2017

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

EXTENDING DATA VISUALIZATION IN SALESFORCE WITH REACT AND REDUX

Matthew W. Cook

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2017

Approved by

Advisory Committee

Dr. Toni Pence, Committee Member 1 _____

Dr. Thomas Janicki, Committee Member 2 _____

Dr. Doug Kline, Chair

_____

Accepted By

_____

Dean, Graduate School

ABSTRACT

Salesforce is a popular cloud based customer relationship management (CRM) platform. Its usefulness is extended in part due to providing subscribers the options to build and manage their environment. With minimal setup and configuration, a business can begin using Salesforce with little customization, or extend its capabilities extensively. Salesforce's data visualization techniques, however, are outdated and lacks features of common graphing libraries like exporting, drill-down and interactivity.

This project produced a build, client and server stack compatible with Salesforce development. The result of the project is a compiled file inside a distributable Salesforce managed package. The build stack is comprised of Webpack, Node.JS and JSForce that builds and zips a single file, and then uploads to Salesforce. The client stack utilizes React and Redux to manage the presentation and application state, and FusionCharts to control the chart and graph rendering. To communicate with the client, Salesforce REST API services are used to extract data from the database.

A basic usability study was conducted to assess the application. The feedback shows the application is generally useful, easy to learn and, with a few additional features, would be a viable replacement of Salesforce's native dashboard. One feature, in particular, which was mentioned several times by the study participants is the ability to customize the data feeding the graphs either by a data source other than Salesforce or custom data.

Table of Contents

CHAPTER 1: INTRODUCTION

The project documented in this paper explains a way to connect the increasingly popular Force.com platform and extend its data visualization capabilities using popular client libraries named React and Redux. Salesforce provides such immense value to those businesses using the customer relationship management (CRM) but visualizing information is not always enlightening. This project uses React and Redux to build a single page application that connects to Salesforce reports. The application is packaged using Salesforce's managed packages which allows for an easy distribution through Salesforce. This project uses the FusionCharts library to manage the client's data visualization and will let the user choose the report data to pass to the visualization library. The project provides an example of an end-to-end software solution on the Force.com platform that includes a front-end build stack, client stack, server stack and deployment mechanism for distributing software built in Salesforce.

CHAPTER 2: BACKGROUND / PROBLEM DESCRIPTION

Background on Live Oak

Live Oak Bank (LOB) is a financial technology banking institution headquartered in Wilmington, North Carolina with a goal of "Empowering the American dream of small business owners". Live Oak was started by a small team with Chip Mahan leading the effort in 2008. This team began offering loan services to businesses in the veterinarian industry by leveraging the Small Business Administration's (SBA) lending program. While Live Oak originally started with a paper process, the core team quickly realized its inefficiencies and needed a way to organize and expedite the loan management system. Brothers Peter and Neil Underwood researched

various platforms to host a dynamic and growing environment and determined Force.com was well sustainable. Live Oak was then able to operate more efficiently and offer products to help expedite the loan process by leveraging Salesforce and a banking operating system, and so founded nCino. nCino has since been spun out from the bank as a stand-alone company. Since 2008, the bank has strategically considered and incorporated additional industries into their loan offerings and now lends in the Agriculture, Family Entertainment Center, Funeral Home, Healthcare/Dental, Hotel, Insurance, Investment Advisory, Pharmacy, Self-Storage, Veterinary, Renewable Energy, and Wine & Craft Beverage verticals. Although Live Oak is a branchless bank and offers all its services online, it has several remote offices for employees throughout the country including Georgia, Washington DC, Texas, and California where loan officers can better serve those geographic areas. As of 2016, the bank now has over 475 employees, and went public on Nasdaq in 2015 under the ticker symbol LOB. (Live Oak Bank, 2017) (Nasdaq, 2015)

Live Oak Bank specializes in small business loans and leverages the Small Business Administration's government loan program to assist businesses in gaining the capital needed to: start a new business, acquire an existing business or even expand. Integrated into approximately a dozen different verticals, Live Oak has become a powerful player in small business lending. Live Oak doesn't just see itself as a lender but a partner to help businesses achieve the American dream. The bank works hard to be fast, transparent, provide open communication, convenient and with top-notch security. Each facet is important in the bank's growth and striving to maintain its primary goal, "to treat the customer as if they're the only customer".

The atmosphere at Live Oak is very casual and relaxed. The bank has built a campus in Wilmington, NC sporting two buildings coated in cedar and sitting next to a lake with many amenities offered to its employees. Building number one offers its employees a gym with two full time trainers while building two provides a new restaurant named "The Grove" and offers meals for breakfast, lunch and take home meals for dinner. Building number two also greets its visitors with a giant copper replica of the oak logo with lights illuminating the leaves. Walking through each of those buildings is like walking through a big treehouse.

I've had the opportunity to work for Live Oak bank for three and a half years as a Software Developer. My primary responsibilities are to help develop new and innovative banking solutions which fit both internal employee and external customer needs. While my project contributions have been broad, our customer portal has been a primary focus for a couple of years and serves as a centralized application platform to assist our customers with their loan, by making the process transparent and digital.

