Cross-Platform Mobile Application for Students, Parents, and Faculty

Jake R. Johnson

1. Brief Purpose

The Foothill Mobile App digitizes archaic systems of student-to-teacher interactions and multi-app requirements for Grade-Access, staff directories, Foothill media outreach and sports scheduling. The app congregates these features into a single app, avoiding the need for ledgers and reducing tedious efforts by the ASB Student Government to manage outlets of campus life. Ease of access is vital in the creation of the application, promoting inclusivity of all devices through independent development on both operating systems (IOS and Android) to fit the requirements and specifications of both markets.

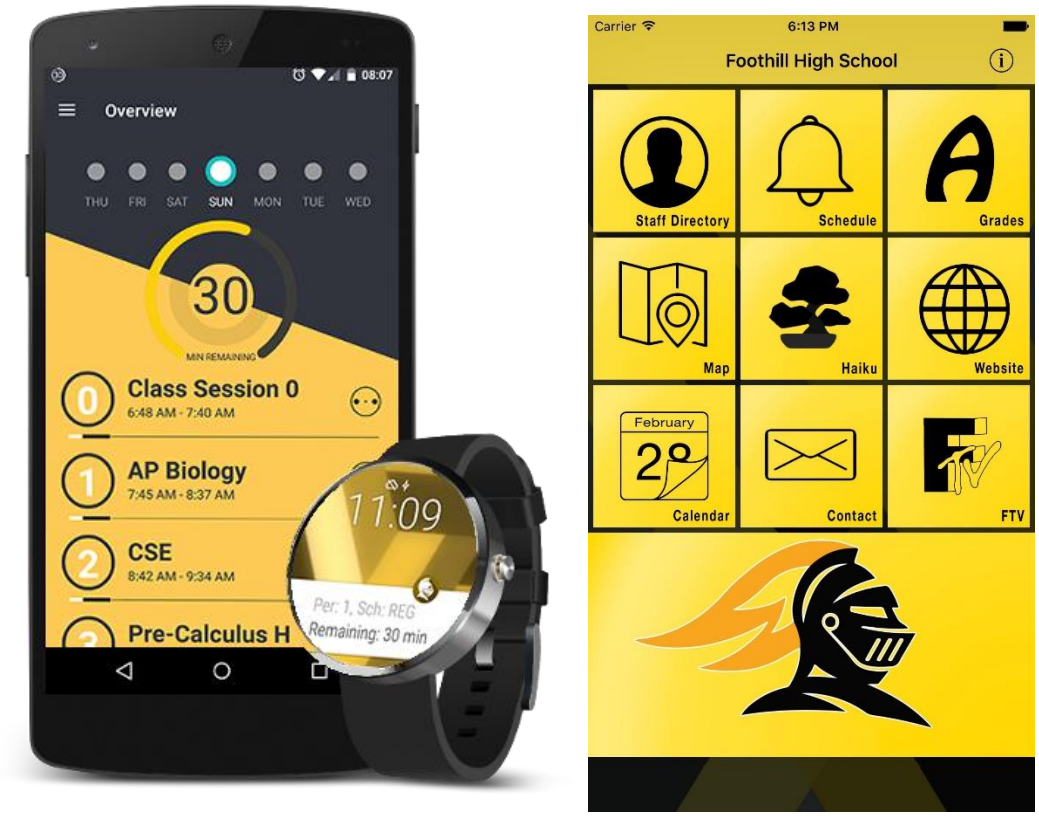


Figure 1.1 Initial Prototypes and UI layouts for home screen views and brief features.

2. Sports/Activities Scheduling Integration

Team management and scheduling for sports events is managed using parsed data from a Home Campus API system, integrated into a local calendar on the apps. XML data from continually updated queries is paginated and merged with school/ASB events on the primary calendar view. Base64 Encrypted ID parameters are required for MySQL queries using GET operations.

```
<?xml version="1.0" encoding="UTF-8"?><team>
<school>FHS</school>
<yearof>2018-19</yearof>
<sport>Football</sport>
<level>Varsity</level>
<eventType>Game</eventType>
<overallRecords>1-0-0</overallRecords>
<leagueRecords>1-0-0</leagueRecords>
<schedules>
        <schedule>
                <date>07-24-2018</date>
                <timeFrom>04:00PM</timeFrom>
                <timeTo></timeTo>
                <eventTitle></eventTitle>
                <gameTitle></gameTitle>
                <gameType>Non-League</gameType>
                <resultWinLossTie></resultWinLossTie>
                <schoolScore></schoolScore>
                <opponentsScore></opponentsScore>
                <notesFromScorePosting></notesFromScorePosting>
                <facilityName></facilityName>
                <facilityAddress></facilityAddress>
                <facilityLinkToGoogleMaps></facilityLinkToGoogleMaps>
        </schedule>
</schedules>
</team>
```

Figure 2.1 Query Response from parameters: year, school_id, sport_id, level, etc.

