STRV

REALM IO

Alternative to SQLite on Android

STRV

SCHEDULE FOR THIS REALM TALK

- How I looked for persistence
- Why I decided to use Realm
- What issues I had solve with Realm

PERSIST DATA

- Shared preferences
- Internal storage
- External storage
- Network connections
- Database

PERSIST MODEL

- Shared preferences
- Internal storage
- External storage
- Network connections
- Local database

PERSIST MODEL OFFLINE

- Shared preferences
- Internal storage
- External storage
- Network connections
- Local database

DATABASE FRAMEWORK

- SQLite & ORM
  - DBFlow
  - ORMLite
  - GreenDao
  - ...

DATABASE FRAMEWORK

- SQLite & ORM
  - DBFlow
  - ORMLite
  - GreenDao
  - ...
- Realm (Tightdb engine)

DATABASE FRAMEWORK

- SQLite & ORM
  - DBFlow (reference)
  - ORMLite
  - GreenDao
  - ...
- **Realm**

WHY TO CHOOSE THE REALM

Performance?

WHY TO CHOOSE THE REALM

Performance?

- NO (performance is good, but DBFlow has also good performance)

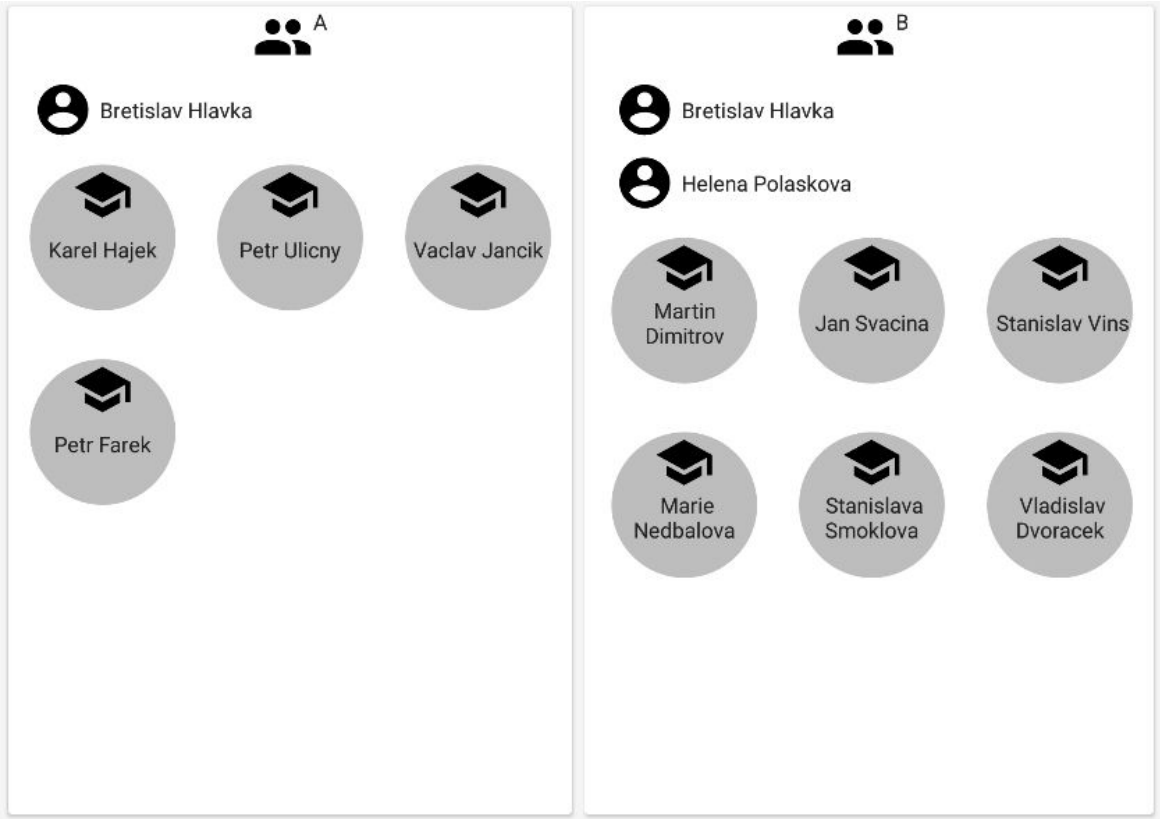
WHY TO CHOOSE THE REALM

Performance?

