Platform-Based Development: Third Party Libraries

BS UNI studies, Spring 2018/2019

Dr Veljko Pejović
Veljko.Pejovic@fri.uni-lj.si

University of Ljubljana
Faculty of Computer and Information Science

Course Admin

- Sprint #3 instructions are out:
  - http://bitbucket.org/veljkop/runsup/
  - All in one .txt (or .md) file
  - Deadline May 12th 23:59
- Sprint #3 main improvements:
  - More sophisticated UI
    - NavigationDrawer and Fragments
  - Persistence in the local database
  - REST API for enabling users to see their data on different devices

Course Admin

- No labs this week!

Third Party Libraries

- Android allows easy integration via implementation command
- Libraries for:
  - Improved UI: Butterknife, MPAndroidChart
  - ORM data access: OrmLite, GreenDAO, Room
  - Easier networking: OkHttp
  - Interacting with backend: Parse (Back4App), Retrofit
  - Innovative data structures: Guava
  - …
- Browse https://android-arsenal.com/

Butterknife

- View-binding library for Android
  - Avoid all those findViewById calls
- Annotations for binding
  - Views: `@BindView(R.id.username) EditText edtUsername;`
  - Resources:
    ```
    @BindDrawable(R.drawable.graphic)
    Drawable graphic;
    ```
  - Listeners:
    ```
    @OnClick(R.id.submit)
    public void submit(View view) {
        // TODO submit data to server...
    }
    ```
- Behind the scenes it uses the annotations to create a new class

Butter Knife Example

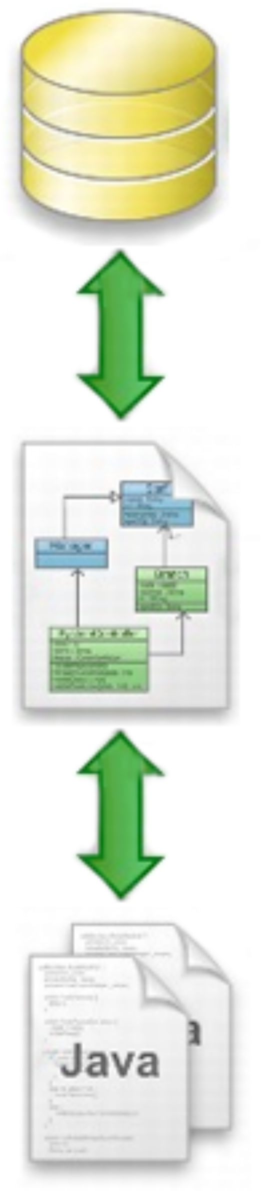
University *of Ljubljana*
Faculty *of Computer and Information Science*

Object-relational Mapping ORM

- Problem:
  - Object-oriented languages work with objects that can be relatively complex
  - (Relational) databases store and manipulate simple scalar values in tables
  - Converting objects to table entries is cumbersome and prone to errors
- Solution
  - Object-Relational Mapping (ORM)

OrmLite

- Data storage
  - A number of relational DBs (MS-SQL, MySQL, Android SQLite)
- Object files
  - Annotated Java models
- Data Access Object (DAO)
  - Interface between the database and Java objects
- Note: this is not another database, but a layer over your SQLite DB!



OrmLite Models

- Use annotations to mark classes that will be persisted

```
@DatabaseTable(tableName = "accounts")
public class Account {

    @DatabaseField(id = true)
    private String name;

    @DatabaseField(canBeNull = false)
    private String password;
    ...
    Account() { // all persisted classes must
      // define a no-arg constructor with at least
      // package visibility
    }
    ...
}
```

OrmLite Models - Annotations

- OrmLite annotations
  - @DatabaseTable: for each Java class you wish to persist
    - **tableName** argument specifies the name of the table that corresponds to the class
  - @DatabaseField: for each field in the class that you wish to persist
    - **columnName** (default: field name normalized)
    - **defaultValue** (default: null)
    - **canBeNull** (default: true)
    - **persisted** (default: true)
    - **unique** (default: false)
    - …

OrmLite Models - Annotations

ID is always unique!

