QuickForms 3.0: A Business Intelligence Application Framework for Integrating Mobile Forms Data

Austin Chamney

Thesis

Submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

uOttawa

University of Ottawa

Abstract

Mobile apps are very popular and widely used to collect data. However, traditional web development tools like .Net and J2EE take a generic approach to such applications that leads to high code complexity and poor quality assurance. At the same time, Business Intelligence (BI) portals and dashboards provide interactive reports that allow one to flexibly explore, analyze, and visualize data. However, they are usually defined independently of the applications which are the source of the data. As a result, complex ETL (Extract, Transfer, and Load data) processes are required to integrate and transform data for reporting. We propose an application framework that flexibly integrates mobile apps and BI dashboards. We implemented three different versions of the framework and conducted case studies using two different health care applications to evaluate the framework. They demonstrated the potential to develop BI applications that integrate mobile forms data with less effort, less complexity, more consistency, and better ease of use. More importantly, the application framework provided the ability to seamlessly integrate data collection and reporting in a single mobile app. The final version of the application framework, QuickForms 3.0, is now published as an open source project, and a growing community of developers are now using the framework to build a wide variety of mobile apps.

Acknowledgements

First, I would like to thank my thesis supervisor Liam Peyton for his dedicated guidance over the years. Without Liam, none of my research or development would have been possible.

I would also like to thank my fellow students that I have worked with along the way. This includes Aladdin Baarah, Shirley Baffoe, Sean Dube, Shawn Dexter, Ashish Chattani, Natalia Castillo, Jaspreet Bindra, Sonia Peri, Dana Al-Rafi, Frank Sun, Ishan Mehta, Pramukh Hiriadka, Riyas Valiya, Isabella Vieira, Pilar Mata, Priyanka Jain, and Kavya Kumar. Each person introduced a unique perspective to the project, and provided valuable feedback through every step of the process. I also thank Dr. Gary Viner, M.D., Dr. Eric Wooltorton, M.D., Dr. Susan Humphrey, M.D., Dr. Doug Archibald and Alexandre Labelle for providing invaluable feedback throughout my research.

Finally, I would like to thank IBM, Bruyère Hospital, the Ottawa Hospital Family Health Team, and the Champlain Local Health Integration Network for their collaboration on the PAL-IS and RPP applications, as well as MITACS, IBM, AIME, Bruyère Hospital and NSERC for funding my research.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
AMD	Asynchronous Module Definition
BI	Business Intelligence
CSS	Cascading Style Sheets
DAO	Data Access Object
DBA	Database Administrator
Dojo	An open source modular JavaScript library
DOM	Document Object Model
ETL	Extract Transform Load
GWT	Google Web Toolkit
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IIS	Internet Information Services
J2EE	Java 2 Enterprise Edition application framework
JIT	Just in Time Medicine
jQuery	An open source JavaScript library
JSON	JavaScript Object Notation

KPI	Key Performance Indicator
LDAP	Lightweight Directory Access Protocol
LHIN	Local Health Integration Network
.NET	Microsoft application framework
OLEDB	Object Linking and Embedding Database
ORM	Object-Relational Mapping
PAL-IS	The Palliative Care Information System application
REST	Representational State Transfer
RPP	The Resident Practice Profile application
SAID	The Standards and Indicators Dashboard application
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UI	User Interface
XML	Extensible Markup Language

Chapter 1. Introduction

1.1. Problem Statement

There is an increasing tendency for organizations to collect data to measure and analyze performance and do business intelligence (BI), but creating a BI application is an arduous task (Mata, Kuziemsky, Singh, Baarah, & Peyton, 2014). Most BI applications use a slow process of data collection, data processing, and transformation from source systems into a specialized reporting database in batch processes that can take from hours to months to complete. This limits the ability to report on data collected while events are taking place, and it results in a disconnect between operation and reporting needs. Often data collected is missing, incomplete, mismatched or badly formatted which limits the ability to report.

Increasingly, mobile applications offer an alternative approach to collecting data and viewing reports to the traditional approach to BI applications. This is especially true, if one wants to improve the timeliness of the reports that are being viewed. Although they offer a timely capability for collecting data into a database, mobile applications require a considerable amount of customized coding to build and maintain. The database developed for these applications is often not optimized for reporting, which can limit the sophistication, effectiveness and completeness of the reporting that is possible.

Developing mobile BI applications is non-trivial as it requires a balance between simple, easy to use interfaces, and powerful BI reporting capabilities. Traditional web development tools like .Net and J2EE take a generic approach to such applications that

leads to high code complexity and poor quality assurance (Johnson, 2005) (Vawter & Roman, 2001). This is due to the large focus on application tier logic. Much of the developer's time is spent writing application logic, leaving the view tier and the model as secondary objectives.

As well, the database that results from such tools is not optimized to support powerful BI reporting. After the deployment of these J2EE and .Net applications, a secondary process is often conducted to transform the transactional database data into a data warehouse for reporting. This process can be as lengthy and difficult as creating the application itself (Simitsis, Vassiliadis, & Sellis, 2005).

1.2. Thesis Motivation and Contributions

The main motivation behind this study is to optimize forms and reporting support in BI Applications and to reduce the complexity and effort needed to develop BI applications in an application framework that links forms to reports. Our intent is to leverage AJAX for simple yet sophisticated and compelling mobile forms rather than traditional web applications. Leveraging AJAX within a BI application framework will introduce powerful reporting and data entry tools that are optimized for BI applications. Another intent is to leverage a star schema database model that is optimized for reporting. Leveraging a star schema database model within a BI application framework will make the linking of forms to reports largely a matter of configuration.

The main contributions of this thesis are:

1. A BI application framework for integrating mobile forms data that includes:
    a. A BI reporting database, which uses a standardized star schema with naming conventions to ensure optimal reporting performance and interoperability within the application framework.
    b. A Data Access Object (DAO) service, which provides a standardized interface to the BI reporting database for BI applications.
    c. A JavaScript client, which contains a library of generic modular JavaScript controls and application templates for common BI application features (including accessing the DAO service).
    d. A User Interface, which supports multi-form factors and name-based mapping between form data fields and the BI reporting database.
2. A published, referenced implementation of our BI application framework as the QuickForms 3.0 open source project, which has an active user community and online tutorials, template library and sample applications (Chamney, et al., 2014).
3. Two innovative, new BI applications for healthcare that were the focus of case studies to validate the application framework.

The design and implementation of the QuickForms 3.0 project were performed by me, however, several other Masters students worked on its development throughout the process. Similarly, for the two innovative case studies, I was the original and main developer while other students helped with bugs and quality assurance.

The following publications have been published related to this thesis:

A. Chamney, P. Mata, G. Viner, D. Archibald, L. Peyton, "Development of a Resident Practice Profile in a Business Intelligence Application Framework", The 4th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2014), Halifax, Canada, September, 2014.
*** **Best Paper Award** ***

A. Baarah, C. Kuziemsky, **A. Chamney**, J. Bindra, L. Peyton, "A Design Strategy for Health Information Systems to Address Care Process Management", 5th IEEE International Conference on Information and Communication Systems (ICICS '2014), Irbid, Jordan, 2014.

O. Badreddin, A. Baarah, **A. Chamney**, C. Kuziemsky, L. Peyton, "Analytics Driven Application Development for Healthcare Organizations", 7th International Conference on Health Informatics, HealthInf 2014, Eseo, France, March, 2014.

1.3. Thesis Methodology and Organization

The thesis methodology is based on design oriented research methodology (Hevner, March, Park, & Pam, 2004). This thesis is early stage research to establish the viability of the application framework approach for developing BI applications and addressing gaps in current practice. Multiple prototypes and two full implementations of our proposed application framework were developed based on a gap analysis of existing research. The steps followed in our methodology are outlined below:

1. Evaluation of the problem. Perform a literature survey of relevant approaches and related works. Determine the gap in existing application frameworks. Identify the evaluation criteria, specifying which areas of the gap are intended to be addressed.
2. Design prototype framework, leveraging existing technologies.
3. Implementation of a full design prototype, along with two case study implementations.
4. Evaluate the prototype, and determine remaining gaps, reviewing and updating the evaluation criteria.

5. Review and update the design of the prototype, extending it with specialized BI controls and templates and integrating components from different existing technologies into a new configuration.
6. Second full implementation of the design prototype, along with the original two case studies, as well as several new types of applications to validate generality.
7. Evaluate the design prototype based on the evaluation criteria and compare to the original design prototype as well as related works from the literature survey.
8. Publish results and identify future work.

This thesis is organized as follows: Chapter 1 gives a general overview of the scope of the research and defines the context. Chapter 2 identifies related works, and gives a general background in the technologies relevant to our research. Chapter 3 develops the problem description with a gap analysis and identifies and discusses evaluation criteria for closing the gap. Chapter 4 proposes the application framework. Chapter 5 describes case studies in which prototypes of our application framework were built and tested in the development of mobile apps for healthcare. Chapter 6 evaluates our application framework in comparison to related work based on the case studies using the evaluation criteria from Chapter 3. Chapter 7 concludes the research, and suggests future work.

Chapter 2. Background

In this chapter, we summarize the key concepts relevant to this thesis for BI applications and application frameworks. We also survey related works relevant to the thesis.

2.1. Business Intelligence Application

2.1.1 Business Intelligence

Business Intelligence (BI) is defined by Trkman as "the application of various advanced analytic techniques to data to answer questions or solve business problems" (Trkman, McCormack, Valadares de Oliveira, & Ladeira, 2010). These analytic techniques aggregate large amounts of data, and compress it into a consumable, visually appealing format that is optimal for decision making.

2.1.2 BI Application

BI applications have typically been thought of as reporting applications that enable business analysts to explore, analyze and visualize data for business process performance (Trkman, McCormack, Valadares de Oliveira, & Ladeira, 2010). More recently, mobile business intelligence applications that provide charts and graphics from IBM Cognos, Oracle Financials, MicroStrategy etc. directly on to smartphones and tables are becoming popular (Chouffani, 2012). There is also work on business activity monitoring (Baarah, 2014) that looks at how to collect data and generate reports while a process is taking place. In this thesis, we define a Business Intelligence (BI) application as a means to both capture data and report in order to monitor and measure performance.

A BI application usually interacts with the database through a multi-dimensional analytics model (Azarm, Peyton, & Nargesian, 2011). This contrasts to a traditional web application which interacts with the database through an object-oriented application model (Rumbaugh, 1991).

Managed process applications are similar but more complex than BI applications. Managed process applications collect data and report on business processes. They leverage business process models that capture roles, activities and relationships between them (Tegegne & Peyton, 2013). BI Applications focus on the real time collection of data from heterogeneous sources, and summarization of data in BI reporting portals (Peyton, Zhan, & Stepien, 2008).

2.1.3 Healthcare Mobile Applications

Increased capabilities of mobile devices have led to greater adoption in healthcare. Physicians have adapted to these mobile technologies, and incorporate them in their daily life (Kafeza, Chiu, Cheung, & Kafeza, 2004). Many web applications have taken advantage of these devices by adapting their web pages to new form factors (Ferenchick & Solomon, 2013). Applications can take advantage of this increased connectivity to collect data and send reports for doctors or nurses while they provide healthcare. For example, they can be used for real time monitoring of emergency room wait times (Baarah & Peyton, 2012).

Another type of healthcare application is a practice profile application. Physicians capture demographic, diagnosis, and self-evaluation data to report on their clinical experience. Charts and graphs summarize the types of patients physicians are seeing, and

the variety of diagnoses that have been recorded. This information can be used for performance management (Badreddin, Baarah, Chamney, Kuziemsky, & Peyton, 2014), or to identify gaps in knowledge and clinical experience. This is especially useful when training medical residents (Chamney, Mata, Viner, Archibald, & Peyton, 2014).

2.2. Application Framework

According to Schmidt, an application framework is "an integrated set of software artifacts (such as classes, objects and components) that collaborate to provide a reusable architecture for a family of related applications" (Schmidt, Gokhale, & Natarajan, 2004). An application framework is used to reduce the effort required to build certain types of applications and improve quality and consistency.

2.2.1 Architectural Patterns

Selecting the correct architecture for an application plays a significant role during the design process (Preuveneers, Yasar, & Berbers, 2008). The design of an application is usually structured around specific architectural patterns that decompose the architecture of the application across different application tiers or abstraction layers (Stal, 2006). An architectural pattern is defined as follows: "A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution" (Buschmann, Henney, & Schimdt, 2007). These generic solution schemes define specific components, which have a particular pattern of interactions.

There are a number of application frameworks for web applications and mobile web applications that define specific architectures that address the collection of data and

reporting including J2EE and .NET. These along with the traditional framework used for BI applications are described in more detail in section 2.3

2.2.2 Web Application Architecture

Web applications are developed following a three-tier web application architecture (Gellersen & Gaedke, 1999). The tiers are designed to delegate the application tasks to optimized application components. A common web application architecture is Model View Controller (MVC) (Leff & Rayfield, 2001). This is designed to separate the look and feel of the application from the underlying data. In a web application, the controller and the view are often coupled on the client to capture clicks and keystrokes, while the server is tasked with maintaining the model of the application.

2.2.3 Data Access Object

The Data Access Object (DAO) is a means to encapsulate access to a data source. It provides an interface to the low level sockets that are often involved with the implementation of the data source. (Alur, Malks, Crupi, Booch, & Fowler, 2003). An example of the interactions with the data source and the DAO can be seen in Figure 1.

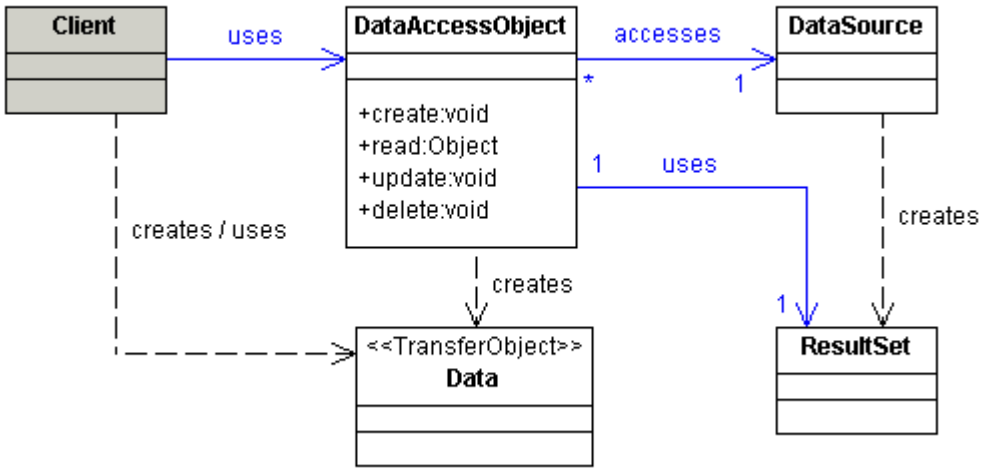


Figure 1: Data Access Object Class Diagram (Alur, Malks, Crupi, Booch, & Fowler, 2003).

The high level abstraction of the data source connection enables changes of database technology without changing the underlying interface with the client system. (Alur, Malks, Crupi, Booch, & Fowler, 2003).

2.2.4 Data Warehouse Architecture

A data warehouse is defined as "a subject-oriented, integrated, time varying, non-volatile collection of data that is used primarily in organizational decision making" (Chaudhuri & Dayal, 1997). It contains a superset of data, aggregated from its satellite production databases. The requirements for a data warehouse are different than an operational database such that it is required to have high performance, and high efficiency (Kimball & Ross, 2013). They are optimized for reporting rather than transactional efficiency.

One such architectural pattern for data warehouses is the Extract Transfer Load (ETL) pattern. ETL performs data integration from multiple sources into one unifying

data warehouse (De Giusti, Oviedo, & Lira, 2011). This pattern is widely used to create a data warehouse that can be used to provide BI applications with relevant lookup data and for the persistence of facts.

2.2.5 Multi-dimensional Analytics Model

A filter or subset of a data warehouse called a data mart (Kimball R. , Ross, Thornthwaite, Mundy, & Becker, 2008) is typically defined through a multi-dimensional analytics model to support analytics and reporting for a particular application. They are defined on one table of "facts" and related tables which define the possible values for fact attributes along different dimensions (date, demographics, etc.) for a particular problem.

2.2.6 Star Schema

A star schema is a dimensional model that is implemented in a relational database (Kimball R. , Ross, Thornthwaite, Mundy, & Becker, 2008). In this model, transactional data is stored in fact tables, while lookup data is stored in several joined dimensional tables. This model allows for hierarchies to be defined in the dimensional tables to support powerful reporting techniques (Kimball R. , Ross, Thornthwaite, Mundy, & Becker, 2008). These hierarchies are highly de-normalized to optimize query performance.

2.2.7 Web Service

A web service is defined as "a software system designed to support interoperable, machine-to-machine interaction over a network" (Booth, 2004). Web services enable

HTTP and SOAP connections between interoperable systems. For SOAP, a pre-defined interface must be created called a Web Service Description Language (WSDL). A WSDL is an XML based language that describes the communication protocol between two interacting components (Curbera, Duftler, Khalaf, Nagy, Mukhi, & Weerawarana, 2002). The REST architectural style is used to transfer HTTP messages between two interacting components (Fielding & Taylor, 2002). This contrasts to the SOAP protocol based on its simplicity and generality. REST interfaces do not require a rigid data definition to communicate. A web service can also act as a tool to publish access to a data warehouse or data mart.

2.3. Related Works

The following sections discuss related approaches to creating an application framework for BI applications.

2.3.1 Traditional Business Intelligence

Traditional BI summarizes operational systems data to provide data access and information about an enterprise (Liya, Barash, & Bartolini, 2007). This is traditionally accomplished with a back end approach, which focuses on data transformation into a data warehouse. This information is then published to a distributed reporting portal. Figure 2 diagrams the traditional approach in terms of different layers or tiers of data collection and processing in order to deliver reports to end users. It starts with data collected in operational data sources and is followed by a data transformation into a data warehouse using an ETL layer. A business view is created on this data which defines hierarchies, and metrics. Finally, the data is published to a front end analytic application.

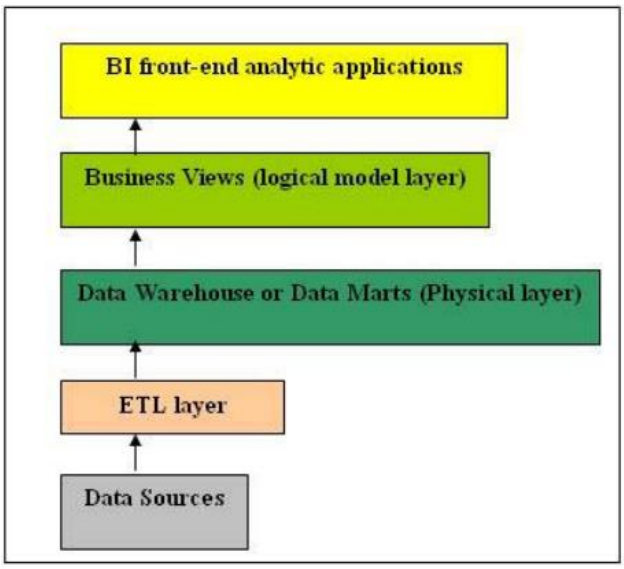


Figure 2: Example of tiers required in a traditional BI system (Liya, Barash, & Bartolini, 2007)

This structure allows operational data from across the enterprise to be transformed and integrated into a single data warehouse. The BI front end analytic applications then provide reports from this data analysis.

2.3.2 Web Application Frameworks

A more recent approach to business intelligence is to build web applications that collect the data directly for reporting. The development of these applications is non-trivial. Web application frameworks can be used to mitigate the effort required to build a web based BI application. Java 2 Enterprise Edition (J2EE) is an application framework built on the java runtime environment (Johnson, 2005) that can be used to build such applications. It defines a generic architecture for such applications in terms of client, web, business and enterprise information tiers.

Several third party libraries have been designed to interact with the core J2EE package to simplify and expand key components. One such library is Hibernate, an

Object-Relational Mapping (ORM) tool. Hibernate will manage the transformation from the business tier to the enterprise information tier automatically. (Singh, Stearns, Johnson, & Team, 2002). This is usually combined with Enterprise Java Beans (EJB), which map the elements in the web tier to business objects in the business tier.

Struts is an application framework built on J2EE to support web application development based on the MVC pattern (Struts, 2014). Struts achieves this goal by providing templates that the developer can use to build user interfaces for the client tier. These templates interact with the web and business tiers, in a well-defined manner which maintains the ideals of the MVC pattern (Leff & Rayfield, 2001).

The Microsoft .Net framework is another web application framework that leverages the MVC pattern to develop web applications (Microsoft, 2014). Both J2EE and .Net provide developers with a data access tier, a user interface tier, and an application logic tier.

2.3.3 AJAX Client Framework

In order to provide a rich and responsive web interface, client side JavaScript tools are often needed. Many frameworks are available that provide developers with extensive toolkits that mitigate the work required to provide such interfaces. An example of such AJAX client frameworks are jQuery and Dojo (Stack Overflow, 2013). Both of these frameworks contribute a series widgets and libraries that are designed to function across a multitude of browsers on a multitude of devices and form factors (jQuery Forum, 2011) (Dojo, 2014). These JavaScript frameworks encourage the use of web services to

dynamically populate the view with server side data. Web services, combined with the JavaScript toolkits will provide a unifying user interface experience.

Figure 3 below depicts the search volume of JavaScript frameworks over the last ten years. It is clear that initially Dojo, and later on jQuery are the popular choice for many developers. The other frameworks with smaller search volumes show that there is less interest, and therefore are less sustainable in the future.

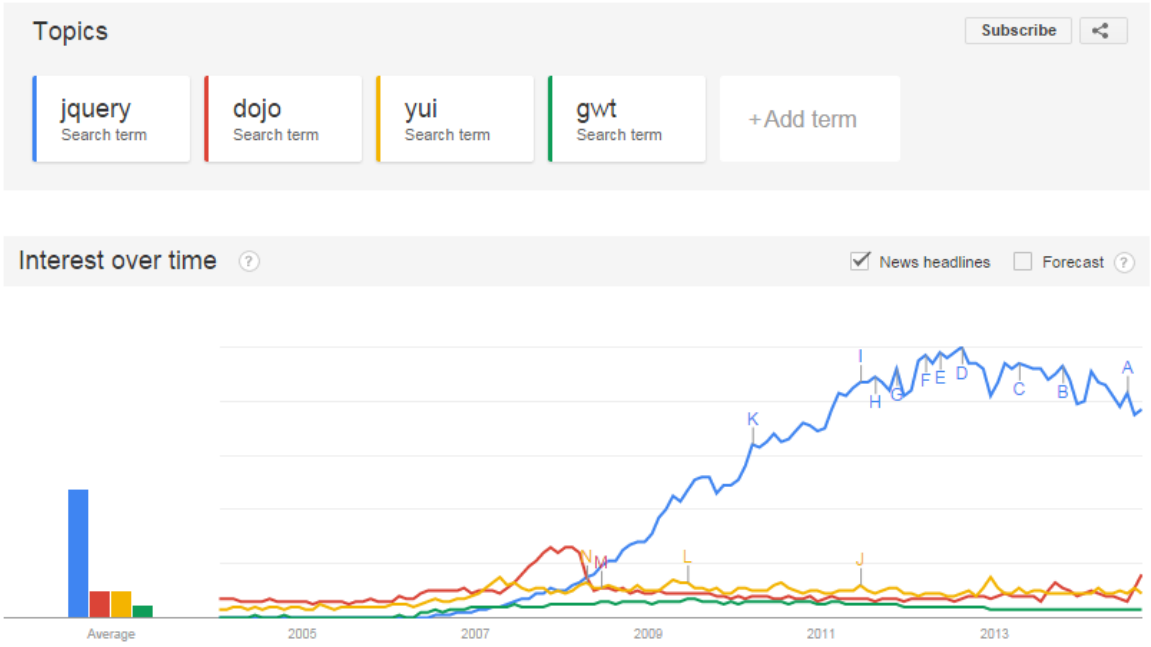


Figure 3: Search volume of JavaScript frameworks over time (Google Inc., 2014).

The JavaScript frameworks that reached the requirements for a scalable BI application framework were found to be jQuery and Dojo. Both frameworks provide a robust series of widgets to enrich the user experience, and have an active development community. These frameworks are compared below using a sample case study application seen in Figure 4 and Figure 5.

Dojo

Figure 4: Dojo Mobile test application html page

The primary focus of Dojo Mobile is presenting data in a unique and meaningful way. The Dojo Mobile user experience seems to focus on list views and popular mobile widgets like maps and graphs. A major requirement for a BI application framework is a robust forms framework that has many widgets for data entry. Since Dojo Mobile focuses on data presentation, the data entry user experience is found to be lacking. The input widget toolkit does not include vital elements such as the select user input. The text input widget by default does not scale with the size of the screen, and maintains the default height of the standard HTML text input. The dexterity of user input on a mobile device is considerably lower, so the size of the widgets poses a problem. The Relative Average Frequency (Jokela, 2006) of the user will decrease due to the inability to correctly use the

widgets. Relative Average Frequency describes the user's ability to perform regular tasks with the mobile device quickly after the initial learning curve is overcome.

JQuery

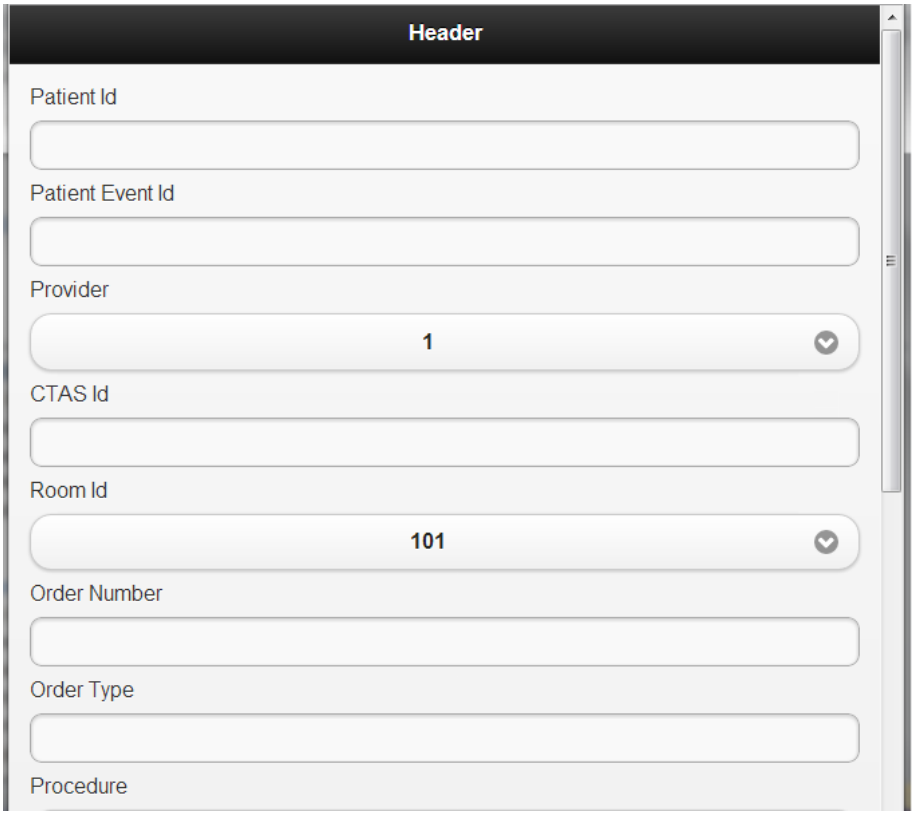


Figure 5: JQuery Mobile test application html page

JQuery Mobile includes a variety of form based widgets by default, including text box and area inputs, select and multi-select, and several versions of checkbox and radio buttons. Many widgets, such as date inputs and auto complete do not come by default with the jQuery Mobile package, but there are many third party jQuery libraries that supplement this problem. JQuery UI contains a rich package of add-ins for jQuery Mobile for the missing forms requirements by default. This extra library management

however poses a problem from a library management standpoint, and makes upgrading to newer versions of jQuery difficult.

