Application of Real-Time Databases

Ryan Goodwin
rgoodwin@smu.edu

Karl Jurek
kjurek@smu.edu

Travis Daun
tdaun@smu.edu

***Abstract*—With the exponential growth of embedded and other real-time systems, database systems are needed that are capable of handling workloads whose state are constantly changing, all while ensuring data correctness. While relational databases contain persistent data that is mostly unaffected by time, real-time databases employ timing constraints and deadlines to determine if data is valid or not. This paper explores the key elements required to employ a real-time database and pilots an application for tracking and broadcasting gasoline prices between users using a real-time database know as Google Firebase.**

***Index Terms*—RTDB, Real-time Systems, Firebase**

I. Introduction

A real-time database can be defined as a database system which uses real-time processing to handle workloads whose state is constantly changing [1]. While relational databases are effective at managing persistent data, they are not effective at dealing with dynamic data that is constantly changing. Real-time databases are capable of providing deadlines and wait periods to ensure data correctness through temporal consistency. While the conventional thought of data is that data is timeless, in a real-time database the validity and correctness of data can quickly decay making the data stale and meaningless. This paper will present the key concepts of data correctness and how data correctness is ensured in a real-time database. This paper will also discuss a pilot project that was developed using the Google Firebase real-time database.

II. Real-Time Databases

A. Overview of Real-Time Databases

As the need for real-time processing in databases for various applications continue to grow, real-time database systems have been developed to provide the functionality of a traditional database system while also assuring that real-time constraints are imposed on both the data and the transactions. Research into real-time database system has provided a significant contribution into defining constraints and performance objectives when implementing real-time databases in various applications. Two key areas of computer science has been integral in the continued development of real-time database systems [2]. Figure 1 shows this important relationship and the key aspects of each that are relied upon for the implementation of a real-time database system.

To understand the need for real-time databases, it is important to have a high level overview of some of the various applications that employ real-time databases. While the applications can be endless, a big picture understanding of a few of them will provide context that can be expanded to almost any other application. Self-driving vehicles employ numerous sensors for geo-location (such as latitude/longitude and car velocity) and object detection (microwave radar and other such sensors) to make real-time decisions. These decisions are the result of the derived data from various dynamic and static data inputs. Flight control software, which a key example of this has been in the news lately with the Boeing Super Max flight control software, was designed to make adjustments to flight control surfaces based on sensor inputs. Industrial control systems which can take various real-time inputs to make changes to industrial process (i.e. a simple pressure temperature relationship which can adjust energy input based on sensors for pressure and temperature). Lastly electrical grid operations employ various sensors to control grid operations based on energy demand or sensed faults to automatically re-route electrical distribution networks. This ”smart grid” technology is used to prevent or minimize service disruptions or even cycle off customer’s air conditioning units during times of high grid usage. These four examples all share the commonality that the state of the data at one point of time can be significantly different than it is in even a few milliseconds. Making determinations or predictions using stale data can be detrimental to the system and thus data correctness needs to be ensured before actions are taken as a result of the data.

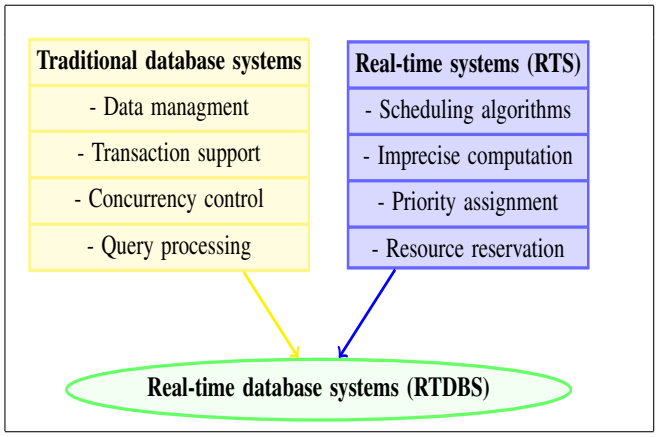

Fig. 1. Real-time database systems

B. Transaction Properties

Real-time transactions within a real-time database are characterized by distinct properties: the nature of real-time constraints, the arrival pattern, and the data-access type [3]. Real-time constraints relate to the urgency of the transaction and the effect of missing its deadline. These constraints can be

categorized into three types; hard, soft, and firm [4]. Hard real-time constraints require that deadlines cannot be missed and the deadline is guaranteed by the system. An example of this would be velocity for a self driving vehicle. Every predefined interval would require that both speed and direction are obtained so that the exact location can be known within an area of uncertainty based upon the accuracy of the sensors employed. With this information the vehicle knows where it is and where it will be at the next predefined interval, and thus must be guaranteed to occur for the system to operate. Soft real-time transactions are not guaranteed by the system and can only result in the performance degradation of the system if the deadline is missed. Again, looking at an example of a self-driving vehicle, a soft constraint could be a GPS location of the vehicle. While a continuous latitude and longitude helps to shrink the overall area of uncertainty of the cars location, at times that GPS may not be available such as when going through a tunnel. The area of uncertainty of the vehicle's location based upon the vehicle's velocity may grow and the parameters relied upon for this may degrade to account for this until the GPS location is regained and compared to the velocity calculated position of the vehicle. As a result of this degraded performance, parameters such as stand-off distance to the car in front of you may increase. Firm constraints are a special case of soft constraints in that when the deadline expires, the constraint is killed. Again employing an example of a self driving vehicle, this could be associated with a sensor that detects if the headlights should be on or not. If the sensor that detects outside illumination does not return a value to the database, the lights would remain on until a valid detection is received that would indicate that the lights could be turned off.