A common trouble with displaying school, sports and non-school events is that their sources are mixed and require multiple endpoints for data retrieval. I have designed combining REST API endpoints used by the school district in their campus-event scheduling systems to standardize data retrieval in the app.

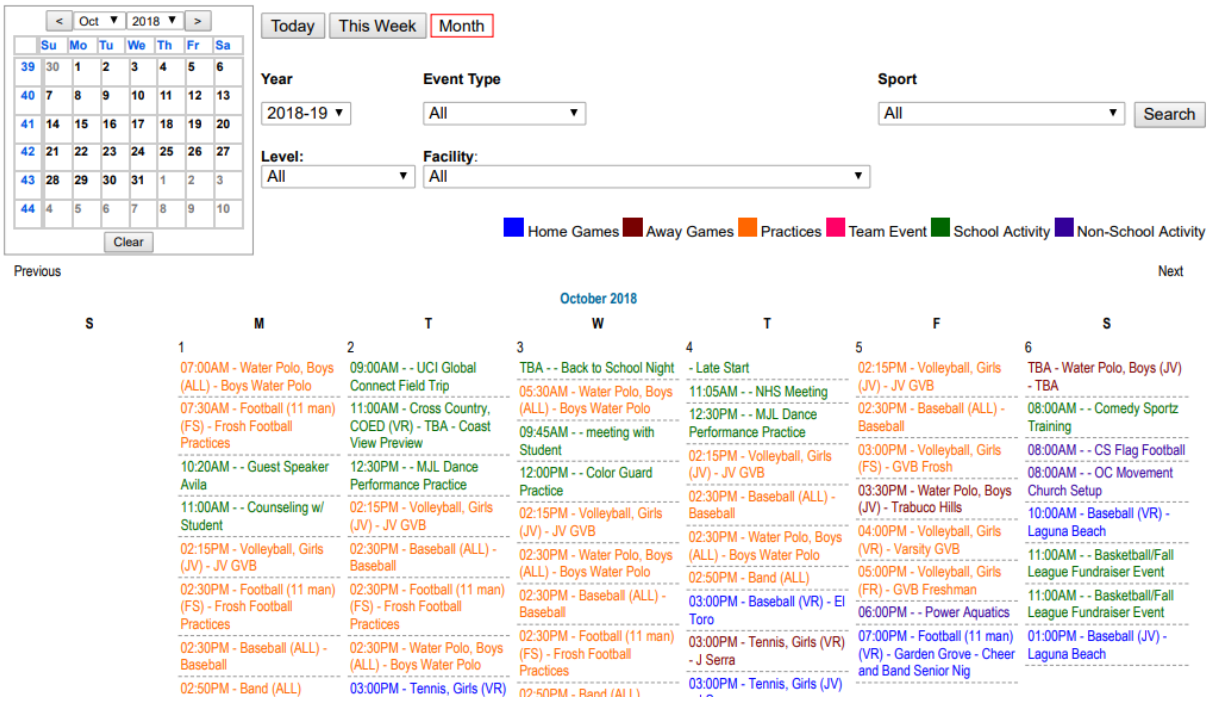


Figure 2.2 Self-populating pagination and cataloging of school/sporting events; source of API retrievals using proprietary web-portal