2.3.4 Forms Application Framework

A forms application framework is used to define an application in terms of the sequence of forms needed to complete a business process (Chusho & Fujiwara, 1998) .

ProntoForms is a forms application framework (Chéné, Peyton, & McGuire, 2010) which is geared towards businesses that wish to move their paper-forms defined processes onto an online mobile platform (ProntoForms Corporation, 2014). It provides a cloud computing platform for businesses to define forms and track their use. These forms are available to team members from the same project on their mobile devices. Different data types such as signatures, audio recordings, and text entries are uploaded to ProntoForms. This data is immediately available for reporting and decision making. ProntoForms focuses on enacting a business process rather than monitoring the business process. It is possible to extend ProntoForms' capabilities for reporting and analytics, but that has not been a focus of the technology so far.

2.3.5 Practice Profile Applications

A practice profile application is a particular type of BI application that is used in healthcare. It is a summary of patient encounters and experiences in an online portal (Iglar, Polsky, & Glazier, 2011). Its purpose is to identify trends or deficiencies in the practice of a particular health care professional, often as an educational tool. Trends or deficiencies are visualized through performance charts made available in practice profile reporting portals based on data collected.

Physicians from Michigan State University wished to assess medical students based on their core competencies during their patient visits. Ferenchick and Solomon (Ferenchick & Solomon, 2013) developed a mobile data entry application in Microsoft .Net and jQuery Mobile called Just in Time (JIT) medicine. It requests medical students to record the details of patient visits during their practice. The medical student assessment information becomes available to the physicians on BI dashboards where they assess the student on their performance. JIT is an example of a BI Application. It collects form data directly into a star schema reporting database. This information becomes available in real time to the reporting dashboard for administrators.

LogMD (LogMD, 2014) was developed in conjunction with the Association of Canadian University Departments of Anesthesia to enable physicians to track their experience against international benchmarks. It uses a tree structure to format different procedures. The physicians expand the tree structures to record the desired procedure for each patient.

Both of these practice profile applications are excellent examples of BI applications. They provide the user with powerful business intelligence tools using many of the technologies as stated in section 2.2. These applications suffer however due to a disconnect between capturing data and reporting as discussed in the following chapter.

Chapter 3. Problem Statement

3.1. Problem Description

Business Intelligence applications take data from business operations and create reports that can be used to analyze and monitor those business operations. The traditional approach to provide this functionality is to transform the operational system data using ETL processes into a data warehouse. The transformation of the data can be difficult and time consuming due to inconsistencies between different sources and incompatibilities with what is needed for reporting. Operational systems are highly optimized to record data that is useful to the domain specialist, but often lack attributes that are required for reporting. This data is often stored in transactional databases, whose structure does not transform properly to a data warehouse schema. Finally, due to differences between operational systems, data transformations between the operational database and the data warehouse take time. This results in a large lag time between daily recording of data, and meaningful reports on this data. This severely limits the ability to make agile decisions to solve immediate problems. Figure 6 below shows an overview of the problem, and portrays the complexity of the system.

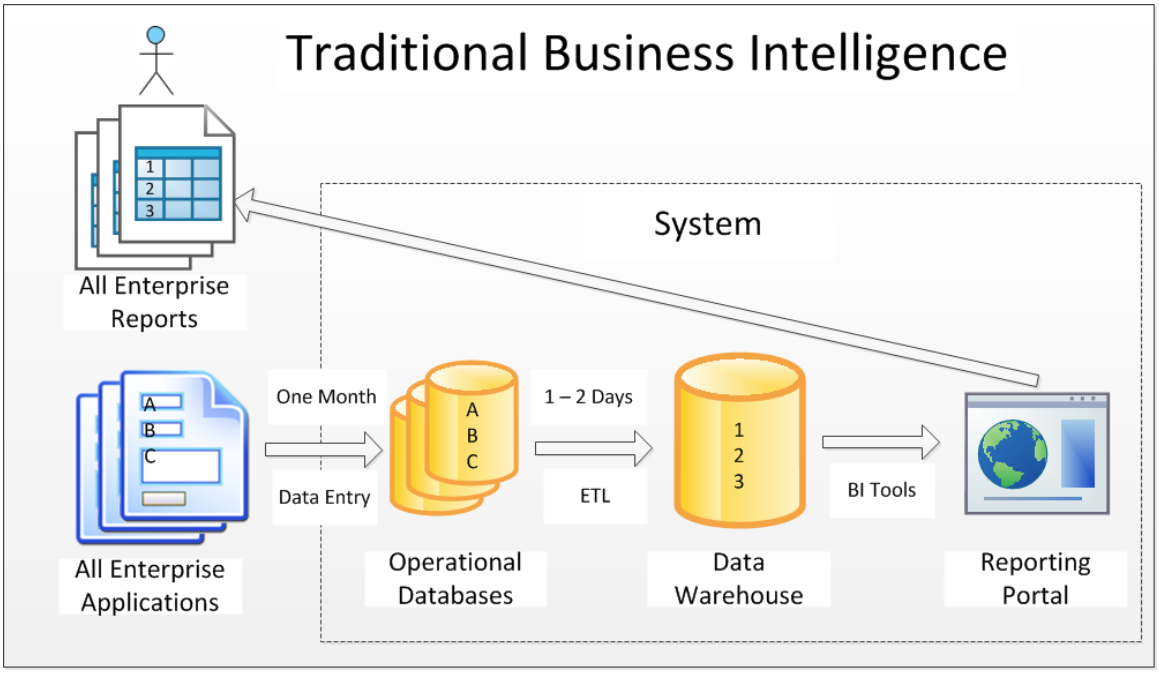


Figure 6: Workflow required between users entering forms and producing reports in a traditional operational system

The diagram above shows the flow of data entered in a traditional business intelligence system. First, the domain specialist enters data using operational forms. This data is entered into operational databases in the raw format captured by the operational forms. This data must be converted into a star schema format that is optimized for reporting. ETL is performed to stage the operational data into a data warehouse. BI tools are used to define hierarchies and metrics, and finally, the data is published to a reporting portal. The data moves between four systems before it is made available on the reporting portal.

Ideally, users would enter BI data on easily accessible online forms that link directly to warehouse BI reporting database. Each data field in the form would be carefully chosen to provide effective reports, allowing for fields that would not normally be stored in operational data. After data entry, reports would ideally become available

instantly, rather than waiting for a lengthy data processing and ETL operation. Figure 7 below shows an overview of this concept.

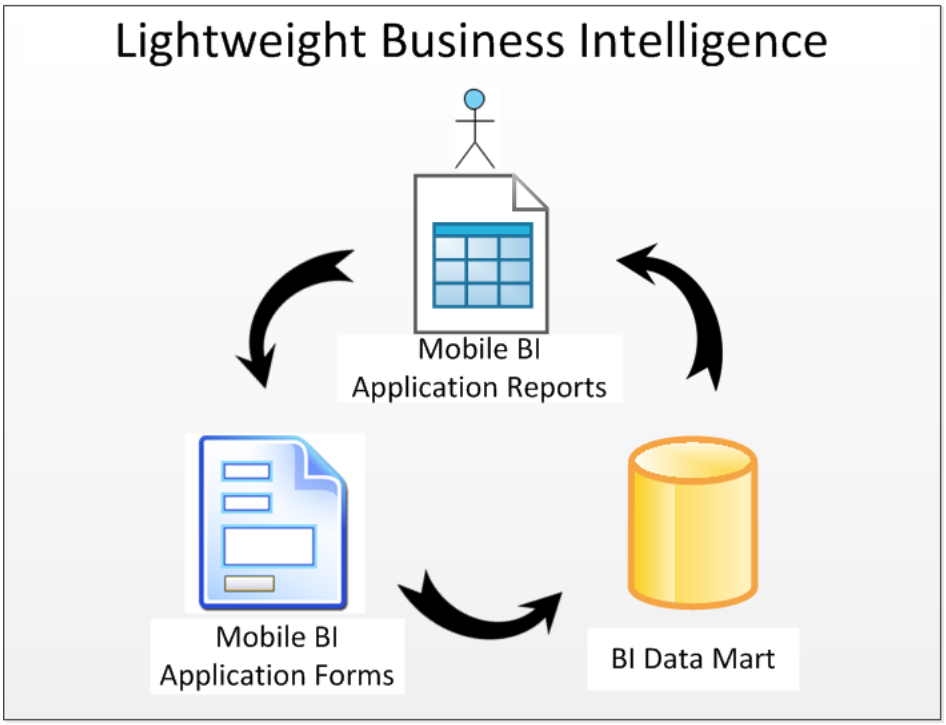


Figure 7: The usability and the practicality of lightweight data collection for reporting only.

This idea involves capturing data from the user, but specifically only the field which are important metrics for reporting. This removes the need for expensive and complicated data integration. It also removes the lag time between the steps of data entry, ETL, data staging, and report generation. Lightweight BI transforms this workflow by allowing multiple users from different organizations to use the same data mart, requiring no data integration. We avoid unnecessary data integration that is caused by storing data in inappropriately formatted databases.

3.2. Gap Analysis

Here we identify the gaps in existing approaches to BI. The approach is explained, and we identify how it falls short from providing the optimal BI solution.

3.2.1 Traditional BI

Traditional BI systems provide an adequate solution for analyzing long term trends after a business process is completed. Users enter data using tools provided by their traditional environment. This data is converted later using ETL into a format which is digested by a data warehouse. Online BI portals are then developed to display the transformed data in reports that the user, or other team members can view for interactive feedback on their operational tasks.

Traditional BI falls short however in terms of the time between form submission and retrieving valuable reports. The entered data is interpreted, moved, transformed, and compiled in several steps in this system using complex ETL. This process can take weeks, which may impact the report's usefulness by the time it reaches the end user. Additionally, every time the user's data moves from its original state, it may be erroneously transformed. These reports are only available in reporting portals which limits the mobility and accessibility of the reports.

3.2.2 Web Application Framework

A web application framework could be used to build mobile apps that address some of the above issues. Web applications increase the mobility and accessibility of the forms and reports, and reduce the wait time between entering form data, and generating reports. Application frameworks like J2EE and .Net allow developers to make

customized forms and reporting tools for the end user, and provide ORMs and other database tools to mitigate the overhead when developing the transactional database.

Traditional application frameworks allow for ultimate flexibility at the cost of complexity. A J2EE or .Net application is capable of having entirely customized application behaviour, but many of the components would need third party integration or to be created from scratch. The learning curves of these frameworks are steep, and code maintainability for large applications is difficult. In the case of a BI application, a diverse set of tools are not required. A BI application developer does not need the full generality these frameworks provide. At the same time, the BI application developer needs to convert the data into multi-dimensional analytical models that are not compatible with ORMs. Figure 8 below explains the tiers of general-purpose web applications, and how the complexity increases as the application requirements grow.

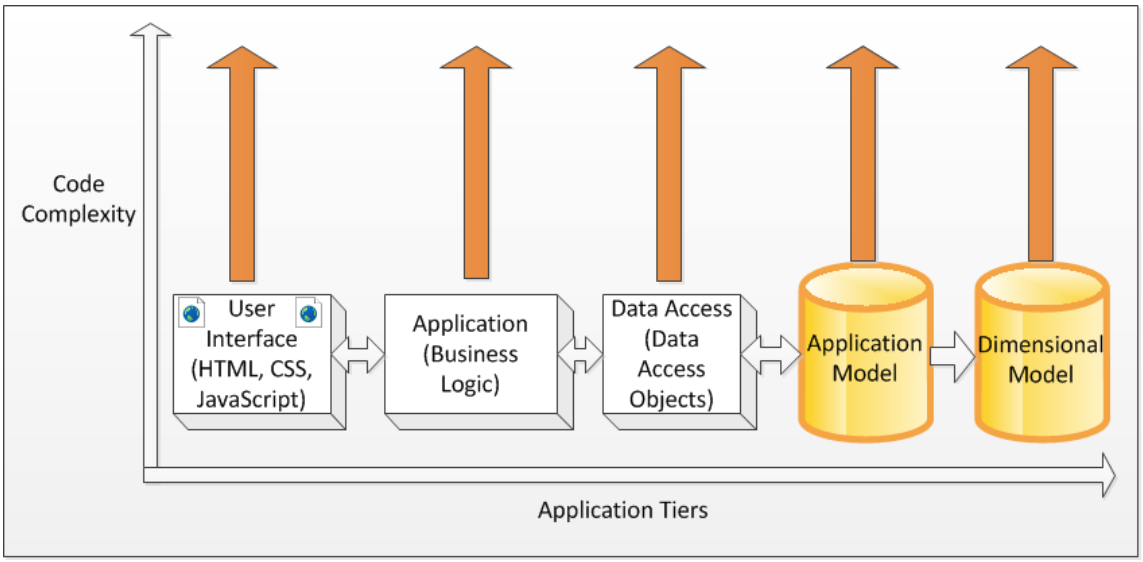


Figure 8: Web application framework architecture by lines of code for equivalent complexity

The arrow length for each of these tiers shows the complexity and effort required to develop the subsystem. Each tier in a traditional J2EE or .Net application has high

complexity, and requires much developmental effort. The user interface must contain a customized look and feel, along with a link to the application logic via server side coding (JSP or ASP pages). The application tier must have customized web services, transformation code between the view and the DAO, and any server side plugins. The data access tier must contain all of the customized queries to access the database. The application model often requires a specialized database architect to handle the transactional schema. Finally, the dimensional model must be constructed using ETL tools. Each of these tiers increases their complexity quickly in relation to application size.

Both of these frameworks have powerful tools to run transactional database queries, but lack such powerful tools on the client side. HTML pages are created dynamically for the user, which allows for simple dynamic web content, but at a cost. Powerful form tools like date pickers and auto-complete entries are not provided with either framework. The developer must then integrate with other frameworks to provide the user with rich front-end controls. Client side control integration is messy, and can conflict with the dynamic HTML creation language provided by the framework.

Even with the powerful transactional database tools like ORMs and Hibernate, more effort is needed to once again transform into a data mart to allow BI tools the ability to generate reports. This transformation once again involves complex ETL processes that take extra time to develop, and may not provide instantaneous feedback for the user.

3.2.3 AJAX Client Framework

AJAX frameworks like Dojo and jQuery address the client-side limitations of web applications. Both frameworks provide a powerful set of client side tools that are capable

of producing rich dynamic content for the user (Guerrero, 2013). Both frameworks support extensions, which have allowed the development community to provide a series of freeware tools that would otherwise have to be created from scratch.

However, these frameworks suffer from the same data access problems as their server side counterpart as well as the same overall application complexity. They must support enough tools to cover all possible web applications, which increases their complexity. As well, jQuery suffers from code maintainability. Developers often create one JavaScript file per page, and append to that file for every new piece of functionality required. This file grows linearly with the complexity of the web application. At a certain point, it becomes very difficult to maintain.

Dojo counters jQuery's complexity issue with JavaScript modules. All of the Dojo library has been extracted horizontally into separate files, and can be loaded on demand when needed. This is an improvement over jQuery due to smaller file size, and the ability to load modules only when needed. A problem is noticed however when a single page requires a significant number of Dojo tools. The list of tools must be maintained, and kept in a specific order. Often, so many modules are loaded in one page of the application that developers can no longer maintain the list of modules.

3.2.4 Summary

Below, Table 1 summarizes the existing approaches to BI applications, and briefly describes their limitations. These technologies will be compared in Chapter 6.

Table 1: Summary of existing BI technologies and their limitations

Approach to a BI Application	Limitations
Traditional BI	- Long wait between form entry and reports. - Data is susceptible to human error. - Difficult to record supplementary data points not found in the day to day operational forms. - Requires ETL processes causing a delay between data entry and report availability.
Web Application Framework	- Requires customized tools and third party integration dependency management. - Requires ETL processes causing a delay between data entry and report availability. - Requires a heavy customized application logic component. - User interface does not map directly to the generated reports. - No built in mobile user interface support.
AJAX Client Framework	- Requires a heavy customized application logic component. - Unknown web service integration component. - Tradeoff between a complex dependency management module and poor scalability.

3.3. Evaluation Criteria

A list of 13 different criteria has been created to evaluate application frameworks in the domain of BI applications. The criteria are grouped into four categories:

- The development effort required to produce a BI application.
- The usability of the tools provided by the application framework.
- The complexity of its development.
- Support for specific BI application features.

Each criterion was chosen based on either support for the criterion from our literature review; support for the criterion as a requirement from either the software engineering expert or healthcare domain experts we interacted with for the applications we built in our case studies; or were learned during the gap analysis from one iteration to

the next as we refined our prototypes for our case studies.. The primary healthcare domain experts were Prof. Craig Kuziemsky from the Local Health Integration Network committee on standards and indicators for palliative care, and Dr. Gary Viner and Dr. Eric Wooltorton from the Ottawa Hospital Family Health team and the University of Ottawa Medical School. The main software engineer expert used with this research was Dr. Liam Peyton who has 23 years' experience building business intelligence applications and is co-author on a patent related to mobile forms applications.

Complexity is a common measurement for evaluating the quality of software as mentioned by (Weyuker E. , 1988). It is a measure of internal interactions between components, and how the number of interactions scales with the difficulty of the problem.

3.3.1 Complexity of Database

This criterion as mentioned by (Papadimitriou & Yannakakis, 1997) measures the difficulty to build and maintain the application database. This can range from simple: templated schema with minor configuration; to moderate: ORM database coupled with a customized data mart; to complex: fully customized snowflake model transactional database coupled with a full scale report optimized data warehouse.

3.3.2 Complexity of Database Transformation

This criterion continues upon the theory described by (Papadimitriou & Yannakakis, 1997), but specifically in reference to the complexity of the transformation between the transactional database and the data warehouse. This includes the complexity

of the queries required to transform the data, and the complexity of any data scrubbing required to maintain data consistency.

3.3.3 Complexity of application tier code

This criterion is based on the number of SQL queries, web service endpoints, and lines of application tier code required in the BI application for identical functionality. As (Tegarden, Sheetz, & Mona, 1995) states, object oriented application tier code controls the complexity of the system. Complexity is considered low if the application tier requires little to no customized coding. It is considered high if the application tier requires significant development of web services, business logic, or the integration of third party tools like LDAP or Hibernate.

Usability is defined by (Squires & Preece, 1999) as the ease of use of software. Software usability is traditionally ranked by measuring the user's response to the software's efficiency, effectiveness, helpfulness, control, and learnability (Kirakowski & Corbett, 1993).

3.3.4 User Interface

This criterion was determined through case study analysis. Several Think Aloud sessions were conducted with the domain experts to identify and qualify attributes of a good user interface (Fonteyn, Kuipers, & Grobe, 1993). Users of the application only found it effective if the user interface was intuitive, and easy to use. This criterion was measured using mobile devices based on the length of time required to enter data into a form.

3.3.5 Real time reporting

The health care expert Craig Kuziemsky established this criterion. It was determined that the software's usability was dependent on the availability of the report while the process was taking place.

Development Effort

Development effort measures the type of skills needed to create a BI application, and the cost and availability of developers with these skills. It also measures the time and cost to develop and maintain customized application specific code.

3.3.6 Technology used to create the BI application

This criterion measures how the technology or application framework assembles the BI application, and what tools were used in the process. This can range from a drag and drop interface, to a model defined interface, templated pages, to hard coded interfaces. This criterion was determined by the software engineering expert.

3.3.7 Effort and skill required to build a simple BI application

This criterion measures the effort required by the application developer to build the BI application. This is a core measure mentioned by (Weyuker & Avritzer, 2002) for the scalability of an application framework.

3.3.8 Effort and skill required to edit and update

This criterion expands upon the effort and skill required to build a BI application. It is often difficult to revisit code developed in the past and remember its intended functionality (Weyuker & Avritzer, 2002). This criterion measures how readable and

editable the application code is after the bulk of application development is finished. This criterion was established by the software engineering expert.

3.3.9 Effort and skill required to create team members

This criterion was determined through our multiple case studies. In every iteration, it was found that the application called for a team member portal. This included authentication, authorization, creating and editing usernames and passwords, and general demographic information about the users of the application. This criterion measures the effort and skill required to implement this functionality in the application.

3.3.10 Effort and skill required to create a useful report

This criterion is a subset of the effort and skill required to create a BI application, but it is important to emphasize. All of the value of a BI application resides in its ability to generate useful reports, graphs and metrics. If a technology struggles to provide its users with usable reports, it fails as a BI application framework. This criterion was created by the domain specialist Gary Viner.

BI Application Features:

These criteria were determined by the software engineer experts and domain experts. They define certain functionality that is traditionally implemented in a BI application. The criteria are measured on the basis of whether or the technology supports the functionality.

3.3.11 Reports linked to form dropdowns

This criterion was created by the software engineer expert. It describes how each drop down list in the form should display the same data that is displayed in the charts and graphs. This is important due to data consistency. If a label is changed in the drop down list in the form, then the corresponding reports should automatically change as well.

3.3.12 Application configurability by administrators

This criterion was created by both the software engineer expert and the domain experts. Application configuration options such as drop down labels in the forms and drop down list priority traditionally changes quite often. Administrators of the application cannot rely on developers to make these small changes in a timely manner. Other changes such as the look and feel of the application should not require more than a single configuration change. The functionality to make these changes must be easily available to the administrator of the application in order to maintain it. It should not require developers to code.

3.3.13 Device and form factor support

This criterion was determined by the domain experts. Traditionally, BI applications are used on mobile devices that require a specialized interface to use effectively. The technology used to provide the BI application must support all form factors to allow users to use any device of their choice.

Chapter 4. BI Application Framework for Integrating Mobile Forms Data

In this chapter, we propose a BI application framework for integrating mobile forms data.

4.1. Application Framework

The proposed BI application framework is designed to simplify the development, deployment, and user experience of a BI application. It seamlessly merges data collection and reporting in a single application.

The application framework is comprised of four tiers: BI Reporting Database, Data Access Object Service, JavaScript Client, and Application Interface. These tiers interact to provide the functionality needed for a BI application. More importantly, they allow us to simply and organize a BI application so that development and maintenance of a BI application is largely a process of template selection followed by component assembly and configuration. The interaction between tiers is depicted in Figure 9 below.

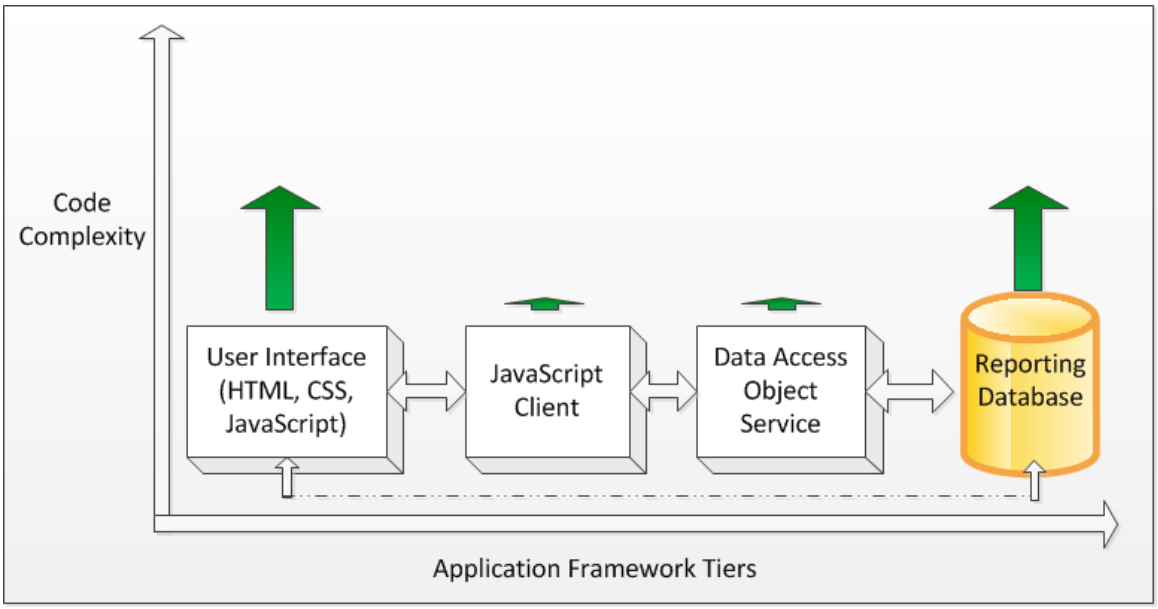


Figure 9: BI application framework architecture by lines of code for equivalent complexity

The User Interface tier contains HTML, CSS, and JavaScript that configure and sculpt the particular look and feel of the BI application as it supports forms for data collection and reports. Naming conventions connect the HTML form and its fields to the form fact table and dimensional lookup tables for form fields in the database. There is a minimal amount of JavaScript required to load the JavaScript Client library, and assemble and configure the required controls for the page. Customizable CSS is loaded to optimize the layout of the application for all form factors and devices.

The JavaScript Client tier contains a set of JavaScript controls that provide advanced BI controls for forms and reports linked to the reporting database. The JavaScript Client also provides a set of API calls for interacting with the Database Access Objection service. For forms, the API calls link form elements to the database, and pulls data from lookup tables to populate the dropdowns for form fields. For reports, the API

can execute application specific queries. This tier also manages authentication and authorization.

The Data Access Object (DAO) service tier handles the form and report-related requests with a series of parameterized SQL queries that provide access to the reporting database. It is completely pre-packaged and requires no configuration or custom coding except to specify the database connection information to be used for a BI application.

Finally, the reporting database is a multi-dimensional star schema optimized for reporting and compatible with standard third party reporting tools (like IBM Cognos) (Gangadharan & Sundaravalli, 2004). There is a strict naming convention for columns, tables, lookups and facts so that the DAO Service tier can interface to it properly. Templates are provided for each type of table in this schema model. That includes facts, dimensions, bridge tables, authentication tables, and multi-select tables.

In the following sections, we describe in detail each tier of the proposed BI application framework model.

4.2. BI Reporting Database

The BI reporting database has a standardized systematic star schema database with structured naming conventions and table relationships. It is designed to optimize reporting functionality and flexibility and be compatible with enterprise BI tools (like Cognos). As events occur, data is collected from forms and stored in a fact table with links dimensional look up tables that contain the possible values for fields on the form. An example of a BI reporting database schema can be seen in Figure 10.

