Vlad Predovic

CS 496

Fall 2015

Evaluating the Strengths and Weaknesses of StudyT

The mobile application in question, StudyT, is designed to capture and allocate TimeStudies to a central database where they can later be viewed. It is primarily designed to function within factory settings, to be used privately by companies hoping to have an easier way to access, gather and improve the data of their processes. The adoption of this application would eliminate the difficulty of trying to keep track or catalogue data gathered manually during efficiency improvement workshops or operations. Additionally, it would allow users to compare their data quickly to that of results of previous TimeStudies over similar processes. This paper hopes to assess the current state of the cloud-backed application, StudyT, in regards to the following six qualities.

Contents

Portability

The mobile aspect of the application StudyT was built completely using web technologies such as HTML, CSS and JavaScript. The JavaScript library ‘App Framework’ was used for styling the UI of the mobile app. This library has the advantage of adapting the look and feel of your application “to the target platform” according to its informational website [1]. The library currently supports Android, iOS, Windows, and Blackberry devices and is compiled using the Cordova platform. This allows the web application to be portrayed with a native icon on a user’s mobile device.

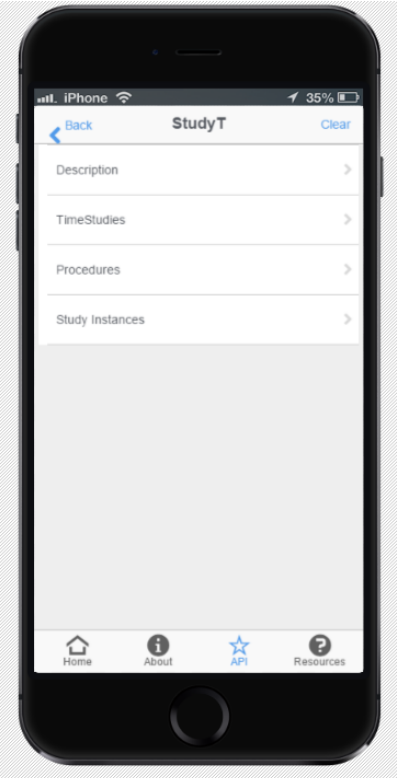
The Cordova implementation also adapts depending on the mobile device by implementing façade functions, allowing the application to access native features such as notifications, the flashlight, and GPS functionality. An advantage that comes with this flexibility is that the app does not suffer from interacting with platform-exclusive API's since the same code will access the native functionality of the many different devices. However, the application portrayed is essentially a web view, wrapped within the confines of the phone screen.

As a last point of clarification, since the app is a hybrid application, it is currently cleared to function smoothly on iOS, Android, and Windows phones with only minor tweaks to the code. This provides the application with a huge advantage regarding its source portability. However, since this was tested only on the available emulators in the Intel XDK, there is currently no knowledge of the application's binary portability. However it is assumed that it will work well on prior versions no notifications of deprecations regarding prior versions were found.

The API built to interact as the intermediary between the database and the mobile application is written in python using the Jinja2 framework. The API is also built using RESTful standards for calling or changing data. Although this is directly portable to a different database, there will have to be changes made to the code since it currently relies on the webapp2 framework to interact with the Google App Engine which houses the database.

Usability

When opening the application the user is confronted with a simple page with the option to login or register. At this point the registering process is extremely simple, requiring a username, password, and no verification. The name appears above the password and the same form can be used to either login or register. Once logged in, the user has two pages to navigate which describe the application and explain how to interact with it. They then have three different lists of items they can view, delete, or add items to. On the TSInstance page, if the user creates an instance of a study, the application redirects the user to a page where they can begin timing and save their results.



One of the advantages of this applications is that it is very simple from a user's point of view. The pages transition you smoothly to a central page from which all the different capabilities can be accessed with one click as shown in the figure to the right. If clarification is ever needed on the three functions of the app, the first button 'Description' serves to remind the user. Widgets on the bottom and top of the screen help the user reach any page quickly. These are built to mimic those popularized in apps nowadays so the user has a familiar feel to the app even upon first using it.

With low amounts of data tasks are done immediately. The API was improved by changing many of the queries that were previously fetching all appropriate values to just get the first one that appeared when it was appropriate. The queries will also time-out after a certain amount of time and display a message. This was done to significantly reduce the reaction time of the program when there are millions of entries and to avoid errors which would freeze the application.

However the mobile app still has some large deficiencies in terms of usability. Currently, in order to add procedures or studies, one must still input the Key id of the entity into the app. This renders the app almost unusable since a user would have to manually type in the keys of procedures or time studies in order to save these in the database. A fix was already applied to the Study Instances functionality regarding this issue. Now the application will query and automate the process of selecting which study to run with a click of a button. These improvements will be implemented for both the TimeStudies and Procedures views fairly easily.

Reliability

For StudyT, the reliability of the application is critical as its functionality is completely dependent on being able to access and edit entities on the database. The google app engine currently offers a 99.9% uptime a year for its servers or your money back. Additionally, GAE has implemented features for load balancing and data replication in order to reduce down time. The app currently uses the NDB which stores backup copies of data across multiple servers, greatly improving the reliability of the system.

