House Keeper

Technical Analysis

Retrieving and Relaying Data

House Keeper allows the communication of and transferring of data and information accordingly, the database that has been used is the Firebase Realtime Database as it allows users to directly add information realtime, and any changes that occurred is also translated realtime without the need to rebuild or reinstall the application.

Initial Demonstration.

User

The initial landing page activity of House Keeper is configured in the Android Manifest file. As we are greeted in the landing page, the user can now be headed towards either the Login activity if the user has an existing account or the Sign Up activity if the user chooses to create his own account.

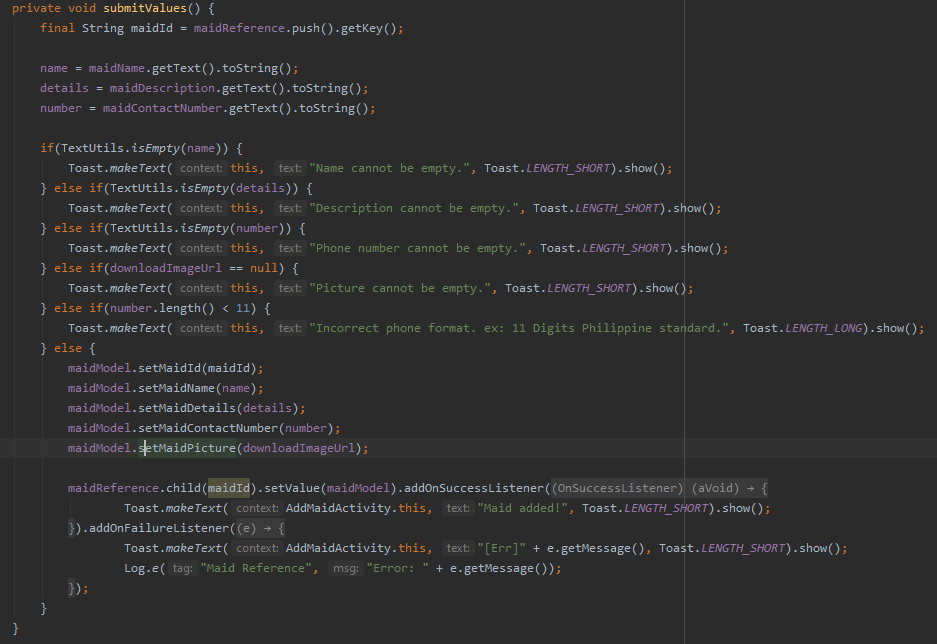
Upon registering within the platform, House Keeper uses Firebase User and Firebase Auth, both for creating the account details of a specific user. One of which is using information such as UID, Email, and Password. Another mechanism within the business model is to inherently create a carbon copy of the user details and send it to the database for further modification, updating or deleting.

The business logic of House Keepers allows any user that registered to their platform a feature on which they could post their own services, while other users can view the previewed services of other maids.

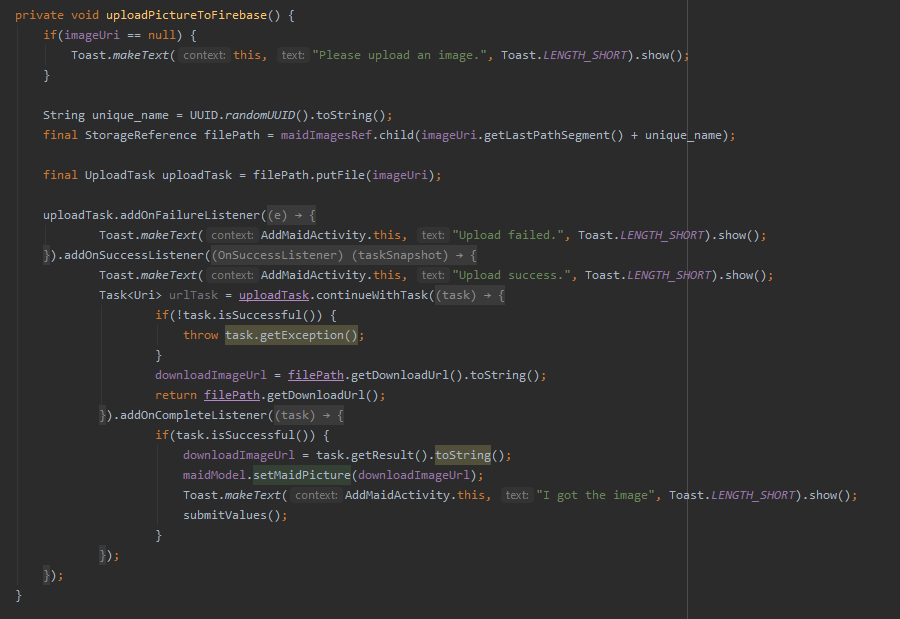
Admin

The administrator account has their own admin panel in due to modifying and deleted maid entries. This is an additional safeguard that is implemented.

Creation of Maids:

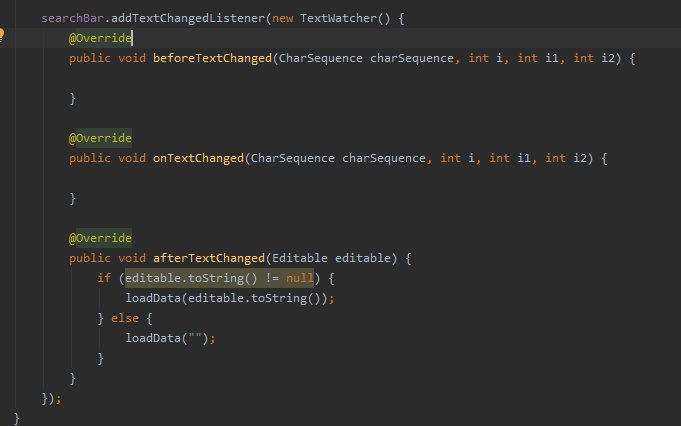


Before the Maids are created and recorded into the database they first have to be initialized with setters and getters of a class model, then they can be populated with data coming from the user. There are several checks and balances that were made in order to sanitize the user’s input.

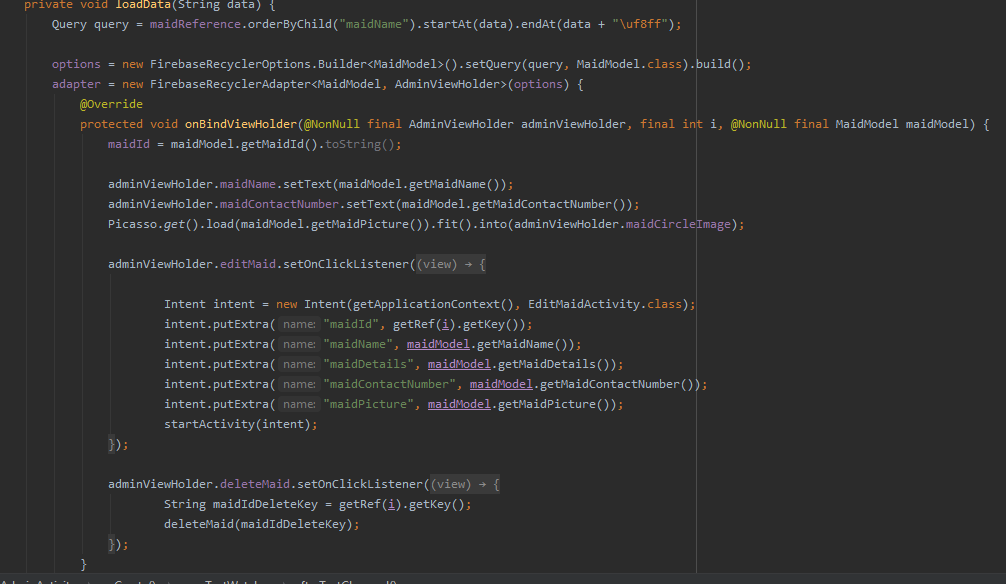


After doing so, an image opened from the Gallery can be added into the database, and a prompt to say if the transfer was completed or failed.