A second property employed with real-time transactions is the arrival pattern of transactions. Arrival pattern transactions refer to the time interval transaction are execution. These arrival patterns can be periodic, aperiodic, or random [3]. Periodic transactions are transactions that are periodically executed within the real-time database. Aperiodic transactions means there is not a regular time interval between the executions of transactions [3]. Again looking at an example of a self driving vehicle. The velocity sensors (speed and direction) would be periodic since they are required to be executed at a fixed interval to ensure the location of the car is known and can be predicted for the next interval. The GPS sensors would be aperiodic since the data is available when they are available, but if in a tunnel or blocked by a building they would not be able to be executed. Random would be the object detection sensors since objects would appear at random as the vehicle is moving. Random transactions would only execute when an object is detected.

The third transaction property of real-time databases relate to the data access type and can be categorized as: write transactions, update transactions, and read transactions. As their names suggest, write transactions obtain the state of the environment and write to the database, update transactions derive new data and store them into the database, and read transactions read the data from the database and send them [3].

As with traditional databases, real-time databases must guarantee internal consistency through ACID properties. A real-time database defines these properties as follows. Atomicity is applied for subtransactions, where subtransactions must be wholly executed or no transaction can be considered from them. Consistency requires that the transaction execution must always change the consistent state of the database in another consistent state. Isolation requires actions of a transaction be visible by other transactions before it commits. Durability requires the actions of a transaction need not be persistent, since both data and transactions have temporal validity [4].

The implementation of a real-time database, vice a relational database, is required for various reasons. These reasons include applications in which the volume of data is large, the responses depend on multiple values, responses to aperiodic events are required, or when there are constrained timing requirements that must be met [4]. The reason for this is that unlike relational database systems, real-time database systems must not only maintain database integrity but they must also meet the urgency of transaction executions.

C. Data Properties

The data correctness of a real-time database is more difficult to assure than for a relational database since the data is dynamic and the state of the database changes constantly. The correctness of the real-time database implies that all usual consistency constraints are satisfied as well as being able to execute transactions within specified timing constraints and satisfy temporal consistency of the data. This temporal data consistency is concerned with the relative recency of data within the database. Data correctness for a real-time database is assured by both logical and temporal consistency [5]. Transactions are only considered correct if they finish within their deadlines using valid data [1].

The data within the database can be classified as static or dynamic. Static data correctness can be guaranteed by logical consistency since it has not become outdated. Dynamic data changes to reflect the real-world state which makes logical consistency impossible and thus temporal data consistency must be used [5].

Since dynamic data is time-stamped, validity intervals can be used to assure the consistency between the state represented by the database content and the actual state of the environment. The absolute validity interval (avi) is defined between the environment state and the value reflected in the database [4]. This can be used to determine if the data is temporally inconsistent by comparing the data time-stamps. For instance, data x is temporally inconsistent if $Current_{time} - Current_{timestamp}(x) > avi(x)$.

The relative validity interval (rvi) is defined among the data used to derive other data [3]. For instance, in a real-time database that contains object speed s and direction d, a moving object's position can be defined by the object's location at the previous time stamp plus the velocity vector times the elapsed time. $position_{t1} = position_{t0} + velocity_{average}(time_{t1} -$

$time_{t0}$). As long as the time interval between the object's speed and direction is small (as a function of the object's ability to change speed and direction), the error of the average velocity will also be small. Therefore the recency of the s vector is relative to the recency of the d vector in such that the actual position of an object can be found with the s and d vectors for any given time. As discussed, this is relative to the objects ability to change speed and direction so the recency of data for a jet plane would have a much smaller relative validity interval required than say for a typical on-road vehicle.

To best understand the data property requirements and how they are implemented for a real-time database, a simple example can be employed. Figure 2 depicts a simple real-time database with two sensors which are writing parameters to the database in real-time and one derived variable which updates to the real-time database based upon the sensors. S_1 and S_2 are dynamic sensors which can be represented by $x : (value, avi, timestamp)$. For this example we will again use our self driving vehicle example for context. S_1 will represent the direction component of the velocity vector based upon a compass heading; 0^o equates to North, 90^o equates to East, 180^o equates to South, and 270^o equates to West. S_2 represents the speed component of the velocity vector in miles per hour (mph). D_1 is a derived variable for the displacement using S_1 and S_2. In order for the dynamic data to be temporally consistent, both the absolute and relative validity interval must be satisfied. For this example the design constraints for the vehicle are $avi(S_1) = 5ms$ and an $avi(S_2) = 10ms$. D_1 is derived from S_1 and S_2 and we assign a relative validity interval $rvi(D_1) = 2ms$. This says that the direction component of the velocity vector must not be "older" than $5ms$, the speed component cannot be "older" than $10ms$, and that the relative difference time between the speed and direction components cannot be greater than $2ms$.

