- Module Code: CS2PJ20
- Assignment report Title: Spring Android Studio App coursework
- Date (when the work completed): 25/03/2024
- Actual hrs spent for the assignment: 16

1. Overview

a. A brief overview of TwentyOneDays

The TwentyOneDays is a habit-tracking application designed to confront the modern challenge of procrastination, exacerbated by social media distractions. Targeted at individuals aiming to enhance their productivity and consistency in habit cultivation, TwentyOneDays is built to integrate effortlessly into daily life, facilitating the tracking and monitoring of habit progress. The app's name, "TwentyOneDays," is inspired by the popular belief that it takes 21 days to form a new habit, symbolizing the app's commitment to helping users embark on a journey of personal improvement and transformative change.

The name "TwentyOneDays" was meticulously chosen to reflect the app's essence and mission. It is based on the concept that consistent effort over a period of 21 days can significantly contribute to the formation of lasting habits. This name highlights the transformative journey users undertake with the app, emphasizing progress, commitment, and the potential for change.

b. Objectives

The development of TwentyOneDays centres around three key objectives:

User-Friendly: ensuring the app is accessible to anyone and straightforward, with a focus on a simple yet effective interface to encourage regular use.

Adaptable: catering to a wide array of habits and goals by supporting multiple input methods, allowing users to personalise their habit-tracking experience.

Motivational: utilising visual indicators of progress and motivational elements to keep users engaged and motivated to maintain their habit streaks.

2. Application Specifications

a. Describing of TwentyOneDays technical specifications:

TwentyOneDays features a modular application structure, with each screen represented as an activity that includes an XML layout and a Java class. These components work together to facilitate smooth user interactions and backend processes. Highlights include:

Welcome Screen: designed with interactive elements to guide users through the app’s features.

User Authentication: secured with Firestore for robust data management, employing dedicated classes for managing user interactions; Utilising a trusted platform like Firebase for Authentication, allows us to provide a seamless user experience, the serverless architecture provided through Firebase may cause some initial concern since there isn’t a central server to be able to protect and filter through requests; however, Firebase allows developers to integrate security rules in order to secure against misuse when developing full scale applications, which will be something that will be considered in the future. Using Firebase allows users to provide an email and think of their own password, which makes it a low barrier to entry for our audience.

Functionality: tailored functions for some of the functions that users may want to access, for example, being able to make a habit, being able to see a habit and other features like being able to tweak settings.

b. User Interface Design

The app embraces a visually soothing theme, using colors and designs that evoke tranquility and trust, making users feel comfortable and secure as they log their activities and track their progress.

I will try to use rounded edges and a more modern font family as rounded edges can improve usability by reducing the risk of accidental taps or swipes on sharp corners, especially on mobile devices where precision can be challenging. This can lead to a smoother and more frustration-free user experience, encouraging users to interact with the app more frequently. Modern font families are often designed with legibility and readability in mind, making them suitable choices for digital interfaces. These fonts typically feature clean and well-defined letterforms, which can improve readability on various screen sizes and resolutions.

To be a successful application, I would have to have a list of features that need to be implemented:

<u>Basic Authentication:</u>

- Users are able to enter an email, username, password and are able to register for the application in an intuitive way
- Users are able to log into the application with their username and password reliably.

<u>Habit System</u>

- A user is able to view a specific habit that they might want to record.

There are more aspirational features that I would want to implement that would extend upon the features mentioned prior:

- a Social system that allows users to search to see if their friends are using the application and then add them, this will allow them to see their friends and if they are keeping on top of their habits.

3. Application Implementation

a. Basic functionality 1 – User Authentication Sign in

The Login activity is responsible for authenticating users using email and password credentials. Here's a summary of its main processes and how it provides user authentication:

Layout Setup:
In the onCreate() method, the activity layout is set using setContentView(). References to the email input field (editEmail), password input field (editPassword), login button (loginBtn), and sign-up redirect text (signUpRedirectTxt) are initialized.

Firebase Authentication Initialization:
An instance of FirebaseAuth (mAuth) is obtained using FirebaseAuth.getInstance().

Sign-In Functionality:
The signIn() method is called when the login button (loginBtn) is clicked. It retrieves the email and password entered by the user from the input field It uses the signInWithEmailAndPassword() method of mAuth to attempt to sign in the user with the provided credentials. A listener is added to handle the asynchronous result of the sign-in attempt. If the sign-in attempt is successful (task.isSuccessful()), the user is directed to the main application activity (Drawer), and the user's unique ID (UID) is passed along with the intent, for easy access of the users information, to be able to display information like their username.
If the sign-in attempt fails, an appropriate error message is displayed to the user via a toast. And they are able to resubmit their password and username.

Sign-Up Redirection:
The signUpRedirectTxt TextView is clickable and triggers an intent to navigate to the sign-up activity (SignUp) when clicked.

Overall, the Login activity integrates with Firebase Authentication to securely authenticate users, and upon successful authentication, it directs them to the main application interface. If authentication fails, appropriate error messages are displayed to the user.