Search feature.  
The search feature’s code is used both for users and admin. They can search the maid’s name in the query.



Loading from recycler view and adapter.

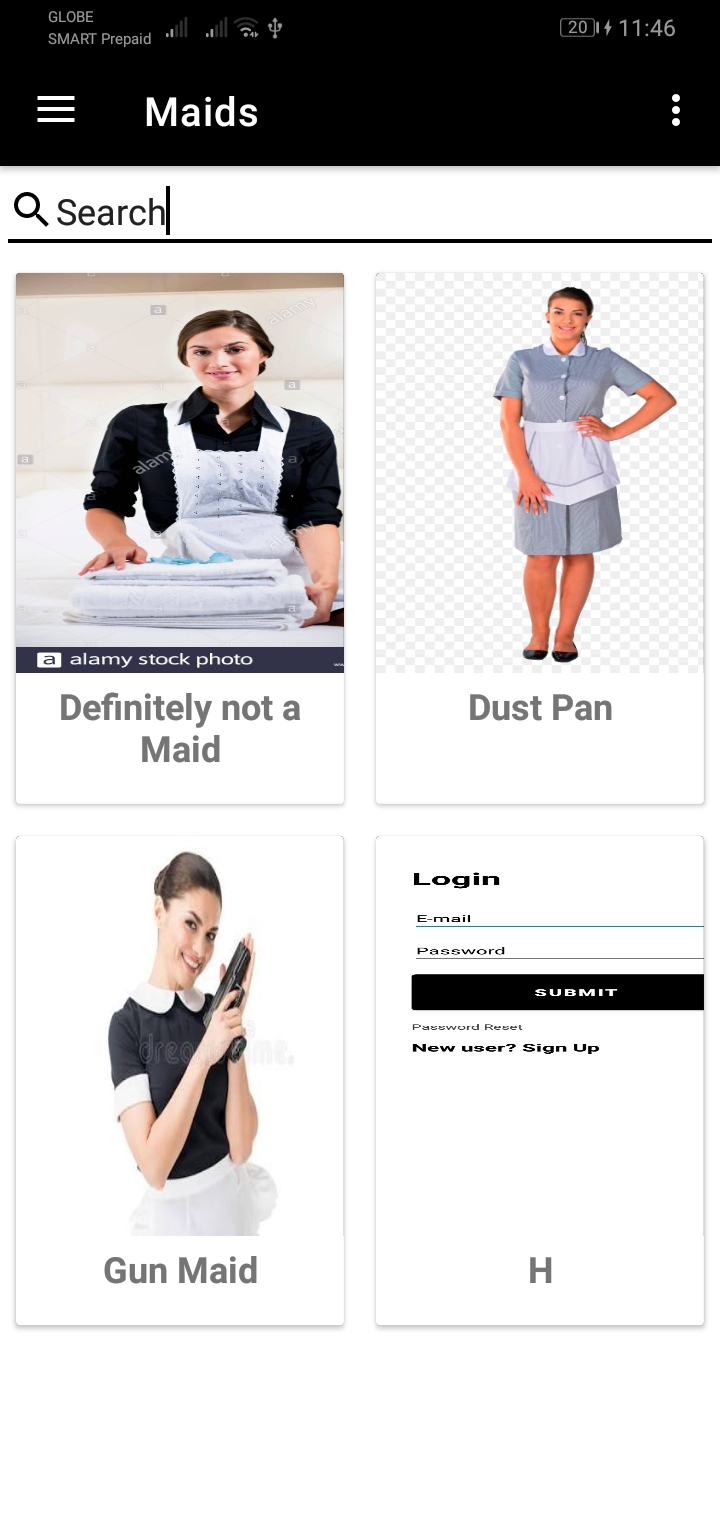


Upon visiting the admin page, they are greeted with a list of existing maids and the option to modify or delete a maid’s entry. Since they are public this is relative to what the maid views with the exception of deleting and updating specific key values.

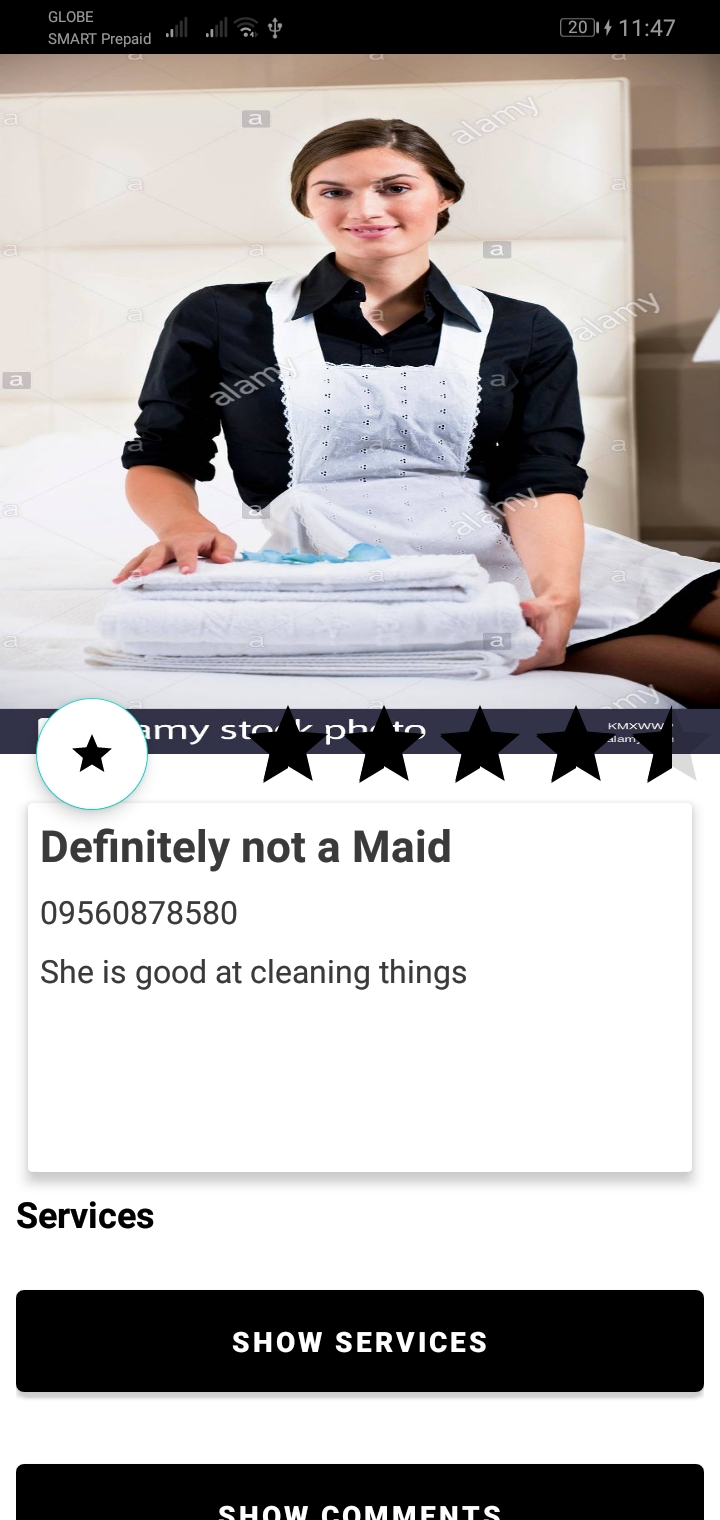
The data that was retrieved from the Firebase Realtime database can be passed onto Editing a Maid Entry, this seemed to be a better option as it prevents in creating more instances of the database, rather to use the initial instance and then pass these data using intents.