3. Scheduling and Personalization: Managing schedule updates and synchronicity across devices

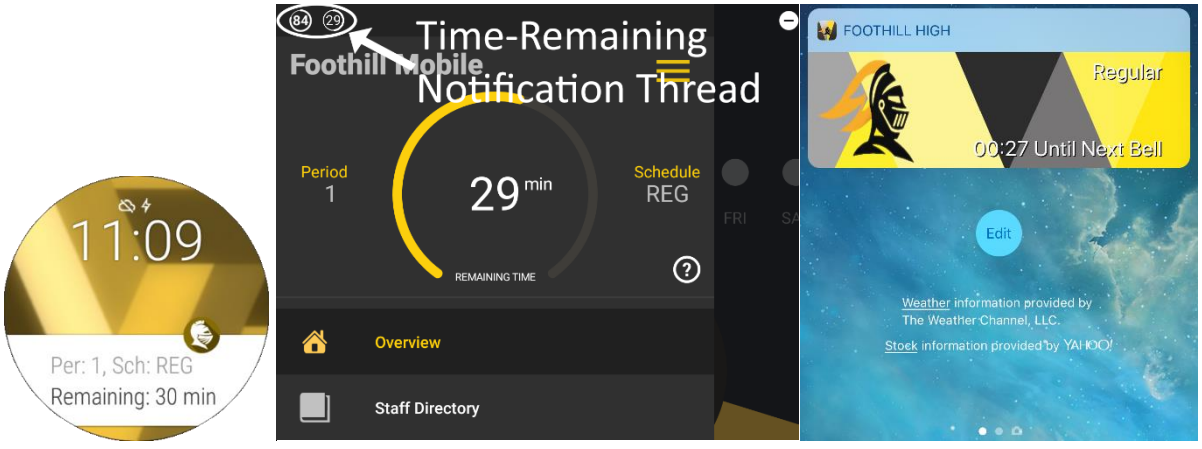


Figure 3.1 Notification Systems across three devices (Apple-Watch system not incl.)

4. UI/UX Design for Sports Integration and Device Interaction

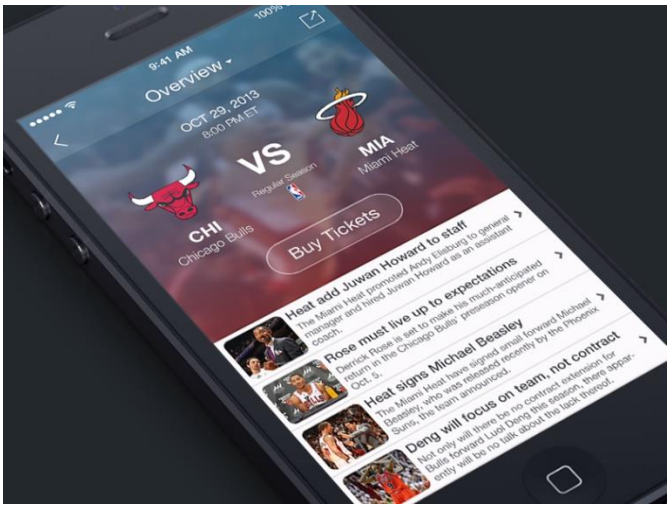


Figure 4.1 Model UI Design for team v. team sports layout using Android Fragment and IOS Listview

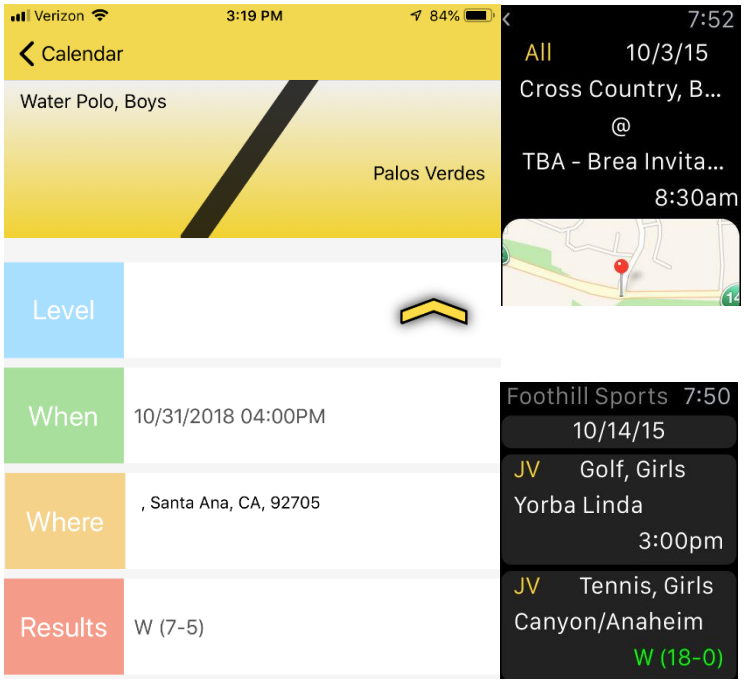


Figure 4.2 Sports UI layout in current usage, Foothill Sports managed through the web-portal are pushed to both devices, maintaining synchronicity between devices

5. Login Functionality and Aeries Portal Iterations

The underlying features of the login application is to remember passwords and prevent user-inputted passwords from becoming compromised on open-sessions. Sessions require an O2 Token authenticated header (with username and password) to enter the

grading/attendance/student view portal, and thus that information must be encrypted, particularly data remembered for repeated logins. Much of this was an investigation into Android's possibilities to encrypt data on-device while preventing root-permissible accounts from accessing that data and decrypting the usernames and passwords.

