TRIP MANAGEMENT

Android Project Report

- Supervised by

Prof. Frederic Mallet

ABSTRACT

To understand the requirement of creating a Trip Management App based on Android Framework and able to create a multiservice trip with ability to share it, with functions to calculate cost, edit trip and service as well as share it keeping up the consistency.

Submitted by:

Cyril Naves

&

Simbarashe Kanengoni

Task Split and Assignment:

Action of the member in the team with respect to functionality are presented in the below table:

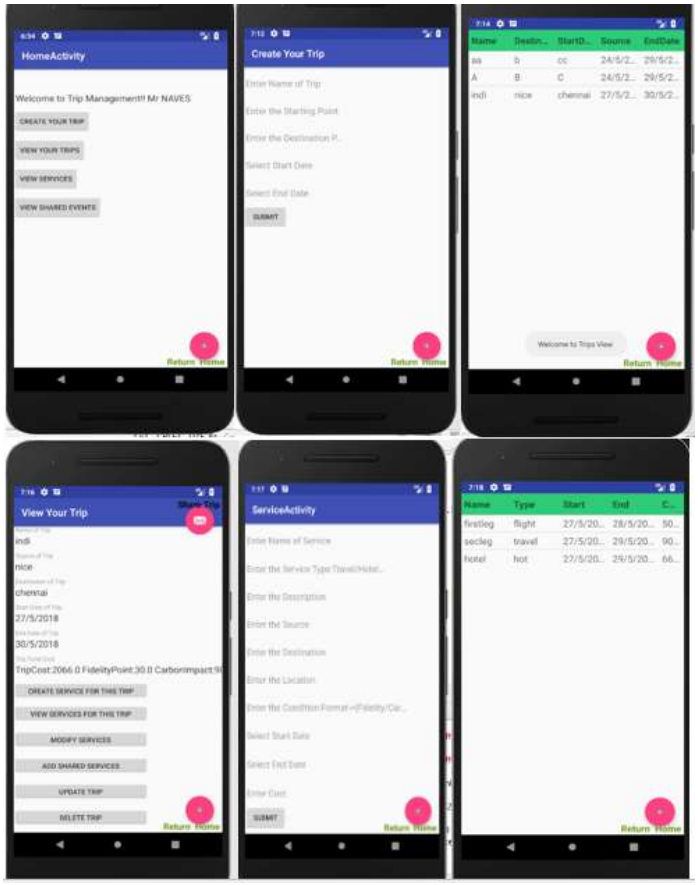
Sub Function	Assigned To	Effort Weightage %	Completion %	Hours Efforts
Requirement Analysis	Cyril & Simba	5	100	2
Design	Cyril	5	100	2
Database design & Implementation	Cyril & Simba	10	100	4
Trip CRUD Operations	Cyril	5	100	6
Service CRUD Operations	Cyril	5	100	2
Shared Trip & Services Exchange &	Cyril	5	100	2
Standalone Shared Service Exchange &	Simba	5	100	2
Cost Calculation	Cyril	10	100	6
Exchange Protocol	Cyril	15	100	24
Consistency and Validation Check	Cyril	15	90	24
UI Improvement	Cyril	5	100	12
Testing & Bug fixes	Simba	5	100	12
Additional Functionality: login	Cyril	5	100	4
Documentation	Cyril & Simba	5	100	3

Functionality Overview & Coverage:

1)**Edit a Trip:**
Coverage: 100%

Sub Function	Create	Read	Update	Delete	Import
Trip	Yes	Yes	Yes	Yes	NA
Service	Yes	Yes	Yes	Yes	NA
Shared Trip & Services	NA	Yes	NA	Yes	Yes
Standalone Shared Service	NA	Yes	NA	Yes	Yes

Crud Operation of Trip & Service Sample Screenshot due to space constraint:



Overview:

1) Each Create Read,Update,Delete of Trip and Service Operations were covered based on the dynamic retrieval of data from the UI and persisting through DAO which were similar for all screens
2)Each Trip contained a list of services which had to be handled.
3)Presentation of the Services and Trips were loaded using TableView Adapter for better UI
4)View and Edit Trip Function was core as it handled many operations like viewing its services,adding services, adding shared services
5) Import of Shared Data was done separately through a separate table and then user has the flexibility to import or not, a trip or service
6) Requirement was achieved since it was more handling the intents and passing data between UI and Dao.

2) Check or Guarantee Consistency of Trip:

Coverage: 90%

Sub Function	Date	Temporal	Services	Ordering	Spatial
Trip	Yes	Yes	Yes	Yes	Yes
Service	Yes	Yes	Yes	Yes	Yes
Shared Trip & Services	Yes	Yes	Yes	Yes	Yes
Standalone Shared Service	Yes	Yes	Yes	Yes	Yes

Overview:

1) Validation was handled more of app based backend java validation since it was not only UI data which had to be validated but consistency as a whole had to be checked, taking a trip

and iterating through its services, looking for the order in the services, data validation,location validation.

2) Since this validation constraint was common, a Util class was created and accessed statically which contained the whole logic of validation to be performed generically for Trip,service across create,read,update,import of data. Util Class: ConsistencyValidator.java
3) Validation was almost achieved for all variations except for some rare missed test cases while testing like in regression cases.