A fragment class within the Android Studio source files, this is the initial landing page for non-admin users who will be viewing the platform. The images are retrieved from different sources online, as a category for different services that a non-admin user can avail to.



The MaidFragment.class list also acts as a fragment, using Recyclerview to list existing maids that are called from a Cardview layout. The MaidAdapter.class populates existing data from the database into the user-interface.

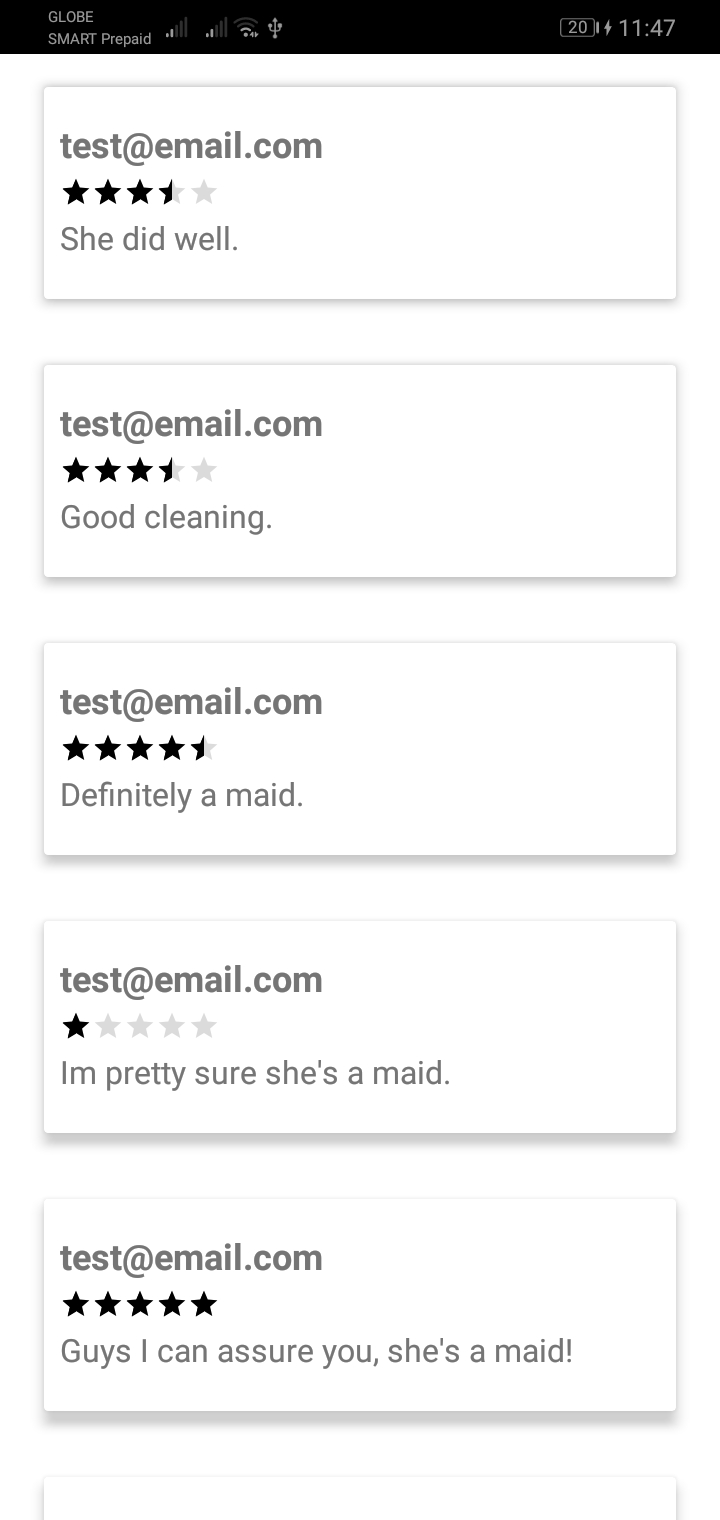


Upon clicking a specific maid entry from the MaidFragment, the user is passed onto an activity called MaidDetailedActivity.class, the Firebase key is called from the Maid’s position in the recyclerview. Along with other data that were passed using .putExtra() method and retrieved using getExtra() method. The user can avail to the maid’s services via contacting the existing information present. Such as the maid’s name, the maid’s contact number, and the user can also see the details of which.

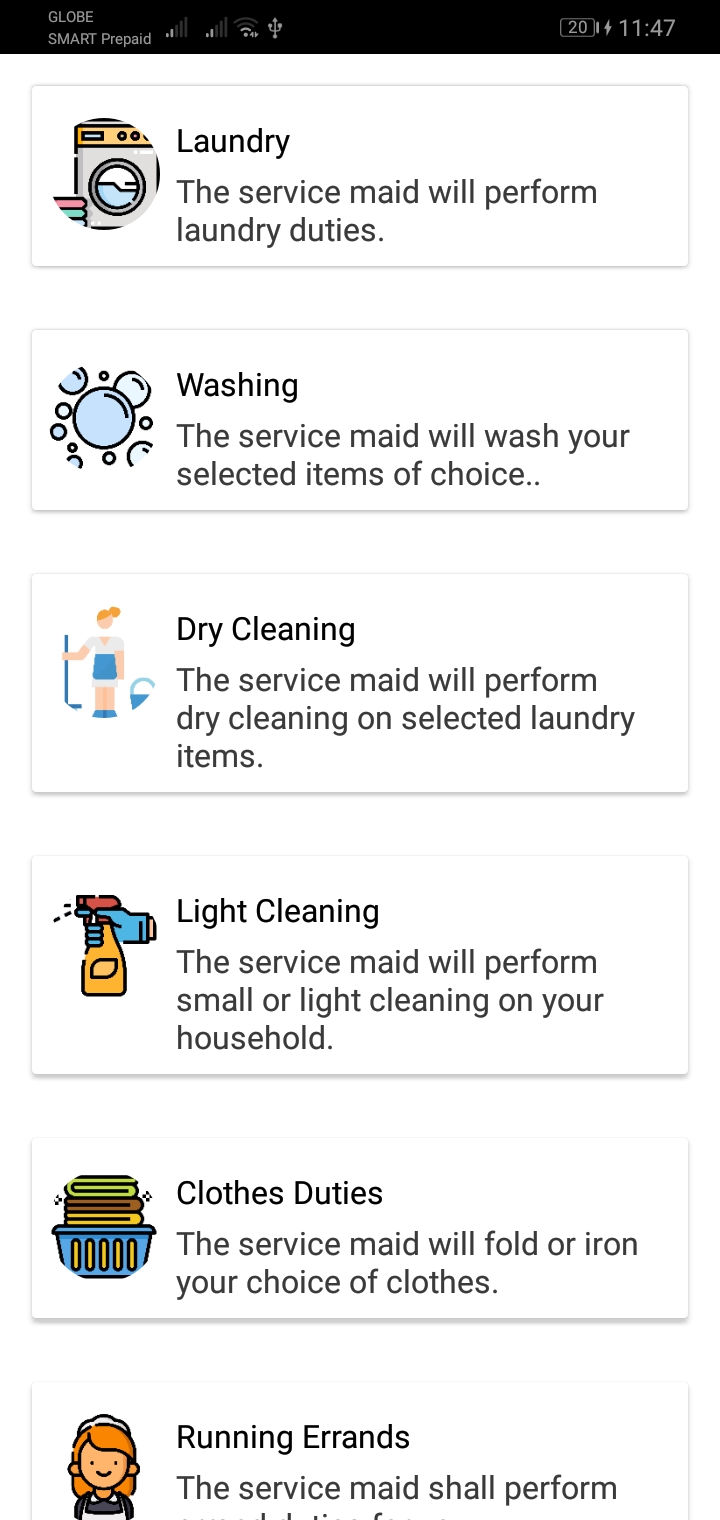
The User can also view what the maid’s services is, this is also another form of using recyclerview with a viewholder attached to it. Which is quite similar to what is used on the MaidFragment.