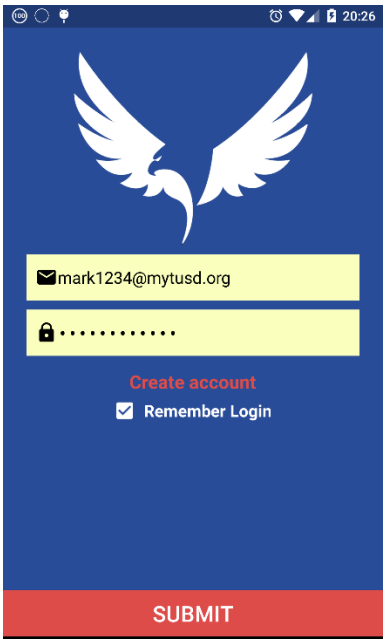
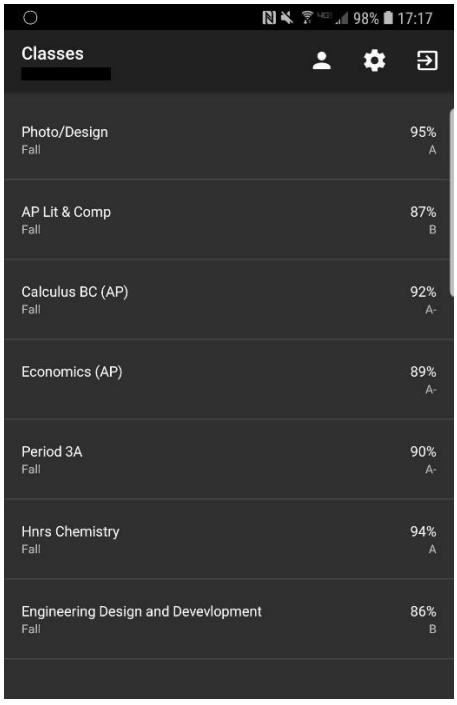


Figure 5.1 Final Design for Login view integrating newest system for remembering login details using advanced KeyStore cryptography (*Detailed Later*)

6. Improvement: Using Aeries REST API system to retrieve test scores, gradebook data, etc.; increased security using "certificate"



Using Aeries' Rest API System, a more streamlined development engine can be developed without simply displaying the results of a web page. Requests from the REST were used to create a more stream-lined grades-portal where the header-materials could be encrypted before being packaged and sent along with the request.

7. Media Syncing, Twitter, ASB and Foothill Television Integration

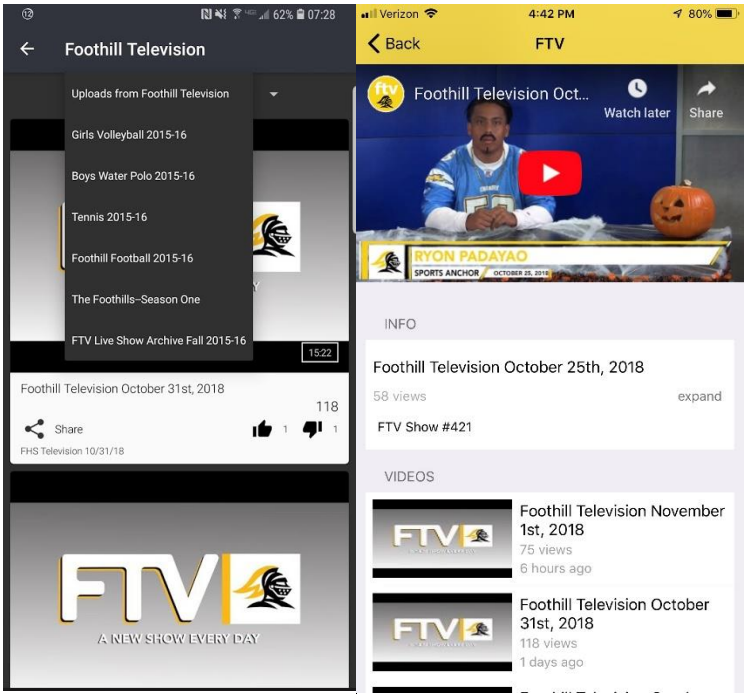


Figure 7.1 Foothill Media integration and custom YouTube-integrated "fragment" view to watch videos in-app

