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

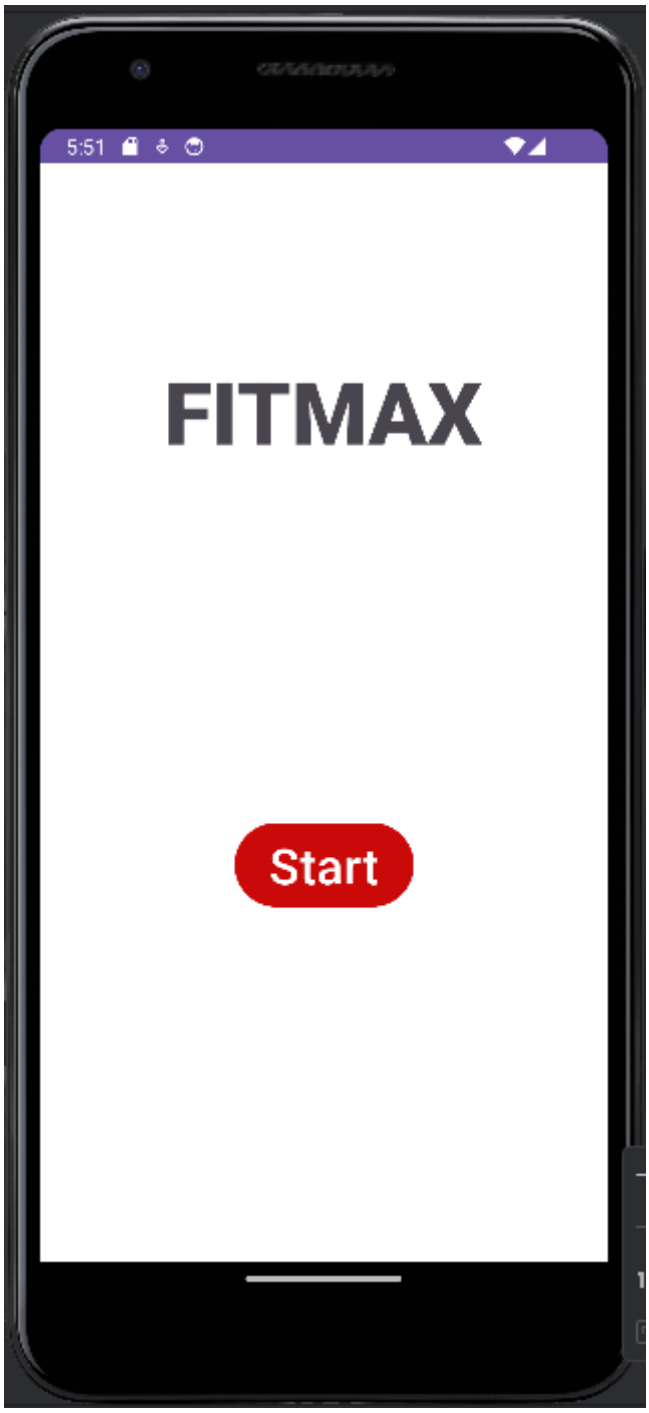
Tables of Contents

Description of Your app

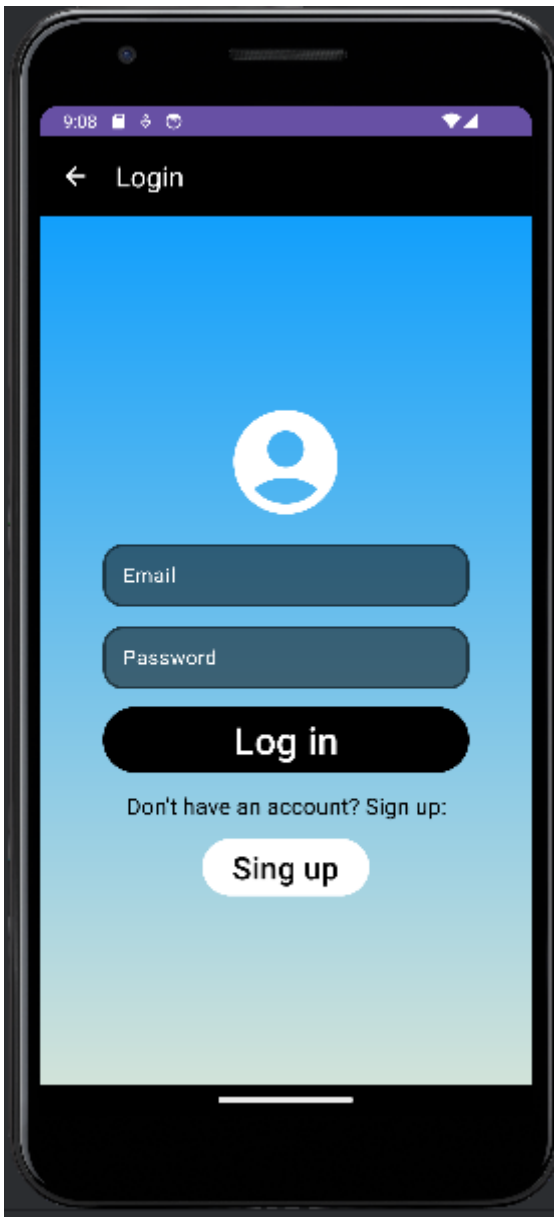
1. What type is your application/game?
    - *Fitness Tracker App - allows users to track their daily physical activities, set fitness goals, and monitor their progress over time:*
2. Description.
    - **User Interface Design:**
        - Design a user-friendly interface with screens for tracking different 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

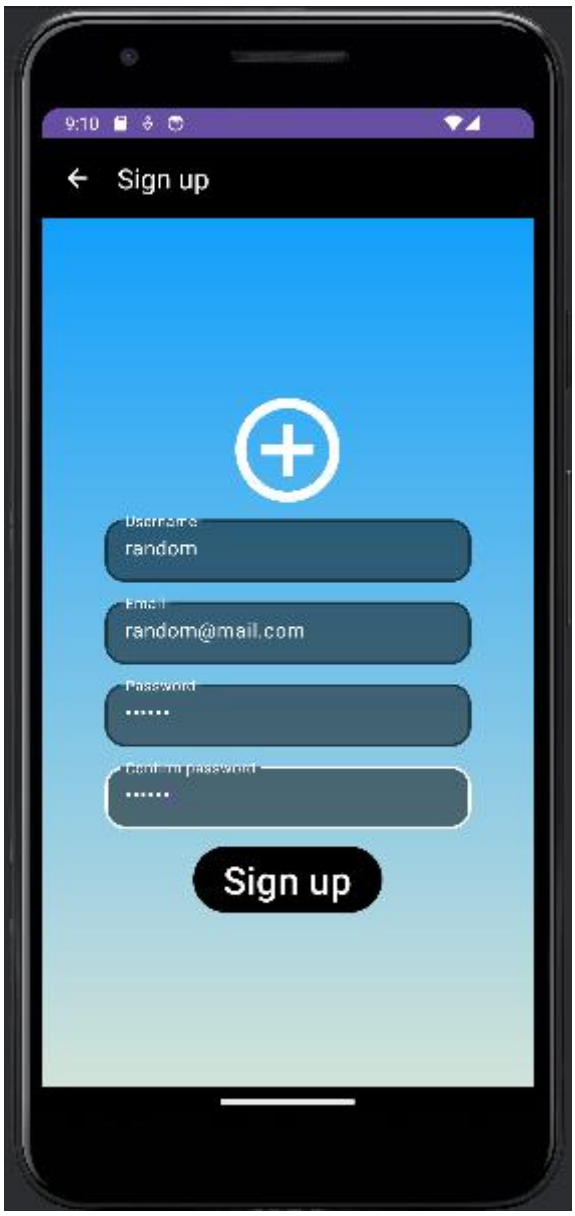
1. Press “Start” in the main menu:



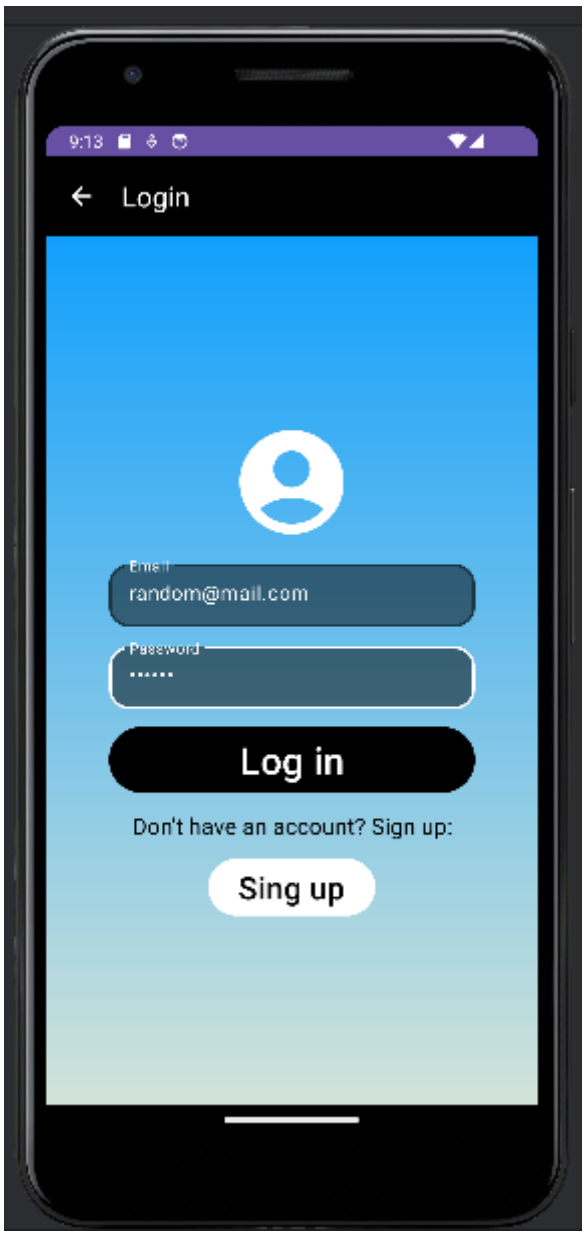
2. Click on “sign up” to create an account:



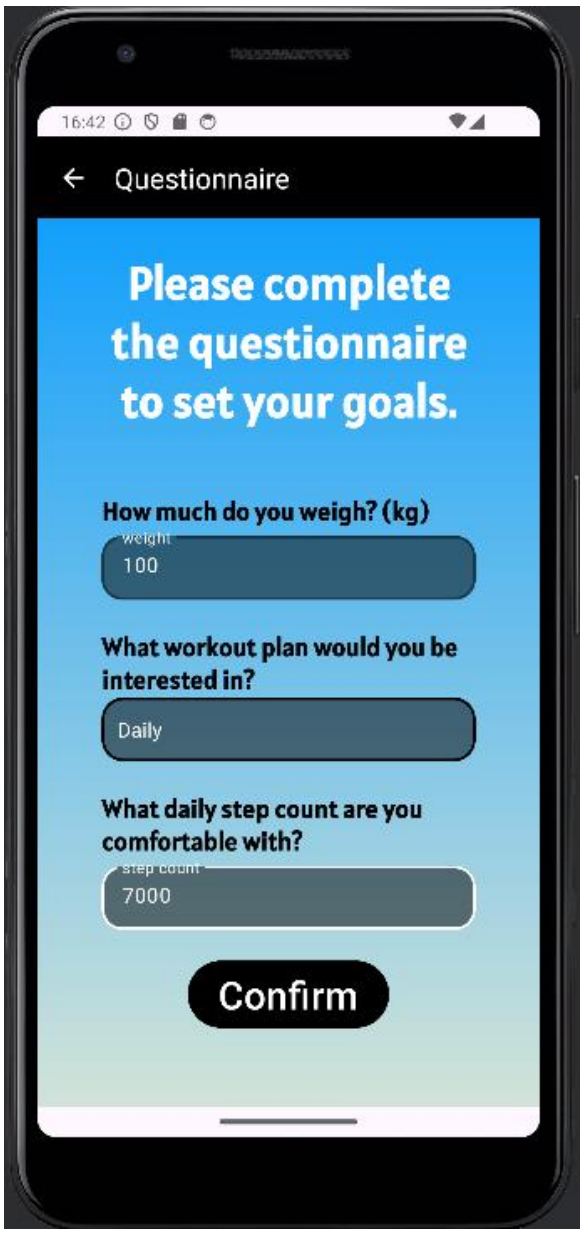
3. Fill in the form:



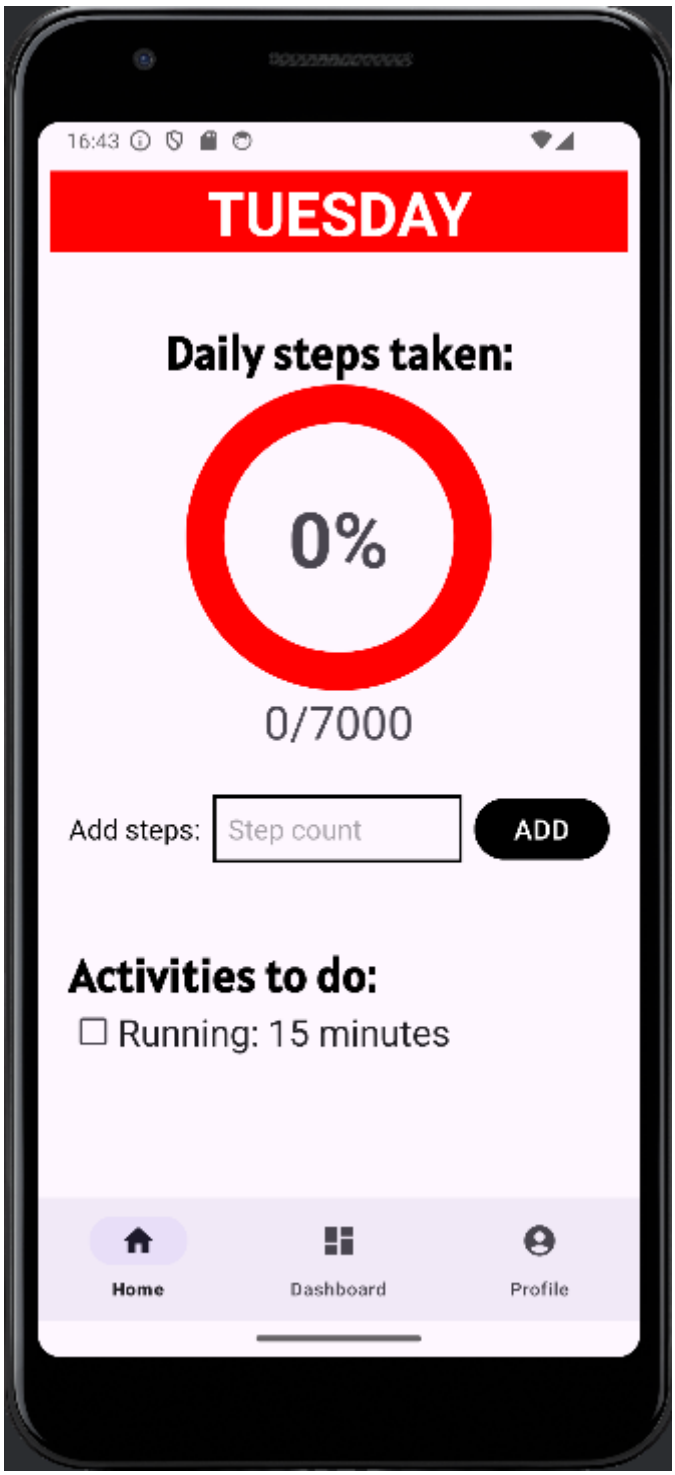
4. Log in with new account:



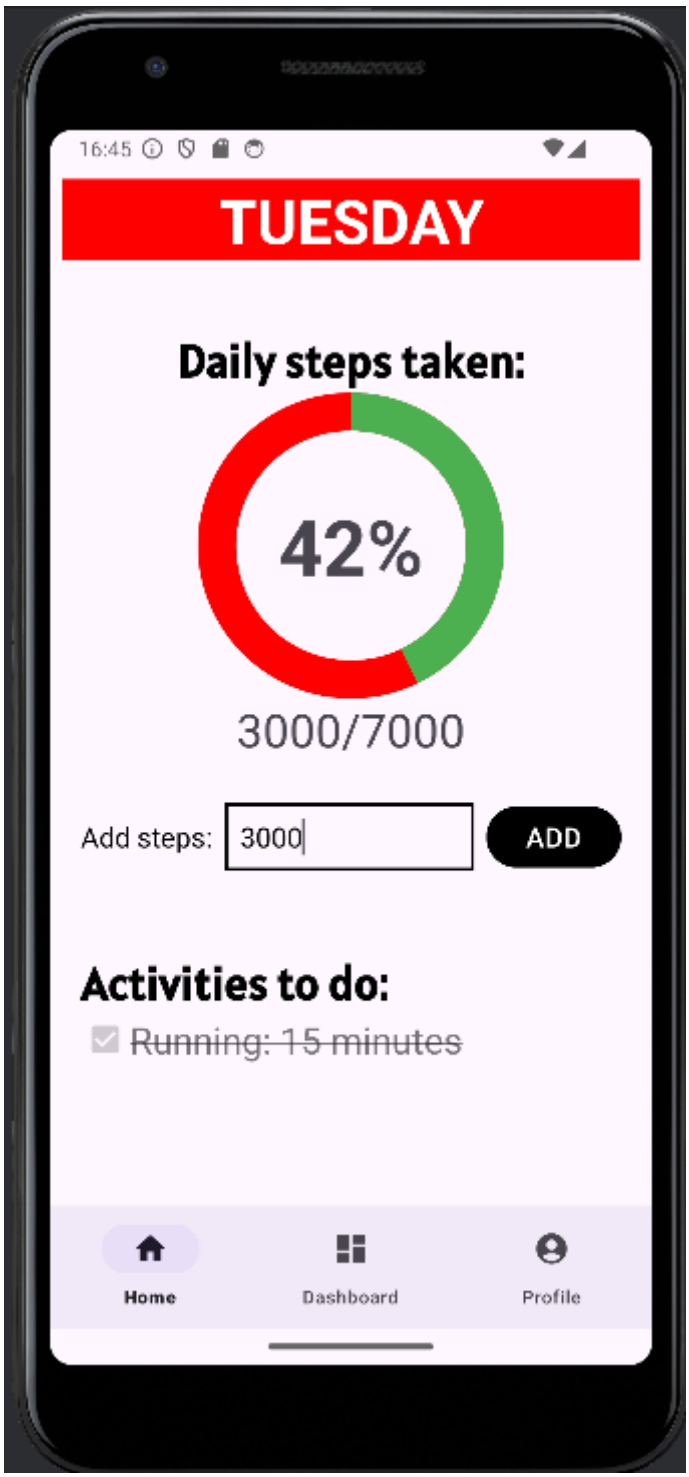
5. Fill out questionnaire:



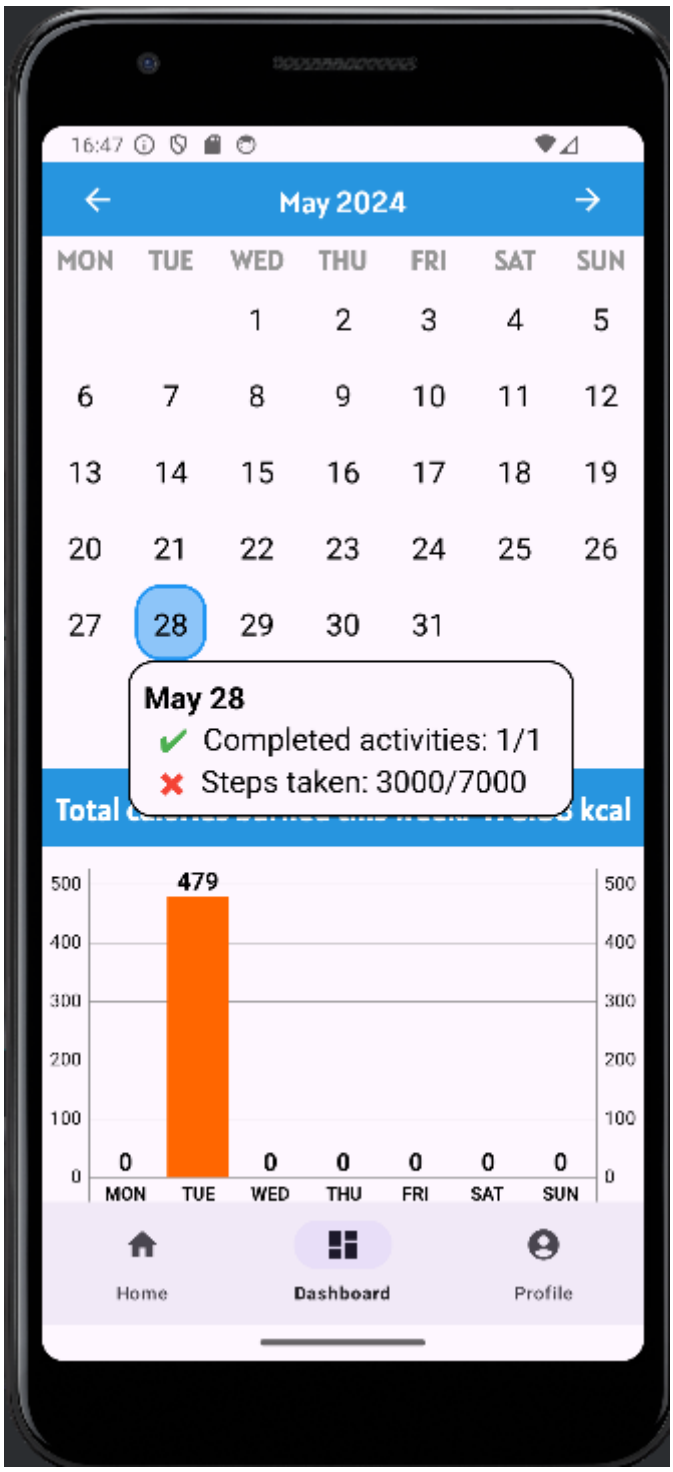
6. Go to “Home” screen to see your daily tasks:



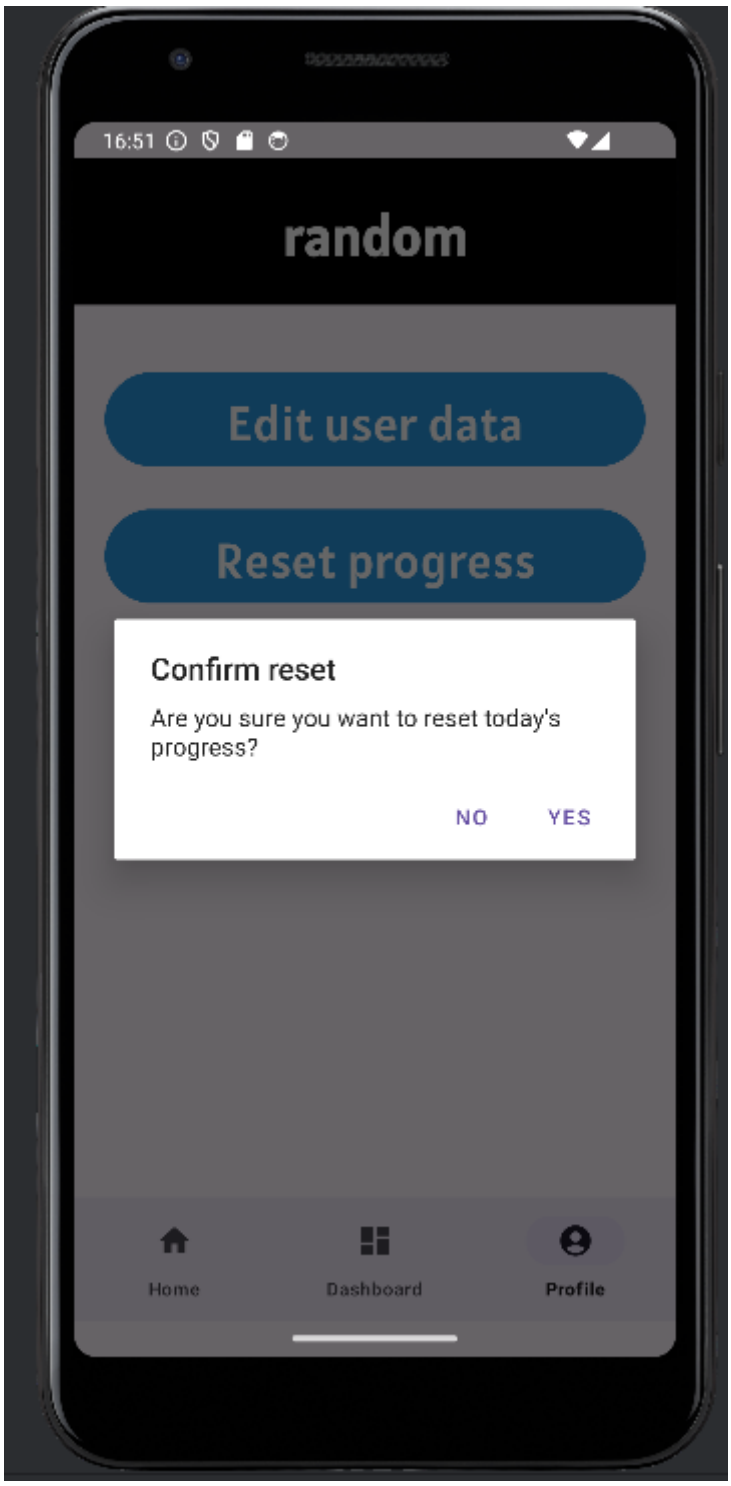
7. Add steps to input field and click the checkbox on any selected activities:



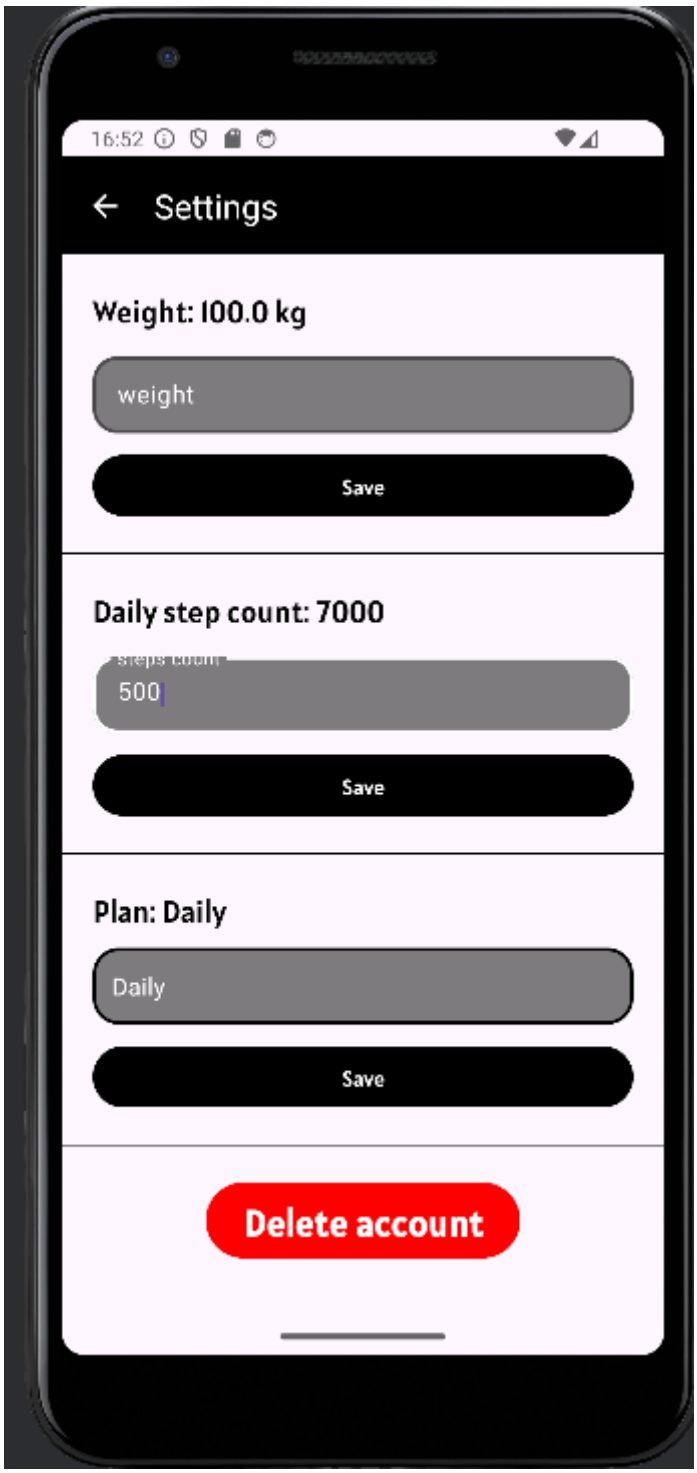
8. Go to the dashboard fragment and click on the current day to see completion progress:



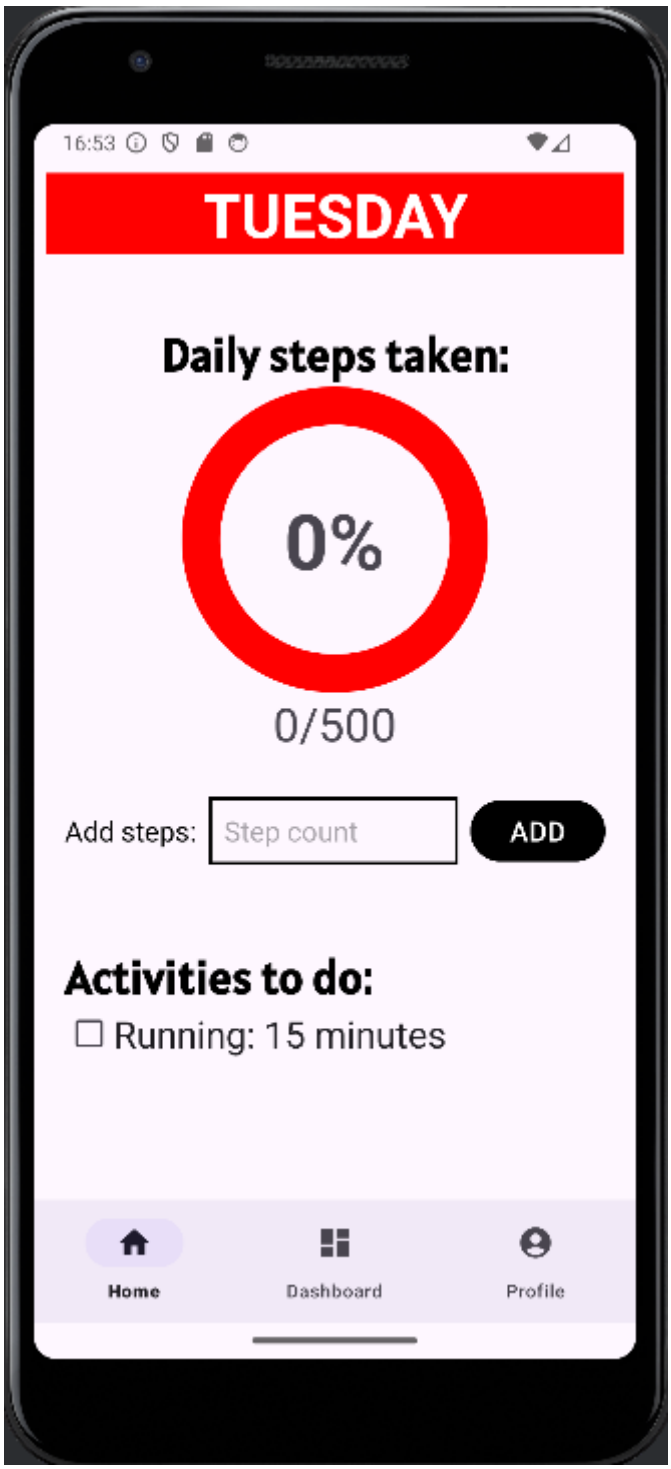
9. Go to the profile fragment, and click on “Reset progress” and then confirm:



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



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



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.

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.

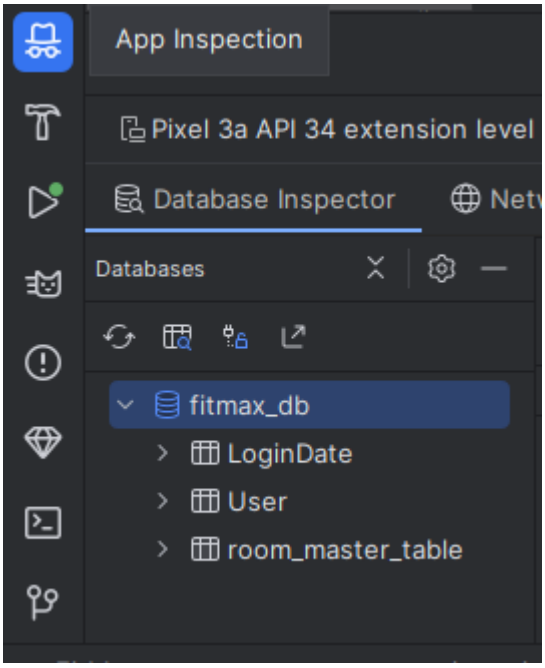


Figure 1. Database layout

```
@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

```java
@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

```java
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

```java
@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.

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.

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.

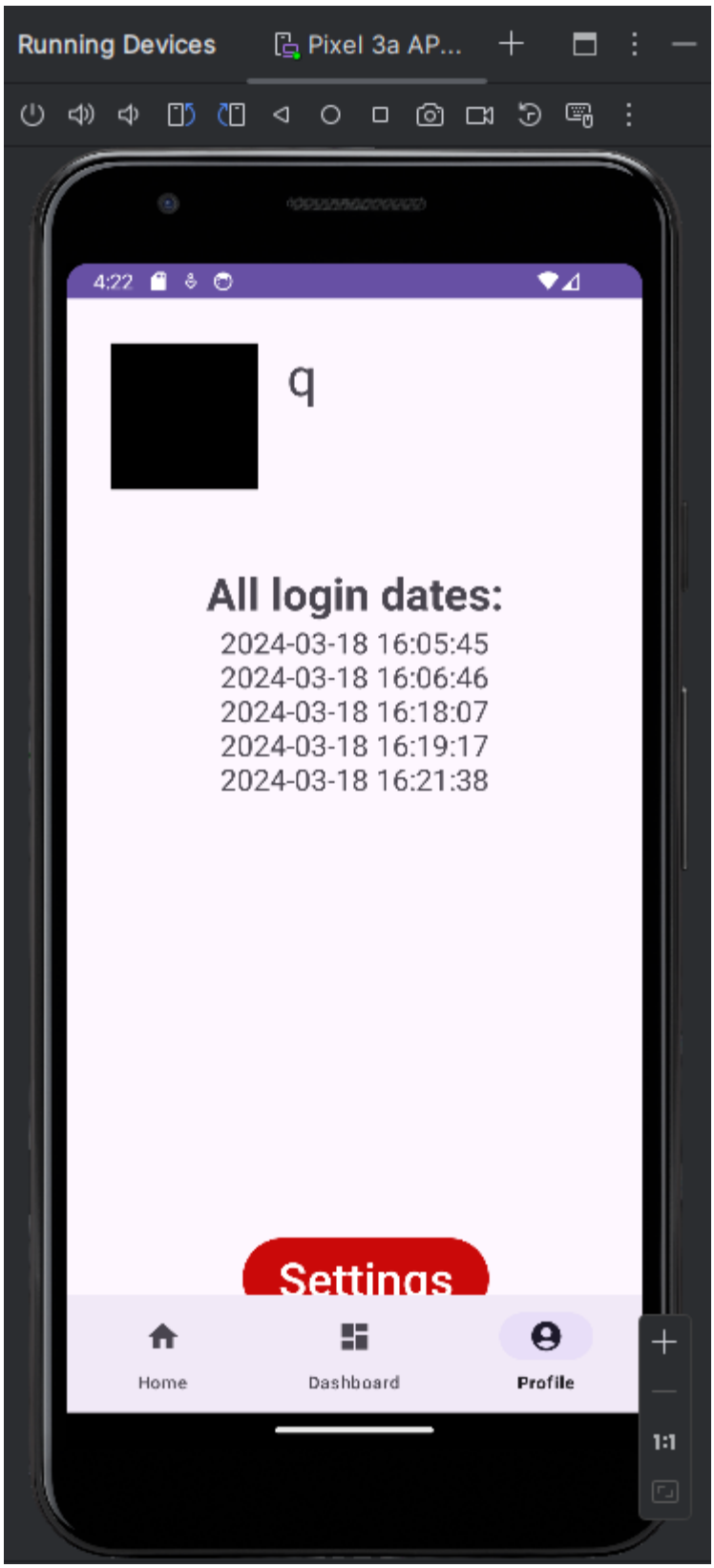


Figure 10 Login date tracker

```
// 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;
```

Figure 13 PhysicalActivity table code fragment

```java
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;
    }
```

Figure 14 Plan table code fragment

```java
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;
```

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.

