New AViK Architecture

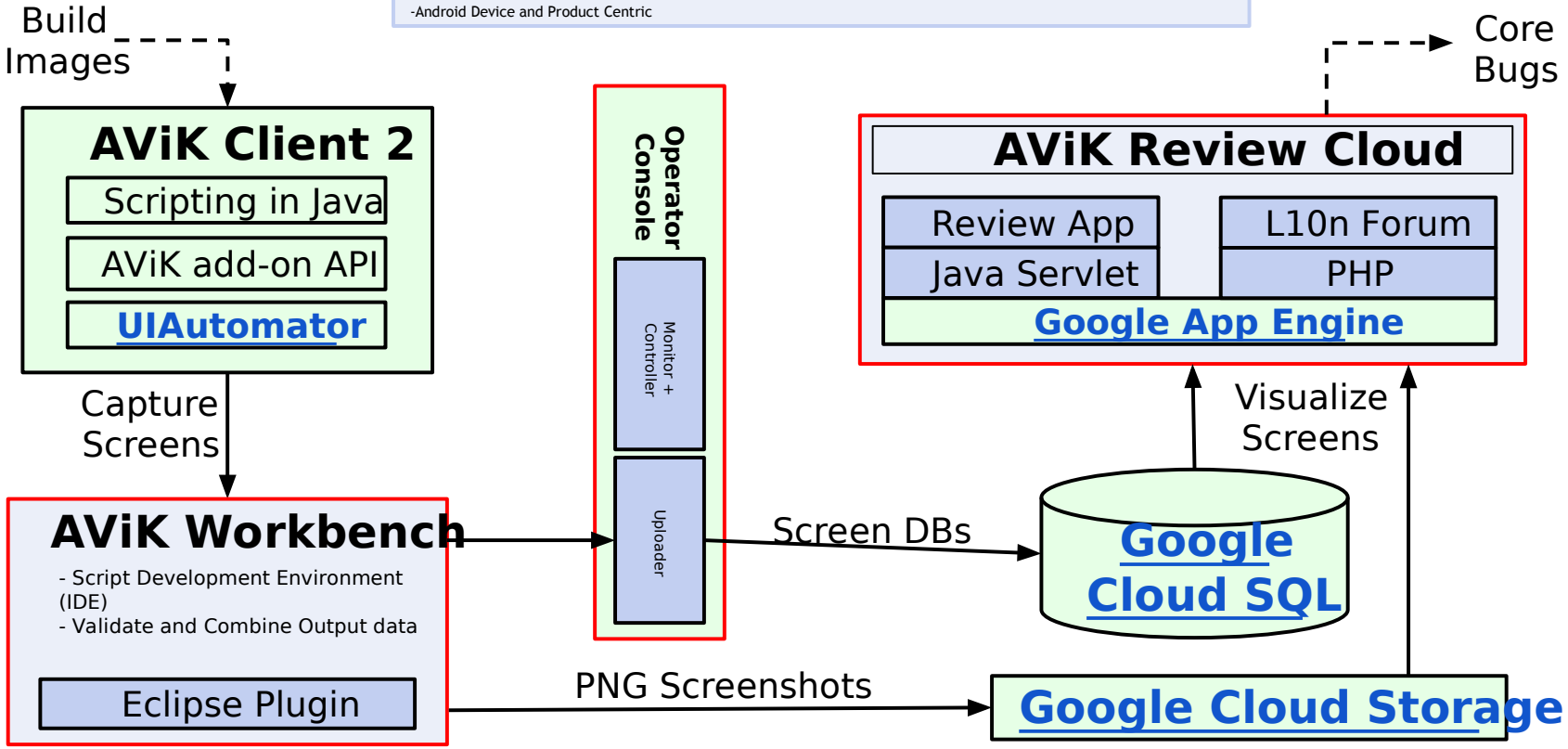
Diagrams
plus comparison with previous version