User can also submit their own rating to the maid.

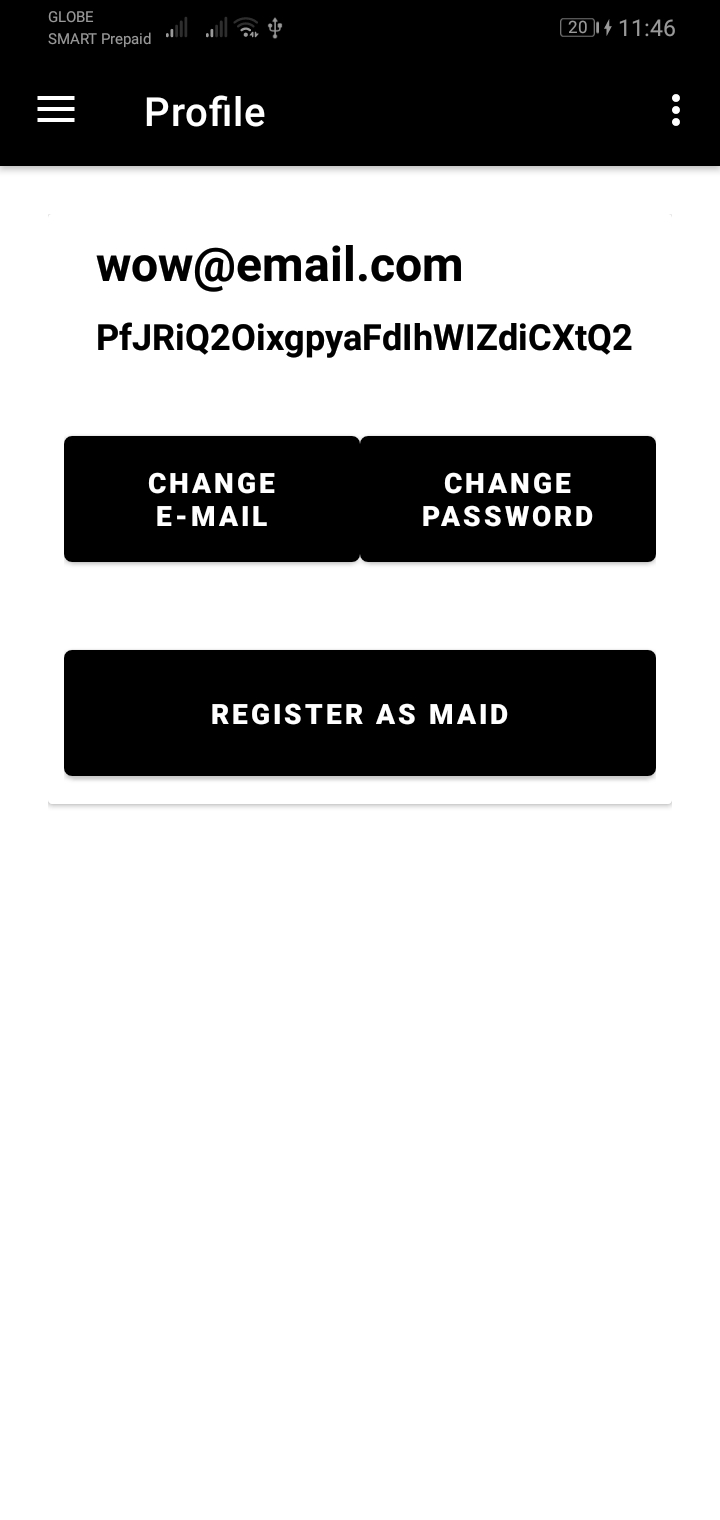
Similar to what I have stated above, user can also view existing comments that are submitted by other users.



This is another activity that shows the number of stars and comments, as well as the user who rated with the specific entry. This is also using a recycler view attached with a ViewHolder. Similar to what is used in the services.

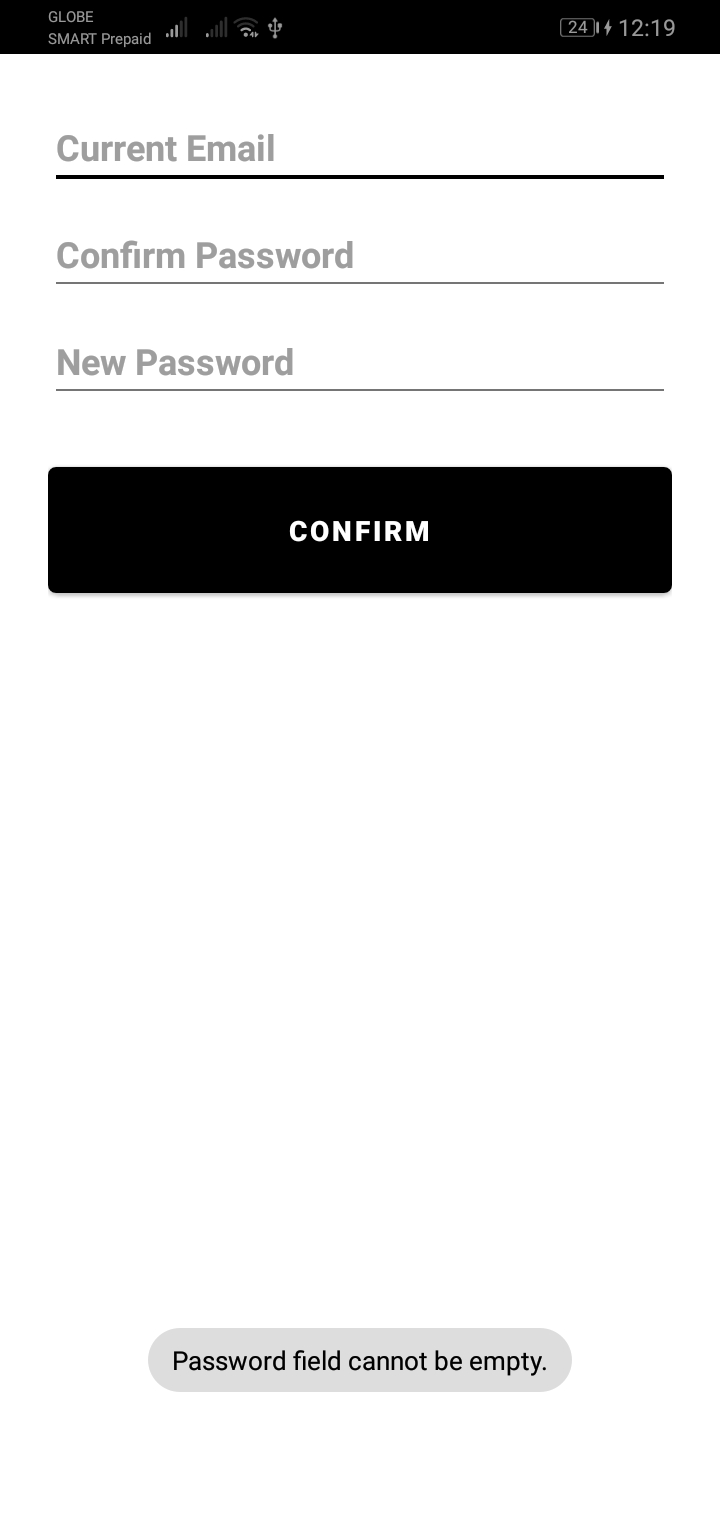


Upon clicking in the MaidDetailedActivity the Show Services button, the user can see that the maid’s services are and his/her capabilities with description. Similar pattern has been used to retrieve data which is unique to a maid using the Maid’s primary key. Again the implementation is similar to what I have stated before. Using a Recycler view attached with a ViewHolder to communicate with the UI. The ShowServicesActivity.class calls on instances from the Firebase Database to show these data in real time.