The Sign-Up activity is responsible for registering new users by collecting their email, password, and username credentials. Below is a breakdown of its main functions and how it facilitates the user registration process:

Layout Setup:
In the onCreate() method, the activity layout is set using setContentView().
References to the username input field (editUsername), email input field (editEmail), password input field (editPassword), sign-up button (signupButton), and login redirect text (loginRedirectText) are initialized.

Firebase Authentication Initialization:
An instance of FirebaseAuth (mAuth) is obtained using FirebaseAuth.getInstance().

b. Basic functionality 2 – User Authentication Sign up / out

Sign-Up Functionality:
The onClick listener is attached to the sign-up button (signupButton). When the button is clicked, the input fields for email, password, and username are retrieved. Input field validation is performed to ensure all required fields are filled. The createUserWithEmailAndPassword() method of mAuth is used to attempt to create a new user account with the provided email and password.

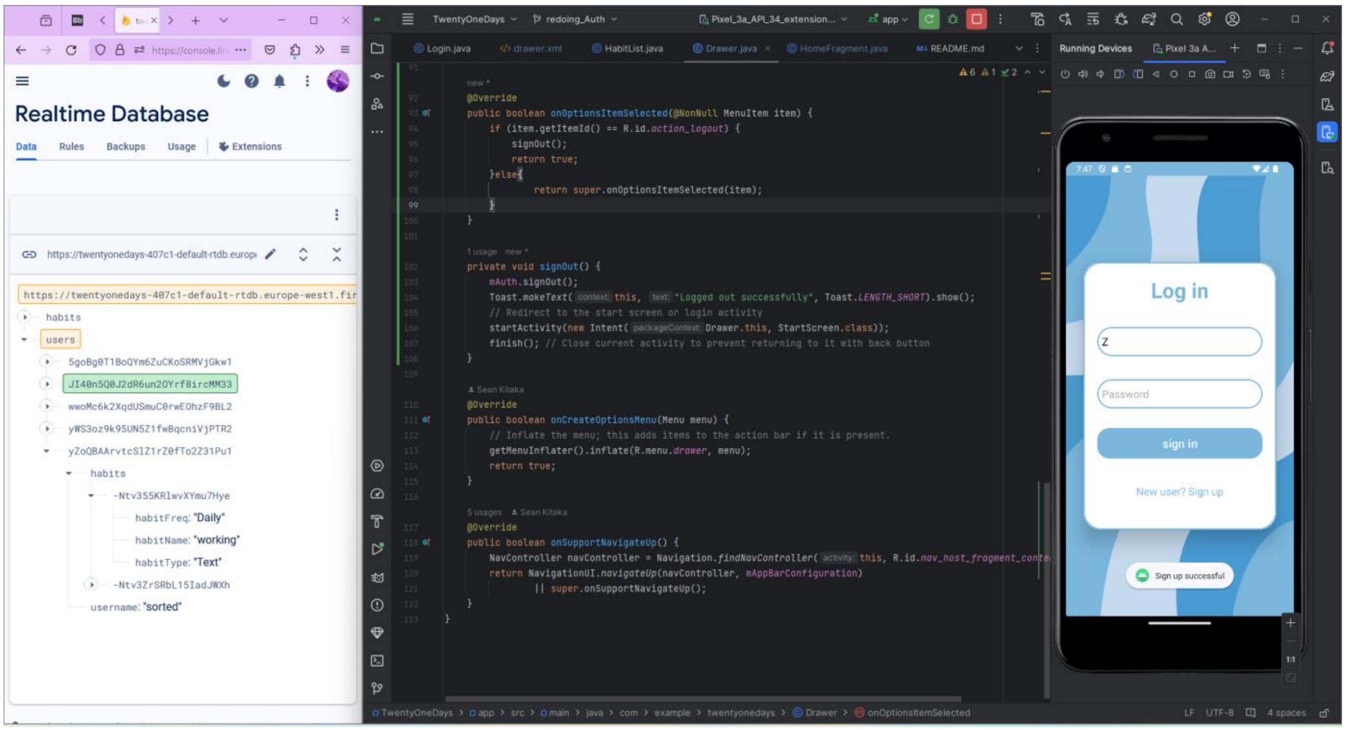
A listener is added to handle the asynchronous result of the sign-up attempt, if the sign-up attempt is successful (task.isSuccessful()), the user's data, including their username, is stored in the Firebase Realtime Database under the "users" collection. A success message is displayed to the user, and they are redirected to the login activity (Login.class) to sign in with their newly created account. If the sign-up attempt fails, an error message containing details of the failure is displayed to the user via a toast.
Login Redirection:
The login redirect text (loginRedirectText) is clickable and triggers an intent to navigate back to the login activity (Login.class) when clicked.

Figure 1 displays the authentication process from the admin's perspective, as the UID is generated when saving the data to the collection of users and the data for the user is nested within the UID to maintain data consistency, hence why it is useful to pass the UID to other activities. Since the authentication is separate to this database updating, the username is the only thing that is associated with the new UID at the moment.