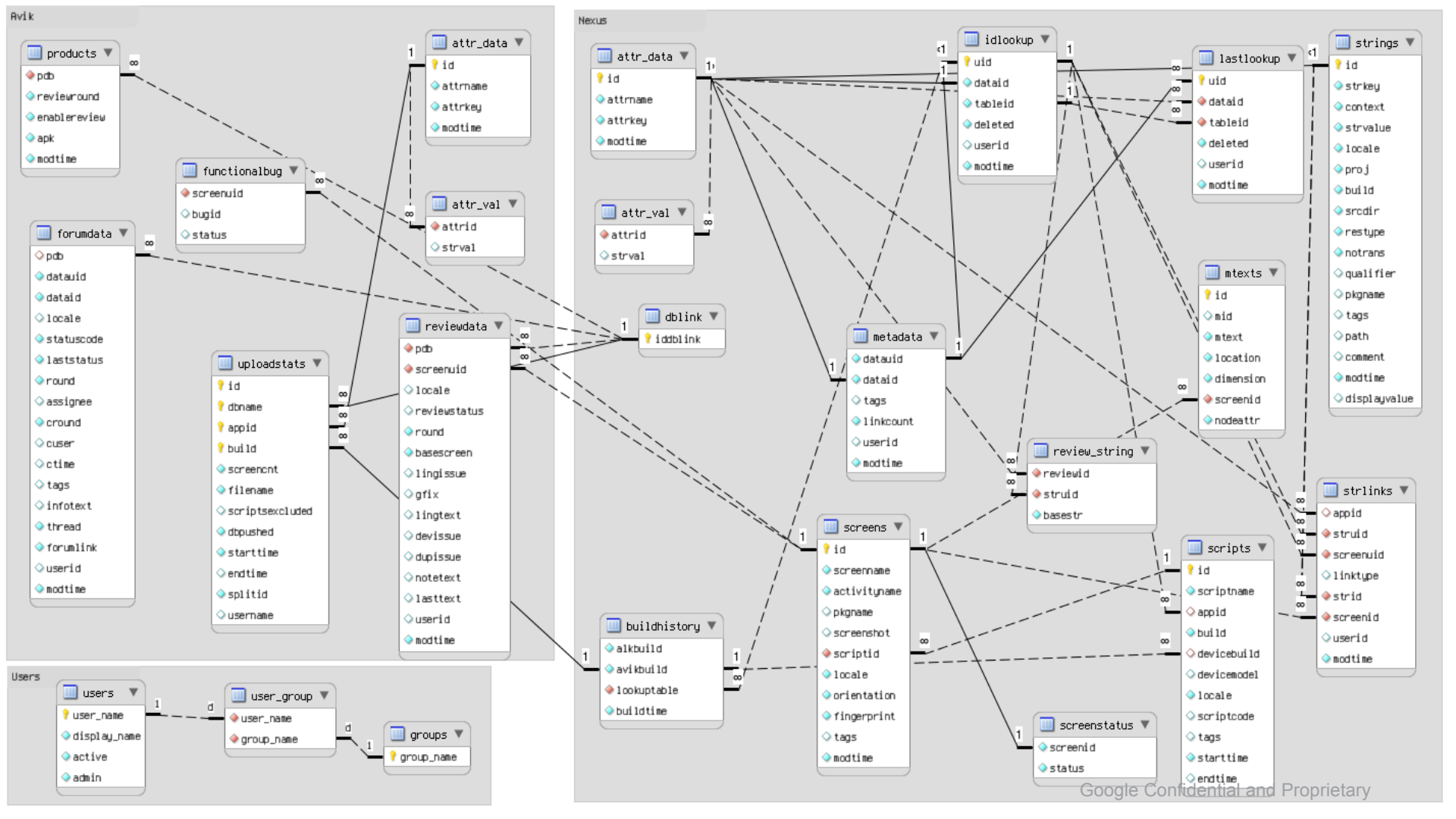
Author: Kevin Guerra guerrak@google.com

"Old" AViK Architecture

-33k~ LOC (lines of code) [Missing Workbench code]
-Dependency on 6 year old libraries
-High Maintenance (database, code)
-High Maintenance (administration)
-Android Device and Product Centric