- NO (performance is good, but DBFlow has also good performance)

- Simplicity (keep implementation simple)
- Live realm (choice in between using live realm or close connection like in sqlite)
- Easy to integrate (GSON, Parceler, Retrofit, Roboelectric, RxJava, Kotlin, Stetho...)
- Build in features (use advanced build-in features)

RELATIONS MODEL



SIMPLE? **RELATIONS** WITH DBFLOW

```java
@Table(database = DbFlowDatabase.class)
@ManyToMany(referencedTable = TeacherDbFlowEntity.class)
public class SchoolClassDbFlowEntity extends BaseDbFlowModel
        implements
        SchoolClassInterface<StudentDbFlowEntity, TeacherDbFlo
{
    @Column
    @PrimaryKey(autoincrement = true)
    long dbId;
    @Column
    @SerializedName("id")
    long serverId;
    @Column
    @SerializedName("name")
    String name;
    @Column
    @SerializedName("grade")
    int grade;
    @SerializedName("studentIdList")
    List<Long> studentIdList;
    @SerializedName("teacherIdList")
    List<Long> teacherIdList;
    List<StudentDbFlowEntity> studentList;
    List<TeacherDbFlowEntity> teacherList;
```

```java
@Table(database = DbFlowDatabase.class)
public class TeacherDbFlowEntity extends BaseDbFlowModel
        implements TeacherInterface<SchoolClassDbFlowEntity>
{

    @PrimaryKey(autoincrement = true)
    @Column
    long dbId;
    @Column
    @SerializedName("id")
    long serverId;
    @Column
    @SerializedName("name")
    String name;
    List<SchoolClassDbFlowEntity> schoolClassList;


    @Bindable
    public long getId() { return dbId; }


    public void setId(long id)
    {
```

SIMPLE? **RELATIONS** WITH DBFLOW

ShowcaseDbFlowModule.java

```
//save all teachers
List<TeacherDbFlowEntity> teacherList =
        new Select().from(TeacherDbFlowEntity.class)
                .queryList();
for(TeacherDbFlowEntity t : teacherEntities)
{
    boolean updated = false;
    for(TeacherDbFlowEntity t1 : teacherList)
    {
        if(t1.getServerId() == t.getServerId())
        {
            t1.update(t);
            t1.save();
            updated = true;
        }
    }
    if(!updated)
    {
        t.save();
    }
}
```

ShowcaseDbFlowModule.java

```
//save M-N relationShip between teacher and class
List<TeacherDbFlowEntity> teacherList1 =
        new Select().from(TeacherDbFlowEntity.class).queryList();
for(SchoolClassDbFlowEntity sc : schoolClassEntities1)
{
    for(SchoolClassDbFlowEntity sc1 : schoolClassEntities)
    {
        if(sc1.getServerId() == sc.getServerId())
        {
            sc.setTeacherIdList(sc1.getTeacherIdList());
        }
    }
    for(TeacherDbFlowEntity t : teacherList1)
    {
        if(sc.getTeacherIdList().contains(t.getServerId()))
        {
            SchoolClassDbFlowEntity_TeacherDbFlowEntity entity =
                    new SchoolClassDbFlowEntity_TeacherDbFlowEntity();
            entity.setTeacherDbFlowEntity(t);
            entity.setSchoolClassDbFlowEntity(sc);
            entity.save();
        }
    }

}
```

SIMPLE **RELATIONS** WITH REALM

```java
@RealmClass
public class SchoolClassRealmEntity
        implements SchoolClassInterface<StudentRealmEntity, TeacherRealmE
        RealmModel
{

    @SerializedName("studentIdList")
    RealmList<RealmLong> studentIdRealmList = new RealmList<>();
    @SerializedName("teacherIdList")
    RealmList<RealmLong> teacherIdRealmList = new RealmList<>();
    RealmList<StudentRealmEntity> studentRealmList = new RealmList<>();
    RealmList<TeacherRealmEntity> teacherRealmList = new RealmList<>();
    @io.realm.annotations.PrimaryKey
    @SerializedName("id")
    private long id;
    @SerializedName("name")
    private String name;
    @SerializedName("grade")
    private int grade;
```

```java
@RealmClass
public class TeacherRealmEntity
        implements TeacherInterface<SchoolClassRealmEntity>,
        RealmModel {


    @io.realm.annotations.PrimaryKey
    @SerializedName("id")
    protected long serverId;

    @SerializedName("name")
    protected String name;

    @SerializedName(value = "birthDate")
    protected Date birthDate;

    protected RealmList<SchoolClassRealmEntity> schoolClassLis
            = new RealmList<>();
```

