Middle East Technical University

Department of Computer Engineering

CENG300
SUMMER PRACTICE REPORT

Student's Name:	*Instructor:*
Kerem Serttaş	Prof. Dr. Murat Manguoğlu

Start Date: 01.07.2020

End Date: 07.08.2020

Total Working Days: 25

Student's Signature: *Organization Approval:*

Table of Contents

1 Introduction

I have completed my summer internship at Innology Bilişim A.Ş. which has its office at Bilkent CyberPark, Ankara. However due to the Coronavirus pandemic I have not worked at the office. In order to follow the safety measures, our team worked remotely throughout the pandemic.

Our team focused on development of a mobile application. For this purpose we worked on UI/UX features, database integration and management of service-client services while mainly using Google's cross platform flutter framework and its engine as Dart programming language. For database modeling and incorporation we predominantly used Firebase services. This will be further explained in the next sections of this document.

2 Information about the Project

Throughout my internship at Innology, I have primarily focused on three projects in order to improve the consistency and user experience of the application. First assignment consisted of designing a suitable database for the application in order to add a searching functionality. Secondly, while the project evolved, the organization determined to change the software architectural pattern. Therefore we have been assigned to change the architectural pattern of the search engine. Finally, I have been assigned to clean and restructure some of the constants that are used throughout the project.

2.1 Search Engine

On our first task for this internship, we as a group have been assigned to design and implement a real-time fast search widget.

2.1.1 Analysis phase

Main problem for this project was to find an easy and suitable database, then to fetch the data from this relatively big database efficiently and update it real time.

Second objective for this project was to find a correlation algorithm to match the key-value pairs more relevantly. Most significant setback was to design a structure for the database in a way that we can query the data fast and at the same time correlate the search results effectively.

Third objective was, if the user searched a phrase that does not exist in the database, we would ask the user to propose it to us so that our team would work on that in order to improve the user experience. My task was to implement a query that will be invoked when necessary and reach the database.

2.1.2 Design Phase

The organization decided to choose the Firebase Services in order to integrate to the project. Initial plan was to use the Firebase Real-time database however after some consideration, Firebase Cloud Firestore has chosen, since it is a better fit for the flutter framework. Since both Real-time DB and Cloud Firestore have noSQL properties, it took some time for the team to adjust to it. Initial design for the algorithm was to use a simple approach, as for each key the user puts in, finding a relevant query for the data to be used in filtering functionalities.

After some time, a better approach is designed using an inverted index. For each data in the main database there exists a counterpart in a second branch called “Inverted Index” which has the index of the selected phrase. This inverted index structure helped the team to find a better way to query the data for multiple word phrases.

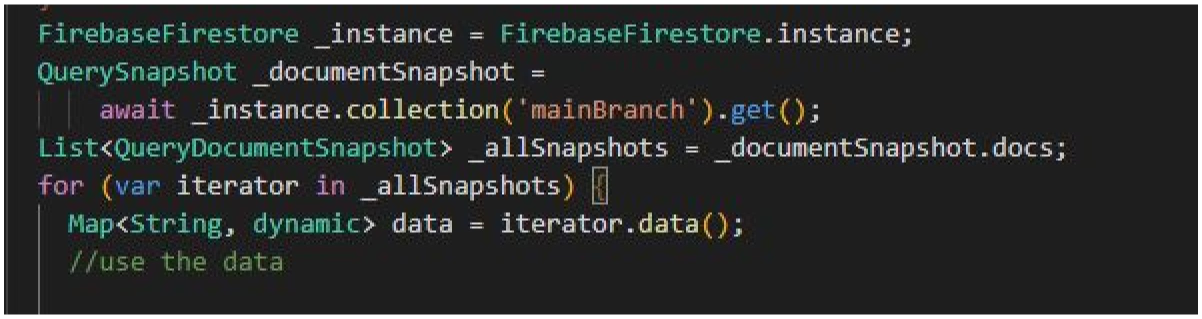
Second objective was to find a correlation algorithm between relevant data and structure the database for a decent correlation and fast querying. Initial design for the database was to include the properties mentioned above as a “Three-Layered Tree”. The correlation algorithm would work such that each phrase that sits in the same or neighboring sub tree has a higher score regarding the results that are shown after each key the user puts in. If the results that are searched for sit in the same or close sub tree, the algorithm would put these results first rather than some results that are in a distant sub tree. However, this structure posed a major threat since Firebase uses noSQL it would download all the sub trees while searching each input. Therefore our development team searched for a more efficient way to store the data. After some consideration, our team decided to use a linear approach rather than a tree. However, for every word that is stored in the database, there exists properties such that these properties consist of a list of its ancestors and list of its children. By designing this way, we keep the advantages of a tree while using a linear approach.

