CET351_AMD DOCUMENTATION

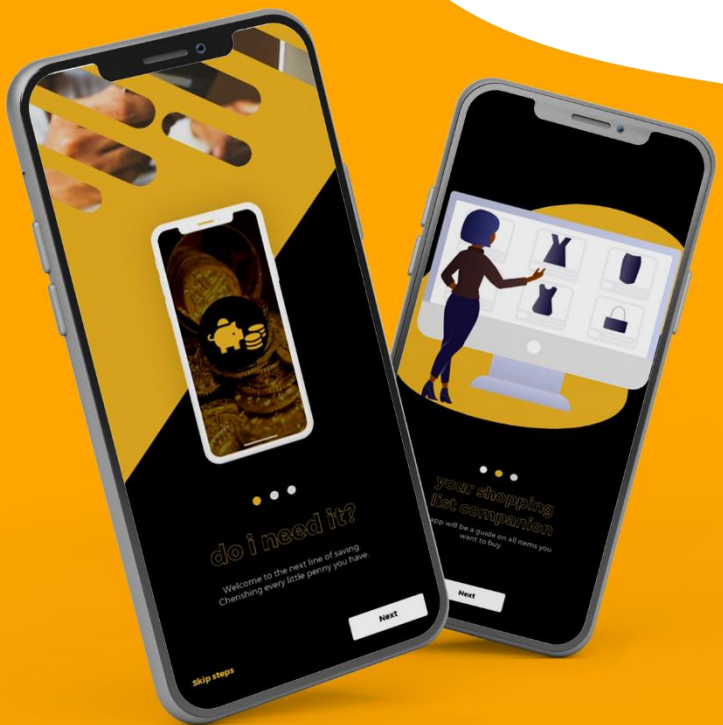
STUDENT NAME: Thabang Fenge Isaka

STUDEDNT ID : BH77NT

REGISTRATION: 209181924

DEGREE: Computer Systems Engineering

COLLEGE: Botswana Accountancy College



CONTENTS

PART A - ANALYSIS, DESIGN, FUNCTIONALITY, TESTING AND EVALUATION

1.0 INTRODUCTION

It is very key for every individual to be conscious of their spending habits through practicing financial discipline. Due to the months of lockdown and social-distancing caused by COVID 19, consumers have been compelled to shop differently – reprioritizing what is vital and substituting checkout queues for online shopping more than ever before. Therefore, the wide range of online markets on the vast virtual space of shops can be pretty tempting, leading one to spend recklessly (Chetty et al. 2020).

According to Loxton et al (2020) the global pandemic has influenced consumer micro behaviors towards spending a lot more clouded than well thought of, due to panic buying. As a result, this calls for a need to guide consumers to keep track of what they buy and if they really need those items. The “Do I Need It “application stands as the most credible app to help consumers purchase wisely and avoid buying in heist non-essential products both online and in actual shops.

This report will provide the steps on how to be build such an app which is able to register users, allow them to add products to assess later, update certain products and even provide an action to whether they are purchased or not.

2.0 ANALYSIS

The simple and not so thought of Motorola DynaTAC 8000X crafted 1983 was the inspiration that birthed modernized mobile applications designed in the 21st Century. With only a dialer and recall number set of one of 30 phone numbers truly no one saw the current revolution coming (King 2019). The industry has rapidly grown and consist of a vast sea of development tools and methodologies. Agile methodologies are the dominant techniques implemented by mobile application developers than traditional methodologies, commonly scrum, Kanban and rapid application development (RAD).

The traditional waterfall model is a linear-sequential app development lifecycle model that consist of 6 phases - requirements elicitation, design, development, testing, deployment, and maintenance. Progress in this methodology flows in a downwards direction and this methodology comprises of limitations, such that it assumes that application requirements at initial stage should be apparent and proceeding to the next phase requires total completion of the preceding stage, thus making it a non-flexible approach (Pandey et al 2018). On the other hand, scrum is a framework tailored for projects with dynamic requirements based on customer feedback. The advantages of these methodologies far outweigh the disadvantages considering their adaptability to everchanging app requirements and ability to produce quality applications from complex projects (Sharma et al 2016).

To develop mobile applications Kanban is another methodology that utilizes visual cues in the development stages. It is based more on customer demands than standard push practices aimed to make goods and immediately push them to market. This method minimizes waste activities to retain productivity and efficiency. Rapid application development is a methodology that prioritizes rapid prototype delivery and iterations. It emphasizes user feedback over stringent planning and requirements documentation.

2.1 METHODOLOGIES EVALUATION

The waterfall model is a very traditional method that cannot be implemented in the modern application development processes. It lacks flexibility due to its sequential phases hence it is bound to inefficiency and delays during the testing stage. Instead, the agile methodologies discussed are far much better as they are customer centric and accommodate application changes at any possible stages. These methodologies are best for most android development applications considering their adaptability to object-oriented programming and complex projects. These methodologies go in tandem with code management systems for version control – the likes of GitHub, Bitbucket and Google Cloud Source Repositories for collaborative work.

2.2 OPERATING SYSTEMS

Besides methodologies there are also tools used in native app development that over the past decades have evolved and become the core foundation of app development. The two renowned technologies Android and iPhone Operation System (iOS), which according to Maradin et al (2020) both share 99% percent of the market share thus have become one of the most precise examples of an oligopoly situation on the globe.

Android is an open-source mobile application operating system laid on a modified Linux kernel and has reached over 1.6 billion users worldwide in the year 2019. This extraordinary number of users is attributed to the fact that the platform is universal and open source (Lazareska et al 2017).

iOS is a proprietary software pioneered by Apple inc. It is an operating system that runs on the iPhone, iPad, and iPod Touch. User interaction with devices is performed through direct manipulation. It comprises of exceptional security features such as face recognition using Lidar technology and fine-grained controls that block apps from accessing private information of the user, such as their location.

2.2.1 OPERATING SYSTEMS EVALUATION

Although both these operating systems are made by well-established firms and are widely used, there are some key factors that distinguish them from one another. The table below shows- these key differences.