This is the user’s profile. Information such as the email address and the user’s ID is displayed. The user can also have the ability to change email and password in their own choosing.

Upon clicking on Change E-Mail button or Change Password. We have used intents on both fragment to - activity, with some information that are passed along using putExtra() and getExtra() methods.



ChangePasswordActivity.class

The class file consists of asking for user input, and then it would later confirm if the existing credentials are valid.

The system would then change the user’s password and would later re authenticate them in order to successfully change the users’ password. The reauthentication is mandatory per Firebase Auth and Firebase User’s standards.

private void reAuthenticateUser(final String email, String password) {

final String TAG = "reAuthenticateUser";

FirebaseUser user = FirebaseAuth.*getInstance*().getCurrentUser();

AuthCredential credential = EmailAuthProvider

.*getCredential*(email, password);

Log.*d*(TAG, "Email: " + email.toString().trim());

Log.*d*(TAG, "Password: " + password.toString().trim());

user.reauthenticate(credential)

.addOnCompleteListener(new OnCompleteListener<Void>() {

@Override

public void onComplete(@NonNull Task<Void> task) {

Log.*d*(TAG, "User re-authenticated.");

FirebaseUser user = FirebaseAuth.*getInstance*().getCurrentUser();

user.updateEmail(newEmailString)

.addOnCompleteListener(new OnCompleteListener<Void>() {

@Override

public void onComplete(@NonNull Task<Void> task) {

if (task.isSuccessful()) {

Log.*d*(TAG, "User email address updated.");

Toast.*makeText*(ChangeEmailActivity.this, "User email address updated.", Toast.*LENGTH\_SHORT*).show();

Log.*d*(TAG, "New Email: " + newEmailString);

updateDatabase(newEmailString);

} else {

Toast.*makeText*(ChangeEmailActivity.this, "Wrong password.", Toast.*LENGTH\_SHORT*).show();

}

}

});

}

});

}

private void updateDatabase(String email) {

final String TAG = "updateDatabase";

currentId = mUser.getUid();

userRef.child(currentId).child("email").setValue(email)

.addOnSuccessListener(new OnSuccessListener<Void>() {

@Override

public void onSuccess(Void aVoid) {

Log.*d*(TAG, "Database changed, success");

}

})

.addOnFailureListener(new OnFailureListener() {

@Override

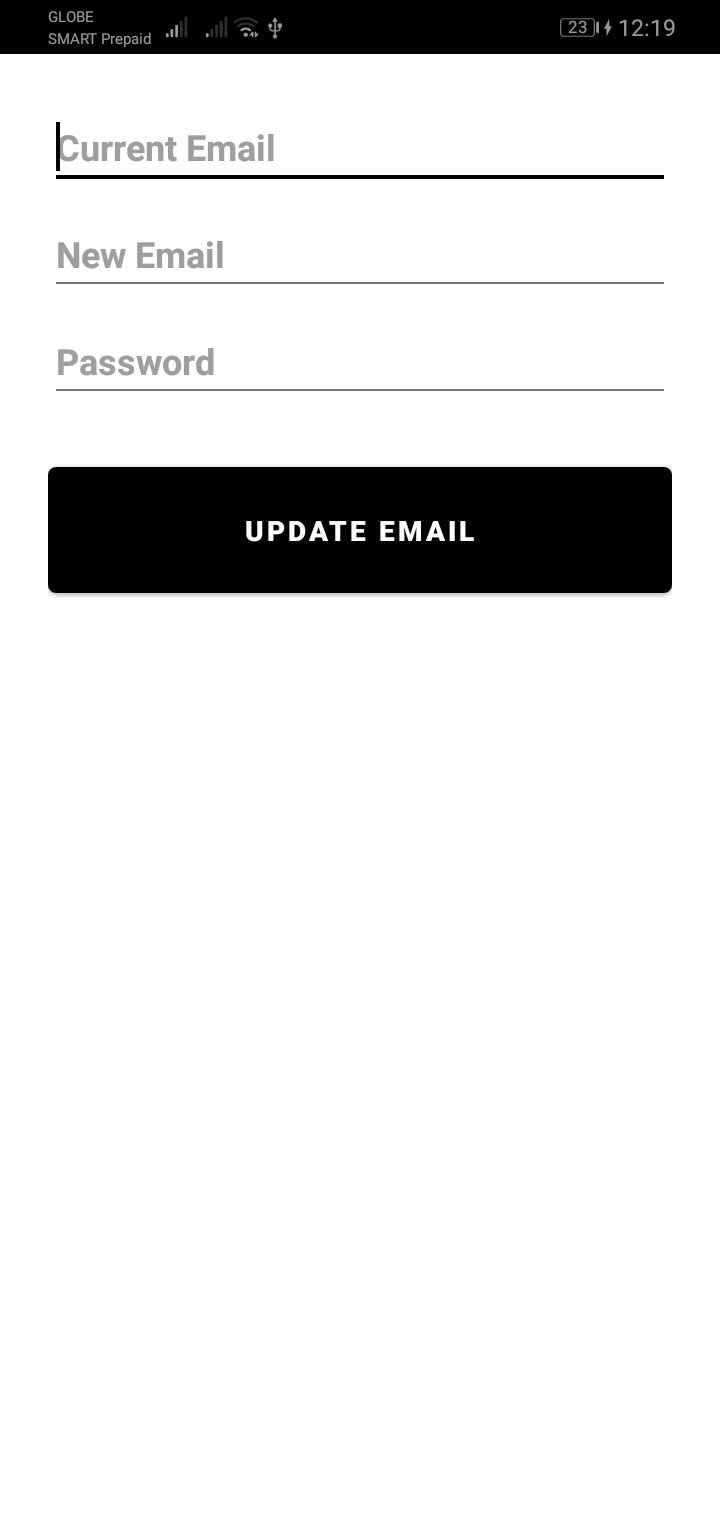
public void onFailure(@NonNull Exception e) {

Log.*d*(TAG, "Error: " + e.getMessage());

}

});

}



UpdateEmailActivity.class

Similar to the preposition of changing the user’s password.

The system checks if the current credentials are existing, attempts to validate it to the Firebase User and Auth if existing. It would then reauthenticate successfully in order to change the user details.

private void validateInput() {

String currentEmailToString = currentEmail.getText().toString();

String currentPasswordToString = currentPassword.getText().toString();

newPasswordToString = newChangePassword.getText().toString();

if(TextUtils.*isEmpty*(currentEmailToString)) {

Toast.*makeText*(this, "Email field cannot be empty.", Toast.*LENGTH\_SHORT*).show();

}

if(TextUtils.*isEmpty*(currentPasswordToString)) {

Toast.*makeText*(this, "Password field cannot be empty.", Toast.*LENGTH\_SHORT*).show();

}

if(mUser != null) {

processChangePassword(currentEmailToString, currentPasswordToString);

}

}

private void processChangePassword(String email, String password) {

final String TAG = "processChangePassword";

mUser = FirebaseAuth.*getInstance*().getCurrentUser();

AuthCredential credential = EmailAuthProvider

.*getCredential*(email, password);

mUser.reauthenticate(credential)

.addOnCompleteListener(new OnCompleteListener<Void>() {

@Override

public void onComplete(@NonNull Task<Void> task) {

if (task.isSuccessful()) {

mUser.updatePassword(newPasswordToString)

.addOnCompleteListener(new OnCompleteListener<Void>() {

@Override

public void onComplete(@NonNull Task<Void> task) {

if (task.isSuccessful()) {

Log.*d*(TAG, "Password updated");

updateDatabase(newPasswordToString);