Third objective was to create a way for the cases if the searched query does not exist in the database. After searching, the result page has an "On Submitted" handler which is called automatically when the user finishes typing. Depending on the response this handler calls the corresponding method to add this keyword to the suitable branch of the database.

2.1.3 Implementation Phase

In order to query the data from firebase cloud firestore database cloud firestore plugin is used. Main task that was given to me was to handle the database interactions while searching.

As shown in Figure 1 below there is an example of a simple way that is used to get each data stored in the database.



```
FirebaseFirestore _instance = FirebaseFirestore.instance;
QuerySnapshot _documentSnapshot =
    await _instance.collection('mainBranch').get();
List<QueryDocumentSnapshot> _allSnapshots = _documentSnapshot.docs;
for (var iterator in _allSnapshots) {
  Map<String, dynamic> data = iterator.data();
  //use the data
```

Figure 1

Using the first line we get an instance of the database then the second line gives us all documents stored in a branch named as 'mainBranch'.

By using the "***.docs***" property we get a list of the snapshots that we can reach.

At the final line we iterate through the list of snapshots and get the actual data using the "***.data()***" method.

We used a similar approach to load the inverted index branch of the database itself.

In addition I was given the task of designing the database in order for the correlation algorithm to reach the data linearly and understand it easily to figure out which results phrases correlate with each other.

As stated above, we first designed the database as a **three-layered tree** just like in figure 2 shown below.

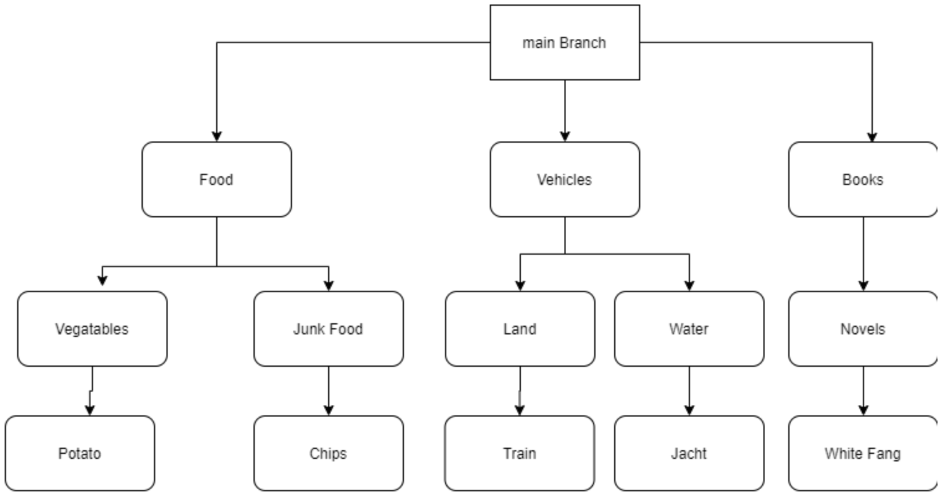


Figure 2

After that we decided to implement the structure shown at figure 3 below. Due to this **linear** approach, the search algorithm would prioritize the query depending on its **children and its ancestors**.