Comparison Criteria		iOS	Android
Model Defense Mechanisms	Code Signing	Yes	Yes
	Application Permissions	Very strict	Less Strict
	Sandboxing	More effective	Less effective
	Code protection	Yes	No
	Data Encryption	More advanced	Advanced but less reliable
	Antivirus Tools	Few available	Numerous available
Development Environments	Development Tools Availability	Few available	Numerous available
	Development Friendliness	Not friendly	Very friendly
	Installation Vectors	Restricted	Multiple
	Unofficial Repositories	No	Yes

2.3 PROGRAMMING LANGUAGES

Since Android and iOS are the commonly used operating systems on mobile phones, especially native apps. The common languages for iOS are swift and objective C, while Android mostly uses Java and Kotlin. These two languages are similar when it comes to object-oriented programming but are still made of attributes that distinguish them from each other.

	JAVA	KOTLIN
ANDROID	❖ Uses object-oriented programming only.	❖ Combines object-oriented programming with functional programming.
	❖ Does not offer extension functions	❖ Allows extension function
	❖ Supports implicit conversions	❖ Does not offer implicit conversions.

	SWIFT	OBJECTIVE-C
iOS	❖ Automatic memory management.	❖ Manual app memory management for CoreFoundation.
	❖ Uses one file with both interface and implementation.	❖ Uses header files and implementation files.
	❖ Allows interactive application development.	❖ Does not allow interactive application development.

2.3.1 PROGRAMMING LANGUAGES EVALUATION

To develop an application using iOS or Android depends entirely on the target audience. Android is globally used and has a higher percentage of the market share thus is the most ideal platform to develop. Android languages allow for customizable applications that suite user designers whereas iOS is very constraining and does not cater for customizability. But Android is prone to attacks due to the open-source notion behind it, unlike iOS which gives developers a simple portion to build their apps on, hence reducing vulnerabilities. Other code-based approaches for hybrid and native applications are single code-based tools. Recently Google launched Flutter a more effective approach to build apps for both iOS and android using a single code based approached that is compiled to suite both operating systems. Tools like PhoneGap, React Native, IONIC are other effective approaches to consider for web-based applications that run on mobile phones.

2.4 APPLICATION STORAGE OPTIONS AND EVALUATION

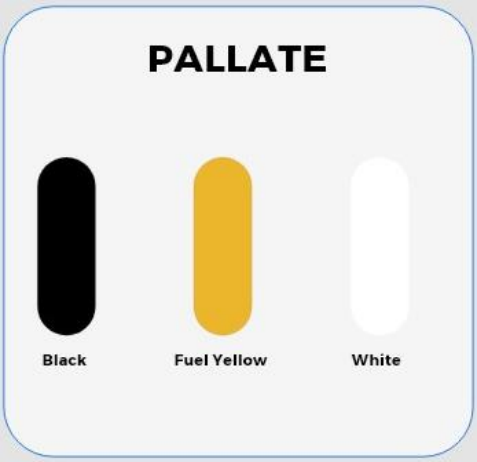
Android and iOS development both cater for cloud storage and local storage. Local storage is recommended for applications that do not depend entirely on internet-based technologies and Realtime server connections. It is highly recommended to use cloud storage as data can be accessed from anywhere. Moreover, it is a good option to backup local storage to the cloud due to common memory failures in physical storage.

3.0 DESIGN

Having discussed the different platforms and storage options for mobile app development. The Do I Need It app will be built using the Android platform considering the vast target market of 1.6 billion users and ease in development. This app aims to take advantage of this huge audience reach by instilling financial discipline in its user.

To follow good mobile application development standards, screen mockups were firstly designed to visualize the different activities to be implemented in the app. Each Screen mockup was designed using adobe Experienced Designer (Adobe XD) and then programmatically converted to XML code.

COLOR PALLATE



PURPOSE OF COLOR PALLETE

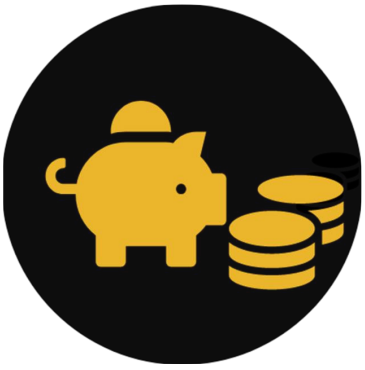
According to the material design color system, color selection should match a brand identity perfectly, must be meaningful and have a smooth user experience.

The colors chosen for the app is Fuel orange which represents gold, and this color was inspired by how banks stress the importance of saving and wise spending.

The black and white color where used to add an extra element of modern minimalistic feel to the app.

LOGO DESIGN

The logo was crafted using Adobe illustrator to compose a brand identity of the whole app.



PURPOSE OF LOGO ICON

An icon-based logo was used since is more memorable to user than text-based logos. A piggy bank with coins resonates savings and wise spending which meets the purpose of the whole app.

3.0.1 SCREEN MOCKUPS

ONBORADING SCREENS MOCKUPS

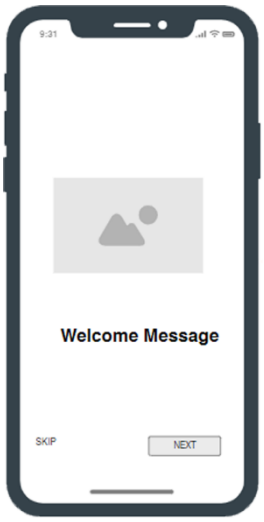
Creative onboarding screens were designed to help explain to the user core functionalities available in the app. This is essential as it gives the user the sense of interest and zeal to use the app.

ONBOARD SCREEN GENERAL MOCKUP

Three onboard screens were designed, and all followed a consistent and meaningful color scheme, as recommended by material design rules.