Figure 2 is a look at the Signup page, it only prompts the user for a username, email and password, to avoid the perception of being invasive with the amount of data that is requested, since anything other than a user name, password and email would be unnecessary.

Relying on Firebase authentication allowed me to abstract out any of the other functions or features that I would have to otherwise implement, such as checking if a username was taken, checking

Logout Button:
Although not explicitly mentioned in the code provided, it's common in authentication flows to include a logout button. This button would log out the current user from their account and bring them back to the start screen or login page, depending on the application's design however within my application I direct the user back to the start screen, they may want to directly sign up with another account.

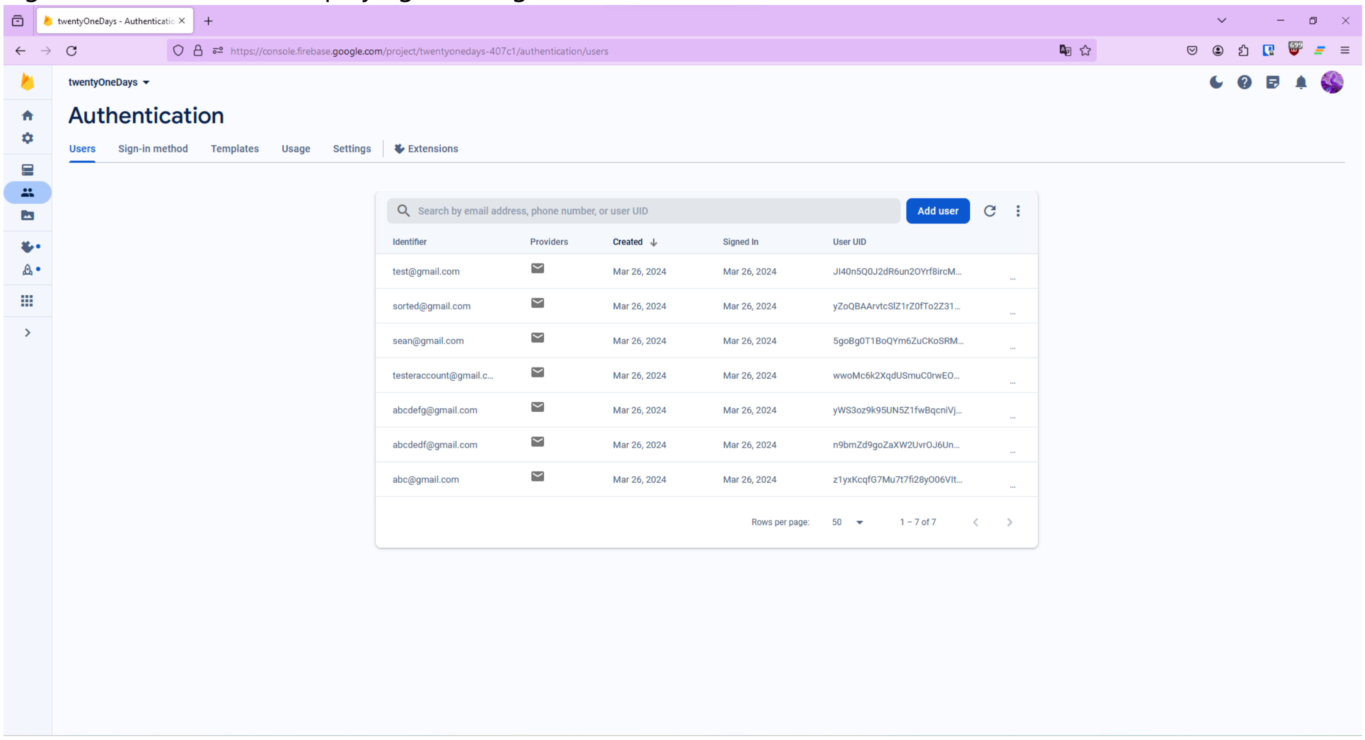


Realtime Database

```
https://twentyonedays-407c1-default-rtdb.europe-west1.fir
habits
users
  5goBg0T1BoQYm6ZuCKoSRMVjGkw1
  JI40n5Q0J2dR6un2OYrf8ircMM33
  wwoMc6k2XqdUSmuC0rwEOhzF9BL2
  yWS3oz9k95UN5Z1fwBqcniVjPTR2
  yZoQBAArvtcSlZ1rZ0fTo2Z31Pu1
    habits
      -Ntv355KRlwvXYmu7Hye
        habitFreq: "Daily"
        habitName: "working"
        habitType: "Text"
      -Ntv3ZrSRbL15IadJWXh
    username: "sorted"
```

```java
    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item) {
        if (item.getItemId() == R.id.action_logout) {
            signOut();
            return true;
        }else{
                return super.onOptionsItemSelected(item);
        }
    }

    private void signOut() {
        mAuth.signOut();
        Toast.makeText(this, "Logged out successfully", Toast.LENGTH_SHORT).show();
        // Redirect to the start screen or login activity
        startActivity(new Intent(Drawer.this, StartScreen.class));
        finish(); // Close current activity to prevent returning to it with back button
    }

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

    @Override
    public boolean onSupportNavigateUp() {
        NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment_conte
        return NavigationUI.navigateUp(navController, mAppBarConfiguration)
                || super.onSupportNavigateUp();
    }
}
```