8. Staff Directory Listview and Cardview UI Design Prototyping

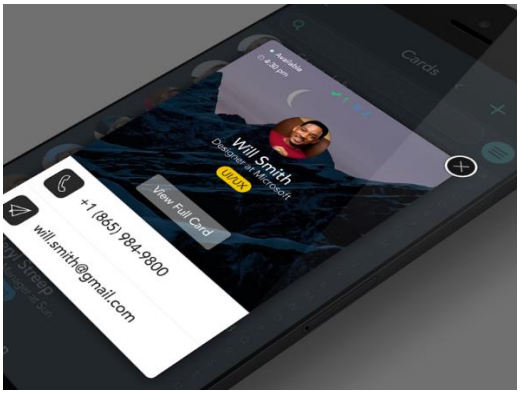


Figure 8.1 Original UI Design for Cardview with contact information

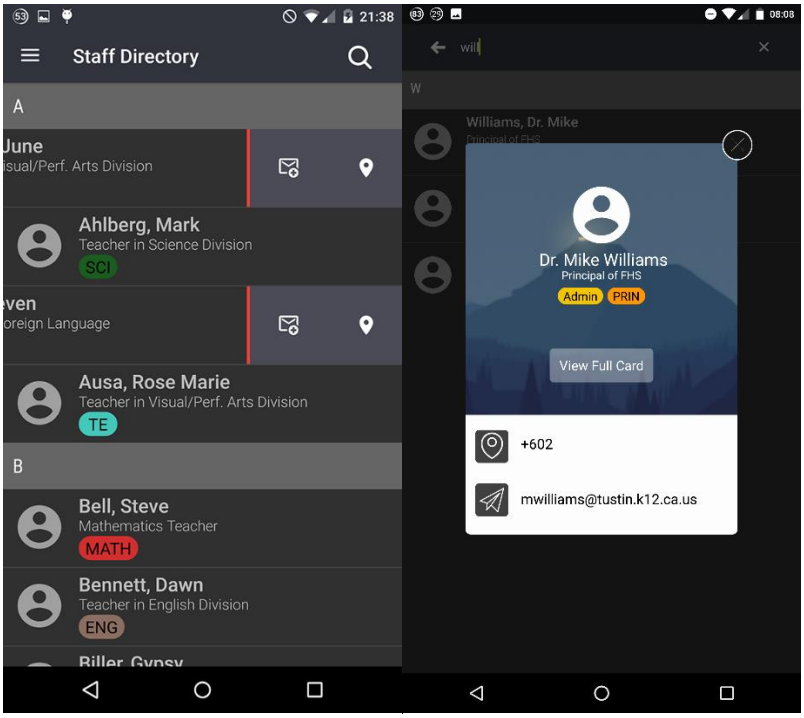


Figure 8.2 Final Design of Listview for Teachers updated sequentially and the final cardview with direct-to-mapview integration

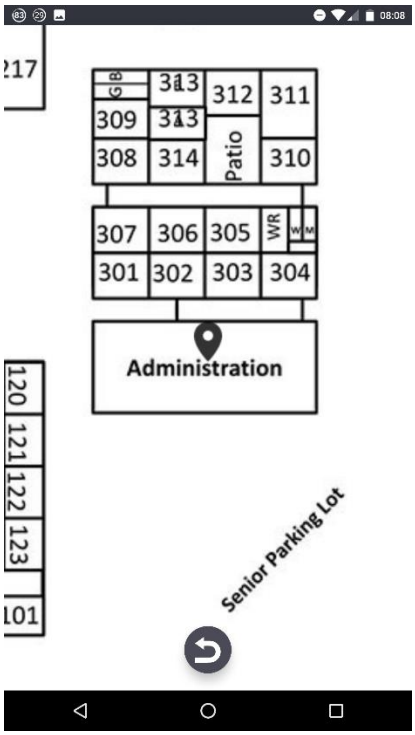


Figure 8.3 Clicking on either map icon in the staff directory or card view will redirect the app to a map-fragment with map location and campus details (particularly useful for students taking SAT).