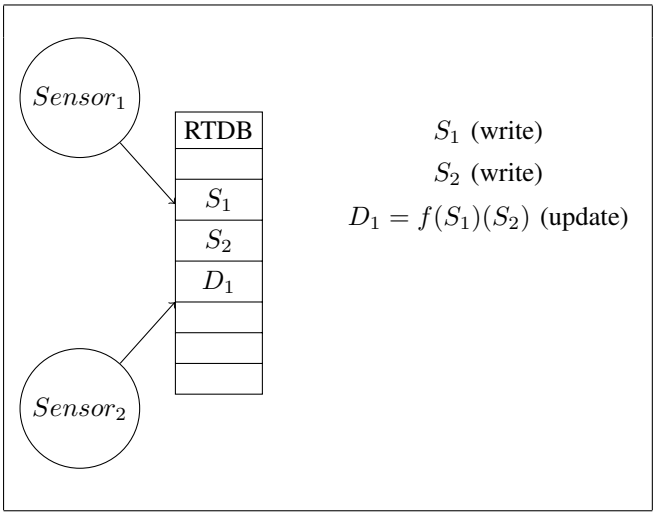

Fig. 2. Simple Real-time database system

Looking at case 1 from Table I you can see that the avi values for both S_1 and S_2 are met:

Case	Time	S_1	S_2	temporal consistency
1	25	(124,5,20)	(16,10,22)	Assured
2	25	(124,5,20)	(16,10,24)	Not Assured
3	25	(124,5,19)	(16,10,17)	Not Assured

TABLE I
TEMPORALLY CONSISTENCY DATA

$$avi(S_1) = (25 - 20 = 5) \leq 5 \quad (1)$$

$$avi(S_2) = (25 - 22 = 3) \leq 10 \quad (2)$$

Looking at the $rvi(D_1)$ it can be seen that

$$rvi(D_1) = (22 - 20 = 2) \leq 2 \quad (3)$$

so relative temporal consistency is assured. In case 1 both the relative and absolute intervals are met so in case 1 temporal consistency is assured.

In case 2 on the other hand, the absolute intervals are met but

$$rvi(D_1) = (24 - 20 = 4) > 2 \quad (4)$$

so the relative interval is not met and temporal consistency is not assured.

In case 3 the avi value for S_2 is met:

$$avi(S_2) = (25 - 17 = 8) \leq 10 \quad (5)$$

but the avi value for S_1 is not met:

$$avi(S_1) = (25 - 19 = 6) > 5 \quad (6)$$

even when the rvi for D_1 is met:

$$dvi(D_1) = (19 - 17 = 2) \leq 2 \quad (7)$$

so the temporal consistency is not assured. In both case 2 and 3, the transaction are not considered correct because the data would not be considered valid.

III. EMPLOYING A REAL-TIME APPLICATION WITH A REAL-TIME DATABASE

A. Google Firebase

The Firebase Realtime Database is a cloud-hosted database in which data is stored as JSON and synchronized in real-time to every connected client. Firebase allows cross-platform apps to share one real-time database instance and automatically receive updates with the newest data. Data is persisted locally, and even while offline, real-time events continue to fire, giving the end user a responsive experience. When the device regains connection, the Realtime Database synchronizes the local data changes with the remote updates that occurred while the client was offline, merging any conflicts automatically [7].

The Firebase real-time database is a NoSQL database and has different optimizations and functionality as compared to a relational database. Instead of typical HTTP requests, the Firebase real-time database uses data synchronization every time data changes, any connected device receives that update within milliseconds [7]. Firebase apps remain responsive even

when offline because the Firebase real-time database SDK persists data to disk. Once connectivity is reestablished, the client device receives any changes it missed, synchronizing it with the current server state [7]. Firebase can be accessed directly from a mobile device or web browser meaning theres no need for an application server. Security and data validation is available through the Firebase real-time database security rules which are expression-based rules that are executed when data is read or written [7].

B. Cheap Gas Finder Application

An application to update real-time gas prices as reported by users to all other users was developed and hosted on Google Firebase. The application enabled cross-platform functionality for both both web based and Android based functionality. The application allows a user to enter the price of gas on their application for a particular gas station and in real-time, update all other connected users of the price. The web based application can be seen in Figure 3 and the Android based application can be seen in Figure 4. The active web application is hosted at the following url: https://project-30a72.web.app.

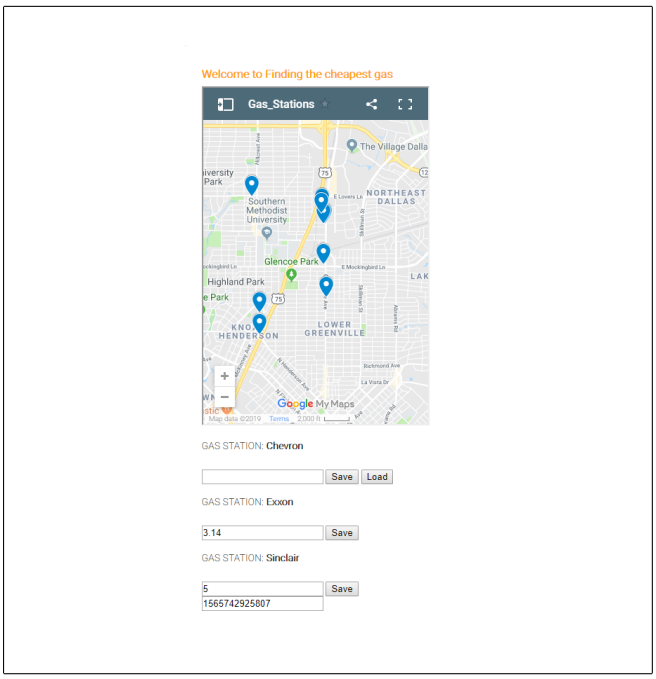

Fig. 3. Web based Application for Cheap Gas Finder

The source code for both applications can be found in appendix A. The web based application source code can be found in section A of the appendix and the Android based application source code can be found in section B.

