# Patient News - T661

### What scientific or technological uncertainties did you attempt to overcome? (Maximum 350 words)

The technical uncertainties we faced in the fiscal year were related to business expansion, which means service expansion, tools integration, database performance enhancements, and the integration of cutting edge tech, like artificial intelligence (AI), to capitalize on the full potential of our application.

We also begin the process of shifting our entire application from Ruby on Rails © to Node.js © and AngularJS ©. We needed a more robust platform that could do more for us, and we hit a wall with Ruby on Rails. We had to figure out how to add the new functionality while swapping out the application's code base.

Our technical uncertainties were related to the pursuit of competitive advantage. Our business is in a constant state of growth. The application and AI integration were to provide a richer tool set for our customers. The application brought the graphical rendering of data so our clients could see in a graph what the data meant, and the AI allowed us to grow our business by analyzing missed telephone calls. This is the competitive advantage in our industry, constantly seeking new and relevant expansions of our core business.

In these integrations and expansions we often hit walls so daunting it seems like we are reinventing our whole application. This creates tremendous uncertainties because there are no easy integration paths for the expansion services. We have had to reconfigure our application from the base code up to facilitate our competitive advantage. All of the work had to be planned, researched, tried, failed, and tried again within a new work scope until we got it right. There were no easy, out-of-the-box solutions.

### What work did you perform in the tax year to overcome the scientific or technological uncertainties described in line 242?

### Tableau Integration

We needed to integrate our application with Tableau ®.  
Our users needed to see Tableau reports in the MPD (Patient News Marketing Application) dashboard without a login to the Tableau server. The uncertainty was accessing the Tableau server without a username and a password. The communication between Ruby on Rails ® — used to develop our application — and the Tableau server was an unknown. The integration needed to enable Security Assertion Markup Language (SAML). We investigated the possibility of an existing Ruby plugin. No such plugin existed. This necessitated the construction of a solution that would manage the connection between Tableau and our application. we tested some other implementations where we directly used the SAML protocol to access Tableau.

We tried to use the civisanalytics/tableau\_api plugin on GitHub ®. It allowed a connection to the Tableau server API, and we could query the data, but we wanted to directly display charts generated by Tableau — embed the generated code into our application. The plugin worked, but users had to login to Tableau with *their* credentials to see the charts. Our requirement was to implement a *single sign-on* that allowed our users to see the Tableau data.

Our intent was to create a connection between our Ruby on Rails application and the Tableau server using the SAML protocol. The implementation was unfinished as we only did research on how to use the SAML protocol and the Single sign-on functionality of Tableau. What we learned from this experiment/investigation was related to the configurations on the Tableau server side that would allow the SAML connection by our users. We had multiple failures related to SAML configurations, but our hypothesis refined itself. We succeeded by using a recent functionality Tableau developed called *Trusted Authentication* — basic in terms of API implementation, but our process to get to this working required a lot of trial and error.

### Voicebase

Another obstacle we faced was implementing a method to determine *missed opportunity* calls and *no score* calls. We worked with a company called Voicebase ® to do the Artificial Intelligence (AI) on this but we were hands-on involved. We analyzed calls to identify details defined as *missed opportunities*. We wrote scripts to train the Voicebase AI.

### System Performance Improvements

We experienced performance issues. The system was very slow. We deconstructed everything to see our load issues. Large datasets were used. Large import jobs were running in parallel with the application's normal usage. The database was over-loaded. The queries performed were very slow. End user performance was very slow. The user interface (UI) was lagging.

