NOTE: The design for the pieces of code I wrote for the code review has been significantly changed and refactored, so it would be difficult to track how feedback on the original code review led to the new design (we changed the design within a few days of the code review). So, here is the current versions of the classes that were included in my code review.

public interface Retriever {  
 Content getAll();  
}

*/\*\**  
 *\* Retrieves a single type of DatabaseObject.*  
 *\*/*  
public class SingleRetriever<T extends DatabaseObject> implements Retriever {  
 private RetrieverSchema schema;  
 private boolean testMode = false;  
  
 public RetrieverSchema getSchema() {  
 return schema;  
 }  
  
 public SingleRetriever(RetrieverSchema rSchema) {  
 schema = rSchema;  
 }  
  
 public SingleRetriever(RetrieverSchema rSchema, boolean testMode) {  
 schema = rSchema;  
 this.testMode = testMode;  
 }  
  
 public Content<T> getAll() {  
 Content<T> content;  
  
 if (testMode) {  
 return getTestContent();  
 } else if ((content = getLiveContent()).getType() == ContentType.LIVE) {  
 Log.e("RETRIEVAL", "getting live content");  
 return content;  
 } else {  
 Log.e("RETRIEVAL", "getting cached content");  
 return getCachedContent();  
 }  
 }  
  
 */\*\**  
 *\* Tries to get content from the database, returns null if the database was unreachable.*  
 *\*/*  
 private Content<T> getLiveContent() {  
 JsonArray json;  
 List<T> objects = new ArrayList<T>();  
  
 Class objClass = schema.getObjClass();  
 Constructor constructor = null;  
 try {  
 constructor = objClass.getConstructor(JsonObject.class);  
 } catch (NoSuchMethodException ex) {  
 *// ERROR getting constructor from the provided class*  
 ex.printStackTrace();  
 }  
  
 json = RestUtil.get(schema.getTableName());  
  
 if (json != null) {  
 for (JsonElement jsonElement : json) {  
 T dbObject = getObjectFromJson(jsonElement, constructor);  
 objects.add(dbObject);  
 }  
  
 *// cache new live data*  
 Cache cache = Cache.getCacheForObjectType(schema.getObjClass());  
 if (cache != null) {  
 new CacheTool<T>().cache(cache, objects);  
 }  
  
 return new Content<>(objects, ContentType.LIVE);  
 } else {  
 return new Content<T>(new ArrayList<T>(), ContentType.CACHED);  
 }  
 }  
  
 */\*\**  
 *\* Tries to get cached data of the correct type, returns null if there*  
 *\* was an error or if there was no cached data*  
 *\*/*  
 private Content<T> getCachedContent() {  
 Cache cache = Cache.getCacheForObjectType(schema.getObjClass());  
 if (cache != null) {  
 return new CacheTool<T>().uncache(cache);  
 } else {  
 return new Content<>(null, ContentType.CACHED);  
 }  
 }  
  
 private Content<T> getTestContent() {  
 return new Content<>(null, ContentType.TEST);  
 }  
  
 private T getObjectFromJson(JsonElement jsonElement, Constructor constructor)  
 {  
 if (jsonElement.isJsonObject())  
 {  
 try {  
 return (T)constructor.newInstance(jsonElement.getAsJsonObject());  
 } catch (InstantiationException ex) {  
 *// ERROR instantiating database object*  
 *// Note: Can't multi-catch for reflection because we support old phones*  
 ex.printStackTrace();  
 } catch (IllegalAccessException ex) {  
 *// ERROR instantiating database object*  
 ex.printStackTrace();  
 } catch (InvocationTargetException ex) {  
 *// ERROR instantiating database object*  
 ex.printStackTrace();  
 }  
 }  
 return null;  
 }  
  
}

public abstract class DatabaseObject {  
  
 protected JsonObject fields;  
 private ImageData imageData;  
  