SIMPLE **RELATIONS** WITH REALM

ShowcaseRealmModule.java

```
/**
 * Persist complete model to realm.
 * Realm will run that transaction on a background thread and report back when the transaction is done.
 *
 * @param persistenceModel
 * @param dataHandlerListener
 */
public void saveOrUpdateRealmSchoolClass(List<SchoolClassRealmEntity> persistenceModel,
                                         DataHandlerListener dataHandlerListener) {
    if(persistenceModel == null) return;
    final Realm realm = Realm.getDefaultInstance();
    try {
        if(!persistenceModel.isEmpty()) {
            // Asynchronously update objects on a background thread
            realm.executeTransactionAsync(
                    bgRealm -> bgRealm.copyToRealmOrUpdate(persistenceModel),
                    () -> {
                        if(realm != null) realm.close();
                        dataHandlerListener.handleSuccess();
                    },
                    error -> {
                        if(realm != null) realm.close();
                        dataHandlerListener.handleFailed(error);
                    });
```

QUERY WITH REALM

```
/**
 * Most queries in Realm are fast enough to be run synchronously - even on the UI thread.
 *
 * @return
 */
@Nullable
public List<SchoolClassRealmEntity> getSchoolClass() {
    final Realm realm = Realm.getDefaultInstance();
    try {
        return realm.copyFromRealm(realm.where(SchoolClassRealmEntity.class).findAll());
    } catch(Exception e) {
        Timber.e(e, "getSchoolClass from realm failed!");
        return null;
    } finally {
        if(realm != null) {
            realm.close();
        }
    }
}
```

QUERY WITH REALM

```
/**
 * For either very complex queries or queries on large data sets
 * it can be an advantage to run the query on a background thread.
 *
 * @return
 */
public void loadSchoolClassAsync(LoadSchoolClassAsyncListener loadSchoolClassAsyncListener) {
    final Realm realm = Realm.getDefaultInstance();
    RealmChangeListener callback = new RealmChangeListener<RealmResults<SchoolClassRealmEntity>>() {
        @Override
        public void onChange(RealmResults<SchoolClassRealmEntity> results) {
            // called once the query complete and on every update
            loadSchoolClassAsyncListener.onSuccess(realm.copyFromRealm(results));
            results.removeChangeListeners();
            if(realm != null) {
                realm.close();
            }
        }
    };

    try {
        realm.copyFromRealm(realm.where(SchoolClassRealmEntity.class).findAllAsync());
        realm.addChangeListener(callback);
    } catch(Exception e) {
        Timber.e(e, "getSchoolClass from realm failed!");
        if(realm != null) {
            realm.close();
```

SIMPLE QUERY WITH REALM

```
/* check for UNIQUE'S FIELD value existence */
if(realm.where(StudentRealmEntity.class).equalTo("uniqueField",
        student.getUniqueField()).findAll().size() > 0) {
    broadcastFail(dataHandlerListener,
            new RuntimeException("Add failed! Student already exists."));
    return;
}
/**
 *  UNIQUE SEQUENCE KEYS
 */
public static <E extends RealmModel>
long getNextFreePrimaryKey(Realm realm, Class<E> clazz, String keyName) {
    return realm.where(clazz).max(keyName).longValue() + 1;
}
```

SIMPLE **INIT** FROM ASSET/RAW

```
/**
 * When opening the Realm for the first time, instead of creating an empty file,
 * the Realm file will be copied from the provided asset file and used instead.
 * <p>
 * This cannot be configured to clear and recreate schema by calling deleteRealmIfMigrationNeeded()
 * at the same time as doing so will delete the copied asset schema.
 * <p>
 * There is no restriction for the size of file in raw/assets directory since Android 2.3
 * But there is limit 50MB of the apk. When you exceed this limit, use expansion.apk
 *
 * @return
 */
@Provides
@Singleton
@ShowcaseRealmConfigurationAsset
public RealmConfiguration provideRealmAssetConfiguration() {
    if(mRealmAssetConfiguration == null) {
        mRealmAssetConfiguration = new RealmConfiguration.Builder()
                .name(REALM_NAME_ASSET)
                .schemaVersion(SCHEMA_VERSION)
                .migration(mRealmMigration == null ? new ShowcaseRealmMigration() : mRealmMigration)
                .assetFile("realm/init.realm")
                .build();
    }
    return mRealmAssetConfiguration;
}
```

