ...

// ...

// ..,  
 // create popup ---------------------------------------------------------------------------  
 final PopupWindow popupWindow = new PopupWindow(  
 view,  
 ViewGroup.LayoutParams.*WRAP\_CONTENT*,  
 ViewGroup.LayoutParams.*WRAP\_CONTENT*,  
 true // Focusable  
 );  
  
 View clickedView = binding.calendarContainer.getLayoutManager().findViewByPosition(position);  
 popupWindow.showAsDropDown(clickedView);  
  
}

Figure 41 Popup code fragment

## **Task #14. Display weekly calorie loss statistics as a graph.**

The base chart was sourced from a GitHub repository [5]. Visual display was created by using BarChart, BarDataSet and BarData classes. Calories are calculated [1] in a separate class.

**A cell phone with a calories burned chart

Description automatically generated**

Figure 42 Weekly calorie loss chart

private void createChart() {  
 long id\_user = SessionManager.*getLoginSession*(getContext());  
  
 LocalDate startOfWeek = LocalDate.*now*().with(TemporalAdjusters.*previousOrSame*(DayOfWeek.*MONDAY*));  
 float totalCalories = 0;  
  
 // get data  
 ArrayList<BarEntry> list = new ArrayList<>();  
 float weight = db.userDAO().getUser(id\_user).getWeight();  
 for (int i = 0; i < 7; i++) {  
  
 int steps = db.completedStepsDAO().getUserSteps(id\_user, startOfWeek.toString());  
 List<PhysicalActivity> activities = db.completedActivitiesDAO().  
 getCompletedInDay(id\_user, startOfWeek.toString());  
 startOfWeek = startOfWeek.plusDays(1);  
  
 float calories = 0;  
 for (PhysicalActivity act : activities) {  
 calories += CommonMethods.*GetActivityCalories*(act, weight);  
 }  
 calories += CommonMethods.*GetStepCalories*(steps, weight);  
 list.add(new BarEntry(i, calories));  
 totalCalories += calories;  
 }

Figure 43 Bar chart code fragment

// chart customization ---------------------------------------------------------------------  
BarChart barChart = binding.calorieChart;  
BarDataSet dataSet = new BarDataSet(list, "Calories burned this week");  
BarData data = new BarData(dataSet);  
barChart.setData(data);  
  
barChart.setDrawGridBackground(false);  
barChart.getDescription().setEnabled(false);  
barChart.setDrawBorders(false);  
barChart.setScaleEnabled(false);  
barChart.getLegend().setEnabled(false);  
  
// data format  
Typeface typeface = Typeface.*create*(Typeface.*DEFAULT\_BOLD*, Typeface.*BOLD*);  
dataSet.setColors(ColorTemplate.*COLORFUL\_COLORS*);  
dataSet.setValueTextSize(16);  
dataSet.setValueTypeface(typeface);  
  
// x axis format  
XAxis xAxis = barChart.getXAxis();  
xAxis.setPosition(XAxis.XAxisPosition.*BOTTOM\_INSIDE*);  
xAxis.setDrawGridLines(false);  
xAxis.setTypeface(typeface);  
xAxis.setTextSize(12);  
  
String[] xLabels = {"MON", "TUE", "WED", "THU", "FRI", "SAT", "SUN"};  
xAxis.setValueFormatter(new IndexAxisValueFormatter(xLabels));  
  
barChart.getAxisLeft().setTextSize(12);  
barChart.getAxisRight().setTextSize(12);

Figure 44 BarChart display

public static float GetActivityCalories(PhysicalActivity activity, float weight) {  
 float kcal = activity.getMet() \* 0.0175f \* weight;  
  
 // rounds to 2 decimals  
 return Math.*round*(kcal \* 100f) / 100.0f;  
}  
public static float GetStepCalories(int steps, float weight) {  
 // 1 step = 0.05 METS  
 float kcal = steps \* 0.0083f \* 0.05f \* weight;  
  
 // rounds to 2 decimals  
 return Math.*round*(kcal \* 100f) / 100.0f;  
}

Figure 45 Calorie caluclations

## **Task #15. Let the user rest daily activity and step count progress.**

Before resetting progress, an AlertDialogue check is presented to the user for confirmation. For daily progress to be reset, both completed activity and completed step tables’ current day entries need to be deleted.

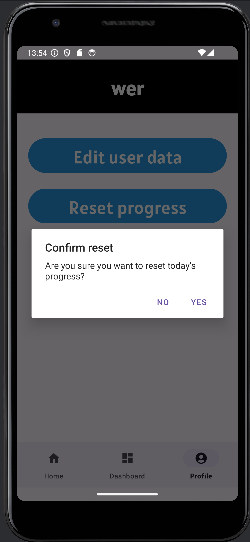


Figure 46 Daily progress reset function

private void resetDailyProgress() {  
 AlertDialog.Builder builder = new AlertDialog.Builder(getContext());  
 builder.setTitle("Confirm reset");  
 builder.setMessage("Are you sure you want to reset today's progress?");  
  
 builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which) {  
 long id\_user = SessionManager.*getLoginSession*(getContext());  
 db.completedStepsDAO().deleteSteps(id\_user, CommonMethods.*getToday*());  
 db.completedActivitiesDAO().deleteActivities(id\_user, CommonMethods.*getToday*());  
 }  
 });  
  
 builder.setNegativeButton("No", new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which) {  
 dialog.dismiss();  
 }  
 });  
  
 AlertDialog dialog = builder.create();  
 dialog.show();  
}

Figure 47 Progress reset code fragment

## **Task #16. Let the user change their user data (weight, plan, step count).**

Changing user data like step count and daily plan requires them to be overwritten in their current day history table only, to not overwrite previous activity/step history. Similar data checks were used in the questionnaire.