4) **Cost Calculator:**

Coverage: 100%

Sub Function	Normal	Fidelity	Carbon
Trip	Yes	Yes	Yes
Service	Yes	Yes	Yes

Sample Screenshots:

Overview:

1) This was likewise made a Util Class accessed statically with ability to calculate generically for fidelity point,carbon credit as well as general cost. Util Class: CostCalculator.java
2) Requirement was achieved since it was more calculation by iteration of service elements and generating dynamically the cost of each trip.Trip cost is not stored anywhere,since it is dynamic and calculated using this functionality.
3) We made it a criteria to enter the fidelity point and carbon credit as a string expression in condition attribute which we parse and then use it to calculate fidelity point and carbon credits. **{Fidelity/Carbon/Other}:{Points/Credits}**

4)Sharing a Trip:

Coverage :100%

Sub Function	Share	Import	View	Delete
Shared Trip containing Only Service	Yes	Yes	Yes	Yes

Overview:

1) Most challenging part in the functionality was this with the consistency being the next but it was achieved after some trials
2) In conventional SMS message passing and receiving is effortless but taking into account the data format, data isolation(recognition of normal message from trip message),data order.
3) Reception of the shared message in a separate table and then giving the flexibility to import into main trip and service table or not is achieved.
4) Also SMS- RECEIVE,SEND,READ, CONTACTS-READ permission was declared in manifest and handled in the ViewTrip.java & ViewService.java
5) More explained in the protocol section and this functionality was achieved.

Database Structure & Overview:

Overview:

1) Database Structure was easily achieved through ROOM, thanks to the implementation of it mere annotation of the entity classes and DAO Interfaces where created for the following data objects.
2) Following represents the overall structure of entities:
3) Trip.java -------(One to Many)---→Service.java
4) SharedTrip.java---(One to Many)--→SharedService.java
5) SharedSingleService.java→ Temporary table for storing only services which are shared
6) User.java→ Additional functionality to store user details
   Corresponding POJO objects were also created for these for UI representation.

   Some custom queries were written using @Query in DAO to makeover for against the usual @Insert, @Update,@Delete.

Android & Generic Java Separability:

Overview:

1) Overall core java libraries used over Android libraries except for front end content and intent passing would be around 80% -JAVA ; 20% -Android
2) Even Validation was more of java oriented where each layer just a Map object was returned instead of returning android display messages.
3) Database part was more of custom queries which were used @query so android specific contribution is less.
4) Other business functionality like consistency, calculation, representation was only on pure java. Even repeated class were place in Util and accessed statically.

Trip Exchange Protocol Overview:

Overview:

1) Most interesting part of the project and more of android and device specific functionality which consumed a lot of experiments,thanks to it.

2) **Trials done:**

a) **Conventional XML format** exchange (Serialization and deserialization which was done through **SIMPLEXML** library. (Sample code is retained as well but commented)
   Drawbacks:
   1) SMS allows 140 characters, so it misrepresented the data for longer length
   2) Although simple to use but was not useful for this project.

b) **Data Message** instead of SMS message but it had the same limitation of characters so was tried and then dropped.

c) Finally **dividing the message** and sending : (Two variations tried)
   1) Simple dividing the message randomly using android split message was very haphazard split and so retrieving it was difficult.

2) So the logic was made in sendSMS() method of ViewTrip to split it well distributed custom data format of calculated some length of packets and the rearranged on receiving in UTIL Class: MessageParser.java

3) Each time a message was sent it was appended with **TripXP,** then for the start of the message was distinguished using **<TripManager>** XML tag,also the no of split messages to arrive is also sent in the parent message, which waits for the child messages after passing out of the broadcast receiver in a static variable.

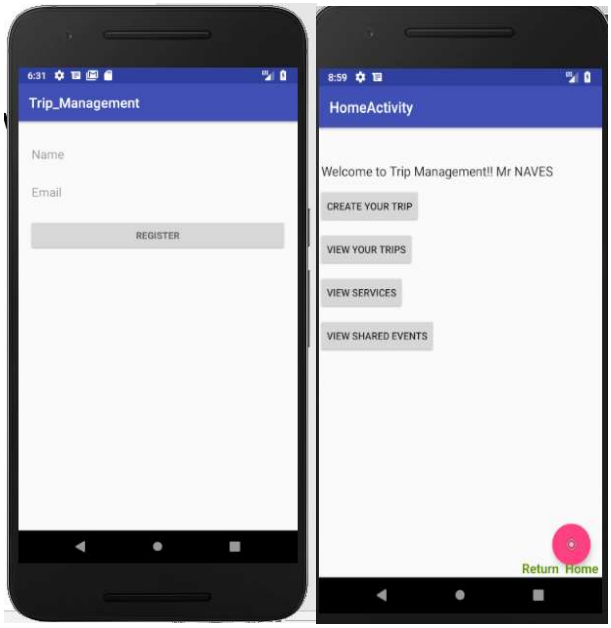
4) Static accessor and static variable are the main logic behind achieving this functionality defined in MessageParser, which is reset all the full message parts are arrived.

5) As a result of this: a) Messages are distinguished from the normal message b) Messages are arrived in order and then reconstructed on receiving c) More or less XML format specification specification is maintained but rather in a String format.

6) This functionality was achieved through the use of the above protocol defined between the sender and the broadcast receiver.

Additional Functionality:

Home and Login Screen:



This functionality was not necessary but to give a more formal touch to the app this was achieved and user details like contact were stored and later retrieved after the first time registration with a personalized Welcome content for a better representation.