SUPPORT FOR LIVE REALM

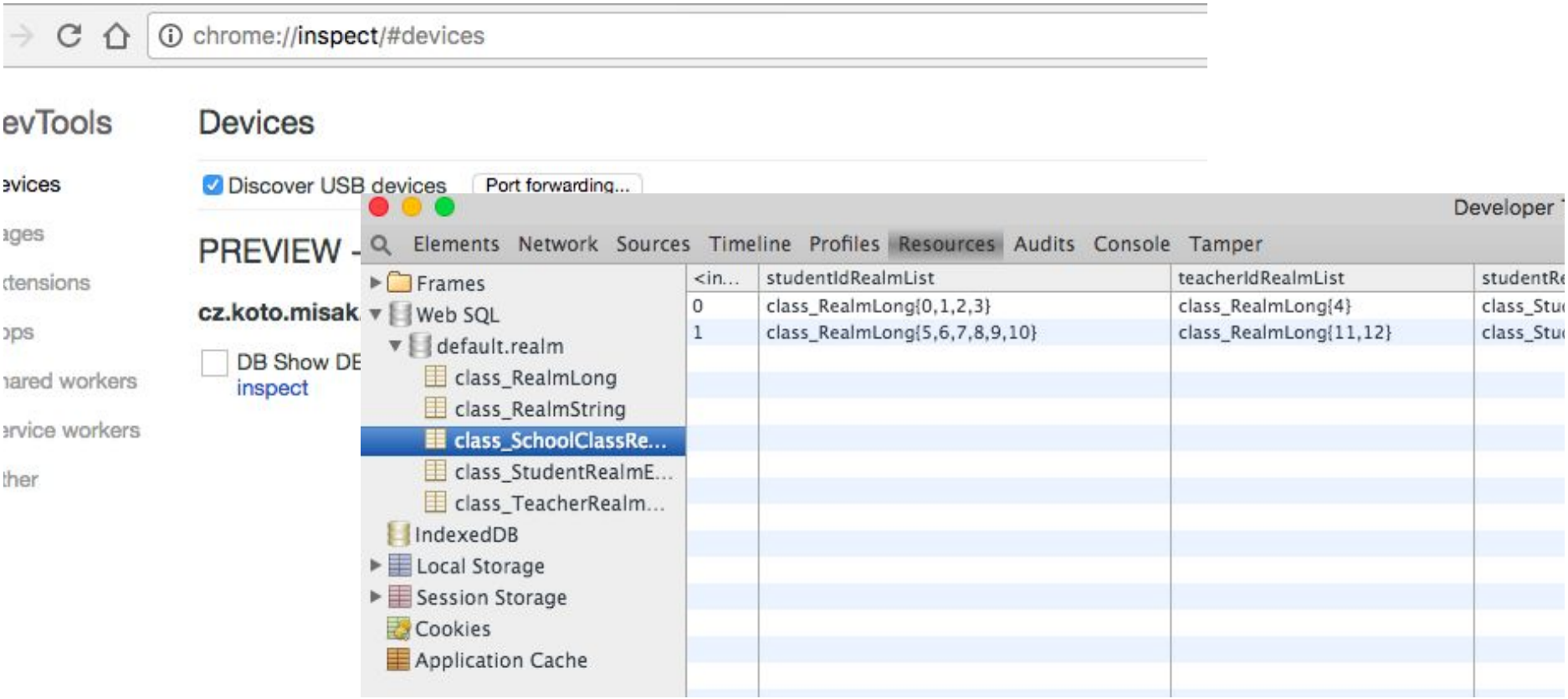
Live realm

- Operate on live objects: the auto updating view to database
- Don’t forget to close realm when your scope (app/activity) is finishing.
- Use it for design case, when realm data are the basics for your model.

Copying data

- Just copy the data from live realm and close this realm immediately.
- Use it for design case, when realm data are the backup of your model.

INTEGRATE STETHO WITH REALM



BUILD IN SECURITY

Encrypt/decrypt *.realm (using 512-bit encryption key)
with standard AES-256 encryption

```
byte[] key = new byte[64];
new SecureRandom().nextBytes(key);
RealmConfiguration config = new RealmConfiguration.Builder()
  .encryptionKey(key)
  .build();

Realm realm = Realm.getInstance(config);
```

BUILD IN REMOTE AUTO SYNC

`build.gradle`:

```
realm {
  syncEnabled = true;
}
```

```
SyncCredentials myCredentials = SyncCredentials.usernamePassword(username, password, true);
```

```
String authURL = "http://my.realm-auth-server.com:9080/auth";
SyncUser user = SyncUser.login(myCredentials, authURL);
```

```
Realm realm = Realm.getInstance(syncConfiguration);
```

PAIN OF THE REALM

- Limited support for data types
- Missing support for inheritance
- Missing support for auto-delete

LIMITED DATA TYPE SUPPORT

Basic types

- boolean, byte
- short, int, long, float, double
- String, Date, byte[]

Boxed types

- Boolean, Byte
- Short, Integer, Long, Float, Double

And what the others? NO WAY, but!

