Last edited by **Denzil Correa** 11 months ago

Beta Prototype

BudgetWidget

Overview

We welcome to our fourth milestone blog; we are almost at the end of our application development process. Pretty much all the user stories were implemented into working prototypes. A software's success or failure is evaluated based on its ability to deliver the requirements, fulfilling the user's expectation and running without faults. Hence, we have reached a phase in the cycle where our application needs rigorous testing so as to provide a seamless and unparalleled experience to our customers. This blog would initially briefly be describing the software testing methods we chose and how we implemented them followed by the summary of changes and also find the apk. file attached below.

General Test Process

As far as our testing is concerned, we have followed Condition Coverage Technique and Branch Coverage Technique for the White Box testing and for Black Box testing, we have followed functional testing. Testing Process

Major testing techniques which are available at our disposal are the following:

• **Unit Testing**
- Testing is carried out by the developer after the completion of a module or functionality.

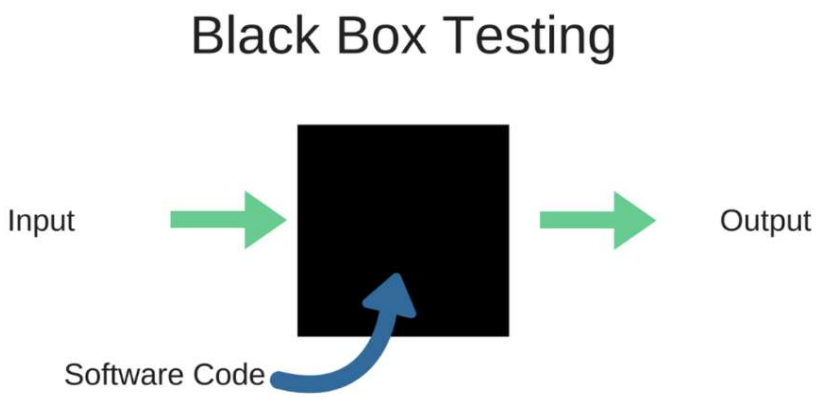
• **System Testing**
- Testing is carried out by a test analyst only for the module or functionality

• **System Integration Testing**
- Testing is carried out by a test analyst after all the modules are integrated.

• **Acceptance Testing**
- Testing is carried out by a user for the entire functionality of the application or module.

Black Box Testing

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on the input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.



The above Black-Box can be any software system you want to test. For Example, an operating system like Windows, a website like Google, a database like Oracle or even your own custom application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

Black Box Testing Types

• **Functional Testing:**
Usually performed by Test Analysts to check the functionality of the application based on the requirements received

• **Non-Functional Testing:**
This testing focuses on the performance, usability, and scalability of the application

• **Regression Testing:**
Testing done immediately after making any changes to the source code whether to know if all the earlier working functionalities are in place and has no impact on any of the functionalities done earlier.

• **Equivalence Partitioning:**
It is a test design technique that involves dividing input valid and invalid partitions and selecting particular values from each partition as test data.

• **Boundary Value Analysis:**
It is a test design technique that involves the determination of boundaries for input values and selecting values that are at the boundaries and just inside/outside of the boundaries as test data.

• **Cause-Effect Graphing:**
It is a test design technique that involves identifying the cause and effects producing a Cause-Effect Graph.

These characteristics define how a system works and when to stop the testing.

So using these test cases derived from use case scenarios. We will describe a few use case scenarios to define how we used black-box testing in our Software Development Life Cycle. Below are the five uses cases which were tested by users