 *//for testing...*  
 public DatabaseObject(String id) {  
 fields = new JsonObject();  
  
 fields.add(Database.JSON\_KEY\_COMMON\_ID, new JsonPrimitive(id));  
 }  
  
 public DatabaseObject(JsonObject obj) {  
 update(obj);  
 }  
  
 public void update(JsonObject obj) {  
 fields = obj;  
  
 if (fields.has(Database.JSON\_KEY\_COMMON\_IMAGE)) {  
 imageData = new ImageData(fields.get(Database.JSON\_KEY\_COMMON\_IMAGE));  
 }  
 }  
  
 public void setField(String key, JsonElement element) {  
 fields.add(key, element);  
 }  
  
 */\*\**  
 *\* Gets a field from this object as a String. If the field can't be represented as a String*  
 *\* or if it does not exist, returns null.*  
 *\*/*  
 public String getFieldAsString(String fieldName) {  
  
 JsonElement fieldValue = fields.get(fieldName);  
  
 if (fieldValue != null && fieldValue.isJsonPrimitive()) {  
 return fieldValue.getAsString();  
 } else {  
 return null;  
 }  
 }  
  
 */\*\**  
 *\* Gets a field from this object as a Double. If the field can't be represented as a Double*  
 *\* or if it does not exist, returns null.*  
 *\*/*  
 public Double getFieldAsDouble(String fieldName) {  
 String field = getFieldAsString(fieldName);  
  
 if (field != null) {  
 return Double.parseDouble(field);  
 } else {  
 return null;  
 }  
 }  
  
 */\*\**  
 *\* Gets a field from this object as a Integer. If the field can't be represented as a Integer*  
 *\* or if it does not exist, returns null.*  
 *\*/*  
 public Integer getFieldAsInt(String fieldName) {  
 String field = getFieldAsString(fieldName);  
  
 if (field != null) {  
 return Integer.parseInt(field);  
 } else {  
 return null;  
 }  
 }  
  
 */\*\**  
 *\* Returns the corresponding JsonElement for key fieldName, or null if that key does not exist.*  
 *\*/*  
 public JsonElement getField(String fieldName) {  
 return fields.get(fieldName);  
 }  
  
 public HashMap<String, JsonElement> getJsonEntrySet() {  
 HashMap<String, JsonElement> map = new HashMap<>();  
 for (Entry<String, JsonElement> entry : fields.entrySet()) {  
 if (entry.getValue().isJsonArray()) {  
 map.put(entry.getKey(), entry.getValue().getAsJsonArray());  
 } else {  
 map.put(entry.getKey(), entry.getValue());  
 }  
 }  
 return map;  
 }  
  
 public String getId() {  
 return getFieldAsString(Database.JSON\_KEY\_COMMON\_ID);  
 }  
  
 public String getName() {  
 return getFieldAsString(Database.JSON\_KEY\_COMMON\_NAME);  
 }  
  
 */\*\**  
 *\* DEPRECATED: USE getImageData*  
 *\* Gets the url for the image associated with this DatabaseObject*  
 *\*/*  
 public String getImage() {  
 if (imageData != null) {  
 return imageData.getUrl();  
 }  
  
 return null;  
 }  
  
 public ImageData getImageData() {  
 return imageData;  
 }  
  
 public String getDescription() {  
 return getFieldAsString(Database.JSON\_KEY\_COMMON\_DESCRIPTION);  
 }  
  
 @Override  
 public boolean equals(Object other) {  
 DatabaseObject dbObj;  
  
 if (other instanceof DatabaseObject) {  
 dbObj = (DatabaseObject) other;  
 return this.getId().equals(dbObj.getId());  
 }  
  
