Project Design Document
NotifyMe

Group 10
Deepanker Mishra - 2013CS50282,
Garvit Jain - 2013CS50284,
Shivanshu Gupta - 2013CS50298

April 17, 2017

1. Introduction

1.1 Purpose

The purpose of this document is to present an innovative android application, NotifyMe, which takes the burden of sifting through all notifications - like discounts, cashbacks, etc. - away from the user. This document will explain the features and interfaces of the app, what the app does and the constraints under which it must operate. It will provide the software requirement specification for the application NotifyMe.

This project is developed as a course project but is useful to anyone who uses a smartphone these days.

1.2 Scope

This project will consist of developing an Android application which will have permissions to read incoming SMS/notifications. It will then employ text processing to extract various characteristics related to offer such as:

- source - which application did it originate from?
- category - what the offer is for? eg. Food, movies etc.
- discount - how much discount/cashback if any is being offered?
- validity - for how long is the offer valid?
- code - coupon code if any?
- contact info if any?
- and more.

The information is stored in a database and periodically updated as new offers come and old ones expire. This information will be used to let the user while in-app, view the best and most recent offers from any category.

1.3 Overview

The next section, the System Overview section, of this document gives a high level overview of the functionality of different subsystems and their interaction with the system.
The third section, i.e. the Detailed Design section, of this document is written primarily to give a detailed description of the working of the system.
The last section, Deployment Design, gives information about the deployment structure of the application.

1.4 References

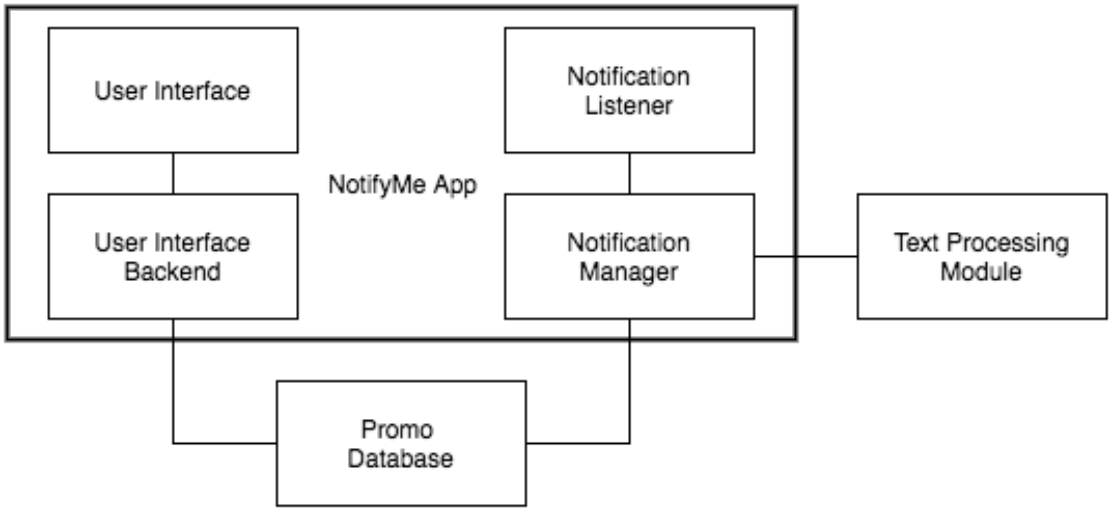
- IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

1.5 Definitions, acronyms and abbreviations

- **User:** Someone who interacts with the mobile phone application
- **App:** is abbreviation for Application and may occur in this document at many places.
- **SMS:** Short Message Service, which is the medium most companies use these days to update their customers on recent coupons, discounts and offers.
- **DB:** stands for Database of the application.
- **Promo:** is any promotional offer received via SMS or as a notification from an e-commerce application.
- **Vendor:** is the source of the promotional offer. In most cases it will be an e-commerce application.
- **ERD:** stands for Enterprise Relationship Diagram.