A screenshot of a phone

Description automatically generated

Figure 48 User data change screen

private void setPlan() {  
  
 long id\_user = SessionManager.*getLoginSession*(getBaseContext());  
 long id\_plan = binding.planSpinner.getSelectedItemId() + 1;  
  
 if (id\_plan == db.planHistoryDAO().getUserPlan(id\_user, CommonMethods.*getToday*()))  
 return;  
  
 // valid  
 PlanHistory planHistory = new PlanHistory();  
 planHistory.setId\_user(SessionManager.*getLoginSession*(getBaseContext()));  
 planHistory.setId\_plan(id\_plan);  
 planHistory.setPlan\_date(CommonMethods.*getToday*());  
  
 db.completedActivitiesDAO().deleteActivities(id\_user, CommonMethods.*getToday*());  
 db.planHistoryDAO().insert(planHistory);  
 displayPlan();  
}

Figure 49 Plan change code fragment

<LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:layout\_margin="20dp"  
 android:orientation="vertical">  
  
 <TextView  
 android:id="@+id/plan\_display"  
 style="@style/settingText"  
 android:text="Plan: Weekly" />  
  
 <Spinner  
 android:id="@+id/plan\_spinner"  
 style="@style/mySpinnerStyle"  
 android:background="@drawable/background\_rounded"  
 android:paddingLeft="6dp"  
 android:layout\_marginVertical="10dp"  
 android:textColor="@color/white" />  
  
 <Button  
 android:id="@+id/save\_plan"  
 android:layout\_width="match\_parent"  
 android:backgroundTint="@color/black"  
 android:fontFamily="@font/alatsi"  
 android:layout\_height="wrap\_content"  
 android:text="Save" />  
</LinearLayout>  
<View style="@style/divider" />

Figure 50 Plan change xml code fragment

## **Task #17.** **Let the user permanently delete their account.**

Before deletion an AlertDialogue window get presented for user confirmation. On account deletion, multiple tables need to be cleared, such as: user, completed activities, completed steps, plan history, step history. After deletion, the user gets logged out and sent to the starting screen.

A screenshot of a phone

Description automatically generated

Figure 51 Account deletion option

private void deleteUser() {  
  
 AlertDialog.Builder builder = new AlertDialog.Builder(this);  
 builder.setTitle("Confirm deletion");  
 builder.setMessage("Are you sure you want to delete your account?");  
  
 builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which) {  
  
 long id\_user = SessionManager.*getLoginSession*(getBaseContext());  
  
 db.completedActivitiesDAO().deleteAll(id\_user);  
 db.completedStepsDAO().deleteAll(id\_user);  
 db.planHistoryDAO().deleteAll(id\_user);  
 db.stepHistoryDAO().deleteAll(id\_user);  
 db.userDAO().deleteAll(id\_user);  
  
 SessionManager.*endSession*(getBaseContext());  
 SessionManager.*GoToStartingActivity*(Settings.this);  
 }  
 });  
  
 builder.setNegativeButton("No", new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which) {  
 dialog.dismiss();  
 }  
 });  
  
 AlertDialog dialog = builder.create();  
 dialog.show();  
}

Figure 52 Account deletion code fragment

## **Task #18. Remove the need for users to sign in if they have not logged out from their previous session.**

Session tracking was implemented through SharedPreferences. On starting application, if a user key exists from a previous session, send user to main screen immediately.

long id = SessionManager.*getLoginSession*(this);  
 AppDatabase db = AppActivity.*getDatabase*();  
  
 if (db.userDAO().checkIfQuestionnaireComplete(id) && id > 0){  
 SessionManager.*storeLoginSession*(this, id);  
 SessionManager.*GoToMain*(this);  
 }  
  
 Button startButton = findViewById(R.id.*startButton*);  
 startButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 openLogin();  
 }  
 });  
}  
public void openLogin(){  
 Intent intent = new Intent(this, Login.class);  
 startActivity(intent);  
}

Figure 53 Starting activity code fragment

public static long getLoginSession(Context context){  
 SharedPreferences sharedPrefs = context.getSharedPreferences("user", Context.*MODE\_PRIVATE*);  
 return sharedPrefs.getLong("user", -1);  
}

Figure 54 SessionManager id storing

## **Task #19. (Defense) Display upcoming activities in calendar view.**

To display future activities in calendar, an additional LinearLayout was added for storing created TextViews. List retrieval used a similar function to one in the home segment for getting selected day activities.

A screenshot of a phone

Description automatically generated

Figure 55 Displayed future activities

// gets current weekday  
String day = fullDate.getDayOfWeek().name();  
  
long id\_plan = db.planHistoryDAO().getUserPlan(id\_user, CommonMethods.*getToday*());  
List<PhysicalActivity> list = db.plansFromActivitiesDAO().getActivitiesOfDay(id\_plan, day);  
  
for (PhysicalActivity act : list) {  
 TextView tv = new TextView(getContext());  
 tv.setText(act.getActivity\_name() + ": " + act.getDuration());  
 tv.setTextSize(18);  
 ((LinearLayout) view.findViewById(R.id.*activity\_container*)).addView(tv);  
  
}

Figure 56 Added code to calendar onItemClick listener

# **Reference list**

1. <https://www.hss.edu/conditions_burning-calories-with-exercise-calculating-estimated-energy-expenditure.asp>
2. <https://www.omicsonline.org/articles-images/2157-7595-6-220-t003.html>
3. <https://www.today.com/health/diet-fitness/weekly-workout-plan-rcna36090>
4. <https://github.com/lopspower/CircularProgressBar>
5. <https://github.com/PhilJay/MPAndroidChart>