Project Arlo: Multi-Tenanted Social IT Helpdesk

by

Tim Guibord

Submitted to the
Faculty of the School of Information Technology
in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Science
in Information Technology

© Copyright 2014 Tim Guibord

The author grants to the School of Information Technology permission
to reproduce and distribute copies of this document in whole or in part.

Tim Guibord

Russell E McMahon

Russell McMahon, Faculty Advisor

Date

4/17/2014

Date

4/17/2014

University of Cincinnati
College of
Education, Criminal Justice, and Human Services
April 15, 2014
Acknowledgments

I would like to thank my faculty advisor Russell McMahon and Senior Design Project Instructors, Patrick Kumpf and Jim Scott for the outstanding guidance on this project.

Karolyn Schalk, Darren Heath, David Garrigan, Thomas Anderson and the rest IT group at Apex Supply Chain Technologies for their help in developing the requirements for the solution and performing functional testing.

The Cincinnati Ruby Brigade, the development team at Top Gun and Jim Anders for their technical support and guidance.

My wife Katie and the rest of my family for continuing to support me and driving me to finish.
Abstract

Project Arlo, a Multi-tenanted, social IT Helpdesk web application, was designed to help provide quality end user IT support to businesses of all sizes. The idea for Project Arlo was conceived during a frustrating search to replace an existing IT Service Management product for a rapidly growing small business. Unlike current solutions, Project Arlo leverages a network of IT professionals to provide solutions to common IT end user problems. In addition, Project Arlo is a powerful cloud-based, SaaS solution which provides a business an individual, customizable tenant for providing quality end user support to their organizations. IT professionals will find Project Arlo to be an excellent professional networking tool; providing a single place to collaborate and build their professional profile. Project Arlo is built on a modern technology stack, utilizing tools like Ruby on Rails, PostgreSQL, and AngularJS.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>i</td>
</tr>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>iii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>iv</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Problem Need</td>
<td>2</td>
</tr>
<tr>
<td>Technical Areas</td>
<td>4</td>
</tr>
<tr>
<td>Solution Definition</td>
<td>4</td>
</tr>
<tr>
<td>User Profile</td>
<td>6</td>
</tr>
<tr>
<td>Design Protocol</td>
<td>9</td>
</tr>
<tr>
<td>Resource and Logistics</td>
<td>12</td>
</tr>
<tr>
<td>Software and Hardware Needs</td>
<td>12</td>
</tr>
<tr>
<td>Budget</td>
<td>12</td>
</tr>
<tr>
<td>Timeline</td>
<td>13</td>
</tr>
<tr>
<td>Conclusion</td>
<td>14</td>
</tr>
<tr>
<td>References</td>
<td>15</td>
</tr>
<tr>
<td>Notes</td>
<td>17</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1. User Diagram</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2. System Diagram</td>
<td>11</td>
</tr>
<tr>
<td>Figure 3. Project Budget</td>
<td>12</td>
</tr>
<tr>
<td>Figure 4. Effort Estimation</td>
<td>13</td>
</tr>
<tr>
<td>Figure 5. Project Timeline</td>
<td>13</td>
</tr>
</tbody>
</table>
1. Introduction

At the beginning of June 2013, my employer, Apex Supply Chain Technologies, approached me about replacing the current IT Helpdesk support and ticketing solution. At the time, the company was using Spiceworks, a free, self-hosted solution which was originally implemented in April of 2011. Spiceworks no longer worked for Apex for two main reasons. First, Apex simply out grew Spiceworks. Spiceworks user management was designed for a single domain Active Directory structure, but Apex was quickly evolving into a multi-domain forest. We arrived at a point where we simply could not manage all organization users and had to employ several manual processes to service users that were not part of the primary domain. The second reason for replacing Spiceworks was Apex’s plan to move to a cloud-based infrastructure. Spiceworks required our organization to maintain an on-site Windows server. Due to this, Apex was required to use limited, internal resources to ensure that Spiceworks and the server software were constantly up to date. As a small business, IT resources are at a premium and even a few hours a week can be an expensive support cost.

As I proceeded with the discovery and evaluation process, I became more and more frustrated with the available choices. The solutions that fulfilled all of our requirements had a base cost that was far out of our budget. The solutions that fell within our budget did not meet many of our requirements and were really nothing more than a more expensive Spiceworks. There really was nothing commercially available that was tailored to growing small businesses. In the time since Spiceworks was implemented, Apex grew from about 40 employees to a multi-national organization of about 300. With expectations that the company would again double in size over the next two years, a scalable, cloud solution that could grow with us was imperative.
2. Problem Need

The central issue with the current selection of commercially available IT helpdesk ticketing solutions is that they fall into one of two categories: Small, self-hosted solutions and large, enterprise level cloud solutions.