2. System Overview

2.1 Architectural Design



2.2 Module Definitions

2.2.1 Notification Listener Module

The Notification Listener Module listens for Notifications from other applications and also for SMSes. It checks whether the source of the Notification/SMS is from a supported vendor and passes the Promos to the Notification Manager module. If a new vendor is found which is not present in the database, it is added in the database of supported vendors.

2.2.2 Text Processing Module

The Text Processing Module processes the messages received from the vendors through the Notification Listener API. The text message is parsed to identify whether the message is a promo or not, and consequently used to extract information about the category, discount, validity and other relevant details for a promo. This information is used to score the message and subsequently saved in the database.

2.2.3 Notification Manager Module

The Notification Manager Module receives promo from the Notification Listener Module. It uses the Text Processing Module to extract information about the Promo message. It then applies a scoring function to the Promo attributes to calculate a score for the Promo. The Promo along with its extracted attributes and score is stored in the database. Additionally, if the promo score is high enough, user is notified about it.

2.2.4 User Interface Module

The User Interface Module mainly handles the user's interaction with the application. The User Interface Module has two parts - the front-end and the back-end. The front-end directly interacts with the user's gestures on the screen and manages the display on the screen. It provides the the user with functionalities such as:

1. view list of promos of a particular category
2. filter promos by vendor, date of receipt and validity
3. order promos by score, date of receipt, validity
4. view detailed information about a promo
5. delete selected promos

The back-end performs the required actions based on the user input from the screen.

2.2.5 Database Module

The Database Module provides APIs to store the Promos, there attributes and score in a SQLite database. The database also contains information about the vendors supported. The User Interface uses the Database Module to retrieve promo information according to the user's needs. Whenever the user applies a filter, the corresponding query is run on the database and the information about the relevant promos is retrieved. In case a promo is received from a vendor not present in the database, that vendor is added to the database.

2.3 Technology/ Tools Used

We have used Android Lollipop version 5.0 minimum to run our application. The database is designed using SQLite.

3. Detailed Design

3.1 Module APIs

This section covers all the APIs of different modules used to build the software.

3.1.1 Text Processing Module

API Function	Description
isPromo(String[] msg)	Checks whether the message is a promo or not
getCategory(String[] msg)	Identifies the category - food, travel, etc. - of the message
getDiscountAmount(String[] msg)	Gives the discount/cashback amount in the promo
getDiscountPercent(String[] msg)	Gives the discount percentage in the promo
getFreebie(String[] msg)	Gives information about the "Buy x Get y" type of promo
getCode(String[] msg, String[] org)	Returns the coupon code in the promo, if any
getValidity(String[] msg)	Gives the date till when the promo can be used
getMaxUses(String[] msg)	Returns the number of times the promo can be availed
parsePromo()	Uses above functions to create a Promo object

3.1.2 Database Module

API Function	Description
storePromo(Promo p)	Stores the given promo.
retrievePromos(Filter filter)	Retrieves promos from the database based on the given filters.
retrieveAllPromos()	Retrieves all promos from the database.
deletePromo(Promo p)	Delete specified promo from database

3.2 Database Design

The application database contains 2 tables. The Promos Table stores Promo Text, parsed promo attributes and Promo score. The Vendor Table lists the vendors supported by the application and the vendor applications installed in the user device.

3.2.1 Promos Table

Field	Type	Description
promoID	Integer	Primary Key
category	Text	Promo Category, Not Null
vendor	Text	Promo Category, Not Null
receivedOn	Datetime	Promo receive date, Not null
discountPercent	Decimal	Discount percent
discountByAmount	Decimal	Discount by amount
discountToAmount	Decimal	Discount to amount
freebies	Text	Freebies
code	text	Promo Code
expiryDate	Datetime	Promo expiry date
maxUses	Integer	max number of times the promo can be used
promoScore	Decimal	promo score
promoMsg	Text	original Promo Text