Background on Salesforce

Headquartered in San Francisco, California, Salesforce provides the world's largest cloud based Customer Relationship Management (CRM) software as a service (SaaS) product. A CRM enables a business, small or large, to connect with their customers and attempt to provide the best level of customer experience as possible. Salesforce started this venture in 1999 with Marc Benioff as one of the founders. Since that time, Mr. Benioff has pioneered the technological giant and helped build it into the business it is today. In 2004, the company went public at a strike price of $11.00 per share and raised $110 mm. By building an architectural eco-system that lives in the cloud and streamlining an environment conducive for customization and innovation, Salesforce used that opportunity to build itself into the influential technology

company it is. The price per share is now around $83 and still climbing. Reportedly, the company has not had a GAAP profit since its inception. As of 2016, the website capterra.com reported Salesforce having 150,000 customers and 7.2 million users. (Nadaq, 2006) (Hollar, 2016) (Wikipedia, 2017)

Salesforce offers a range of products including, of course, Salesforce which is its original and primary CRM product. Another is Force.com which provides developers the capability to customize Salesforce to their particular need. The highly flexible Platform as a Service (PaaS) incorporates a Java like language known as Apex for its server-side development language. It uses an object query language called Structured Object Query Language (SOQL) and Structured Object Search Language (SOSL) for all interactions to the database. With tables known as *Objects*, one can use SOQL and SOSL to obtain data in raw form from the database. The architecture of Force.com uses a scalable, multi-tenant environment helping Salesforce customers remove the headache of managing a server and allowing businesses and developers to focus on more important issues than the physical and logical design and maintenance of a server. (Salesforce, n.d.)

In 2011, Salesforce acquired *Rypple* who built an application which focuses on corporate growth and anonymous employee feedback. In that same year, the company implemented a strategy known as OKR (Objectives and Key Results) for Spotify which proved to be a valuable addition. Rypple built its clientele to include major companies like Facebook, Spotify and Mozilla. Rypple was then rebranded by Salesforce and added as a product currently called Work.com. (Wikipedia, 2017)

As Salesforce continues to grow, it appears to have recognized the importance of adapting to the ever-changing world of technology. It continues to add products and services, not

only for corporate growth as seen by Work.com, but also technological tools like the incorporation of Heroku and Salesforce Connect which enables a business to extend Salesforce to external data sources. Even though Salesforce is implicitly required to protect itself by building restraints like governor limits for coding and objects (i.e. database tables) and field quantities, their partnerships and compatibility with other widely known technical frameworks and libraries make the Force.com platform useful, reliable and helpful. (Salesforce, n.d.)

Limitations of Salesforce

Salesforce allows their Visualforce framework to build web application. Within Visualforce pages, there are two main ways to build an application: components, and static resources. Visualforce components are out-of-the-box items enabling smaller features to be built and help conform to the traditional Salesforce look and feel, however, the level of customization is very limited. Salesforce has done a good job at providing a large standard set of tools but these tools lack the ability to brand and create individuality which many companies seek. Additionally, components must be rendered through a Salesforce server which can have lengthy time implications as additional components are added to a page and more remote resources are consumed.

On the other hand, static resources can be used to provide the general need for customization as it allows libraries to be imported into a Visualforce page. Static resources are a fundamental means of packaging files to be used within Salesforce and served on a page through a URL. One example of using and implementing a static resource is to download a copy of jQuery, compress within a folder and upload to Salesforce. Salesforce renders resources just like any other HTML page so Visualforce pages act and behave very similarly. A Visualforce page renders HTML so static resource references resolve by loading content over the network through

a URL; however, it's easy to see how each additional library can begin to add overhead. Every file referenced requires a network request to be initiated, network resources to deliver the content and server resources to service the request. Each additional network request adds time to load the page and generally there's a set of synchronous dependencies which must be upheld to appropriately build an application. The maximum size of a static resource is 5 MB and depending on how an application is built, that size can be quickly reached if CSS stylesheets, images, fonts and multiple JavaScript libraries are used. Generally, it's best practice to split JavaScript from CSS and images but especially important to do the same with static resources which results in a frustrating and difficult to maintain build.

Charting, Graphing & Dashboards

Salesforce provides several ways to build graphs, charts and dashboard. One method is to use the standard dashboard page (example in APPENDIX A). The dashboard is available to all users and is fairly simple to build for those who are non-technical. To build a graph or chart, one must choose from a number of available reports or Visualforce pages to add and render within the dashboard. Although there are many types of graphs available, they are limited in both features and functionality - like exporting, drill down and interactivity - making them simplistic and basic.

One way to build a custom component is to have a developer build a Visualforce page which can be imported to a Salesforce dashboard. Custom components can be added to a dashboard and offer a level of customization which basic graphs and charts do not allow. This option is much more flexible in terms of construction and customization because a developer can leverage data directly from the Salesforce database and implement an external graphing library, if desired. The major drawback to a custom Visualforce page is the time and resources needed to

complete this type of functionality. Depending on the scope, at least one developer would be needed to complete the effort. Dedicated resources to design, create, test and release a component can be a major drawback for many companies needing customization.

