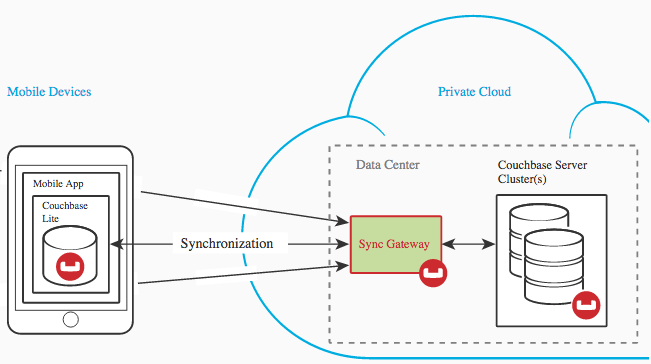
**Backend Architecture**

The figure below illustrates how the components mentioned above work together. Specifically, this section will provide details on how Smart waiter utilizes this architecture.



**Backend Components**

As the following figure illustrates, the backend of the application is designed using Couchbase Lite, Sync Gateway and Couchbase server cluster.

**Couchbase Lite**

Couchbase Lite is a fully functional local database, which runs on the mobile device. This is a local storage system, which is integrated into SmartWaiter application. Similar to the Couchbase Server, Couchbase Lite uses JSON data format, which allows for a flexible data model. Algorithms within SmartWaiter application will be responsible for mapping the JSON data to the application objects. In addition, this solution decouples application data and network connectivity. Thusly, the local data is always available if there is a network connectivity issue or the Couchbase server is down.

**Sync Gateway**

As the figure above illustrates, the sync gateway sits in the middle of the couchbase server and Couchbase Lite running on the users mobile device. This is a key component of the architecture. The sync gateway is responsible for synchronizing the local database with the cloud. Mobile devices only make changes to their local database, and sync gateway takes care of the rest. The gateway will implement authorization for users, which will allow for access control. In addition, channels will be used for access control (details later). Most importantly, the gateway runs Smartwaiter business logic to regulate the synchronization and access control.

**Couchbase Server Cluster**

Couchbase server is the component, which sits in the private cloud along with the sync gateway. As previously described the sync gateway only acts as the manager and administers the traffic between mobile devices and the Couchbase server. The Couchbase server itself is responsible for all data storage in the cloud. The server also uses same JSON data format as Couchbase Lite. For the purpose and scale of the SmartWaiter project, only one server node will be used to hold all data. More importantly, the server will run on a local machine for demonstration purposes.

**Couchbase Lite – Detailed Design**

Creating local database

A top-level manager object is created when the application launches, which manages collection of couchbase instances. In the case of this application, there are two database instances, one that holds menus and another to hold orders. The Manager creates a directory in the filesystem (if one does not already exist) and stores the databases inside it. The file path is determined by the application context. Thusly, if the application is ever deleted, all of the content will be deleted as well. In addition, creating the database at this point makes it readily available for functions, which might try to use it immediately following the application’s launch.

Code

**public** CouchBaseLite(MainActivity AndroidContext) **throws** IOException, CouchbaseLiteException {  
AndroidContext.**manager** = **new** Manager;  
 AndroidContext.**database** = **manager**.getDatabase(“menus”);  
 print(**"Created Couch Base Lite database"**);  
}

**Accessing local database**

The CouchbaseLite API is used to access the documents by barcode. Furthermore, the couchbase get API is used to extract all menu details. An example of extracting a menu from the database is given below.

Code

**public** Document getRestaurantByBarcode(String barCode){  
 restaurantMenu = getDocument(barCode);  
 **return** restaurantMenu;  
}

**Replications**

Couchbase API will be used to replicate data from the cloud. For the scope of the application a replication of the whole database is used. However, in the future when there are thousands of restaurants, a filter can be used to pull specific menus from the cloud on demand. Both a continuous pull and push sync are setup so the application can push and pull at any time the application is running. In addition, a change listener is set on each pull to confirm the status.