3.2.2 Vendor Table

Field	Type	Description
vendorName	Text	Name of the vendor, Primary Key
appInstalled	Boolean	App is installed on the user device

3.3 Screen Layouts

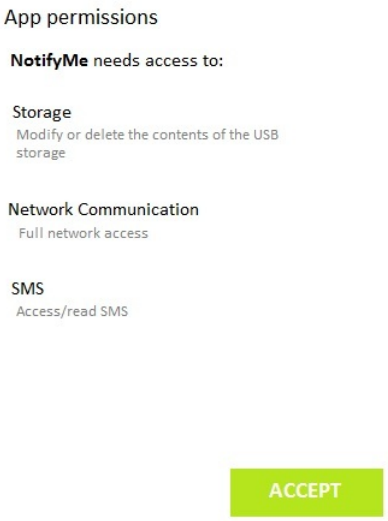


Figure 1: Permissions UI

Figure 2: Home Screen with Categories

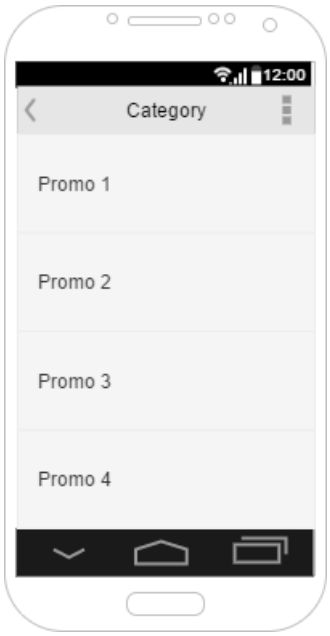


Figure 3: List of Promos in a Category

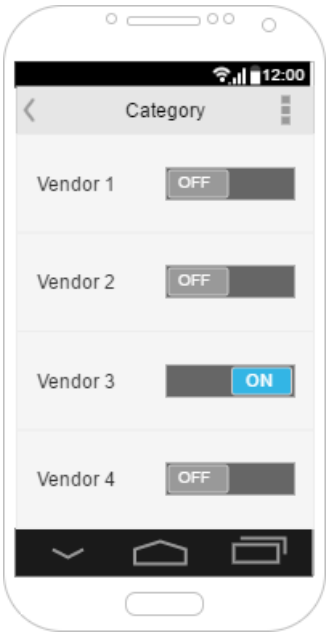


Figure 4: Filters for vendors

3.4 Use Cases

3.4.1 Use Case 1

- **Brief description -** This use case describes how the user uses the NotifyMe app to view all promos.
- **Actors -** App User
- **Preconditions -** User has given permissions to the app.
- **Basic flow of events**
  1. The user opens the NotifyMe app.
  2. App displays the categories on the screen.
  3. User selects the 'All' category to view the promos displayed on the screen.
- **Alternative flows**
  1. *No offer present*- If there are no offers in the database, the app displays an error message.
- **Post-conditions**
  a *Successful operation* - User can view all the offers at once.
  b *Failed operation* - The system state remains unchanged.
- **Special Requirements -** None

3.4.2 High level Code (using module APIs)

```
User presses All Button
⇒ viewAllPromos()

viewAllPromo(){
    promoList = retrieveAllPromo()
    displayPromos(promoList)
}
retrieveAllPromo(){
    promoList = query(db, PROMO_TABLE)
    return promoList
}
```

