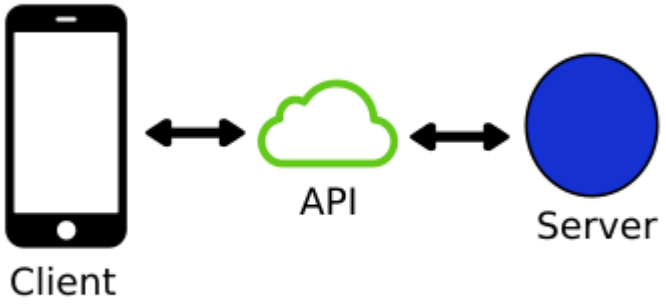
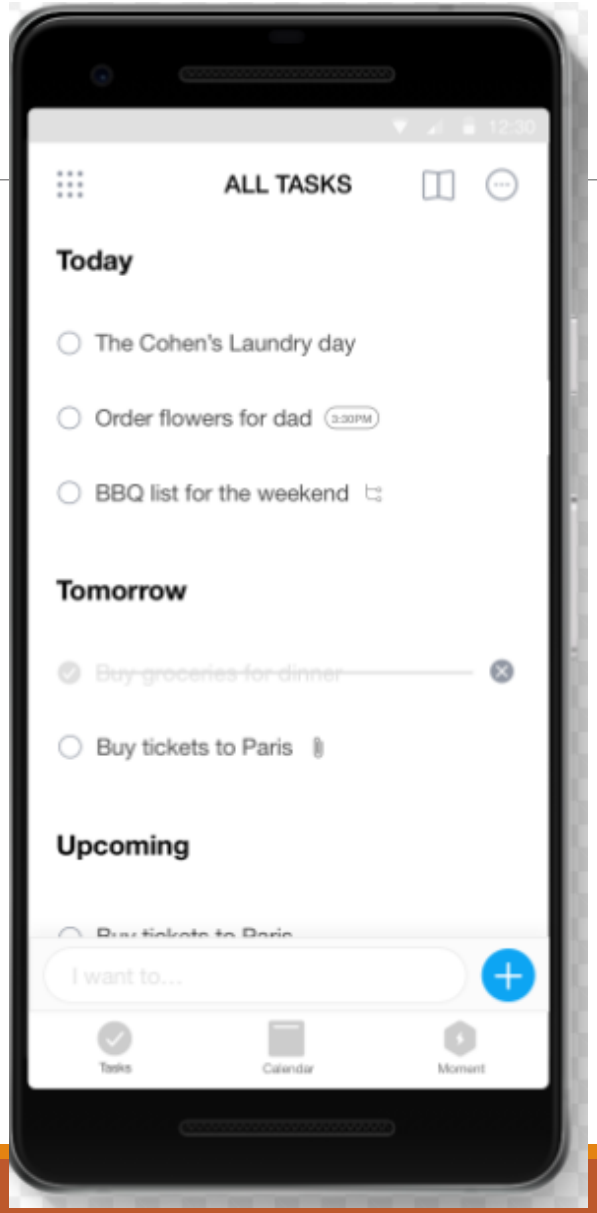
Web service trong android



Retrofit 2

{REST}
Restful Webservices

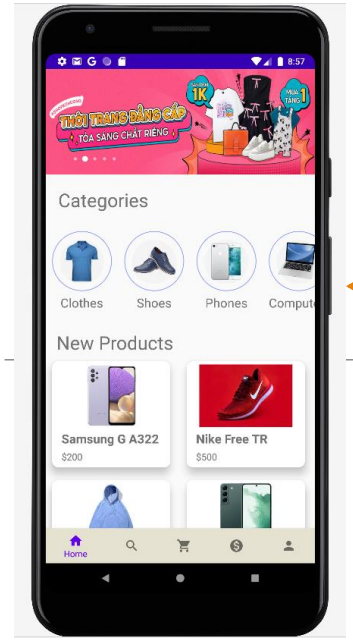
I. Tổng quan về web service trong android



Luồng cơ bản của 1 ứng dụng android lưu trữ local

SQLiteHelper,
SQLite Database

List<Task> getListTasks()
void updateTask()
void addTask()
...



Mobile

android, ios, flutter, react native

json, xml,...

API server

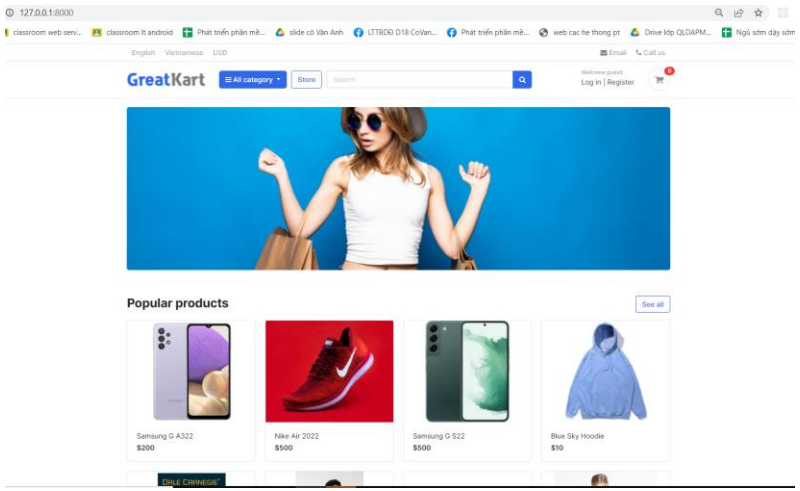
Nodejs, PHP, Java, ASP.NET, Django

json, xml,...