motorola
a Google company




Avik
products
pdb
reviewround
enablereview
apk
modtime
attr_data
id
attrname
attrkey
modtime
functionalbug
screenuid
bugid
status
attr_val
attrid
strval
forumdata
pdb
dataid
dataid
locale
statuscode
laststatus
round
assignee
cround
cuser
ctime
tags
infotext
thread
forumlink
userid
modtime
reviewdata
pdb
screenuid
locale
reviewstatus
round
basescreen
lingissue
gfix
lingtext
devissue
dupissue
notetext
lasttext
userid
modtime
uploadstats
id
dbname
appid
build
screencnt
filename
scriptsexcluded
dbpushed
starttime
endtime
splitid
username
dblink
iddblink
Nexus
attr_data
id
attrname
attrkey
modtime
attr_val
attrid
strval
idlookup
uid
dataid
tableid
deleted
userid
modtime
lastlookup
uid
dataid
tableid
deleted
userid
modtime
strings
id
strkey
context
strvalue
locale
proj
build
srcdir
restype
notrans
qualifier
pkgname
tags
path
comment
modtime
displayvalue
mtexts
id
mid
mtext
location
dimension
screenid
nodeattr
metadata
dataid
dataid
tags
linkcount
userid
modtime
review_string
reviewid
struid
basestr
strlinks
appid
struid
screenuid
linktype
strid
screenid
userid
modtime
screens
id
screenname
activityname
pkgname
screenshot
scriptid
locale
orientation
fingerprint
tags
modtime
scripts
id
scriptname
appid
build
devicebuild
devicemodel
locale
scriptcode
tags
starttime
endtime
buildhistory
alkbuild
avikbuild
lookuptable
buildtime
screenstatus
screenid
status
Users
users
user_name
display_name
active
admin
user_group
user_name
group_name
groups
group_name
Google Confidential and Proprietary

NOTE: Schema and SQL queries as provided are inconsistent when a new database is created. Causing the new db to be incompatible with the current code base and the system unusable for the new database.

Pros and Cons for (not) using old AViK database schema and implementation

Pros:

- Well known mysql database technologies, therefore allowing easy transfer to other systems, (ok, sorry, this is actually not really a pro, I mean do we not rather use Google technologies?)

Cons:

- 2 database set design (Users db is unused), problematic distributed data, maintenance difficulties
- AEV (Attribute-Entity-Value) technique is unnecessary for a simple system, convoluted implementation. AEV is a technique that allows a design to add more attribute ‘types’ without having to add more tables. As implemented it uses yet more levels of indirection because the system can have many ‘nexus’ databases and this must be reflected in the data. The other level of indirection is through another table that keeps table information for each record identifier.
- User required/encouraged to create databases, main Avik db references and saves external db data, this creates data portability issues due to distributed dependencies
- Diagram above does not actually have the relationships that you see through the arrows implemented at the database level, but at the application level, this burdens the programmer by having to do a lot of extra work for referential integrity
- Schema has several legacy tables for a design that is no longer applicable to the current user needs
- Poor performance, due to complex queries and data access patterns, sql does not scale as well as Google’s Big Table.

...continued...

- Necessity for a database administrator, who must/should regularly back back or otherwise maintain the system. This would be a manual and labor intensive process. Google technologies already provide mechanisms that could be used in a manual process, or otherwise could be done programmatically in the application, in a more timely fashion, because libraries and API's are readily available for this purpose.
- System difficult to understand, implement, code, maintain and test, due to its complexity and inconsistent naming conventions
- Inconsistent queries in the current AViK modules. Queries that purport to do the same thing based on their names have differences, sometimes not minor. In one instance, this caused a breakage in the application (it's actually currently broken.) The breakage is manifested in the following scenario, 1) create database 2) upload screens 3) approve screens, 4) the background tasks that makes the new screens available fails. No error messages available, simply the screens won't show up.

NOTE: The original design actually follows a hybrid approach in which we can see the use of AEV only for some operations, and actually most of the usage patterns do follow traditional RDBMS design. However, because of these extra tables and extra queries, the architecture, design and implementation are quite convoluted and make it time consuming and laborious to understand the system, not to mention maintain.

NOTE: Please refer to AViK documentation currently available to understand how the system currently works.

Below are some links and quotations from the same regarding the original schema implementation.

wikipedia

Entity–attribute–value model (EAV) is a data model to describe entities where the number of attributes (properties, parameters) that can be used to describe them is potentially vast, but the number that will actually apply to a given entity is relatively modest. There are certain cases where an EAV schematic is an optimal approach to data modelling for a problem domain. However, in many cases where data can be modelled in statically relational terms an EAV based approach is an anti-pattern which can lead to longer development times, poor use of database resources and more complex queries when compared to a relationally-modelled data schema.