 *//for comparing ids directly*  
 else if (other instanceof String) {  
 return this.getId().equals(other);  
 }  
  
 return false;  
 }  
}

CruCardScreen has been removed

public class Campus extends DatabaseObject {  
 private Location location;  
 private String websiteUrl;  
  
 public Campus(JsonObject obj) {  
 super(obj);  
  
 websiteUrl = this.getFieldAsString(Database.JSON\_KEY\_CAMPUS\_URL);  
 location = new Location(this.getField(Database.JSON\_KEY\_COMMON\_LOCATION));  
 }  
  
 public String getWebsiteUrl() {  
 return websiteUrl;  
 }  
  
 public Location getLocation() {  
 return location;  
 }  
}

The use of different implementations of SingleRetriever for each type of database object has been changed: the following classes are the main ones used in database object retrieval, which supports caching (the less important ones can be viewed on our bitbucket repository).

public enum RetrieverSchema {  
 CAMPUS (Campus.class, Database.REST\_CAMPUS),  
 MINISTRY (Ministry.class, Database.REST\_MINISTRY),  
 EVENT (Event.class, Database.REST\_EVENT),  
 SUMMER\_MISSION (SummerMission.class, Database.REST\_SUMMER\_MISSION),  
 RIDE (Ride.class, Database.REST\_RIDE),  
 RESOURCE(Resource.class, Database.REST\_RESOURCE),  
 MINISTRY\_TEAM (MinistryTeam.class, Database.MINISTRY\_TEAM);  
  
 public Class<? extends DatabaseObject> getObjClass() {  
 return objClass;  
 }  
  
 public String getTableName() {  
 return tableName;  
 }  
  
 private final Class<? extends DatabaseObject> objClass;  
 private final String tableName;  
  
 RetrieverSchema(Class<? extends DatabaseObject> objClass, String tableName) {  
 this.objClass = objClass;  
 this.tableName = tableName;  
 }  
}

*/\*\**  
 *\* Caches objects of a certain type*  
 *\*/*  
public class CacheTool<T extends DatabaseObject> {  
 */\*\**  
 *\* Caches a list of database objects to a file, specified by the cache*  
 *\*/*  
 public boolean cache(final Cache cache,  
 final List<T> list) {  
 List<String> stringList = new ArrayList<>();  
 for (T obj : list) {  
 HashMap<String, JsonElement> map = obj.getJsonEntrySet();  
 for (String key : map.keySet()) {  
 stringList.add(key + LocalStorageIO.HASHMAP\_DELIMITER + map.get(key));  
 }  
 stringList.add(" ");  
 }  
 return LocalStorageIO.writeList(stringList, cache.fname);  
 }  
  
 */\*\**  
 *\* Gets cached data from a cache, or null if there was an error*  
 *\*/*  
 public Content<T> uncache(final Cache cache) {  
 List<String> stringList = LocalStorageIO.readList(cache.fname);  
 List<T> objects = new ArrayList<>();  
 if (stringList != null) {  
 *// get constructor for the type of cached object*  
 Constructor<T> constructor = null;  
 try {  
 constructor = cache.objectType.getConstructor(JsonObject.class);  
 } catch (NoSuchMethodException ex) {  
 Log.e("Cache Error", "Could not get JSON constructor for " + cache.objectType);  
 } finally {  
 if (constructor == null) {  
 Log.e("Cache Error", "Error getting constructor for " + cache.objectType);  
 return null;  
 }  
 }  
 JsonObject json = new JsonObject();  
 *// create objects*  
 for (String string : stringList) {  
 if (string.trim().length() > 0) {  
 *// add fields to current object*  
 String[] line = string.split("\\" + LocalStorageIO.HASHMAP\_DELIMITER);  
 json.add(line[0], new JsonParser().parse(line[1]));  
 } else {  
 *// object finished*  
 try {  
 objects.add(constructor.newInstance(json));  
 } catch (Exception ex) {  
 Log.e("Cache Error", "Could not create " + cache.objectType + " object");  
 }  
 json = new JsonObject();  
 }  
 }  
 return new Content<>(objects, ContentType.CACHED);  
 } else {  
 Log.e("Cache Error", "Could not read from cache " + cache);  
 return null;  
 }  
 }  
}