Test cases	Scenario	Expected Output	Inserted Data 1	Actual output	Inserted Data 2	Actual output
Test sign up page	Enter special characters in username	Enter valid username	sachin_*#gmail.com	Signed up with the given username		
	Enter password less than 6 characters	Password should be atleast 6 characters	ten	Password should be atleast 6 characters	tendul	Signed up
	Enter username and leave password blank	password is required		password is required		
	Enter password and leave username blank	username is required		Signed up with no username		
	Give name as numbers in the next page	Name cannot be numbers	12345	signed up		
	Give name and enter a long digit date of birth	Date of birth is invalid	19343458345	signed up		
	Give no input to name and dob	Please enter name and dob	NA	signed up		
Sign in page	Enter both username and password as capital letter	Please check the password		password is invalid		
	Enter username and leave password as empty	password is required		Password is required		
	Enter password and leave username empty	please enter username		App crashes		
	Enter wrong password	password is invalid		password is invalid		
Add expense	Enter all the null values	please fill the required fields(Amount and Notes)	null	Please fill the required fields		
	Entered all the values	Data is inserted	Item=Entertainment Amount=20 Notes = Movie	Data inserted and notified with expense item added		
	Enter the required data and add a contact	Contact should be selected and added	selected a contact	Contact is added		
	Enter the required data and select recurring expense	Data is added as recurring expense	selected recurring expense	Data is added as recurring expense		
	Enter the expense with the a high value 1000000000	please enter the right amount	1000000000	App is crashing sometimes, but it is taking the high value		
Edit expense	Add and expense first and then update the entry	Data should be updated	updated amount from 20 to 40	Entry is updated		
	Add and expense and then delete the expense	Entry should be deleted		Entry is deleted		
Export the expense	Export the data with any option	Data should be exported as csv file		Csv file is created and shared		
Select currency	Select currency of payment	should be able to select a currency	clicked the button to change currenc	button not working		
			Entertainment =10			
Analytics	Enter the expense in a category	It should reflect in analytics	charity = 10	Analytics is updated with the entered expenses		
	Enter the high expense in a category	It should show red indicating a warning	transport=40	showing infinity used of 0.0 status for individual category		
	Enter the expenses in every category	Analytics should show each category	gave expense in each category	house expense not showing		

Yellow is low, Orange is medium and Red as a high priority

White box testing

The white box comprises Unit testing and other techniques such as System Testing and Acceptance Testing, Integration Testing is classified under Black Box Testing. White Box Testing

White Box Testing is a software testing technique in which internal structure, design and coding of software are tested to verify the flow of input-output and to improve design, usability and security. In white-box testing, code is visible to testers so it is also called Clear box testing, Open box testing, transparent box testing, Code-based testing and Glass box testing.

It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user type perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

The term **"WhiteBox"** was used because of the see-through box concept. The clear box or WhiteBox name symbolizes the ability to see through the software's outer shell (*or "box"*) into its inner workings. Likewise, the "black box" in "Black Box Testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested.

From in developers point of view, they can follow the below two simple steps:

• First to get the complete working knowledge of the written code.

• To select the most suitable coverage technique, that is applicable to the written code. White Box Testing Techniques

There exist many coverage techniques for the developers to scrutinize blocks of written codes. They are:

• Statement Coverage
This technique requires every possible statement in the code to be tested at least once during the testing process of software engineering functions coverage -Tests each function at least once.

• Edge coverage
-Tests every edge in the control flow graph in the program at least once

• Branch Coverage
- This technique checks every possible path (if-else and other conditional loops) of a software application.

• Condition Coverage
Or expression coverage is a testing method used to test and evaluate the variables or sub-expressions in the conditional statement. The goal of condition coverage is to check individual outcomes for each logical condition. Condition coverage offers better sensitivity to the control flow than decision coverage. In this coverage, expressions with logical operands are only considered.

For example, if an expression has Boolean operations like AND, OR, XOR, which indicates total possibilities.

User Authentication on the login
The following code is from the login activity. In this, it will check the user credentials like email and password.

Scenario 1

When valid details are given:
At line 90 and 93 the code checks whether an email and password are provided by the user or not. If the user has provided the user name and password then the code block from 97-126 is executed.

Scenario 2

When the user leaves email or password empty
In this case, the line 90 and 93 will check for the credentials and if they are empty this will lead to the execution of line 91 and line 94.

```
            if (TextUtils.isEmpty(emailString)) {
                email.setError("Email is required");
            }
            else if (TextUtils.isEmpty(passwordString)) {
                password.setError("Password is required");
            }

            else {

                progressDialog.setMessage("login in progress");
                progressDialog.setCanceledOnTouchOutside(false);
                progressDialog.show();

                mAuth.signInWithEmailAndPassword(emailString, passwordString).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {...}
                });
            }
```

Scenario 1:

User valid login into application

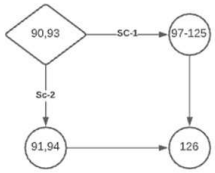
Path: 90,93,97-125,126

Scenario 2:

User login empty

Empty username or password

Path: 90-95,126



Auto-login

When a user exits the app without logging out or when the user exits the app after logging out this particular code is executed.