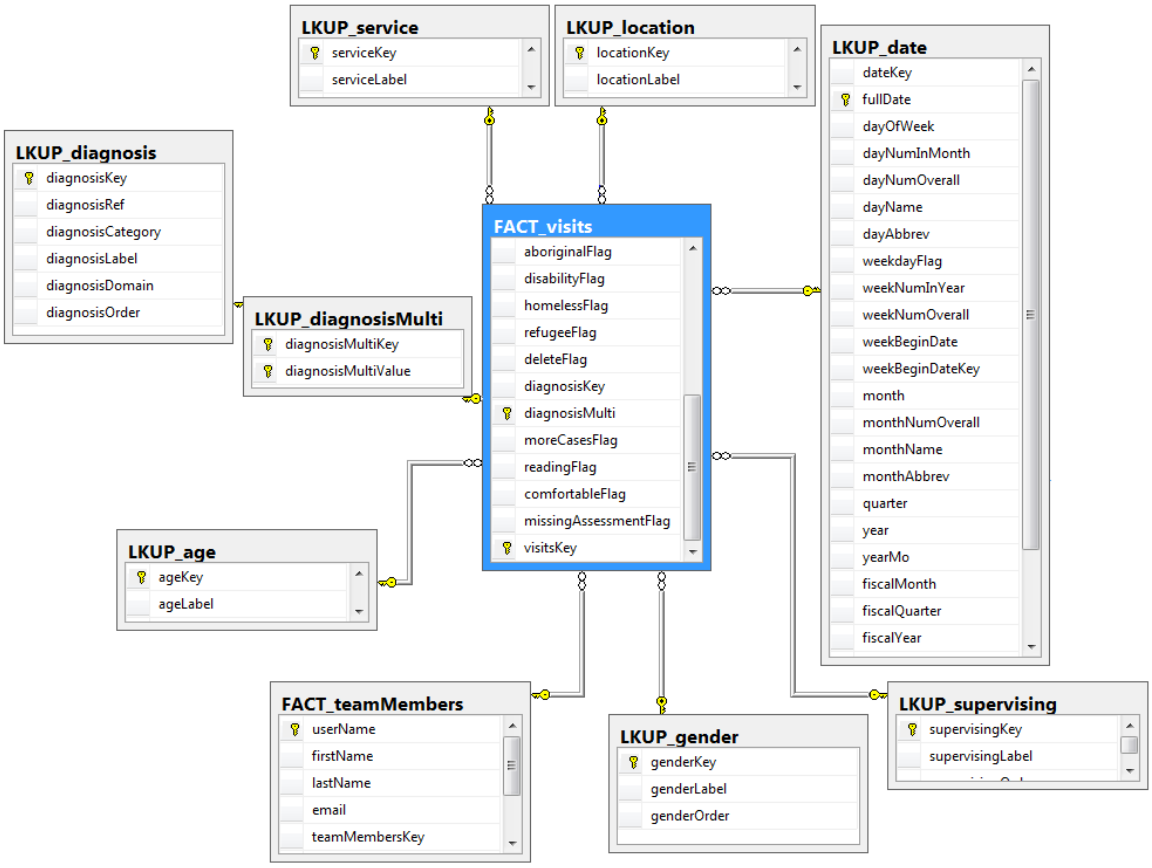


Figure 10: Reporting database tier rigid database schema with several lookups and one many to many bridge table.

The fact table seen in Figure 10 corresponds to a single form in a sample BI application. The fact table contains a key for unique identification, the username of the user who entered the data, and the other metrics and attributes that are needed for reporting. Other metadata such as recorded date, updated date, and a delete flag are also included. Columns that are used for measures (such as length, or any numeric attribute) are entered "as is" into the fact table. Columns that contain dimensional data, such as date, use a key which links to the corresponding dimensional lookup table. Many-to-many relationships (for multi-select columns) are represented with an automatically generated incremental lookup key into a bridge table (e.g. LKUP_diagnosisMulti in Figure 10).

Dimensional tables (or lookup tables) contain a key for unique identification, a label to describe the lookup value, and a priority column to determine the order in which the entries appear. Other dimensional attributes can also be added to categorize the data for "slice and dice" reporting. For example, the LKUP_date table in Figure 10 has multiple dimensional attributes such as month, quarter, and year that categorize the entries in the table. When generating a report off of the fact table, a join to these tables allow for powerful filtration and aggregation based on these dimensional attributes.

Bridge tables, such as LKUP_diagnosisMulti in Figure 10, use a paired key to uniquely identify a row. One of the keys references into the fact table, and the other into the lookup table. This table is primarily used to allow fact tables to select multiple lookup values from one lookup table. Each unique selection is represented in this table as a row.

Each of these table types have several requirements, naming conventions, and recommended attributes to accurately represent the data in the star schema pattern described above. The naming conventions are enforced to ensure the DAO is able to call upon tables, make joins to lookup and bridge tables, and for parameterized queries.

4.2.1 Fact Table Naming Requirements

The fact table is designed to capture all of the transactional data that is entered from the view of the BI application. The name of the fact table must start with "fact_" to increase the readability of the schema. The DAO automatically appends this prefix to all references to the fact table. An example fact name would be "fact_sales". There must be a key column named "<factname>Key" where <factname> is the name of the fact without

the prefix. This column is automatically incremented every time a row is inserted to ensure unique identification.

There are three optional columns to collect metadata about the fact insertions. The column “addedBy” is a numeric reference to the current user of the application who submitted the fact. “createdDate” is a date which references to the server time when a new fact is inserted. Finally, “updatedDate” is a date which references to the server time for every subsequent update to the row in the fact table. When a new fact row is inserted, the “updatedDate” defaults to the “createdDate”.

Fact tables link to dimensional lookup tables that provide more detailed information about fact attributes. A column in the fact table is used to index into these lookup tables. The proposed BI reporting database schema requires that the lookup index column has a name that is identical to the lookup table name. For example, if a “date” lookup is required, the fact table column must be named “date” and the lookup table must be named “date”.

When a form requires multiple selections from a single lookup table, a bridge table is required. In this case, the fact table provides a column that indexes into the bridge table instead of the lookup table. Indexes into bridge tables have the same rules as lookup tables; however, there must also be a “Multi” postfix on both the column name and the bridge table name.

If there is a field that is used for measures in the enterprise reporting suite, but also has an associated lookup table, two columns are required. The first column is the

standard lookup index column with the regular conventions. The second column has the same name as the first, with the extra “Measure” postfix. The data type of this variable must be the same as the “Label” column in the lookup table.

4.2.2 Lookup Table Naming Requirements

Lookup tables traditionally provide detailed domain information about the facts in a fact table. For example, a retail store reporting database would have a purchase fact table and a lookup table for “product” that defines serial numbers, stock quantity, and other product related details. The proposed BI application framework schema requires that the name of the lookup table must start with the prefix “lkup_”. This naming convention distinguishes the lookup tables from the fact tables for easier maintainability.

There are three required columns in the lookup table schema. A key column called <lookupName>Key is used to uniquely identify the row. A text descriptive column called <lookupName>Label is used to provide a short description of the information contained in the row. This data is used later to populate dropdown menus in the view. Finally, a column called <lookupName>Order is required to give priority to which drop down elements appear first in the list. The lookup values are automatically sorted based on this column. Negative number entries in this column are omitted from the list of drop downs.

4.2.3 Bridge table naming requirements

Bridge tables as mentioned in section 4.2.1 are used to store multiple lookup values for the same fact row. The fact table uses a generated key that indexes into the bridge table, the bridge table matches this key with the selected entries from the lookup table. Every selected entry from the lookup table is represented as a row in the bridge

table. The bridge table name must start with the prefix “lkup_”. In addition, there must also be a “Multi” postfix to help distinguish between the different types of tables in the schema. The name of the bridge table (without the multi postfix) must be the same as the name of the corresponding lookup table.

There are two required columns in the bridge table, both of which are used as a composite key to uniquely identify a row. The column <bridgeName>Key is used to index into the fact table, while the column <bridgeName>Value is used to index into the lookup table. Neither of these columns uniquely identifies a row on their own due to duplicate values. They must be used together to retrieve a single row. In general however, multiple rows are retrieved to give the selected values for a multi-select field of a fact.

4.2.4 SQL Query Table

The last requirement for a BI application framework reporting database is a sql_queries table. This table is designed to allow the application developer to create customized reporting queries. These queries are called upon by the application when a report or table is requested by the user. A query is selected from this table, executed, and a result set is returned to the DAO. Only three columns are required for this table: queryKey, queryLabel, and query. QueryKey is an automatically generated key to ensure unique identification. QueryLabel is used in the view to request the execution of a desired query. The query column contains the desired customized SQL for report generation or table population.

The naming conventions of this schema are enforced in order to reduce complexity, and optimize reporting performance. We found that this schema reduces the

number of tables required to build a BI application, and subsequently reduces the number of joins required to report off of the data. The naming conventions also facilitate the use of templates. The well-defined nature of the schema allows for robust templates that only requires minimal configuration. Finally, this schema simplifies the DAO. Minimal metadata is needed about the schema for the DAO to select data, generate queries, and join tables.

4.3. Data Access Object Service

The Data Access Object (DAO) defines several rigid methods to access the reporting database, each of which provide a unique view on the data or metadata within. The DAO service is a refinement or specialization of the DAO pattern with a special restricted set of interface calls (available by REST) that provide just the specific functionality needed by a BI app: save fact, get lookup values, update fact, get result set from query, execute query. Figure 11 below shows a high level view of the DAO service, and its interactions into the reporting database.

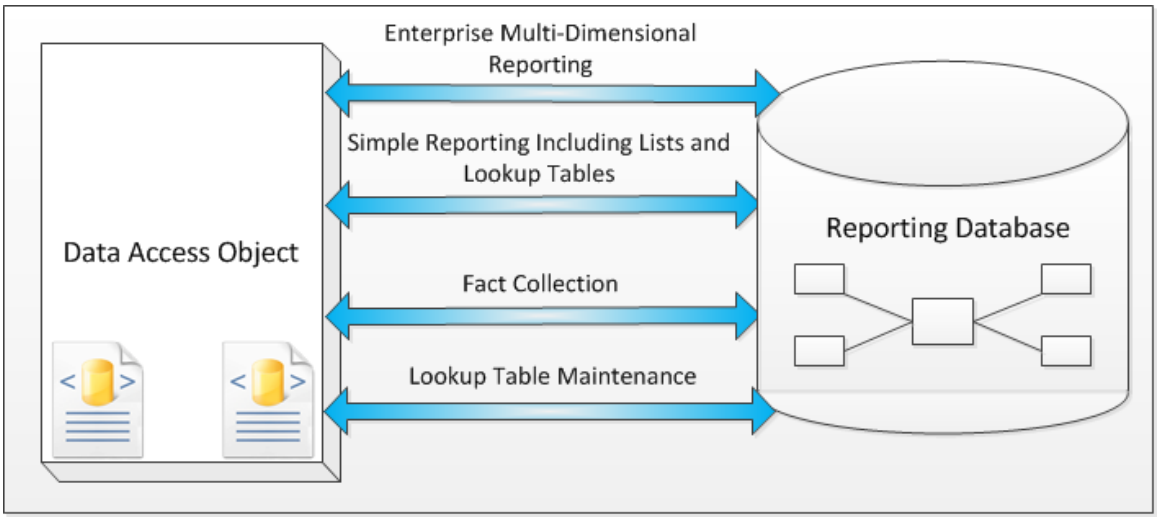


Figure 11: The Data Access Object, and its common interactions with the reporting database

The “Enterprise Multi-Dimensional Reporting task” is achieved by the execution of a highly parameterized named query. This query has dynamic where clauses and case clauses that allow for filtration of different dimensional tiers. The “Simple Reporting Including Lists and Lookup Tables” interaction is achieved through listing lookup values. All of the rows from a lookup table are retrieved and sorted based on their priority. Fact collection uses a simple insert interface that matches the columns from the fact table to the parameters provided by the JavaScript client. Finally, a series of queries and functions are provided by the DAO to manage lookup table maintenance. Below, we list in detail all of the functionality the DAO provides, and how it interacts with the database.

4.3.1 Fact Table Insertion

This DAO function inserts the data captured from a form into the reporting database. The view captures all form elements, and sends them as named parameters. The REST endpoint captures all of these parameters and inserts them into a dictionary. An inner join is made between the columns in the fact table and all of the names in the dictionary. An INSERT statement is constructed off of the results of the join, and is executed. The new auto increment key is captured after by selecting the maximum key in the fact table.

As a first step however, the parameter dictionary is queried for any names with the postfix “Multi”. A “Multi” postfix keyword flags this column as a many-to-many database relationship. In this event, a new index must be created into the corresponding bridge table. Each dictionary entry under this name is inserted into the bridge table. A new reference is made in the dictionary that refers to the new rows created in the bridge

table. This way the fact insertion step inserts this column as if it were a regular lookup index, when in fact the index references to the bridge table instead of the lookup table.

If an update id is provided, the same steps are performed. The only difference is the SQL construction. A large UPDATE statement is constructed to update each of the parameters in the dictionary. In this case, the createdDate is omitted from the dictionary to preserve the date when the record was first created.

Relevant Built-in DAO Queries:

Skeleton of the fact insertion query

```
insert into fact_?1 (%2, %3, … ,%n) values (?2, ?3, … , ?n)
```

?1 name of the fact

%2 … %n is a computed list of all of the columns in the fact table intersected with the columns filled in by the user from the client.

?2 … ?n is the values entered by the user.

Skeleton of the fact update query

```
update fact_?1 set %2 = ?2, %3 = ?3, … ,%n = ?n where ?1Key = @1
```

?1 is the name of the fact

%2 … %n is a computed list of all of the columns in the fact table intersected with the columns filled in by the user from the client.

?2 … ?n is the values entered by the user.

@1 is the id of the row to be updated.

Find the most recent fact key (executed after fact insertion)

```
select max(?1Key) as id from FACT_?1
```

?1 is the name of the fact

4.3.2 Form Data Lookup Tables

This feature is the main connection between the forms and the reporting database. The JavaScript Client can retrieve the lookup values from the database by making a call to this function. The parameters given are the database name, the current fact name, the lookup table name, and an optional update id. The update id corresponds to a previous entry in the fact table. If this parameter is given, a join will be made onto the lookup table to find the existing selected value. This is used to populate the form with an existing entry for updating.

Relevant Built-in DAO Queries:

Get hierarchical data from the lookup table

```
select ?1Key, ?1Label from lkup_?1 where ?1Order > -1 order by ?1Order
```

?1 is the name of the lookup table

4.3.3 Multi-Dimensional Lookup Tables

This DAO method is similar to listing lookup values mentioned above with the addition of a bridge table relationship. Often with larger sets of multi-dimensional lookup data, the view must display the data in a categorical manner. To solve this problem, a category parameter is used to separate the lookup data with a WHERE clause. If an UPDATE parameter is given, two extra queries must be executed to retrieve the required data. First, the fact table must be queried to find the index into the bridge table. Then, the bridge table is queried with the index as a parameter to retrieve all of the previously selected values. This could range from zero to many form multi-select entries. Each entry found in the bridge table related to this index is returned to the view as “selected”.

Relevant Built-in DAO Queries:

Get data from bridge table by fact index

```
Select * from lkup_?1Multi where ?1Key = ?2
```

?1 is the name of the lookup table,

?2 is the fact table index into the bridge table.

4.3.4 Populate Lookup Tables

This function allows the user to upload an Excel file to update the information in a lookup table. There are four parameters to this DAO function: database name, target lookup table, update data, and filter. This is a complex process, so this section is broken down into three parts: update data parameter, filter parameter, and data insertion.

The update data contains an updated in-memory representation of the entire lookup table. It is stored in a dictionary of named parameters. The name of each entry in the dictionary is the key that indexes into the lookup table. A negative integer value for the key column is a signal that this row is an insert rather than an update. Under each key entry is a list of named pairs that correspond to the column names, and the values under each column.

The filter parameter is necessary when updating a many-to-many lookup table with a bridge table. Often this data is categorized in the view, and is necessary to split into multiple Excel files. When uploading this Excel file, a button is pressed that generates the correct filtration data that corresponds to the rows that will be updated with the target file. This is necessary for deletion, as every row is signaled for delete unless it

has a corresponding key in the update data. The filter ensures that only entries in the filter category are marked for deletion.

Three steps are performed during this DAO call: INSERT, UPDATE and DELETE. As a preliminary measure, each row is marked for delete by setting the sort order to negative one. The UPDATE data dictionary is then scanned for negative key entries. Each of these entries are dispatched to the insert SQL construction, while the positive keys are dispatched to the update SQL construction. The UPDATE SQL construction matches the keys with the existing data in the lookup table, and updates the relevant information in the corresponding named pairs. The insert SQL construction first allocated new keys for each of the rows based on the current maximum. Then an insert statement is prepared for each entry.

As a final step, the new lookup table data is queried a final time to retrieve the updated values. These values are then returned to the user in an updated Excel sheet. This is important so that the keys marked for insertion with negative numbers are replaced with the real values found in the database. The user is required to use this new Excel sheet as the previous file will continue inserting new unintended lookup values to the table.

Relevant Built-in DAO Queries:

Find max lookup key

```
select max(?1Key) from lkup_?1
```

?1 is the name of the lookup table

lookup delete

```
update lkup_?1 set ?1Order = -1 where ?1Key  = ?2
```

?1 is the name of the lookup table

?2 is the row to be deleted

lookup update

```
update lkup_?1 Set %2 = ?2, … , %n = ?n where ?1Key = ?0
```

?1 is the name of the lookup table,

?2 .. ?n are the new values of the lookup,

%2 … %n are the names of the columns of the lookup table,

?0 is the key of the row to be updated

lookup insert

```
insert into lkup_?1 (%2, … , %n) values (?2, … , ?n)
```

?1 is the name of the lookup table,

?2 .. ?n are the new values of the lookup,

%2 … %n are the names of the columns of the lookup table.

4.3.5 Named Query Result Set

This DAO call is the counterpart to Execute Named Query. It maintains the same parameters and fields, and queries the sql_queries table in the same manner. The only difference is the returned SQL is run to retrieve a result set rather than to change the state of the database. This is useful for customized reports and summary views on the fact tables of the application. The filter control uses the whereclause parameter to dynamically filter the data on the server side.

Relevant Built-in DAO Queries:

Find a named query from the sql_queries table

```
select * from sql_queries where queryLabel like '?1'
```

?1 is the name of the query

4.3.6 Named Query Execution

This query allows the developer to alter a row or a table through the application. Predefined named queries are inserted into the sql_queries table. Calling this function with reference to the query name will execute the query. Parameters can be given to the function as well to add application context. This is used mostly for marking a field or a row as "deleted".

Relevant Built-in DAO Queries:

Find an executable named query in the sql_queries table

```
select * from sql_queries where queryLabel like '?1'
```

?1 is the name of the query

4.4. JavaScript Client

The JavaScript client is the generic tier that interacts between the application-specific User Interface and the Data Access Object Service. The client contains a library of JavaScript modules that are loaded dynamically when needed. One subset of the library packages up reusable controls that are fundamental to BI applications that are fundamentally linked to the Reporting Database. The other subset of this library interfaces with the DAO service defined in section 4.3. Each REST call that is provided by the DAO has a corresponding JavaScript module that matches the request parameters, and formats the returned data. The data received from the DAO passes through a series of abstraction layers until it is transformed into HTML to be consumed by the user.

There are three main packages in the JavaScript Client: "server", "dom", and "form". All of the modules that correspond to DAO REST calls are located in the

"server" package. All data transformations from server data to HTML are encapsulated in JavaScript modules. These modules are located in the "dom" package. The final package is the "form" package which is a subset of the "dom" package. These JavaScript modules specialize in form generation, and maintain references between the server and the view to ensure a valid connection is produced. Each form module corresponds to a unique form element such as text entry or a select drop down. The modules can transform server data into HTML to populate the form elements, and encode user input into a digestible format for the DAO.

There are many proposed JavaScript modules in the JavaScript client, so it would provide more benefit to only highlight the important and unique modules which work in tandem to produce a specialized BI application experience. In this section, we will provide insight as to how these controls will benefit the user, and simplify development effort.

4.4.1 Application templates: overall Look (Forms, Reports, Team, new Form)

The JavaScript Client provides many tools to enrich and enhance the BI experience for the user with little development effort. Below, we depict an example application. On the main screen alone, four unique tools have been implemented. The main dashboard can be seen in Figure 12.

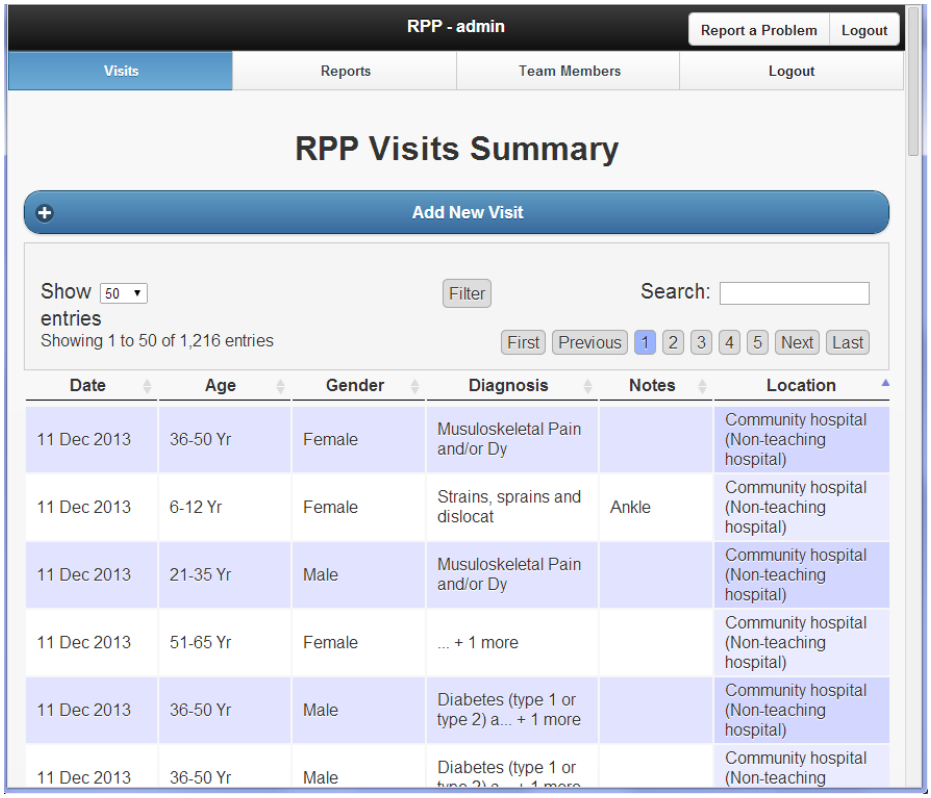


Figure 12: Sample application home dashboard

The user is brought to this screen directly after authentication, and is presented with the most recent form information they have entered. The facts that they have entered so far (Visits in Figure 12) are displayed using the DataTables client side filtration control. Each column can be sorted by alphabetical order, or by date. If the user wishes to search for a specific note made in the past, they can type in the search bar. This action will only display the rows that meet the search terms. For a deeper analysis of the form history, the user can use the DAO server side filtration tool. This will be discussed in detail in section 4.4.3.

The user may then click on any of the tabs available in the navigation bar. Reports will bring them to the BI reporting portal. Data is gathered by the server, and displayed to

the user in a series of graphs and charts. The user will use this as a tool to report on key performance metrics.

Finally, the user can enter in a new form by clicking the "Add New Form" button, or edit an existing form by clicking a row in the summary table.

4.4.2 Simple forms, simple lookup table drop downs

Much of the research for the BI application framework was devoted to optimizing the form view for the users. Our goal was to create an intuitive interface that allowed for rapid data entry. We used a series of select elements (drop downs), and checkboxes to capture the majority of the data in the form. Each of these elements connects to a column in the fact table, and pulls the lookup data from the dimensional tables. An example form is detailed in Figure 13.

Figure 13: An example form depicting select attributes and checkboxes.

After each form entry, the user can navigate to the reports tab and visualize how their form entries have affected their key performance metrics. Figure 14 displays the report that corresponds to the form described above.

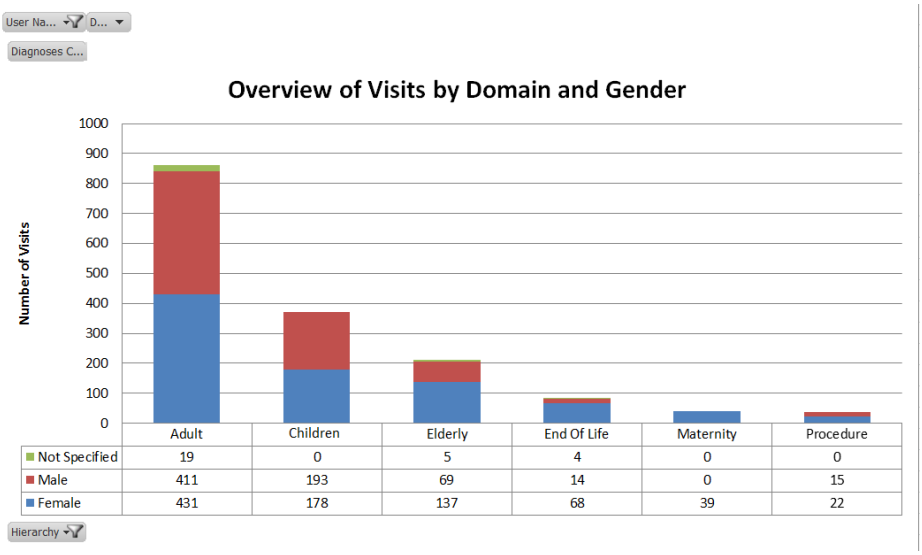


Figure 14: Demographic report correlated to the sample sub-form dialog

4.4.3 List and Filter Control

When representing a list of previously entered data, a table view is used to give a quick history. A DataTables control allows for text based searching and column sorting. Although these controls are powerful, they were not sufficient in terms of filtering controls. A JavaScript Client filter control was designed to supplement the client side DataTables pagination and filtration with DAO where clause insertion.

This control uses form elements to let the user select some attributes from the fact table to filter by. It includes powerful date range filtering, and allows for year, block, or any other date dimensional filtering. The filter data is compiled, and is inserted into the where clauses of the customized DAO queries. Figure 15 shows how the server side filter button interfaces with the DataTables header. It also shows the summary of the user's previous selection for the current filter.