9. Android Cryptographic Issues: Solving issue of attacker-prone key storage implementations, given over-use of local storage

The cryptographic functionalities on Android rely on over-simplified, attacker-prone interfaces. The standard *KeyStorageTest* implemented on the first iteration of the Foothill App generates a RSA key pair using the code in Figure **9.1** with a key-size of 2048bits. The *KeyPairGenerator* superclass also generates a self-signed certificate in which the details (serial number and validity period) have to be manually updated. The main fallacy of this operation is a reliance on the application to load and store the data. There is an over-simplified use of the InputStream to read the contents of an encrypted file on the device, prone to reverse-engineering. The only other model of security using this approach involves multiple key stores or additional encryption of the key store data.

```
KeyPairGenerator rsaKeyGen ;
try {
        rsaKeyGen = KeyPairGenerator . getInstance (" RSA " , " AndroidKeyStore ") ;
} /* ... */
}
KeyPairGeneratorSpec rsaKeyGenSpec = new KeyPairGeneratorSpec . Builder (
this )
        . setAlias (" TestKeyPair ")
        . setSubject ( new X500Principal (" CN = test ") )
        . setSerialNumber ( new BigInteger ("1") )
        . setStartDate ( new Date () )
        . setEndDate ( new GregorianCalendar (2015 , 0 , 0) . getTime () )
        . setKeySize (2048)
        . build () ;
try {
        rsaKeyGen . initialize ( rsaKeyGenSpec ) ;
} /* ... */
rsaKeyGen.generateKeyPair () ;
```

Figure 9.1 Generating a KeyPair using *AndroidKeyStore* and *KeyPairGenerator* classes on older API levels (18/19 = older versions Android)

10. Solution: AES-based encryption in CBC Mode to prevent multi-target attacks and increased file-based security for password and username storage

128-Bit security using AES-based encryption and HMAC keys is sufficient for Android-based key-storage and is an improvement on RSA fallacies. Using this approach an Initialization Vector (IV) and a SALT increase security at the primitive level. The IV is uniformly random and the other is hardware dependent, preventing on-device decryption of the key-store using root-permissions and other mainstream attacking methods. The random IV is generated using a strong PRNG on every cryptographic iteration, avoiding implicit assumptions made typically in RSA systems.

```
final String saltEncryption = loginPreferences.getString(SALT_STORE, null); //grab hardware-dependent SALT
if (saltEncryption == null) String salt = saltString(generateSalt()),
        loginPrefsEditor.putString(SALT_STORE, salt), loginPrefsEditor.commit();
key = generateKeyFromPassword(SALT_PASSWORD, loginPreferences.getString(SALT_STORE, null));
keyStr = keyString(key);
key = null; //Recycle to demonstrate converting it from str
/* ... */
AesCbcWithIntegrity.CipherTextIvMac mCipherTxtUsername = AesCbcWithIntegrity.encrypt(input_email.getText().toString();
AesCbcWithIntegrity.CipherTextIvMac mCipherTxtPassword = AesCbcWithIntegrity.encrypt(input_password.getText().toString();
str_input_username = mCipherTxtUsername.toString(); //Local variables shown for conversion purposes
str_input_password = mCipherTxtPassword.toString();

key = null;
/* ... */
```

Figure 10.1 Generating a KeyPair using *AndroidKeyStore* and *KeyPairGenerator* classes on older API levels (18/19 = older versions Android)

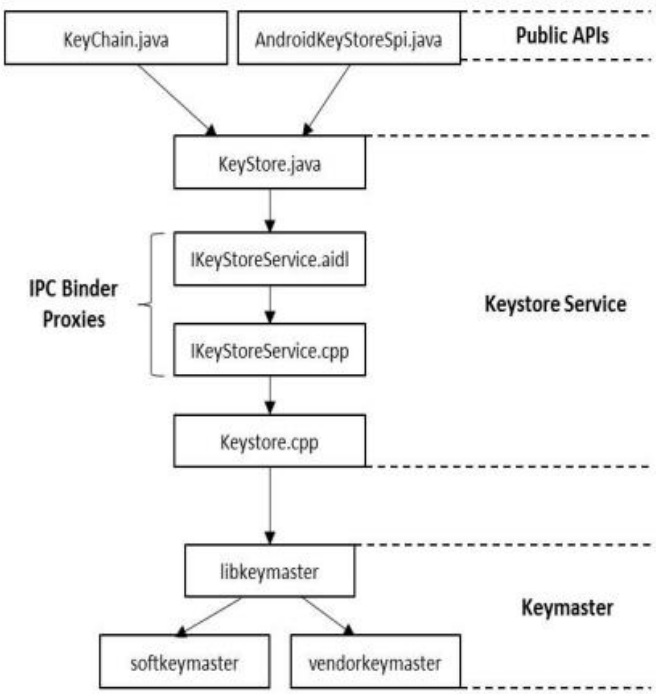


Figure 10.2 Architecture of Android KeyStore comprised of three layers: APIs, Keystore service, and Keymaster, all accessed through the Keymaster. Security can be threatened by an attack model focusing on “breaking into” the Keystore and weakening stored keys

```
private byte[] rsaEncrypt(byte[] secret) throws Exception{};
private  byte[]  rsaDecrypt(byte[] encrypted) throws Exception {};
private Key getSecretKey(Context context) throws Exception{};
public String encrypt(Context context, byte[] input) throws Exception{};
public byte[] decrypt(Context context, byte[] encrypted) throws Exception{};
```