Although more affordable for small businesses, small, self-hosted solutions, such as Spiceworks and JitBit, require a business to have the technical expertise on staff in order maintain the long term health of the solution. Many of these products claim that they are install and run, however, the technical reality is some knowledge of maintaining server hardware and software would be required in order to ensure the solution remains in ideal working order. Our experience with Spiceworks showed that we needed to reconfigure the application every time we made a change to our domain. Furthermore, periodic updates and changes to our workflows required the application database to be backed up and then restored after work was complete. Small, self-hosted solutions are not designed to scale with growth. One key requirement for Apex was a solution with Single Sign-On (SSO) capabilities. Although many of the small, self-hosted solutions had Active Directory integration, they were not designed to function in a forest structure. These products do not do more than replicating AD users with its own users table within an embedded database then periodically checks for changes. In Apex’s experience, this methodology of user management was clunky and never capable of keeping the user base correctly up to date. Multiple times, the users table would have to be manually edited to ensure all organization users had proper access. A solution which utilizes federation services and SAML messages, or allowed for access via third party OAuth identities are needed to make user management within the application as seamless as possible. Large, enterprise level cloud solutions, although covering all of the necessary requirements for a replacement solution, presented a problem when it came to initial cost of use. For example, ServiceNow, one of the leaders in this area, had a $50,000 a year starting cost for our organization. With the cost floor
being that high, the long term cost as our organization continued to grow was not manageable. We were looking at possibly spending close to $100,000 after the second year. This was an expense that we could not justify budgeting for our IT ticketing application. One of the biggest issues with products like ServiceNow is that the pricing model was based on the number of agents that an organization had on staff. Each agent user was allotted so many support users and any other support users above that allotment were added as a surcharge. In the case of ServiceNow, it had a 20:1 support to agent user ratio. The current ratio at Apex is about 100:1 support to agent users and the expectation is that ratio would only increase. Another large, enterprise solution we evaluated was Zendesk. Although the price point of Zendesk was more affordable, about $5,000 a year at our current need, the solution was designed for large customer support outside of the organization. Zendesk would have required all of our support users to register a support account with Zendesk which would have no relation to any SSO service that our organization utilizes. Furthermore, Zendesk is designed mainly around phone agent support whereas our organization provides support mostly through email and the online ticketing application. We did not want to pay for features that would not be utilized. With being a small organization, the desire to have advanced self-service options was critical in any replacement solution. Almost all of the solutions that were evaluated had some sort of knowledgebase feature; however, many of these features were not very user friendly and still required some technical understanding of the issue. Our organization has attempted to utilize knowledgebase self-service with little success. We have found that users end up submitting tickets for issues that are well documented in our knowledgebase. Although many of these tickets are low severity and take minimal effort to resolve, they are still using support resources which we are not rich in.

To conclude, we have found many issues with the current offerings in the IT Helpdesk ticketing products. None of the more affordable products were cloud-based meaning that on-site hardware
and software would have to be supported and maintained. Products that were cloud-based were
often unaffordable and the solutions that did fall into budget were not primarily designed for the
type of inner organization IT support we provide. Finally, many solutions evaluated, that were
within our budget, did not offer the single sign-on capabilities require, nor did they offer robust
and dynamic self-service options that would greatly assist a small IT support group.

3. Technical Areas

This project encompasses the primary areas of the University of Cincinnati Information
Technology program: software development and database development.

This project required the development of a complex Web application as well as the configuration
of the server environment that will host the application. The application was built using the
Model-View-Controller design paradigm of modern Web application development. The project
also required significant understand of front and backend software development. It will require
a dynamic frontend using CSS 3 and cutting edge JavaScript frameworks. The backend of the
application will require the building of a robust RESTful Web service layer as well as building a
stable application that can support many concurrent connections.

The project required the design and development of a large backend database. In addition, to
achieve mutli-tenancy, the project utilizes advanced database features such as partitioning and/or
dynamic schema creation.

