Use Databases other than TinyWebDB with App Inventor: Tying the knot with ColdFusion

Alan Hau
Department of Computer Science
Cornell University
Ithaca, NY
United States

alan@firstcodeacademy.com

I am an instructor of K-12 students at First Code Academy, and when students want to prototype a new App Inventor app, I only have to make them a new table in my DB and some simple changes in ColdFusion (CFML) scripts for their procedures. The key benefits of this approach include the ease of management, streamlined security and minimal coding to expose the API.

NoSQL DB (TinyDB) vs. Relational DB (MySQL)

TinyWebDB is a great way to store key/value entries on the cloud but there are times a user wants more privacy and structure than saving all her data on App Inventor's public DB. One of the most important limitation is that the user doesn't know if her key or tag is unique when she first uses this on the DB. This is troublesome when users follow the same tutorial where the keys share similar naming conventions. One way around this is to install new instances of TinyWebDBs but this is a waste of resources. It's also difficult to use any SQL statements on the data. Selecting a sorted list is easily done in SQL but very difficult with App Inventor's language alone.

Another challenge in connecting with SQL is to accommodate the many brands and flavors in the market. Perhaps the user wants to connect to an existing database at work that cannot be changed. A user can try to write a web API and return JSON data for their different DBs but taking up such projects at whim could be daunting.

Problems with Ad Hoc Solutions

If the developer only wanted to connect to his own database, then one of the solutions could be to write an API in PHP. The syntax is also relatively easy to follow and tutorials are readily available on the web. However, as the project grows, some of the problems become apparent.

Database logins are sensitive material, it shouldn't be exposed as cleartext in code. If the developer is looking for tutorials online for a simple script, security details are usually overlooked.

Suppose the developer needs to communicate with multiple databases. Administration becomes a nightmare. Different code snippets in multiple folders make maintenance very difficult. A single bug or feature change would trigger multiple changes all over the place.

SQL injection attacks are also common with poorly written code and it is common for junior developers just looking for a proof of concept. This happens when users introduce malicious code when communicating with your web API to perform unintended operations (eg. delete a table or viewing all username and passwords). These overlooked points will become very troublesome when a breach happens and affects the users.

Using ColdFusion

I'm using an open source solution offered by OpenBD (Railo also works well). Once you have your favorite SQL scripts wrapped in a stored procedure, use CFML to expose the stored procedure securely and return any data in JSON format. The syntax is simple and the flexibility is high.

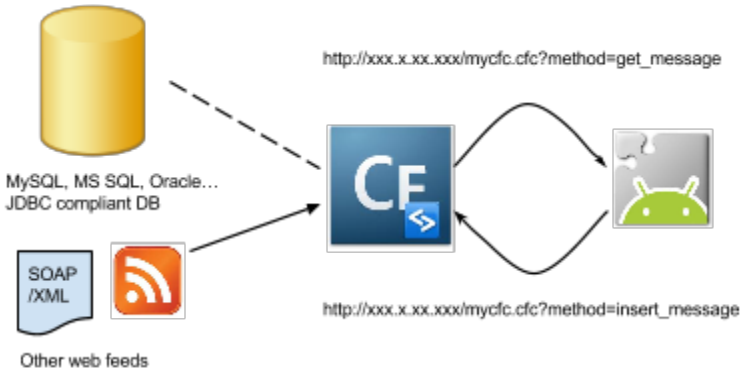

Figure 1

Out of the box, the OpenBD and Railo distributions can be run on Windows / Mac or Linux with a web server embedded in the installation. There are no additional configuration or dependent installations needed to start running this service.
ColdFusion is essentially a background service that runs alongside your web server that has its own admin interface accessible via a webpage. You can register your databases and other web feeds on this interface. Any passwords stored are encrypted.

Once these credentials are stored, the database will be given a tag that refers to the database. Now I can safely share my code with others without revealing any sensitive login information. Suppose I have a simple stored procedure written here which selects and sorts the students' age descendingly.

id	name	age	gender
1	Alice	12	female
2	Bob	8	male
3	Charles	15	male
NULL	NULL	NULL	NULL

Variations of database operations can also be wrapped in the procedure, such as INSERT, UPDATE, DELETE... etc.