The old-implemented (now obsolete) RSA system involved encrypting bytes of hash digest “inside” the code and using the *SharedPreference* libraries, transferring raw encrypted output to the KeyStore.

$$C_0 = E_k\left(E_k\left(IV\right) \oplus P_0\right) \oplus P_1$$

IV: IV used in generation of packet P_0
E_k: Evaluation of the block cipher (used for analysis)

The above encryption scheme details the newest system using CBC mode AES Encryption with a PRNG generated non-constant IV, resulting in a more secure non constant $E_k\left(IV\right)$. If the attacker can predict the value of the IV in advance, they can influence the value P_0 and eventually derive the value of $E_k(Q)$ (Where the KeyStore value and passwords/usernames can be determined). Therefore, to increase the safety of using IVs in CBC mode, we use a SALT as a form of double-encryption standard on the KeyStore.

11. Background Threads and Notification/Widget Updating

```
mUpdaterIntent = new Intent(getApplicationContext(),
                DrawerHeaderUpdater.class);
startService(mUpdaterIntent);

registerReceiver(mBroadcastReciever, new IntentFilter(
                DrawerHeaderUpdater.BROADCAST_ACTION));

/* ... */
mBroadcastReciever = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {};
```

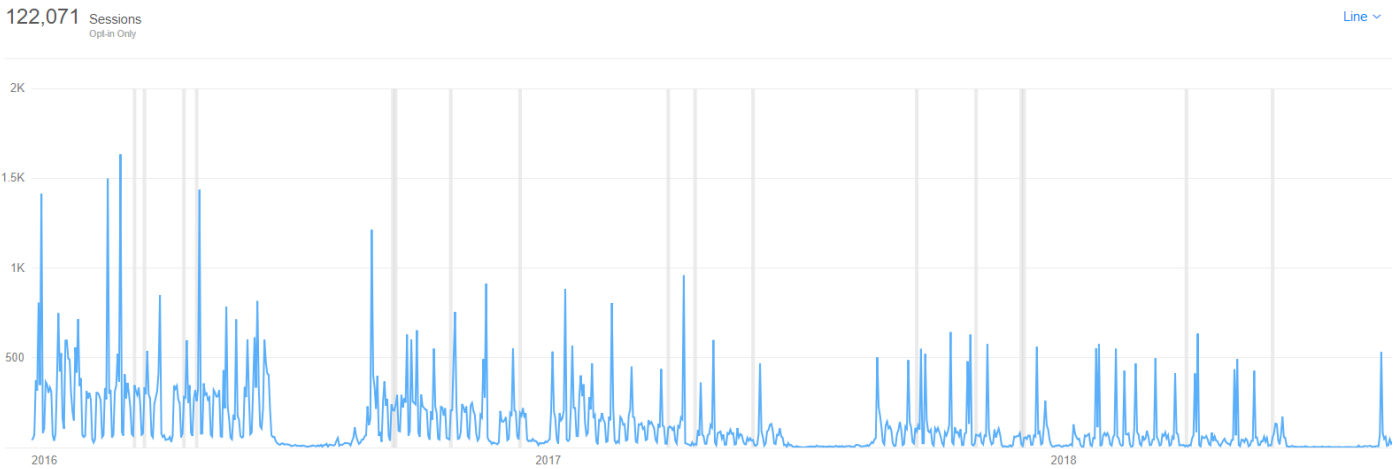
Figure 11.1 On Android, a background receiver used in conjunction with global service states provide a non-intensive notification system updated by internal app threads

```
class Updater extends Thread {
        @Override
        public void run() {
                super.run();
                isRunning = true;
                //Send broadcast for start of service progress bar
                while(isRunning){
                        Log.d("acd", "Running... DisplayLoggingInfo");
                        // sendbroadcast and create instances
                        compareDates();
                        DisplayLoggingInfo();
                        try {
                                Thread.sleep(DELAY);
                        } /* ... */
                }
        } // run end
        public boolean isRunning() {return this.isRunning;}

} // inner class end
```