3.4.3 Sequence Diagram

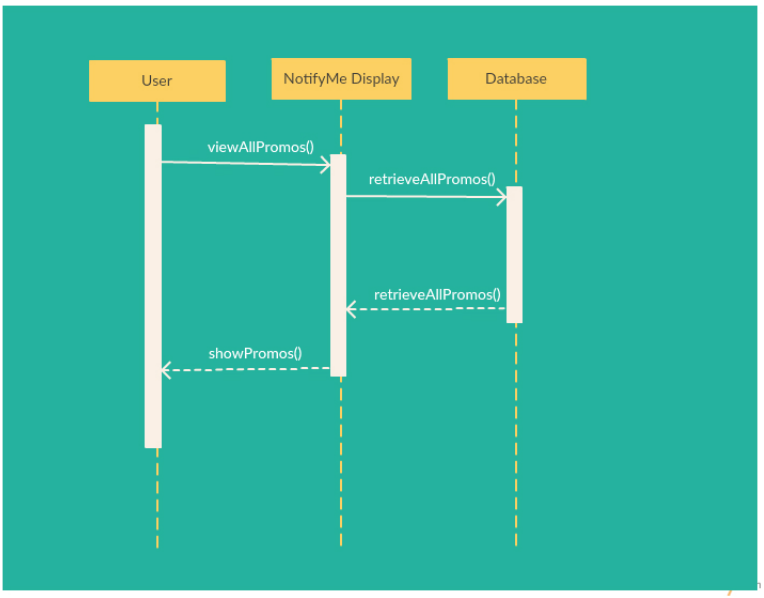


Figure 5: Sequence Diagram for use case 1

3.4.4 Use Case 2

- **Brief description -** This use case describes how the user uses the NotifyMe app to filter notifications by category.
- **Actors -** App User
- **Preconditions -** User has given permissions to the app.
- **Basic flow of events**
  1. The user opens the NotifyMe app.
  2. App displays the categories on the screen.
  3. User selects a category of his choice from the circular menu.
- **Alternative flows**
  1. *No such offer* - If no offer meets the filter requirements, or there are no offers in the databse, the app displays a blank screen.
- **Post-conditions**
  a *Successful operation* - User gets information about his offer of interest.
  b *Failed operation* - The system state remains unchanged.
- **Special Requirements -** None

3.4.5 High level Code

```
category = User chooses a category from Home
promoList = retrievePromo(Filter{Category:category});
```

```
displayPromos(promoList)

retrievePromo(filter){
    promoList = query(db, PROMO_TABLE, filter)
    return promoList
}
```

3.4.6 Sequence Diagram

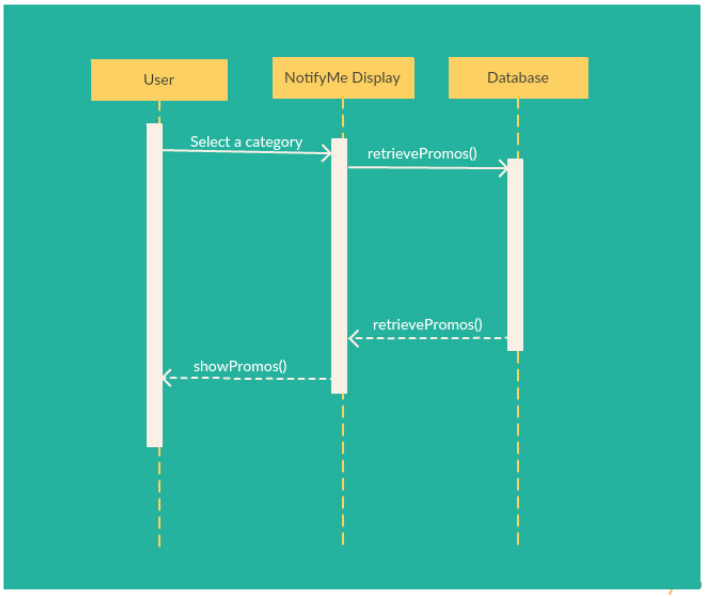


Figure 6: Sequence Diagram for use case 2

3.4.7 Use Case 3

- **Brief description -** This use case describes how the user uses the NotifyMe app to filter notifications by vendor.
- **Actors -** App User
- **Preconditions -** User has given permissions to the app.
- **Basic flow of events**
  1. The user opens the NotifyMe app.
  2. App displays the categories on the screen.
  3. User selects a category.
  4. User specifies to filter the promos according to the vendor.
  5. The filter is applied on the database of offers and the required offers are displayed.
  6. User notes the coupon code and other relevant information about his offer of interest.