Another API Server

Database

MySQL, PostgreSQL, Oracle, MongoDB



Browser

html,css ,js

Web server

Reactjs, Vuejs, Angular

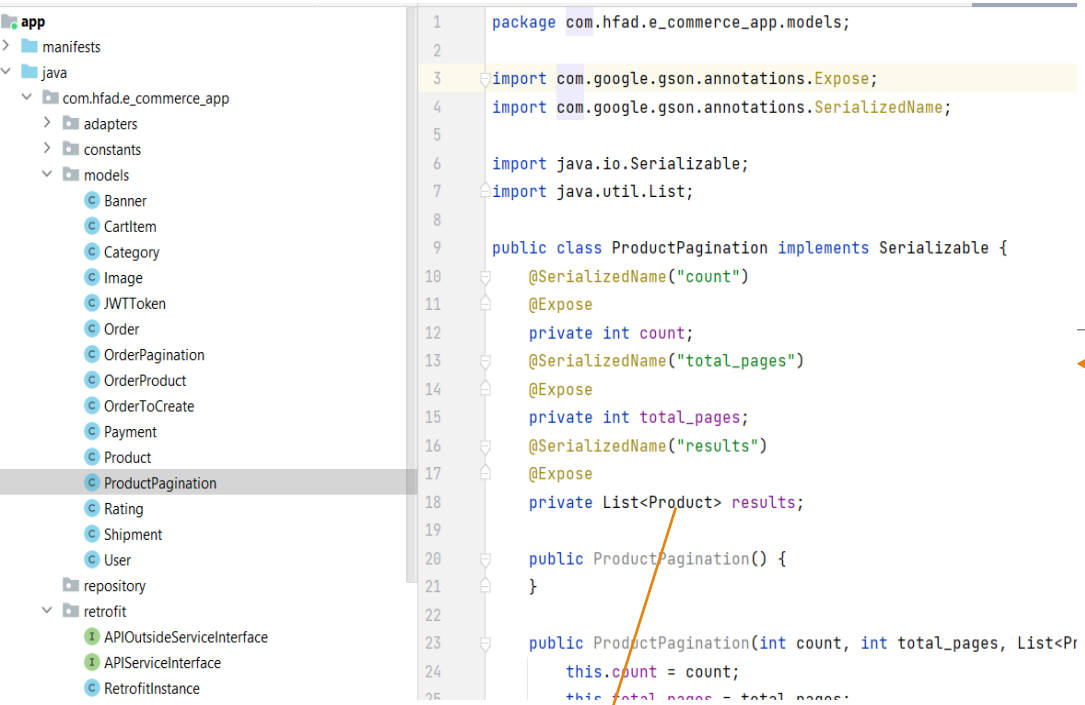
Retrofit 2

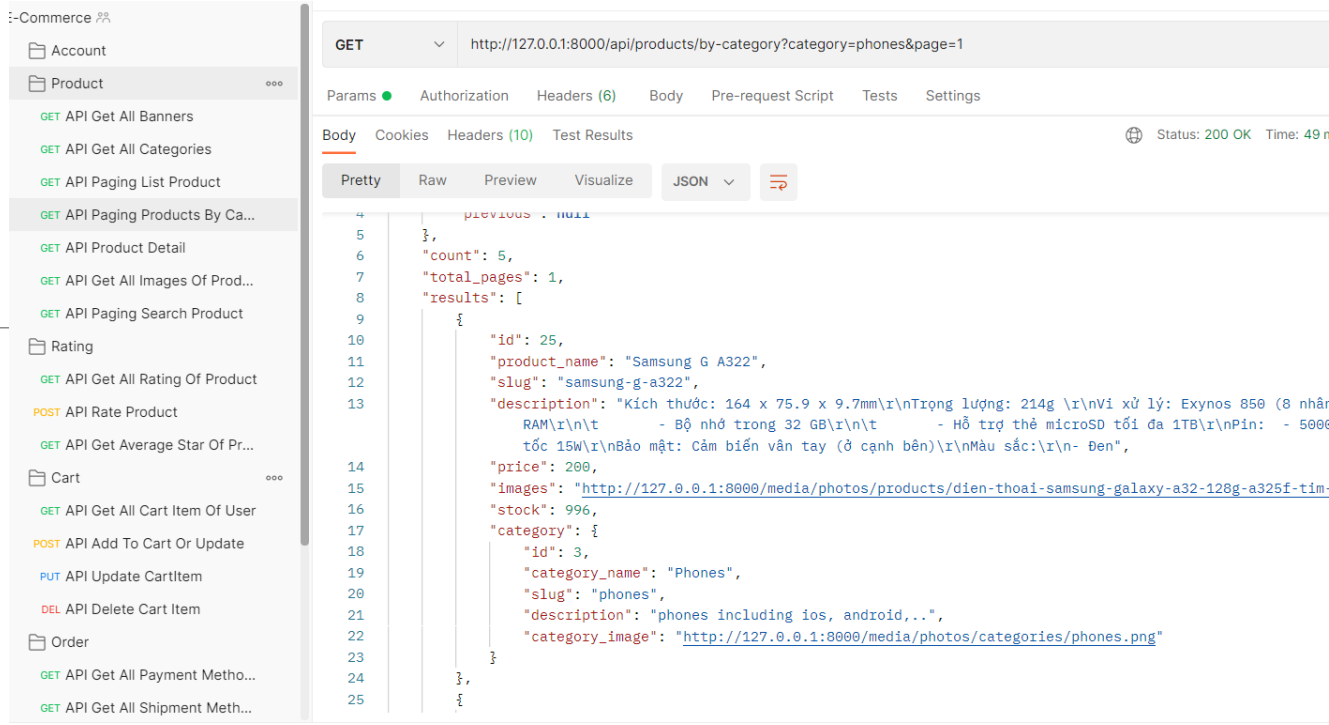
{REST}

Restful Webservices

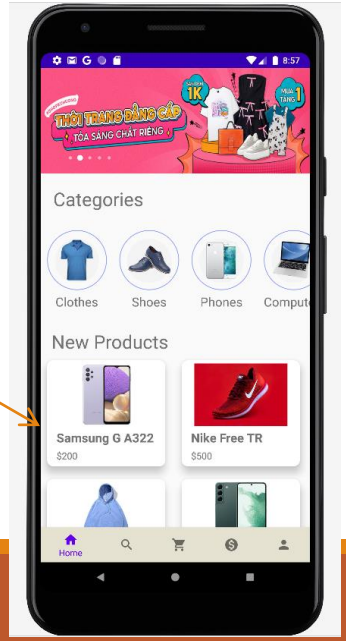
Thư viện retrofit giúp call web service từ ứng dụng android

Thư viện GSON của google giúp convert dữ liệu dạng json sang code java và ngược lại






```
mListProducts.addAll(productPagination.getResults());
productAdapter.setmListProduct(mListProducts);
```



II. Cài đặt sử dụng retrofit

```java
package com.hfad.e_commerce_app.retrofit;

import com.hfad.e_commerce_app.constants.AppConstants;

import okhttp3.OkHttpClient;
import okhttp3.logging.HttpLoggingInterceptor;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

// Singleton
public class RetrofitInstance {
    private static Retrofit retrofit;

    public static Retrofit getRetrofitInstance(){
        HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
        interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
        OkHttpClient client = new OkHttpClient.Builder()
                .addInterceptor(interceptor)
                .build();
        if(retrofit == null){
            retrofit = new Retrofit.Builder()
                    .baseUrl(AppConstants.baseURL)
                    .addConverterFactory(GsonConverterFactory.create())
                    .client(client)
                    .build();
        }
        return retrofit;
    }
```

programmer, architect, is concerned, **an instance of a class and an object mean the same thing and can often be used interchangeably**. I have never met anyone in my career that would be

+ Tạo class RetrofitInstance theo design pattern Singleton để chắc chắn rằng trong ứng dụng chỉ có 1 instance(object) của class Retrofit chứ không tạo mới instance mỗi lần gọi api.

+ Truyền baseURL(ví dụ http://10.0.2.2:8000/api/) và thư viện convert từ json sang java mong muốn(ví dụ: Gson) cho retrofit

```java
public class AppConstants {
    public static final String baseURL = "http://10.0.2.2:8000/api/";
}
```

Cài đặt sử dụng retrofit

```
import retrofit2.http.POST;
import retrofit2.http.PUT;
import retrofit2.http.Part;
import retrofit2.http.Path;
import retrofit2.http.Query;

public interface APIServiceInterface {
    @FormUrlEncoded
    @POST("accounts/login")
    Call<JWTToken> login(@Field("email") String email, @Field("password") String password);

    @FormUrlEncoded
    @POST("accounts/register")
    Call<User> register(@Field("email") String email,
                        @Field("password") String password,
                        @Field("username") String userName,
                        @Field("first_name") String firstName,
                        @Field("last_name") String lastName);

    @GET("accounts/")
    Call<User> getUserInfo(@Header("Authorization") String authHeader);

    @Multipart
    @PUT("accounts/")
    Call<ResponseBody> changeUserInfo(@Header("Authorization") String authHeader,
                                      @Part("username") RequestBody userName,
                                      @Part("first_name") RequestBody firstName,
                                      @Part("last_name") RequestBody lastName,
                                      @Part("phone_number") RequestBody phone,
                                      @Part MultipartBody.Part imageFile);
```

- Định nghĩa ra interface để call api, sử dụng các anotation của retrofit để định nghĩa xem phương thức gọi là gì (GET,POST, PUT, DELETE,...), những chuỗi như “accounts/login” chẳng hạn sẽ được retrofit append vào sau base Url ta đã định nghĩa ở bước trước, ta sẽ nhận được kết quả là “http://10.0.2.2/api/accounts/login”.

- Truyền các tham số như các trường trong form (@Field), các query param đính với Url(@Query), các Path Variable(@Path), header(@Header), body(@Body), hay @Part để truyền dạng multipart

- Kiểu dữ liệu trả về sử dụng class Call, kiểu generic là kiểu dữ liệu muốn trả về

Cài đặt sử dụng retrofit

```java
import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;

public class User {
    @SerializedName("email")
    @Expose
    private String email;
    @SerializedName("username")
    @Expose
    private String userName;
    @SerializedName("first_name")
    @Expose
    private String firstName;
    @SerializedName("last_name")
    @Expose
    private String lastName;
    @SerializedName("phone_number")
    @Expose
    private String phoneNumber;
    @SerializedName("image")
    @Expose
    private String image;

    public User(String email, String userName, String firstName,
        this.email = email;
```

- Với các lớp model ví dụ như User trong kiểu trả về Call<User> chẳng hạn, có thể sử dụng các anotation của GSON như @SerializedName (String trong dấu () ví dụ @SerializedName(“email”) sẽ tương ứng với thuộc tính của đối tượng json là {“email”: “huy123@gmail.com”,...} chẳng hạn) và @Expose

- Trước đó ta đã config cho retrofit sử dụng thư viện GSON nên lát nữa khi gọi API, retrofit sẽ gọi đến GSON để map các đối tượng json sang các đối tượng java một cách tự động.

