IIE-CIS MOBILE APP COMPETITION

StudyT

Vlad Predovic
Predovic900@gmail.com

Contents

Introduction

Industrial companies focus an ever increasing amount of resources nowadays on improving the efficiency of their systems. Existing processes are dissected and data collected in a meticulous fashion by Industrial Engineers before developing possible solutions for the future. However, a lot of the collected data is left isolated in project folders or physical data collection sheets. The result is that when teams are tasked with updating or improving another area of a process, they are unaware of existing resources provided by previous projects which could help them with their current analysis.

Time Studies are a widely used and respected, although sometimes controversial method of gathering quantitative data on processes throughout the manufacturing world. They allow someone to dissect a complex chain of events into procedures and gather information on the time it takes to accomplish these tasks.

StudyT seeks to make it easier for Industrial Engineers to detect process improvements within industry and automate much of the data collection and storage process. The application is enhanced by a combination of newer technologies which allow it to be flexible across phone platforms and manage robust quantities of data without affecting performance on the user's mobile device.

System Specifications

The StudyT is a hybrid, mobile, cloud-based application.   When choosing the tools used for development, the main issues kept in mind were speed, storage, and accessibility. To account for this, other features such as power optimization were given a secondary level of importance. The following sections go into detail over the choices that were made and how they come together to fulfill the needs proposed.

Development Platform:

The app was built using the Intel XDK, an integrated development environment designed for hybrid builds. A hybrid mobile application is essentially a web view entrenched within a native phone wrapper. Some examples of successful hybrid mobile applications in today's market are Instagram and Netflixes' mobile apps. The advantage of a hybrid application lies in that it draws some of the most important features from both web and native applications while sacrificing some power and flexibility.

Front-End Frameworks:

The user-based functionality of the application is built using HTML5 and JavaScript, a level of abstraction made possible because it is a web application. The application then employs a front-end framework known as Apache Cordova to interact with native features of a mobile app. The StudyT needed the capability to access a phone's GPS functionality. The Cordova framework allows the app to access this specific capability across platforms by detecting the operating system in question.

From a stylistic standpoint, the JavaScript library App Framework 3.0 is used to develop the UI of the application. One of the criticisms of web-based applications is that they lose the native flow and UI provided by building applications specifically for a platform which is done when using Android Studio or Xcode for IPhone. However, this library helps overcome this issue. Much like Apache Cordova did for the functionality, App Framework 3.0 detects what mobile operating system the application is running on and implements native element views. Essentially, the StudyT will act and feel native regardless of whether you are opening it on an IPhone, Android Phone, Windows phone, or Blackberry.

Back-End Capabilities:

The user application is integrally connected to an application program interface developed solely for use by the StudyT. The API is built in python using the Jinja2 framework, with plans of a web-based view of all time study data set for development. All connections to and from the mobile portion of the StudyT are done using HTTP requests from the mobile device to the server. The interface was built using cloud services provided by Google.

Outlined below are details on the current RESTful state of the API.

- Statelessness: In the body of any request made all the parameters and data are sent to the server needed to generate a response. No client context data is stored on the server temporarily.
- Client-server: Concerns of client and server are separated. The server is able to interface with multiple clients, while the client does not have any need to see the inner-workings of the server. Most of this is achieved through use of the google-app engine.
- Layered System: Client has no need to understand the different layers of the system. All functions interact with server to complete client's task without his knowledge of the interaction.
- Uniform Interface: The client can only request JSON representations at the moment and all representations on the live API website site are in text/html format. Although caching was attempted it is not currently active for test-development purposes.

Database Decisions

The StudyT interacts with the google App Engine to retrieve, create, and update all data related to the time studies by way of an internal API. The GAE is known as a platform as a service, providing robust database systems and access to Google's computing power on highly efficient querying algorithms in exchange for a monthly fee.

Possibly the most important functionality provided by the StudyT is the movement, creation, and storage of data at the click of a button. The successful implementation of this application at a large industrial company could result in thousands of transactions per day with data going back decades on certain processes. As a result, the application utilizes a non-relational database, the google Datastore, to manage all data. The main advantage behind this choice is the ability to scale enormously without serious compromises to performance. Given that the StudyT is built primarily to create data and then store it with significantly altering, and that the schema is simple, consisting of only four entities, the google Datastore provided a perfect balance of accessibility and growth capability for the data to be stored.

Model Architecture

Outlined below is a brief explanation of the entities that make up and interact with the StudyT. Ultimately, all combinations of data and functionality are derived from the objects made from these models.

TimeStudy:

Declaration of existing studies, the associated procedure keys, and a picture for visualization of the product being studied. All associated procedures have an iteration number of 0. Procedures with an iteration number of 0 are considered the root procedure, from which all other iterations are copied (*see TSInstance)*. The TimeStudy class is the blueprint object from which all instances are to be derived. A deletion of time-study class results in all associated instances being deleted.

Study Procedures:

The timed procedures that when combined in a specific manner make up a study. There are root study procedures from which all consequent procedures are derived. These have an iteration number and timestamp of creation that correspond directly with an instance of the time study they are associated with. Study Procedures can be combined in any way within a given TimeStudy. This decision was taken with the thought that studies can sometimes overlap at stages, sharing similar tasks for which comparative data could be analyzed.

TimeStudy Instances:

The real-time instance of a TimeStudy. Each instance is a copy of the main TimeStudy object with unique StudyProcedures associated directly to it through the creation timestamp and iteration. The timestamp is identical because both instance and procedure entities are inserted into the database at the same time. The location attribute is automatically updated when you run the instance using the native phone applications. The idea behind this is that it will be important for employees to know where the data is coming from when conducting studies in multi-factory companies.

App Users:

Simple verification for the user. Registering and logging in is done through the initial view on the mobile application. Each user has a set of unique TimeStudy instances which he has ran and can view privately. The *user* property of the TSInstance class is a mirror of the *username* property of the appUsers class. Every username must be unique.

Supplemental Information

A video demonstration of the application can be found here:

https://www.youtube.com/watch?v=zIs7GI7h2EY