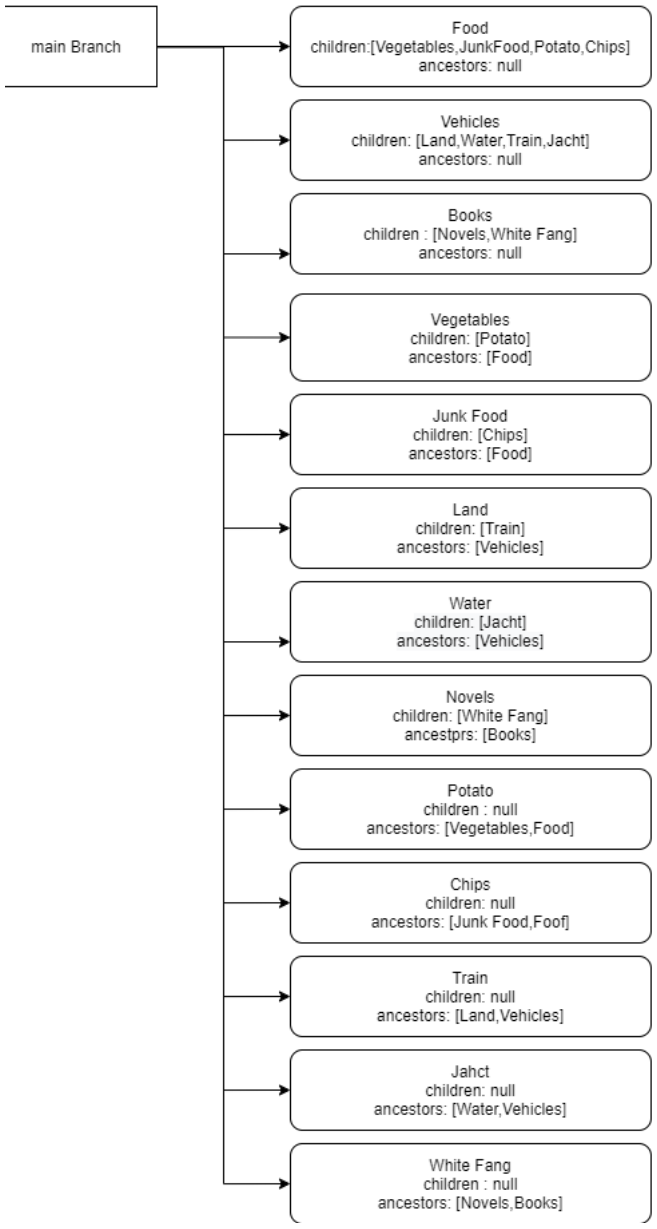
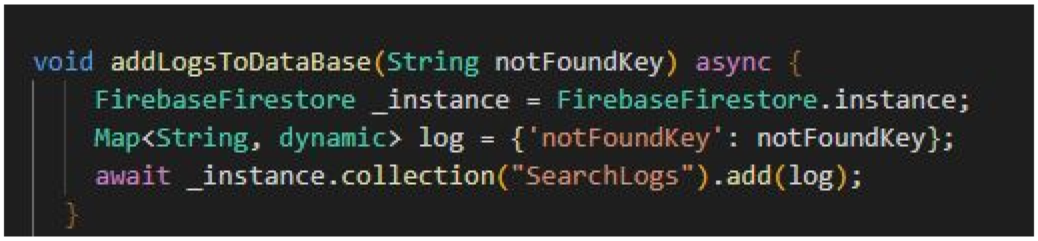


Figure 3

I was also given the role to create a branch to keep the logs of each case when the searched query does not reach any result.

For this purpose I created a new branch called “Search Logs” into the database and updated it using the Cloud Firestore plugin.



```
void addLogsToDataBase(String notFoundKey) async {
    FirebaseFirestore _instance = FirebaseFirestore.instance;
    Map<String, dynamic> log = {'notFoundKey': notFoundKey};
    await _instance.collection("SearchLogs").add(log);
  }
```

Figure 4

This function shown in Figure 4 above uses the properties of cloud firestore library in flutter using the principles of asynchronous programming. First it takes an instance of the database.

Then using this instance it reaches out to the specific branch using ***".collection(branchName)"*** method.

Since the data stored in the database as Map<String, dynamic>, "***addLogsToDatabase()"*** function first creates the Map and adds it to the corresponding branch using the "**.add*(Map<String,dynamic>)"*** method.

Data type "dynamic" is a special data type in dart language which has methods for every possible identifier and arity, with every possible combination of named parameters [1].

2.1.4 Testing Phase

For the three subparts of this project, unit-testing responsibility is divided into other personnel on the team. However since my work primarily consisted of Firebase Database integration, try-catch blocks and exception handling have been done throughout the project .Furthermore, a lot of erroneous inputs and odd exceptions have been considered.

2.2 Software Architectural Pattern Change

Initially the project that our team was given worked with a mix of different architectures. After some testing phases, the organization decided to change the architecture pattern of this project to MVVM (Model-View-ViewModel).

2.2.1 Analysis phase

By changing the design pattern to MVVM, the application code could be separated to three categories which are Models, Views and ViewModels. This separation creates a distinction between UI related components of the code from the underlying system. Therefore testing for each category could be done faster and more accurately than before. Plus, design changes or data model changes could be done separately without needing to change the whole system.