Cài đặt sử dụng retrofit

```
import com.hfad.e_commerce_app.retrofit.APIOutsideServiceInterface;
import com.hfad.e_commerce_app.retrofit.APIServiceInterface;
import com.hfad.e_commerce_app.retrofit.RetrofitInstance;

import retrofit2.Retrofit;

public class APIUtils {
    public static APIServiceInterface getApiServiceInterface(){
        Retrofit retrofitInstance = RetrofitInstance.getRetrofitInstance();
        return retrofitInstance.create(APIServiceInterface.class);
    }
```

- Trước đó ta mới chỉ tạo ra interface để call API, vẫn còn 1 bước nữa, ta phải tạo ra 1 insance(object) có kiểu trả về là interface mà ta đã định nghĩa ra -> sử dụng phương thức create() của lớp Retrofit rồi truyền APIServiceInterface.class(interface ta đã định nghĩa) vào.

- Nên viết ra 1 lớp APIUtils để tiện đỡ viết đi viết lại code.

Cài đặt sử dụng retrofit

```
private void callAPIGetUserInfo(){
    APIUtils.getApiServiceInterface().getUserInfo( authHeader: "Bearer "+tokenManager.getAccessToken())
            .enqueue(new Callback<User>() {
                @Override
                public void onResponse(Call<User> call, Response<User> response) {
                    if(response.isSuccessful() && response.body()!=null){
                        User user = response.body();
                        if(user.getImage() !=null){
                            Glide.with(getActivity()).load(user.getImage()).into(circleImageView);
                        }
                        edEmail.setText(user.getEmail());
                        edFirstName.setText(user.getFirstName());
                        edLastName.setText(user.getLastName());
                        if(user.getPhoneNumber() !=null){
                            edPhone.setText(user.getPhoneNumber());
                        }else{
                            edPhone.setText("");
                        }
                        edUsername.setText(user.getUserName());

                        tokenManager.saveUserInfo(user.getEmail(),user.getFirstName()
                        ,user.getLastName(),user.getPhoneNumber(),user.getImage());
                    }

                }
```

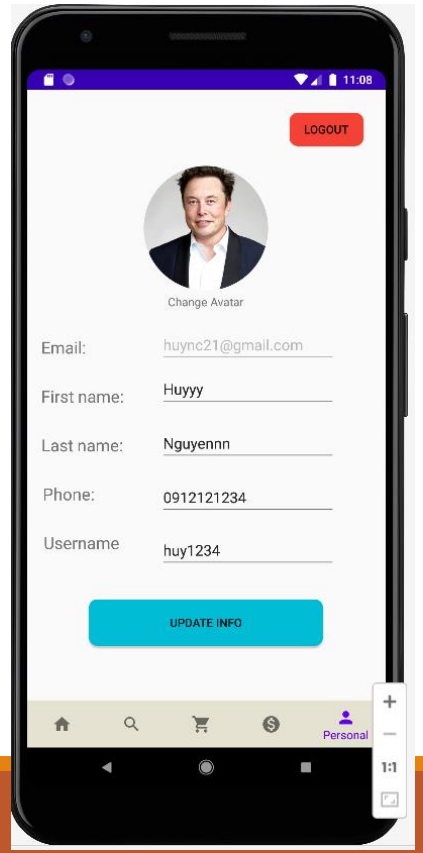
- Xong xuôi thi ta có thể sử dụng được instance của interface và gọi hàm được rồi.

- Đối tượng CallBack<User> sẽ có hàm onResponse lắng nghe sự kiện gọi api xong để trả về kết quả, sử dụng response.isSuccessful() để check thành công hay lỗi, response.body() để lấy giá trị trả về rồi sau đó sử dụng kết quả trả về để cập nhật UI.

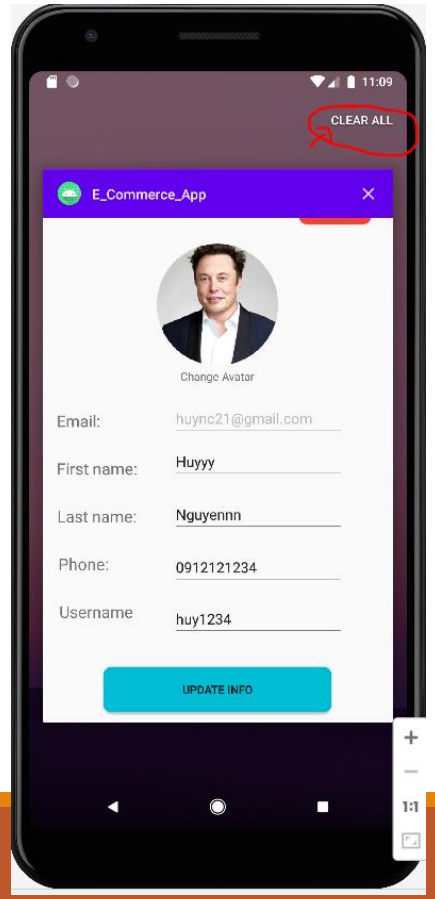
III. Một vài mẹo khi làm việc với web service trong android

1. Lưu thông tin đăng nhập vào SharedPreference

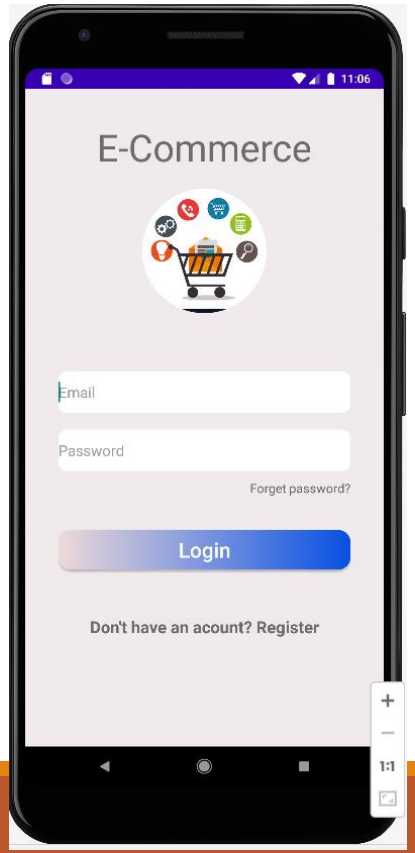
Thông tin đăng nhập ví dụ như jwt token, nếu ta chỉ lưu tạm thời vào một biến nào đó của app thì khi tắt app và mở lại, ta sẽ mất thông tin đăng nhập và người dùng lại phải đăng nhập lại