We tried a lot of approaches. We used materialized views to load the data and save it in the needed format — as flat tables. The queries were simpler and without complex joins. The issue was that these views needed to be refreshed from time to time so the user had the latest data. The refresh of a single view was took a lot of time. We didn’t have the guarantee that we were showing recent and reliable data to the users. We studied and employed the latest features of PostgreSql ® — filters — to speed up the queries. We tried existing tools in the database to speed up the queries, allowing queries to select data faster. We replaced where statements and sped up retrieval time, but this wasn’t fast enough. Page load decreased from 30-40 seconds to 20-25 seconds. We exhausted all the database features and had to experiment with developing custom methods. We optimized the queries to retrieve and lookup the optimum amount of data. Large queries looking up data from four to five tables. We experimented with moving conditions between queries to optimize the query structure — joins and things like this — to see if it would speed up the searches. The page then loaded in 10-15 seconds. All the queries were triggered at the same time, and, when they were all ready, the data was displayed.

We added indexes in the conditions of all relevant fields used in queries, This was a very good improvement, especially for large tables. In the case of joins, we moved conditions form the where clause to the join on clause to restrict the number of rows returned. In the case of inserts into the database, we used multiple rows inserts. We don’t do a single insert statement per row. We added multiple rows at once. This improved the general database performance, as it doesn’t stay too much on inserts and updates and can work more on the queries. We removed unnecessary columns from the SELECT statement, and we adopted a policy of not using Order By unless necessary.

We changed the loading of the data in the application to asynchronous so users don't have to wait for the whole page to load, and they see data as it’s coming from the database. This made the page load much faster. We split the page into different sections, with each section doing a request to the server, and, when the data was ready, it was displayed. We now call the queries as needed and display the data as it is available. Parallel requests are sent to the server, and data is displayed as processing ends. Users don’t wait for all the requests to finish. We improved the system performance with this database overhaul.

### Salesforce to PrintVis integration

We migrated the operations from Salesforce ® to PrintVis ® — Microsoft Dynamics Nav ® — so an Extract, Transform, Load (ETL) tool needed to be developed to migrate and sync the data. After some research and investigation, we discovered there was no existing extractor between the two systems. We had to build a custom tool to accommodate the company’s data and the two systems' data structures. We used the APIs of Salesforce and PrintVis to get / insert the data from one system to another, but we had a laundry list of considerations, like data sync. We had to make sure we didn't duplicate the same data is in both systems. We had to do some experimentation to ensure the data structure was correctly achieved in both systems. The sync needed to be reasonably fast for our performance promises. With the large amount of the data to import we had to experiment and test hypotheses to optimize queries and also the read / write of the data.

### Ruby and Angular integration

We are transitioning our system from Ruby on Rails to a Node.js ® AngularJS ® microservices system. We had requirements to integrate Angular modules one by one once they are ready, due to the length of the project. A logged in user should be able to go from a Ruby page to an Angular page without even knowing that they're on another application. We had to experiment to achieve this transparency. A regular approach was to duplicate the login functions and call the login methods for both system once or call the login method every time the user would go from ruby to angular. However, this was not acceptable at both the user experience and system performance levels. The final version used some domain shared cookies between the two systems that held the state of the logged in user. When the user logs into Ruby, we simultaneously call the Node.js endpoint to log them in and save the token in a cookie. Afterwards, every time an Angular page is accessed, it would search for these cookies and authorize or deny the request.

### What scientific or technological advancements did you achieve or attempt to achieve as a result of the work described in line 244? (Maximum 350 words)

We were able to bring visual data to our application by experimenting with and, ultimately, creating an integration between our application and the new visual data application. We brought AI to our application by analyzing our telephone calls and training AI to identify missed opportunities. We wrote an ETL to merge with another data tool. We transferred our system from one code base (Ruby on Rails) to others (Node.js and AngularJS). We had to do this transparently for the user, meaning they moved in the application from pages generated by Ruby on Rails to Node.js and AngularJS. We did an entire system analysis to improve speed and performance and load capability.

.. References

.. https://www.tableau.com/

.. https://help.tableau.com/current/server/en-us/trusted\_auth.htm

.. https://patientnews.com/

.. https://rubyonrails.org/

.. https://www.voicebase.com/

.. https://nodejs.org/en/

.. https://angularjs.org/