In the words of Prof. Dr. Daniel Masys (formerly Chair of Vanderbilt University's Medical Informatics Department), the challenges of working with EAV stem from the fact that in an EAV database, the "physical schema" (the way data are stored) is radically different from the "logical schema" – the way users, and many software applications such as statistics packages, regard it, i.e., as conventional rows and columns for individual classes. EAV tables conceptually mix apples, oranges, grapefruit and chop suey

Scenarios that are appropriate for EAV modeling

1) Modeling sparse attributes

2) Modeling numerous classes with very few instances per class: highly dynamic schemas

Stack Overflow - EAV Pros and Cons Stack Overflow EAV in SQL

EAV is notoriously problematic as it leads to severe deployment performance and scalability problems.

Wordpress - Mike Smithers, The anti-pattern AEVil database design

The EAV pattern looks very appealing at first glance. The upsides of EAV are usually most apparent during design and development. The downsides don't tend to manifest themselves until after the application has gone live.

SQL Central - David Poole on AEV data model

I have long had a dislike of Entity-Attribute-Value models within RDBMS's based on my experience troubleshooting and supporting them in various forms over my career. The premise of such a model is that they offer huge flexibility as objects that would usually require extensive physical modelling can be represented simply by records within a generic schema.

My experience is that the promised flexibility of such models is illusive and more than offset by the penalties and inconveniences they incur. With the rise of NOSQL solutions, native handling of XML within SQL Server and polyglot data solutions the limited number of appropriate use cases is dwindling still further.

A colleague of mine was studying for a masters degree in BI. One of the professors described EAV designs as write-only. Obviously such a description is an exaggeration but the comment was made to prove the general point about the difficulty of getting data out of an EAV design.

US National Library of Medicine - AEV Proper use and application

SQL Team - David M.

Issues with current AViK system

- Reliance in xml manifest files: Forces screen producer to list all ‘valid’ screens to submit and then to validate them a priori. This is an issue because our usage pattern will utilize tools that ‘crawl’ an application, being script agnostic as such. The manifest files restrict this use case by design, requiring the user to add ‘all’ screen types to this manifesto.
- Reliance on database creation. Difficult to maintain data due to distributed nature. This is an issue because, for reports and data analysis it would be necessary to ‘jump through so many hoops.’ Difficult to maintain.
- Android and product centricity limit its usage as well. The new system has been designed such that we account for all android data for whatever purpose, while keeping the system inner-workings agnostic to this extra data. This extra data might add value for the future, but as it is right now, it is not used for review purposes.
- Currently a database mismatch between Reviewer and Console that prevents its use.
- Lack of proper error handling and logging
- Brittle schema, architecture, design and code, absence of tests, and poor testability
- Performance, Reliability, Security, Code Maintenance Difficulties, Reliance on very old libraries,
- Very large code base for very simple functionality
- Old style of web applications, page centric, affects performance, instead of the new application centric approach
- Distributed logic and business rules causes maintenance difficulties and make the system difficult to understand
- Missing source code for the Workbench, would have caused us to use code in DBHandler which is likely out of sync, and re-engineer the Workbench in order to upload screens
- Reliance on the Workbench for screen submissions, creating challenges to extend or to maintain the system
- Incompatible code with Google Java Coding Style Standards, and due to the large code base would be time consuming to bring the code to compliance

Original AViK source code

SLOC	Directory	SLOC-by-Language (Sorted)
7266	Client	java=6610,xml=656 [NOTE: this is the data and screenshot generator]
13960	Console	java=12355,xml=1382,jsp=216,python=7 [Upload screens, create database, etc…]
6634	Reviewer	java=4913,xml=1621,perl=60,jsp=40
5345	DBhandler	java=4472,xml=873 [NOTE: this is a superset of code for some functionality in Workbench, which code we've not been provided]

Totals grouped by language (dominant language first):

java:	28350 (85.38%)
xml:	4532 (13.65%) [NOTE: mostly SQL queries]
jsp:	256 (0.77%)
perl:	60 (0.18%)
python:	7 (0.02%)

Total Physical Source Lines of Code (SLOC) = 33,205

Note that it does not account for HTML, JavaScript or CSS content. This should bring it close to 40k~