Figure 1 – Screenshots displaying user Login

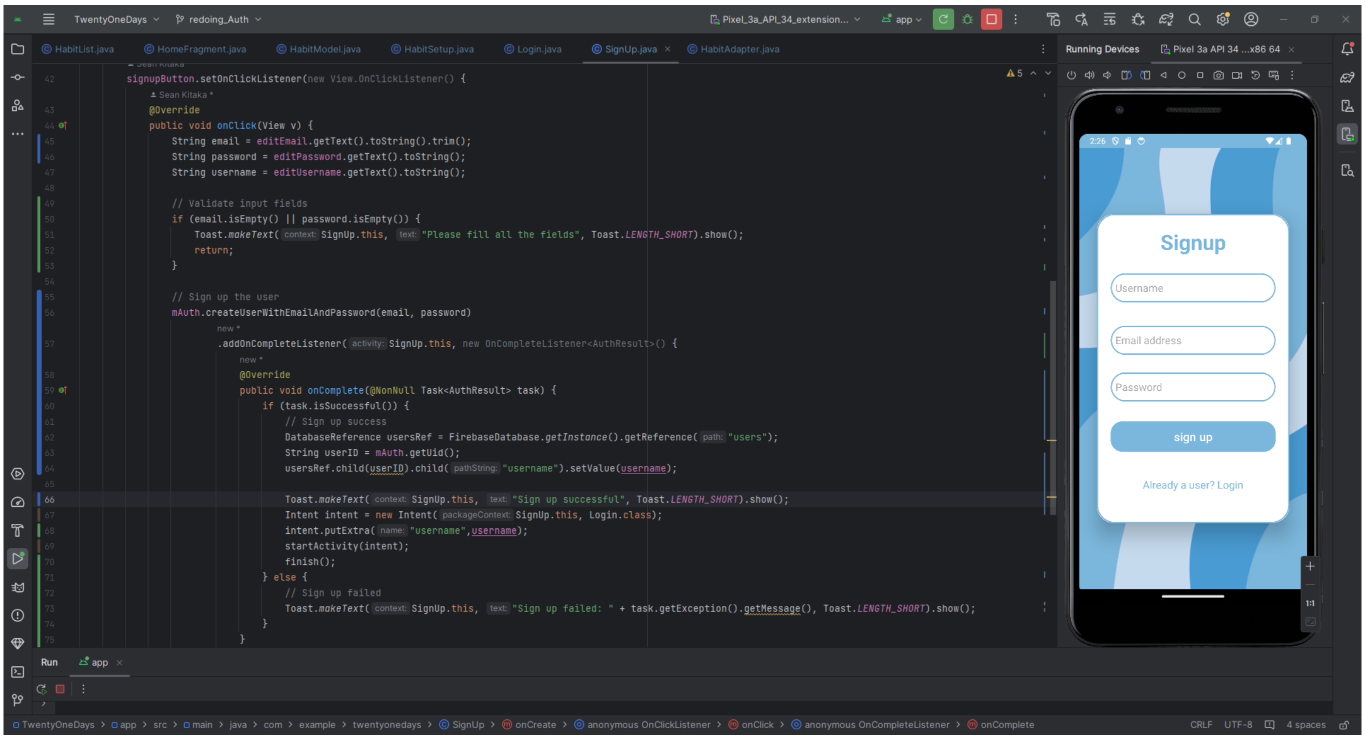


Authentication

Identifier	Providers	Created	Signed In	User UID
test@gmail.com	✉	Mar 26, 2024	Mar 26, 2024	JI40n5Q0J2dR6un2OYrf8ircM...
sorted@gmail.com	✉	Mar 26, 2024	Mar 26, 2024	yZoQBAArvtcSlZ1rZ0fTo2Z31...
sean@gmail.com	✉	Mar 26, 2024	Mar 26, 2024	5goBg0T1BoQYm6ZuCKoSRM...
testeraccount@gmail.c...	✉	Mar 26, 2024	Mar 26, 2024	wwoMc6k2XqdUSmuC0rwEO...
abcdefg@gmail.com	✉	Mar 26, 2024	Mar 26, 2024	yWS3oz9k95UN5Z1fwBqcniVj...
abcdedf@gmail.com	✉	Mar 26, 2024	Mar 26, 2024	n9bmZd9goZaXW2UvrOJ6Un...
abc@gmail.com	✉	Mar 26, 2024	Mar 26, 2024	z1yxKcqfG7Mu7t7fi28yO06VIt...