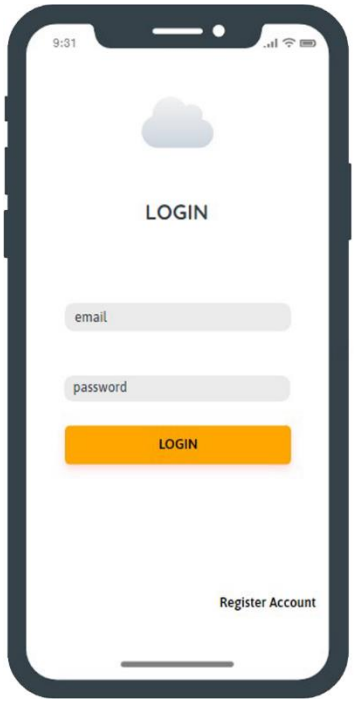
Below are the actual onboard screens designed in Adobe XD.

Mockup

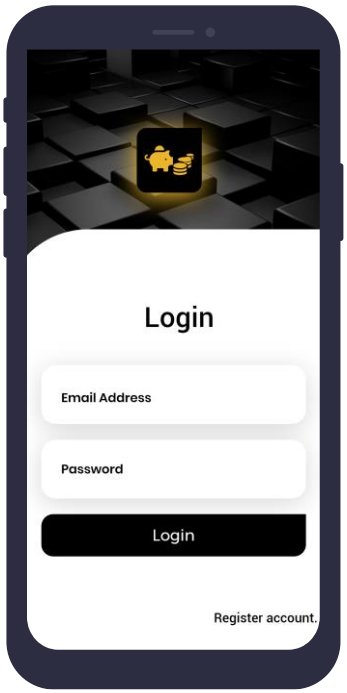


LOGIN SCREEN & REGISTER SCREEN

Mockup

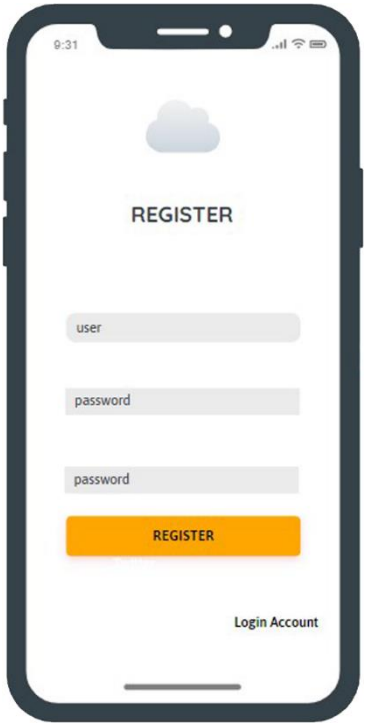


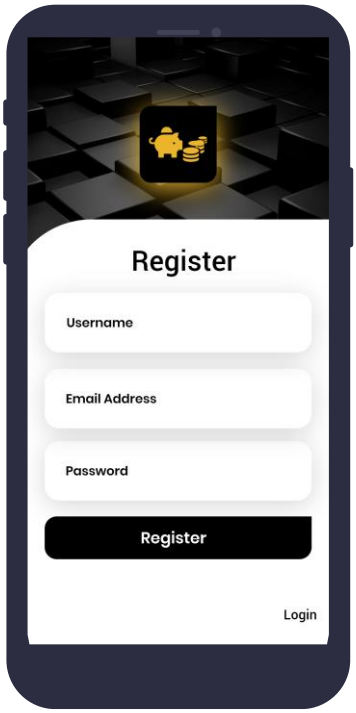
Actual Design



According to material design text fields should stand out and indicate that users can input information. The login screen and register complemented this statement by adding shadows to input fields to make them more visible.

A black color was added to the button to add contrast to the design and a logo to add creativity to the component behind it.

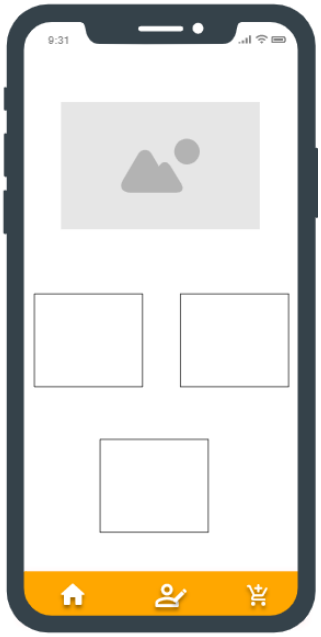




DASHBOARD & MENU NAVIGATION

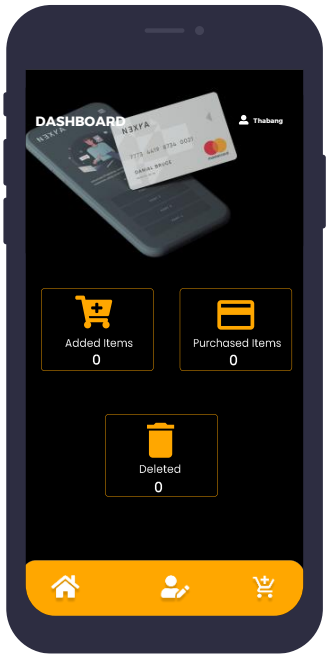
A bottom navigation was implemented with three destinations – dashboard, profile, and products. An optional label appears below the icons with a consistent margin from the icon. Navigation icons have been made to be ergonomic and easily accessible with proper padding and distance to allow a simple tap for the user. The navigation has been consistently used on each fragment.

Mockup



Card designs were made on the dashboard with a contrasting colors to establish a hierarchy between the components and a clear focus.

Actual Design



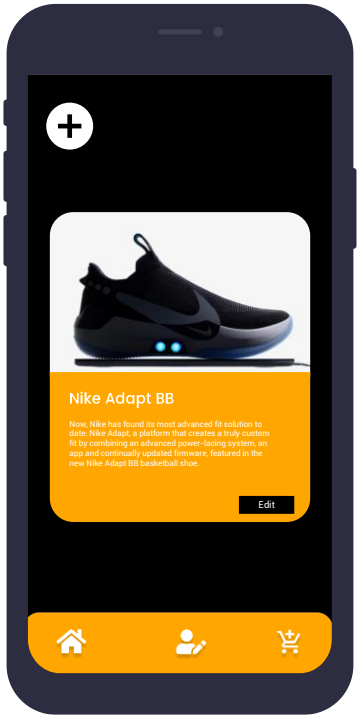
VIEW PRODUCT SCREEN

Mockup

The products page was designed with a card item of border radius 72 pixel. Coupled with an image view and an embedded edit button.