Scenario 1

User already signed in when the user exits the app without logging out and opens the app again this code executes and logs the user into the app and moves him into the main activity view. In this case, line 49 checks if a user is already logged in to the app or not. If a user is already logged in then the code from 50-51 will be executed and the user is automatically logged into the app and moved to the main activity view.

Scenario 2

The user signed out of the application, in this case, line 49 check for a user and as there is no user this condition will fail and the code from 54-125 is executed where the user is asked to log in again into the app.

```
    if (mAuth.getCurrentUser()!=null) {
        startActivity(new Intent( packageContext: LoginActivity.this, MainActivity.class));
    }


    authStateListener = new FirebaseAuth.AuthStateListener() {...};

    email = findViewById(R.id.email);
    password = findViewById(R.id.password);
    loginBtn = findViewById(R.id.loginBtn);
    loginQn = findViewById(R.id.loginQn);


    progressDialog = new ProgressDialog( context: this);


    loginQn.setOnClickListener(new View.OnClickListener() {...});


    loginBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String emailString = email.getText().toString();
            String passwordString = password.getText().toString();

            if (TextUtils.isEmpty(emailString)) {
                email.setError("Email is required");
            }
            else if (TextUtils.isEmpty(passwordString)) {
                password.setError("Password is required");
            }

            else {

                progressDialog.setMessage("login in progress");
                progressDialog.setCanceledOnTouchOutside(false);
                progressDialog.show();

                mAuth.signInWithEmailAndPassword(emailString, passwordString).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {...}
                });
            }
```

Scenario 1:

Already signed in.

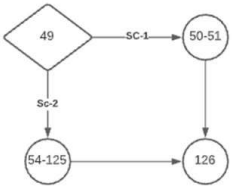
User already signed into the application.

Path: 49,50-51

Scenario 2:

Logging again into the application after logout.

Path49,54-125



Setting budget to any category

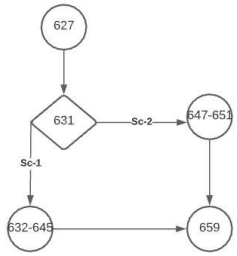
Scenario 1

When the user wants to set a budget for the Food category (similar code for all other categories)
When a user is updating the budget and selects a category and enter the amount for that particular category this particular part of the code is executed. When a change has been made to the budget the condition on line 631 executes and checks for a change in the program. Then the code block from 632 till 637 is executed and the amount is updated for that particular category and from line 639 to 646 the amount is then divided so that it can be set for weekly and daily and monthly budget limits.

Scenario 2

When the user has not set any budget for the Food category(similar code for all other categories)
When there is no budget set, the condition at line 631 will be checked but it will be false so the code block from line 647-651 (else part of the condition) executes and the amount is set "zero".

```
627     Query query = budgetRef.orderByChild("item").equalTo("Food");
628     query.addValueEventListener(new ValueEventListener() {
629         @Override
630         public void onDataChange(@NonNull DataSnapshot snapshot) {
631             if (snapshot.exists()) {
632                 int pTotal = 0;
633                 for (DataSnapshot ds : snapshot.getChildren()) {
634                     Map<String, Object> map = (Map<String, Object>) ds.getValue();
635                     Object total = map.get("amount");
636                     pTotal = Integer.parseInt(String.valueOf(total));
637                 }
638
639                 int dayFoodRatio = pTotal / 30;
640                 int weekFoodRatio = pTotal / 4;
641                 int monthFoodRatio = pTotal;
642
643                 personalRef.child("dayFoodRatio").setValue(dayFoodRatio);
644                 personalRef.child("weekFoodRatio").setValue(weekFoodRatio);
645                 personalRef.child("monthFoodRatio").setValue(monthFoodRatio);
646
647             } else {
648                 personalRef.child("dayFoodRatio").setValue(0);
649                 personalRef.child("weekFoodRatio").setValue(0);
650                 personalRef.child("monthFoodRatio").setValue(0);
651             }
652         }
653
654         @Override
655         public void onCancelled(@NonNull DatabaseError error) {
656
657         }
658     });
659 }
```



Scenario 1:

Setting budget for a category

627,631,632-645

Scenario 2:

No budget set for a category

627,631,647-651

Currency conversion

When the customer enters his daily spending and selects a particular currency of payment and had a different default currency this particular segment of the code is executed.