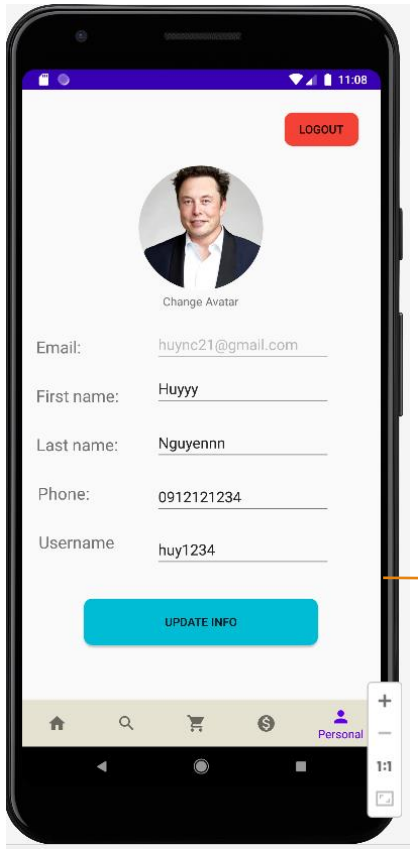
Thoát app



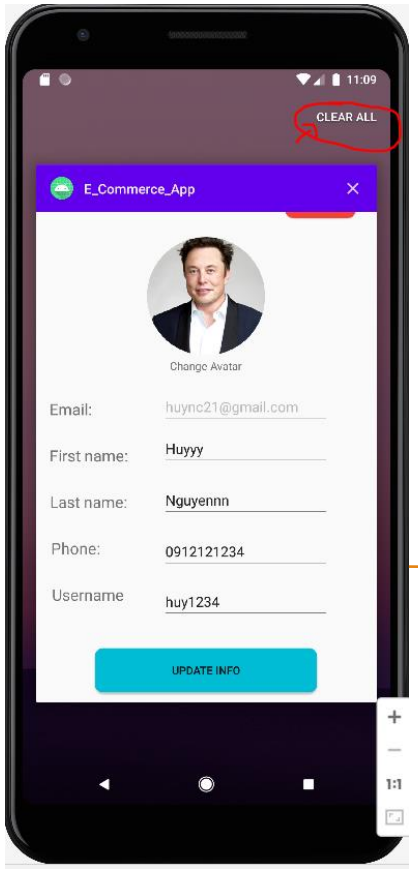
Mở lại phải đăng nhập lại

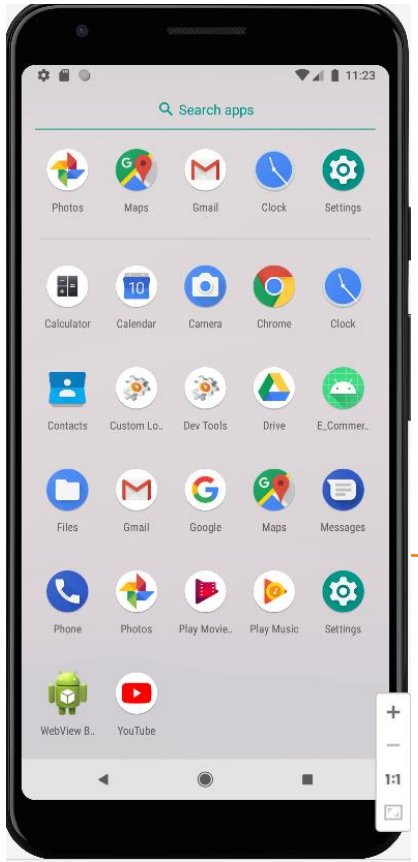


Điều ta muốn là khi user thoát app và mở lại app thì thông tin đăng nhập lần trước vẫn còn lưu lại và không phải login lại lần nữa.

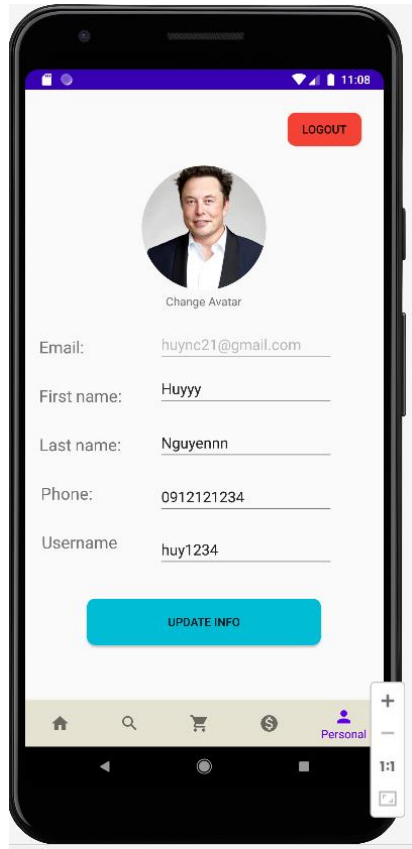


Thoát app

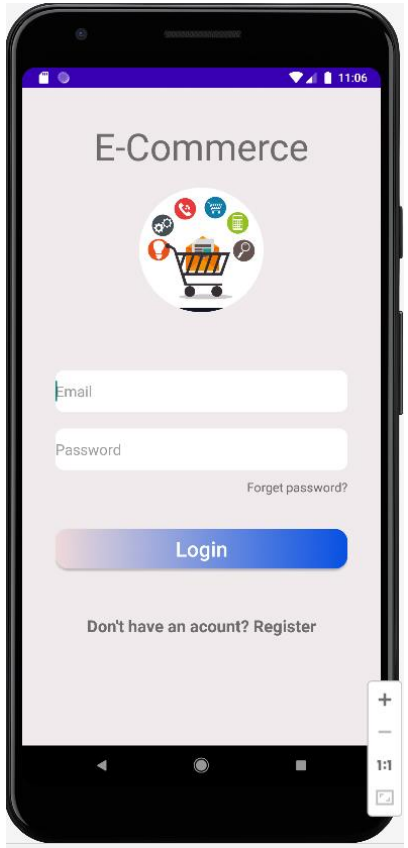




Mở lại app vẫn còn thông tin đăng nhập nên user không phải login lại



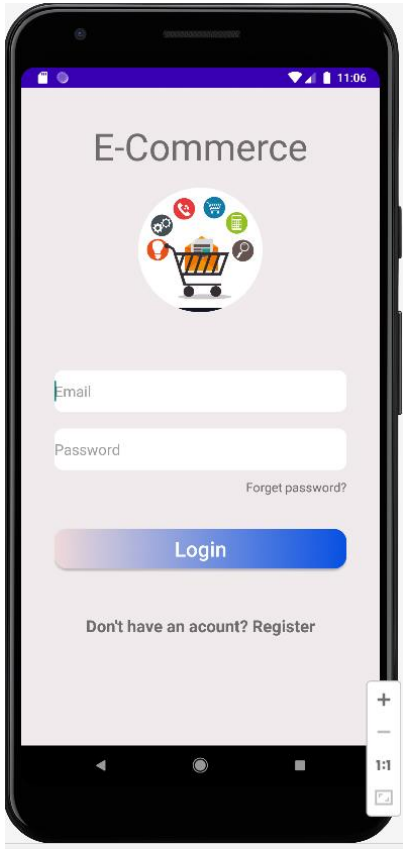
Giải pháp có thể là lưu thông tin đăng nhập vào SQLite, khi mở lại app lại thì lại truy cập vào SQLite để lấy thông tin user ra, khi hết hạn token hoặc người dùng đăng xuất thì xóa khỏi db.



thông tin user

lưu thông tin user

Hoặc một giải pháp nhẹ nhàng hơn là Shared Preferences, nơi lưu trữ các thông tin dưới dạng key-value được xây dựng sẵn trong hệ điều hành Android



thông tin user

lưu thông tin user

Có hai lớp là SharedPreferences và Editor hỗ trợ ta thao tác lưu trữ, ở đây viết ra 1 lớp là Token Manager

Lấy được Shared Preferences từ context truyền vào từ activity hay fragment sau đó lấy được Editor từ cái SharedPreferences đó để thao tác dữ liệu

Viết hàm lưu thông tin người dùng, sử dụng đối tượng editor đã lấy được trước đó để thao tác lưu dữ liệu

```
public class TokenManager {
    private SharedPreferences sharedPreferences;
    private SharedPreferences.Editor editor;
    private int Mode = 0;
    public static final String REFNAME = "JWT_SHARED_PREFERENCE";
    public static final String USER_EMAIL = "email";
    public static final String USER_FIRST_NAME = "firstName";
    public static final String USER_LAST_NAME = "lastName";
    public static final String USER_PHONE = "phone";
    public static final String ACCESS_TOKEN = "accessTOKEN";
    public static final String REFRESH_TOKEN = "refreshTOKEN";
    public static final String IMAGE_URL = "imgurl";
    private Context context;

    public SharedPreferences.Editor getEditor() { return editor; }

    public SharedPreferences getSharedPreferences() { return sharedPreferences; }

    public TokenManager(Context context) {
        this.context = context;
        sharedPreferences = context.getSharedPreferences(REFNAME,Mode);
        editor = sharedPreferences.edit();
    }

    public void saveTokenAndUserInfo(String email, String firstName, String lastName, String phon
        editor.putString(USER_EMAIL,email);
        editor.putString(USER_FIRST_NAME,firstName);
        editor.putString(USER_LAST_NAME,lastName);
        editor.putString(USER_PHONE,phone);
        editor.putString(ACCESS_TOKEN,accessToken);
        editor.putString(REFRESH_TOKEN,refreshToken);
```