- Serializable & in edge case Parcelable objects

LIMITED DATA TYPE SUPPORT & ENUMS

```
public enum MyEnum {
  FOO, BAR;
}

public class Foo extends RealmObject {
  private String enumDescription;

  public void saveEnum(MyEnum val) {
    this.enumDescription = val.toString();
  }

  public MyEnum getEnum() {
    return (enumDescription != null) ? MyEnum.valueOf(enumDescription) : null;
  }
}
```

```
@StringDef({START, PAUSE, SKIP, EMOTION, SHARE, EXTEND})
@Retention(RetentionPolicy.SOURCE)
public @interface ActionType {

    String START = "start";
    String PAUSE = "pause";
    String SKIP = "skip";
    String EMOTION = "emotion";
    String SHARE = "share";
    String EXTEND = "extend";
}
```

```
public long createId() { return getActionTimeSt

public
@ActionType
String getActionType() { return mActionType; }

public void setActionType(@ActionType String a

public
@ActionPlace
String getActionPlace() { return mActionPlace;

public void setActionPlace(@ActionPlace String
```

LIMITED DATA TYPE SUPPORT & LISTS

```java
import ...

/**
 * Wrapper over String to support setting a list
 * of Strings in a RealmObject
 * To use it with GSON, please see RealmStringDeserializer
 *
 * http://www.myandroidsolutions.com/2015/05/29/gson-realm-string-array/
 */
@Parcel(implementations = { RealmStringRealmProxy.class },
        value = Parcel.Serialization.BEAN,
        analyze = { RealmString.class })
public class RealmString extends RealmObject {

    private String stringValue;

    public RealmString(){}

    public RealmString(String stringValue){
        this.stringValue =  stringValue;
    }


    public String getStringValue() { return stringValue; }

    public void setStringValue(String stringValue) {
        this.stringValue = stringValue;
    }

}
```

```java
*/
public class RealmStringDeserializer implements
        JsonDeserializer<RealmList<RealmString>> {

    @Override
    public RealmList<RealmString> deserialize(JsonElement json, Type typeOfT,
                                              JsonDeserializationContext context)
            throws JsonParseException {

        RealmList<RealmString> realmStrings = new RealmList<>();
        JsonArray stringList = json.getAsJsonArray();

        for (JsonElement stringElement : stringList) {
            realmStrings.add(new RealmString(stringElement.getAsString()));
        }

        return realmStrings;
    }
}
```

RestModule.java ×

```java
    @Provides
    @Singleton
    Gson provideGson(){
        GsonBuilder gsonBuilder = new GsonBuilder()
                .setDateFormat("yyyy-MM-dd'T'HH:mm:ss.sss'Z'")//ISO-8601
                .setExclusionStrategies(new DbFlowExclusionStrategy(),
                        new RealmExclusionStrategy())
                //Realm adapters
                .registerTypeAdapter(new TypeToken<RealmList<RealmString>>() {
                }.getType(), new RealmStringDeserializer())
                .registerTypeAdapter(new TypeToken<RealmList<RealmLong>> () {
```

LIMITED DATA TYPE SUPPORT & LISTS

```java
import io.realm.RealmObject;

public class RealmLong extends RealmObject {

    private Long longValue;

    public RealmLong(){}


    public RealmLong(Long longValue){
        this.longValue = longValue;
    }

    public Long getLongValue() {
        return longValue;
    }

    public void setLongValue(Long longValue) {
        this.longValue = longValue;
    }

}
```

```java
public class RealmLongDeserializer implements
        JsonDeserializer<RealmList<RealmLong>> {

    @Override
    public RealmList<RealmLong> deserialize(JsonElement json, Type typeOfT,
                                            JsonDeserializationContext context)
            throws JsonParseException {

        RealmList<RealmLong> realmLongs = new RealmList<>();
        JsonArray longList = json.getAsJsonArray();

        for (JsonElement longElement : longList) {
            realmLongs.add(new RealmLong(longElement.getAsLong()));
        }

        return realmLongs;
    }
}
```

RestModule.java ×

```java
    @Provides
    @Singleton
    Gson provideGson(){
        GsonBuilder gsonBuilder = new GsonBuilder()
                .setDateFormat("yyyy-MM-dd'T'HH:mm:ss.sss'Z'")//ISO-8601
                .setExclusionStrategies(new DbFlowExclusionStrategy(),
                        new RealmExclusionStrategy())
                //Realm adapters
                .registerTypeAdapter(new TypeToken<RealmList<RealmString>>() {
                }.getType(), new RealmStringDeserializer())
                .registerTypeAdapter(new TypeToken<RealmList<RealmLong>> () {
                }.getType(), new RealmLongDeserializer());
```