Scenario 1

Default currency: Euro; Payment is done in Dollar, Pound, INR
On line 299 The default currency is obtained than on line 301 the default currency is checked and when the currency is Euro the code block from 302 till 316 is executed. This converts the amount which is entered in a different currency to Euro and is saved into the database.

Scenario 2

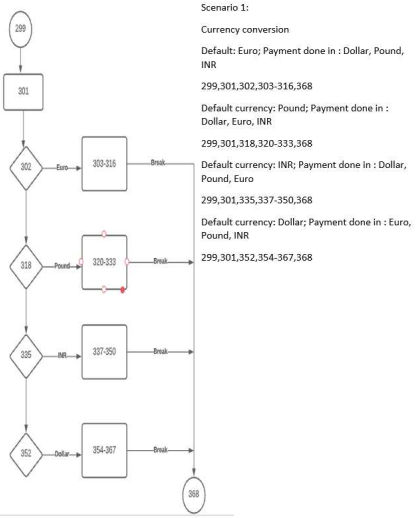
Default currency: Pound; Payment is done in Dollar, Euro, INR
On line 299 The default currency is obtained than on line 301 the default currency is checked and when the currency is Pound the code block from 318 till 333 is executed. This converts the amount which is entered in a different currency to Pound and is saved into the database.

Scenario 3

Default currency: INR; Payment is done in Dollar, Pound, Euro
On line 299 The default currency is obtained than on line 301 the default currency is checked and when the currency is INR the code block from 335 till 350 is executed. This converts the amount which is entered in a different currency to INR and is saved into the database.

Scenario 4

Default currency: Dollar; Payment is done in: Euro, Pound, INR
On line 299 The default currency is obtained than on line 301 the default currency is checked and when the currency is Dollar the code block from 352 till 362 is executed. This converts the amount which is entered in a different currency to a Dollar and is saved into the database.



Scenario 1:

Currency conversion

Default: Euro; Payment done in : Dollar, Pound, INR

299,301,302,303-316,368

Default currency: Pound; Payment done in : Dollar, Euro, INR

299,301,318,320-333,368

Default currency: INR; Payment done in : Dollar, Pound, Euro

299,301,335,337-350,368

Default currency: Dollar; Payment done in : Euro, Pound, INR

299,301,352,354-367,368

Summary of Change

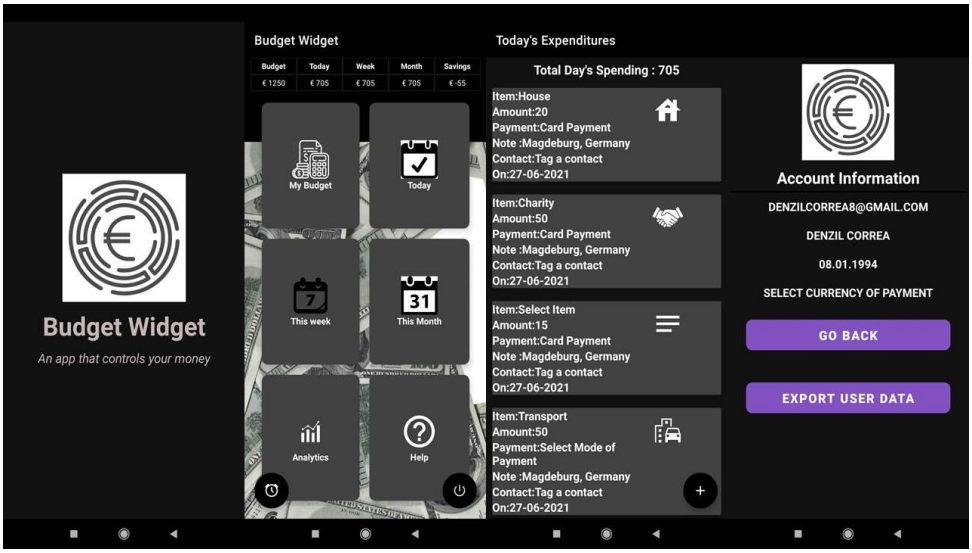
Updated User Stories

USER STORIES
As a User, *I want to export all my transaction data.*
As a User, *I want to want to have a recurring transaction for weekly, monthly.*
As a User, *I want to get a notification.*
As a User, *I want to have a detailed help menu on analytics.*

You can find and download the .apk file using this link. --> .apk

App Preview

Our final content, App preview to show an overview of the app experience, focusing on the app's final developed features aiming to tell a cohesive story that gives users a sense of the journey they'll experience when using our App.

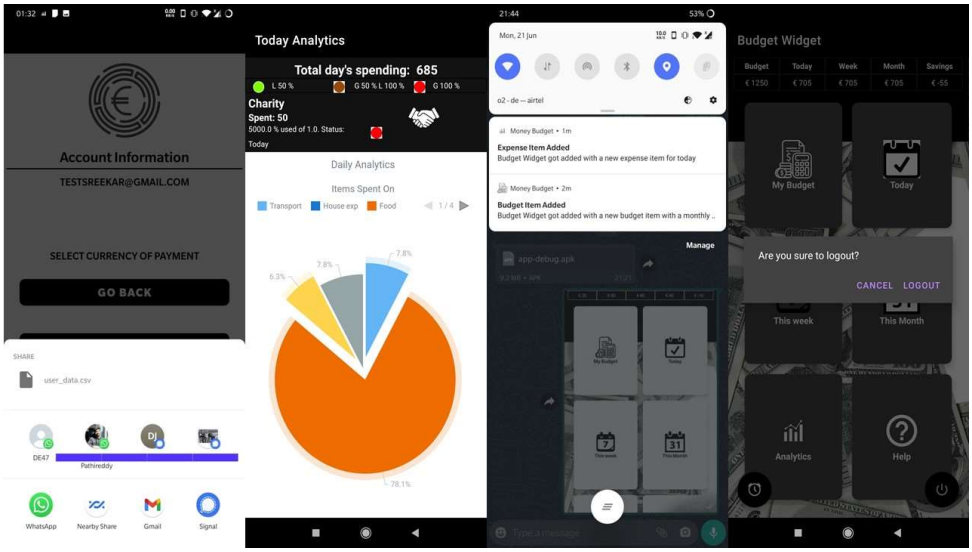


When the user opens the application, they are presented with a splash screen page, (fig 1.1) where the user can see to app moto -
"An App that controls your money"

Once the user logins the app takes them to the main page, (fig 1.2) where several features are shown such as My Budget, Today, This week, etc. along with and a quick overview on the tab above describing the amount that is been spent with a margin titled budget.

On clicking Today, the user can add expenses (fig 1.3) quoting item, currency, amount, mode of payment along with a not an option to add it as a recurring expense. Once clicked the data will be stored in the expense which could be edited or deleted further.

Under the bottom left user info float button the user can select also to export data of his/her transaction as an external file. (fig 1.4)



The user can also choose to send the exported data by mail or any other social media application and further view the transcript in an excel format .csv (fig 2.1) and save it.

(fig 2.2) Shows the graphical representation of Monthly and weekly analytics, with a green, brown, and red flag as a ticker to indicate expenses against the budget threshold along with a percentage of each expense category.

It's crucial for every app to have a notification feature these days to have some sought of the remainder and therefore (fig 2.3) gives our user notifications to make the better use of our application.

Lastly, a Small pop-up appears when pressed the log out floating button (fig 2.4) - where the user can confirm to log out or cancel if not.

Hope you had a nice time reading through our blog. We like to continue to keep our readers interested, so do visit us back for our upcoming final and last blog on entering the play store.

Home Page