An add button was made accessible at the top for a user to upload a product. The idea was that a swipe will be implemented to view more products. This design is minimalistic and emphasis a good use of colors and space between components.

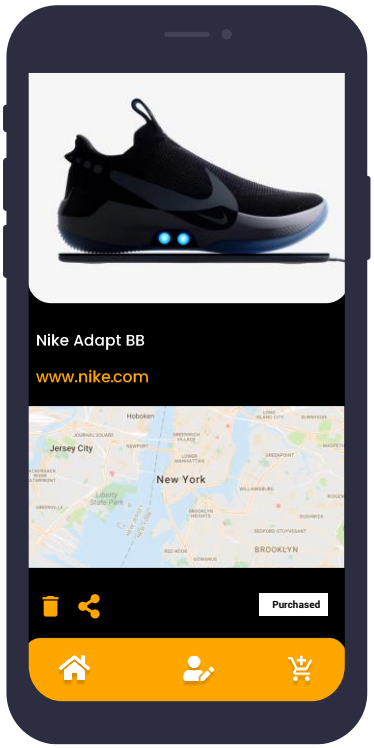
Actual Design



VIEW PRODUCT DETAILS

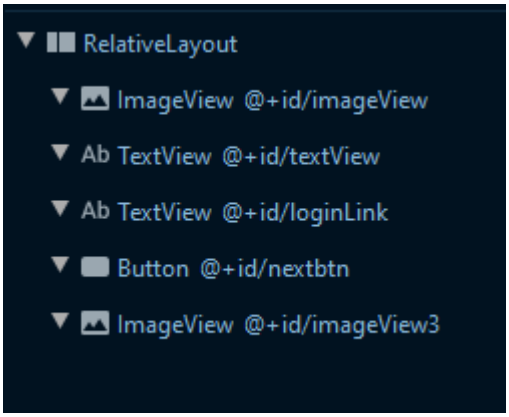
Product details page-maintained consistency with the view product design. It was made a bit more broad and creative following material recommendations to reduce the usage of unnecessary text that clutter views.

Moreover, visual cues standards emphasis that elegance in imagery give focus to user and help the feel in control and comfortable.

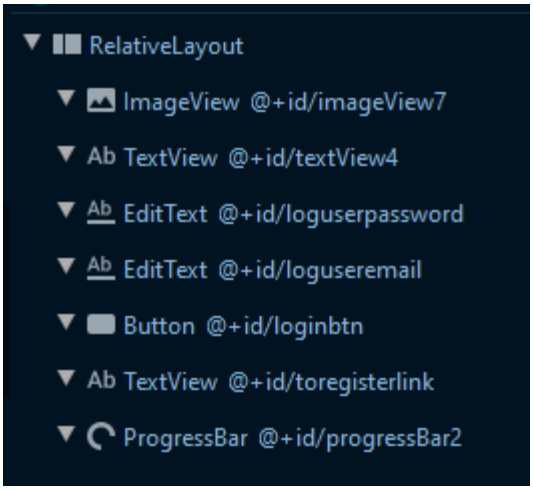


3.1 SCREEN HIERACHY

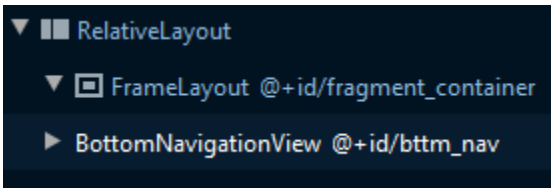
The screen hierarchies of each layout were designed considering the recommendations given by the Android documentations. Each layout emphasized the use of constraint, relative, linear and frame layouts. These layouts were purposefully implemented according to the complexity of each design hence fragments were the most ideal implementation to allow reuse and sharing of resources between activities.



On board Screen hierarchy, followed a relative layout with two text view that displayed the application name and a welcome message. The image view was vector images, and the button was a navigation to the next screen.



The Login and Register screen both used a relative layout. Consisting of input fields to collect user data for validation and registration and a submit button.



A frame layout with a fragment container was used to allow navigation between fragment layouts for the dashboard, profile, and products.

3.2 DESIGN EVALUATION

The Do I Need It App was designed following the material design standards, adhering to the color system guidelines – the color palette was well thought of and made modern yet minimalistic. The card views and image views were not cluttered with unnecessary text to avoid an unappealing user experience. The fonts used was Roboto and Poppins Bold and these are fonts that are dominantly used in current UX trends. Page viewers for swipe functionalities were designed with perfect padding and marginal distances to give a user the smooth transition from one item to the next.

4.0 FUNCTIONALITY & IMPLEMENTATION JUSTIFICATIONS

In these sections we will discuss the core functional implementations of the application – the logic and storage systems. The application was executed on a Nexus 5X API 30.

STORAGE ARCHITECTURE

The application storage system used was Firebase, a google cloud hosting platform that facilitates Realtime databases for fast and easy application development. We utilized Firebase Firestore due to its intuitive data model and rich features - faster queries and scales further than the Realtime Database. The cloud storage is the best approach as it caters for easy integrations with the Google Map API which requires internet connection.