IV. Connecting the Google Firebase RTDB

One of the biggest challenges was connecting the Android application to the Firebase real-time database. This was accomplished in the following manner.

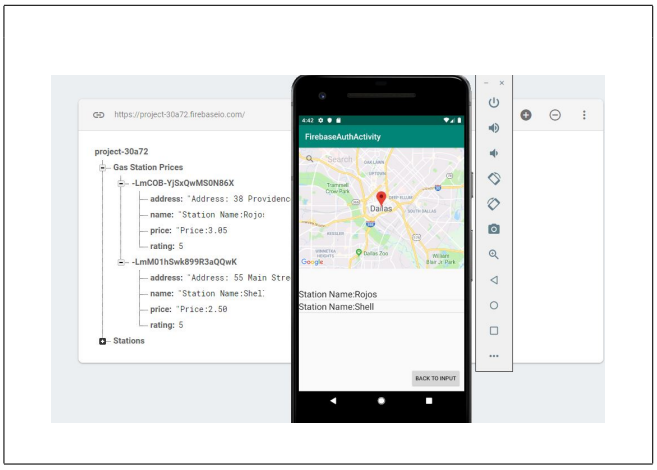

Fig. 4. Android Application for Cheap Gas Finder

1) packageID: Each Android project comes with a packageID in the build.gradle file which is the file used to compile the application. Firebase needs this ID since it is what provides the connection from Firebase to the local application. This can be seen in Figure 5

Fig. 5. Android Application packageID

2) Firebase Configuration File: Firebase will then give generate a configuration file called `google-services.json`. This sets up the application with everything it will need when contacting the database to the application. The `google-services.json` is then copied into the application level folder. This can be seen in Figure 6

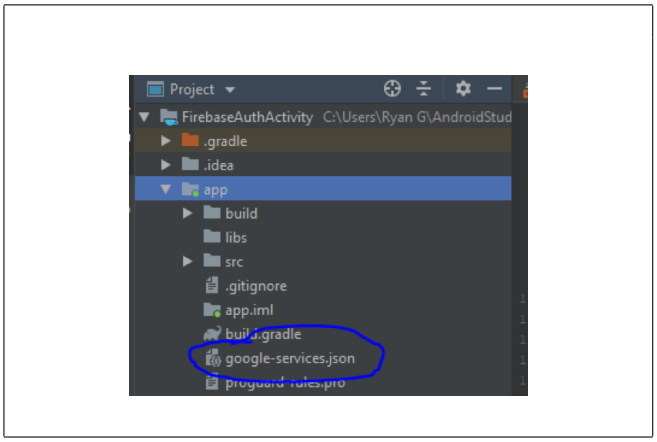

Fig. 6. Android Application packageID

3) Adding Implementations: To get the application to read the json file when it needs to, go back to the build.gradle file and add the implementations into the dependencies setup to be able to utilize the commands the json configuration file has provided. This can be seen in Figure 7

Fig. 7. Adding implementations to build.gradle

4) Application Building: Now the application can be built making all variables and layout. When variable need to be pushed to the database, these commands will write directly to the Firebase real-time database in a collection called `Gas Station Prices` at the push of a button. This can be seen in Figure 8

Fig. 8. Build Application

5) Application Verification: The basic setup of the app in action can be seen in Figure 9 and then verify what gets written to the database as seen in Figure 10.

V. Conclusions

Real-time databases provide the functionality of real-time processing for workloads that are constantly changing as needed for real-time systems. The traditional transaction and data properties of relational databases have to be extended to ensure not only logical consistency but temporal consistency in addition to the other usual consistency requirements. Without temporal data consistency, data within a real-time database can become stale. Since transactions are only considered correct if they finish within their deadlines using valid data, both transaction properties and data properties must be well understood for the system being employed.