Trong login activity, tạo ra 1 đối tượng thuộc lớp TokenManager trong hàm onCreate(), sau đó dùng hàm saveTokenAndUserInfo() đã viết trước đó để lưu thông tin đăng nhập

```java
public class LoginActivity extends AppCompatActivity {
    private TextView editTextEmail, editTextPassword;
    private Button btnLogin;
    private TextView txtRegister;
    private TextView txtErrorEmail, txtErrorPassword;
    private ProgressBar progressBar;

    // Token manager để lưu access token vào SharedPreference
    private TokenManager tokenManager;
    public static final int REQUEST_CODE = 1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        initView();
        tokenManager = new TokenManager( context: this);
        btnLogin.setOnClickListener(view -> {
            progressBar.setVisibility(View.VISIBLE);
            disableGUI();
            callLoginApi();
        });

        txtRegister.setOnClickListener(view -> {
            Intent intent = new Intent( packageContext: LoginActivity.this, RegisterActivity.cl
            startActivityForResult(intent, REQUEST_CODE);
        });
    }
```

```java
private void callLoginApi() {
    APIUtils.getApiServiceInterface()
            .login(editTextEmail.getText().toString(), editTextPassword.getText().toString())
            .enqueue(new Callback<JWTToken>() {
                @Override
                public void onResponse(Call<JWTToken> call, Response<JWTToken> response) {
                    enableGUI();
                    progressBar.setVisibility(View.GONE);
                    if (response.isSuccessful() && response != null) {
                        Log.d( tag: "RESPONSE Login Activity", response.body().toString());
                        String message = response.body().toString();
                        // Đăng nhập thành công thì lưu token vào share preference thông qua lớp TokenManager mà
                        // mình đã viết
                        JWTToken jwtToken = response.body();
                        tokenManager.saveTokenAndUserInfo(jwtToken.getEmail(), jwtToken.getFirstName()
                                , jwtToken.getLastName(), jwtToken.getPhone(), jwtToken.getAccessToken(),
                                jwtToken.getRefreshToken(), jwtToken.getImage());
                        returnToMainActivity();
                    } else {
                        // Nếu đăng nhập thất bại thì hiển thị ra khung message lỗi
                        try {
                            String errorMessage = response.errorBody().string();
                            JSONObject errorJson = new JSONObject(errorMessage);
                            Iterator<String> iter = errorJson.keys();
                            while (iter.hasNext()) {
                                String key = iter.next();

                            }
                            Toast.makeText(LoginActivity.this, errorMessage, Toast.LENGTH_LONG).show();
                            AlertDialog.Builder builder = new AlertDialog.Builder( context: LoginActivity.this);
```

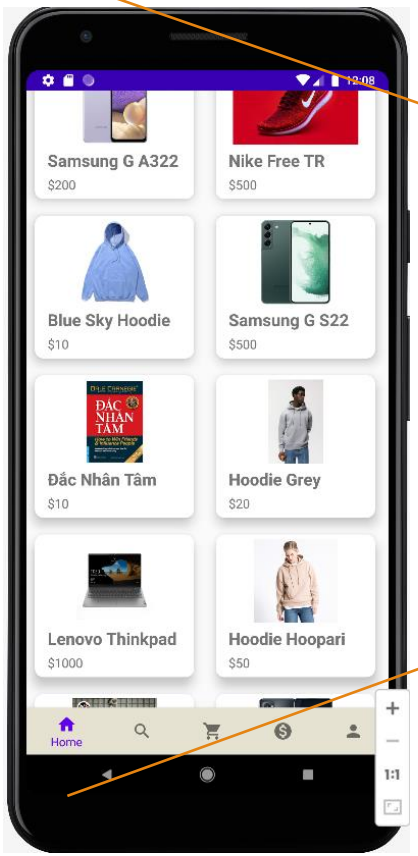
Khi gọi 1 web service nào đó mà cần có quyền truy cập của người dùng, ví dụ như lấy ra tất cả đơn hàng của người dùng đó thì ta cần access token. Khi đó ta chỉ việc lấy ra access token ta đã lưu trước đó từ đối tượng tokenManager và gửi kèm vào header của request

```
public class TokenManager {
    private SharedPreferences sharedPreferences;
    private SharedPreferences.Editor editor;
    private int Mode = 0;
    public static final String REFNAME = "JWT_SHARED_PREFERENCE";
    public static final String USER_EMAIL = "email";
    public static final String USER_FIRST_NAME = "firstName";
    public static final String USER_LAST_NAME = "lastName";
    public static final String USER_PHONE = "phone";
    public static final String ACCESS_TOKEN = "accessTOKEN";
    public static final String REFRESH_TOKEN = "refreshTOKEN";
    public static final String IMAGE_URL = "imgurl";
    private Context context;
    public SharedPreferences.Editor getEditor() { return editor; }
    public SharedPreferences getSharedPreferences() { return sharedPreferences; }
    public TokenManager(Context context) {...}
    public void saveTokenAndUserInfo(String email, String firstName, String lastName, String phone, String
    public void saveUserInfo(String email, String firstName, String lastName, String phone, String image){
    public void removeTokenAndUserInfo(){...}

    public String getAccessToken(){
        return sharedPreferences.getString(ACCESS_TOKEN, s1: null);
    }
```

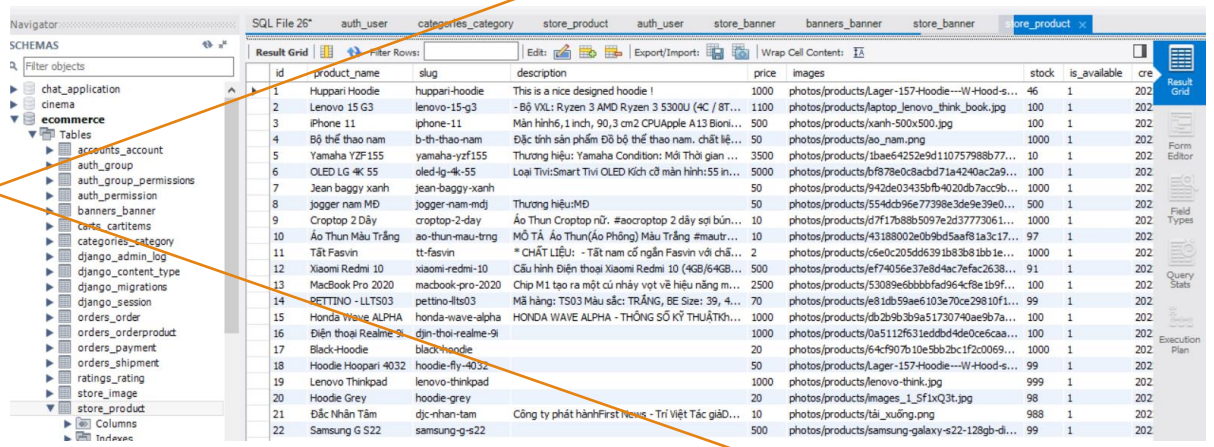
```
private void callAPIGetOrderWithPagination(int page){
    if(page<=totalPage && !isLoading){
        ordersAdapter.addLoadingEffect();
        isLoading = true;
    }
    APIUtils.getApiServiceInterface().getAllOrdersWithPagination( authHeader: "Bearer "+tokenManager.getAccessToken(), page)
            .enqueue(new Callback<OrderPagination>() {
                @Override
                public void onResponse(Call<OrderPagination> call, Response<OrderPagination> response) {
                    if(isLoading){
                        ordersAdapter.removeLoadingEffect();
                        isLoading=false;
                    }
                    if(response.isSuccessful() && response.body()!=null){
                        OrderPagination orderPagination = response.body();
                        List<Order> listOrder = orderPagination.getResults();
                        mListOrder.addAll(listOrder);
                        totalPage = orderPagination.getTotal_pages();
                        ordersAdapter.setmListOrders(mListOrder);

                        OrdersFragment.this.page++;

                    }
                }

                @Override
                public void onFailure(Call<OrderPagination> call, Throwable t) {
                    if(page<totalPage)
                        ordersAdapter.removeLoadingEffect();
```

2. Xử lý phân trang

Việc lấy ra tất cả sản phẩm trong 1 lần người dùng thực hiện request chắc chắn là 1 điều không thực tế vì 1 bảng product trong cơ sở dữ liệu có thể chứa hàng triệu bản ghi, và số lượng người dùng app có thể lên tới hàng triệu người dùng, mỗi người dùng có thể thực hiện nhiều request liên tục -> quá tải.