Due to the nature of the program, it is not foreseen that there will be a high amount of network traffic. This is because the main functionality of the mobile-backed part of the application is running time studies and these take quite a bit of time to conduct in a formal environment. Therefore the one write per second limit should not be an issue. However, the web-based aspect of the application which is focused more on extracting and analyzing al the TimeStudy data gathered by the application could easily face issues since there will be much more functionality for the querying and manipulation of data. This issue can be alleviated by caching a copy of all a certain entity tree. All queries from then on will access the data on the client's personal computer.

In case of a failure when capturing remote data, the mobile app has built in error handling which notifies the user that their submission failed, This is done through a simple error handling function which alerts the user and interprets the given error code.

Performance:

One of the most prevalent drawbacks to hybrid applications is the loss of performance. Since the app StudyT is essentially a web view, it would have difficulty accessing the full capacity of a given phone's hardware or running intensive features. Luckily, that is not the case for StudyT. In fact, it completely makes sense that it be a web-view app because almost all its functionality rests on making HTTP calls to the API.

The application has the advantage of utilizing the Google App Engine's optimized querying system to return results from any request. AS mentioned previously, the API attempts to implement lighter queries as much as possible by using calls such as get() and count() which are much quicker than the fetch() command while providing sufficient results in certain situations. Additionally, the instances and procedures, two of the models in my database, are indexed in terms of their creation date. The sorting being done in chronological order allows the program to use quicker searching algorithms to locate certain studies completed in the past. The GAE also allows for queries to be sent asynchronously. The product could be improved by implementing these asynchronous queries to fire as soon as a user signs in as opposed to when they enter a certain page since there is data that is consistent viewed.

On the mobile side, there is room for improvement in the act of parsing the data returned from the calls to the API. The prototype of the application relies heavily on brute-forced regular expressions to parse and display whatever data the user is asking for. It is commonly known that regular expressions can eat up a lot of your CPU time as it tries to match with things you are not looking for. I perceive this causing an issue if this app were to reach an industrial scale as the expressions currently in place would match with an exponential amount of entities when searching for the correct values.

Security

The app is not designed to need high-level security. It primarily consists of taking time-studies and sharing the results with others. Some basic security was added in for creating user accounts, as it far too easy to access a list of users at the moment. When registering, the password is hashed and encoded. Therefore the user is the only one who actually knows their password. The data is then sent by a POST request to the cloud where it s saved as an entity. When logging in, the app queries the database searching for a match of the username and the encrypted form of the password. The figure below shows how the data appears in the database. In this case, the encrypted password translates to the word 'test'.

A robust improvement to user identification would be to implement the OpenID API provided with the GAE and have users sync their accounts with google. Implementing OpenID could also improve the Usability of the application as users would not have to verify and create new accounts. A large amount of professionals nowadays already use google accounts. However at the same time, in order to not compromise their identities within the workplace, implementing OpenID as opposed to building a unique identification system could be detrimental for the use of this application.

The app still has no features designed to mitigate access to studies or procedures. This would be improved by associating users with a company entity. One way of implementing this would be using the company itself as the root level of an entity tree in order to encompass all the data within the given scope. A user is limited to access the study instances which they have created themselves, by having the service query the database for matches with the current username logged in.

One of the greater deficiencies than nullifies the security of the app is that users input TimeStudies, Procedures, and Instances, as well as identify themselves using the keyboard. There is no method in place to check the string inserted by the user or nullify an injection. Instead, I have circumvented this option by implementing parametrized queries. For the same reason the application does not utilize SQL-style queries.

The application is secure from an API communications standpoint. The only interaction it has with an alien service is using native functionality from devices to garner the current geolocation of the device. During this interaction, no data is sent to the API, as it is primarily a GET request. The information returned is then parsed for the longitude and latitude and this is saved in the database under its respective entity.

Interoperability

Being built as a hybrid application allows StudyT to be implemented across phones, interacting with the native API's of multiple different systems without having to significantly alter the program. Basing the app in HTML5 works exceptionally well for multi-platform development since most browsers on mobile apps can render HTML5 content.

At the same time, because of this the application loses the ability to use many of the advanced featured provided for native phones, which could hinder future development by having to develop costly and complex functionality.

The application fails at successfully implementing a data-formatting standard. This is probably the biggest upgrade the mobile application and API need before development can continue. Although a previous version successfully implemented the JSON standard for simple calls, the current prototype does not do a good job of transmitting the data in a JSON format from the API to the mobile application. Improving this feature could be considered critical. The features provided by the mobile app are all rendered using JavaScript, which is known to have powerful tools to parse and manipulate JSON formatted data into objects.

Conclusion

In conclusion, the StudyT is a conceptually simple and minimalistic application from a user's perspective. It thrives on providing a straight-forward interface and the majority of its functionality is based on interacting with a web-based API and the database. In its current state, the app is intuitive but difficult to use as it requires that you type in keys in order to make changes to the database. Security is currently at a minimal point and needs to be developed further before the app can be considered for serious use. StudyT's greatest asset probably lies in the portability of the application. It uses a minimal amount of native API's and is fairly lightweight, with the brunt of the work being done on the cloud thanks to the Google App Engine.

References

[1]: "App Framework Explained." *App Framework*. Intel, n.d. Web. 08 Dec. 2015.