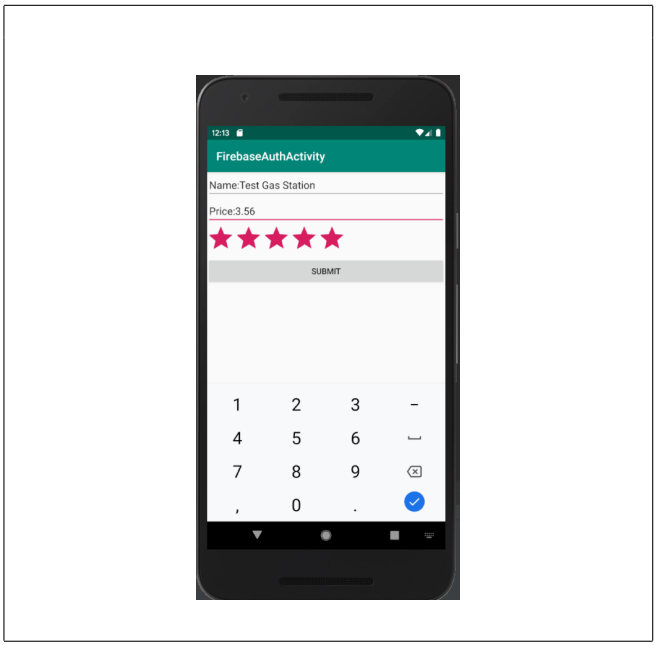

Fig. 9. Enter data into application

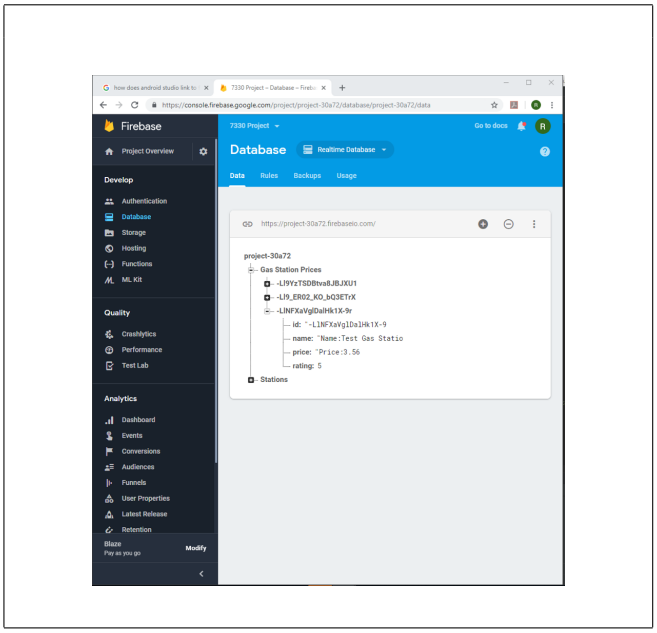

Fig. 10. Verify what is written to database

The Google Firebase real-time database provided the necessary tools to employ a cross-platform real-time application. The ability to perform real-time updates and broadcast them out to users was implemented and tested. While there was a steep learning curve in integrating the web based and Android based applications with Google Firebase, the end result was an application that provided real-time updates on gasoline prices that geo-located the stations on a Google map and transmitting those updates out to the various users using the Cheap Gas Prices applications.

REFERENCES

[1] Buchmann, A. Real Time Database Systems. Encyclopedia of Database Technologies and Applications. 2005

[2] Lam, Kam-Yiu., and Kuo, Tei-Wei. Real-Time Database Systems Architecture and Techniques. Boston, MA: Springer US, 2002.

[3] Erickson, John. Database Technologies: Concepts, Methodologies, Tools, and Applications. Hershey, PA: Information Science Reference, 2009.

[4] Halpin, Terry. Selected Readings on Database Technologies and Applications. Hershey, PA: Information Science Reference, 2009.

[5] Kang, K. "qRTDB: QoS-sensitive real-time database. PhD thesis, Department of Computer Science, University of Virginia, Charlottesville. 2001

[6] Han, Song, Lam, Kam-yiu, Wang, Jiantao, Son, Sang H., and Mok, Aloysius K. Adaptive Co-Scheduling for Periodic Application and Update Transactions in Real-Time Database Systems. The Journal of Systems & Software 85.8 (2012): 17291743.

[7] Firebase Realtime Database. Firebase, Google, last retrieved July 13, 2019 from https://firebase.google.com/docs/database/

[8] Jian-Feng, Lu, Chun-Yi, Wang, and Jie, Hu. A High Performance Data Storage Method for Embedded Linux Real-Time Database in Power Systems. Energy Procedia 16 (2012): 883888.

[9] Ferrucci, Francesco, Bock, Stefan, and Gendreau, Michel. A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods. European Journal of Operational Research 225 (2013): 130141.

[10] Fernandes Ribeiro Neto, P., & Yao, J. (2012). Exploration on the application of real-time database in ship electronic information system. Applied Mechanics and Materials, 263-266, 1414.

[11] Han, Song et al. Adaptive Co-Scheduling for Periodic Application and Update Transactions in Real-Time Database Systems. The Journal of Systems & Software 85.8 (2012): 17291743.

[12] Sun, Q et al. Implementation of Massive Real-Time Database System Using Network Sensors and Sector Operation. Sensors & Transducers 174.7 (2014): 123128.

[13] S. Samaiya and M. Agarwal, "Real time database management system," 2018 2nd International Conference on Inventive Systems and Control (ICISC), Coimbatore, 2018, pp. 903-908.

[14] RTDB: A Memory Resident Real-Time Object Database. United States. Dept. of Energy. Office of Energy Research, 2003

[15] Huang, K., Zhang, J., & Yao, J. (2012). Exploration on the application of real-time database in ship electronic information system. Applied Mechanics and Materials, 263-266, 1414.

APPENDIX

A. Source Code for WebAppIndex.html

Listing 1. Web Application Source Code (WebAppIndexHTML.txt).

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Gas Station Project</title>
  <!-- update the version number as needed -->
  <script defer src="/__/firebase/6.3.4/firebase-app.js"></script>

  <!-- include only the Firebase features as you need -->
  <script defer src="/__/firebase/6.3.3/firebase-auth.js"></script>
  <script defer src="/__/firebase/6.3.3/firebase-database.js"></script>
  <script defer src="/__/firebase/6.3.3/firebase-messaging.js"></script>
  <script defer src="/__/firebase/6.3.3/firebase-storage.js"></script>
  <!-- initialize the SDK after all desired features are loaded -->
  <script defer src="/__/firebase/init.js"></script>

  <style media="screen">
    body {
      background: #ECEFF1;
      color: rgba(0, 0, 0, 0.87);
      font-family: Roboto, Helvetica, Arial, sans-serif;
      margin: 0;
      padding: 0;
    }

    #message {
      background: white;
      max-width: 360px;
      margin: 100px auto 16px;
      padding: 32px 24px;
      border-radius: 3px;
    }

    #message h2 {
      color: #ffa100;
      font-weight: bold;
      font-size: 16px;
      margin: 0 0 8px;
    }