2.2.2 Design Phase

In this particular task we were given the task to change the architecture of the search page. Initially the search page had a complicated and intertwined set of stateful widgets which mixes the UI code with underlying data models and backends.

Our new design consisted of changing all UI-related components to specific widgets as View's. Each View would have its own ViewModel that would control and provide the data. By doing that we make sure that no flutter code will be written in any files rather than the view files. Only the files that contain View's would have flutter code written in them. Furthermore each specific data model is taken out from the Views and View Models and put in a specific library which is called "Data Models". As a consequence of that View Models and Data Models will be written with pure Dart code.

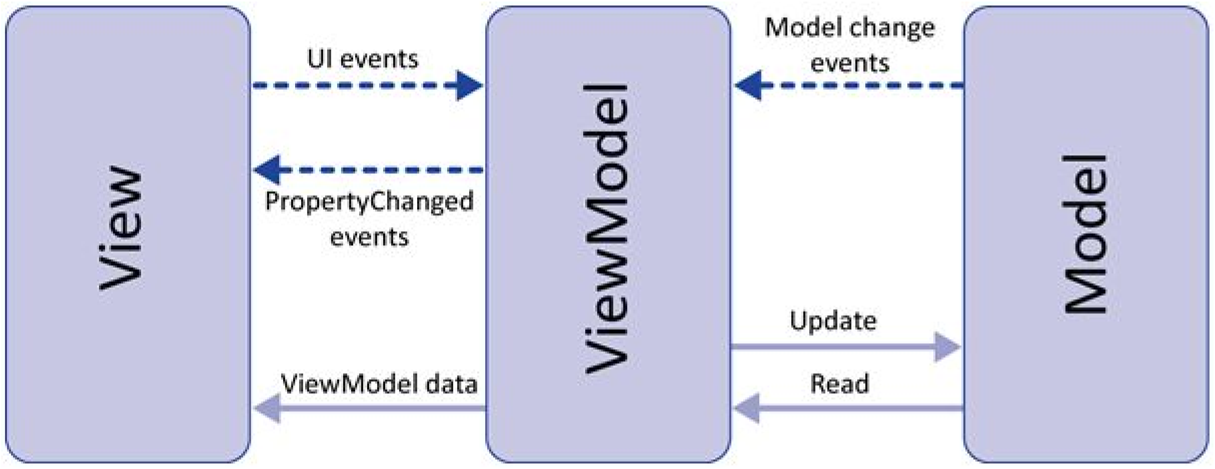


Figure 5

2.2.3 Implementation Phase

At first we decided to have 3 different alternatives for the search page, depending on whether it shows the results or error or loading. Therefore we implement an enumeration that would represent 3 states that the screen could be at (shown in Figure 6 below).

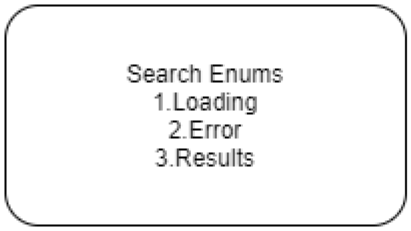


Figure 6

Therefore for each enumeration there exists a specific View Widget. Depending on the user's input, ViewModel decides which View widget will be built and kept on the screen. By doing that we separate the UI components from the dart code. It will be a great advantage once we start unit testing. Since UI components and ViewModel components can be tested individually we can easily separate and detect the problem whether it is a performance issue or a fundamental issue.

For this purpose a third party library named ‘stacked’ [2] is used. This specific library handles the ViewModel to View connection. Major benefit of this library consists of removing a lot of boilerplate code.

Every time there happens a massive change in the ViewModel, *notifyListeners()* method is called. This method ensures the change is rebuilt in the View widget therefore we can make state changes. In other words, we make sure that View widgets depend on the ViewModel widget. If a specific property of the ViewModel is used to determine a property in View widget and this specific property of the ViewModel widget changes, this method guarantees that corresponding View widget must be rebuilt using the new value.

After moving the search page structure to MVVM architecture we had to make sure that Cloud Firestore database integration of this page is taken out from the messy structure and added into the Viewmodel’s properties. Due to this action, corresponding ViewModel widget controls which View widget is allowed to reach or change database values.