} else {

Log.*d*(TAG, "Error password not updated");

Toast.*makeText*(ChangePasswordActivity.this, "Wrong account details.", Toast.*LENGTH\_SHORT*).show();

}

}

});

} else {

Log.*d*(TAG, "Error auth failed");

Toast.*makeText*(ChangePasswordActivity.this, "Incorrect Account Details.", Toast.*LENGTH\_SHORT*).show();

}

}

});

}

private void updateDatabase(String password) {

final String TAG = "updateDatabase";

currentId = mUser.getUid();

passwordRef.child(currentId).child("password").setValue(password)

.addOnSuccessListener(new OnSuccessListener<Void>() {

@Override

public void onSuccess(Void aVoid) {

Log.*d*(TAG, "Database changed, success");

}

})

.addOnFailureListener(new OnFailureListener() {

@Override

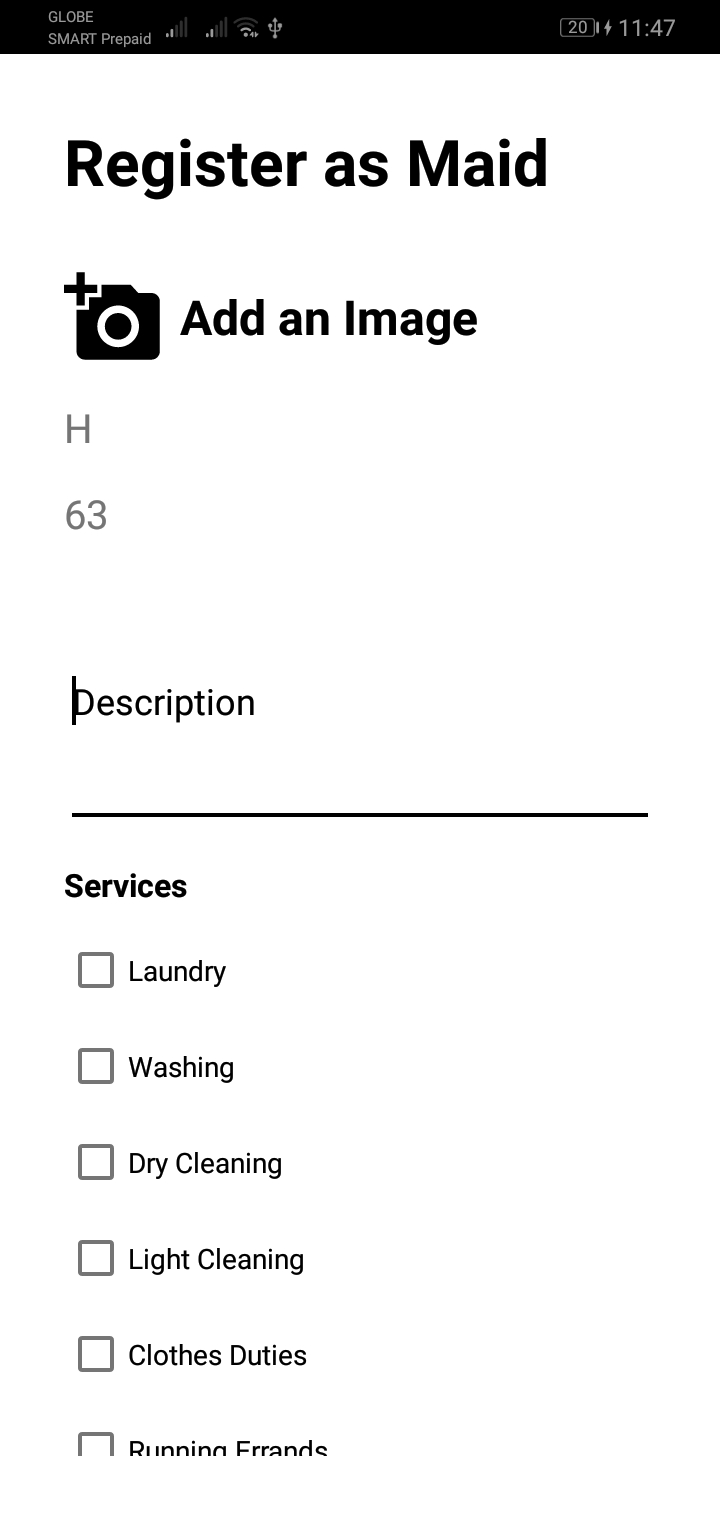
public void onFailure(@NonNull Exception e) {

Log.*d*(TAG, "Error: " + e.getMessage());

}

});

}



CreateUserMaidActivity.class

The feature allows users to also avail their services as maid if they would like to.

This method submits the existing values onto the database, a message prompt will soon appear if the data has successfully been accepted.

private void submitValues() {

String name = maidName.getText().toString();

details = maidDescription.getText().toString();

number = maidContactNumber.getText().toString();

if(TextUtils.*isEmpty*(details)) {

Toast.*makeText*(this, "Description cannot be empty.", Toast.*LENGTH\_SHORT*).show();

} else if(TextUtils.*isEmpty*(number)) {

Toast.*makeText*(this, "Phone number cannot be empty.", Toast.*LENGTH\_SHORT*).show();

} else if(downloadImageUrl == null) {

Toast.*makeText*(this, "Picture cannot be empty.", Toast.*LENGTH\_SHORT*).show();

} else {

maidModel.setMaidId(currentUserId);

maidModel.setMaidName(name);

maidModel.setMaidDetails(details);

maidModel.setMaidContactNumber(number);

maidModel.setMaidPicture(downloadImageUrl);

maidReference.child(currentUserId).setValue(maidModel).addOnSuccessListener(new OnSuccessListener<Void>() {

@Override

public void onSuccess(Void aVoid) {

Toast.*makeText*(CreateUserMaidActivity.this, "Maid added!", Toast.*LENGTH\_SHORT*).show();

addServicesToMaid();

}

}).addOnFailureListener(new OnFailureListener() {

@Override

public void onFailure(@NonNull Exception e) {

Toast.*makeText*(CreateUserMaidActivity.this, "[Err]" + e.getMessage(), Toast.*LENGTH\_SHORT*).show();

Log.*e*("Maid Reference", "Error: " + e.getMessage());

}

});

}

}

This attempts to upload an image of the maid to the Firebase Storage.

private void uploadPictureToFirebase() {

if(imageUri == null) {

Toast.*makeText*(this, "Please upload an image.", Toast.*LENGTH\_SHORT*).show();

}

String unique\_name = UUID.*randomUUID*().toString();

try {

final StorageReference filePath = maidImagesRef.child(imageUri.getLastPathSegment() + unique\_name);

final UploadTask uploadTask = filePath.putFile(imageUri);

uploadTask.addOnFailureListener(new OnFailureListener() {

@Override

public void onFailure(@NonNull Exception e) {

Toast.*makeText*(CreateUserMaidActivity.this, "Upload failed.", Toast.*LENGTH\_SHORT*).show();

}

}).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {

@Override

public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {

Toast.*makeText*(CreateUserMaidActivity.this, "Upload success.", Toast.*LENGTH\_SHORT*).show();

Task<Uri> urlTask = uploadTask.continueWithTask(new Continuation<UploadTask.TaskSnapshot, Task<Uri>>() {

@Override

public Task<Uri> then(@NonNull Task<UploadTask.TaskSnapshot> task) throws Exception {

if(!task.isSuccessful()) {

throw task.getException();

}

downloadImageUrl = filePath.getDownloadUrl().toString();

return filePath.getDownloadUrl();

}

}).addOnCompleteListener(new OnCompleteListener<Uri>() {

@Override

public void onComplete(@NonNull Task<Uri> task) {

if(task.isSuccessful()) {

downloadImageUrl = task.getResult().toString();

maidModel.setMaidPicture(downloadImageUrl);