FIREBASE AUTHENTICATION

Implementation for registration and logging into the system was implemented using an instance of Firebase Authentication. This functionality is enabled in the Firebase dashboard coupled with dependencies that allow access to the prebuilt classes. A Google JSON file was added to the application directory consisting of keys that link the app with the hosted database.

```
//Create and Store user authentication credentials

fireAuth.createUserWithEmailAndPassword(email,password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {

    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if(task.isSuccessful()) {

        Toast.makeText(getApplicationContext(),  text: "Account successfully created.", Toast.LENGTH_SHORT).show();

        // Store user details:
        userId = fireAuth.getCurrentUser().getUid();


        DocumentReference userInformation = fireStore.collection( collectionPath: "user").document(userId);

        Map<String, Object> user = new HashMap<>();
        user.put("user_name", username);


        userInformation.set(user).addOnSuccessListener(new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
                Toast.makeText(getApplicationContext(),  text: "user made ", Toast.LENGTH_SHORT).show();
                Log.d(TAG,  msg: "User details successfully stored.");
                System.out.println("This was accessed");
            }
        });
```

Implementation of firebase instance to register user with an email and password. A username is added into a Map and stored in Firestore.

```
}

progressBar.setVisibility(View.VISIBLE);
fireAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener( activity: UserLogin.this, new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if(task.isSuccessful()) {
            progressBar.setVisibility(View.INVISIBLE);
            FirebaseUser user = fireAuth.getCurrentUser();


            finish();
            goToDashboard();

        }else {
            progressBar.setVisibility(View.INVISIBLE);
            Toast.makeText( context: UserLogin.this,  text: "Sign in failed." + task.getException(),
                    Toast.LENGTH_SHORT).show();


        }
    }
});
```

Login functionality invokes an email and password function from FirebaseAuth to validate user credentials. Correct credentials redirect user to the dashboard.

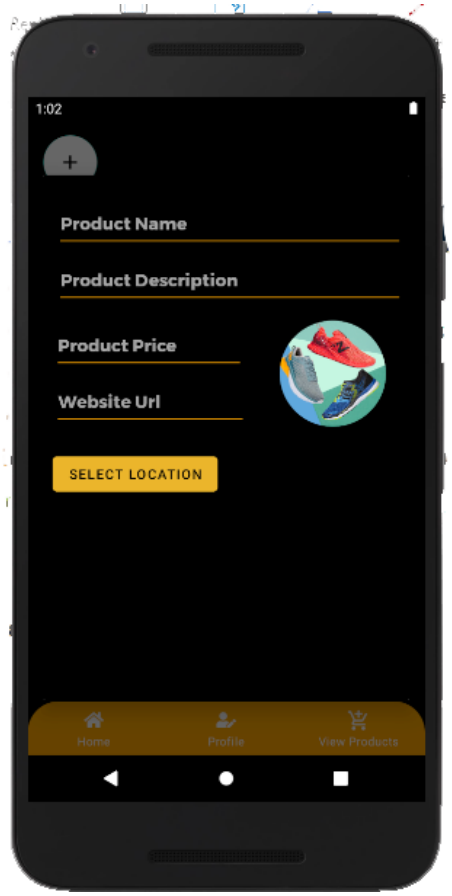
ADD PRODUCT

To add a product to Firestore an instance of Firestore storage was invoked to add an image file of the product. While the image is uploaded a string URL of the image is added to a Map or collection to be uploaded to Firestore with the product details. The product consists of Latitude, longitude, and address to the store the product location.

MAP API (DESIRABLE FUNCTIONALITY TO BE MARKED)

To geographically tag a product with a product added by a user the Google maps API was used. This API was significant as it is a robust tool to create custom maps with check in functions, Realtime location data synchronization and has a simple map SDK for to incorporate to an android project.

STEP 01: Product details – product name, image, URL, price is inserted to input views. Thereafter an intent to carry the data to a location selection activity to add a location of the product.




```
Intent intent = new Intent(getContext(), MapSelectLocation.class);
intent.putExtra( name: "product_name", productName);
intent.putExtra( name: "product_description", productDescription);
intent.putExtra( name: "product_price", productPrice);
intent.putExtra( name: "product_site", productSite);
intent.putExtra( name: "image-uri", imageUri.toString());
startActivityForResult(intent, SimplePlacePicker.SELECT_LOCATION_REQUEST_CODE);
dialog.dismiss();

System.out.println("Image here" + imageUri);
```

STEP 02: On the onReadyMapCallback the user current location is fetched to navigate the marker to be set on the current user location. The marker will then be used to select a location of the products with its address coupled with the latitude and longitude. Once a user confirms the product location the image is uploaded to Firebase Storage and an image URL is appended to Firestore with rest of the product details.

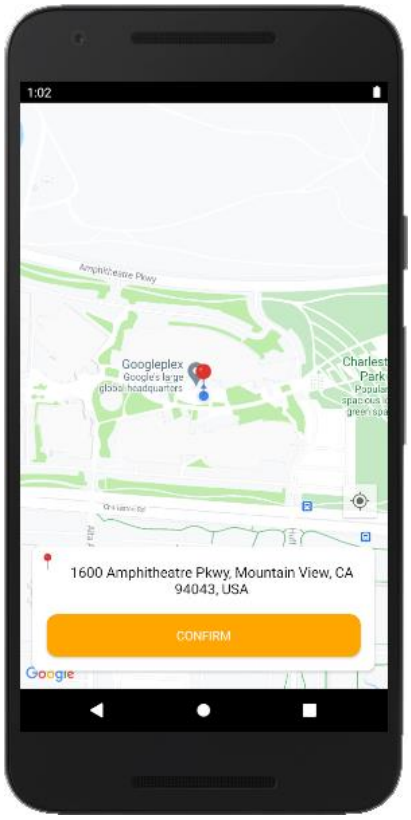
```
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    mMap.setMyLocationEnabled(true);
    //enable location button
    mMap.getUiSettings().setMyLocationButtonEnabled(true);
    mMap.getUiSettings().setCompassEnabled(false);

    //move location button to the required position and adjust params such margin
    if (mapView != null && mapView.findViewById(Integer.parseInt( s: "1")) != null) {
        View locationButton = ((View) mapView.findViewById(Integer.parseInt( s: "1")).getParent()).findViewById(Integer.parseInt( s: "2"));
        RelativeLayout.LayoutParams layoutParams = (RelativeLayout.LayoutParams) locationButton.getLayoutParams();
        layoutParams.addRule(RelativeLayout.ALIGN_PARENT_TOP, subject: 0);
        layoutParams.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM, RelativeLayout.TRUE);
        layoutParams.setMargins( left: 0, top: 0, right: 60, bottom: 500);
    }

    LocationRequest locationRequest = LocationRequest.create();
    locationRequest.setInterval(1000);
    locationRequest.setFastestInterval(5000);
    locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

    LocationSettingsRequest.Builder builder = new LocationSettingsRequest.Builder().addLocationRequest(locationRequest);

    SettingsClient settingsClient = LocationServices.getSettingsClient( activity: this);
    Task<LocationSettingsResponse> task = settingsClient.checkLocationSettings(builder.build());
```




```
productImages.putFile(imageuri).addOnCompleteListener(task -> productImages.getDownloadUrl()
        .addOnSuccessListener(new OnSuccessListener<Uri>() {
            @Override
            public void onSuccess(Uri uri) {
                url = uri.toString();
                    String status ="Pending";

                //Store Product Details To firestore

                //Add product document to firestore for user
                userId = fireAuth.getCurrentUser().getUid();
                DocumentReference product = fireStore.collection( collectionPath: "products").document();
                Map<String, Object> productinformation = new HashMap<>();
                productinformation.put("date_added", new Date().toString());
                productinformation.put("product_name", productName);
                productinformation.put("product_owner", Objects.requireNonNull(fireAuth.getCurrentUser()).getEmail());
                productinformation.put("product_description", productDescription);
                productinformation.put("product_site", productSite);
                productinformation.put("product_price", productPrice);
                productinformation.put("user_id", userId);
                productinformation.put("product_address", address);
                productinformation.put("latitude", latitude);
                productinformation.put("longitude", longitude);
                productinformation.put("image_url",url);
                productinformation.put("status",status);
```