    #message h1 {
      font-size: 22px;
      font-weight: 300;
      color: rgba(0, 0, 0, 0.6);
      margin: 0 0 16px;
    }

    #message p {
      line-height: 140%;
```

```
      margin: 16px 0 24px;
      font-size: 14px;
    }

    #message a {
      display: block;
      text-align: center;
      background: #039be5;
      text-transform: uppercase;
      text-decoration: none;
      color: white;
      padding: 16px;
      border-radius: 4px;
    }

    #message,
    #message a {
      box-shadow: 0 1px 3px rgba(0, 0, 0, 0.12), 0 1px 2px rgba(0, 0, 0, 0.24);
    }

    #load {
      color: rgba(0, 0, 0, 0.4);
      text-align: center;
      font-size: 13px;
    }

    @media (max-width: 600px) {

      body,
      #message {
        margin-top: 0;
        background: white;
        box-shadow: none;
      }

      body {
        border-top: 16px solid #ffa100;
      }
    }
  </style>
</head>

<body>

  <div id="message">
    <h2>Welcome to Finding the cheapest gas</h2>
    <iframe src=
    "https://www.google.com/maps/d/u/0/embed?mid=10HRl3h7TFWcfmgj03_hCgXQYe0qNGi49"
      width="320"
      height="480"></iframe>
    <p>GAS STATION: <b>Chevron</b></p>
    <!--<h1 id="ChevronHeader">CHEVRON: </h1>-->
    <input type="textfield" id="textChevron" />
    <button id="saveButtonChevron">Save</button>
    <button id="loadButtonChevron">Load</button>
    <p>GAS STATION: <b>Exxon</b></p>
```

```
    <input type="textfield" id="textExxon" />
    <button id="saveButtonExxon">Save</button>
    <p>GAS STATION: <b>Sinclair</b></p>
    <input type="textfield" id="textSinclair" />
    <button id="saveButtonSinclair">Save</button>
    <input type="textfield" id="textSinclairTimeStamp" />
  </div>
  <p id="load">Firebase SDK Loading&hellip;</p>

  <script>

    document.addEventListener('DOMContentLoaded', function () {

      // // The Firebase SDK is initialized and available here!
      //
      // firebase.auth().onAuthStateChanged(user => { });
      // firebase.database().ref('/path/to/ref').on('value', snapshot => { });
      // firebase.messaging().requestPermission().then(() => { });
      // firebase.storage().ref('/path/to/ref').getDownloadURL().then(() => { });
      //


      try {
        let app = firebase.app();
        let features =
          ['auth', 'database', 'messaging', 'storage'].filter(feature =>
            typeof app[feature] === 'function');
        document.getElementById('load').innerHTML =
            `Firebase SDK loaded with ${features.join(', ')}`;

        const config = {
          apiKey: "AIzaSyBW8cyEkjFX4vx3DtU1pl3ZxLqCndM5zWg",
          authDomain: "project-30a72.firebaseapp.com",
          databaseURL: "https://project-30a72.firebaseio.com",
          projectId: "project-30a72",
          storageBucket: "project-30a72.appspot.com",
          messagingSenderId: "500650925586",
          appId: "1:500650925586:web:820ac7156cb03576"
        };
        if (!firebase.apps.length) {
          firebase.initializeApp(config);
        }

        // Get a reference to the database service
        var database = firebase.database();
        getData();
        streamData();

      } catch (e) {
        console.error(e);
        document.getElementById('load').innerHTML =
          'Error loading the Firebase SDK, check the console.';
      }

    });

```

```
164     savePriceChanges();
165
166     function getData() {
167
168       loadButtonChevron.addEventListener('click', function() {
169         return
170           firebase.database().ref('Stations/001').
171             once('value').then(function(snapshot) {
172           var priceChevron = (snapshot.val().Price && snapshot.val().Price)
173           textChevron.value = priceChevron;
174           console.log ("Chevron Price = " + priceChevron);
175           })
176       })
177     }
178
179     function streamData() {
180
181       firebase.database().ref('Stations/002/Price').on('value', snap => {
182
183         console.log("EXXON CHANGED TO:" + snap.val());
184         textExxon.value = snap.val();
185
186       });
187
188       firebase.database().ref('Stations/003/Price').on('value', snap => {
189
190         console.log("SINCLAIR CHANGED TO:" + snap.val());
191         textSinclair.value = snap.val();
192
193       });
194
195       firebase.database().ref('Stations/003/Time').on('value', snap => {
196
197         console.log("SINCLAIR CHANGED TO:" + snap.val());
198         textSinclairTimeStamp.value = snap.val();
199
200       });
201
202     }
203
204     function savePriceChanges(){
205
206         saveButtonChevron.addEventListener('click', function() {
207           var timestamp = firebase.database.ServerValue.TIMESTAMP;
208           const docRef = firebase.database().ref('Stations/001');
209           const textToSave = textChevron.value;
210
211           console.log("CHEVRON NEW PRICE: " + textToSave)
212           firebase.database().ref('Stations/001').set({
213             Name: 'Chevron',
214             Price: textToSave,
215             Time: timestamp
216           }).then(function() {
217             console.log("Status saved!");
218           }).catch(function(error) {
219             console.log("YOU HAVE AN ERROR: ", error);
```