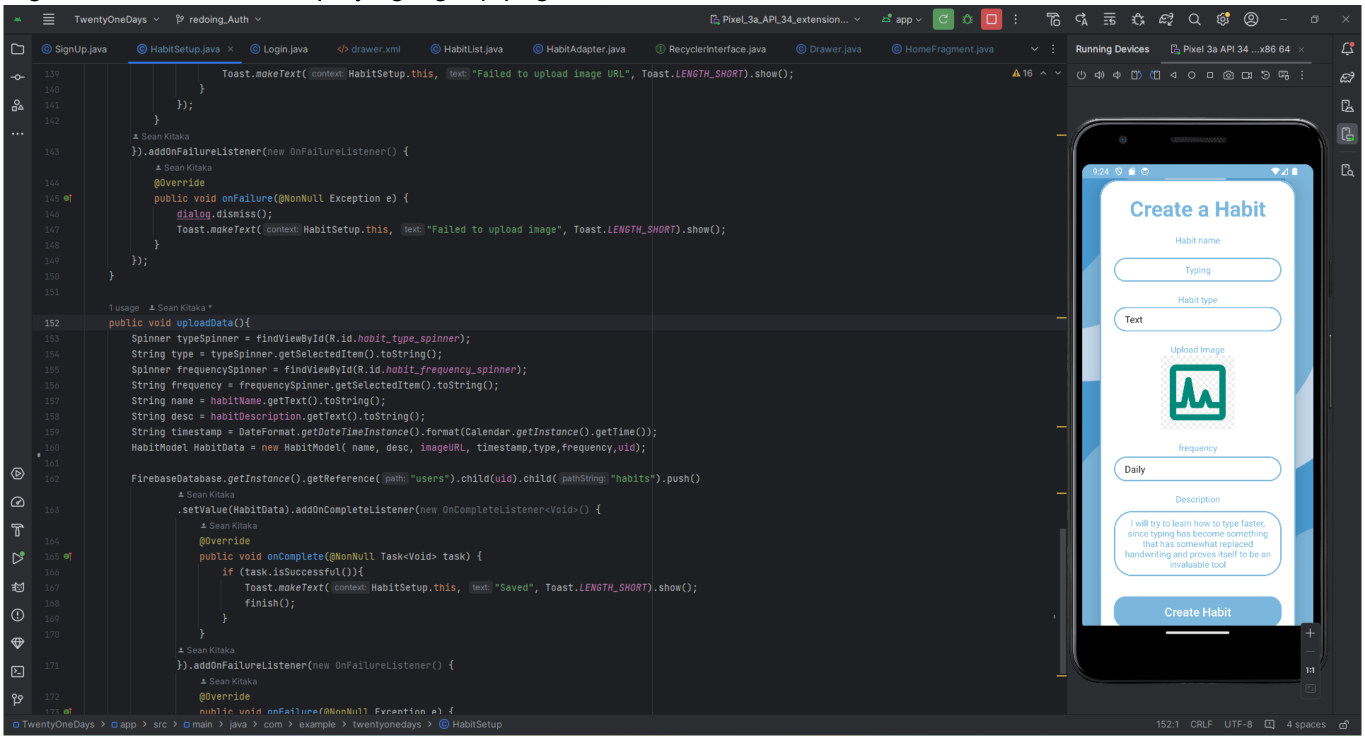
Rows per page: 50 1 – 7 of 7

Figure 2 – Screenshot displaying Firebase Authentication



```java
signupButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String email = editEmail.getText().toString().trim();
        String password = editPassword.getText().toString();
        String username = editUsername.getText().toString();

        // Validate input fields
        if (email.isEmpty() || password.isEmpty()) {
            Toast.makeText(SignUp.this, "Please fill all the fields", Toast.LENGTH_SHORT).show();
            return;
        }

        // Sign up the user
        mAuth.createUserWithEmailAndPassword(email, password)
                .addOnCompleteListener(SignUp.this, new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {
                        if (task.isSuccessful()) {
                            // Sign up success
                            DatabaseReference usersRef = FirebaseDatabase.getInstance().getReference("users");
                            String userID = mAuth.getUid();
                            usersRef.child(userID).child("username").setValue(username);

                            Toast.makeText(SignUp.this, "Sign up successful", Toast.LENGTH_SHORT).show();
                            Intent intent = new Intent(SignUp.this, Login.class);
                            intent.putExtra("username",username);
                            startActivity(intent);
                            finish();
                        } else {
                            // Sign up failed
                            Toast.makeText(SignUp.this, "Sign up failed: " + task.getException().getMessage(), Toast.LENGTH_SHORT).show();
                        }
                    }
```