Toast.*makeText*(CreateUserMaidActivity.this, "I got the image", Toast.*LENGTH\_SHORT*).show();

submitValues();

}

}

});

}

});

} catch (NullPointerException e) {

Toast.*makeText*(this, "Please add an image.", Toast.*LENGTH\_SHORT*).show();

}

}

private void openGallery() {

Intent galleryIntent = new Intent();

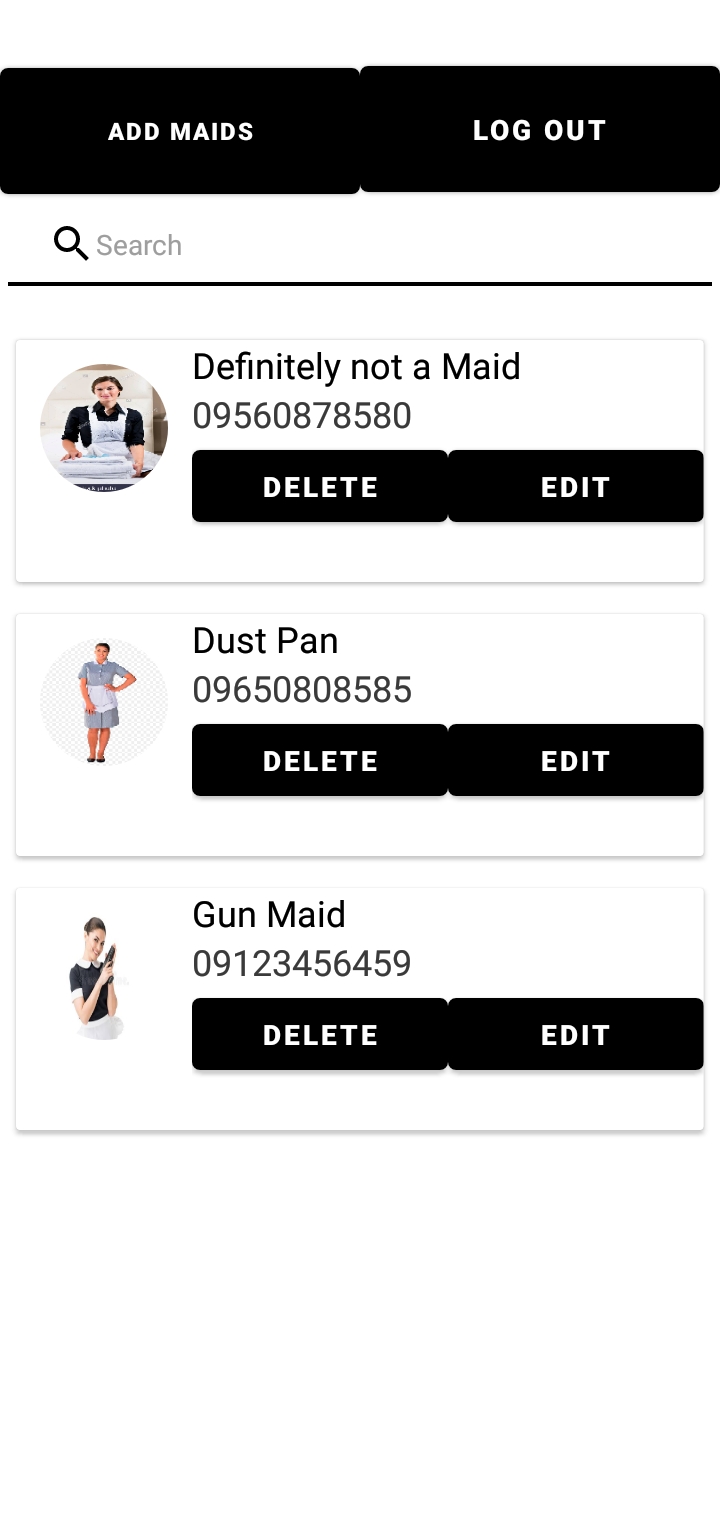
galleryIntent.setType("image/\*");

galleryIntent.setAction(Intent.*ACTION\_GET\_CONTENT*);

startActivityForResult(Intent.*createChooser*(galleryIntent, "Select picture"), *PICK\_IMAGE\_REQUEST*);

}

Admin Views



The AdminActivity.class is using recycler view to preview existing maids that are referenced from the Maid database.

Search feature.

In using the addTextChanged listener, this is referenced from querying what the Firebase has loaded. If the text is matching to what is currently queried, the maid entry should appear.

searchBar.addTextChangedListener(new TextWatcher() {

@Override

public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {

}

@Override

public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {

}

@Override

public void afterTextChanged(Editable editable) {

if (editable.toString() != null) {

loadData(editable.toString());

} else {

loadData("");

}

}

});

This is the query that should load the maid entries, the “data” on the other hand retrieves values from what is queried and relays that to the search function.

Query query = maidReference.orderByChild("maidName").startAt(data).endAt(data + "\uf8ff");

options = new FirebaseRecyclerOptions.Builder<MaidModel>().setQuery(query, MaidModel.class).build();

adapter = new FirebaseRecyclerAdapter<MaidModel, AdminViewHolder>(options) {

@Override

protected void onBindViewHolder(@NonNull final AdminViewHolder adminViewHolder, final int i, @NonNull final MaidModel maidModel) {

maidId = maidModel.getMaidId().toString();

adminViewHolder.maidName.setText(maidModel.getMaidName());

adminViewHolder.maidContactNumber.setText(maidModel.getMaidContactNumber());

Picasso.*get*().load(maidModel.getMaidPicture()).fit().into(adminViewHolder.maidCircleImage);

adminViewHolder.editMaid.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

Intent intent = new Intent(getApplicationContext(), EditMaidActivity.class);

intent.putExtra("maidId", getRef(i).getKey());

intent.putExtra("maidName", maidModel.getMaidName());

intent.putExtra("maidDetails", maidModel.getMaidDetails());

intent.putExtra("maidContactNumber", maidModel.getMaidContactNumber());

intent.putExtra("maidPicture", maidModel.getMaidPicture());

startActivity(intent);

}

});

adminViewHolder.deleteMaid.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

String maidIdDeleteKey = getRef(i).getKey();

deleteMaid(maidIdDeleteKey);

}

});

}

@NonNull

@Override

public AdminViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

View view = LayoutInflater.*from*(parent.getContext()).inflate(R.layout.*maid\_card\_view\_admin*, parent, false);

return new AdminViewHolder(view);

}

};

adapter.startListening();

recyclerView.setAdapter(adapter);

The delete maid feature deletes the maid’s entry from the key’s position in the recycler view.

private void deleteMaid(final String maidIdDeleteKey) {

AlertDialog.Builder builder = new AlertDialog.Builder(this);

builder.setMessage("Delete this Maid entry?");

builder.setPositiveButton("Delete", new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialogInterface, int i) {

maidReference.child(maidIdDeleteKey).removeValue().addOnSuccessListener(new OnSuccessListener<Void>() {

@Override

public void onSuccess(Void aVoid) {

Toast.*makeText*(AdminActivity.this, "Entry has been deleted.", Toast.*LENGTH\_SHORT*).show();

}

}).addOnFailureListener(new OnFailureListener() {

@Override

public void onFailure(@NonNull Exception e) {

Toast.*makeText*(AdminActivity.this, "Delete failed.", Toast.*LENGTH\_SHORT*).show();

}

});

}

}).setNegativeButton("Cancel", new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialogInterface, int i) {

dialogInterface.dismiss();

