**Project Title:** Firebase – Price Watcher

**App Details**: The project is mainly the homework 3 conversion from SQLite database to Firebase Realtime database.

**Project Requirements**:

1. User authentication using Firebase Authentication option.
   1. Registered user login.
   2. New user signup.
   3. Forget password, reset request.
2. Store item data on Firebase Realtime database.
   1. Add / Update / Delete items.
3. Additional requirements implemented from homework 3.
   1. Fetch item price from internet from at least 2 online stores and UTEP test website. This network process will run on a separate thread. The app will either return the price or inform the user price cannot be found.
   2. Awareness of network connectivity, the app shall route the user to Wi-Fi settings if no internet is available.

**Why this app**?

I have decided to go ahead with this project because I wanted to learn something new that we have not covered in class and share the findings with the whole class.

**New Features Learned**:

I have learned how to use Firebase as a backend service instead of storing the data on the device. The main advantage using Firebase is the “information availability”, any device you login will fetch your data from the cloud. Secondly, the implementation in terms of lines of code is far less and easier than SQLite.

**Rough schedule**:

* Planning, designing and research phase (1 day)
* Firebase Authentication setup and Testing (1day)
* Firebase Realtime Database setup and Testing (1day)
* Report writing and Slides (1 days)

**Solution: Step 1: Connecting Project to Firebase:**

First step in the project development was to add Firebase to the project, how my android app will talk to the Firebase backend service. You can do this manually by adding lines of code to your *Project level gradle* file and *App level gradle* file. Or, you can let Android Studio add these lines of code for you automatically. There are advantages and disadvantages for both. By adding manually, first of all you know where the lines are added and what versions of API you want to use. Adding automatically, you do not need to find where to add what, Android Studio does it for you but in case if the API versions are not compatible with what you are trying to do, you will need to make the changes manually again.

**Manual Process**: Login firebase console with your *Google ID* and add a new project. You will be guided step by step on what you need to do, it is a simple process. However, the lines of code you will be provided

to add in the gradle files, change “***compile***” to “***implementation***”.

**Automatic Process:** From Android Studio, click Tools -> Firebase. A new pane will open on the right side of screen giving you options to add Firebase features to your project. Each feature you select, first step will to connect the app to firebase. A new window will open, and you can sign-in with your *Google ID* and either create a new project or connect to existing project.

Based on the project requirements, this process enables and provides you the code required for a specific firebase feature that you are trying to implement.

**Solution: Step 2: Adding User Authentication:** Just like step 1, you can add User Authentication related libraries to your

project from Firebase option in the Tools menu. Section 3 – 6 under User Authentication provides you the code you need to perform authentication operations. The 1st requirement can be completely satisfied in this step.

**Solution: Step 3: Adding Database:** This step is critical in terms of understanding the architecture. There are two things to consider in this step.

1. **Database Structure**: How your data will be structured.
2. **Database Rules**: Who will access your database and how.

**Database Structure**: In a normal relational database including SQLite, the table structure looks like

|  |  |
| --- | --- |
| **ID** | **Autoincrement (Primary Key)** |
| **ItemName** | String |
| **URL** | String |
| **Price** | REAL |
| **Change** | String |

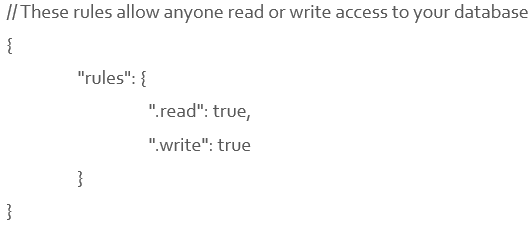
🡨 ID is primary key and automatically generated

And the data is stored in the table like

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **ItemName** | **URL** | **Price** | **Change** |
| **1** | UTEP Test Item | http://www.cs.utep.edu/cheon/... | 303.17 | Price dropped 30% |
| **2** | Amazon Item | https://www.amazon.com/dp/... | 329.99 | Newly Added Item |

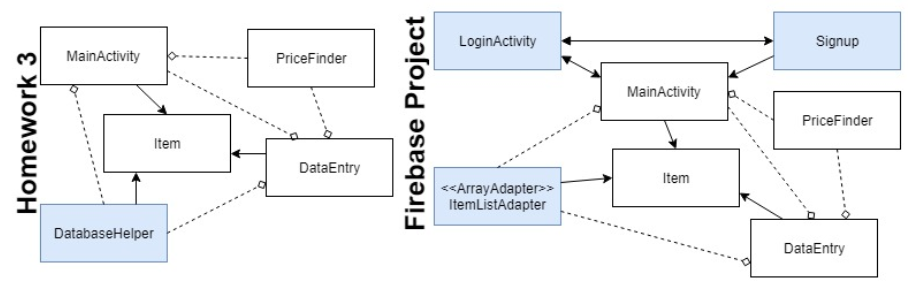
|  |  |
| --- | --- |
| However, in Firebase, data is stored in JSON tree structure. The same data will look something like 🡪  Since Firebase does not automatically generate a unique id for each node, it provides a function to generate unique id for each node: **FbDbRef.push().getKey()**. |  |

**Database Rules**: Rules can be customized from a minimum to maximum security on who can read/write to a database. In the example below, anyone can read and write to the database.