4. Solution Definition

In order to solve the problems detailed in the previous sections, a cloud-based, multi-tenanted
IT Helpdesk Web application was designed and builds. With the purposed functionality to be
discussed, this application will be able to fulfill the needs of a growing small business with
limited resources to spend on IT support staff. In addition, this solution will give organizations
a new means of providing top level IT support to its users. The solution will also serve as a professional networking tool that IT support professionals can use to connect and collaborate on complex problems.

This solution will live in the cloud. Utilizing Salesforce.com’s cloud platform as a service product, Heroku, application users will never have to install or maintain local hardware and software. The Heroku development ecosystem also provides the application with numerous add-ons which will help facilitate higher level functionality such as single sign-on and full text searching. Heroku also has native integration with PostgreSQL which will help achieve full multi-tenancy.

This solution will be multi-tenanted. Like many software as a service products, this solution will partition users’ data, yet distribute it through a single application instance. Using service-oriented architecture (SOA), the application will be able to deliver a unique experience customized to its individual tenants. Furthermore, utilizing SOA and multi-tenancy, the users’ application data will be far more secure. Using this approach will provide users with a greater level of customization that is not available in many of the current commercial off the shelf products. Also, although the application will initially only be developed for full Web browser, following the SOA principles and responsive design will allow mobile support to be quickly implemented.

This solution will be social. The multi-tenanted design of the application will allow users to submit issues to both a private forum (group of support professionals within the user’s organization) as well as to a public forum (all technical users for the application). Allowing issues to be opened up to a greater public forum will expedite issue resolutions for common end user issues. It will also allow technical support users to collaborate amongst a community to discover new and innovative ways to solve common and complex IT issues. “Gamification” of issue resolution will give IT support professional incentive to go outside of their own
organizations to provide support to users from organizations with smaller, or no support staffs. Through “gamification”, this application will also act as a powerful networking and professional portfolio tool that IT support professionals can use in the advancement of their careers.

5. User Profile

Three primary user profiles have been identified: normal end or common users, technician users, and administrator users.

Normal end or common users are individuals that will be using the application as a means to obtain IT support. Basic functionality such as registering an account with the applications and logging will be available. These users will also be able to perform keyword searches for issue solutions that are either public or have been submitted within their own group. The key differential with these users as opposed to the other profiles is the ability to submit new issues needing resolutions. In addition, this user profile will be able to add comments to submitted issues and accepted solutions. This functionality will be a means for technicians to gather more information on the user submitted issues and get feedback regarding solutions. The final set of functionality normal end users will be able to perform ties into the “gamification” of the application for technician users. End users will be able to not only flag submitted solutions to their issues as accepted, but can also pass incentive to technician users based on the quality of support they have given.

Technician users are IT support professionals whose primary role in the application is providing solutions to submitted issues. These users will be able to submit solutions to issues that are either public or submitted by member of their organization’s group. Technician users will also be able to open issues in their organizations group to the public community of the application in order to help assist on complex issues that they might not have the necessary expertise in solving. Also, these users will be able to comment on submitted issues as a means of gathering
more information regarding the problem the end user is experiencing. Technicians will also be able to comment on submitted solutions as both a means to gather feedback on their own solutions, but also on other technician’s solutions as a way to collaborate on solving problems.

Since there will be a social aspect to the application, Technicians will be able to maintain a profile which can be viewed by other technician users. These users can also connect with other technicians to build a network. Technicians can use this feature as a way to tap into others that hold expertise in different functional areas of IT. As part of the “gamification” feature, technicians can give incentives to other technician users that assist them in solving issues. Technicians will also be able to display various pieces of “flair” that will be gained through obtaining specific incentive levels.

Administrative users are specialized technicians who serve a leader role in their organization. These users will be able to establish private groups within their organization as well as manage the users, both end and technician, in each group. Finally, admin users will be able to add a level of customization to the application by creating specialized issue forms and pages that can be utilized by the groups they manage.
Figure 1. User Diagram
6. Design Protocol

Since SOA development principles will be utilized, the application will have a distinctive front and backend design protocols.

The backend of the application will be written in Ruby on Rails. It will use MRI 2.0 for the Ruby version and Rails 4.0. The Ruby on Rails framework will allow for rapid agile development as well as offering the best framework for creating a SOA application. A RESTful Web services layer, include data transfers via JSON messages, can easily be configured within the framework without the need for additional 3rd party libraries. Ruby on Rails also has a build in testing framework which allow for easy test driven development to be employed. Using test driven development methodology will accelerate the development time and will integrate well with the agile project plan. In addition, the JavaScript libraries will be written in CoffeeScript. CoffeeScript is a language that compiles in JavaScript when applications assets are complied. CoffeeScript will allow for rapid JavaScript development. The dynamic stylesheet language LESS will be utilized in the creation of the application styling. LESS compiles into CSS and offers automatic cross browser capability when compiled. Using LESS will reduce the time needed to make the application work across all browser platforms.