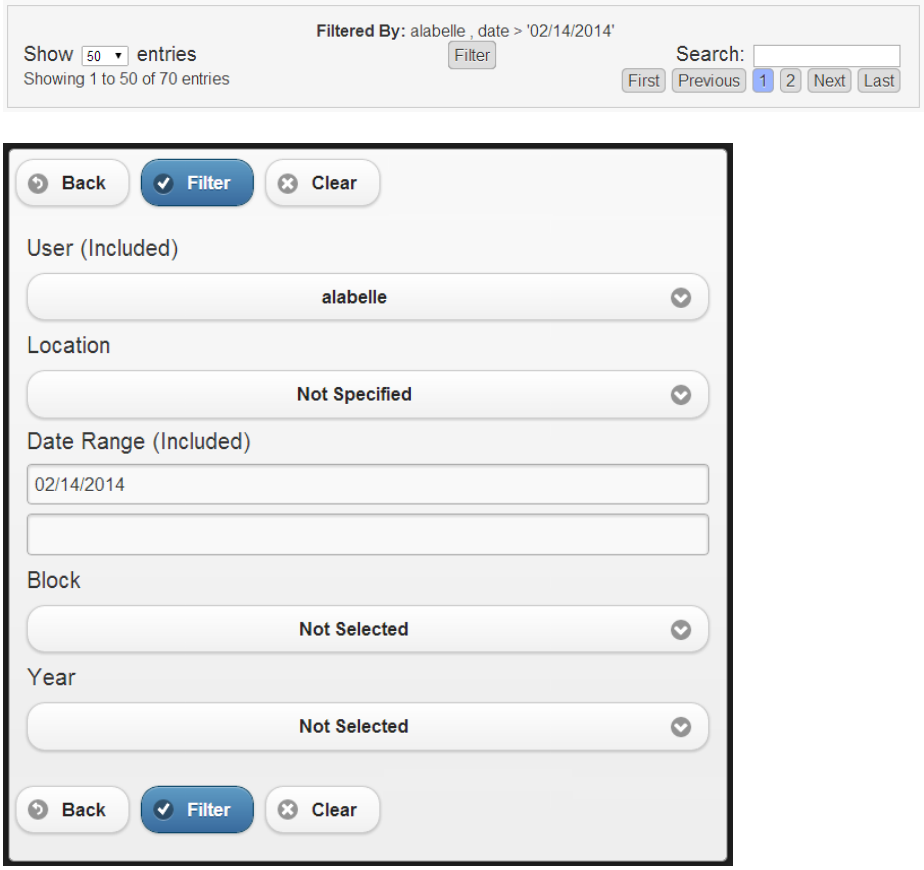


Figure 15: Server side filter button along with a summary of a previous filtration

4.4.4 Sub-forms and Sub-form summary

In order to provide an optimal user interface for large multi-element forms, categorized sub-forms are required. The user may open the sub-forms using the buttons depicted below in Figure 16 and only view a subset of the form data at a time. After the user enters data in the sub-form, a summary is generated and displayed on the main form view. This summary will help the user remember their previous input as they transition between the sub-forms and the main form.

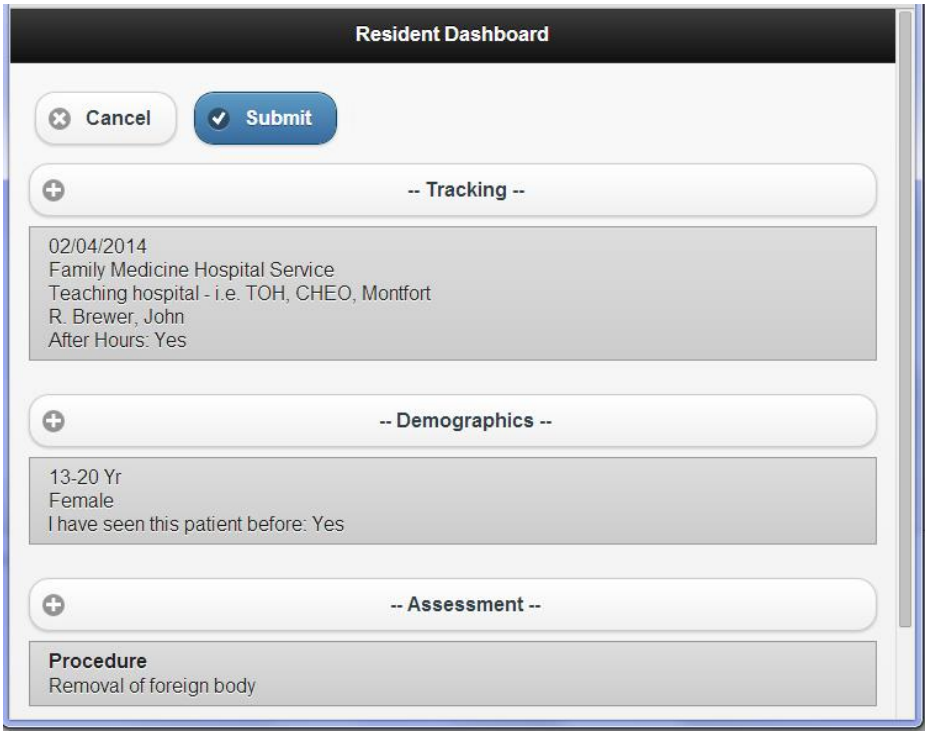


Figure 16: An example application sub-form summary view.

The significance of this value can be learned when some form elements are often similar between multiple form entries. The example form above has time and location data that would often persist between multiple entries. This form data is set to remember the user's previous input, and automatically populate the sub-form without any user interaction. This was shown to save users a considerable amount of effort and time when entering forms. The categorization of form elements into sub-forms not only provides a cleaner view of the form, but also increases productivity of the users.

4.4.5 Tabbed Multi-Dimensional Control

Displaying a large set of data in a single select element often results in an unusable UI design. Splitting the lookup data into categorical sections is one simple way to improve the usability of the form. This creates a non-trivial design task. This control,

named fillDiv, seeks to simplify the process into an easy configuration step to display the lookup data in categories.

The fillDiv control combined with the tabbed dialog allows for two levels of categorization for further readability. An example of this control is displayed below in Figure 17.

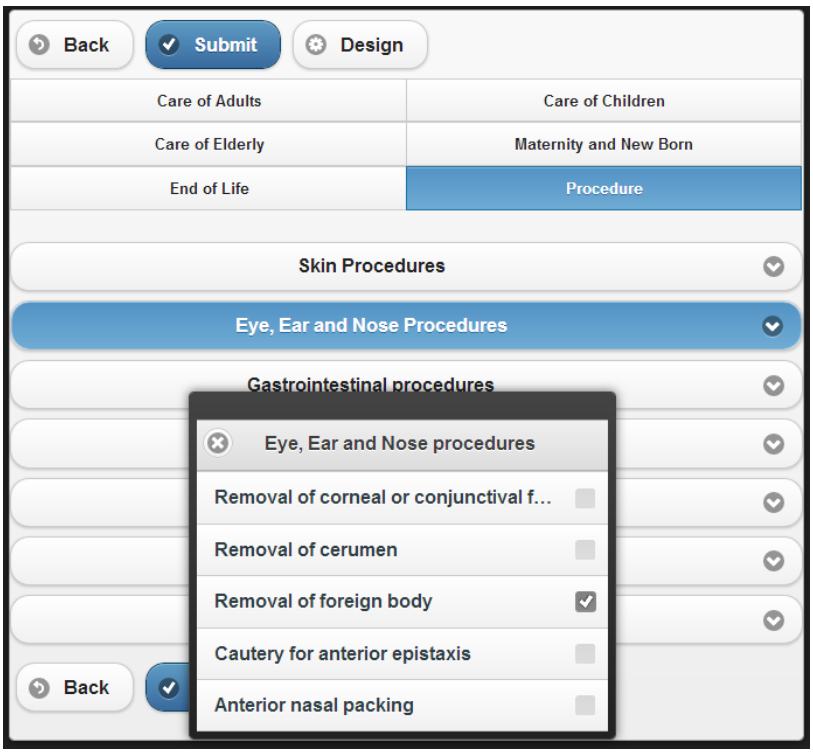


Figure 17: Sample forms application tabbed dialog interface for multi-dimensional selection

This control allows the user to select multiple choices within a drop down category. It also enables multiple selections in other higher level categories. This is reflected in Figure 17. The user can make any number of selections in the Procedure category, as well as multiple selections in the Adult category.

The main benefit of this control is database simplification. The alternate method of storing the same data without the fillDiv control required 12 separate tables (6 for each tabbed category, and 6 bridge tables). Reporting queries off of this database schema performed poorly due to the large number of joins required. With the fillDiv and tabbed dialog control, only 2 tables are required to store the sample data. This provides optimal reporting performance.

After entering a number of forms, the user can navigate to the corresponding report in the reports tab to summarize their data. Figure 18 shows an example of a multi-dimensional specific report a user would use to report on the example data.

Domain	Female	Male	Not Specified	Grand Total
Adult ⊞	**431**	**411**	**19**	**861**
Coca ⊞	**178**	**193**	**0**	**371**
Elderly ⊞	**137**	**69**	**5**	**211**
End Of Life ⊞	**68**	**14**	**4**	**86**
Maternity ⊞	**39**	**0**	**0**	**39**
Procedure ⊟	**22**	**15**	**0**	**37**
Eye, Ear and Nose procedures ⊟	**1**	**1**	**0**	**2**
Anterior nasal packing	0	0	0	0
Cautery for anterior epistaxis	0	0	0	0
Instillation of fluorescein	0	0	0	0
Removal of cerumen	0	0	0	0
Removal of corneal or conjunctival foreign body	0	0	0	0
Removal of foreign body	1	1	0	2
Slit lamp examination	0	0	0	0
Gastrointestinal procedures ⊞	**0**	**0**	**0**	**0**
GU and Women's Health procedures ⊞	**8**	**1**	**0**	**9**
Injections ⊞	**1**	**0**	**0**	**1**
Musculoskeletal procedures ⊞	**3**	**4**	**0**	**7**
Resuscitation procedures ⊞	**0**	**1**	**0**	**1**
Skin Procedures ⊞	**10**	**9**	**0**	**19**
Grand Total	**876**	**703**	**28**	**1607**

Figure 18: Report summarizing assessment data by gender

Using powerful BI tools the user can drill up and down on the hierarchical columns to accurately focus on the data granularity that they require. Above in Figure 18

the user has drilled down into ear, eye and nose procedures. This will display all of the possible procedures in this area, and the total number of forms entered respectively.

4.4.6 In-app design control for database customization

Application administrators often require minor changes in the labels of the application. To meet this need, the JavaScript Client provides a design portal to edit the data within any dimensional table linked to the fact table. The design portal is identical to the form entry view with the addition of edit buttons next to the lookup data. The edit button navigates the administrator to the edit view seen in Figure 19. The edit view enables the administrator to insert, edit and delete values from the lookup table. The order of the entries can also be changed by clicking the up or down arrows next to each row.

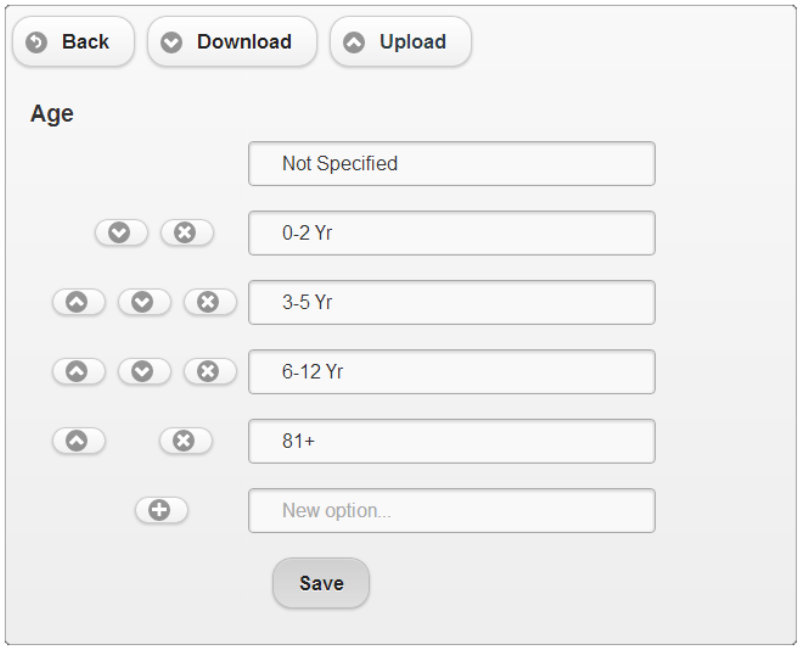


Figure 19: Edit view for the age dimensional table.

This simple control not only increases the productivity of the application administrators, but even the application developers. Front end BI application developers can skip setting up database management studios, and instead simply use the in-app design control during initial development to quickly insert lookup data.

In the event that an element is deleted from the list, all previous references to this field are maintained. New form entries however do not have the capability to select the new entry.

For more intricate dimensional tables with multiple columns, the design view provides downloading and uploading of Excel spreadsheets. The administrator enters the edit view for a dimension, and clicks download to receive the most recent lookup data. They can then make any changes such as ordering, or label corrections etc. Once the edits are completed, the administrator clicks upload and points to the file of interest. This will update the dimension table with all of the changes instantaneously.

4.4.7 One click authentication and authorization

The requirements for BI application's authentication and authorization are often very similar. This allowed for a very standardized template to be provided by the JavaScript Client. The login form template requires one line of configuration, along with the execution of an SQL script. This simple integration allows the developer to use the templated team members portal. The team members portal provides a view of all logins for the application, the ability to edit accounts, and create new team members. The authentication view of this control is visible in Figure 20.

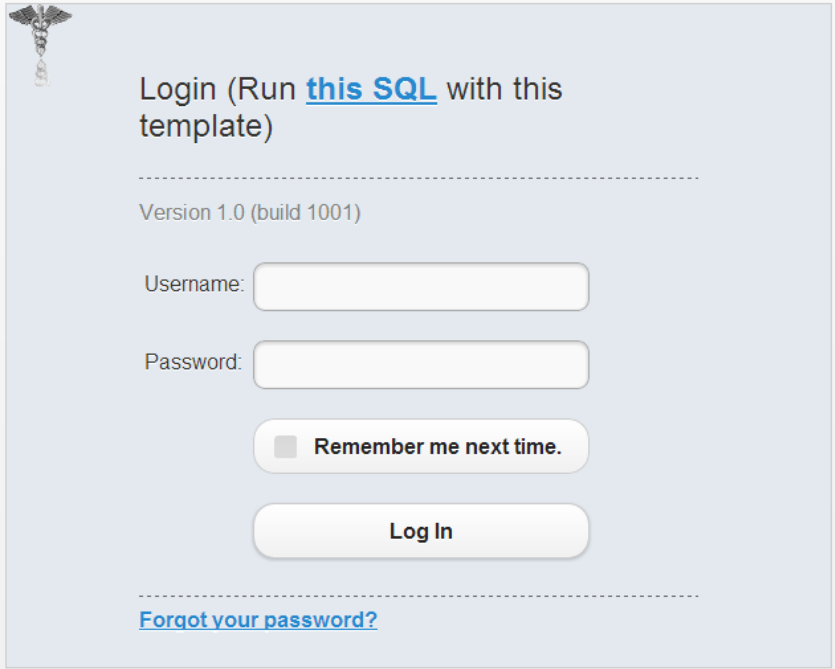


Figure 20: Authentication template with required SQL query attached

The design control also enables the administrators to add new user types for different authentication requirements. By default, the application is provided with an administration view, and a standard user view. Any extra user types that are required can be added in the design view. All user type changes are available for selection in the team member creation page.

4.5. User Interface

The user interface of the proposed BI application framework consists of a compilation of templates, and configuration. A library of templates is provided to the developer which can be combined to provide all the necessary functionality to build a BI application.

The BI application framework comes coupled with an application that visually displays all of the available templates. The templates are retrieved, and configured to

reflect the unique requirements of the BI application. A series of categories are used to help the developer filter through the template library. These categories are shown below in Figure 21.

Figure 21: Application Template Interface library view

The BI application developer would navigate to the template library with their browser, and select the category that corresponds to the current page they are mapping to the reporting database. Login page templates and landing pages are available in "Getting Started", all possible form templates are found in "Form Elements", form entry summary views, and filtration controls are in "Table Controls" and BI charts, graphs and report templates are available in "Reporting".

4.5.1 Getting Started

These templates help the BI application developer get their application started. Often the first requirement for a BI application is to add authentication and authorization.

The developer needs only to follow the very simple SQL and HTML instructions provided in the template. The BI application framework supplies a built-in user management tool to allow administrators to add, edit and delete team members. The HTML template is seen on the left in Figure 22.

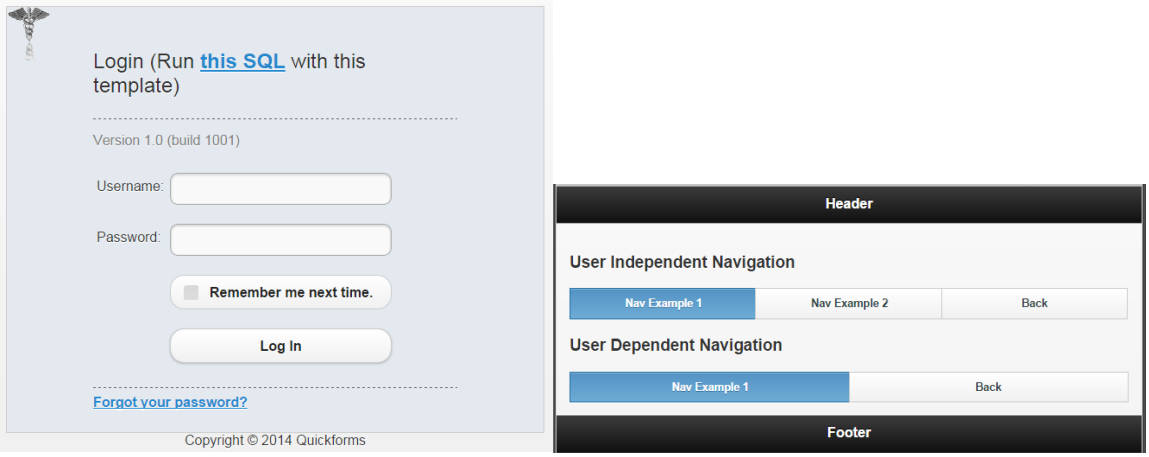


Figure 22: Getting started templates. Left: Login template. Right: Navigation template.

The other getting started templates give the developer standard navigation, and button tools to add to their application. A template is also provided for loading the JavaScript Client, and loading extra module functionality. Each template provides sample HTML which is designed for insertion into a particular file or area of an existing HTML document. The HTML samples have highlighted configuration areas where the developer can customize functionality. For the navigation template example, the page transitions and labels are configurable. This can be seen below in Figure 23.

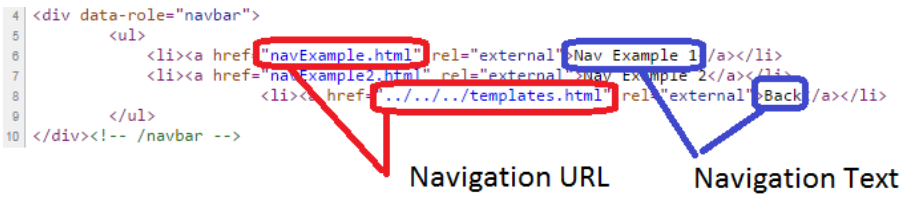


Figure 23: Navigation HTML Template

4.5.2 Form Elements

The template library provides templates for all of the supported form elements. This includes Text, Select, Checkbox, Radio, Dialog, FillDiv, Date, Date-Time, Design, Rich Text, and file upload. Each of these templates provide an example of how they may be used in a form, and how to load the corresponding module through the JavaScript Client. The select template and textbox template are displayed above in Figure 24. Some of the more complex templates such as fillDiv and dialog also come with troubleshooting tips, and database optimization suggestions.

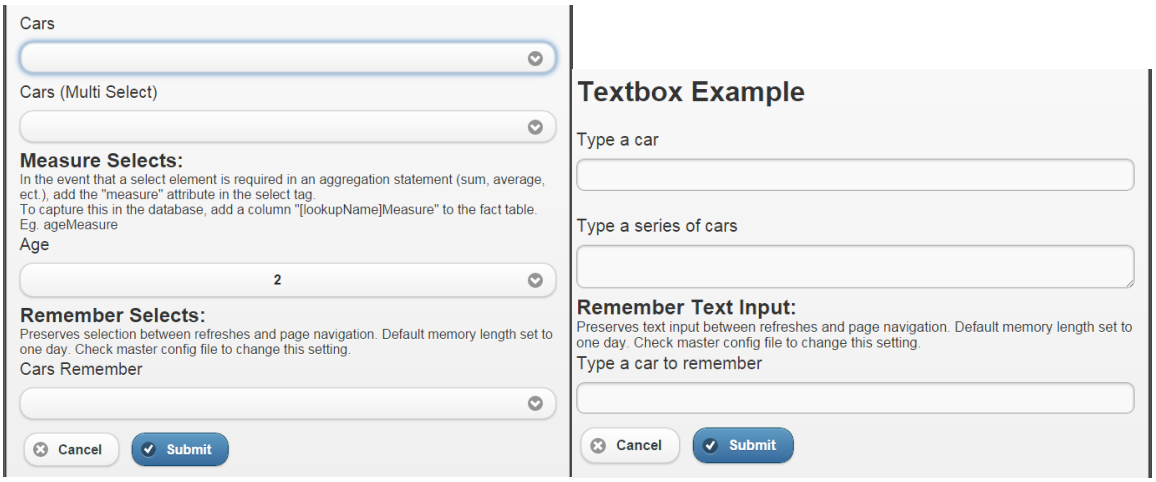


Figure 24: Form element templates. Left: Select templates. Right: Text input templates.

Form select elements are often the core element of a mobile BI application. Four different select templates are available as seen on the left in Figure 24. This template contains a traditional single select element, a multi-select element, a measure select, and an example "remember" select. The measure select acts as a lookup in the database, as well as a fact metric. The remember select can be applied to any of the select templates above. It triggers the application framework to persist the user's selection across multiple form entries. The standard select template HTML can be seen below in Figure 25.

Figure 25: HTML template for a single select element.

In this case, two stages of linking are required. The label that describes the required form entry must link to the form element. This is done through the identification attribute. The form element's name attribute maps to the lookup table in the database as mentioned in section 4.2.

The form text templates follow a similar format. The first element see on the right in Figure 24 is the standard text entry template. This enters a single value to the reporting database. The element below is a similar template, but it allows for a larger area for text entry. The final element shows an example of a remember text field for automatic text memory between form submissions.

4.5.3 Table Controls

These templates are often used for enriching the landing page after login. The table control and filter control allow the user to see previous form entries at a glance. The respective templates allow the developer to seamlessly integrate these controls together, and provide navigation as seen in Figure 26. The templates include navigation for editing previous forms, creating new forms, and creating server side filtration.

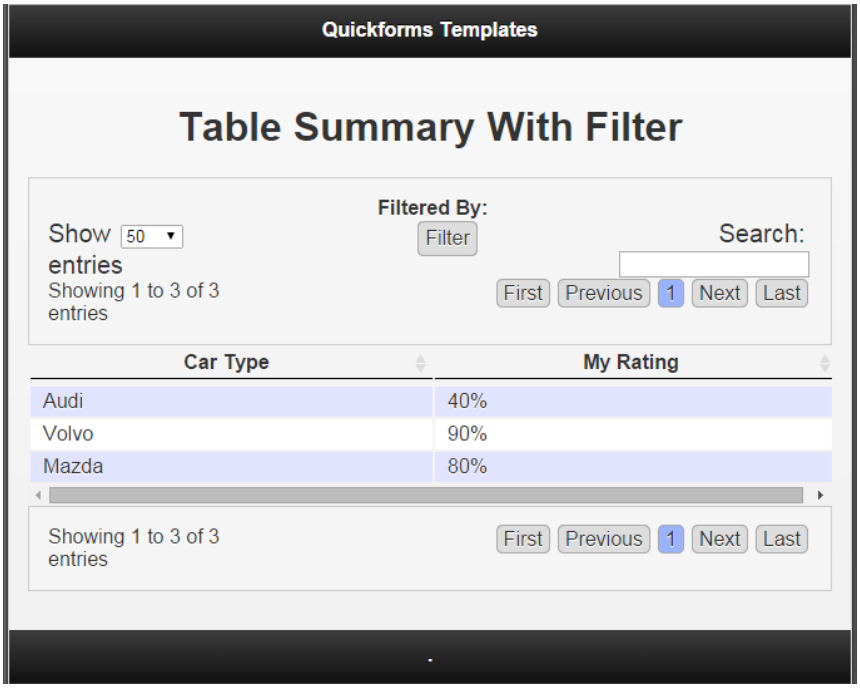


Figure 26: Table template with server side filtration control.

These controls require more development effort due to its increased customization. Each table template in the page requires a JavaScript function call that executes a customized query from the database. The HTML template can be seen below in Figure 27.

```
function(){
        quickforms.loadTable(
                {queryName: exampleQuery });

});

<form id="tableTemplate">
        <h1>Example Table Control</h1>
        <table id ="mainData"></table>
</form>
```

Named Query for Table Data

Table Placeholder

Figure 27: HTML template for the table control

The JavaScript function call here requests to execute the "exampleQuery" query. The data is inserted into the table placeholder displayed below. This example shows the default configuration for a data table. Many other parameters are available in the template to suit the needs of the application.

4.5.4 Reporting Controls

The reporting templates in the template library allow the BI application developer to easily summarize previous form entries. Pie charts, graphs, plots, tables, and filtration are available as templates. Each template must come with an accompanied query to retrieve and summarize the data, but the rest of the effort is simply configuration. Example queries are provided with each template to guide the developer through the query design. These example queries are displayed along with the reporting templates in Figure 28.

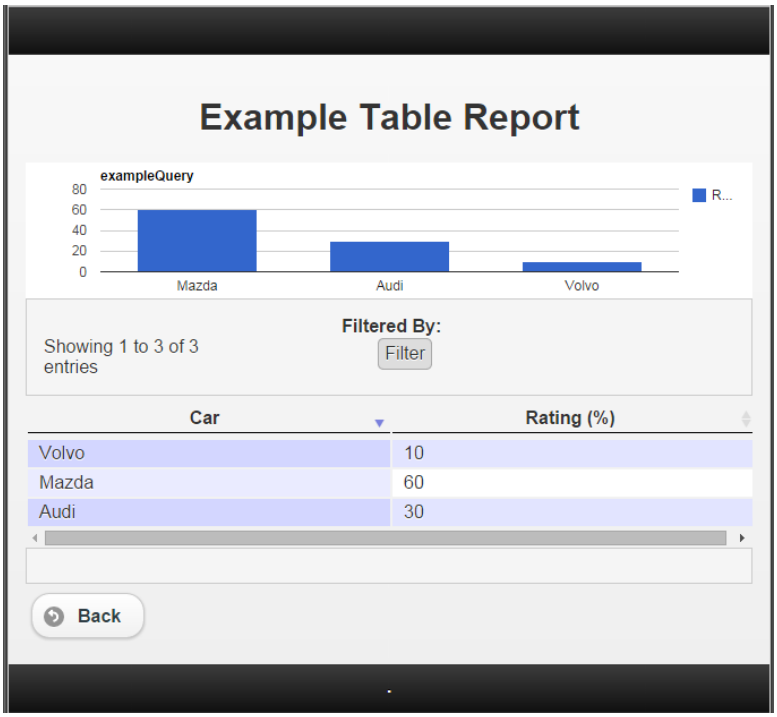


Figure 28: Table and Graph template with server side filtration.