- **Alternative flows**
  1. *No such offer* - If no offer meets the filter requirements, or there are no offers in the databse, the app displays a blank screen.
- **Post-conditions**
  a *Successful operation* - User gets information about his offer of interest.
  b *Failed operation* - The system state remains unchanged.
- **Special Requirements -** None

3.4.8 High level Code

```
current_filter = currently applied filter
vendor = User chooses vendors
promoList = retrievePromo(filter{current_filter, Vendor:vendor})
displayPromos(promoList)

retrievePromo(filter){
    promoList = query(db, PROMO_TABLE, filter)
    return promoList
}
```

3.4.9 Sequence Diagram

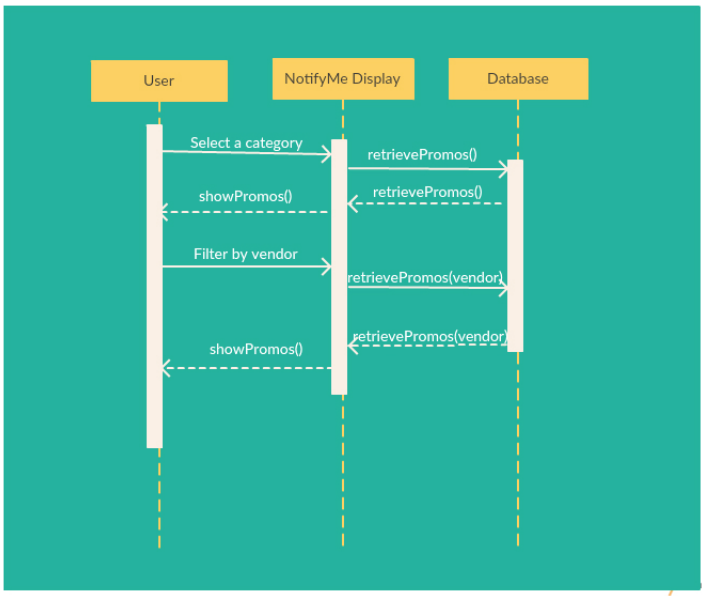


Figure 7: Sequence Diagram for use case 3

3.4.10 Use Case 4

- **Brief description -** This use case describes how the user uses the NotifyMe app to filter out notifications received before a given date of receipt.
- **Actors -** App User
- **Preconditions -** User has given permissions to the app.
- **Basic flow of events**
  1. The user opens the NotifyMe app.
  2. App displays the categories on the screen.
  3. User selects a category.
  4. User specifies to filter the promos by receive date
  5. The filter is applied on the database of offers and the required offers are displayed.
  6. User notes the coupon code and other relevant information about his offer of interest.
- **Alternative flows**
  1. *No such offer* - If no offer meets the filter requirements, or there are no offers in the databse, the app displays a blank screen.
- **Post-conditions**
  a *Successful operation* - User gets information about his offer of interest.
  b *Failed operation* - The system state remains unchanged.
- **Special Requirements -** None

3.4.11 High level Code

```
current_filter = currently applied filter
receiveDate = User chooses date of receipt
promoList = retrievePromo(filter{current_filter, ReceiveDate:receiveDate})
displayPromos(promoList)

retrievePromo(filter){
    promoList = query(db, PROMO_TABLE, filter)
    return promoList
}
```