This application will use PostgreSQL for its data backend. PostgreSQL integrates well with Ruby on Rails and is the preferred database for Heroku applications. In addition to its compatibility, PostgreSQL offers to key functionalities which will aid application development. First, PostgreSQL has schema functionality. Schemas offer a partitioning of data within a single database which will make multi-tenancy possible. There are many Ruby libraries that will help utilize this advance feature of PostgreSQL. Next, PostgreSQL offers full text search functionality. Full text search will allow the self-servicing functionality planned to be possible as it will allow users to begin to type issues and quickly be presented with associated solutions.
The backend of the application will be built on the Heroku platform. Heroku utilizes the Nginx HTTP server for speed and efficiency. This application will mainly be using Heroku’s Nginx instance for request routing and load balancing. The application will utilize the Puma Web server to deliver server content to the client. Puma is Ruby based Web server that designed for concurrency which will be necessary to deliver a multi-tenanted, SOA application.

The frontend of the application will be written in Google’s AngularJS framework. AngularJS extends the MVC architecture to the client-side of an application. AngularJS creates its own client-side model and controller objects and communicates to a SOA application via AJAX and JSON messages. Using AngularJS will give modern sleekness to the application and increase the overall user experience.

To assist in creating a responsive design, the application will use Twitter’s Bootstrap framework. Bootstrap offers a prebuilt grid framework for rapid design and responsiveness for multiple viewports. Bootstrap also offers many prebuilt design elements and functions which will add greatly to the overall user experience.
Figure 2. System Diagram
7. Resources and Logistics

Resources on this project were minimal as many open source elements are being utilized. The biggest resource need was person hours.

8. Software and Hardware Needs

This project did not require any special software or hardware to be purchased. A development server was provided free by Heroku. A server to host the pilot application on was obtained via a modest monthly service charge from Heroku.

9. Budget

The following budget was necessary for completion of the project.

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>Period</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heroku Pilot Server and Database</td>
<td>$44/mo</td>
<td>2 months</td>
<td>$88.00</td>
</tr>
<tr>
<td>Domain Name</td>
<td>$10/yr</td>
<td>1 year</td>
<td>$10.00</td>
</tr>
<tr>
<td>SSL Certificate and Enablement</td>
<td>$20/mo</td>
<td>2 months</td>
<td>$40.00</td>
</tr>
<tr>
<td>Development Hours</td>
<td>$1500/mo</td>
<td>4 months</td>
<td>$6000.00</td>
</tr>
<tr>
<td>Total Cost</td>
<td></td>
<td></td>
<td>$6134.00</td>
</tr>
</tbody>
</table>

Figure 3. Project Budget
10. Timeline

The project will have 5 phases, each with a specific number of hours estimated to the effort needed to complete the phase:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Hours</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Analysis</td>
<td>40</td>
<td>Completed 10/1/2013</td>
</tr>
<tr>
<td>Requirements Gathering</td>
<td>40</td>
<td>Completed 10/31/2013</td>
</tr>
<tr>
<td>Application Design</td>
<td>40</td>
<td>Completed 12/31/2013</td>
</tr>
<tr>
<td>Implementation and Testing</td>
<td>400</td>
<td>Completed 4/1/2014</td>
</tr>
<tr>
<td>Validation and Deployment</td>
<td>40</td>
<td>Completed 4/15/2014</td>
</tr>
</tbody>
</table>

Figure 4. Effort Estimation

Figure 5. Project Timeline
In addition, the “Implementation and Testing” phase will follow agile development methodologies. The phase has been broken down into nine sprints each lasting two weeks. I have estimated 140 total story points for the project which will require a sprint velocity of 15 points per sprint.

11. Conclusion

The goal of this project was to build an IT Helpdesk ticketing solution that a small business can easily and rapidly implement, but can scale to the needs of a larger enterprise. By promoting social IT and a collaborative approach to problem solving, I believe that an application can be developed that allows organizations to focus more on the quality of support resources and less on total head counts. This project will demonstrate many modern application development features such as multi-tenancy, SOA, and single page application development. Finally, it was developed on a modern technology stack that is conducive to agile development, which will allow for rapid development and ease of maintenance over time.
References


Notes