VIEW PRODUCT

After adding a product, a QuerySnapshot to Firestore is implemented to fetch the products from Firebase to a Page Adapter and then displayed in a card view.

Model Class with getters, setters, and a constructor.

```
public class MyModel {

    public String getName() {
        return name;
    }

    public void setName(String Name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getDate() {
        return date;
    }

    public void setDate(String date) {
        this.date = date;
    }

    public String getImage() {
        return image;
    }

    public void setImage(String image) {
        this.image = image;
    }

    public String getAddress() {
        return address;
    }
```

Firebase query to fetch products added by the user.

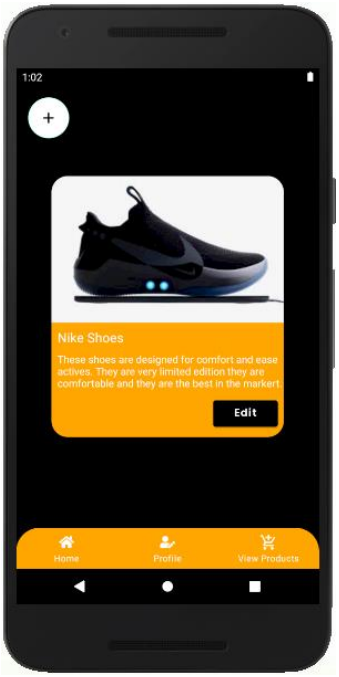
```
Query query = mFirestore.collection( collectionPath: "products").whereEqualTo( field: "product_owner", fireAuth.getCurrentUser().getEmail());
query.get()
        .addOnCompleteListener(task -> {
            if (task.isSuccessful()) {

                //Fetch from firebase all the documents
                note.setVisibility(View.VISIBLE);
                for (QueryDocumentSnapshot document : Objects.requireNonNull(task.getResult())) {
                    Log.d(String.valueOf(TAG),  msg: document.getId() + " => " + document.getData());
                    String prodId = document.getId().toString();
                    String date = document.getString( field: "date_added");
                    String productName = document.getString( field: "product_name");
                    String owner = document.getString( field: "product_owner");
                    String product_description = document.getString( field: "product_description");
                    String product_site = document.getString( field: "product_site");
                    String product_price = document.getString( field: "product_price");
                    String user_id = document.getString( field: "user_id");
                    String product_address = document.getString( field: "product_address");
                    int lat = document.getLong( field: "latitude").intValue();
                    String latitude = String.valueOf(lat);
                    int longi = document.getLong( field: "longitude").intValue();
                    String longitude = String.valueOf(longi);
                    String image_url = document.getString( field: "image_url");
```

An adapter to display the products on the card view.

```
MyModel model = modelArrayList.get(position);

    String name = model.getName();
    String description = model.getDescription();
    String image = model.getImage().trim();
    Glide.with(context).load(image)
            .placeholder(R.drawable.ic_launcher_background)
            .error(R.drawable.ic_launcher_background)
            .disallowHardwareConfig()
            .into(bannerIv);
    titleTv.setText(name);
    descriptionTv.setText(description);
```



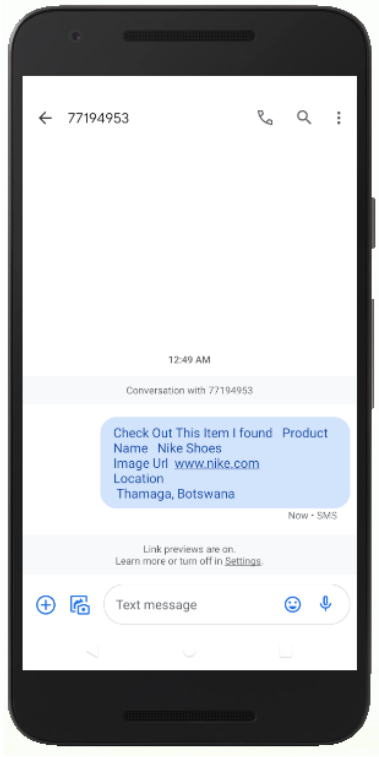
SHARE PRODUCT

Sharing a product forms the core functionality of these app to do so an OnClickListner on the share icon is implemented. The listener invokes an ACTION SEND intent type to provide applications to share with. The builder is appended with the product details that are automatically filled to the message box.

```
//Share Item to another App
share.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        //Send Item to another application
        Intent sendIntent = new Intent(Intent.ACTION_SEND);
        sendIntent.setType("text/plain");
        sendIntent.putExtra(Intent.EXTRA_TEXT,  value: "Check Out This Item I found   " +"Product Name   " + display_product_name + "\n" +
                "\n" + "Location   "+ "\n" + address);

        startActivity(sendIntent);
    }
});
```



5.0 TEST STRATEGY AND TEST RESULTS

To test our application and see if it meets all necessary requirements, a functional test is performed and relayed in a tabular format. This table will be made of these attributes; test case, description, expected outcomes, actual results, and comments. To perform these tests the application will undergo a series of stress test to evaluate each functionality to find flaws. Application bugs will be recorded to be refined in future versions.