```
220             });
221         })
222 
223         saveButtonExxon.addEventListener('click', function() {
224           var timestamp = firebase.database.ServerValue.TIMESTAMP;
225           const docRef = firebase.database().ref('Stations/002');
226           const textToSave = textExxon.value;
227 
228           console.log("EXXON NEW PRICE: " + textToSave)
229           firebase.database().ref('Stations/002').set({
230             Name: 'Exxon',
231             Price: textToSave,
232             Time: timestamp
233           }).then(function() {
234             console.log("Status saved!");
235           }).catch(function(error) {
236             console.log("YOU HAVE AN ERROR: ", error);
237           });
238         })
239 
240         saveButtonSinclair.addEventListener('click', function() {
241           var timestamp = firebase.database.ServerValue.TIMESTAMP;
242           const docRef = firebase.database().ref('Stations/003');
243           const textToSave = textSinclair.value;
244 
245           console.log("SINCLAIR NEW PRICE: " + textToSave)
246           firebase.database().ref('Stations/003').set({
247             Name: 'Sinclair',
248             Price: textToSave,
249             Time: timestamp
250           }).then(function() {
251             console.log("Status saved!");
252           }).catch(function(error) {
253             console.log("YOU HAVE AN ERROR: ", error);
254           });
255         })
256 
257     }
258 
259   </script>
260 </body>
261 
262 </html>
```

B. Source Code for Android Application

Listing 2. gradle file that allows the app to implement SDKs such as Firebase and Google Services for Google Maps. (buildGradle.txt).

```
apply plugin: 'com.android.application'

apply plugin: 'kotlin-android'

apply plugin: 'kotlin-android-extensions'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.1"
    defaultConfig {
        applicationId "com.example.firebaseauthactivity"
        minSdkVersion 16
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        multiDexEnabled true
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles
            getDefaultProguardFile('proguard-android-optimize.txt'),
            'proguard-rules.pro'
        }
    }
}

android {
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.core:core-ktx:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    // SKDs
    implementation 'com.google.firebase:firebase-core:17.0.1'
    // Authentication
    implementation 'com.google.firebase:firebase-auth:18.1.0'
    // Cloud Firestore
    implementation 'com.google.firebase:firebase-firestore:20.2.0'
    // RTDB
    implementation 'com.google.firebase:firebase-database:18.0.1'
    // Google Play
    implementation 'com.google.gms:google-services:4.2.0'
    implementation 'androidx.multidex:multidex:2.0.0'
    //Google location services
```

```
    implementation 'com.google.android.gms:play-services-location:17.0.0'
    implementation 'com.google.android.gms:play-services-maps:17.0.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test:runner:1.2.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    implementation 'com.google.android.gms:play-services-maps:17.0.0'
    //user interface for searching data through firebase
    implementation 'androidx.recyclerview:recyclerview:1.0.0'
    implementation 'androidx.preference:preference:1.0.0'
    implementation 'com.firebaseui:firebase-ui-database:5.0.0'
    implementation 'androidx.recyclerview:recyclerview:1.1.0-beta02'
    implementation 'com.google.android.gms:play-services-maps:17.0.0'
    implementation 'com.google.android.gms:play-services-places:17.0.0'
    implementation 'com.google.android.libraries.places:places:2.0.0'
}

apply plugin: 'com.google.gms.google-services'
```

Listing 3. Provides functionality to the first page of the application allowing user to write to the firebase database (MainActivity.txt)

```
package com.example.firebaseauthactivity

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.RatingBar
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.FirebaseDatabase
import android.content.Intent
import android.util.Log
import android.view.View
import androidx.fragment.app.FragmentActivity
import kotlinx.android.synthetic.main.activity_main.*
import com.google.firebase.database.DatabaseError
import com.google.firebase.firestore.auth.User
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.ValueEventListener
import com.example.firebaseauthactivity.Gas as Gas1


class MainActivity : AppCompatActivity() {

    /* This next portion relates to the code on the the activity_main */
    lateinit var editTextName: EditText
    lateinit var editTextAddress: EditText
    lateinit var editTextPrice: EditText
    lateinit var ratingBar: RatingBar
    lateinit var buttonSave: Button
    lateinit var nextPage: Button


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        /* This next portion relates to the code on the the activity_main */
        editTextName = findViewById(R.id.editTextName)
        editTextAddress = findViewById(R.id.editTextAddress)
        editTextPrice = findViewById(R.id.editTextPrice)
        ratingBar = findViewById(R.id.ratingBar)
        buttonSave = findViewById(R.id.buttonSave)
        nextPage = findViewById(R.id.nextPage)


        // Submit button setting to perform a save
        buttonSave.setOnClickListener {
            saveGas()
        }

        //nextPage button setting to click to the next page
        nextPage.setOnClickListener() {
            val intent = Intent(this, ReadActivity::class.java)
            startActivity(intent)
        }
```

```
56      }
57
58
59          private fun saveGas() {
60              val name = editTextName.text.toString().trim()
61              val address = editTextAddress.text.toString()
62              val price = editTextPrice.text.toString()
63
64              if(name.isEmpty()) {
65                  editTextName.error = "Please enter a name"
66                  return
67              }
68              /* Write to the database */
69              val ref =
70                FirebaseDatabase.getInstance().getReference("Gas Station Prices")
71              val gasId: String? = ref.push().key
72              val gas =
73                gasId?.let { Gas1(name, address, ratingBar.numStars, price) }
74              if (gasId != null) {
75                  ref.child(gasId).setValue(gas)            }
76
77
78
79
80
81      }
82
83  }
```

Listing 4. Second page of the application containing Google Map (ReadActivity.txt)