**Transformation of Homework 3 from SQLite to Firebase**: In case of SQLite, a database helper class is used to perform database operations along with SimpleCursorAdapter. In case of using Firebase, we do not need the DatabaseHelper class

since database operations are very simple, we just need one ArrayAdapter. The UML design (w/o Activities) is as follows:



**Problems / Issues / Challenges faced**:

1. **Gradle terms and dependencies**: Some of the terms in the provided code for gradle files have changed. E.g. “***compile***” to “***implementation***”, “***testCompile***” to “***testImplementation***”. Firebase automatically adds the latest library files to your project but depending on your project needs, you may need to find a suitable version of libraries for your project. In my case, I was able to perform smooth operations on Android tools to 3.1.3, Google play services to 4.0.1 and firebase to 10.0.1.
2. **App to database communication issue:** After database setup, you need to enable “***Identity Toolkit API***” without this, your app will not be able to communicate with the database. To do this, use the following link:

https://console.developers.google.com/apis/library/identitytoolkit.googleapis.com/

1. **App always signed in:** If you do not add the logout code on the right places, the connection to Firebase cloud remains open and your app skips login when launched next time. Logout: **FirebaseAuth.getInstance().signOut();**
2. **Database Options under Firebase:** Firebase provides two options under Database section on web console “***Cloud Firestone***” and “***Realtime Database***”. By default Cloud Firestone is selected, change it to Realtime Database. Rules and data storage structure is different for Cloud Firestone.
3. **Firebase project limit on portal:** There is a limit for the projects you can create on free account. Even if you delete a project, it is still counted under the allowed limit so better to change project settings without deleting it.
4. **Additional consoles related to your Android/Firebase activities:**

**APIs**: https://console.developers.google.com/apis/dashboard?project=fir-project-a74ff

**Google Cloud Platform**: https://console.cloud.google.com/home?project=fir-project-a74ff

**Firebase**: https://console.firebase.google.com/project/fir-project-a74ff/

**Findings:**

* Add / Update / Delete is extremely easy with Firebase.
* Platforms supported by Firebase are Android, iOS, Web, C++ and Unity.
* Data is stored on the cloud that can be accessed from any other device after login. No worries about data.
* Security can be customized from minimum to maximum as per the requirements.
* In case of SQLite, data is stored offline and can be viewed however, for Firebase, active internet is a must.
* Data retrieval time is good but depending on your internet connection, retrieval can be slower.
* **Matching dependencies can be a problem. Every firebase service requires added dependencies to your project.**

**Useful Links:**

**Firebase Features**: https://firebase.google.com/

**Firebase Pricing**: https://firebase.google.com/pricing/

**Firebase Docs**: https://firebase.google.com/docs/

**DB Rules**: https://angularfirebase.com/lessons/understanding-firebase-database-rules-by-example/

**Firebase vs SQLite:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Firebase Realtime Database** | **SQLite** |
| **Description** | Cloud-hosted real-time document store. NoSQL database. iOS, Android, and JavaScript clients share one Realtime Database instance and automatically receive updates with the newest data. | local database on Android device with SQL interface. Widely used in-process RDBMS |
| **Primary database model** | Document store | Relational DBMS |
| **Developer** | Google | Dwayne Richard Hipp |
| **Initial release** | April, 2012 | 2000 |
| **License** | Commercial, comfortable for rapid development third-party service. | Open Source, independent, absolutely free. |
| **Cloud-based** | Yes. A distributed database and it gives real-time streaming support which means a continual input, process and output of data. | No. Not a distributed database where if you want to share data between multiple users you can’t do it with SQLite. |
| **Server operating systems** | hosted | server-less |

|  |  |  |
| --- | --- | --- |
| **Data scheme** | schema-free | yes |
| **SQL** | no | yes |
| **APIs and other access methods** | Android iOS JavaScript API RESTful HTTP API | ADO.NET  JDBC  ODBC |
| **Supported programming languages** | Java JavaScript Objective-C | Actionscript, Ada, Basic, C, C#, C++, D Delphi, Forth, Fortran, Haskell, Java, JavaScript, Lisp Lua, MatLab, Objective-C, OCaml Perl, PHP, PL/SQL, Python, R, Ruby, Scala, Scheme, Smalltalk, Tcl |
| **Server-side scripts** | limited functionality with using 'rules' | no |
| **Foreign keys** | no | yes |
| **Concurrency** | yes | yes |
| **Durability** | yes | yes |