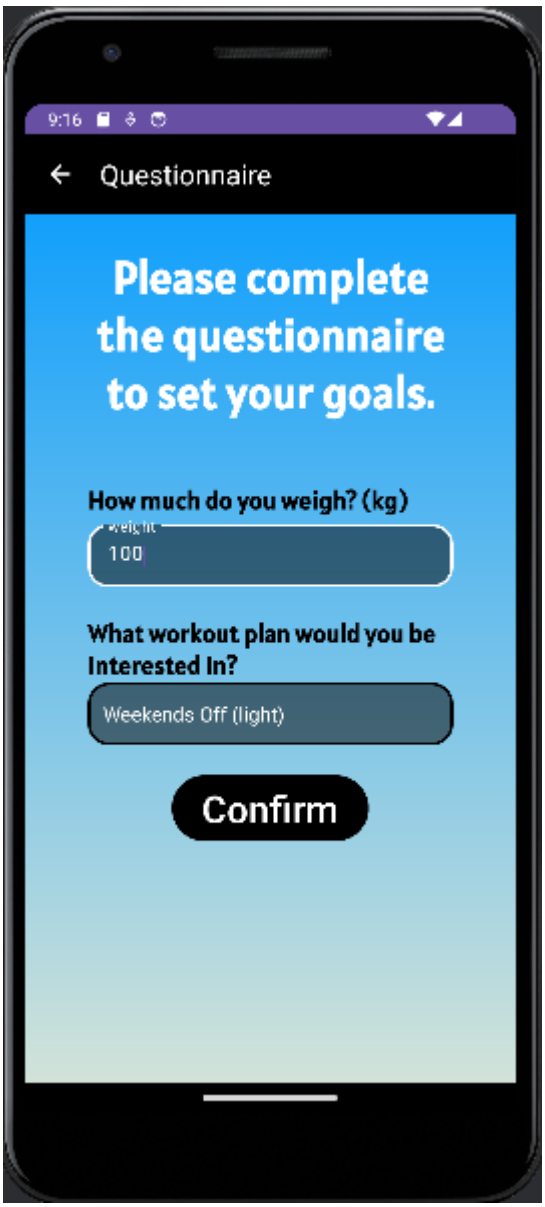


Figure 17 Questionnaire activity

```
// 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

```java
@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(tru
e);
        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;
                }
```

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”.

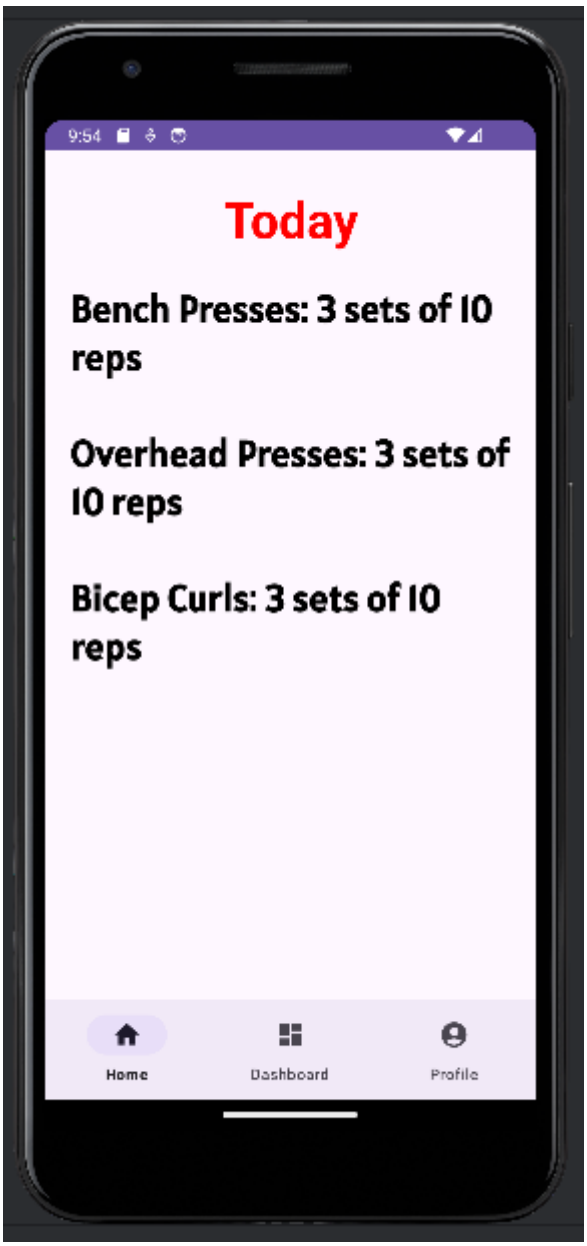


Figure 21 Displayed tasks in home fragment

```
@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.

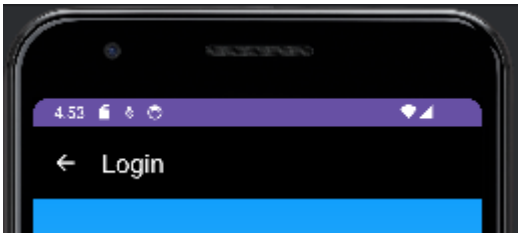


Figure 24 Appbar in login screen

```
<?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(tru
e);
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.

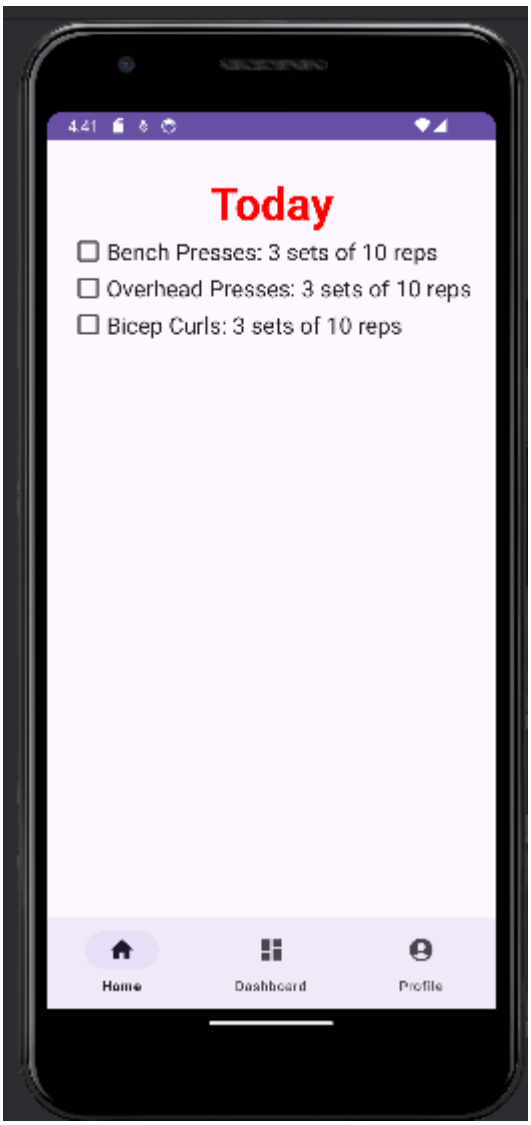


Figure 27 Activity checklist