2.2.4 Testing Phase

Unit-testing part of this project has been done by other members on our team. My main focus consisted of making a smooth transition to MVVM. However in order for these new systems to function properly, try-catch blocks and exception handling have been used excessively. Plus, by making this architectural change we controlled the amount of instances used in the application therefore we observed a significant amount of improvements considering the performance.

2.3 Colors, Fonts and Box Sizes Refactoring

2.3.1 Analysis Phase

The project had complex and interconnected pieces of screens in it. These screens come with various designs. In addition, these designs usually consisted of mixtures of colors and different sizes of boxes which were the bases for almost all UI elements. Furthermore, almost every screen has some particular text which is also designed using different types of calligraphy styles. However since every screen was not implemented by the same team, these screens were not alike each other. They did not pose any coherence whatsoever. As a result I was given the task of uniting the elements used in these screens while not changing each screen drastically.

2.3.2 Design phase

Initial action for this assignment was to detect the specific changes that needed to be made. Due to this, three main categories stood out. These were:

1- Colors used throughout the screens
2- Font types and their sizes which are used throughout the screens
3- Height and width of the boxes that are used throughout the screens

After detecting the objectives, I decided to determine main colors (background color, button color, appbar color, divider color etc.) throughout the application and ask the designers for approval. Following the separation of the main theme colors I dove deep into the screen specific colors and determined the necessary ones.

Second task was about refactoring the calligraphy of the application. I started out by determining the main font styles and its sizes depending on its purpose. At the end, I decided to select main text colors (appbar text color, header text color, input text color etc.).

At last, I planned to detect all the boxes used throughout the application and then unify them accordingly. Therefore, there would be no random numbers in the UI code itself.

2.3.3 Implementation phase

At the beginning, I created three separate files which were named as *colors.dart*, *fonts.dart* and *sizes.dart*.

In order to arrange the *colors.dart* file I determined to create different classes such as:

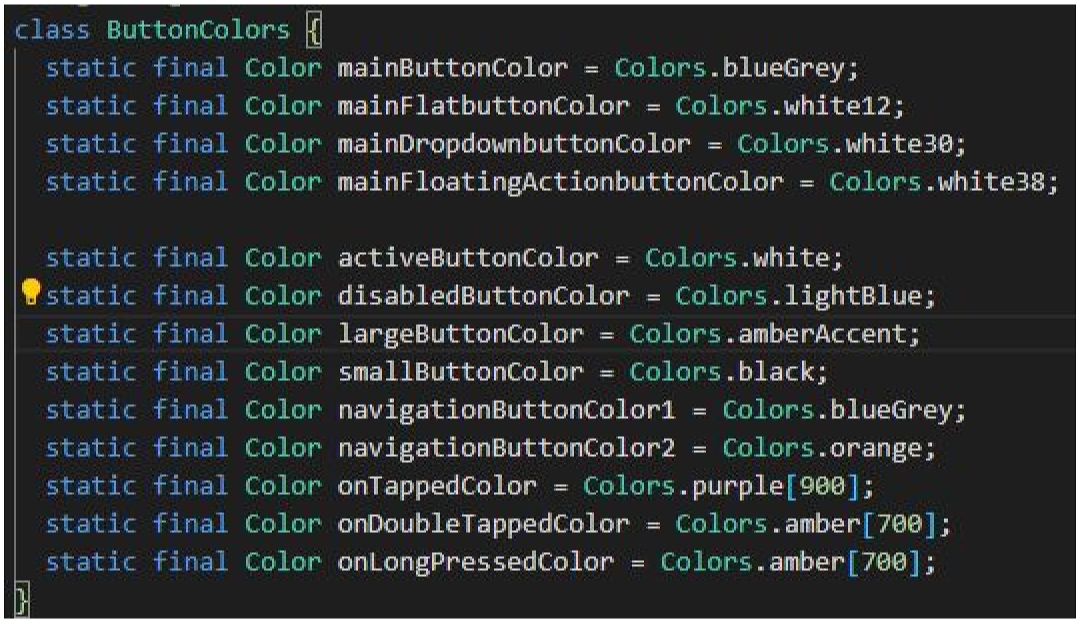
1. IconColors
2. TextColors
3. CustomColors,
4. ThemeColors,
5. BackgroundColors
6. ButtonColors
7. ScreenSpecificColors

For the purpose of making these class structures more efficient I decided to use static class properties. The aim for this was not to create an instance rather than using the class itself to reach and use the properties.