REQUIREMENTS TESTED

REGISTER

TEST CASE	DESCRIPTION	EXPECTED OUTCOME	ACTUAL RESULTS	COMMENTS
1. Form Validation	When a user submits the registration form with empty fields, they are prompted to first fill the required details.	**01:** Empty user fields invoke a toast message insisting the user to input an email, username, and password.	✓ Input fields were validated for null fields prompting user to fill them.	Successful form validation that prevents empty submissions that lead to null pointer exceptions.
2. Generate user account.	Once a user submits their credentials to register. Their account is created in Firebase.	**01:** Form data from input fields is captured and stored. **02:** Once the data is captured user details are updated to firebase. Hence an account is created.	✓ Form data was captured, and user account was registered on Firebase users, with an email, password, and username.	Successful account creation.
3. Redirect user to login activity.	Right after successfully registering a user must be redirected to the Login activity to sign in.	**01:** After form data is submitted a toast message is displayed to the user confirming their account. **02:** The user is then navigated to the Login screen to sign in.	✓ User received a toast message confirmation for successful registration.	Application syncs seamlessly with the database.

LOGIN

TEST CASE	DESCRIPTION	EXPECTED OUTCOME	ACTUAL RESULTS	COMMENTS
4. Form Validation	Attempt to submit login credentials with empty fields.	**01:** Empty user fields invoke a toast message insisting the user to input an email with password.	✓ Input fields were validated for null fields prompting user to fill them.	Successful form validation that prevents empty submissions that lead to null pointer exceptions.
5. Authenticate user account.	Submit correct and wrong credentials to login. Wrong credentials respond with an error message while correct one a success message.	**01:** Form data from input fields is captured and stored. **02:** Form data is captured and verified from the database.	✓ Form data was captured, and credentials were validated from Firebase users by comparing the email and password.	Account verified.
6. Redirect verified user to dashboard activity.	For successful authentication, the user is redirected to the dashboard.	**01:** After form data is submitted a toast message is displayed to the user confirming their account is verified. **02:** The user is then navigated to the dashboard.	✓ User received a toast message confirmation a successful authentication.	

ADD ITEM DIALOG

TEST CASE	DESCRIPTION	EXPECTED OUTCOME	ACTUAL RESULTS	COMMENTS
7. Add item dialog.	On the add product fragment when a user clicks the add icon a form appears with input fields.	**01:** Dialog must appear with a form layout to add product details.	✓ Dialog with a layout appears when the add button is tapped. With product input fields.	Dialog appeared perfectly with an overlay of an xml relative to the layout.
8. Add product details.	User adds the product name, price, description, URL, and image to the form.	**01:** Form captures from data in the editable fields	✓ Input fields accepts characters typed and displays them.	
9. Submit empty product fields.	Empty fields are not accepted when the submit is clicked rater user is prompted to fill them	**01:** If fields are submitted empty user is prompted to fill the fill the accordingly.	✓ Only the product name, price, description, URL toast messages appeared if empty. × Image field was not validated and lead to null pointer exception causing app to stop.	Product upload works only if everything is filled. If the image is empty it does not validate it rather cause the app to stop.
10. Pass product details to Map activity.	After submitting product details, they are submitted as intent data to the location activity.	**01:** Intent data with product details from the form are passed to Map Activity.	✓ Data accessed at MapActvity using Intent getData() method.	The product is added to the database at the MapActvity.

MAP ACTIVITY

TEST CASE	DESCRIPTION	EXPECTED OUTCOME	ACTUAL RESULTS	COMMENTS
11. Get user current location.	After user attempts to add product location a map is displayed with their current location.	**01:** Google marker must move to the current location of the user.	✓ After location request the marker moves to the current location of the user.	
12. Pin product location.	Product location is through a dragging the marker to the product location.	**01:** Map background service should capture the address, latitude, and address.	✓ When marker is dragged to pin a location. The latitude, longitude, and address of product is captured.	
13. Upload product to database.	Add a product to database with its details and the pinned location.	**01:** Products must be added to firebase while image must be uploaded to Firestore.	✓ Image and product details uploaded successfully to the database.	Images take some time to upload thus a progress message.