```
@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;
```

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.

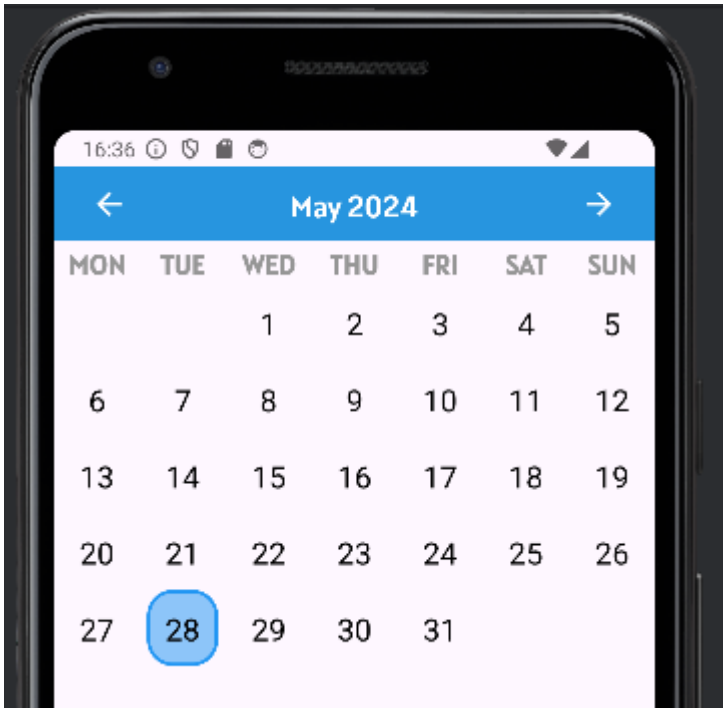


Figure 30 Calendar display

```
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.

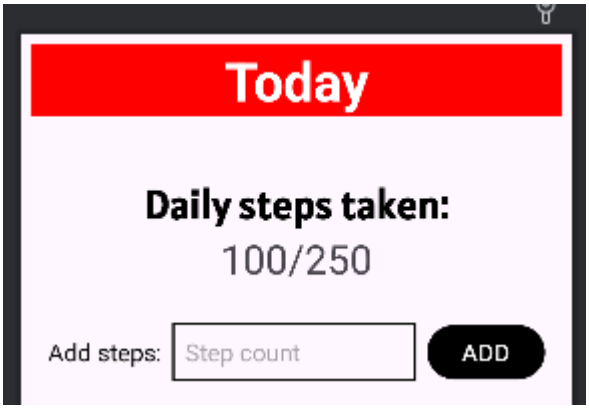


Figure 34 Daily step counter

```
@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;
```

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.

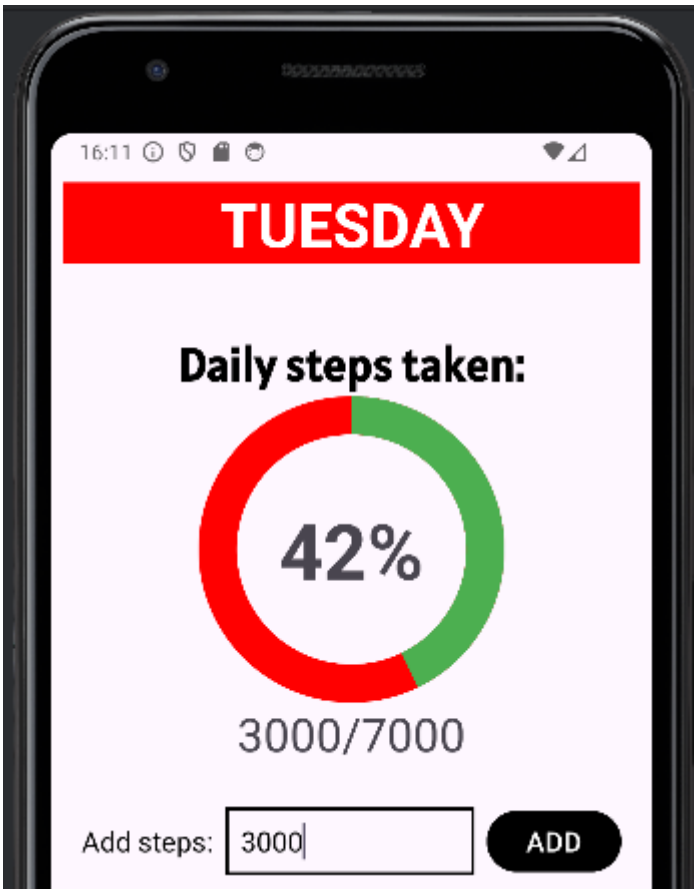


Figure 37 Step count progress bar

```
<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.

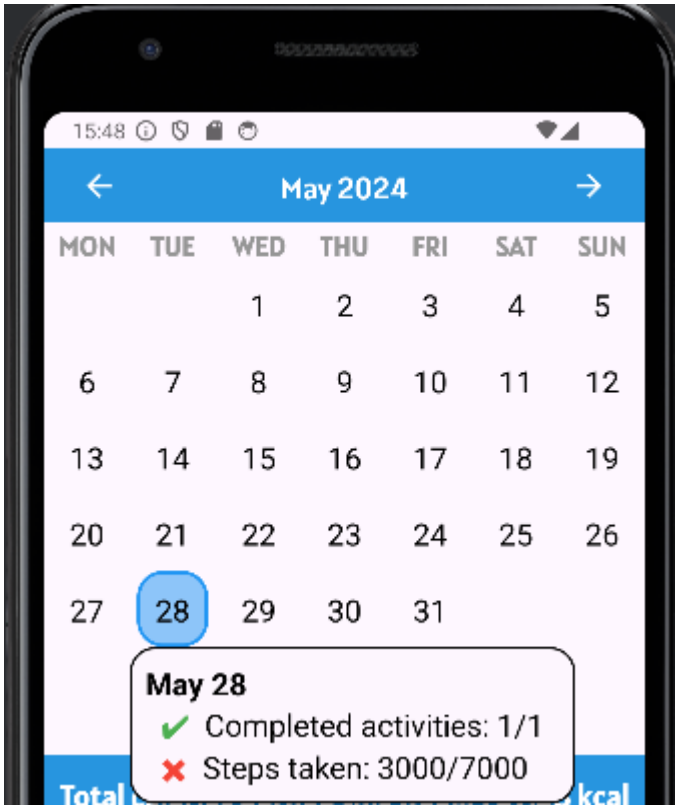


Figure 40 Calendar popup

```
@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.