“New” AViK Architecture

-7k~ LOC (half~ auto gen)
-Object Oriented
-Best Practices
-Security
-Performance
-Latest Stable Technologies
-Minimal Dependencies
-Google Standards Compliant

Process Screens

GenDB from Dir Java

Screenshot generation

AViK Client 2 (former client) Java

Navigation: IAutomator Android

Optional

Eclipse

- Script (IDE)

Upload Screens

Visualize Screens

-Authentication
-Validation, Caching

Dart/ JavaScript

Web Browsers

Google Cloud Endpoints REST API Client

HTTP

Google Cloud Endpoints REST API Server

Java

Security Authentication[Oauth2], Authorization, Encryption

Reviewer
-Forum?
-Stats

Producer
-Upload
-Validate
-Revise

Admin
-Apps
-Users
-Groups

-Tasks
-Monitoring
-Validation
-Caching
-Morpheus/QM
-Effort Deduplication

Google App Engine

PNG Screenshots

Google Cloud Storage

Google Data Store

Core Bugs

“New” AViK Architecture

-7k~ LOC (half~ auto gen)
-Object Oriented
-Best Practices
-Security
-Performance
-Latest Stable Technologies
-Minimal Dependencies
-Google Standards Compliant

Process Screens

GenDB from Dir Java

Screenshot generation

AViK Client 2 (former Client) Java

Navigation: IAutomator Android

Former Workbench: Optional

Eclipse

- Script (IDE)