This template requires three JavaScript calls. Two of the calls are required to load the data into the chart and the table. The third JavaScript call loads the server side filtration control, and links the filtration elements to the table and the chart. Each of these

JavaScript calls inserts data into their respective HTML placeholders. This template can be seen below in Figure 29.

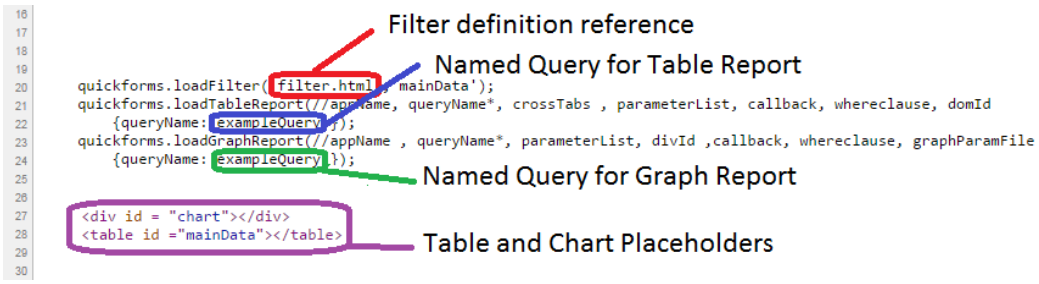


Figure 29: Graph and Table HTML template with filter.

The first JavaScript call seen above in red loads the "filter.html" file into the placeholders below. This enables server side filtration. The attributes for the filter are defined inside this external file. The format of the external file is identical to a standard form. Form elements are used to allow the user to select which attributes to filter by. A specialized "Date Range" form element template can be seen below in Figure 30 in blue. This control allows the user to select from an arbitrary date range. The name parameter in this case links to the name of the columns in the named query.

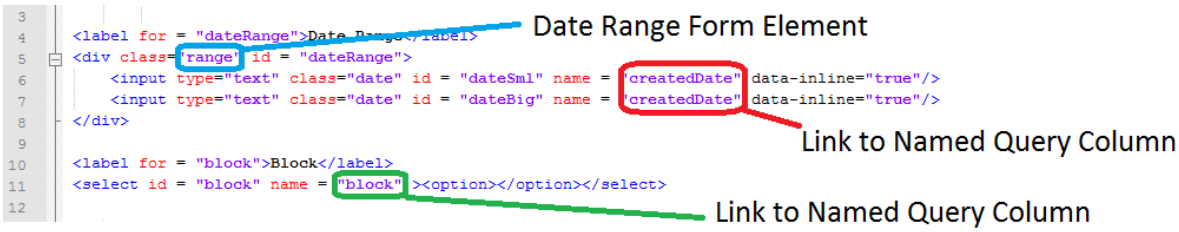


Figure 30: Filter definition HTML template.

Chapter 5. Case Studies

In this chapter, we present the initial implementation and evolution of a prototype application framework based on our thesis. Two in depth case studies are presented of how the prototype application framework was used to build two BI applications for healthcare. As well, the final version of the prototype application framework, QuickForms 3.0, has been published as an open source project, and we describe some of the applications that have been developed by that community.

5.1. QuickForms Implementation and Evolution

This section describes the implementation of our application framework, named QuickForms, and its evolution as a series of prototypes. Throughout my master's program, I was the lead architect for each of these prototypes, and collaborated with many others during their development. We give a general overview of the framework and how it evolved in 5.1.1, and then in subsequent sections we explain in detail how the specific layers or tiers of the framework evolved. Each evolution of the prototype is denoted by its version number. Below is a list of all the prototypes, and their version numbers.

- QuickForms 0.1
- QuickForms 1.0 J2EE
- QuickForms 1.0 .Net
- QuickForms 3.0

There was a QuickForms 2.0 implementation that experimented with a customized JavaScript engine that failed to provide consistent results. The implementation was discarded before any significant applications were built and re-engineered into what we now call QuickForms 3.0.

5.1.1 Application Framework

Here we give a brief overview of each QuickForms prototype, and how they have evolved between each iteration. Table 2 below identifies the differences between the tiers of each prototype.

Table 2: Overview of QuickForms prototype evolution by framework tiers.

Prototype	User Interface	JavaScript Client	DAO	BI Reporting Database
J2EE/.Net	Generic HTML.	None.	Object-Relational model.	None.
QuickForms 0.1	Generic HTML. Generated from the database.	Very simple jQuery stylization.	Database metadata parsing to generate HTML.	Rigid star schema with metadata fact tables.
QuickForms 1.0 .Net	Stylized HTML with links to database schema.	Interface to DAO. Limited support for forms controls.	Parameterized queries for OLEDB coupled with an IIS web service interface.	Rigid star schema.
QuickForms 1.0 J2EE	Stylized HTML with links to database schema.	Interface to DAO. Limited support for forms controls.	Parameterized queries for JDBC coupled with a Tomcat web service interface.	Rigid star schema.
QuickForms 3.0	Stylized HTML templates with links to database schema.	Framework for DAO interface, form controls, and reporting controls.	Parameterized queries for JDBC coupled with a Tomcat web service interface.	Templated star schema.

The initial QuickForms prototype version 0.1 was primarily a proof of concept that acted as a stepping stone for QuickForms version 1.0 for .Net. QuickForms 0.1 focused on the key component of connecting the User Interface (UI) elements to the reports. This was achieved by generating the UI from metadata on the fact table in the reporting database. Each form in the application has a corresponding fact table and a metadata table. The fact table stores the data from the form, while the metadata table defines the structure for the form. The QuickForms application logic transforms the information from the metadata table into HTML code, which is inserted into the UI.

The QuickForms 1.0 implementation took an opposite approach to developing BI applications. Both .Net and J2EE versions of QuickForms 1.0 put most of the control in application code and HTML naming conventions rather than the database. Data insertion and lookup population is achieved using web services, and the view is defined entirely by hard coded HTML. To connect the HTML elements to the database, a strict naming convention is applied. A form element is required to have a name property. The value of the name must match the column of the form's fact table.

We use the previous patient entry form example to explain the workflow of QuickForms 1.0 form entry. The HTML page is loaded, and a JavaScript control parses the HTML page to identify form elements. Each named attribute calls a web service to retrieve the related lookup elements. If the name of the element matches the fact table column, and the corresponding lookup table, the lookup table rows are returned in JSON format. The JSON is parsed by the JavaScript controller, and inserted into the view as

options for the drop down. Each time a new data type is required by the application (such as date, or file upload), the web developer must make a corresponding JavaScript control.

QuickForms 3.0 uses the same web service JavaScript model as explained for QuickForms 1.0. The main difference is the introduction of reusable JavaScript modules. Instead of creating application specific JavaScript controls to connect to the web services, QuickForms 3.0 supplies the web developer with generic modules. The web developer is only required to configure the controls, and the data connection between the form and the database.

In the patient form example, the text, date, and select modules are loaded. The modules parse the form to search for their corresponding elements. The select module requests for the options available from the gender dimension, and populates the view with "male" and "female". The submit button is pressed, and the three modules summarize the inputted data, and sends it to the putFact web service.

5.1.2 BI Reporting Database

Each version of QuickForms uses the standard star schema practices as defined by (Kimball & Ross, 2013) as mentioned in Chapter 4. There are few subtle differences between the database schemas in the different iterations of QuickForms. As identified in section 5.1.1 J2EE and .NET had an ORM-style database (generated from Object Model) while the QuickForms releases were all based on a multi-dimensional star schema model. QuickForms 0.1 used a complex metadata table mechanism to generically map form fields to fact table columns and lookup tables, while QuickForms 1.0 and 3.0 has a much simpler and direct mechanisms based on a standardized naming mechanism. QuickForms

3.0 improved on QuickForms 1.0 with a much simpler and consistent mechanism for multi-dimensional lookup tables and provided a standardized template.

Each version of QuickForms has an extra table to store application specific queries for reports, charts, and summary tables. Each of these queries are named, and made available through the QuickForms web service. The parameterization of the queries in this table differed with each iteration. Initially, in iteration 0.1, a where clause keyword "%whereclause%" was inserted into the query. This keyword is replaced by actual parameters in the data access object service. QuickForms 1.0 approached parameterization differently. Each parameter corresponded to a question mark in the named query. This allowed parameterization in "case" clauses as well as "where" clauses, but creates more rigid queries that cannot dynamically change filtration columns. Finally in QuickForms 3.0, both types of parameterization were made available to flexibly support both filtered (adding additional where clauses to constrain queries at run-time) and parameterized (varying the conditions of where clauses at run time) query for dynamic reporting.

5.1.3 Data Access Object Service

The DAO Service provides very specific functionality for BI applications that greatly simplifies the application logic so that there is no need to map between forms and an object model and between an object model and a database as there is in generic application development using general purpose J2EE and .Net application frameworks. This component does not exist in either the J2EE or .Net frameworks. The DAO Service for each QuickForms version was very similar in terms of the basic API they supported

for mapping forms to the reporting DB (saving facts and retrieving lookup values), but had significant differences in how that functionality was implemented and what additional functionality they supported. In QuickForms 0.1, the DAO generated the HTML for the form from the metadata table definitions whereas all subsequent versions simply returned data in JSON format and a naming convention was used to infer table names from form elements. All versions of the QuickForms' DAO generate parameterized SQL to insert and modify fact information, as well as retrieve lookup table data. QuickForms 3.0 makes a small addition of an administrative lookup table editor, which allows for insertion, deletion, and modification of lookup table rows. Below in Table 3 is a list of capabilities for each iteration of QuickForms:

Table 3: QuickForms DAO capabilities by iteration.

Capability	QuickForms 0.1	QuickForms 1.0	QuickForms 3.0
HTML form element generation based on fact metadata table	Yes	No	No
Insert a row into a fact table	Yes	Yes	Yes
Request lookup data for drop down population	Yes	Yes	Yes
Execute a named query for reporting	Yes	Yes	Yes
Execute a named query for updating a fact or marking a fact as deleted.	Yes	Yes	Yes
Where clause named query parameterization	Yes	No	Yes
Question mark named query parameterization	No	Yes	Yes
Upload a file (such as an audio clip or image)	Yes	Limited	Limited
Download lookup table in CSV format	No	No	Yes
Upload lookup table data in CSV format	No	No	Yes

List lookup data from a many to many relationship table	No	Limited	Yes
Save data into a many to many relationship table	No	Limited	Yes
Categorize lookup table retrieval by subcategory	No	No	Yes

The file upload DAO capability is limited for QuickForms version 1.0 and 3.0 due to its inability to save this data to the database. Instead, these files are stored locally on the web server. The QuickForms 1.0 many-to-many relationship data insertion is limited due to a design error.

5.1.4 JavaScript Client

The JavaScript client was rewritten in every version of the QuickForms prototype. The JavaScript client barely existed at all in QuickForms 0.1, since HTML and JavaScript was generated by the DAO service. The only purpose of the client in QuickForms 0.1 was to apply CSS styles to the tables and reports. QuickForms 1.0 used jQuery to have full control of the look and feel of the user interface on the client, removing the need to generate HTML and JavaScript in the DAO Service, but the resulting JavaScript client was very complex.

The QuickForms 3.0 JavaScript client was reengineered to simplify the application logic required. An investigation was conducted to determine the most appropriate JavaScript client for the QuickForms application framework, including a review of the most popular JavaScript frameworks available (see Chapter 2 for a complete description). In the end, we built a hybrid client by combining the best elements of Dojo and jQuery.

Both jQuery and Dojo frameworks have their strengths and pitfalls. The QuickForms JavaScript client requires a simple, yet scalable environment to allow app developers of all background to build BI applications. As a result, the two frameworks were combined in the implementation of QuickForms 3.0. All of jQuery and jQuery Mobile are included in the QuickForms JavaScript client, along with the Dojo AMD module loader RequireJS. These libraries together provide the optimal JavaScript client to optimize scalability, and decrease complexity.

Table 4: QuickForms JavaScript client capabilities by iteration.

Capability	QuickForms 0.1	QuickForms 1.0	QuickForms 3.0
Stylize the navigation bar	Yes	Yes	Yes
Stylize summary data tables	Yes	Yes	Yes
Apply jQuery Mobile for multi-form-factor support	No	Yes	Yes
Generate HTML for select drop down elements	No	Yes	Yes
Automatically apply jQuery datepicker style to date elements	No	Yes	Yes
jQuery date-time picker	No	No	Yes
Populate an HTML tag with categorical input or select elements	No	No	Yes
Application specific JavaScript configuration file	No	No	Yes
Built in popup dialog and form summary support	No	No	Yes
In-app lookup table design control	No	No	Yes
Table filter control for automatic where clause generation	No	No	Yes
Dynamic navigation bar insertion	No	No	Yes
QuickForms feature plugin/override support	No	No	Yes

Flexible and configurable Google charts control	No	No	Yes
"Offline" mode for application testing without a reporting database	No	No	Yes

Using this JavaScript client, QuickForms 3.0 has similar automated triggers to the 1.0 JavaScript client, but all of the functionality is separated out horizontally into JavaScript Modules. The multiple additions to the QuickForms 3.0 client were seamless due to this modular nature. The chart above describes the change in functionality between the versions of the QuickForms AJAX clients.

5.1.5 User Interface

The QuickForms 0.1 had generated code from the DAO Service which limited the usability, appeal and appearance of the user interface and limited the ability to support a variety of devices and form factors. QuickForms 1.0 redesigned the original interface using jQuery Mobile. This interface was well received by the case study participants, so minimal changes were made to the look and feel of QuickForms 3.0 applications. There were however large changes in the method of user interface development. QuickForms 1.0 applications were developed using tutorials and sources from the jQuery Mobile API reference. These code references had to be modified to fit the JavaScript client, and redesigned to cover the necessary BI application requirements. QuickForms 3.0 simplified UI development down to mostly configuration of templates and controls. Developers use the built in template library to introduce new content into the application rather that customized HTML development. Several administrative buttons and controls were also added to the framework to improve the administration capabilities of BI applications.

5.2. RPP

Resident Practice Profile (RPP) is a practice profile application that enables residents in the Family Medicine program to record their experiences during their rounds. Residents record patient demographic information, diagnoses that are covered in the postgraduate Family Medicine curriculum, and self-assessments for their clinical experience comfort level. This information can then be used to identify areas where residents require more experience, and to ensure residents are seeing a breath of demographics during their practice. Dr. Wooltorton and Dr. Viner at the University of Ottawa medical school provided details on the Family Medicine curriculum, and monitored the progress of the residents during their trial program. Dr. Archibald, a data science researcher at Bruyère hospital did an iterative, qualitative evaluation of the usability of the application throughout the project using hosted Think Aloud sessions with application users (medical residents). Four residents and two physicians used this application for a trial period of six months to test its validity, and to provide valuable feedback on the residency program for Family Medicine. In total, they logged 1043 patient visits from a wide variety of locations.

5.2.1 Application

The RPP BI application leveraged many controls and tools that were provided by the QuickForms application framework. Residents were capable of navigating through their previous patient visits, adding new encounters, and searching through a diagnosis selection menu. All of these capabilities have a corresponding QuickForms control that make the development of this application largely a matter of configuration. See

(Chamney, Mata, Viner, Archibald, & Peyton, 2014) for a published article on the development of this application using QuickForms.

Figure 31 shows an overview of the RPP application. On the left is the home page which uses one of the built-in home page templates with header, footer, application tabs, and a standard login/logout. The titles, tabs etc. are customizable in an application configuration file. The grid control for displaying visits supports filtering, searching, and sorting by clicking on columns. The columns used in the grid and in the associated filters are all easily configured. On the right is the overview summary for a particular visit. The user can see at a glance a summary of what has been logged for Tracking, Demographics and Assessment without opening their related sub-forms. The form summaries are a built-in control that can be flexibly configured to determine what fields are captured. The idea for form summaries came from one of the Think Aloud sessions when residents were having difficulty seeing the complete overview of what they had filled in for a patient visit. As well, the Tracking summary coupled with form memory greatly simplified data entry. Typically, a resident would see many patients in a row at a given clinic. When the Tracking information is entered for the first visit of the day, it is automatically repeated and pre-filled for subsequent visits. The resident can see at a glance that no data entry is needed for tracking.

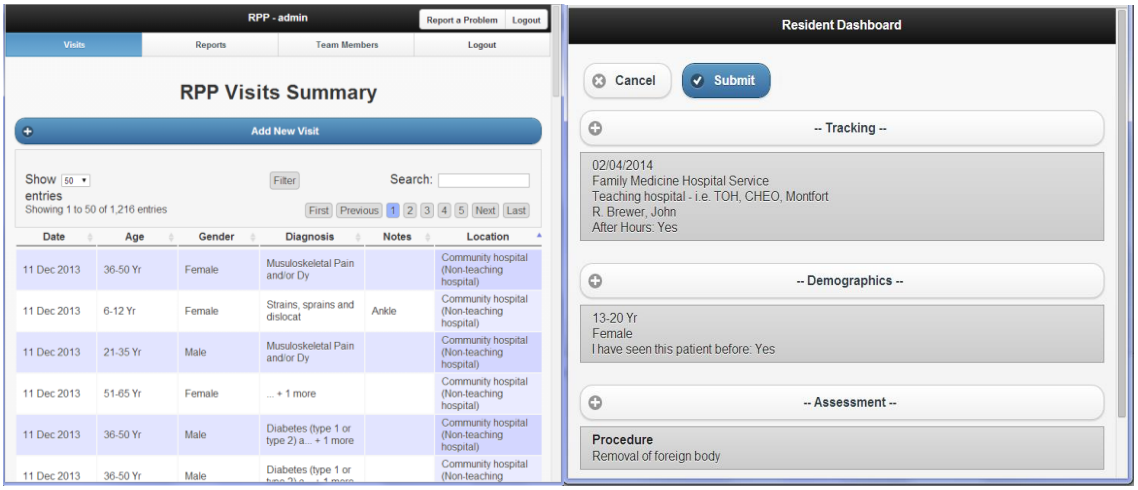


Figure 31: RPP application overview. Left: Login landing page with visits history. Right: Visits form entry page

The critical element of the QuickForms architecture is, of course, the ability to separate and yet automatically link forms with the Reporting Database. In Figure 32, we see the Demographics Form on the left and its associated Age / Gender report on the right. The developer links a field in the form to the dimensional table that supplies the drop down values for that field by simply adding the Model Name of the table to the form. These same values are used in the report associated with the form. The Age/Gender Report uses the same values for Age and Gender that appear in the selection controls for Age and Gender. In this case, the resident has seen a disproportionate number of females to male patients in the 21-35 year range, but this is normal for family doctors who see maternity patients. In addition, QuickForms adds a Design button on the form on the left that is available for users with Administrator privileges. The design button brings up a dialog where administrators can edit the values in the lookup tables for any of the form fields, or download and upload files of the values for a particular field.

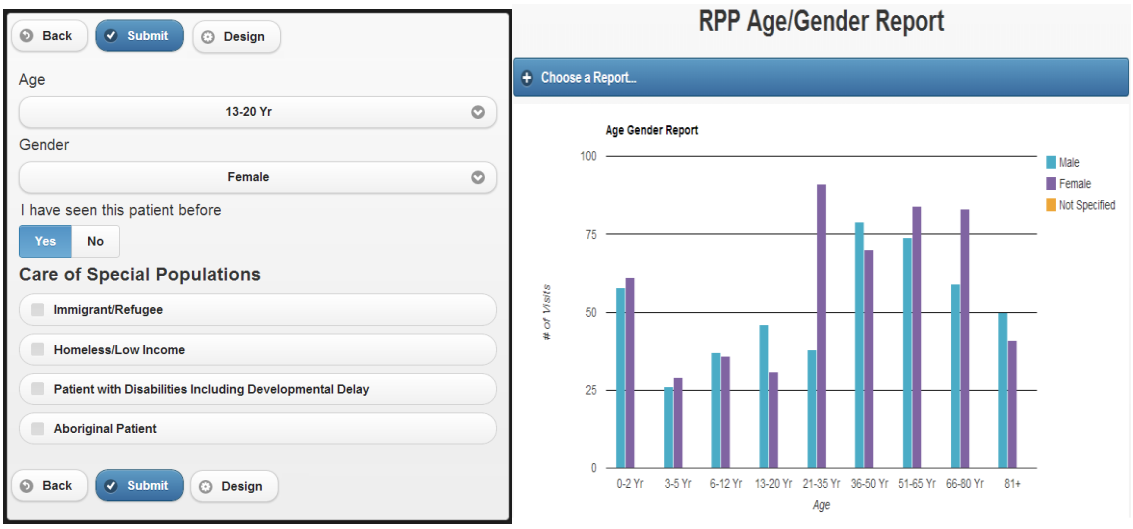


Figure 32: Demographics Form and Age/Gender Report

QuickForms excels at leveraging dimensional models for data filtration. Figure 33 shows the filter control and associated dialog for RPP forms. It includes built-in date range filtering, and allows for any date dimension filtering based on the calendar defined in the reporting database. In this case, the RPP calendar defines thirteen 4-week Blocks (rather than twelve months). This type of calendar is typical in many industries including manufacturing. The Block attribute, along with an arbitrary date range can be used for customized data filtering. As well the filter control is re-usable and allows one to configure which dimensions can be used filtering. In this case, in addition to date, one can filter on location and user.

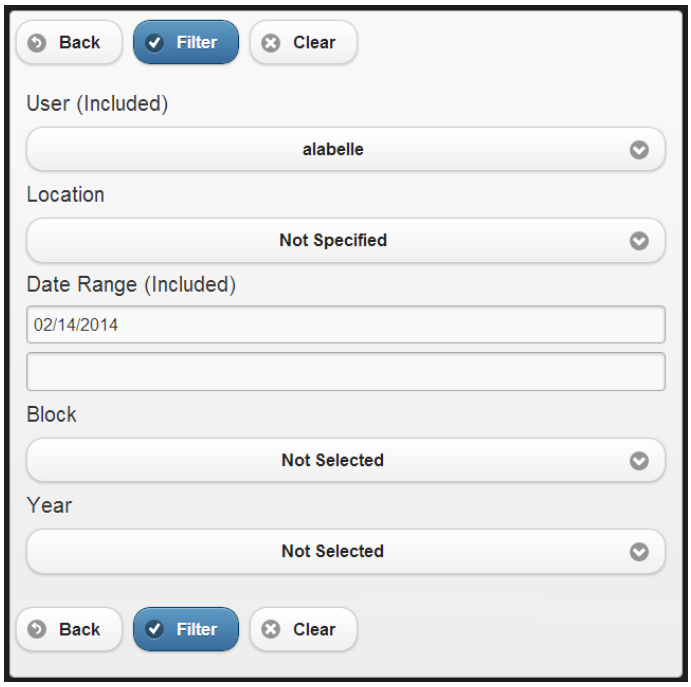
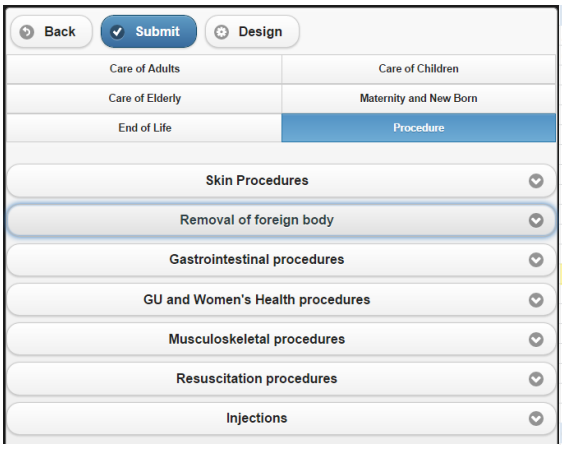


Figure 33: Filter Control and Associated Dialog

The sub form for "Assessment" is more complex. Assessment is a multi-select field to capture the diagnoses and procedures that a resident has assessed for a particular patient. The multi-level selection is shown on the left in Figure 34, and on the right shows a drillable report (displayed in MS Excel, using a MS Analysis Studio reporting tool) of all diagnoses and procedures experienced by the resident.



Domain	Female	Male	Not Specified	Grand Total
Adult	431	411	19	861
Coca	178	193	0	371
Elderly	137	69	5	211
End Of Life	68	14	4	86
Maternity	39	0	0	39
Procedure	22	15	0	37
Eye, Ear and Nose procedures	1	1	0	2
Anterior nasal packing	0	0	0	0
Cautery for anterior epistaxis	0	0	0	0
Instillation of fluorescein	0	0	0	0
Removal of cerumen	0	0	0	0
Removal of corneal or conjunctival foreign body	0	0	0	0
Removal of foreign body	1	1	0	2
Slit lamp examination	0	0	0	0
Gastrointestinal procedures	0	0	0	0
GU and Women's Health procedures	8	1	0	9
Injections	1	0	0	1
Musculoskeletal procedures	3	4	0	7
Resuscitation procedures	0	1	0	1
Skin Procedures	10	9	0	19
Grand Total	876	703	28	1607

Figure 34: Multi-Level Selection Control for Diagnoses. Left: Diagnosis control in the form. Right: Report generated based on diagnosis data.

There are roughly 500 diagnoses and procedures covered by the Family Medicine curriculum and they are organized in the multi-selection control in a similar, though simplified, fashion to their classification in the curriculum. At the top level, there are six tabs corresponding to the major curriculum domains: Care of Adults, Care of Children, Care of Elderly, Maternity, End of Life, and Procedures. Within each tab, there is another list of groupings, and within each grouping there are the selections to be made. In this case, the resident is selecting "Removal of Foreign Body" from the list of Eye, Ear and Nose Procedures under the Procedures tab. The organization of the tabs and groupings and selections is automatic based on the lookup table for diagnoses and procedures in the database. The organization is driven by the classification columns (one for tabs, one for groupings, and the sequence number to order entries). As well, the "Design" button facilitates maintenance of diagnoses and procedures by allowing separate CSVs to be uploaded and downloaded for each tab. The drillable report in Figure 34 leverages the same classification columns to organize the report of how many assessments and procedures the resident has seen. One can see at a glance, that in

addition to Eye, Ear and Nose procedures, the resident has done very few Gastrointestinal, Injection, or Resuscitation procedures and the vast majority of their diagnoses have been for the Care of Adults. This flexible report enables the resident to flexibly navigate through the multiple hierarchies, gaining important metrics on the gaps of their practice. The standard QuickForms multi-dimensional schema is capable of generating this report without any ETL or views.

5.2.2 Framework

The final RPP application as seen in section 5.2.1 was developed in several iterations which used several types of application frameworks. This section discusses which application frameworks were used, and the differences between each iteration.

RPP Implemented in J2EE

In this release, the reporting database was poorly structured and there was no support for multi-dimension attributes which were needed for the diagnosis multi-selection and reporting. As well, mapping from form to database was ad hoc and difficult to maintain, and the user interface lacked consistency which made it hard to use and difficult to maintain. The jQuery library was eventually incorporated so that a complete User Interface was created that was used to get feedback from Dr. Viner's research team. However, at that point the maintenance issue and multi-dimensional reporting issue prompted a rethink of the architecture that resulted in the adoption of QuickForms 1.0.