Static means a member is available on the class itself instead of on instances of the class. That's all it means, and it isn't used for anything else. Static modifies “members” [3].

Additionally I decided to use the “final” keyword for the static member properties. The keyword “final” is specific to dart language and it means single-assignment: a final variable or field must have an initializer. Once assigned to a value, a final variable's value cannot be changed [3].

By using “static final” properties I made sure that these could not be changed dynamically and not instantiated, which makes it more efficient rather than a regular variable.



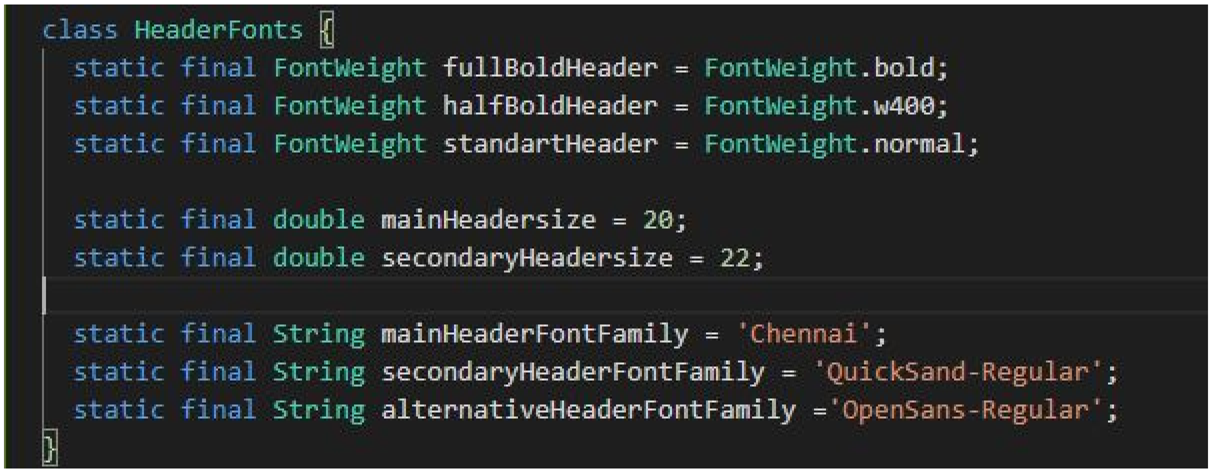
```
class ButtonColors {
  static final Color mainButtonColor = Colors.blueGrey;
  static final Color mainFlatbuttonColor = Colors.white12;
  static final Color mainDropdownbuttonColor = Colors.white30;
  static final Color mainFloatingActionbuttonColor = Colors.white38;

  static final Color activeButtonColor = Colors.white;
  static final Color disabledButtonColor = Colors.lightBlue;
  static final Color largeButtonColor = Colors.amberAccent;
  static final Color smallButtonColor = Colors.black;
  static final Color navigationButtonColor1 = Colors.blueGrey;
  static final Color navigationButtonColor2 = Colors.orange;
  static final Color onTappedColor = Colors.purple[900];
  static final Color onDoubleTappedColor = Colors.amber[700];
  static final Color onLongPressedColor = Colors.amber[700];
}
```

Figure 7

An example class ButtonColors given above at figure 7. By the assistance of this class, every button used throughout the application is unified and coherent.

Similar to *colors.dart* case *fonts.dart* had simple classes that contained the unified information about the necessary fonts used in the application. For the purpose of not repeating, I would not explain every class. I will rather provide an example class such as HeaderFonts shown in Figure 8 below.