Figure 2

```
7  CREATE DEFINER=`root`@`localhost` PROCEDURE `get_agebygender`(igender varchar(45))
8  BEGIN
9      select `name`, `age`, `gender` from AJDB.StudentAge where gender = igender
10     order by age desc;
11 END
```

Figure 3

```
<cfcomponent>
    <!-- cffunction name is the API method's name -->
    <cffunction name="get_agebygender" access="remote" returnformat="json">

        <!-- Specify API arguments here -->
        <cfargument name="gender" type="string" required="true"/>

        <!-- Specify Database Prodcedure arguments here -->
        <cfstoredproc procedure="get_agebygender" datasource="AJDB">
            <cfprocparam cfsqltype="cf_sql_varchar" value="#gender#">
            <cfprocresult name="qcf">
        </cfstoredproc>

        <!--- this is what's returned when the API is called --->
        <cfreturn qcf>

    </cffunction>
</cfcomponent>
```

Database reference used, no credentials exposed

Figure 4

Once this short script is saved as a .cfc file in the server, it can be tested immediately via your favorite browser in any device: http://119.9.76.xxx/myapp.cfc?method=get_agebygender&gender=male

On App inventor

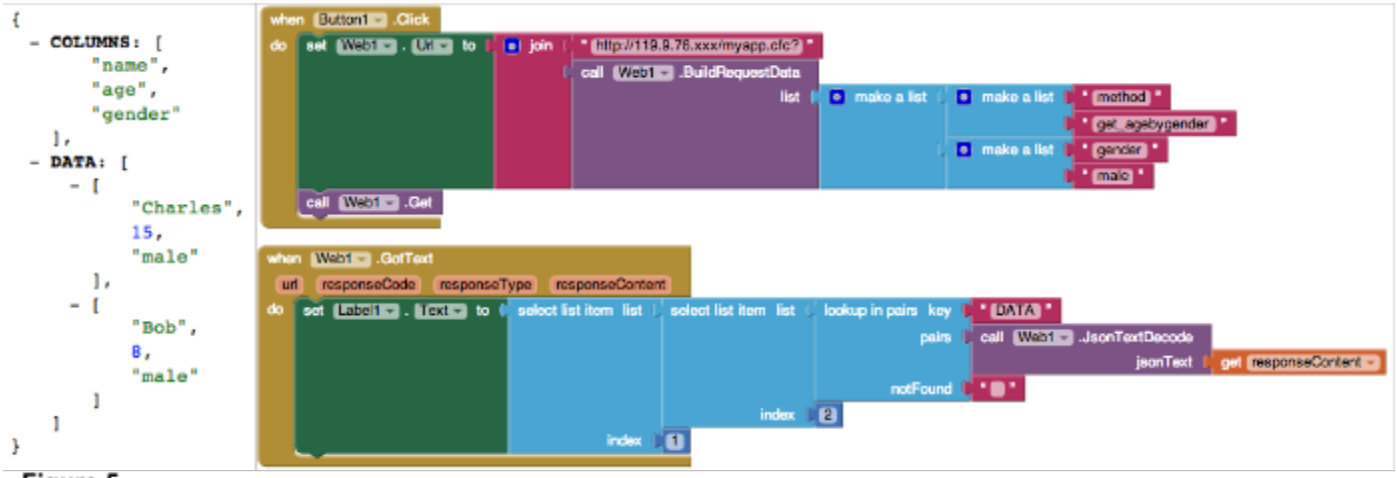

Figure 5

This example converts the JSON string to a list and then displays “Bob” in Label1.Text

Other ways to use CFML and App Inventor

Using CFML, App Inventor features can be greatly extended. I can now rapidly insert geolocation data and even create a simple chat client. App Inventor can also send emails via a web API call. NoSQL plugins for the popular MongoDB is also available for those who want a tag based database solution. With security handled internally and simple scripts to connect with the database, my students now spend less time reinventing the wheel and more time creating features for their apps.