RPP Implemented in QuickForms 1.0

This version was used to build a production version of RPP that was piloted with 4 medical residents using it as part of their training for the 2013-2014 year as well as a

complete evaluation by Dr. Viner, Dr. Wooltorton, Mata and Archibald. The feedback was positive and a further rollout to the entire class of 75 residents is planned for 2014-2015. However, this rollout will be done with an improved re-engineered version of RPP in QuickForms 3.0.

The QuickForms 1.0 implementation of RPP suffered from the complexity of the customized JavaScript code. All three of the new features (form summary, multi-diagnosis, and filtration) introduced to this RPP implementation were custom-coded and not directly supported by the QuickForms framework. Many customized queries had to be created to display the diagnosis data based on category, and the filter control doubled the required development effort.

The QuickForms 1.0 reporting database schema did not support dimensional hierarchy columns to organize lookup data by category. In order to mitigate this problem, six different lookups were created for each diagnosis category (adult, children, elderly, end of life, maternity, and procedure). Six bridge tables were also created to allow multi-selection from each of these categories. In total, twelve tables were created to support the diagnosis feature in the QuickForms 1.0 RPP implementation.

RPP Implemented in QuickForms 3.0

The goal of this iteration was to re-implement all of the previous iteration features of RPP to the new QuickForms 3.0 framework. All of the features of the previous implementation remained with the addition of administrative controls, and small changes to the filter. The introduction of JavaScript modules in QuickForms 3.0 replaced much of the application specific logic with configuration. The reporting database schema was

simplified. The twelve diagnosis tables in the previous RPP version were reduced to two tables (one bridge table and one lookup table). All of the reports were rewritten to incorporate the changes made to the diagnosis tables.

5.2.3 Summary of Experience and Results

The RPP case study is very successful in terms of positive feedback from the residents, and the showcasing of the capabilities provided by the QuickForms application framework (Mata, In Progress). The vast number of iterations that RPP has undergone produced a polished product that provides valuable feedback for the residents. RPP also drove much of the innovation that went into the QuickForms 3.0 application framework. RPP helped to identify gaps in traditional web application frameworks, and QuickForms 1.0.

An intensive qualitative analysis was conducted based on the feedback of the application through think aloud sessions with the residents and physicians (Mata, In Progress). The feedback from these sessions incrementally improved the quality of the application, along with its usefulness to the residents. As a result, a full scale application will be deployed to the Bruyère hospital in 2015 to be used by seventy five residents.

5.3. PAL-IS / SAID

PAL-IS is a palliative care management system designed to help healthcare providers manage their consultation data. Palliative care helps terminally ill patients live their remaining days with the highest quality of life as possible. This is achieved by maintaining monthly visits, administering the correct medication, and having available staff for emergencies. Emergency visits to the ER are costly, and traumatic to patients, so

much of the focus in palliative care is to maintain the patient as comfortable as possible at home and minimize the average number of visits to the hospital. The QuickForms application PAL-IS was designed to record visits to patients homes, and the number of visits the patient has made to the ER. The hope is that this application will show proof that palliative care has improved the general well-being of terminally ill patients.

5.3.1 Application

This application is significant from the QuickForms perspective because it stresses the application framework from a scalability standpoint. Each form in PAL-IS is relatively small and simple compared to the form RPP, however there are many more forms, and a considerable number of forms that can be filled out by a wide breadth of user groups. Nurses, physicians, education managers, and administrators use the following forms: Education, non-case-based, rounds, face-to-face consultation, initial consultation, follow-up consultation, ESAS, and patient info.

The patient information form is seen below in Figure 35. This form is entered by a nurse when a patient is entered into the palliative care program. It contains the fields for date of birth, number of visits to the Emergency Room (ER), and number of emergency consultations.

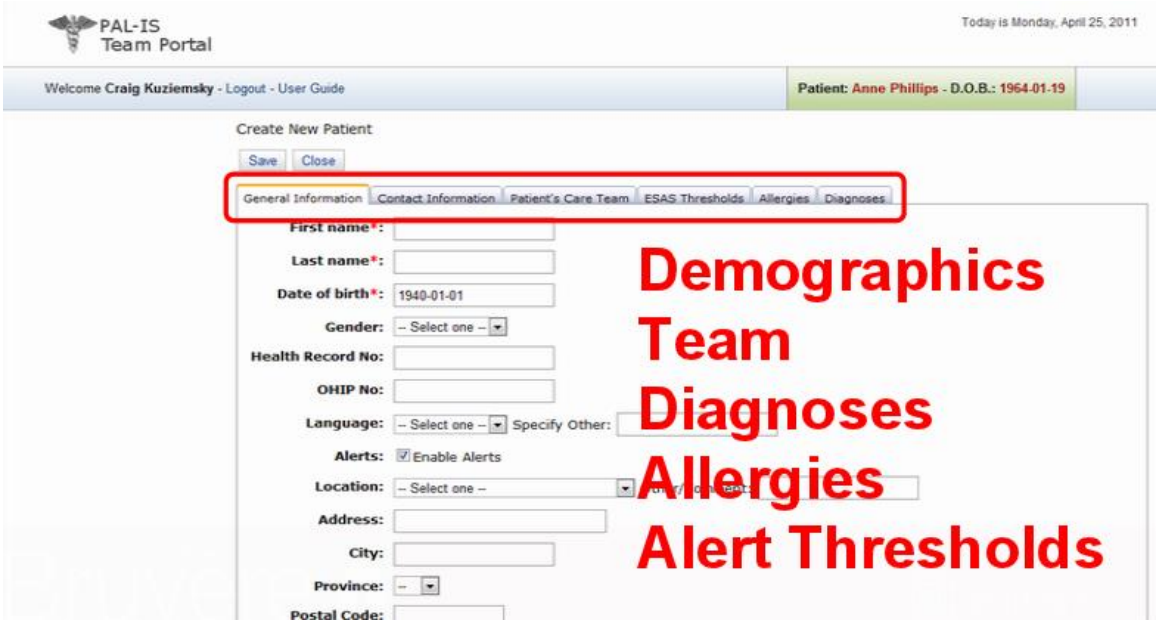


Figure 35: PAL-IS v1 patient entry form developed in Microsoft .Net

PAL-IS also measures education about the palliative care program, and records the number of nurses and physicians that receive this training. As more nurses and physicians receive training about the palliative care program, it is expected that the quality of life of patients increase, as well as having fewer emergency visits to the ER. This form can be seen below in Figure 36. This data is recorded to capture whether there is a correlation between education about the Palliative care program, and the number of ER visits that are required for palliative care patients.

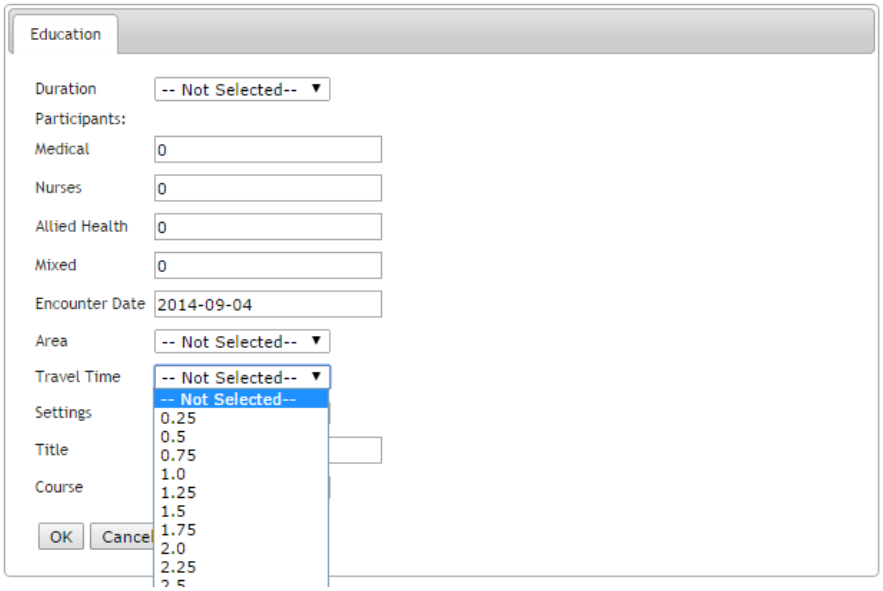


Figure 36: Implementation of the education form in PAL-IS v2 using QuickForms 0.1

An Edmonton Symptom Assessment System (ESAS) ranking is used to assess the general wellbeing of the patient. Pain, tiredness, nausea, and other such attributes are measured as seen in Figure 37 below. This score is expected to rise as more palliative care training is received.

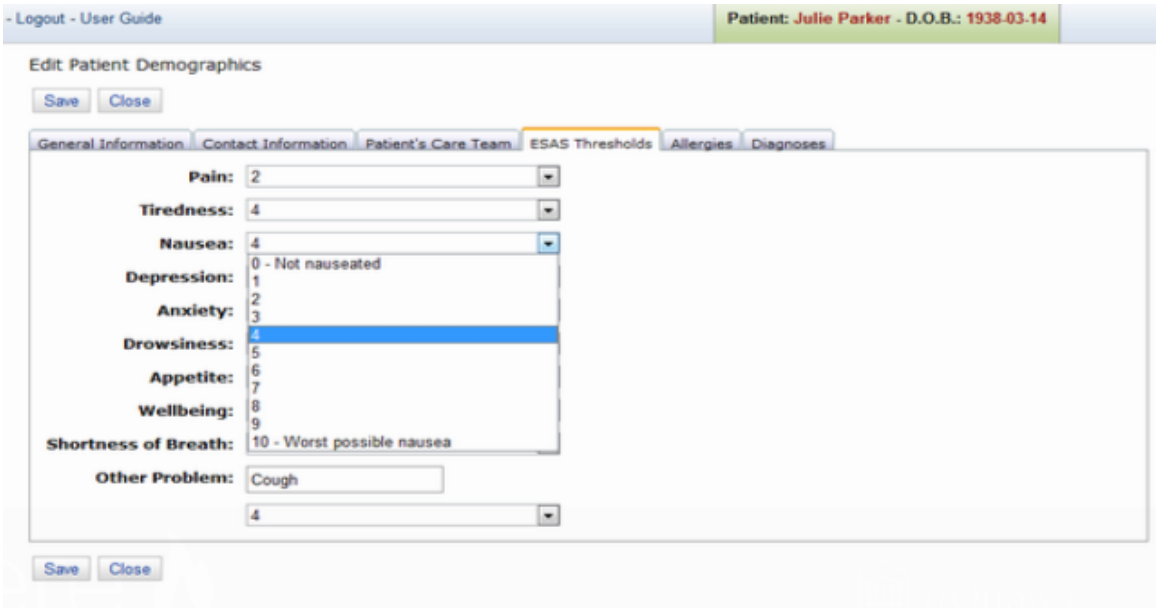


Figure 37: Implementation of the ESAS form in PAL-IS v1 using Microsoft .Net

5.3.2 Framework

This case study was initially developed in Microsoft .Net framework. PAL-IS v1 was a huge development effort and was used for a year after it completed in production. The design of PAL-IS 1 focused on enacting an operational system, with reporting added on after. The reports did not accurately reflect the needs of the physicians however. PAL-IS 2 was prototyped with the QuickForms 0.1 application framework. This iteration succeeded by focusing on the attributes needed for reporting rather than enacting a business process. The users of the application were able to retrieve significant knowledge about the palliative care program at Bruyère hospital, but the application suffered from a lack of availability. A lack of multi form factor support caused the interface to be unusable on a mobile device. PAL-IS was redesigned in version 3.0 to optimize usability on a mobile device. This iteration was built using QuickForms 1.0. Finally, an application called SAID was developed in order to move from a BI application for health care providers at Bruyère hospital in Ottawa to a BI application that would monitor palliative care from all providers across the Champlain Local Health Integration Network (LHIN) district (Greater Ottawa region). This iteration was developed with QuickForms 3.0.

Microsoft .Net Application

PAL-IS 1 was developed using Microsoft .Net. The application attempted to mimic the operational palliative care referral process. This included 100 different form fields, and a considerable number of reports. In total, four different types of users used the application. All of the functionality was available to all four user types in the home page. The application was found to be too confusing for the physicians and nurses due to the form complexity, and the busy interface. One of the main reasons that the forms were

unusable was due to their rigidity. Any change to form labels or drop downs required a whole different build to be deployed. The reports for the application also failed at capturing the needs of the physicians.

QuickForms 0.1

PAL-IS 2 was implemented using QuickForms 0.1, and introduced the much desired form dropdown configurability to the administrators of the application. The user interface is comprised of tabs and views. Each user role has assigned portals that only show the desired tabs to the user. Each tab has a view that will either display a form, a report, or a form selection list.

The feedback from the nurses and physicians was that the functionality was acceptable, but the application was missing form factor support. Most of the users of the application used mobile devices to access the application, and the tabbed navigation view did not scale with the form factor of the device. Since the form elements were also generic HTML elements, they caused a poor user experience on a mobile device.

QuickForms 1.0

PAL-IS 3 was implemented using QuickForms 1.0. This implementation solved the pressing issue of form factor support, and poor user experience. The nurses and physicians quickly became accustomed to the new look and feel, and enjoyed the tactile approach.

QuickForms 3.0

The PAL-IS 1.0 application was successful for the use of Bruyère Hospital, but the physicians realized that it did not have the usability and reporting support they wanted (due to limitations of traditional apps). PAL-IS 3 was evaluated and approved by Bruyère. However, the delivery of palliative care is evolving quickly in the Ottawa region, and they are now expanding and integrating their health care delivery with the Community Care Access Centers of Ontario and are required to use their information systems. PAL-IS 3 has served as a reference for articulating the extensions to CCAC's systems that are required to support palliative care. Meanwhile, the Local Health Integration Network (LHIN) for the greater Ottawa region has formed a Palliative Care Standards and Indicators committee that will monitor all palliative care across the region. We have adapted PAL-IS 3 to create The Standards and Indicators Dashboards (SAID) as a prototype of how they can monitor the delivery of palliative care across all health sectors in the greater Ottawa region. The application manages the referrals of palliative care patients to a facility, measures the number of these patients that have visited the emergency room, and the number of nurses and physicians that have had palliative care training. The reports on this data measure the trends of ER visits over time to find whether or not the palliative care program is successful.

5.3.3 Summary of Experience and Results

QuickForms was a good fit for PAL-IS and SAID due to the simple nature of the forms, and the table based reports that were integrated into the application. The application framework succeeded with simplifying the approach to form development, and maintained the desired element of forms directly linked to reports. PAL-IS provided

good feedback not only to the nurses and physicians, but also to the development team. The problems experienced with each iteration of the application drove innovation for the QuickForms application framework. This led not only to a polished palliative care monitoring application, but also a polished BI application framework. This case study was a trial for Bruyère Hospital in Ottawa to see if it can provide valuable feedback about its palliative care program. The success of the PAL-IS application lead to the roll out of SAID application for the Champlain LHIN district. Although the SAID application is still in early development, its potential value is highly anticipated by physicians and nurses in the Champlain LHIN district.

5.4. QuickForms User Community

QuickForms 3.0 was published as an open source project during the later stages of its development. Several University of Ottawa students became interested in QuickForms, and conducted case studies using the framework. Originally, RPP and PAL-IS were the main case studies developed by these students. The number of students who were interested in QuickForms increased, and they began creating their own case study applications. A community developed around the application framework which is still active today.

In addition, the community has developed a tutorial website to guide new QuickForms developers through the startup process to develop a sample application. Two assignments and a project are provided on the website to challenge new developers to learn the framework, and build an application from scratch. The website is available at this URL: https://code.google.com/p/QuickForms3/ .

In total, there are fourteen members of the QuickForms community who have developed BI applications and built a QuickForms course (with online tutorials, assignments, project, and sample application to help new developers learn the framework). The members of the community are all students from diverse backgrounds and programs (nursing, science, computer science, electrical engineering, MBA, and electronic business). They have varying degrees of experience from novice web developer (no programming experience) to expert (more than 5 years experience building web applications).

Many different applications were created using the QuickForms application framework since its development. These applications were smaller in size, and were often only used for a short period. Each application showcases unique elements of QuickForms, and provides insight as to how the framework can be improved.

5.4.1 Open Source Project

All of the source code for the QuickForms application framework is publicized on the user community website as seen in Figure 38. Many of the community members have taken the application framework code, and made minor modifications to fix bugs, or to make it more generic. These changes have been committed back to the repository, which makes this project a continuously evolving and polished product.

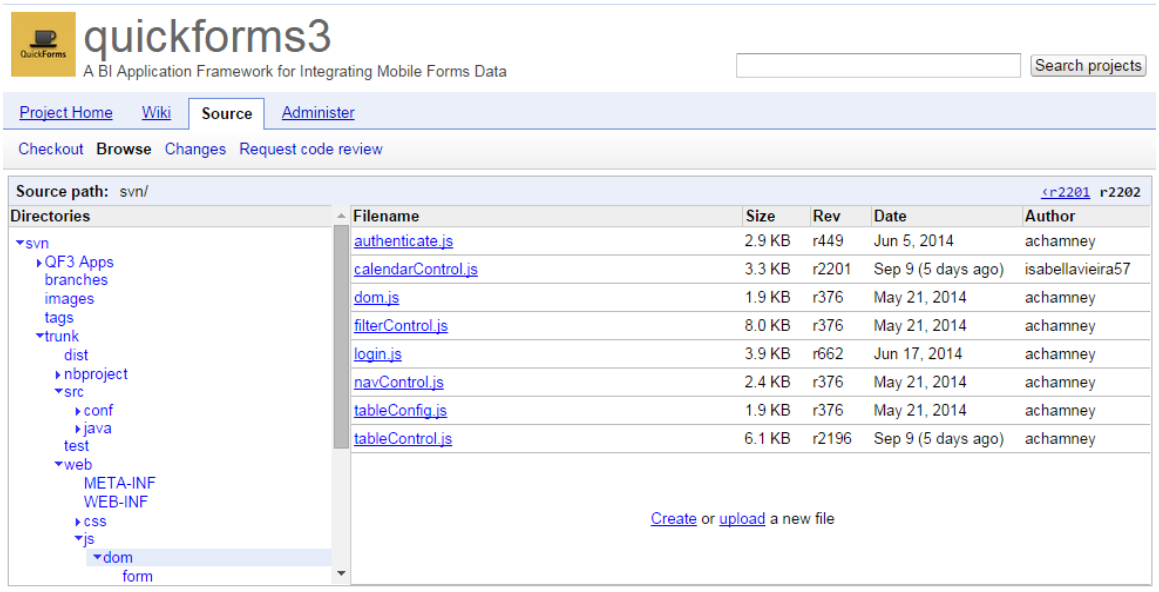


Figure 38: QuickForms open source SVN source browser.

Most of the applications that have been developed using QuickForms have also been uploaded to this repository. These applications are useful as a base to start other projects. The entire source code of a similar QuickForms community app can be downloaded from the apps repository, and modified to become a new application. This app repository also shows all of the different capabilities of QuickForms, and how it can suit the needs of many different BI application needs. Developers can pull ideas from the different applications as well to make a better interface for their project.

5.4.2 Tutorials, Libraries, and Templates

The QuickForms user community built a learning tool to help new developers get started with the BI application framework. Tutorials are provided, as seen in Figure 39, to get the new developer started, and to teach them the basics about business intelligence, and basic application structure.

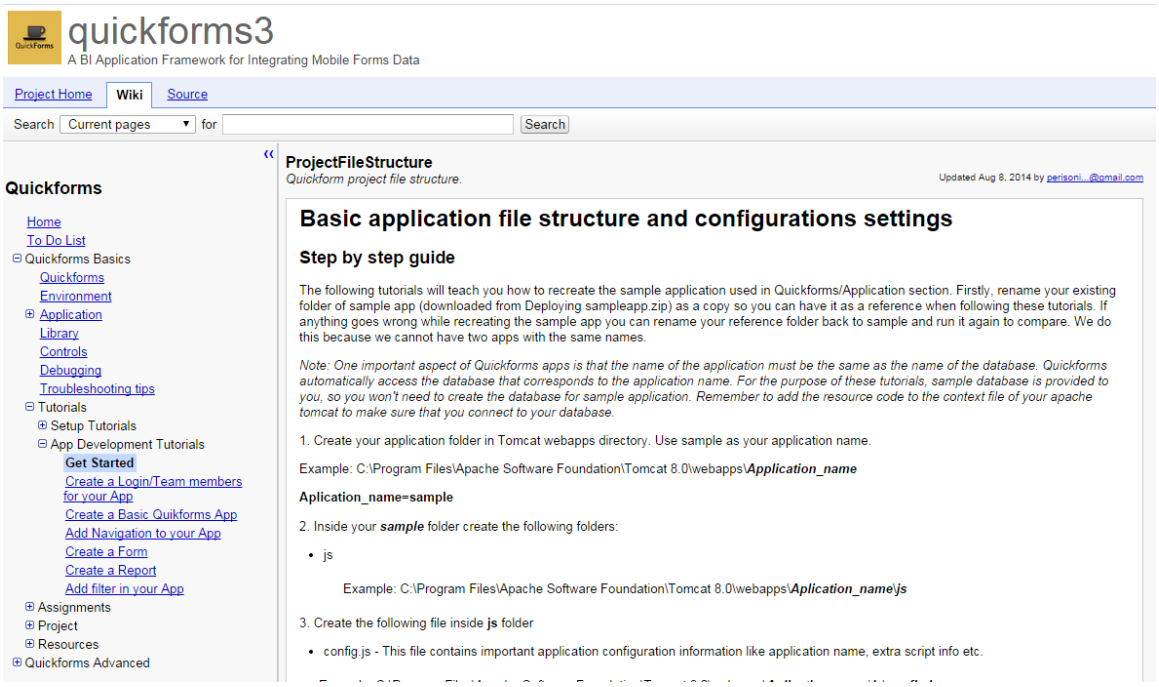


Figure 39: QuickForms user community tutorial hub. Getting started tutorial displayed on the right.

These tutorials provide basic knowledge about the libraries that are included in QuickForms, and give templates that are to be used with these controls. For example, the navigation tutorial explains the navControl library. The different navigation options are described, and an HTML template is provided that can be transformed to fit the developer's needs. Forms, reports, and filters are also available under this category.

5.4.3 Applications

The two largest projects that the QuickForms user community developed have been previously discussed in section 5.2 and 5.3. There are however nine smaller applications that have been developed by the community. These applications are detailed below, along with the number of developers involved with the project, and the time required to make a finished product. A summary of the QuickForms 3.0 user community applications is listed below in Table 5.

Table 5: Summary of the QuickForms user community applications

Project	Number of Developers Involved	Total Development Time	Lines of Code	Number of Forms and Sub-Forms	Number of Reports
RPP	3	Four Months	1273	5	19
SAID	1	Four Months	1415	3	9
SegCourse	2	Two Months	602	4	1
Pregnancy Coach	2	Three Months	1212	40	0
CHAPS	2	Two Months	1485	4	6
Patient Flow	1	Two Months	1212	28	1
Cardiac	2	Two Weeks	1369	6	4
Blogging Application	1	One Month	860	1	1
Sample Application	3	Three to Four Weeks	466	2	3
Hospital Application	2	Three to Four Weeks	483	2	3
Sec Charge	1	One Month	657	2	1

This table shows that there is a correlation between the lines of code, and the number of forms and reports. Applications that deviate from this correlation are a sign that there were controls needed for the application that QuickForms did not support. An example of this is the Blogging Application. Even though the entire application consists of one form and one report, there are almost a thousand lines of code. This is due to the lack of support for blog views. Presumably the lines of code would be reduced, if effort was made to packaging the custom code into reusable controls that were part of the QuickForms Library.

The Sec Charge application is still in development. The values displayed in the table above are the current progress of the application. We estimate that the application will grow significantly due to the integration of the Google maps service.

SegCourse

The Software Engineering Course Selection Application (SegCourse) is a tool provided to students to allow them to pick their desired courses. They are displayed with a list of available courses as seen in Figure 40, and links to literature about the respective programs and professors. This data is compiled into a report showing how many students wish to take different courses. This is used as an indicator for the popularity of the courses. This data can be used by the University of Ottawa to help decide which courses require more professors.

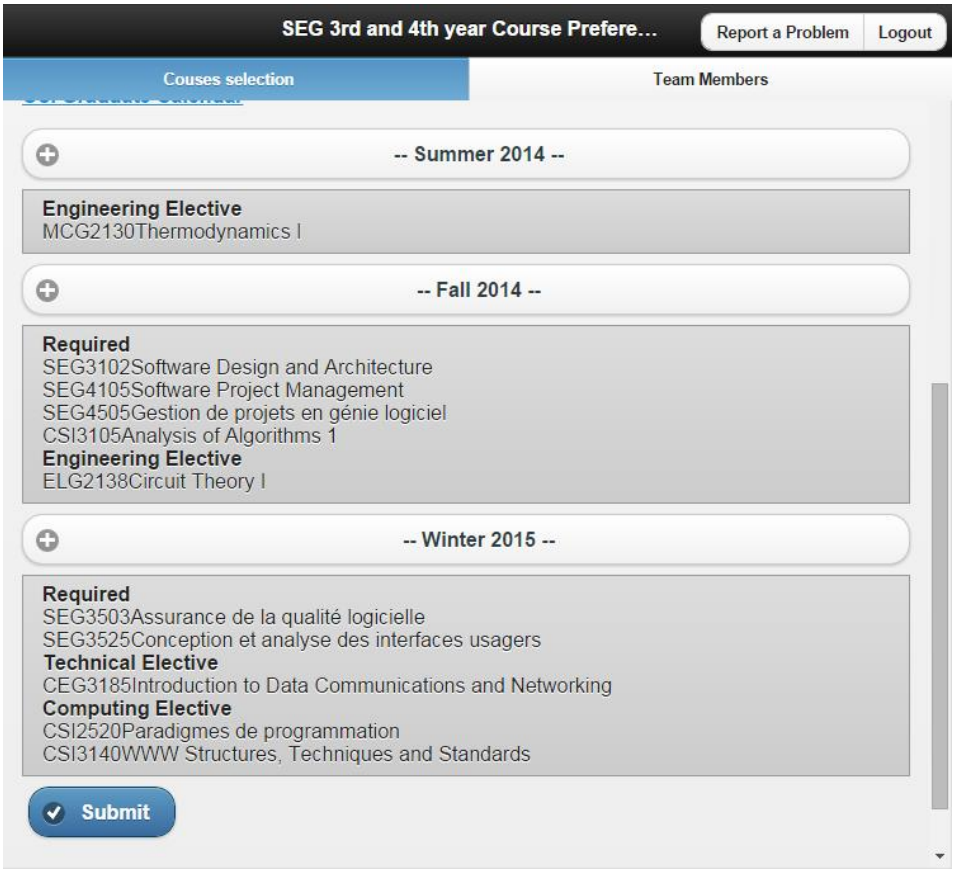


Figure 40: Course select application user interface

This application is significant because it has the highest variability of lookup elements. The information about the courses, students, and programs change at least every four months. All of the information in the lookup tables of the application must change every time the software engineering program is altered. This can be accomplished simply with the administrative view of the application. Excel spreadsheets of the courses and student are made available in the administrator console for download. Modifications of these spreadsheets are made, and can once again be uploaded. These changes are instantly reflected in the application.