- Managing relations
  - @DatabaseField:
    - **id** (default: false) indicates whether the field is an ID
    - **generatedId** (default: false) tells the database to auto-generate a corresponding id for every row inserted
    - **foreign** (default: false) identifies this field as corresponding to another class that is also stored in the database. The field must not be a primitive type
    - **foreignAutoRefresh** (default: false) automatically refresh the foreign field
    - **foreignAutoCreate** (default: false) automatically create a foreign field (in its table)
    - …

University *of Ljubljana*
Faculty *of Computer and Information Science*

OrmLite Models - Annotations

- Managing relations
  - @ForeignCollectionField:
    - Enables one to many relationships
    - **eager** (default: false) a separate query is made immediately and the foreign key items are stored as a list within the collection; otherwise – lazy – accessed only when a method is called on the collection

```
public class Account {
    @ForeignCollectionField(eager = false)
    ForeignCollection<Order> orders;
    …
}
```

OrmLite Models – Data Types

- Persisted data types
  - Standard/Primitive:
    - String, int/Integer, long/Long, float/Float, double/Double, etc.
  - Date/Time:
    - java.util.Date, DateTime, java.sql.Date, java.sql.Timestamp
  - Serializable:
    - You must explicitly set the field type

```
// image is an object that implements Serializable
@DatabaseField(dataType = DataType.SERIALIZABLE)
Image image;
```

University *of Ljubljana*
Faculty *of Computer and*
Information Science

OrmLite – Android

- OrmLiteSqliteOpenHelper
  - Extend this class to create and upgrade the database when your application is installed and provide the DAO classes used by your other classes
- OpenHelperManager
  - To manage Helper usage

```
private DatabaseHelper databaseHelper = 
null;

@Override
protected void onDestroy() {
    super.onDestroy();
    if (databaseHelper != null) {
        OpenHelperManager.releaseHelper();
        databaseHelper = null;
    }
}

private DBHelper getHelper() {
    if (databaseHelper == null) {
        databaseHelper =

OpenHelperManager.getHelper(this, 
DatabaseHelper.class);
    }
    return databaseHelper;
}
```

OrmLite – Android

- OrmLiteConfigUtil
  - Creates a configuration for your database in res/raw/ormlite_config.txt
  - Run it on your local machine, the file is shipped with your application (resource file)

```
public class DatabaseConfigUtil extends OrmLiteConfigUtil {
  private static final Class<?>[] classes = new Class[] {
    SimpleData.class,
  };
  public static void main(String[] args) throws Exception {
    writeConfigFile("ormlite_config.txt", classes);
  }
}
```

OrmLite – Data Access

- Via DAO object

```
Dao<Workout, Long> workoutDao = null;
DatabaseHelper databaseHelper = OpenHelperManager.getHelper(context,
DatabaseHelper.class);
try {
    workoutDao = databaseHelper.workoutDao();
} catch (SQLException e) {
    e.printStackTrace();
}
```

  – Query

```
try {
    QueryBuilder<Workout, Long> workoutBuilder = workoutDao.queryBuilder();
    Where where = workoutBuilder.where();
    where.eq(Workout.colStatus, 1);
    return workoutBuilder.query();
} catch (SQLException e) {
    e.printStackTrace();
}
```

OrmLite – Data Access

- Via DAO object

```
Dao<Workout, Long> workoutDao = null;
DatabaseHelper databaseHelper = OpenHelperManager.getHelper(context,
DatabaseHelper.class);
try {
    workoutDao = databaseHelper.workoutDao();
} catch (SQLException e) {
    e.printStackTrace();
}
```

  – Insert

```
workout = new Workout("Workout", sportActivity);
workout.setUser(user);
try {
    workoutDao.create(workout);
    workout.setTitle("Workout " + Long.toString(getId()));
    // update name
    workoutDao.update(workout);
} catch (SQLException e) {
    e.printStackTrace();
}
```