EDIT PRODUCT ACTIVITY

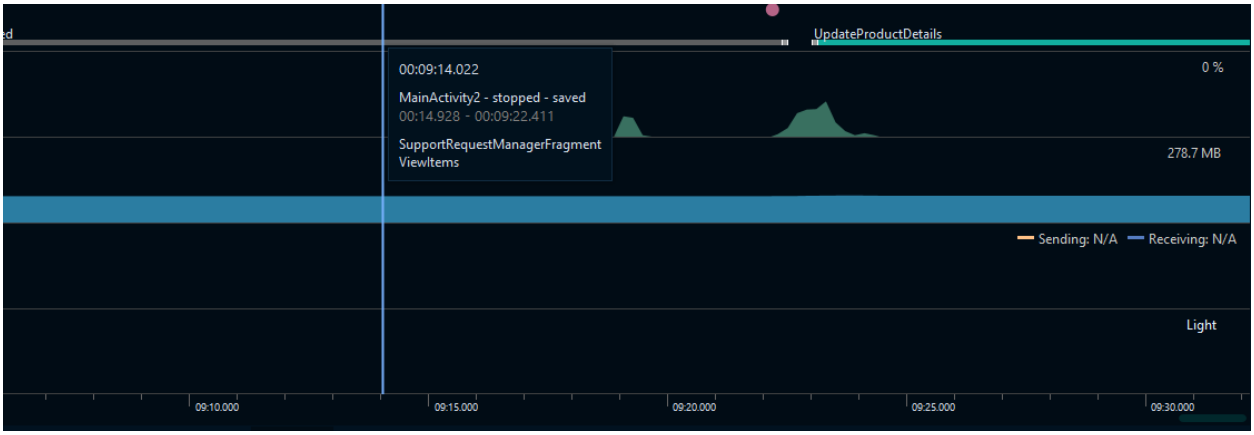
TEST CASE	DESCRIPTION	EXPECTED OUTCOME	ACTUAL RESULTS	COMMENTS
14. Product details Autofill.	When user clicks the edit button on the card item it must open the update product activity.	**01:** Update product page must have auto filled input fields with details of the product selected.	✓ Product details inputs auto filled according to the selected product to be edited.	
15. Edit item field.	On the auto filled input fields the user must be able to clear the fields and add new information desired.	**01:** Input fields must be able to be cleared and accept new user data.	✓ Input fields accepted new content and cleared current one.	
16. Update product in Firestore.	After submitting the edited product fields. They are submitted and uploaded to firebase.	**01:** Product must be updated in Firestore according to the details provided.	✓ Product details updated in the database.	
17. View item update on card view.	After update user is redirected to the products activity to view the update on the product.	**01:** User is should be redirected to product page. **02** Product details are visibly update.	✓ User redirected to product page to view successful update of the product. Information.	

PRODUCT ACTIVITY

TEST CASE	DESCRIPTION	EXPECTED OUTCOME	ACTUAL RESULTS	COMMENTS
18. Mark product as purchased.	When user clicks the purchased button, the product is removed from the list.	**01:** Onclick on the purchase button changes status of product to purchased. **02:** Product should then be removed from the list of items.	✓ Product status on firebase was modified to purchased and removed from the product list.	
19. Log purchased item on the dashboard.	Once a user marks an item as purchased. this is logged on the dashboard statistics card.	**01:** Items purchased numbers must increase on the dashboard card.	✓ Item purchased was as a count on the dashboard statistics count.	
20. Delete Item	Delete item button removes item from the list and records it in the counter found on the dashboard.	**01:** Item deleted is recorded in the Items deleted count card in the dashboard.	✓ Deleted item successfully counted and recorded on the dashboard analytics.	
21. Item Delegation	User clicks a share icon on a product to share it with his/her contacts on the messaging app.	**01:** When a user clicks the share button of a product an intent builder Is invoked to open the messaging app with an auto filled message about the product details.	✓ Intent builder opened the contacts of the user and auto filled a message containing the product details.	
22. Open product website.	When a user clicks the website URL of the product it must redirect them to a browser to open the page.	**01:** OnClickListner on the URL opens a browser and autofill's the URL bar with address and renders the page.	✓ Browser opened with the web page of the product page.	

APP PROFILING

Another test on the app is visualizing CPU and memory management. This test is important as it helps correct CPU consuming threads. Hence providing a faster and smoother user experience and preserving device battery life. When profiled the application consumed less than 30% CPU memory on each touch listener this is something than can be refined in future.



6.0 APPLICATION EVALUATION

The Do I Need It App was built using the android platform crafted using the Java Programming language. The App utilized a cloud storage – Firebase storage and a Google Map API for location functionalities.

APPLICATION SUCCESS

- ✓ Application was able to meet all core requirements of adding, managing, and inserting a status of deleted or purchased to an item. The application was able to add extra functionalities by uploading an image that goes with the product.
- ✓ The application was able to integrate the Google Maps API to fetch the current user location and allow the user to pin the location of product.
- ✓ The application was able to synchronize with the database at Realtime.

APPLICATION FAILURES

- × One of the major flaws with the application is when a user does not add the image of a product. This results in a null pointer exception that closes the app.
- × The Map does not have a search functionality instead of just pinning with a marker it would have been easier to search a location.

FUTURE IMPROVEMENTS

MAP API – One of the core future improvements of the app is to use the latest MAP SDK and API recently launched by Google. Off recent they discarded the places API used in our current application, hence the need to migrate to the latest API since it is recommended and comes with more advanced geo functionalities.

IMAGE EXCEPTIONS – Improvements on exception handling more especially on the image when a user does not upload an image. They must receive a message prompting them to fill all necessary fields, this is an improvement that can be made on the next application version.

OOP PRINCIPLES – The application needs to be refined in later versions to capitalize on OOP principles to cater for code reuse. This will be a great breakthrough for the app as it will contribute to its efficiency and memory management.

REFERENCE LIST

Chetty, R., Friedman, J.N., Hendren, N. and Stepner, M., 2020. How did covid-19 and stabilization policies affect spending and employment? a new real-time economic tracker based on private sector data (No. w27431). National Bureau of Economic Research.

Loxton, M., Truskett, R., Scarf, B., Sindone, L., Baldry, G. and Zhao, Y., 2020. Consumer behaviour during crises: Preliminary research on how coronavirus has manifested consumer panic buying, herd mentality, changing discretionary spending and the role of the media in influencing behaviour. Journal of Risk and Financial Management, 13(8), p.166.

King, A.H., 2019. Our elemental footprint. Nature materials, 18(5), pp.408-409.

Pandey, M., Litoriya, R. and Pandey, P., 2018. Mobile APP development based on agility function. Ingénierie des Systèmes d'Information, 23(6), p.19.

Sharma, S. and Hasteer, N., 2016, April. A comprehensive study on state of Scrum development. In 2016 International Conference on Computing, Communication and Automation (ICCCA) (pp. 867-872). IEEE.

Maradin, D. and Đipalo, E., 2020. The Market Structure Of The Smartphone Operating Systems Industry In The Eu.

Lazareska, L. and Jakimoski, K., 2017. Analysis of the Advantages and Disadvantages of Android and iOS Systems and Converting Applications from Android to iOS Platform and Vice Versa. American Journal of Software Engineering and Applications, 6(5), pp.116-120.