Figure 11.2 Thread running while app is closed. Threads can be pushed to a frontal stack in the Android engine which allows dormant functions to be called (in this case if there is a change in schedule) and update the Notification System in real-time

12. Market Analysis: Statistical analysis of market trends and cross-platform interoperability



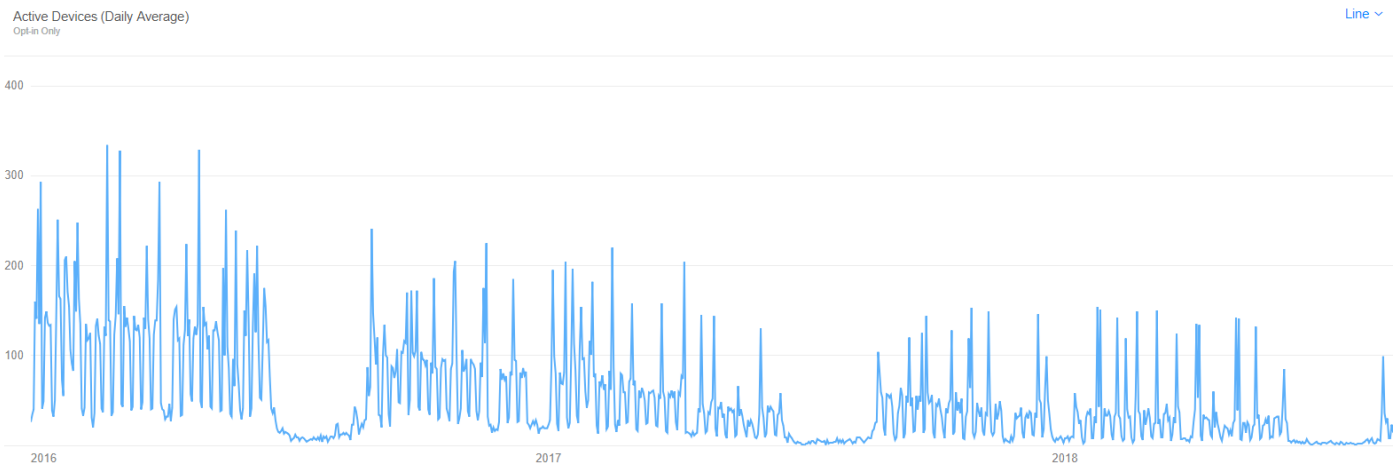


Figure 12.1 / 12.2 The above chart depicts all sessions between initial launch and current day, with flat patches attributable to update points. The lower photo shows active devices across IOS devices during those same periods from early 2016 to current day.

Overall, there is a trend of high retention across devices, leading to more stable session usage as stability of the app has risen and targeted demographics have become more affixed to their usage. These charts do not include sessions for the Android Watch and Apple Watch as they are tied directly into interactions with mobile devices. Crashes similarly reduced following a cycle of bug-fixes leading to more stability in the app and in the consumer base.

13. Summary and Final Notes

Android Features	IOS Features
Watch/Mobile Wear Capabilities: scheduling, sports, notification System	Watch/Mobile Wear: More advanced sports/map capabilities, notification system, ASB twitter/media access
Background Threads inc. Notification System	Home screen Widget and background notification system
Login Integration, Advanced Grade/Attendence/Student View	Second iteration of the Aeries portal was integrated in to the IOS App to prevent security concerns across platforms
Media Integration/FTV and ASB Content view	Custom portal designed for viewing Foothill Media
Advanced CBC Mode (Second Iteration) AES Encyption for secure header requests and remembering login	IOS KeyStore integration and secure login-system using innate security

Personalized home-page and schedule view	Dynamic notificiation system resulting from personalized schedule
40% reduction in Crash reports (due to increased stability from encryption schemes)	Bullish stability after 2nd year launch resulting from overall cross-platform security

Ultimately, the cross-platform app line succeeded in connecting students, faculty and parents to campus assets and offering a highly-secure method for accessing their personalized schedules and managing their school-cultures. Improvements were most necessary in the development of the custom portal for centralizing campus/sports activities and in the development of a more secure encryption scheme using the Android *KeyStore* libraries. Particularly on Android, a computationally-minimal notification system required robust and efficient thread development to prevent overflows and additional power abuse.

Note: I developed all the vector assets used in the app and brought many UI concepts developed in conjunction with Foothill Media into fruition. Minimalism and efficient personalization was the key focus on the development of the app during its 3 year (and present) cycle.