MISSING INHERITANCE SUPPORT & WORKAROUND

BroadcastPlayerModel.java

```
public class BroadcastPlayerModel extends PlayerModel {
    public static final int DELAY_IN_MILLIS_FOR_INTERSTITIAL_REMOVE = 50
    private final BonusProvider mBonusProvider;
    private final EndingControllerInterstitialBroadcastPlayerItem endingIt
    private final EndingLoaderInterstitialBroadcastPlayerItem endingItemLo
    private final PlaybackController mPlaybackController;
    private final BroadcastPlayTimeCounter mBroadcastPlayTimeCounter;
    List<BroadcastPlayerItem> mPlayerItems;
    int mCurrentPosition = 0;
    boolean mCurrentItemBlocked = false;
    private BonusInterstitialBroadcastPlayerItem bonusInterstitial;
    private BroadcastProgram mBaseBroadcastProgram;
    private BroadcastProgram mBonusBroadcastProgram;
    private BroadcastLogEnd mPreparedLogEnd;
    private boolean mEndingProcess;


    /**
     * @param currentState
     * @param itemPlayTimeCounter
     * @param currentItemStartLocation
     * @param bonusProvider
     * @param endingItemControl
     * @param endingItemLoader
     * @param playbackController
     * @param broadcastPlayTimeCounter
     * @param playerItems
     * @param currentPosition
```

OttoPersistence.java

```
    public void saveOrUpdateBroadcastPlayerModel(BroadcastPlayerModel bpm) {
        if(bpm == null || bpm.isEndingProcess()) return;
        Logcat.d("BROADCAST.PLAYER.MODEL-saveOrUpdate started.");
        BroadcastPlayerPersistence persistence = new BroadcastPlayerPersistence
        //PlayerModel persistence
        persistence.setCurrentState(bpm.getCurrentState());
        persistence.setCurrentItemBlocked(bpm.isCurrentItemBlocked());
        persistence.setItemPlayTimeCounter(bpm.getItemPlayTimeCounter());
        persistence.setBroadcastPlayTimeCounter(bpm.getBroadcastPlayTimeCounter
        persistence.setCurrentItemStartLocation(SerializeUtility.serializeB64Fr
        //BroadcastPlayerModel persistence
        persistence.setBonusProvider(bpm.getBonusProvider());
        persistence.setCurrentPosition(bpm.getCurrentPosition());
        if(bpm.getBaseBroadcastProgram() != null)
            bpm.getBaseBroadcastProgram().setSaveTypeKey(BroadcastProgram.BROAD
        persistence.setBaseBroadcastProgram(bpm.getBaseBroadcastProgram());
        if(bpm.getBonusBroadcastProgram() != null)
            bpm.getBonusBroadcastProgram().setSaveTypeKey(BroadcastProgram.BROA
        persistence.setBonusBroadcastProgram(bpm.getBonusBroadcastProgram());
        persistence.setPreparedLogEnd(bpm.getPreparedLogEnd());
        persistence.setPlaybackController(bpm.getPlaybackController());

        RealmList<PlayerItemPersistence> playerItemPersistenceRealmList = new R
        int i = 0;
        for(BroadcastPlayerItem item : bpm.getPlayerItems()) {
            if(item instanceof ArticleBroadcastPlayerItem) {
                PlayerItemArticlePersistence persistentPlayerItem = new PlayerI
                persistentPlayerItem.setArticle(((ArticleBroadcastPlayerItem) i
```

PlayerModel.java

```
public abstract class PlayerModel extends BaseObservable {

    PlaybackService.PlaybackState mCurrentState = PlaybackService.PlaybackState.STOPPED;
    boolean mBuffering = false;
    ItemPlayTimeCounter mItemPlayTimeCounter = new ItemPlayTimeCounter();
    Location mCurrentItemStartLocation; //TODO setup in ArticlePlayerModel as well
```