Figure 3 - Screenshot displaying signup page and Firebase Authentication



```java
                Toast.makeText(HabitSetup.this, "Failed to upload image URL", Toast.LENGTH_SHORT).show();
            }
        });
    }
}).addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        dialog.dismiss();
        Toast.makeText(HabitSetup.this, "Failed to upload image", Toast.LENGTH_SHORT).show();
    }
});
}

public void uploadData(){
    Spinner typeSpinner = findViewById(R.id.habit_type_spinner);
    String type = typeSpinner.getSelectedItem().toString();
    Spinner frequencySpinner = findViewById(R.id.habit_frequency_spinner);
    String frequency = frequencySpinner.getSelectedItem().toString();
    String name = habitName.getText().toString();
    String desc = habitDescription.getText().toString();
    String timestamp = DateFormat.getDateTimeInstance().format(Calendar.getInstance().getTime());
    HabitModel HabitData = new HabitModel( name, desc, imageURL, timestamp,type,frequency,uid);

    FirebaseDatabase.getInstance().getReference("users").child(uid).child("habits").push()
            .setValue(HabitData).addOnCompleteListener(new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    if (task.isSuccessful()){
                        Toast.makeText(HabitSetup.this, "Saved", Toast.LENGTH_SHORT).show();
                        finish();
                    }
                }
            }).addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
```

TwentyOneDays
redoing_Auth
SignUp.java
HabitSetup.java
Login.java
drawer.xml
HabitList.java
HabitAdapter.java
RecyclerInterface.java
Drawer.java
HomeFragment.java
Running Devices
Pixel 3a API 34 ...x86 64
Create a Habit
Habit name
Typing
Habit type
Text
Upload Image
frequency
Daily
Description
I will try to learn how to type faster, since typing has become something that has somewhat replaced handwriting and proves itself to be an invaluable tool
Create Habit
TwentyOneDays > app > src > main > java > com > example > twentyonedays > HabitSetup

Figure 4 – Adding a habit to the Firebase Realtime database

c. Extended Functionality 2 – Habit Tracker

One of the pivotal features of the Habit Tracker application is its implementation of a RecyclerView, which serves as the backbone for displaying and managing users' habits. The RecyclerView is a robust component in Android development, known for for its efficiency in handling large datasets while ensuring smooth performance. Unlike its predecessors like ListView and GridView, the RecyclerView optimizes memory usage by recycling item views as users navigate through the list, thereby enhancing the overall user experience.

To effectively implement the RecyclerView, several essential classes were developed:

1. **HabitModel** Class: This class encapsulates important data fields related to each habit, including its name, type, frequency, and other details. By creating a dedicated model class, we can organize and manage habit data more efficiently, thus improving code readability and maintainability.
2. **HabitAdapter Class**: The HabitAdapter class acts as a bridge between the HabitModel objects and the RecyclerView, facilitating the seamless display of habit data within the UI. It inflates the corresponding view for each habit item and binds the data accordingly, ensuring a cohesive user experience.
3. **HomeFragment Class**: main screen of the application's user interface, the HomeFragment plays a crucial role in fetching habit data from Firebase and passing it to the HabitAdapter for display within the RecyclerView.
4. **RecyclerInterface Interface**: This interface, implemented by the HomeFragment class, defines the onItemClick() method to handle click events on habit items within the RecyclerView, facilitating seamless navigation to detailed habit views.

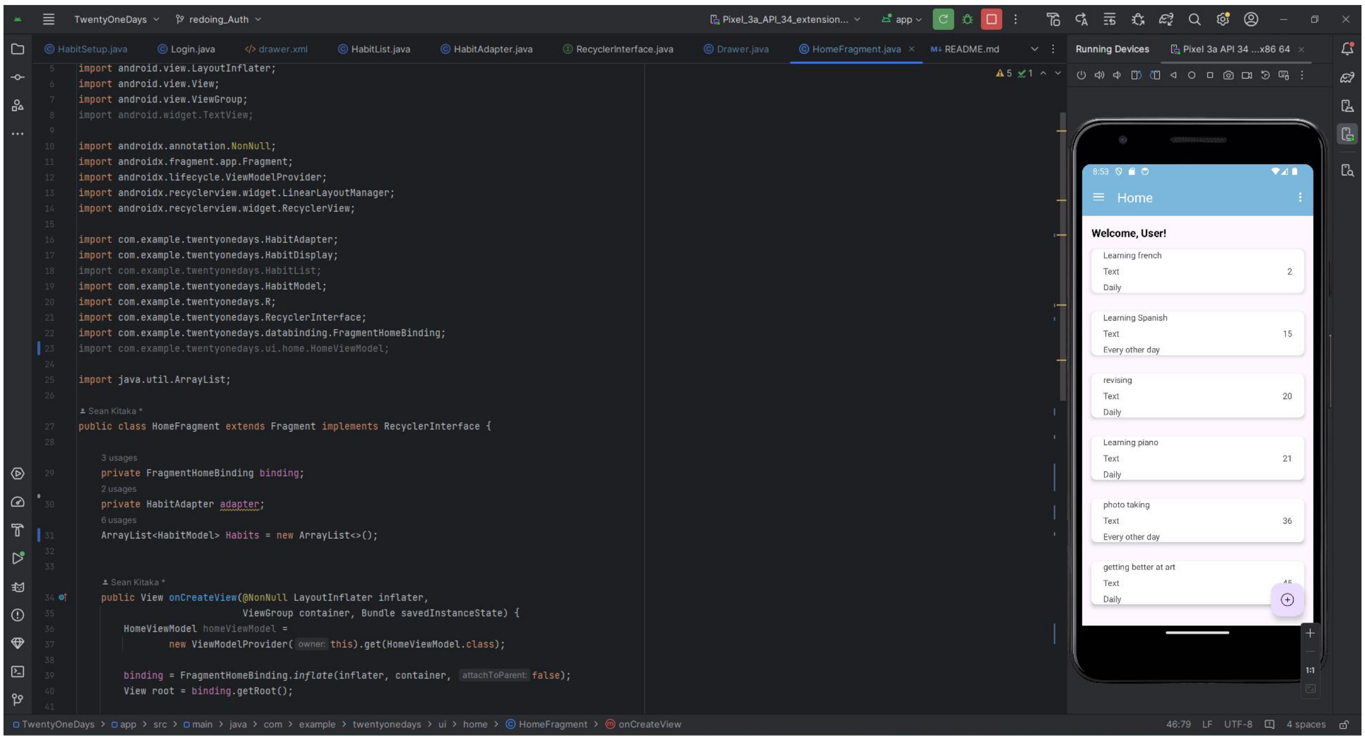
After creating a habit, users have the capability to review their progress by adding media and additional notes to mark milestones along their habit-building journey. This feature enhances user engagement and provides a comprehensive overview of their habit progress over time.