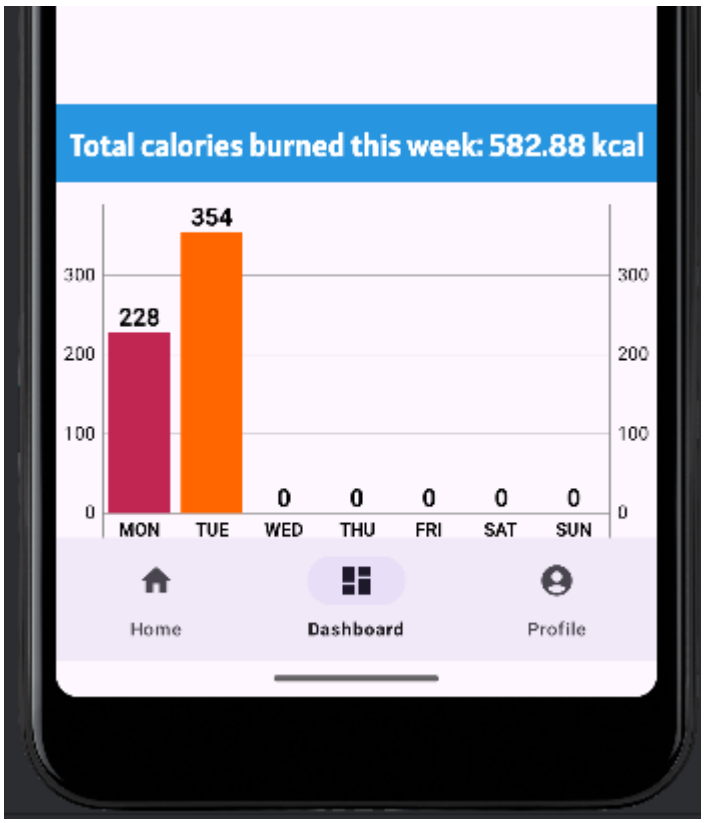


Figure 42 Weekly calorie loss chart

```
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);
```

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.

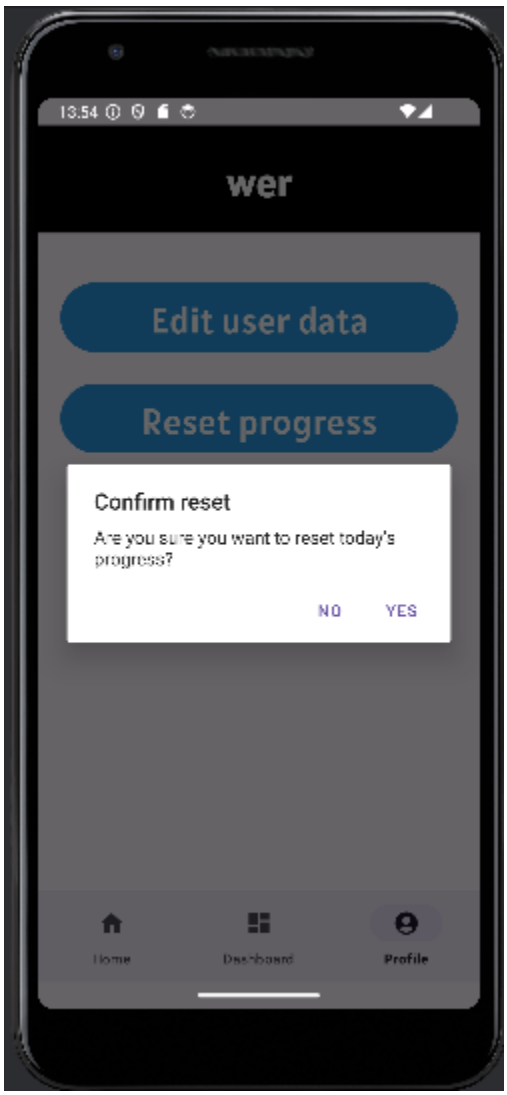


Figure 46 Daily progress reset function

```
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.

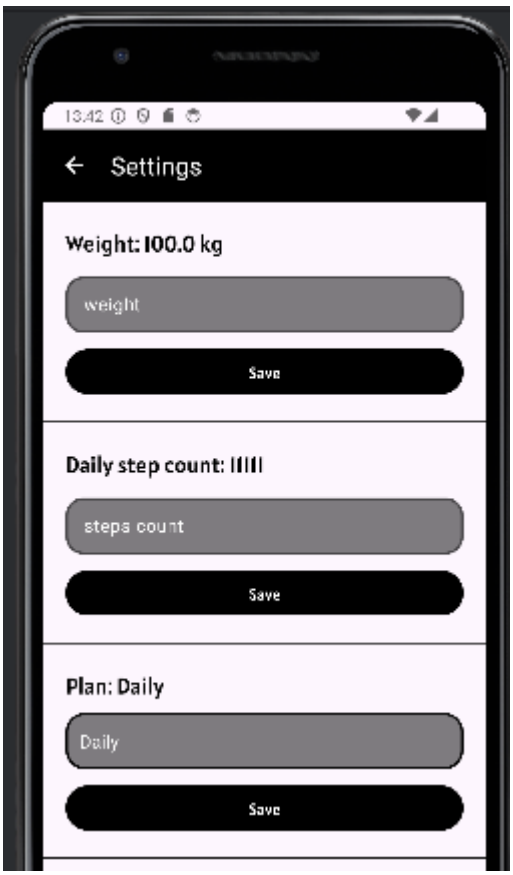


Figure 48 User data change screen

```
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.

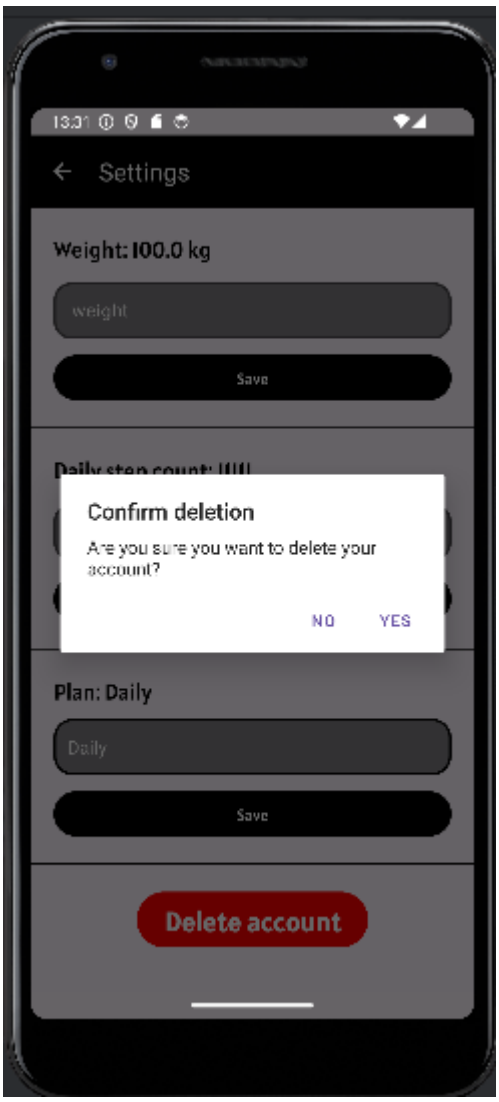


Figure 51 Account deletion option

```
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

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