```
package com.example.firebaseauthactivity

import android.content.Intent
import android.os.AsyncTask
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.Button
import android.widget.ListView
import android.widget.Toast
import androidx.recyclerview.widget.RecyclerView
import androidx.recyclerview.widget.RecyclerView.ViewHolder
import com.google.android.gms.maps.CameraUpdateFactory
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.OnMapReadyCallback
import com.google.android.gms.maps.SupportMapFragment
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.MarkerOptions
import com.google.firebase.database.*
import javax.annotation.Nonnull


class ReadActivity : AppCompatActivity(), OnMapReadyCallback {



    lateinit var prevPage: Button
    private lateinit var mMap: GoogleMap
    lateinit var listView: ListView
    lateinit var gasList: MutableList<readGas>
    lateinit var ref: DatabaseReference

    // map calling function?
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_read)
        //  Obtain the SupportMapFragment and get notified
        //  when the map is ready to be used.
        val mapFragment = supportFragmentManager
            .findFragmentById(R.id.map) as SupportMapFragment
        mapFragment.getMapAsync(this)

        gasList =  mutableListOf()
        ref =
      FirebaseDatabase.getInstance().getReference().child("Gas Station Prices")
        listView = findViewById(R.id.listView)
        ref.addValueEventListener(object: ValueEventListener {
            override fun onCancelled(p0: DatabaseError) {

            }

            override fun onDataChange(p0: DataSnapshot) {
                if(p0.exists()){
                    for(h in p0.children){
```

```
                        val gas = h.getValue(readGas::class.java)
                        gasList.add(gas!!)
                    }
                val adapter = gasAdapter(applicationContext, R.layout.gas, gasList)
                        listView.adapter = adapter
                    }
                }

        })


// Previous page button
        prevPage = findViewById(R.id.prevPage)


        prevPage.setOnClickListener() {
            val intent = Intent(this, MainActivity::class.java)
            startActivity(intent)


        }







    }


    override fun onMapReady(googleMap: GoogleMap) {
        mMap = googleMap

        // Add a marker in SMU and move the camera
        val SMU = LatLng(32.7767, -96.7970)
        mMap.addMarker(MarkerOptions().position(SMU).title("Marker at SMU"))
        mMap.moveCamera(CameraUpdateFactory.newLatLng(SMU))
        mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(SMU, 12.0f))

    }









}
```

Listing 5. Layout structure for placement of the editText and Button fields which are given functionality from the MainActivity code (ActivityMain.txt)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:orientation="vertical"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">
    <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="textPersonName"
            android:text="Station Name:"
            android:ems="10"
            android:id="@+id/editTextName"/>
    <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="textPostalAddress"
            android:ems="10"
            android:id="@+id/editTextAddress" android:text="Address: "/>
    <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="numberDecimal"
            android:ems="10"
            android:id="@+id/editTextPrice" android:text="Price:"/>
    <RatingBar
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:id="@+id/ratingBar"
            android:max="5"
            android:stepSize="1"/>
    <Button
            android:text="Submit"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" android:id="@+id/buttonSave"/>

    <Button
            android:text="Find Stations"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" android:id="@+id/nextPage"/>
</LinearLayout>
```

Listing 6. Lays out where the google map will be and where the database reads will be displayed (ActivityRead.txt)

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".ReadActivity" android:id="@+id/linearLayout">
    <Button
            android:text="Back to Input"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/prevPage" app:layout_constraintTop_toTopOf="parent"
            android:layout_marginTop=
            "16dp" app:layout_constraintEnd_toEndOf=
            "parent" android:layout_marginEnd="8dp"
            android:layout_marginRight="8dp"
            app:layout_constraintBottom_toBottomOf="parent"
            android:layout_marginBottom="8dp"
            app:layout_constraintVertical_bias="1.0" />
    <fragment xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:tools="http://schemas.android.com/tools"
              xmlns:map="http://schemas.android.com/apk/res-auto"
              android:layout_width="match_parent"
              android:layout_height="300dp"
              android:id="@+id/map"
              tools:context=".mapsHELP"
              android:name="com.google.android.gms.maps.SupportMapFragment"
              map:layout_constraintTop_toTopOf="parent"
              map:layout_constraintEnd_toEndOf="parent" />
    <fragment
            android:id="@+id/place_autocomplete_fragment"
            android:name=
           "com.google.android.gms.location.places.ui.PlaceAutocompleteFragment"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintEnd_toEndOf="parent" />
    <ListView
            android:layout_width="match_parent"
            android:layout_height="200dp"
            app:layout_constraintBottom_toBottomOf="parent"
            android:layout_marginBottom="56dp"
            app:layout_constraintStart_toStartOf="parent"
            android:id="@+id/listView" />


</androidx.constraintlayout.widget.ConstraintLayout>
```

Listing 7. Lays out where the google map will be and where the database reads will be displayed(readGasFunction.txt)

```
package com.example.firebaseauthactivity;

public class readGas {
    private String address;
    private String name;
    private String price;


    public readGas() {
    }

    public readGas(String address, String name, String price) {
        this.address = address;
        this.name = name;
        this.price = price;

    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getName() {
        return name;
    }

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

    public String getPrice() {
        return price;
    }

    public void setPrice(String price) {
        this.price = price;
    }


}
```

Listing 8. Called by MainActivity to allow input of data (address, name, price) and mapped to Android Screen through ActivityMain.xml (gasFunction.txt)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="match_parent"
              android:layout_height="match_parent">

    <TextView
            android:textAppearance="@style/TextAppearance.AppCompat.Large"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:id="@+id/textViewName" />
</LinearLayout>
```