OrmLite in Android Example

University *of Ljubljana*
Faculty *of Computer and Information Science*

Other ORM Options

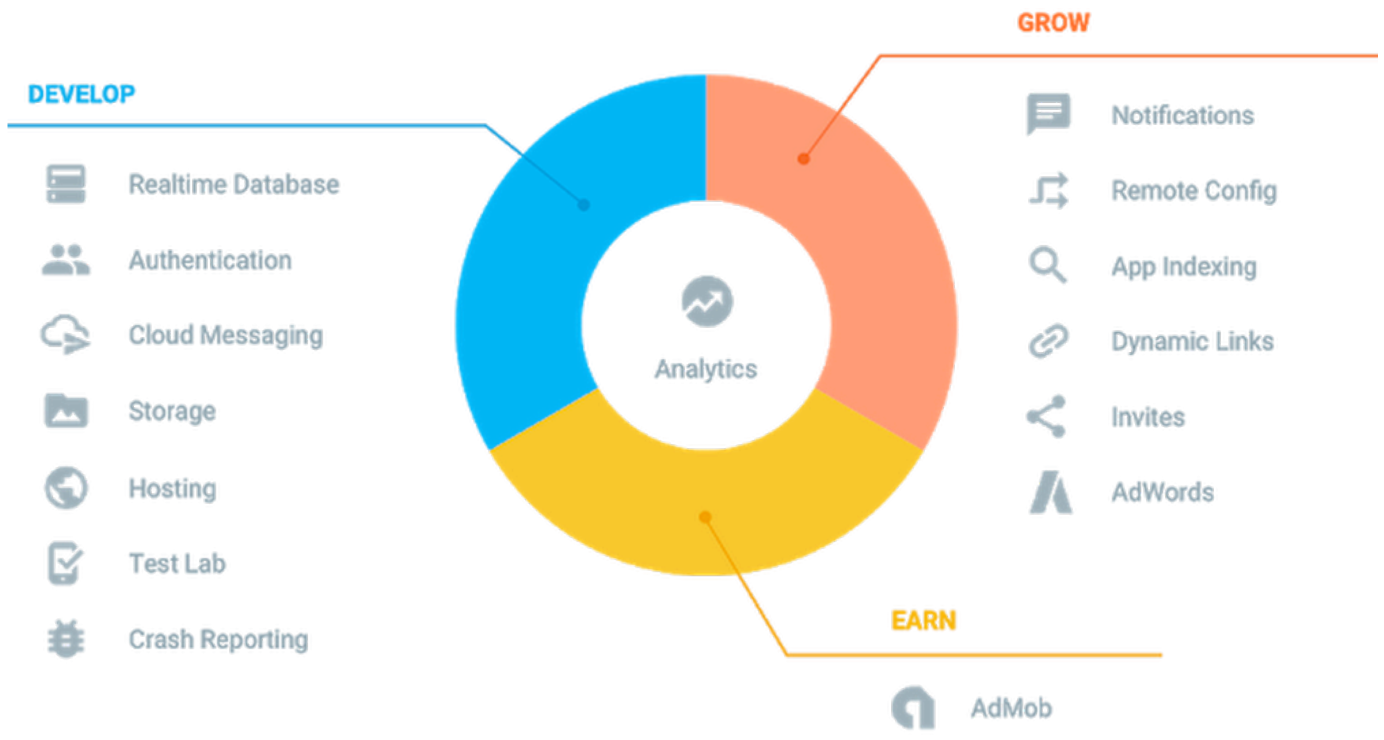
- Room (part of AndroidJetpack)
  - Pros:
    - Optimised to work with recent Android components
    - LiveData – notified when data is changed
    - Does not allow main thread execution
    - SQL queries checked at runtime
  - Cons:
    - Fewer Java methods for querying (compared to OrmLite)
    - Different data types than OrmLite
    - Supports only Android SQLite
- GreenDao
- SugarORM

Backend for Mobile Apps

- Android (or iOS for that matter) do not lock you into a particular backend technology
  - PHP, Node.js, Java Web apps, etc.
  - AWS, Google Cloud Platform, etc.
- Some solution easier to work with than others
  - Firebase
  - Parse Server (Back4App)

Firebase

- Mobile and Web app development platform supported by Google



Firebase

- Mobile and Web app development platform supported by Google
- Great for:
  - Authentication with Google ID (you have to use it)
  - Notifications (think chat-like app)
  - Crashlytics
  - Machine learning support

```
implementation 'com.google.firebase:firebase-core:16.0.8'
```

Parse Server

- Open source backend as a service (BaaS) platform initially developed by Facebook
  - Back4App is a Parse Server hosting platform
- Great for:
  - Building different REST APIs
  - Cron Jobs – schedule server jobs
  - User management (auto emails, social login)
  - Multiple SDKs
    - Including for Android

Back4App

- NoSQL database
- REST API to access data
- Access via HTTP using a number of languages/ platforms
- Different pricing tiers, but the free one is sufficient for prototyping
- Android library

```
implementation "com.github.parse-community.Parse-SDK-Android:parse:1.18.5"
```

Back4App – Create Backend

- Go to back4app.com, log in, and create a new application
  - Manage via a dashboard
  - Add collections (tables)
  - Add custom code
  - Initiate communication (notifications)
- Get the following (and put in your Android app) in order to access the backend:
  - Application ID
  - Client Key

Back4App Example

- At https://bitbucket.org/veljkop/parseexample/

Google Sign In Example

- At https://bitbucket.org/veljkop/googlesigninexample/