3.4.12 Sequence Diagram

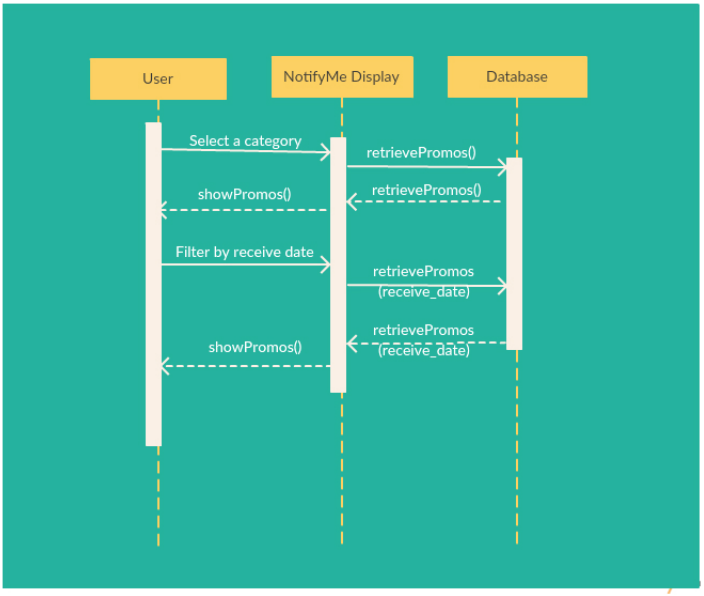


Figure 8: Sequence Diagram for use case 4

3.4.13 Use Case 5

- **Brief description -** This use case describes how the user uses the NotifyMe app to filter notifications by expiry date.
- **Actors -** App User
- **Preconditions -** User has given permissions to the app.
- **Basic flow of events**
  1. The user opens the NotifyMe app.
  2. App displays the categories on the screen.
  3. User selects a category.
  4. User specifies to filter the promos according to their validity.
  5. The filter is applied on the database of offers and the required offers are displayed.
  6. User notes the coupon code and other relevant information about his offer of interest.
- **Alternative flows**
  1. *No such offer* - If no offer meets the filter requirements, or there are no offers in the databse, the app displays a blank screen.
- **Post-conditions**
  a *Successful operation* - User gets information about his offer of interest.
  b *Failed operation* - The system state remains unchanged.
- **Special Requirements -** None

3.4.14 High level Code

```
current_filter = currently applied filter
expiryDate = User chooses date of expiry
promoList = retrievePromo(filter{current_filter, ExpiryDate:expiryDate})
showPromos(promoList)

retrievePromo(filter){
    promoList = query(db, PROMO_TABLE, filter)
    return promoList
}
```

3.4.15 Sequence Diagram

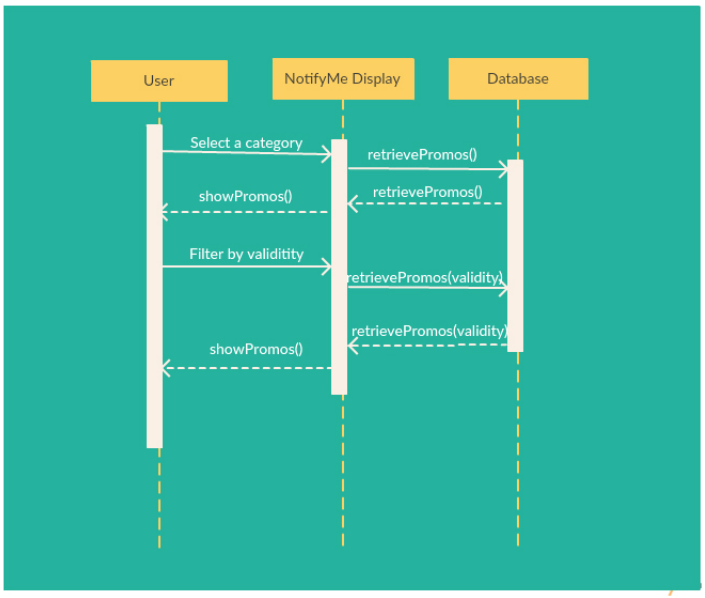


Figure 9: Sequence Diagram for use case 5

3.4.16 Use Case 6

- **Brief description -** This use case describes how the user uses the NotifyMe app to hide irrelevant notifications.
- **Actors -** App User
- **Preconditions -** User has given permissions to the app.
- **Basic flow of events**

1. The user opens the NotifyMe app.
2. App displays the categories on the screen.
3. User opens the 'Settings' menu in the app.
4. User can select/deselect the 'Hide irrelevant notifications' checkbox to hide those notifications which have a score less than the user specified score. The score is specified in the 'Score threshold' textbox.