e. Additional Feature 2 – Habit Interactivity

Users can interact with their habits by clicking on individual items within the RecyclerView to access detailed views. These detailed views provide comprehensive information about each habit, empowering users to edit habit features and review their progress effectively.

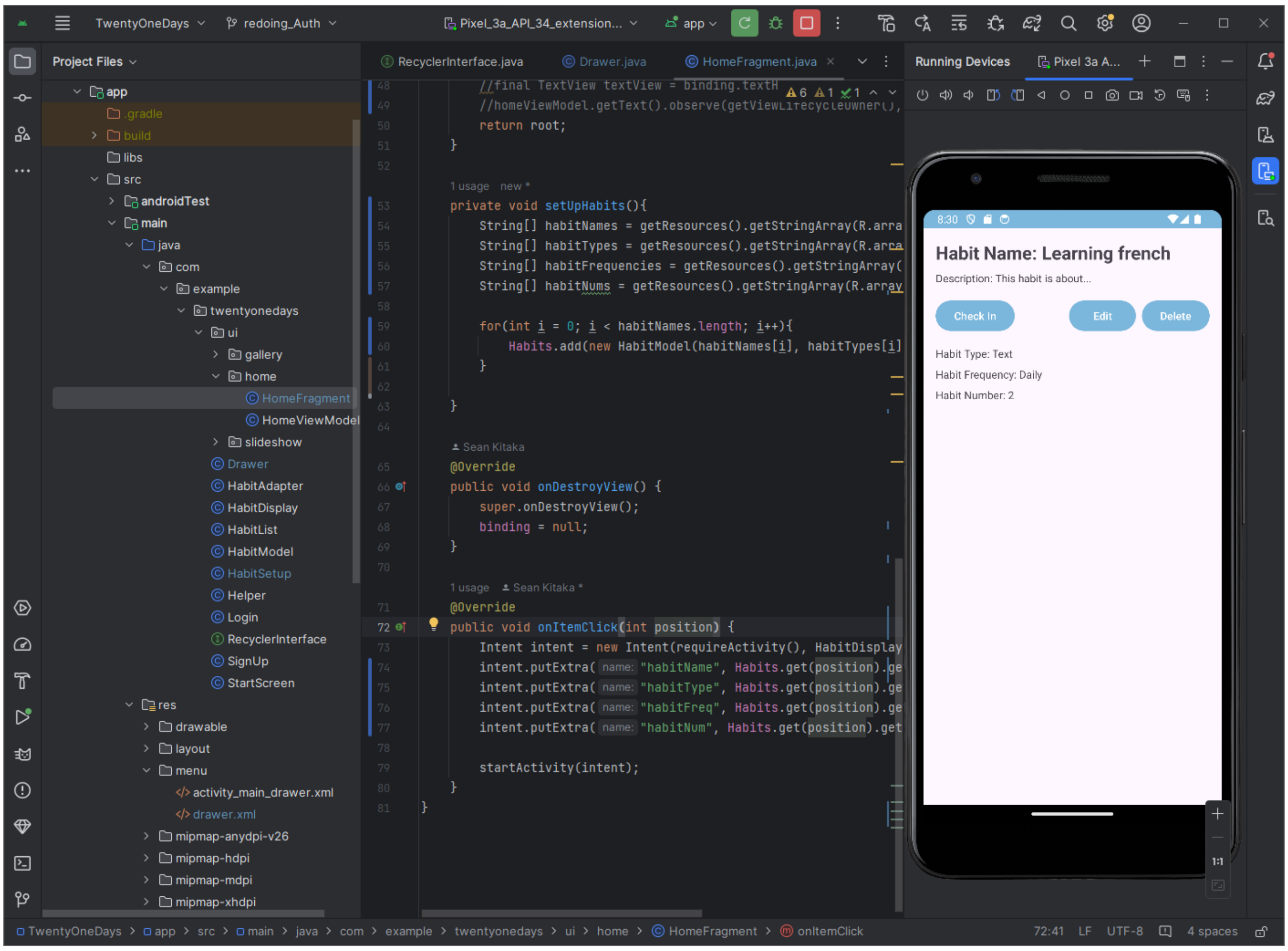
The following classes contribute to the implementation of habit interactivity:

1. **HabitDisplay Class**: This class displays detailed information about a specific habit, including its name, type, frequency, and other relevant details. Users can access this detailed view by clicking on habit items within the RecyclerView.
2. **HabitSetup Class**: This class allows users to add new habits to the database by providing necessary details such as habit name, description, type, frequency, and optional media attachments. Upon submission, the habit data is saved to the Firebase database, enabling seamless integration with the application's core functionality.



```java
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.ViewModelProvider;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.example.twentyonedays.HabitAdapter;
import com.example.twentyonedays.HabitDisplay;
import com.example.twentyonedays.HabitList;
import com.example.twentyonedays.HabitModel;
import com.example.twentyonedays.R;
import com.example.twentyonedays.RecyclerInterface;
import com.example.twentyonedays.databinding.FragmentHomeBinding;
import com.example.twentyonedays.ui.home.HomeViewModel;

import java.util.ArrayList;

public class HomeFragment extends Fragment implements RecyclerInterface {

    private FragmentHomeBinding binding;
    private HabitAdapter adapter;
    ArrayList<HabitModel> Habits = new ArrayList<>();


    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState) {
        HomeViewModel homeViewModel =
                new ViewModelProvider( owner: this).get(HomeViewModel.class);

        binding = FragmentHomeBinding.inflate(inflater, container,  attachToParent: false);
        View root = binding.getRoot();
```

Figure 5 – Home page displaying Habits in a recycler view



```java
        //final TextView textView = binding.textH
        //homeViewModel.getText().observe(getViewLifecycleOwner(),
        return root;
    }

    private void setUpHabits(){
        String[] habitNames = getResources().getStringArray(R.arra
        String[] habitTypes = getResources().getStringArray(R.arra
        String[] habitFrequencies = getResources().getStringArray(
        String[] habitNums = getResources().getStringArray(R.array

        for(int i = 0; i < habitNames.length; i++){
            Habits.add(new HabitModel(habitNames[i], habitTypes[i]
        }

    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        binding = null;
    }

    @Override
    public void onItemClick(int position) {
        Intent intent = new Intent(requireActivity(), HabitDisplay
        intent.putExtra( name: "habitName", Habits.get(position).ge
        intent.putExtra( name: "habitType", Habits.get(position).ge
        intent.putExtra( name: "habitFreq", Habits.get(position).ge
        intent.putExtra( name: "habitNum", Habits.get(position).get

        startActivity(intent);
    }
}
```

Figure 6 – more detailed view of one of the habits

4. Conclusions and Future work

a. Concluding remarks (Summary of the TwentyOneDays outcomes).

TwentyOneDays sets out to tackle procrastination by empowering users to form and sustain new habits. Achieving its objectives of being user-friendly, adaptable, and motivational, the app has made significant strides in enhancing user engagement and facilitating personal growth.

b. Reflection on overall learning experience and achievements.

The journey of developing TwentyOneDays has been profoundly educational, offering insights into the complexities of app development, user experience design, and the psychology of habit formation. This project in particular also provided a reminder to me about not wandering too far from the scope of the project, which is a lesson that I will be sure to apply in future tasks. The project's challenges have fostered a spirit of creativity and resilience, culminating in the successful realization of the app's core features and the acquisition of positive initial feedback.

c. Future Work

The roadmap for TwentyOneDays includes exciting opportunities for expansion and enhancement:

Enhanced Social Features: Building on the app's social aspects to encourage community engagement and mutual support among users.

Gamification Strategies: Incorporating elements of gamification to reward progress and celebrate milestones, boosting user motivation.

Personalized Insights: Integrating AI to provide customized advice and insights, making the habit formation process even more tailored and effective.

Platform Diversification: Expanding the app's availability to web and desktop platforms, increasing its accessibility to a wider audience.

Wellness Integration: Adding features focused on mental and emotional well-being, such as stress management and mindfulness practices, to support users' overall health.

In conclusion, while TwentyOneDays has successfully met its initial goals, its journey towards facilitating meaningful habit change and personal development is just beginning. Future enhancements, guided by user feedback and technological advancements, will ensure that TwentyOneDays continues to be a pivotal tool for individuals seeking to make lasting changes in their lives.