MISSING INHERITANCE SUPPORT & WORKAROUND

```
public void saveOrUpdateBroadcastPlayerModel(BroadcastPlayerModel bpm) {
    if(bpm == null || bpm.isEndingProcess()) return;
    BroadcastPlayerPersistence persistence = new BroadcastPlayerPersistence();
    //PlayerModel persistence
    persistence.setCurrentState(bpm.getCurrentState());
    persistence.setCurrentItemBlocked(bpm.isCurrentItemBlocked());
    persistence.setItemPlayTimeCounter(bpm.getItemPlayTimeCounter());
    persistence.setBroadcastPlayTimeCounter(bpm.getBroadcastPlayTimeCounter());
    persistence.setCurrentItemStartLocation(SerializeUtility.serializeB64FromParcelable(bpm.getCurrentItemStartLocation()));
    //BroadcastPlayerModel persistence
    persistence.setBonusProvider(bpm.getBonusProvider());
    persistence.setCurrentPosition(bpm.getCurrentPosition());
    if(bpm.getBaseBroadcastProgram() != null)
        bpm.getBaseBroadcastProgram().setSaveTypeKey(BroadcastProgram.BROADCAST_PROGRAM_PERSISTENCE_KEY_SCOPE_PLAYER);
    persistence.setBaseBroadcastProgram(bpm.getBaseBroadcastProgram());
    if(bpm.getBonusBroadcastProgram() != null)
        bpm.getBonusBroadcastProgram().setSaveTypeKey(BroadcastProgram.BROADCAST_PROGRAM_PERSISTENCE_KEY_SCOPE_PLAYER);
    persistence.setBonusBroadcastProgram(bpm.getBonusBroadcastProgram());
    RealmList<PlayerItemPersistence> playerItemPersistenceRealmList = new RealmList<>();
    int i = 0;
    for(BroadcastPlayerItem item : bpm.getPlayerItems()) {
        if(item instanceof ArticleBroadcastPlayerItem) {
            PlayerItemArticlePersistence persistentPlayerItem = new PlayerItemArticlePersistence();
            persistentPlayerItem.setArticle(((ArticleBroadcastPlayerItem) item).getArticle());
            persistentPlayerItem.setPosition(((ArticleBroadcastPlayerItem) item).getPosition());
            persistentPlayerItem.setPrefferedDurationInSeconds(((ArticleBroadcastPlayerItem) item).getPreferredDurationInSeconds());
            persistentPlayerItem.setMoreLikeThisRequested(((ArticleBroadcastPlayerItem) item).isMoreLikeThisRequested());
            persistentPlayerItem.setFirstMeaningfulSent(((ArticleBroadcastPlayerItem) item).isFirstMeaningfulSent());
            persistentPlayerItem.setActionPlace(((ArticleBroadcastPlayerItem) item).getActionPlace());
            playerItemPersistenceRealmList.add(new PlayerItemPersistence(persistentPlayerItem, i, ArticleBroadcastPlayerItem.class.getName()));
        } else {
            playerItemPersistenceRealmList.add(new PlayerItemPersistence(i, item.getClass().getName()));
        }
        i++;
    }
    persistence.setItemPersistenceRealmList(playerItemPersistenceRealmList);
    OttoRealmModule.get().saveOrUpdate(persistence);
}
```

LIMITED SUPPORT FOR AUTODELETE

- Child objects are not deleted from realm with their parent’s removal!
- Check it on your case and eventually handle delete sequence yourself.

NOTE ABOUT REALM CONCURRENCY RULES

Threads

- Realm files can be accessed by multiple threads concurrently
- You can’t hand over Realm objects, queries, and results between threads.

Processes

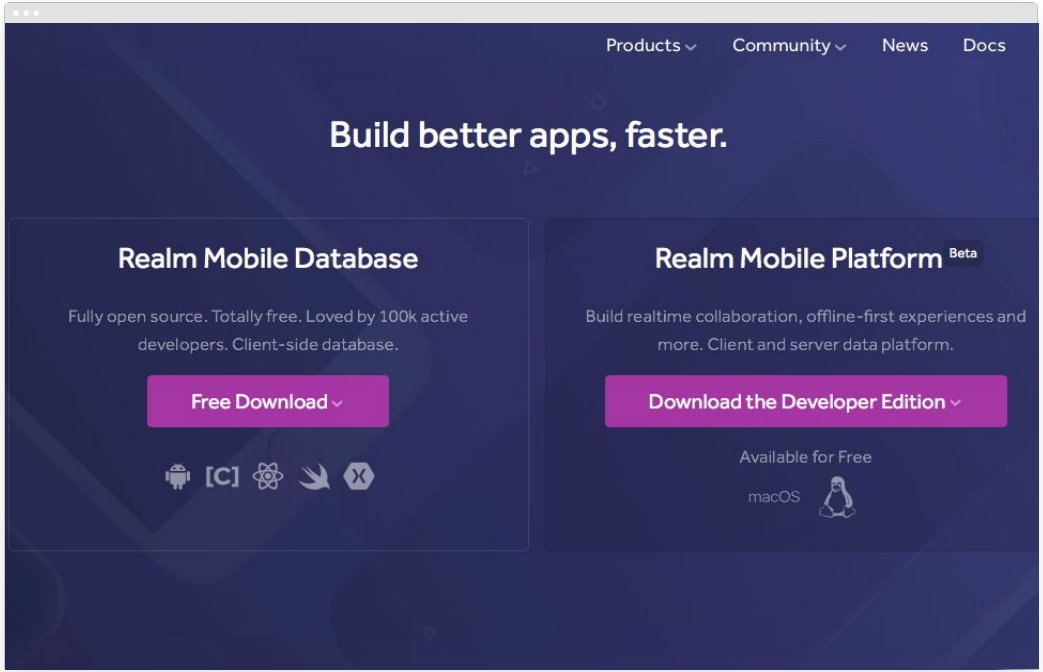
- Realms can only be accessed by a single process at a time.
- Different processes should either copy *.realm files or create their own.
- Multi-process support is promised coming soon.

Aren’t you afraid of mentioned pains of the Realm ?

Want to simply persist the model ?