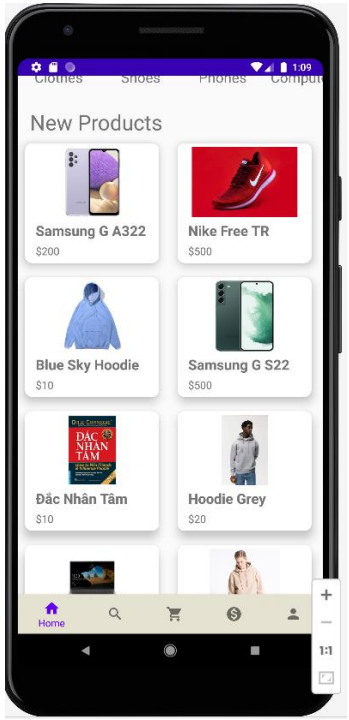


all records



Vì vậy cơ chế phân trang là rất cần thiết, trong 1 lần request, client sẽ chỉ nhận được 1 trang dữ liệu với 1 số lượng item nhất định

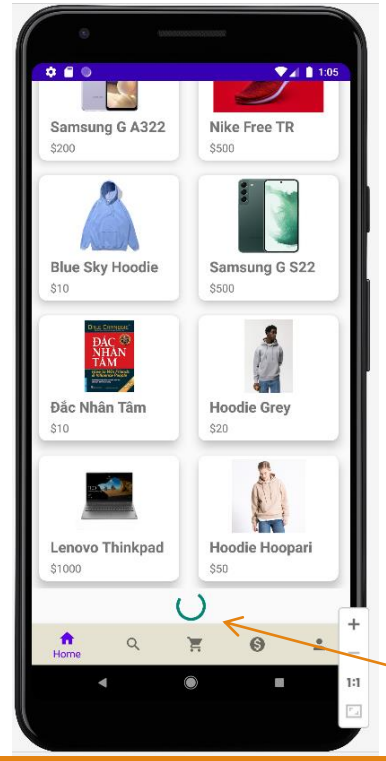
ê cho tôi danh sách sản phẩm trang 1 với 8 item



OK

Ê người dùng scroll hết 8 sản phẩm rồi, cho tôi trang 2 với 8 sản phẩm tiếp theo với

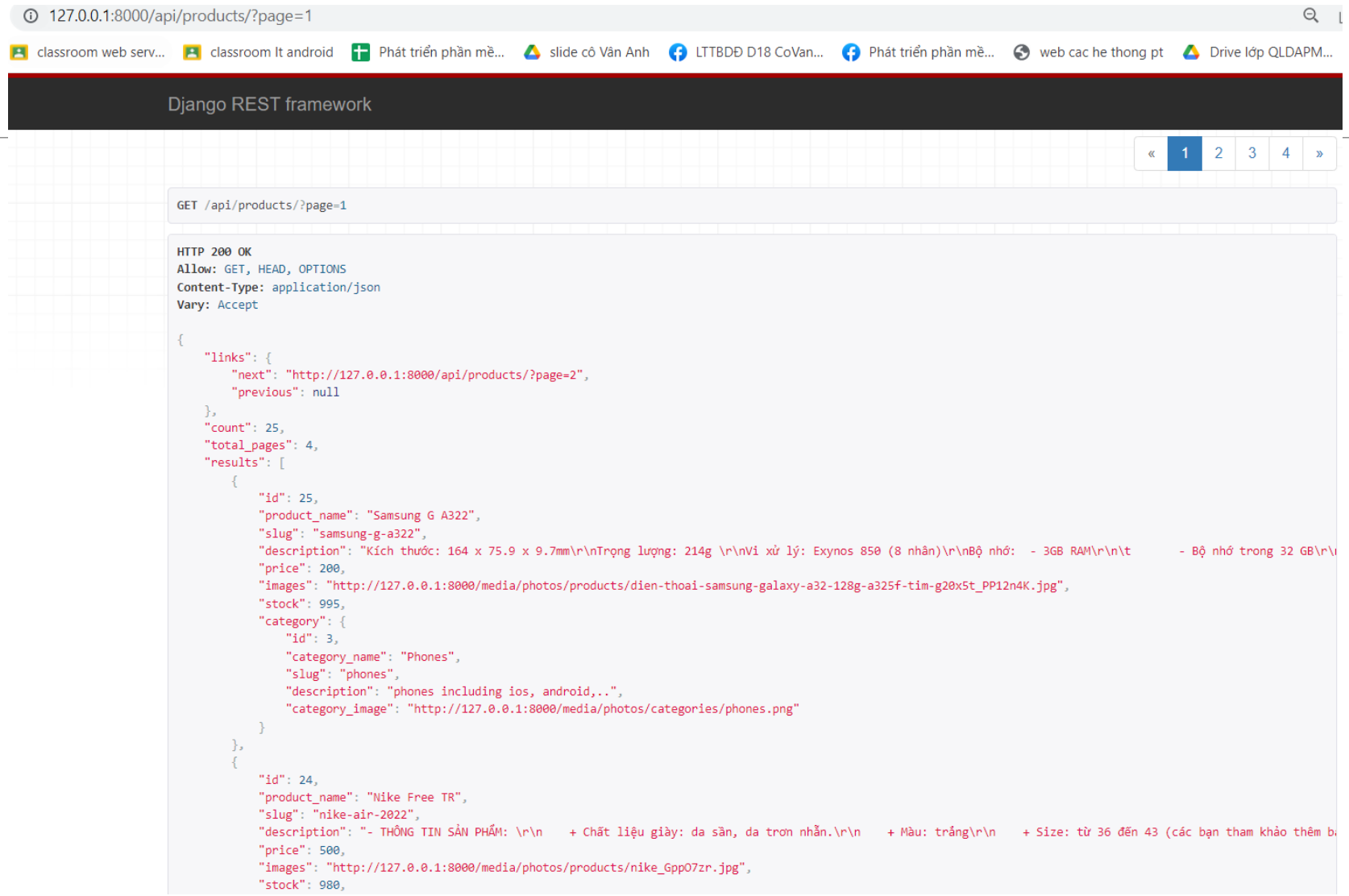
người dùng scroll hết 8 sản phẩm



OK

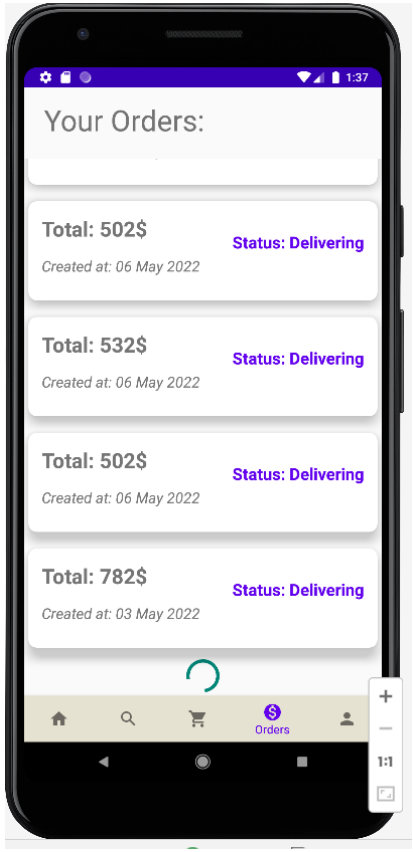
Hiệu ứng loading lúc chờ phản hồi lấy dữ liệu trang 2

Đầu tiên cần phải có 1 cái api phân trang các item, mình dùng django restframework, cái này thì mỗi 1 framework back end sẽ có 1 kiểu implementation riêng nên mình sẽ không đề cập ở đây mà chỉ tập trung vào phần xử lý ở client