- **Alternative flows** - None
- **Post-conditions**

  a *Successful operation* - Notifications will be hidden from the user.

  b *Failed operation* - The system state remains unchanged.

- **Special Requirements -** None

3.4.17 High level Code

```
User opens settings
⇒ UI settings are displayed
⇒ hide notifications{
    toggles sets notification boolean
}
```

3.4.18 Sequence Diagram

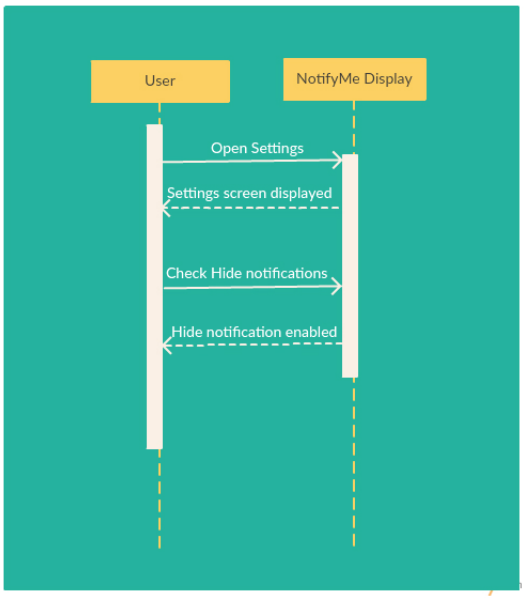


Figure 10: Sequence Diagram for use case 6

3.4.19 Use Case 7

- **Brief description -** This use case describes how the user uses the NotifyMe app to select apps for which offers will be displayed
- **Actors -** App User
- **Preconditions -** User has given permissions to the app.
- **Basic flow of events**
  1. The user opens the NotifyMe app.
  2. App displays the categories on the screen.
  3. User opens the 'Settings' menu in the app.
  4. User can select/deselect the apps for which offers will be covered.
- **Alternative flows** - None
- **Post-conditions**
  a *Successful operation* - Only selected apps' offers will be displayed.
  b *Failed operation* - The system state remains unchanged.
- **Special Requirements -** None

3.4.20 High level Code

```
User opens settings ⇒
UI settings are displayed
User chooses relevant apps
Relevant app list is updated in the database
```

3.4.21 Sequence Diagram

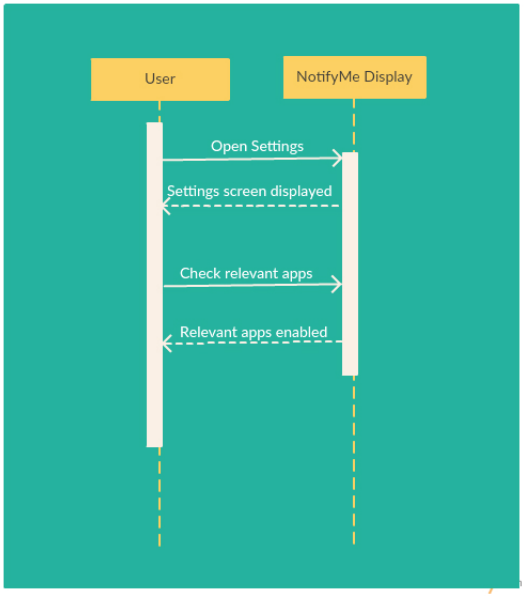


Figure 11: Sequence Diagram for use case 7

3.4.22 Use Case 8

- **Brief description -** This use case describes how the user uses the NotifyMe app to order promos according to score.
- **Actors -** App User
- **Preconditions -** User has given permissions to the app.
- **Basic flow of events**
  1. The user opens the NotifyMe app.
  2. App displays the categories on the screen.
  3. User selects a category.