**public void** startReplications() **throws** CouchbaseLiteException, MalformedURLException {

Replication push = database.createPushReplication(url); Replication pull = database.createPullReplication(url); pull.setContinuous(true); push.setContinuous(true);  
  
pull.addChangeListener(ChangeListener() {  
 @Override  
 **public void** changed(ChangeEvent event) {  
 *// will be called back when the pull replication status changes* **if** (getStatus() == ***REPLICATION\_IDLE***) {  
 print(**"The replication is complete"**);  
 } **else** {  
 print(**" The replication Failed"**);  
 }  
 }  
 });

}

**Sync Gateway – Implementation Details**

The Sync Gateway requires some basic configuration to correctly synchronize mobile devices and the Couchbase server. The basic configuration is given below. Firstly, the sync gateway is pointed towards the Couchbase server node and the data bucket, which holds menus and orders. Furthermore, the Sync gateway is assigned a “sync\_data” bucket. This bucket is necessary because the sync gateway needs to store meta-data for accessed documents, such as revision etc. Since we want to decouple the sync-gateway from the Couchbase server and the restaurant menu’s and orders data. A new bucket is created for storing sync gateway metadata.

"couchbaseevents": {

"server":"http://127.0.0.1:8091",

"bucket":"Smartwaiter\_menus",

"shadow": {

"server": "http://127.0.0.1:8091",

"bucket": "sync\_data"

},

“bucket”:”orders”

"shadow": {

"server": "http://127.0.0.1:8091",

"bucket": "sync\_data"

}

"import\_Docs": true,

"sync":`

function (doc) {

channel (doc.channels);

}

**Couchbase server – Implementation detail**

The couchbase server uses some basic rules to store data on the cloud. Here are some of the following;

* Data is stored as key-value pairs, where the key is the barcode and the value is the entire menu of the restaurant.
* Buckets are basically equivalent of a database.
* Each item in the database is referred to as a document.
* The database is schema-less, thusly there is no rigid structure

JSON data format

As mentioned above the database is schema less, meaning the JSON format and attributes can change from one restaurant to another. However below is an overview of the general JSON formatted restaurant menu.

Example JSON document

{

"Res\_Name": "Habibiz",

"category": [

{

"type": "Appetizers",

"url": "https://blah.com/pic"

},

{

"type": "Drinks",

"url": "https://blah.com/pic"

},

{

"type": "Main Course",

"url": "https://blah.com/pic"

},

{

"type": "Dessert",

"url": "https://blah.com/pic"

}

],

"Appetizers": [

{

"name": "Buffalo Chicken Dip",

"price": "12.99",

"details": "Buffalo wing sauce, cream cheese and ranch make a great party dip."

},

{

"name": "Antipasto Kabobs",

"price": "9.99",

"details": "Stewed ground beef with minced potatoes"

}

],

"Drinks": [

{

"name": "Coke",

"price": "1.99",

"details": "Served in a iced glass"

},

{

"name": "beer",

"price": "6.99",

"details": "12oz pitcher"

}

],

"Main Course": [

{

"name": "Minced Beef Mushroom",

"price": "12.99",

"details": "Ground beef with soup-cream of Mushrooms"

},

{

"name": "Beef diced potatoes",

"price": "10.99",

"details": "Stewed ground beef with minced potatoes"

}

],

"Dessert": [

{

"name": "Plum Pudding Cake",

"price": "8.99",

"details": "This old-fashioned dessert has a sweet cake layer on top and a juicy plum pudding filling beneath! "

},

{

"name": "Blackberry Coffee Cake",

"price": "10.99",

"details": "A quick coffee cake with blackberries folded in and a cinnamon crumb topping!"

}

]

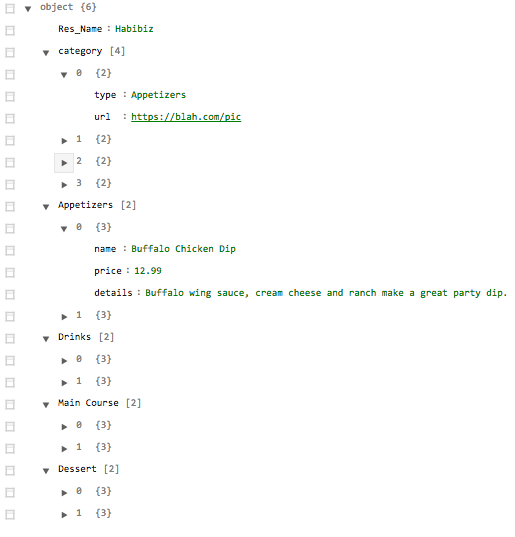
}

JSON Structure

As mentioned earlier this structure is not strictly enforced and can be changed for certain restaurants. Granted that the Smartwaiter application will need to update the parser. This is a generic structure which the parser currently handles and supports majority of the restaurants.

Breakdown details

Where “{ }” braces represent JSON objects and “[ ]” represent array.



Structure breakdown figure