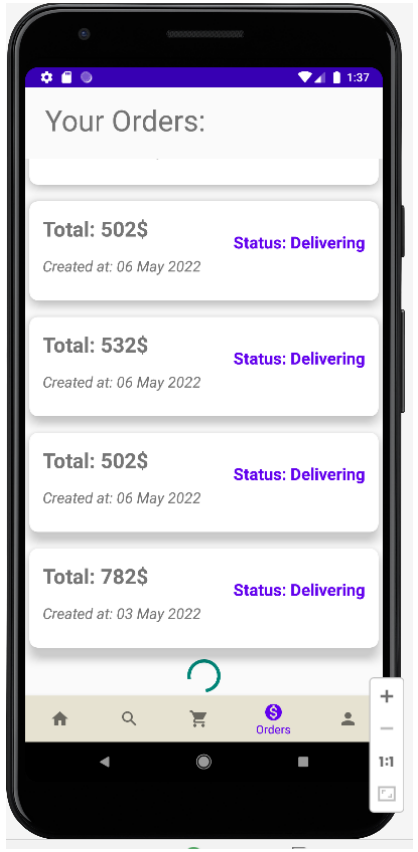
Có rất nhiều cách để xử lý logic phân trang load more trong android, ở đây là cách khá hay

Đầu tiên ta xét ví dụ hiển thị các đơn hàng của người dùng



Đầu tiên ta viết 1 cái adapter cho danh sách các đơn hàng, vì mục đích ở đây là muốn hiển thị ra hiệu đứng loading ở cuối recycler view nên sử dụng kỹ thuật multiple view type khi viết 1 cái recyclerview adapter. Đầu tiên ta phải có 2 lớp, 1 là OrderViewHolder để chứa item order, 2 là LoadingViewHolder để chứa item loading

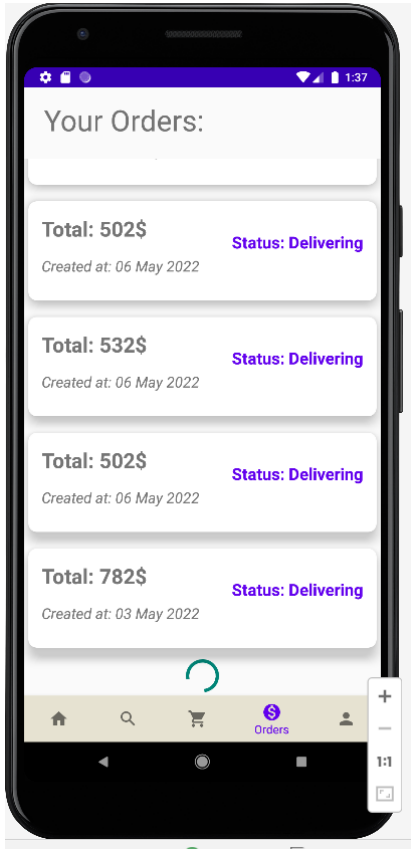
```
public class OrdersAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {
    private List<Order> mListOrders;

    private static final int TYPE_ITEM = 1;
    private static final int TYPE_LOADING = 2;

    public OrdersAdapter(List<Order> mListOrders) { this.mListOrders = mListOrders; }

    class OrderViewHolder extends RecyclerView.ViewHolder{
        private TextView tvOrderTotal, tvCreatedDate, tvStatus;
        public OrderViewHolder(@NonNull View itemView) {
            super(itemView);
            tvOrderTotal = itemView.findViewById(R.id.tv_order_total_item_order);
            tvCreatedDate = itemView.findViewById(R.id.tv_created_at_item_order);
            tvStatus = itemView.findViewById(R.id.tv_status_item_order);
        }
    }

    class LoadingViewHolder extends RecyclerView.ViewHolder{
        private ProgressBar progressBar;

        public LoadingViewHolder(@NonNull View itemView) {
            super(itemView);
            this.progressBar = itemView.findViewById(R.id.my_progress_bar);
        }
    }

    @Override
    public int getItemViewType(int position) {
        return mListOrders.get(position) == null ? TYPE_LOADING : TYPE_ITEM;
```



Sử dụng mẹo này: viết hàm addLoadingEffect() để tí nữa lúc đợi kết quả api thì gọi để thêm hiệu ứng loading. Mẹo ở đây là ta thêm 1 phần tử null vào cuối list data trong adapter.

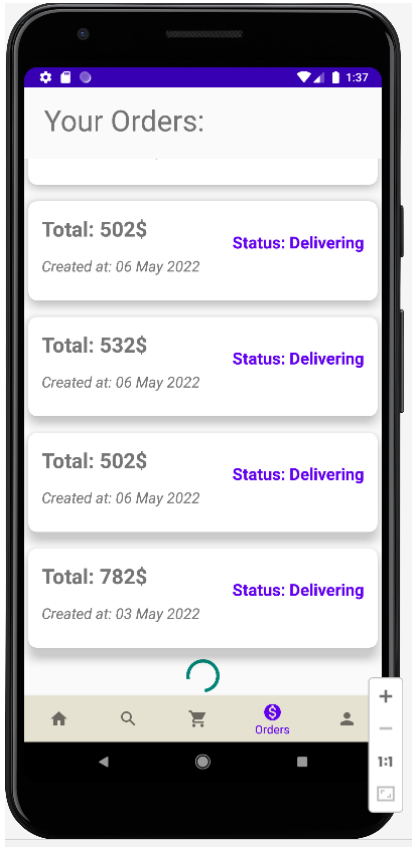
Sau đó, override lại hàm getItemViewType() của Adapter, lát nữa adapter sẽ duyệt qua các phần tử và gọi hàm getItemViewType check xem nếu phần tử đó null => kiểu item là loading, còn nếu không null thì là item

```
public void addLoadingEffect(){
    mListOrders.add(null);
    notifyItemInserted( position: mListOrders.size()-1);
}

public void removeLoadingEffect(){
    int position = mListOrders.size()-1;
    Order order = mListOrders.get(position);
    if(order==null){
        mListOrders.remove(position);
        notifyDataSetChanged();
    }
}


@Override
public int getItemViewType(int position) {
    return mListOrders.get(position) == null ? TYPE_LOADING : TYPE_ITEM;
}
```



Nếu là TYPE_ITEM thì ở hàm onCreateViewHolder trả về item order, còn nếu không thì trả về item loading, như vậy adapter của chúng ta đã vừa có thể hiển thị được các item order thông thường, vừa có thể hiển thị được item hiệu ứng loading

```
@NonNull
@Override
public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    if(viewType == TYPE_ITEM){
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_order,par
        return new OrderViewHolder(view);
    }else{
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.my_progress_ba
        return new LoadingViewHolder(view);
    }

}

@Override
public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position) {
    if(holder.getItemViewType() == TYPE_ITEM){
        Order order = mListOrders.get(position);
        OrderViewHolder orderViewHolder = (OrderViewHolder) holder;
        orderViewHolder.tvOrderTotal.setText("Total: "+order.getOrderTotal()+"$");
        orderViewHolder.tvCreatedDate.setText("Created at: "+order.getCreatedAt());

        switch (order.getStatus()) {
                case "Delivering":
                    orderViewHolder.tvStatus.setTextColor(orderViewHolder.itemView.getContex
                    break;
                case "Completed":
                    orderViewHolder.tvStatus.setTextColor(orderViewHolder.itemView.getContex
                    break;
                case "Canceled":
                    orderViewHolder.tvStatus.setTextColor(orderViewHolder.itemView.getContex
```



Vậy là xong phần adapter, ta tiếp tục sang phần chính là gọi api.
Đầu tiên khi mới vào 1 fragment hay 1 activity, ta cho 1 biến của lớp là page=1, sau đó, gọi api lấy ra trang đầu tiên và hiển thị ra

```
public class OrdersFragment extends Fragment {
    private RecyclerView recyclerViewOrders;
    private OrdersAdapter ordersAdapter;
    SwipeRefreshLayout swipeRefreshLayout;
    private TokenManager tokenManager;
    private int page=1;
    private int totalPage;
```

```
    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        recyclerViewOrders = view.findViewById(R.id.recycler_view_orders);
        swipeRefreshLayout = view.findViewById(R.id.swipeLayout);
        ordersAdapter = new OrdersAdapter(mListOrder);
        recyclerViewOrders.setAdapter(ordersAdapter);
        LinearLayoutManager linearLayoutManager = new LinearLayoutManager(getActivity());
        recyclerViewOrders.setLayoutManager(linearLayoutManager);
        recyclerViewOrders.addOnScrollListener(onScrollStateChanged(recyclerView, newState) -> {
                super.onScrollStateChanged(recyclerView, newState);

                if (!recyclerView.canScrollVertically( direction: 1) && newState==RecyclerView.SCRO
                    if(page<=totalPage)
                        callAPIGetOrderWithPagination(page);
                }
        });

        tokenManager = new TokenManager(getActivity());

        callAPIGetOrderWithPagination(page);
```