The SegCourse application also showcased the benefit of reusable modules and controls. The multi-select course menu reuses the same JavaScript Client control that the RPP application used. The only difference is the configuration of the multi-select tabs, and the connecting reporting database. This sub-form can be seen below in Figure 41.

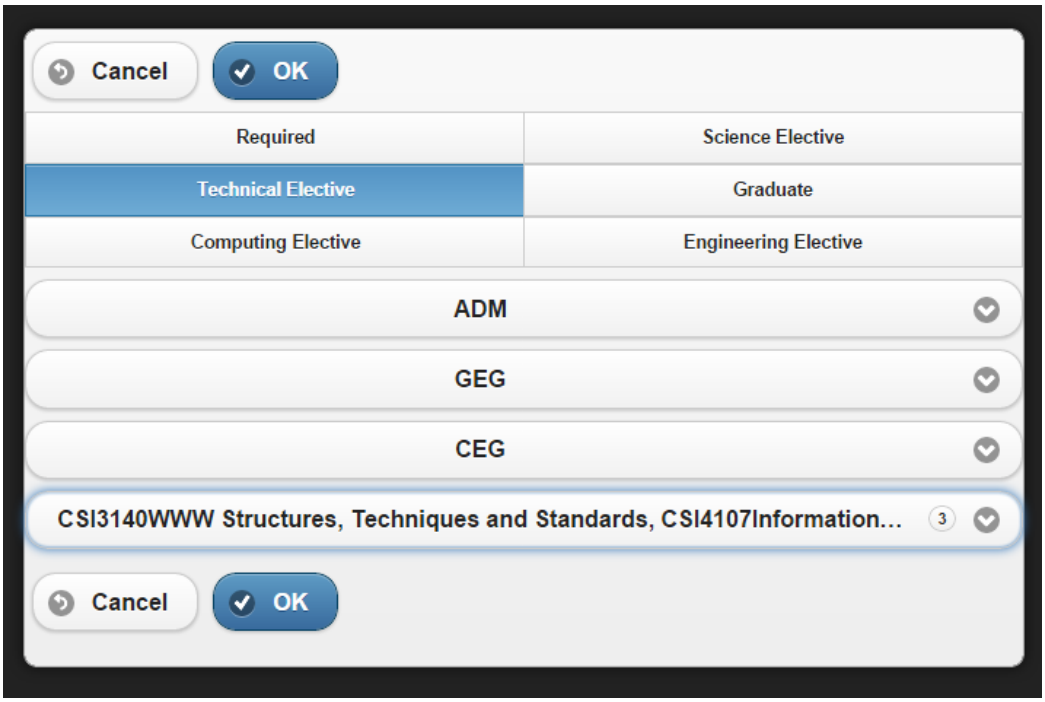


Figure 41: SegCourse winter course selection sub-form.

Pregnancy Coach

The purpose of the pregnancy coach application was to inform special populations about the different stages of pregnancy. The pregnant user subscribes to the application, and inputs their expected due date. The application will navigate them to literature to help guide the user through their pregnancy seen below in Figure 42, and provide vital information about the process.

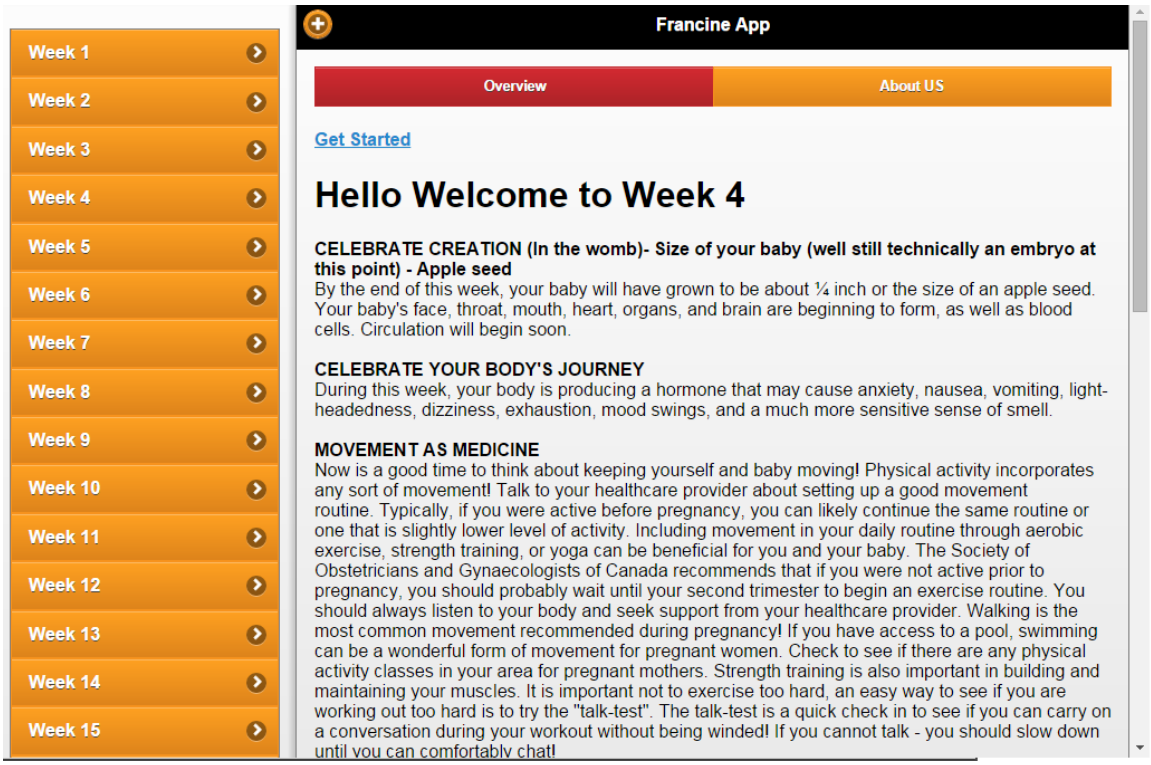


Figure 42: Pregnancy coach application dashboard.

The pregnancy application introduced a reusable HTML editor control for rich text entry. The pregnancy literature of the webpage is entirely dynamic, and can be changed using the administrator portal. The administrator uses an advanced HTML editor, which allows for editing of hyperlinks, images, and a multitude of font styling

capabilities. Once the HTML is submitted to the server, it becomes instantly available to the users of the application. This application is significant as it shows how two different views of an application based on user role can be integrated into the same interface. The same HTML application configuration code is used for every page on the application. The only difference is the method of authentication for the user. This is important because it cuts the effort required to maintain the application in half.

CHAPS

The Cardiovascular Health Awareness Program (CHAP) is an application designed to promote healthy and active living to reduce the risk of high blood pressure and heart disease. The application requests measurements of an individual to produce a numeric score to determine the risk of heart disease. The questionnaire that is used to determine this score can be seen in Figure 43.

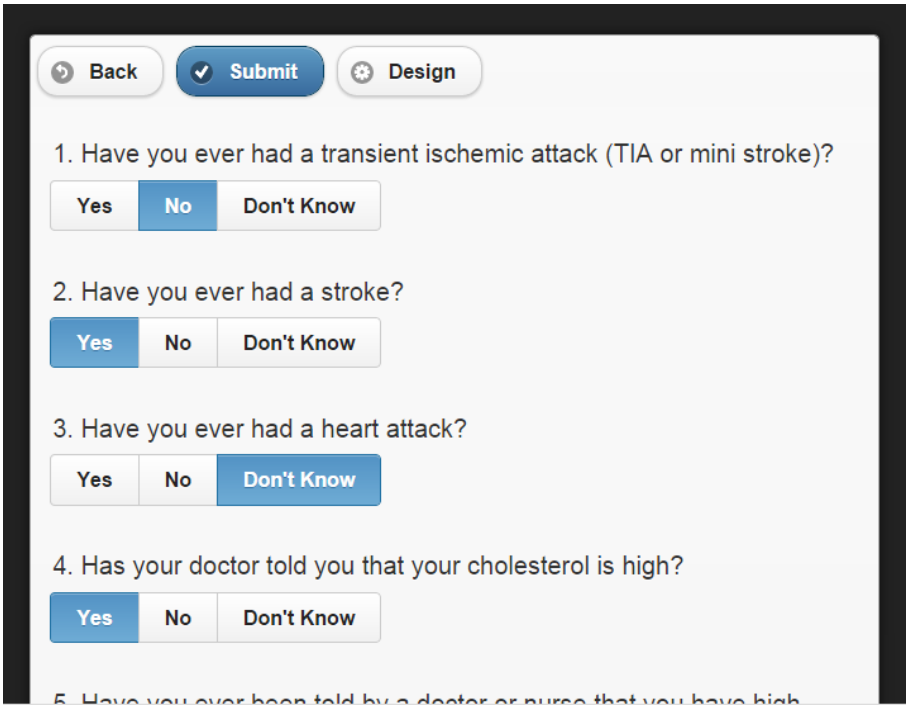


Figure 43: The CHAPS assessment questionnaire form.

CHAPS uses a JavaScript rule engine, but work is under way to replace with a QuickForms rule engine. This application was developed by two community members in a period of two months.

Patient Flow

This application was developed as a test driver/demo to simulate events from a business process management server and real-time location server as part of a project to monitor wait times in hospital emergency rooms by instrumenting them with sophisticated BPM and RTLS technology. It mimics a patient as they move through the hospital during a visit. The user picks from a list of possible patient events, all of which have associated forms. The list of possible events is displayed below in Figure 44.

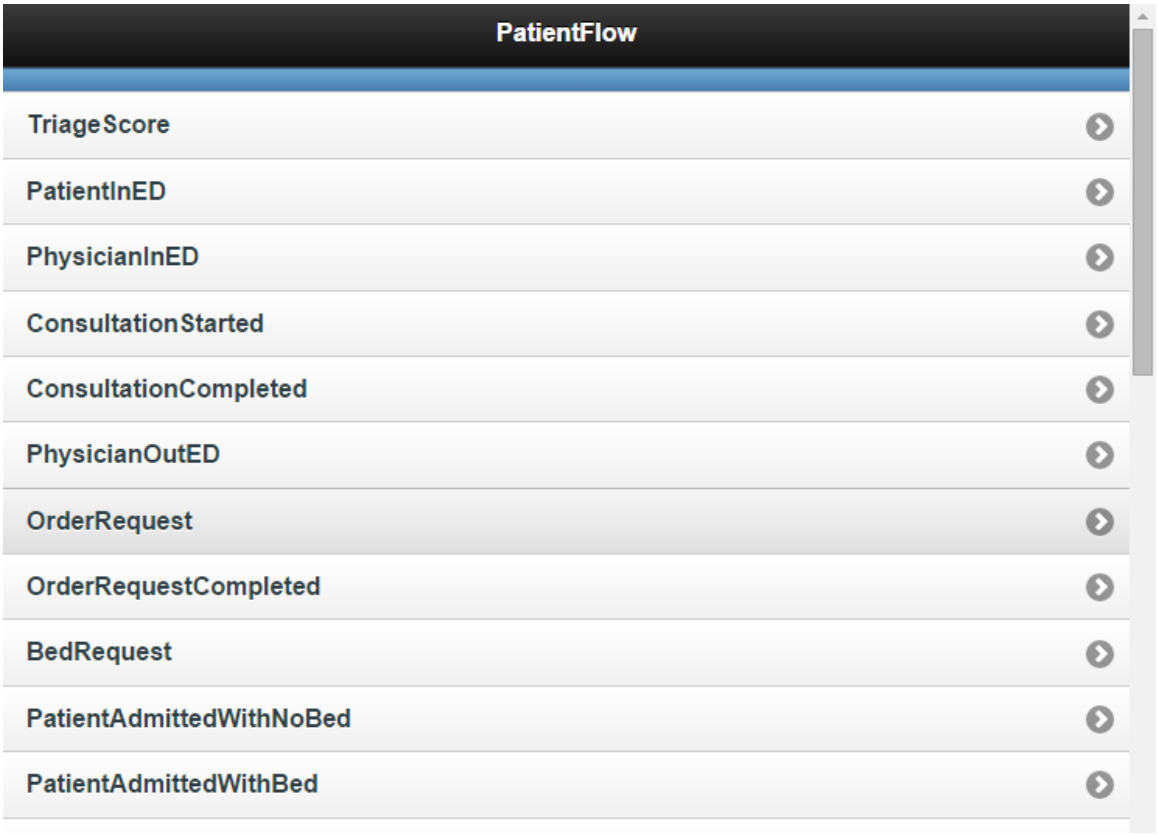


Figure 44: The list of possible event simulations in the Patient Flow application.

It uses a custom made JavaScript rule engine to determine the next path in the process for the patient. This application was developed by one community member in a period of four months.

Cardiac

The cardiac application is similar to the Patient Flow application except its purpose is to use a standalone QuickForms application to monitor wait times rather than requiring expensive and sophisticated BPM and RTLS instrumentation. It also uses an experimental QuickForms rule processing engine instead of a JavaScript rule processer. This change allowed for an alerts dashboard, along with the standard patient flow forms. This application was developed by two community members in a period of two months.

Blogging Application

As a side project, a blogging application was developed to test how QuickForms performs outside of its core competency of producing BI applications. The blogging application allows an administrator to post images, text, hyperlinks, and other HTML elements to the blog page. The blog page has customized CSS styling, and omits the jQuery mobile library as seen in Figure 45.

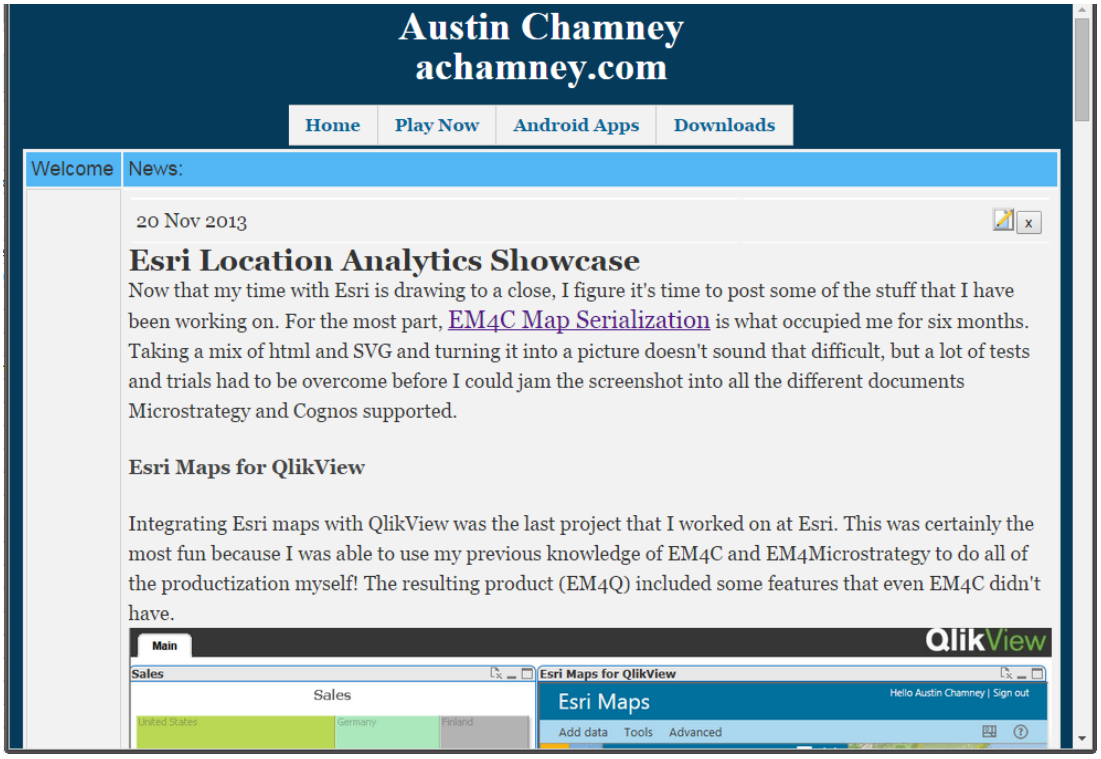


Figure 45: Blogging application home page displaying most recent post by the blogger.

The main blog page displays the blog posts in a unique fashion that is not included in the QuickForms library. A customized JavaScript control was developed to parse the blog data and insert it into the view. Apart from this single customized control, QuickForms fit seamlessly into the different variety of application. We will realize that more and more applications fit the BI framework model: Logging of events (a blog post), and summarizing the data for consumption (the blog view).

Sample Application

The Sample application is designed to aid new QuickForms developers during their learning process. All of the QuickForms user community tutorials are based off of the forms, reports and navigation of this app. It is available for download to start as a

basis for new developers. This application was developed by three community members in a period of two months.

Hospital Application

The hospital application is a published solution to the project that is part of the QuickForms user community tutorial. It is designed to be completed after the assignments, and to give a breadth of the capabilities available by QuickForms. The application mimics some of the functionality that is implemented in the PAL-IS case study, but cuts down much of the complexity. The patient entry for the hospital application is shown in Figure 46 below.

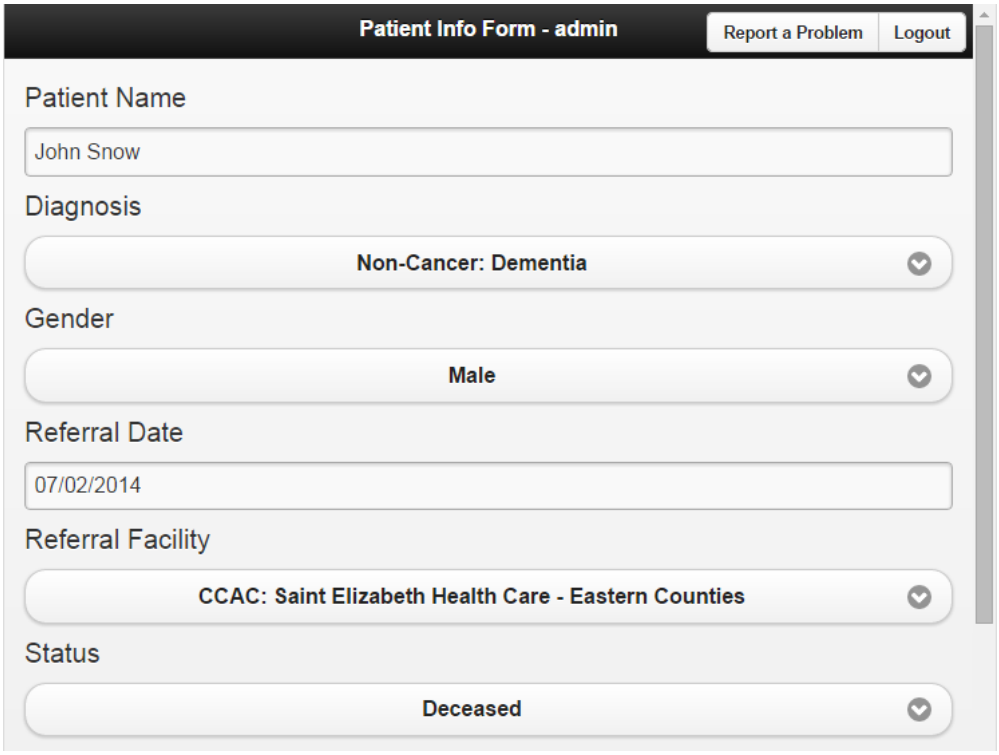


Figure 46: Hospital application patient entry page.

This application was developed and refined by two community member over a period of one month.

Sec Charge

This application is designed to aid electric car users on long journeys. The user enters in their location and their destination, and the application will find all of the charging stations along the route. The user can then reserve the charging station to ensure fast access. This application requires Google maps API integration to the QuickForms service. Sec charge is still in development by one community member. Its proof of concept design phase is planned to be completed by late winter 2014.

Chapter 6. Evaluation

In this chapter, we evaluate our application framework versus other approaches, with a particular focus on applying our evaluation criteria from Chapter 3 to the case studies that are described in Chapter 5. In section 6.1, we compare and evaluate the different application frameworks used in our case studies, including the different versions of QuickForms. In section 6.2 we compare QuickForms 3.0 to the related works identified in Chapter 2 based on our understanding of them from the literature survey and our experiences in our case studies. Chapter 6.3 evaluates QuickForms 3.0 based on the experiences of the QuickForms user community. A comparison is made in Chapter 6.4 between the RPP application developed in our case study and other practice profile applications. Finally in Chapter 6.5, we evaluate different AJAX clients, and how they compare to the QuickForms 3.0 JavaScript client.

6.1. Application Framework Evolution

In this section, we look at the two main applications developed in our case studies and compare the different versions of application frameworks used to implement them, in order to trace the evolution of our application framework in its ability to address the evaluation criteria listed in Chapter 3.

6.1.1 RPP

We compare our experience building RPP using three different frameworks in Table 6. We can see that QuickForms 1.0 achieved our main user requirements of

optimizing and linking user experience and reporting, while QuickForms 3.0 greatly reduced development effort and enhanced configurability.

Complexity

Table 6: Evaluation of complexity for different frameworks used to implement RPP.

Criteria	RPP in J2EE	RPP in QuickForms 1.0	RPP in QuickForms 3.0
Lines of Code	3154	4551	1273
Complexity of Database	ORM and simple reporting database.	Reporting database with multi-dimensional star schema.	Templated reporting database with multi-dimensional star schema.
Complexity of Database Transformation	ETL required between ORM and data mart.	Some ETL required.	No ETL.
Complexity of Application Tier Code	Complex. Difficult to maintain.	Complex JavaScript.	No application tier code.

RPP implemented in J2EE used an Object Relational Model (ORM) which uses a snowflake schema to model the transactional database. This model was not appropriate for reporting, so ETL was required to stage the data in a reporting database. QuickForms 1.0 and 3.0 simplify the traditional approach by using a star schema model for its transactional database. RPP implemented in QuickForms 1.0 did not use hierarchies to store diagnosis data, which increased the number of tables by an order of magnitude compared to both the J2EE and QuickForms 3.0 implementation. With QuickForms 3.0, the schema was standardized and simplified with pre-defined templates, resulting in a minimal number of joins, and tables, and the effort required to develop and maintain.

RPP created in J2EE had most of the business logic in the application tier. This included REST services, data access objects, and drop down population. The complexity of this code grew quickly. QuickForms 1.0 removed all the work from the application tier, and placed it instead in the controller layer of the JavaScript client. In effect, this

pushed the bounds of the application tier to the view, mixing the tiers of the application and causing more complexity. QuickForms 3.0 encapsulated the REST services, data access objects, authentication and authorization, and drop down population to a single tier. This removed the need for any application tier code which greatly reduced complexity.

Usability

Table 7: Evaluation of usability for different frameworks used to implement RPP.

Criteria	RPP in J2EE	RPP in QuickForms 1.0	RPP in QuickForms 3.0
User Interface (forms)	Poor (Generic).	Optimized Tabbed diagnosis control.	Optimized Tabbed diagnosis control.
User Interface (Reports)	Poor. Generic tables.	Moderate. No filtration capabilities.	Optimized. Filtration, charts, graphs, tables.
Real Time Reporting	No, ETL required.	Yes.	Yes.
Technology Used to Create BI App	Hard coded forms, hard coded reports.	Templates for forms, hard coded JavaScript.	Templates for forms and reports, no hard coding.

RPP created in J2EE was designed with the classic look and feel of HTML. The standard HTML form elements handled poorly on a mobile device, and struggled to display the full diagnosis data. QuickForms 1.0 and 3.0 implemented the tabbed diagnosis control that was discussed in the Think Aloud sessions with the residents, and greatly improved the application performance. The reports user interface improved further with QuickForms 3.0 due to the built in filtration control, chart form flexibility, and a details table listed below every graph.

Since response time is important for most BI applications, the wait between form entry and reporting is unacceptable. Due to the issue discussed in the database complexity

criteria, J2EE was not capable of reporting in real time. QuickForms 1.0 and 3.0 have an instantaneous reaction time between form entry and publishing the data on a BI portal.

With RPP created in J2EE, we used standard HTML forms and HTML reports. The forms displayed all of the drop downs with hard coded values, and the reporting portal used hard coded reports. QuickForms 1.0 improved upon this by generating the drop downs in the form rather than hard coding the values. This way, any changes in the database will be reflected in the view instantly. This implementation fell short with reporting capabilities, as a static query was still used to generate reports. QuickForms 3.0 introduced a filtration control that allowed for dynamic changes to the query based on user input.

Development Effort

Table 8: Evaluation of development effort for different frameworks used to implement RPP.

Criteria	RPP in J2EE	RPP in QuickForms 1.0	RPP in QuickForms 3.0
Effort and Skill required to Create BI App	Full stack development required.	Qualified web developer and database architect.	Database architect required.
Effort and Skill required to Maintain BI App	Difficult to maintain.	Difficult to maintain JavaScript.	Simple to maintain, very modular.
Effort and Skill required to Create Team Members Portal	Requires full set of application tier development.	Requires front end and back end development.	Provided with framework.
Effort and Skill required to Create Reporting Portal	Requires full set of application tier development.	Requires front end and back end development.	Templated charts and graphs, but complex SQL required.

In QuickForms 3.0, RPP was assembled by scripting and configuration of reusable-controls, simplifying its development process. The availability of templates and examples simplified coding even further. Fields are linked to tables by declaration in the

HTML. In QuickForms 1.0, RPP had a significant amount of business logic located in the JavaScript code. A strong developer was required to create and maintain this code. The J2EE implementation of RPP required full development of all tiers of the application, taking a great amount of effort and skill.

Creating a reporting dashboard in J2EE for RPP was particularly difficult. An extra JavaScript module had to be developed to transform the data from the application tier to a third party JavaScript chart API. In completion, the charts did not fully conform to the requirements of RPP. QuickForms 1.0 introduced a report dashboard control, which simplified the development process, but the users of the application still expected more functionality. RPP developed in QuickForms 3.0 used the new filtration control to enhance the reporting portal experience. Despite the addition of functionality to this version, the encapsulated controls simplified the development of the RPP reporting portal.

It could be argued that these criteria are subjective, and have their own personal biases associated with them. We attempted to remove these biases by involving a high number of developers with several different backgrounds. These developers unanimously agreed that QuickForms provided a framework which decreases the effort and skill required to build the application. This will be explained in more detail in section 6.3.

BI Application Features

Table 9: Evaluation of BI application capabilities for different frameworks used to implement RPP.

Criteria	RPP in J2EE	RPP in QuickForms 1.0	RPP in QuickForms 3.0
Reports Linked to Form Dropdowns	No.	Yes.	Yes.
Application	None	Lookup Tables.	Lookup Tables.

Configurability by Administrators			HTML
Device and Form Factor Support	PC browser optimized.	All form factors Smartphone to PC.	All form factors Smartphone to PC.