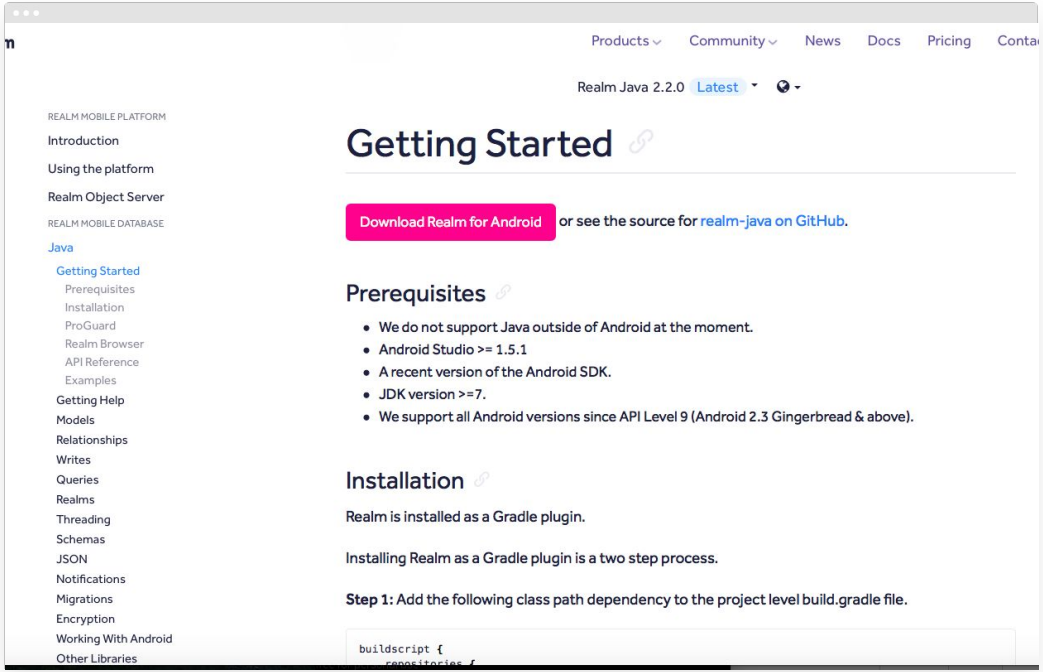
IF SO, USE THE REALM

REALM IO



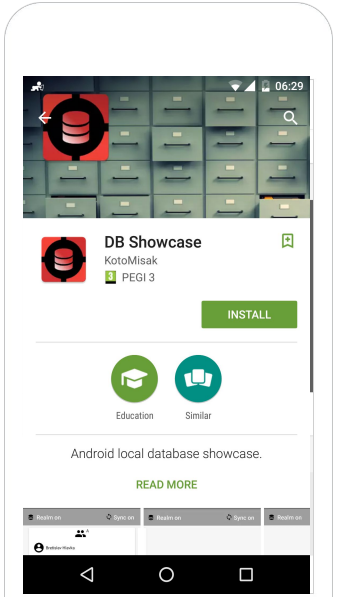
Ready for Android, React Native, Xamarin, Objective C & Swift.
Designed for mobile.

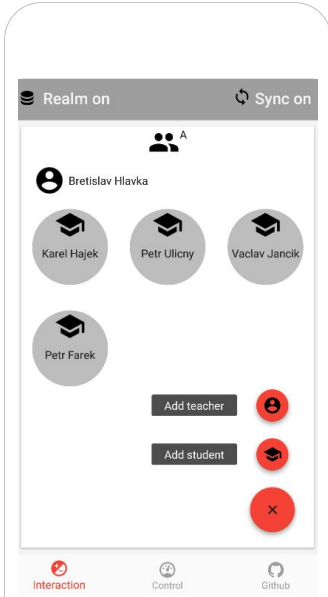
MORE ON **REALM IO** DOC

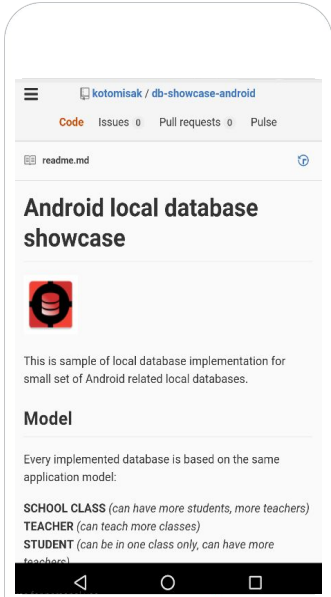


https://realm.io/docs/java/latest/

DB SHOWCASE







https://github.com/kotomisak/db-showcase-android

THANK YOU

michal.jenicek@strv.com

STRV

QUESTIONS

STRV