Upload Screens

Visualize Screens

-Authentication
-Validation, Caching

Dart/ JavaScript

Web Browsers

Google Cloud Endpoints REST API Client

HTTP

Google Cloud Endpoints REST API Server

Java

Security Authentication[Oauth2], Authorization, Encryption

Reviewer
-Forum?
-Stats

Producer
-Upload
-Validate
-Revise

Admin
-Apps
-Users
-Groups

-Tasks
-Monitoring
-Validation
-Caching
-Morpheus/QM
-Effort Deduplication

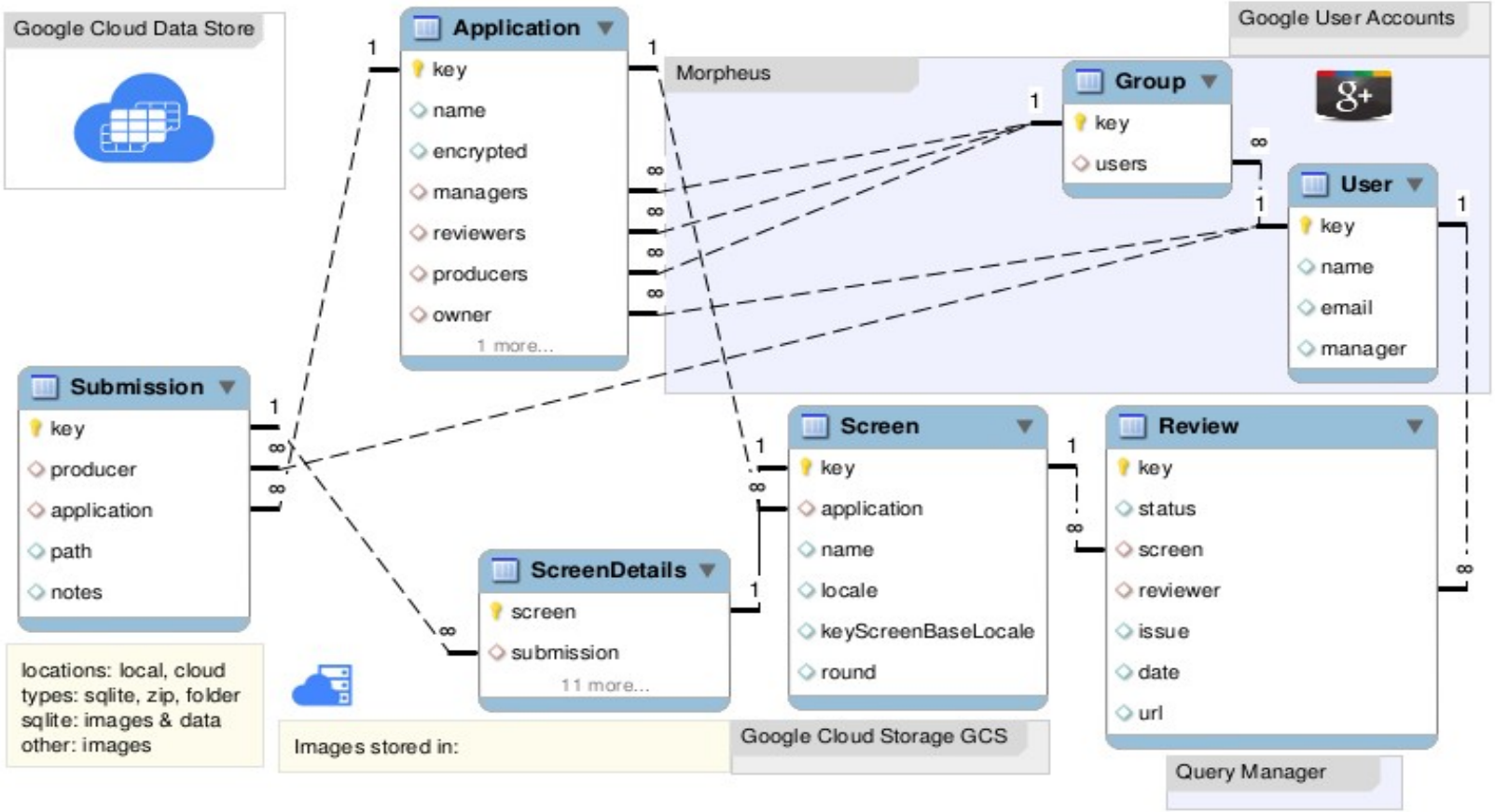
Google App Engine

PNG Screenshots

Google Cloud Storage

Google Data Store

Core Bugs


Google Cloud Data Store
Application
key
name
encrypted
managers
reviewers
producers
owner
1 more...
Morpheus
Google User Accounts
Group
key
users
User
key
name
email
manager
Submission
key
producer
application
path
notes
ScreenDetails
screen
submission
11 more...
Screen
key
application
name
locale
keyScreenBaseLocale
round
Review
key
status
screen
reviewer
issue
date
url
Query Manager
locations: local, cloud
types: sqlite, zip, folder
sqlite: images & data
other: images
Images stored in:
Google Cloud Storage GCS

New Features	Benefits
Removal of legacy schema	Performance, Comprehensibility
Data centric using Google Data Store, replacing database centric mysql approach	Performance, Scalability, Reliability, Ease of Use
Switch Android centricity to Application centricity	Ease of Use, Extend use to web apps. Note that system will still capture android data for future use and data mining
Discard the concept of product in the processing. Product information is available in Morpheus, Discard the concept of script	Comprehensibility. Note that product and script information will still be captured
Integration with Morpheus. Application, Product and User information, Role based Authorization	Synergy with existing systems, Avoid effort duplication
Removal of xml manifest processing, which imposes labor intensive workflows for developers and users alike	Performance, Comprehensibility, Ease of Use
OAuth2 Authentication and Role Based Authorization, Encryption of images for sensitive applications	Security
Separation of backend (Java) from frontend (Dart, JavaScript), API based interface (Google Cloud Endpoints)	Performance, Scalability, Comprehensibility, Security, Easier Mobile Device Support
Google Technology Stack, Object Oriented, Modularity, Caching, Validation,	Performance, Reliability, Comprehensibility, Security

Demo milestone: TBD (3 weeks)

- Complete API
- Integration with Morpheus for Authorization
- Complete UI: Screen, Issue Submit, User/Group/Application Admin, Upload Process, Approval Process
- Screenshot storage and retrieval
- Upload Process and Submit Approval Process
- Update AViK client

TBD (1 day)

- Frontend: Add caching, validation

V2

- Encryption of sensitive screens/applications (2 days, schema and transfer mechanisms already in place)
- Screen deduplication to avoid duplication of billable work (Checksum support already in place, FFT based would catch more)

LQA Lab Platform: Scope

- User Management:
  - v1.- Role-based authorization, view/edit permissions, sesame-compliant
  - v2.- Morpheus Integration
- E2E Upload:
  - v1.- Screenshot upload from local to DataStore/GCS
  - v2.- GUI-based screenshot validation
- Simplified database schema
- Authentication (OAuth2)
- Dart-based Front End:
  - v1.- Screenshot navigation, linguistic review, functional error submission, activity log, QM integration, upload approval
  - v2.- LQA reporting capabilities
  - v3.- TC integration
- Google Cloud End Points API Front End - Back End integration
-

- Database Schema 90%+

- API: 60%, currently transfers java objects, but will convert to a light weight API, with up to 20 calls

    - Resource identification: 90%; call implementation: 70%; file upload integration: 0%
    - Completion time left: 1 week

- Caching mechanism : 100%, Caching 10%

    - Note: tested with 1 API call, speed improvement 8 seconds to 200ms for medium data set.