4. User can select the 'order by' checkboxes to view the offers in ascending or descending order of their score.

- **Alternative flows**

  1. *No offer present*- If there are no offers in the database, the app displays a blank screen.

- **Post-conditions**

  a *Successful operation* - User can view the promos ordered according to the score.

  b *Failed operation* - The system state remains unchanged.

- **Special Requirements -** None

3.4.23 High level Code

```
current_filter = currently applied filter promoList =
current_order = user chooses order
retrievePromo(filter{current_filter,SortingOrder = current_order })
showPromos(promoList)
```

3.4.24 Sequence Diagram

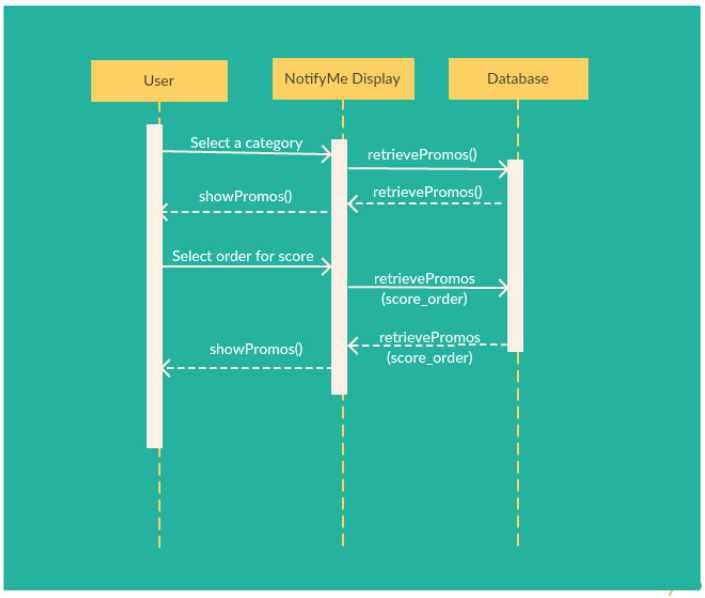


Figure 12: Sequence Diagram for use case 8

4. Deployment Design

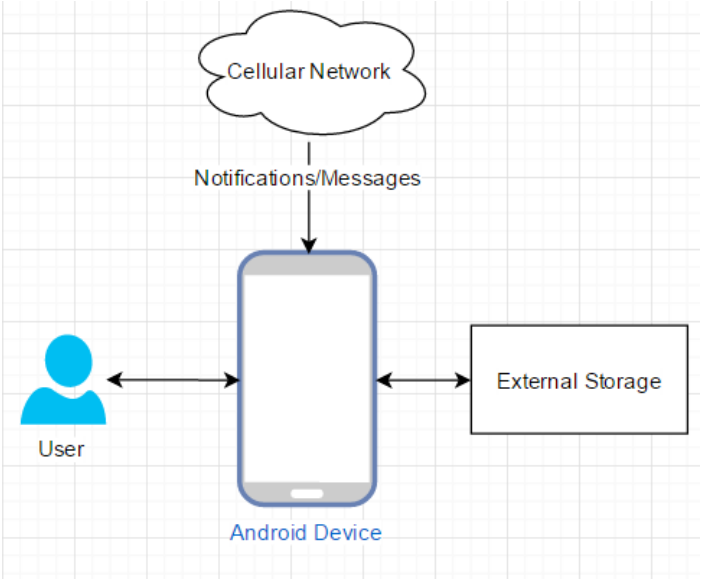


Figure 13: Deployment Diagram for the NotifyMe Application

A user opens the NotifyMe application on his/her Android device. The Android device receives messages/notifications through the cellular network which are processed by the Text Processing Module and stored in a database in its external storage. Whenever the user wishes to view particular promos by applying filters, the NotifyMe application interacts with the external storage of the device to fetch information about the relevant promos which is displayed on the screen as per the user's convenience.