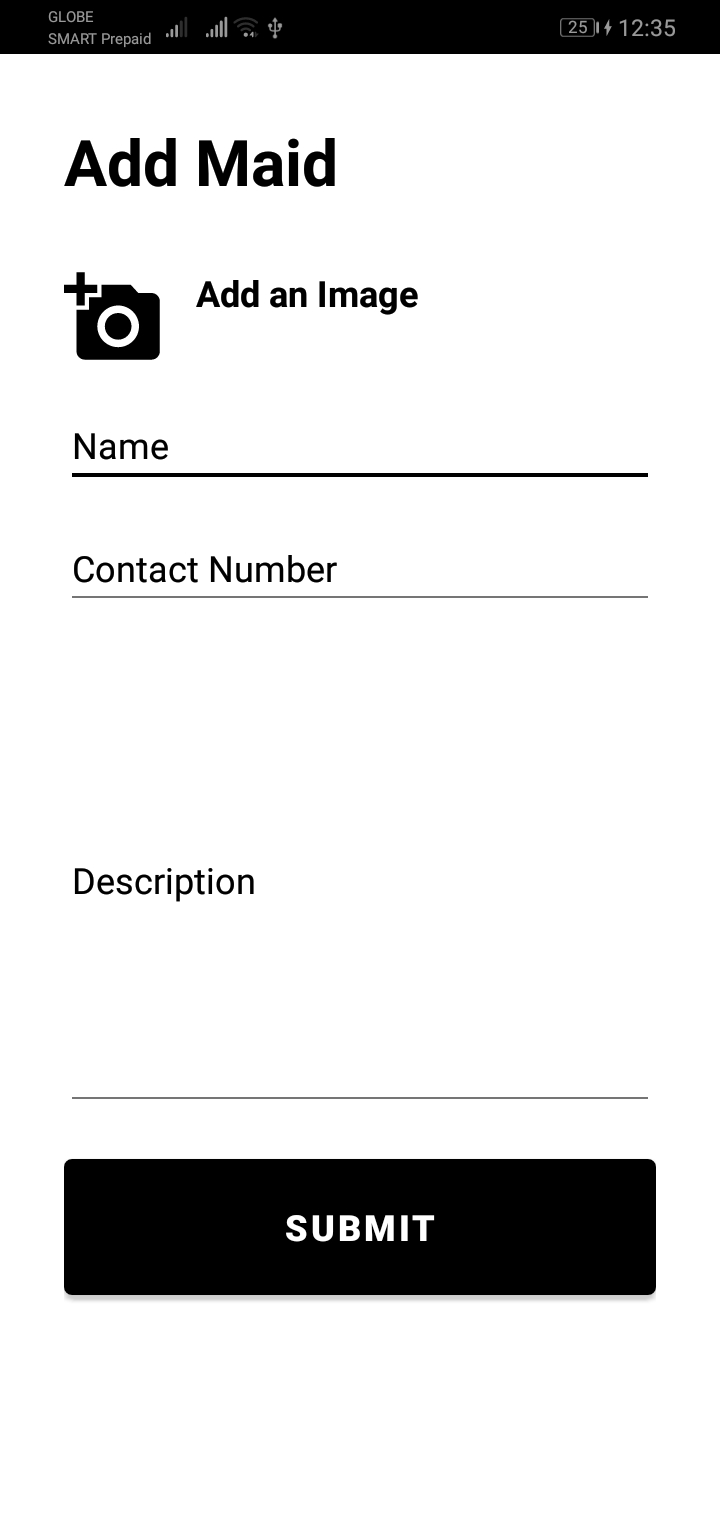
}

});

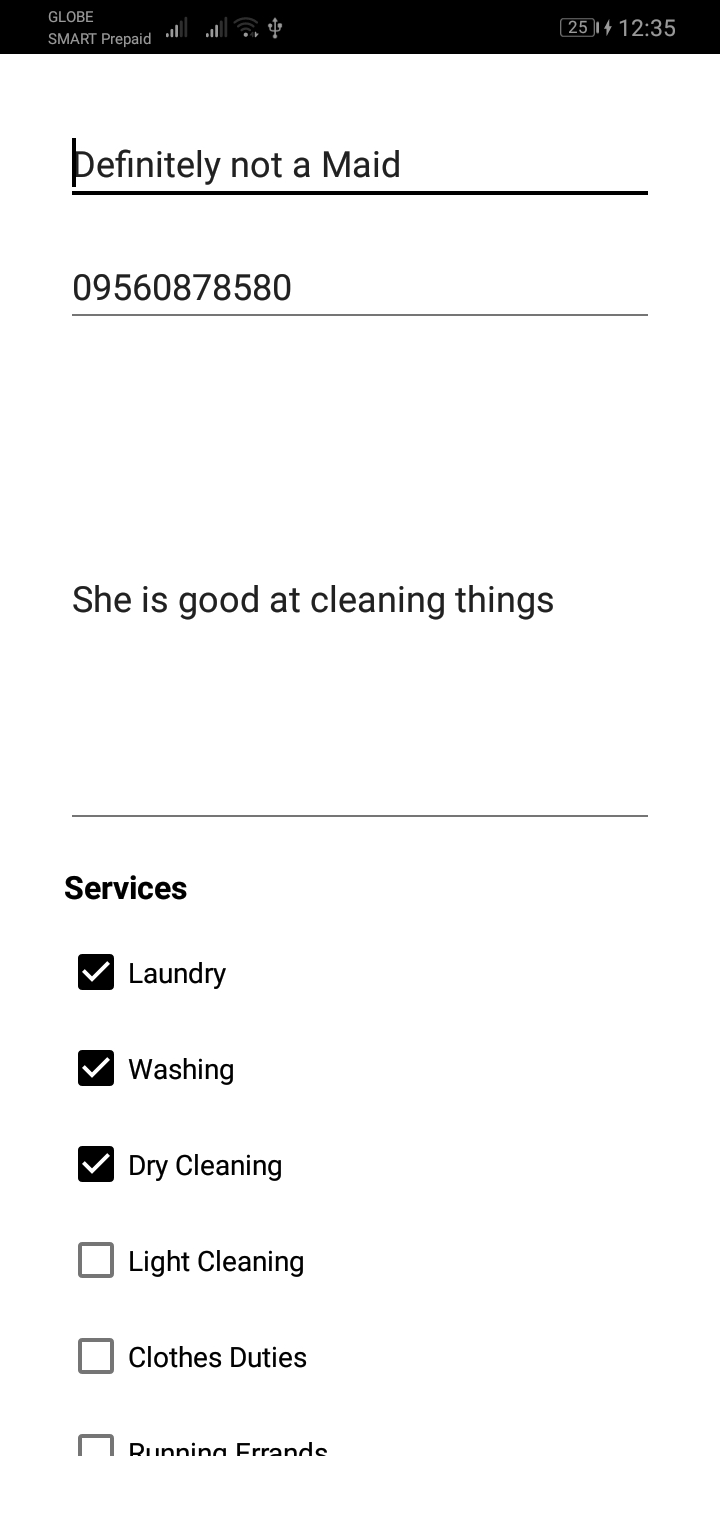
builder.create();

builder.show();

}



The AddMaidActivity.class code is similar to the UserAddMaidActivity.class, the only difference is that the admin can add more maids compared to what the User can only edit their own entry that’s referencing from their user key.



The EditMaidActivity.class allows the Admin to edit a specific maid’s entry from referencing a key in a recyclerview’s position.

loadData() method from AdminActivity.class

adminViewHolder.editMaid.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

Intent intent = new Intent(getApplicationContext(), EditMaidActivity.class);

**intent.putExtra("maidId", getRef(i).getKey());**

intent.putExtra("maidName", maidModel.getMaidName());

intent.putExtra("maidDetails", maidModel.getMaidDetails());

intent.putExtra("maidContactNumber", maidModel.getMaidContactNumber());

intent.putExtra("maidPicture", maidModel.getMaidPicture());

startActivity(intent);

}

});

The bold text shows getRef(i) meaning we attempt to get the reference of variable i. I = is the key of the position in the database that we are attempting to retrieve.