- Authentication: 100 % (for both client and server)

- Authorization: 60 %

    - Users (who) and operations (what locales, what activities based on role) authorization

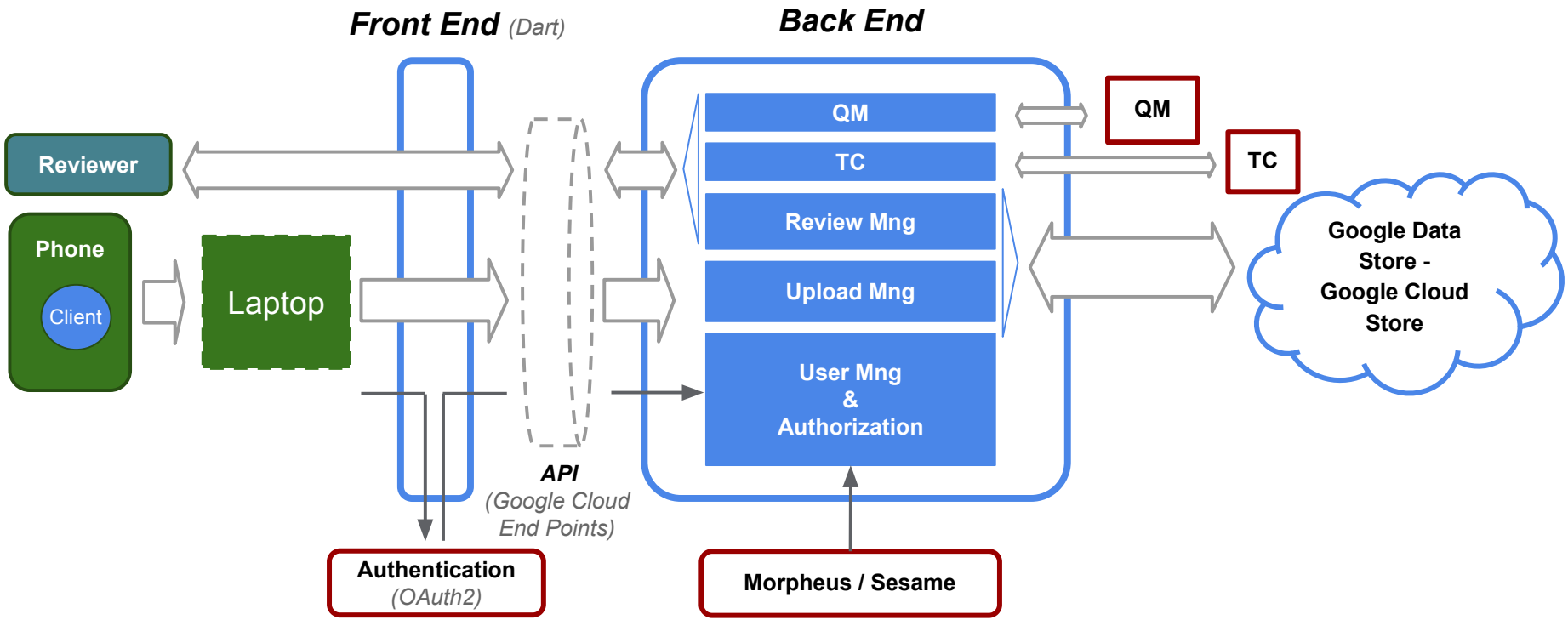
- Morpheus: 90% (two calls to #1: list for vendor + product; #2 get languages for this reviewer). Postponed

- Frontend: Prototype interface 10%, Authentication in place, Ability to call API in place