The drop down elements were hard coded for RPP in J2EE. Any change of labels or attributes in the database would cause a difference between the reports and forms. RPP implemented in QuickForms 1.0 and 3.0 used the same lookup values in the forms and reports. This was particularly useful for RPP because the administrators of the application managed the dropdown lists from an administrator console.

RPP implemented in J2EE did not provide any configurability for administrators. This did not change with the implementation of RPP in QuickForms 1.0. Finally, QuickForms 3.0 provided administrators with full control over drop downs, and team members.

The static HTML pages of RPP implemented in J2EE did not support multiple form factors and devices. As mentioned in User Interface, RPP implemented QuickForms 1.0 and 3.0 automatically provide a mobile ready user interface, which is capable of supporting tablets, mobile phones, and many other form factors.

6.1.2 PAL-IS / SAID

We evaluate our experience building PAL-IS and SAID using four different frameworks in Table 10. We can see that QuickForms 0.1 achieved our main user requirements of optimizing and linking the form elements to the reports, while QuickForms 1.0 greatly reduced development effort.

The SAID application which was an adaptation of PAL-IS is a vast reduction in complexity and development effort through its re-engineering with QuickForms 3.0.

Complexity

Table 10: Evaluation of the complexity for different frameworks used to implement PAL-IS.

Criteria	PAL-IS in .Net	PAL-IS in QuickForms 0.1	PAL-IS in QuickForms 1.0	SAID in QuickForms 3.0
Lines of Code	44,334	7941	3699	1415
Complexity of Database	ORM + data mart schema design.	Reporting database schema design, and form design	Reporting database schema design.	Templated reporting database design.
Complexity of Database Transformation	ETL required between ORM and data mart.	Significant ETL in that the user interface was generated from database using a complex meta data approach.	Some views needed to match BI tools.	No ETL required.
Complexity of Application Tier Code	Mixed layers, no encapsulation.	Complex JavaScript.	Complex JavaScript.	Minimal application tier code.

Usability

Table 11: Evaluation of the usability for different frameworks used to implement PAL-IS

Criteria	PAL-IS in .Net	PAL-IS in QuickForms 0.1	PAL-IS in QuickForms 1.0	SAID in QuickForms 3.0
User Interface (forms)	Too complex, busy interface.	Simple interface optimized for desktop.	Optimized Multi Form Factor.	Optimized Multi Form Factor.
User Interface (reports)	Did not capture physician's needs	Useful reports, but generic interface.	Useful reports, but generic interface.	Optimized charts and graphs.
Real Time Reporting	No. ETL required.	Yes.	Yes.	Yes.
Technology Used to Create	Hard coded forms, hard coded	Forms generated from	Some templates for HTML. Some	Templates for HTML, no hard

BI App	reports.	database. Hard coded reports	hard coding.	coding.

Development Effort

Table 12: Evaluation of the development effort needed for different frameworks used to implement PAL-IS

Criteria	PAL-IS in .Net	PAL-IS in QuickForms 0.1	PAL-IS in QuickForms 1.0	SAID in QuickForms 3.0
Effort and Skill required to Create BI App	Full stack development required Including database architect.	Qualified web developer and database architect.	Qualified web developer and database architect.	Simple HTML, JavaScript configuration. DBA for database.
Effort and Skill required to Maintain BI App	Difficult to maintain.	Difficult to maintain JavaScript and form database	Difficult to maintain JavaScript.	Easy to maintain.
Effort and Skill required to Create Team Members Portal	Requires full set of application tier development.	Requires front end and back end development.	Requires front end and back end development.	Minimal.
Effort and Skill required to Create Reporting Portal	Requires full set of application tier development.	Requires front end and back end development.	Requires front end and back end development.	Requires complex SQL.

BI Application Features

Table 13: Evaluation of the BI application capabilities for different frameworks used to implement PAL-IS

Criteria	PAL-IS in .Net	PAL-IS in QuickForms 0.1	PAL-IS in QuickForms 1.0	SAID in QuickForms 3.0
Reports Linked to Form Dropdowns	No.	Yes.	Yes.	Yes.
Application Configurability by Administrators	None	Lookup Tables.	Lookup Tables.	In app lookup tables, user roles.

Device and Form Factor Support	PC browser optimized.	PC browser optimized.	All form factors Smartphone to PC.	All form factors Smartphone to PC.

The evolution of the PAL-IS application through its different implementations directly mimic the RPP implementation evolution. Both began with a robust and complex web application framework, and slowly evolved to a QuickForms 3.0 implementation. The reduction of complexity and the addition of usability are identical in both cases. For this reason, further discussion about the individual evaluation criteria has been omitted.

6.2. Gap Analysis Evaluation

Now we take the knowledge learned from the case study framework evaluation and compare QuickForms 3.0 to the related works that were identified in our gap analysis described in Chapter 3. This comparison is based on our experience from our case studies, our understanding from the literature survey and discussions with our domain experts.

Complexity

Table 14: Analysis of complexity evaluation criteria for different BI application technologies.

Criteria	Traditional BI	Web Application Framework	AJAX Client Framework	QuickForms 3.0
Complexity of Database	High. Requires full operational systems, standardized enterprise data warehouse, and complex ETL.	Moderate. Automatic creation of ORM. Lookup table support is ad hoc.	Unknown.	Low. Templates for reporting database with full multi-dimensional lookup table support.
Complexity of Database Transformation	High. Transformation from multiple transactional databases to data warehouse.	High. Potentially incompatible transformation between ORM and data mart.	Unknown.	None. No ETL required.

Complexity of Application Tier	Low for application reporting off data warehouse. But completely disconnected from operational applications where data is collected.	High. Requires fully customized development for business logic and ORM integration.	Very high. Requires fully customized development for business logic plus AJAX controls.	Low. No business logic required. Supplied DAO.

Traditional BI suffers heavily from high complexity. BI is often integrated with multiple operational systems, and merged into a single data warehouse. Web application frameworks mitigate some of the complexity due to its independence from the operational system. ORMs remove much of the complexity for transactional databases. Web application frameworks fall short however during the ETL process, and application tier development. Creating a separate reporting database and transforming data from the ORM is an arduous task. A customized application tier is required to support all of the required BI features.

QuickForms mitigates much of the database complexity due to database templates, and the merged functionality of the transactional database and reporting database. Application tier complexity is nearly negligible due to the preconfigured DAO along with the business logic being encapsulated in customizable JavaScript modules.

Usability

Table 15: Analysis of usability evaluation criteria for different BI application technologies.

Criteria	Traditional BI	Web Application Framework	AJAX Client Framework	QuickForms 3.0
User Interface (forms)	Generic UI.	Generic UI.	Multi-form factor support.	Multi-form factor support.
User Interface (reports)	Drillable, slice and diced, charts and	Generic HTML tables.	Charts and graphs.	Customized BI Charts and

	graphs.			graphs.
Real Time Reporting	Slow. Lengthy process from forms to reports	Moderate. Potential slow ETL schedule	Unknown	Fast. Instantaneous feedback.
Technology Used to Create BI App	Enterprise forms, BI reporting software.	Hard coded forms, hard coded reports.	Some built in functionality (text entry, date), some third party integration (charts,)	Templates for forms and reports, no hard coding.

Traditional BI has optimal usability for its reporting tools. It suffers however from a long wait time between data entry and report generation. Web application frameworks provide generic user interfaces that do not provide optimal performance. AJAX client frameworks provide many of the tools needed for BI applications, but fall short depending on their server side support. QuickForms provides a mobile optimized user interface that allows for fast data entry, and immediate report availability.

Development Effort

Table 16: Analysis of development effort for different BI application technologies.

Criteria	Traditional BI	Web Application Framework	AJAX Client Framework	QuickForms 3.0
Effort and Skill required to Create BI App	High. Difficult to create multi-tier system. Subject to data inconsistencies.	High. Difficult to create multi-tier system. High application logic complexity.	High. Large learning curve. Considerable maintenance required.	Low. HTML and database only. Low skill level needed.
Effort and Skill required to Maintain BI App	Difficult to propagate changes across multi-tier system.	High. Changes must be propagated across all tiers.	High. Application tier complexity limits maintainability.	Low. Good configurability. Forms and reports linked to database.
Effort and Skill required to Create Team Members Portal	N/A	High. LDAP integration or full development required.	Moderate support for authentication. Difficult to create team portal.	Low. Templated team members portal and authentication and authorization.
Effort and Skill required to	High. Complex BI reporting tools.	High. Development	High. Reporting tools are limited	Medium. Requires DBA

Create Reporting Portal	Database architect required.	required across all tiers.	or require 3rd party integration.	and template configuration.

Traditional BI requires colossal effort and skill to build and maintain every aspect of the BI system. This can be seen with the need for integration of operational data, along with data warehouse design and BI reporting tool configuration. Web application frameworks require a highly skilled development team to produce a BI application. A database architect, business logic developer, and user interface developer are required to manage all tiers of the application. AJAX client frameworks require a similar team of developers as a web application framework, but with an unknown database portion. QuickForms BI application development only requires basic knowledge of HTML for template configuration, along with a database administrator to apply the reporting database templates. The application of templates to both the reporting database and the user interface drastically lowers the skill and effort required to make a BI app.

BI Application Features

Table 17: Feature availability for different BI application technologies.

Criteria	Traditional BI	Web Application Framework	AJAX Client Framework	QuickForms 3.0
Reports Linked to Form Dropdowns	No. Separate database used.	No. Hard coded forms.	N/A. No direct access to database.	Yes.
Application Configurability by Administrators	No. Uses operational forms.	No. Changing application configuration requires redeployment.	No. Web developer required.	Yes. Lookup Tables, HTML.
Device and Form Factor Support	Potentially.	No. Generic user interface.	Yes.	Yes.

6.3. QuickForms User Community Assessment

The QuickForms user community has developed eleven different applications using QuickForms 3.0. To write this chapter, we contacted each developer and solicited their feedback on their experience using QuickForms in their projects and applications. This information is displayed in Table 18 below.

Table 18: Summary of QuickForms user community profile, and their app development.

Developer Background	Developer Experience	Application	Time spent using QuickForms	Was QuickForms a good fit for application	Application features not supported by QuickForms
Undergraduate Computer Science Brazilian exchange student.	Web development using PHP, HTML, Wordpress, Joomla, and MySQL.	RPP	Two months.	Yes.	No.
Computer Science Masters Student.	Java Courses (Beginner, Intermediate, Advance and Swing)	RPP	One year.	Yes.	No.
Computer Science Masters Student.	Developer-Intern, QA Specialist(web QA development)	PAL-IS	One month.	Yes.	No.
Computer Science Masters Student.	Basic HTML lab sessions	SAID	Three months.	Yes.	No.
Computer Science Masters Student.	Java and internet technologies.	Sample App	Three to four weeks.	Yes.	No.
E-Business Technologies Masters Student	HTML, CSS, JavaScript and jQuery.	Hospital App	Three to four weeks.	Yes.	No.
Computer Science Masters Student.	HTML, CSS, JavaScript, J2EE, .Net, PHP, jQuery.	Blogging Application	One Month	Yes.	Yes. (Blog list view)
E-Business Technologies Masters Student	HTML, CSS, JavaScript and jQuery.	Patient Flow	Seven months.	Yes.	Yes. (Integrated rule engine)
E-Business Technologies Masters Student	HTML, CSS, JavaScript and jQuery.	Pregnancy Coach	Three months.	Yes.	Yes. (Content provider)

E-Business Technologies Masters Student	None	CHAPS	Three months.	Yes.	Yes. (Heart disease calculator)
Software Engineering Undergraduate Student.	HTML, Ruby, jQuery, JavaScript, Python	SegCourse	Three months.	No.	No.
Computer Science Masters Student.	Java and internet technologies.	Sec Charge	Three months.	Yes.	Yes. (map integration)
Computer Science Masters Student.	HTML, CSS, JavaScript and jQuery.	Cardiac	Two weeks.	Yes.	Yes. (Integrated rule engine)

The consensus shows that the majority of the community developers believed that QuickForms was a good application development platform for their application. It was learned that some applications required features that were not supported by QuickForms. These features required customized development, which increased development effort considerably. Blogging application, Patient Flow, Pregnancy Coach, CHAPS, Sec Charge, and Cardiac experienced these problems. Sec Charge, Cardiac, Pregnancy Coach, and Blogging application do not fall under the BI application category, and may be better suited for other application frameworks. Patient Flow does follow the standard BI application requirements, and would have benefitted from a greater breadth of supported controls.

6.4. Practice Profile Application Comparison

RPP is a practice profile application that has similar high level functionality to other applications. For example, Just in Time (JIT) Medicine and LogMD are, at a high level, similar practice profile applications with small differences. All three are focused on reporting and ease-of use for data entry. The three application are evaluated in Table 19 below.

RPP benefitted from several BI application features supplied by QuickForms. It was designed for faster data entry, because it was important for a resident to log all patient encounters so they could self-assess their entire clinical experience. The built in support for the linking between forms and reports made for a direct and in general improved reporting experience.

Table 19: Evaluation of RPP versus JIT and LogMD

Criteria	JIT	LogMD	RPP
Purpose	Student assessment.	Physician self-assessment.	Resident self-assessment.
Device and Form Factor Support	All form factors.	All form factors.	All form factors.
Data Entry	2-5 minutes. Rigid.	<2 minutes.	<30 seconds.
Diagnosis Selection	One level.	Two levels (procedures).	Two levels (all).
Forms linked to reports?	No.	Yes.	Yes.
Templated Summaries	No.	No.	Yes.
Drill-through Reporting	No.	No.	Yes.
3rd Party Reporting	No. Hard-coded reports.	No. Hard-coded reports.	Yes. Dimensional model.
Application Configuration	None.	None.	Data tables, UI controls.

6.5. Ajax JavaScript Client Comparison

The JavaScript client was re-architected significantly between QuickForms 1.0 and 3.0. Several JavaScript clients were assessed for their front end capabilities (see chapter 2.3.3). In particular, QuickForms 3.0 used a hybrid of jQuery and Dojo components. Below in Table 20, we measure the benefits and tradeoffs of each framework.

Although QuickForms uses a unique combination of jQuery and parts of Dojo, it is clear that a combination of the two provides the optimal JavaScript client for the development of BI applications. QuickForms manages all of the third party integration that would normally be required by an app developed in jQuery, and the QuickForms modules encapsulate most of the complexity that would be seen in a Dojo application. These benefits reduce the skill and effort required to build a BI application, and produce an optimized user interface.

Table 20: JavaScript client comparison summary

Criteria	jQuery	Dojo	QuickForms 3.0
Device and Form Factor Support	Yes.	Partial support (not enough form controls).	Yes.
Learning Curve	Small.	High. Anonymous methods, vast module library.	Minimal. Only one function call required per HTML page.
Development Effort	Low for initial application development. Difficult app maintenance.	High for initial application development. Low application maintenance.	Low for both maintenance and creation.
Integrated Reporting Tools	No, 3rd party integration required.	Yes.	Yes.
Integrated BI Specific Features (Data filters, application administration)	No.	No.	Yes.

Chapter 7. Conclusions

The QuickForms 3.0 framework has the potential to greatly reduce the effort in building a BI application like RPP or PAL-IS / SAID. For both of these, it greatly improved the configurability and user experience for both data collection and reporting compared to traditional approaches.

The main contribution of this thesis is an application framework for developing BI apps that is much more effective than traditional ETL-oriented applications because it directly maps data-entry forms to reporting databases. It is also much easier to use than traditional web application frameworks because it reduces the problem of developing a BI application to one of largely selecting and configuring application templates and components. This application framework includes a multi-dimensional star schema reporting database for optimal reporting performance, and a generic DAO service that removes the need for all application tier coding allowing developers to focus on the all-important front end of the application.

The second contribution of this thesis is a published, referenced implementation of our BI Framework as the open source project QuickForms 3.0, which has an active user community and online tutorials and sample applications (Chamney, et al., 2014). QuickForms 3.0 also includes a set of 38 BI application specific controls and templates which automatically connect link forms and reports. This enables developers to optimize the user interface to provide a comfortable, intuitive, and multi-form factor experience.

This user community is continuing to improve and evolve the framework along with its controls, templates, tutorials and sample applications.

The final contributions of this thesis are two innovative BI applications for health care that were used to validate the QuickForms application framework as a powerful BI application development tool.

More work is needed of course before we can make any definitive claims about how well the framework would work for other developers and other applications. We can, however, make some observations on the types of applications it seems suited for, and the work still needed in the framework itself. QuickForms is directed at a very specific class of application: BI Applications. We believe that the industry and the research community have enough experience with star schema databases and mobile forms applications that our approach should be useful for most form-based monitoring and logging applications where data is collected for analysis and/or performance management.

In order to use QuickForms 3.0 the following assumptions need to be true:

- The storage needs for such applications are not compromised by restricting to a Star schema.
- No rich content or content management (Voice, Pictures, Attachments).
- Multi locale issues are ignored.
- Concurrency issues in which two or more users might edit the same form or edit the same lookup table at the same time are ignored. The nature of our applications so far allows us to adopt a “last in, wins” policy.

- Communication, service, and/or application integration issues. We have treated the application as a strictly standalone forms application.
- Security and fault tolerance issues are not addressed except for basic security, authentication and authorization as all our applications have so far been done as proof of concept prototypes that were piloted under controlled circumstances.

7.1. Future work

So far we have only tested out our framework on a few example applications from healthcare. We need more case studies and applications from other industry segments. Even in healthcare, we have only scratched the surface and need more case studies and applications. Ongoing work is needed to address all of the limiting assumptions listed above.

There has also been some preliminary work to add a rule engine to the QuickForms Service (Pregnancy, Cardiac, CHAPS), but more work is needed. Both in terms of the rule engine, rule language, and configuration support, but also in terms of its integration strategy. This will enable QuickForms applications to enact and monitor a business process, and enable applications to write customized server side code. This is useful for sending automated emails, and to provide advanced business logic.

Chapter 8. References

Alur, D., Malks, D., Crupi, J., Booch, G., & Fowler, M. (2003). *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies (2 ed.)* (2nd ed.). Mountain View, CA: Sun Microsystems, Inc.

Azarm, M., Peyton, L., & Nargesian, F. (2011). Tool Support and Data Management for Business Analytics Applications in Healthcare. *International Journal for Infonomics (IJI) 4(3)*, 484-493.

Baarah, A. (2014). *An Application Framework for Monitoring Care Processes.* Doctoral dissertation, University of Ottawa.

Baarah, A., & Peyton, L. (2012). Engineering a State Monitoring Service For Real-time Patient Flow Monitoring. *The 9th Middleware Doctoral Symposium of the 13th ACM/IFIP/USENIX International Middleware Conference*, 1-6.

Badreddin, O., Baarah, A., Chamney, A., Kuziemsky, C., & Peyton, L. (2014). *Analytics Driven Application Development for Healthcare.* Eseo, France: 7th International Conference on Health Informatics.

Booth, D. (2004, February 11). *Web Services Architecture*. Retrieved August 2014, from W3C: http://www.w3.org/TR/ws-arch/

Buschmann, F., Henney, K., & Schimdt, D. (2007). *Pattern Oriented Software Architecture* (Vol. 5). John Wiley & Sons.

Chamney, A., Mata, P., Viner, G., Archibald, D., & Peyton, L. (2014). Development of a Resident Practice Profile in a Business Intelligence Application Framework. *Procedia Computer Science, 37*, 266-273.

Chamney, A., Mehta, I., Valiya, R., Mata, P., Rao, P., Vieira, I., et al. (2014, September 14). *quickforms3*. Retrieved September 2014, from code.google.com: https://code.google.com/p/quickforms3/

Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and OLAP technology. *ACM Sigmod record, 26(1)*, 65-74.

Chéné, M., Peyton, L., & McGuire, K. (2010, August 10). *Patent No. 7,774,504.* U.S.

Chouffani, R. (2012, July 12). *10 mobile business intelligence apps for on-the-go analysis*. Retrieved 09 02, 2014, from Computerworld: http://www.computerworld.com/article/2474290/enterprise-applications/10-mobile-business-intelligence-apps-for-on-the-go-analysis.html

Chusho, T., & Fujiwara, K. (1998). wwHww: an application framework of distributed systems for enduser-initiative development. *In Software Engineering Conference. Proceedings. 1998 Asia Pacific*, 102-109.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the Web Services Web. *IEEE Internet computing, 6(2)*, 86–93.

De Giusti, M., Oviedo, N., & Lira, A. (2011). Extract, Transform and Load architecture for metadata collection. *In VI Simposio Internacional de Bibliotecas Digitales*.

Dojo. (2014, September). *Dojo*. Retrieved September 15, 2014, from Dojo Toolkit: http://dojotoolkit.org/

Ferenchick, G., & Solomon, D. (2013). Using cloud-based mobile technology for assessment of competencies among medical students. *PeerJ, 1*, e164.

Fielding, R. T., & Taylor, R. N. (2002). Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology (TOIT), 2(2)*, 115-150.

Fonteyn, M., Kuipers, B., & Grobe, S. (1993). A description of think aloud method and protocol analysis. *Qualitative Health Research, 3(4)*, 430-441.

Gangadharan, G., & Sundaravalli, S. (2004). Business intelligence systems: design and implementation strategies. *Information Technology Interfaces*, 139-144.

Gellersen, H., & Gaedke, M. (1999). Object-oriented Web application development. *Internet Computing, 3*(1), 60-68.

Google Inc. (2014, September). *Trends*. Retrieved September 25, 2014, from Google Trends: https://www.google.ca/trends/

Guerrero, J. (2013). HTML5, CSS3 and JQuery mobile for intelligent home control. *Brazilian symposium on Multimedia and the web*, 9-10.

Hevner, A. R., March, S. T., Park, J., & Pam, S. (2004). Design Science in Information Systems Research. *MIS Quarterly, 28(1)*, 75-105.

Iglar, K., Polsky, J., & Glazier, R. (2011). Using a Web-based system to monitor practice profiles in primary care residency training. *Canadian Family Physician*, 1030-1037.

Johnson, R. (2005). J2EE development frameworks. *Computer, 38(1)*, 107,110.

Jokela, T. (2006). Methods for quantitative usability requirements: a case study on the development of the user interface of a mobile phone. *Personal and Ubiquitous Computing, 10(6)*, 345-355.

jQuery Forum. (2011). *Jquery vs GWT*. Retrieved September 15, 2014, from forum.jquery.com: http://forum.jQuery.com/topic/jQuery-vs-gwt

Kafeza, E., Chiu, D., Cheung, S., & Kafeza, M. (2004). Alerts in mobile healthcare applications: requirements and pilot study. *Information Technology in Biomedicine, IEEE Transactions on, 8(2)*, 173,181.

Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling.* John Wiley & Sons.

Kimball, R., Ross, M., Thornthwaite, W., Mundy, J., & Becker, B. (2008). *The Data Warehouse Lifecycle Toolkit* (2 ed.). Indianapolis, Indiana, USA: Wiley Publishing Inc.

Kirakowski, J., & Corbett, M. (1993). SUMI: the Software Usability Measurement Inventory. *British journal of educational technology, 24(3)*, 210–212.

Leff, A., & Rayfield, J. (2001). Web-application development using the Model/View/Controller design pattern. *In Enterprise Distributed Object Computing Conference. EDOC'01. Proceedings. Fifth IEEE International*, 118,127.

Liya, W., Barash, G., & Bartolini, C. (2007). A Service-oriented Architecture for Business Intelligence. *In Service-Oriented Computing and Applications. SOCA'07. IEEE International Conference on*, 279-285.

LogMD. (2014). *LogMD*. Retrieved September 15, 2014, from LogMD: http://www.logmd.com/

Mata, P. (2014). Pilar's thesis on thesis stuff.

Mata, P., Kuziemsky, C., Singh, J., Baarah, A., & Peyton, L. (2014). Engineering a Performance Management System to Support Community Care Delivery. *International Conference on Software Engineering*.

Microsoft. (2014). *Overview of the .NET Framework*. Retrieved September 15, 2014, from Microsoft: http://msdn.microsoft.com/en-us/library/zw4w595w.aspx

Papadimitriou, C., & Yannakakis, M. (1997). On the complexity of database queries (extended abstract). *Sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 12-19.

Peyton, L., Zhan, B., & Stepien, B. (2008). A Case Study in Integrated Quality Assurance for Performance Management Systems. *In MSVVEIS*, 129-138.

Preuveneers, D., Yasar, A.-U.-H., & Berbers, Y. (2008). Architectural Styles for Opportunistic Mobile Communication: Requirements and Design Patterns. *In Proceedings of the International Conference on Mobile Technology, Applications, and Systems*, 44.

ProntoForms Corporation. (2014). *ProntoForms: mobile forms made easy*. Retrieved August 2014, from ProntoForms: https://www.prontoforms.com/

Rumbaugh, J. B. (1991). *Object-oriented modeling and design 199(1).* Englewood Cliffs, NJ: Prentice-hall.

Schmidt, D. C., Gokhale, A., & Natarajan, B. (2004). Leveraging Application Frameworks. *Queue, 2(5)*, 66.

Simitsis, A., Vassiliadis, P., & Sellis, T. (2005). Optimizing ETL processes in data warehouses. *In Data Engineering. ICDE 2005. Proceedings. 21st International Conference on*, 564 - 575.

Singh, I., Stearns, B., Johnson, M., & Team, T. E. (2002). *Designing Enterprise Applications with the J2EE Platform* (2nd ed.). Addison-Wesley Professional.

Squires, D., & Preece, J. (1999). Predicting quality in educational software: Evaluating for learning, usability and the synergy between them. *Interacting with computers, 11(5)*, 467-483.

Stack Overflow. (2013, December). *Which JavaScript framework (jQuery vs Dojo vs ...)*. Retrieved September 15, 2014, from Stack Overflow: http://web.archive.org/web/20131230072844/http://stackoverflow.com/questions/394601/which-javascript-framework-jquery-vs-dojo-vs

Stal, M. (2006). Using architectural patterns and blueprints for service-oriented architecture. *Software, IEEE, 23(2)*, 54-61.

Struts. (2014). *Struts*. Retrieved July 2014, from Apache Struts: http://struts.apache.org/

Tegarden, D., Sheetz, S., & Mona, D. (1995). A software complexity model of object-oriented systems. *Decision Support Systems, 13(3)*, 241-262.

Tegegne, A., & Peyton, L. (2013). Application framework support for process-oriented software development. *International Journal of Electronic Business, 10(3)*, 232 - 253.

Trkman, P., McCormack, K., Valadares de Oliveira, M. P., & Ladeira, M. B. (2010, June). The impact of business analytics on supply chain performance . *Decision Support Systems, 49*(3), 318-327.

Vawter, C., & Roman, E. (2001). "J2EE vs. Microsoft .NET." A comparison of Building XML-based web services. *The Middleware Company*, 1-37.

Weyuker, E. (1988). Evaluating software complexity measures. *Software Engineering, IEEE Transactions on, 14(9)*, 1357-1365.

Weyuker, E., & Avritzer, A. (2002). A metric to predict software scalability. *In Software Metrics. Proceedings. Eighth IEEE Symposium on*, 152,158.