Thêm hiệu ứng loading

Bỏ hiệu ứng loading nếu gọi thành công

Cộng số page lên nếu gọi api thành công

```
private void callAPIGetOrderWithPagination(int page){
    if(page<=totalPage && !isLoading){
        ordersAdapter.addLoadingEffect();
        isLoading = true;
    }
    APIUtils.getApiServiceInterface().getAllOrdersWithPagination( authHeader: "Bearer "+tokenMa
            .enqueue(new Callback<OrderPagination>() {
                @Override
                public void onResponse(Call<OrderPagination> call, Response<OrderPagination>
                    if(isLoading){
                        ordersAdapter.removeLoadingEffect();
                        isLoading=false;
                    }
                    if(response.isSuccessful() && response.body()!=null){
                        OrderPagination orderPagination = response.body();
                        List<Order> listOrder = orderPagination.getResults();
                        mListOrder.addAll(listOrder);
                        totalPage = orderPagination.getTotal_pages();
                        ordersAdapter.setmListOrders(mListOrder);

                        OrdersFragment.this.page++;

                    }
                }

                @Override
                public void onFailure(Call<OrderPagination> call, Throwable t) {
```

Bắt sự kiện scroll xuống cuối màn hình, khi sự kiện xảy ra thì check xem page có lớn hơn page cuối ko, không thì gọi api lấy thêm trang

```
@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    recyclerViewOrders = view.findViewById(R.id.recycler_view_orders);
    swipeRefreshLayout = view.findViewById(R.id.swipeLayout);
    ordersAdapter = new OrdersAdapter(mListOrder);
    recyclerViewOrders.setAdapter(ordersAdapter);
    LinearLayoutManager linearLayoutManager = new LinearLayoutManager(getActivity());
    recyclerViewOrders.setLayoutManager(linearLayoutManager);

    recyclerViewOrders.addOnScrollListener(new RecyclerView.OnScrollListener() {
        @Override
        public void onScrollStateChanged(RecyclerView recyclerView, int newState) {
            super.onScrollStateChanged(recyclerView, newState);

            if (!recyclerView.canScrollVertically( direction: 1) && newState==RecyclerView.SCROLL_STATE_IDLE) {
                if(page<=totalPage)
                    callAPIGetOrderWithPagination(page);
            }
        }
    });

    tokenManager = new TokenManager(getActivity());

    callAPIGetOrderWithPagination(page);

    ordersAdapter.setItemClickedListener(new OrdersAdapter.ItemClickedListener() {
```

3. BottomNavigation kết hợp ViewPager hoặc ViewPager2 để tránh gọi lại api mỗi lúc chuyển fragment

```
    @Override
    public boolean onNavigationItemSelected(@NonNull MenuItem item) {
        Fragment fragment;
        switch (item.getItemId()) {
            case R.id.navigation_shop:
                toolbar.setTitle("Shop");
                fragment = new StoreFragment();
                loadFragment(fragment);
                return true;
            case R.id.navigation_gifts:
                toolbar.setTitle("My Gifts");
                fragment = new GiftsFragment();
                loadFragment(fragment);
                return true;
            case R.id.navigation_cart:
                toolbar.setTitle("Cart");
                fragment = new CartFragment();
                loadFragment(fragment);
                return true;
            case R.id.navigation_profile:
                toolbar.setTitle("Profile");
                fragment = new ProfileFragment();
                loadFragment(fragment);
                return true;
        }

        return false;
    }
};

private void loadFragment(Fragment fragment) {
    // load fragment
    FragmentTransaction transaction = getSupportFragmentManager().beginTransact
    transaction.replace(R.id.frame_container, fragment);
    transaction.addToBackStack(null);
    transaction.commit();
}
```

Cái này cô dạy rồi nên chắc cũng chỉ nói qua. Thì việc chỉ sử dụng BottomNavigation kèm framelayout sẽ xảy ra tình trạng khi click vào 1 item menu ở dưới để chuyển fragment thì nó sẽ new ra 1 cái fragment mới, đồng nghĩa vòng đời của nó được start lại từ đầu, mà ta biết ở trong hàm onViewCreated lúc đầu ta đã gọi api để hiển thị ra 1 số thứ -> Cách này sẽ khiến mỗi lúc chuyển trang lại bị gọi lại api

Giải pháp là sử dụng FragmentStateAdapter kết hợp ViewPager2 và BottomNavigationView, thì nó sẽ tự động quản lý được các fragment và không tạo lại fragment mỗi khi chuyển tab => không gọi lại api

```java
package com.hfad.e_commerce_app.adapters;

import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentActivity;
import androidx.fragment.app.FragmentManager;
import androidx.lifecycle.Lifecycle;
import androidx.viewpager2.adapter.FragmentStateAdapter;

import java.util.ArrayList;
import java.util.List;

public class MyViewPagerAdapter extends FragmentStateAdapter {
    private final List<Fragment> mFragmentList = new ArrayList<>();
    public MyViewPagerAdapter(@NonNull FragmentManager fragmentManager, @NonNull Lifecycle lifecycle) {
        super(fragmentManager, lifecycle);
    }

    public void addFragment(Fragment fragment) { mFragmentList.add(fragment); }

    @NonNull
    @Override
    public Fragment createFragment(int position) { return mFragmentList.get(position); }

    @Override
    public int getItemCount() { return mFragmentList.size(); }
}
```

note: Lớp FragmentStatePagerAdapter tuy vẫn hoạt động nhưng đã bị deprecated nên không dùng nữa

```
extends FragmentStatePagerAdapter {
```

```
// set up view pager 2
MyViewPagerAdapter myViewPagerAdapter = new MyViewPagerAdapter(getSupportFragmentManager(),getLifecycle());

HomeFragment homeFragment = new HomeFragment();
SearchFragment searchFragment = new SearchFragment();
CartFragment cartFragment = new CartFragment();
OrdersFragment ordersFragment = new OrdersFragment();
PersonalFragment personalFragment = new PersonalFragment();

myViewPagerAdapter.addFragment(homeFragment);
myViewPagerAdapter.addFragment(searchFragment);
myViewPagerAdapter.addFragment(cartFragment);
myViewPagerAdapter.addFragment(ordersFragment);
myViewPagerAdapter.addFragment(personalFragment);
viewPager2.setAdapter(myViewPagerAdapter);
```

Bắt sự kiện click vào item dưới bottom navigation view thì đổi current item của viewPager2 => fragment hiển thị đổi

```
bottomNavigationView.setOnItemSelectedListener(new NavigationBarView.OnItemSelectedListener() {
    @Override
    public boolean onNavigationItemSelected(@NonNull MenuItem item) {
        tokenManager = new TokenManager( context: MainActivity.this);
        accessToken = tokenManager.getAccessToken();
        Fragment fragment = null;
        switch (item.getItemId()) {
            case R.id.menu_item_home:
                viewPager2.setCurrentItem( item: 0, smoothScroll: false);
                break;
            case R.id.menu_item_search:
                viewPager2.setCurrentItem( item: 1, smoothScroll: false);
                break;
            case R.id.menu_item_cart:
                // Nếu không có accessToken nghĩa là đã logout
                // => Chuyển hướng đến trang login
                if (accessToken == null) {
                    Intent intent = new Intent( packageContext: MainActivity.this, LoginActivity.class);
                    startActivity(intent);
                } else {
                    viewPager2.setCurrentItem( item: 2, smoothScroll: false);
                }

                break;
            case R.id.menu_item_orders:
                // Nếu không có accessToken nghĩa là đã logout
                // => Chuyển hướng đến trang login
                if (accessToken == null) {
                    Intent intent = new Intent( packageContext: MainActivity.this, LoginActivity.class);
                    startActivity(intent);
                } else {
```

Ở đây xóa sự kiện swipe ngang vì trong view có 1 vài cái view lướt ngang mà nó bị đè hiệu ứng với cái swipe ngang của ViewPager nên khá khó chịu

```
// disable cai swipe ngang cua viewpager
viewPager2.setUserInputEnabled(false);
```