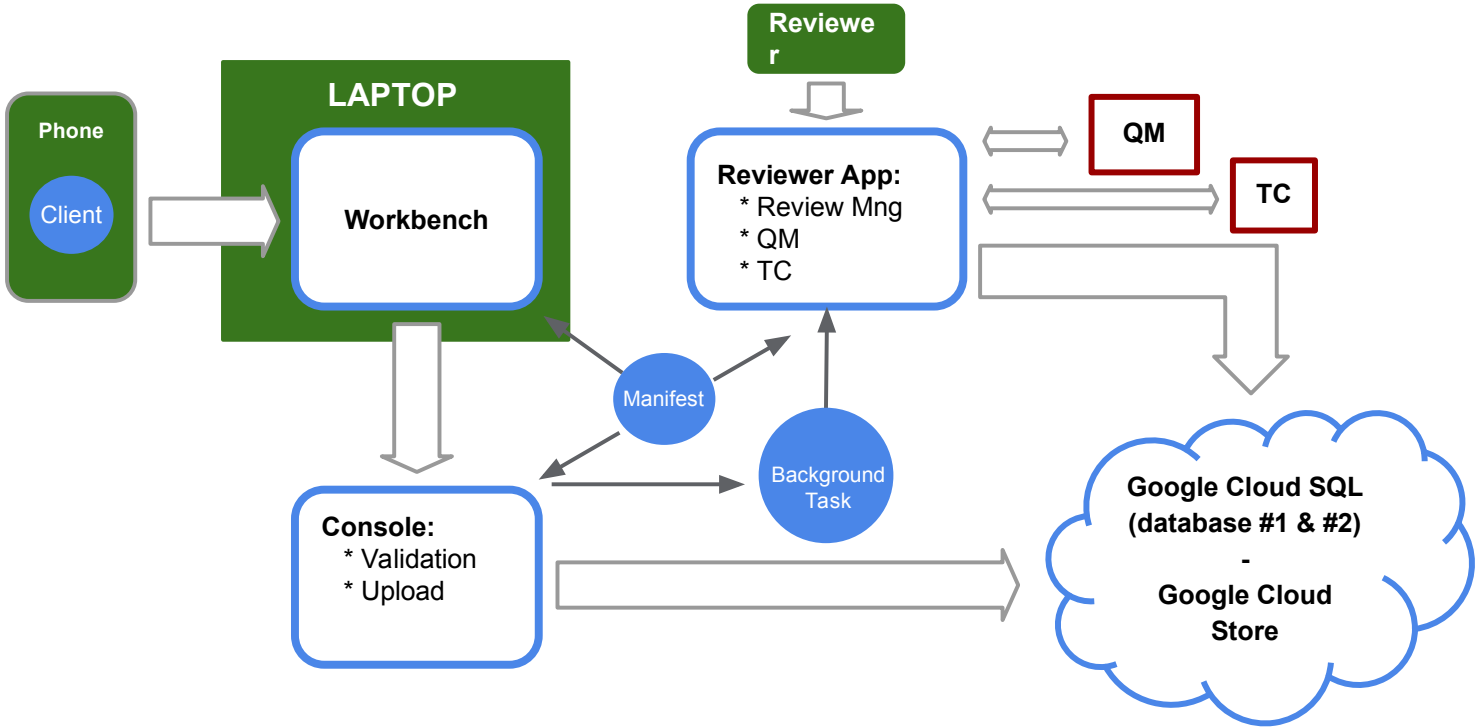
- Elimination of the Workbench and Console applications, only needed functionality moves to main app.

- Ability to upload screens independently from AViK client

LQA Lab Platform



AViK: Old Architecture



New source code

SLOC	Directory	SLOC-by-Language (Sorted)
1108	endpoints	java=1108 (autogenerated)
996	server	java=996
161	client	dart=161

Totals grouped by language (dominant language first):

java: 2265 (99.00%)

dart: 161 (1.00%)

Total Physical Source Lines of Code (SLOC) = 2,265 -

Subtract auto generated endpoints code 1,108

TOTAL = 1,157

NOTE: Dart client code, in 161 LOC has:

1) Very basic UI with panels

2) Authentication

3) Ability to call server API

4) About ½ of the code is actually data

5) HTML code is very small boilerplate, since there are no ‘pages,’ content is dynamic

6) JavaScript code is generated from Dart and it is consumed by the web browser as the client ‘application’