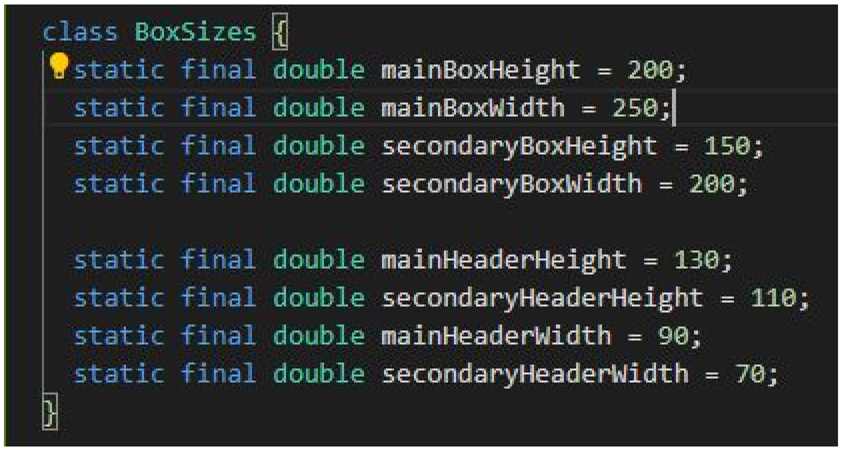
```
class HeaderFonts {
  static final FontWeight fullBoldHeader = FontWeight.bold;
  static final FontWeight halfBoldHeader = FontWeight.w400;
  static final FontWeight standartHeader = FontWeight.normal;

  static final double mainHeadersize = 20;
  static final double secondaryHeadersize = 22;

  static final String mainHeaderFontFamily = 'Chennai';
  static final String secondaryHeaderFontFamily = 'QuickSand-Regular';
  static final String alternativeHeaderFontFamily ='OpenSans-Regular';
}
```

Figure 8

Similar to *colors.dart* and *fonts.dart* *sizes.dart* had simple classes that contained the unified information about the necessary sizes used in the application. For the purpose of not repeating, I would not explain every class. I will rather provide an example class such as BoxSizes given at figure 9 below.



```
class BoxSizes {
  static final double mainBoxHeight = 200;
  static final double mainBoxWidth = 250;
  static final double secondaryBoxHeight = 150;
  static final double secondaryBoxWidth = 200;

  static final double mainHeaderHeight = 130;
  static final double secondaryHeaderHeight = 110;
  static final double mainHeaderWidth = 90;
  static final double secondaryHeaderWidth = 70;
}
```

Figure 9

2.3.4 Testing phase

Due to the fact that this project only changes and rearranges constant variables. Unit testing was not possible however by the use of static member variables, performance should be improved theoretically.

3 Organization and Structure

Innology Bilişim A.Ş. is located at Bilkent Cyberpark/Ankara. Innology works on building tailor-made products from scratch depending on the needs of the client.

3.1 Methodologies and Strategies Used In the Company

Throughout my internship our team of 8 people worked with a project co-founded by Innology. This project consisted of a mobile application. At the beginning, engineers at the company showed us the tools and techniques that are used throughout the company. After a while we had been accustomed to using Gitlab as the official version control system.

Since we were working on a project that was co-founded, we were in close contact with the designers and executives of the other company. Therefore we could reach and ask for specifics whenever necessary.

3.2 Products offered by Innology

3.2.1 Taxi dispatch systems

One of the main products offered by Innology consists of a service called Itaksi. Itaksi is the official taxi hailing service of Istanbul. Their work is a top-notch taxi dispatch services, included but not limited to developing software, setting up and managing the hardware and ensuring flawless execution of the project among multiple shareholders [4].

iTAKSi

3.2.2 Digital out-of-home advertising

Another product patented by Innology is called ZAP. ZAP is an eye-catching digital advertising channel for rideshare car-tops which display ads in a geo-targeted and daily-parted manner, publishing the right message to the right audience in the right time and place. This rideshare car advertising platform is developed using AI/ML technologies [4].

4 Conclusion

At the beginning of my internship, I had no prior experience about a mobile application development environment and how project management systems work. Before working on an actual project, I always tried to make a project work as fast and as simple as possible. However throughout my internship I once realized that developing an efficient system is so much more crucial. In addition, it is rather important to plan ahead and design the project first, then make the necessary steps to implement and test it. However, even though we plan and design carefully, project needs and market needs may change dramatically. Therefore I concluded that a team should adapt to the changes while redesign when it’s necessary.

Finally, being part of this amazing environment was a great pleasure. I would like to thank the Innology Organization for making this happen.

5 References

[1] Dart Programming - Data Types. (n.d.). Retrieved October 17, 2020, from https://www.tutorialspoint.com/dart_programming/dart_programming_data_types.htm

[2] Stacked: Flutter Package. (2020, August 07). Retrieved October 17, 2020, from https://pub.dev/packages/stacked

[3] Ladd, S. (2002, June 1). Const, Static, Final, Oh my! Retrieved October 17, 2020, from https://news.dartlang.org/2012/06/const-static-final-oh-my.html

[4] Innology. (n.d.). Watch your idea materialize. Retrieved October 17, 2020, from https://www.innology.com.tr/