Business Need

In the digital age, businesses are required to maintain some sense of digital presence which generally means storing information like customers, products or purchasing information. Many businesses have recognized the importance of being able to visualize the data they store and, in fact, a massive market has emerged specifically in data visualization. Complete software packages for extracting and presenting data, and even various frameworks, have been built to perform each part of the data visualization process. The up-front cost of software offering end-to-end solutions is expensive to purchase, or the time investment needed to implement frameworks and libraries can be costly. Many companies are moving to the subscription tier based pricing model which means a business would pay monthly and per user for a service solution and have access to either a desktop app or website. Take, for example, a software called Domo which uses the subscription tier pricing model (shown in Figure 1).

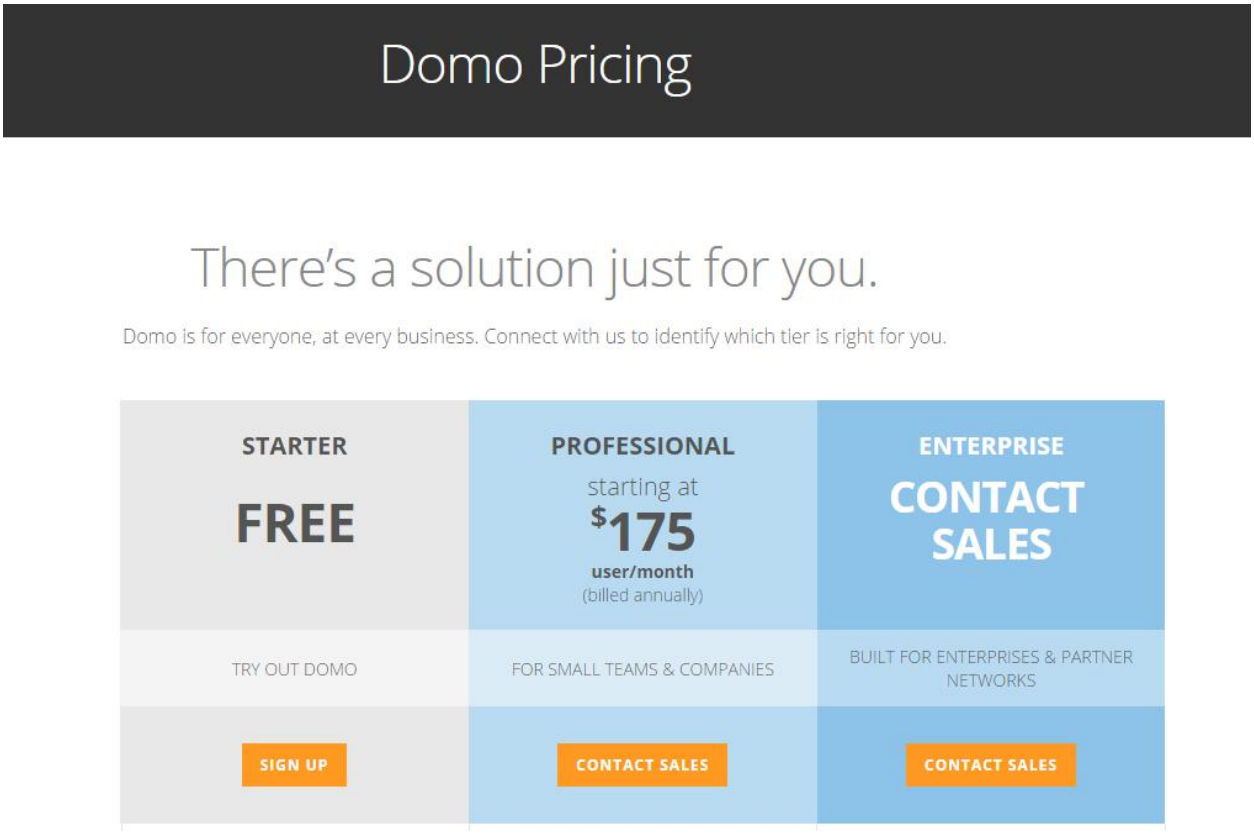


Figure 1: Domo Pricing Model

The free solution offers basic options such as a limited number of data source connections, limited scalability and customization options. Utilize the capabilities to the professional subscription and the price increases to $175 per month per user. An annual cost for a small team of, for example, 15 employees at the professional price would be approximately $31,500 annually. Domo advertises the ability to connect to cloud apps but cloud apps are also under similar pricing structure which means the total cost to utilize software like Domo coupled with a cloud based data source, like Salesforce, can quickly become expensive.

Data visualization is all about perspective. Every person creates their own opinions and morals based on life lessons and experiences. Those same opinions and beliefs help drive individual perspective. The same is true about what you see. By having the ability to manipulate

and modify one's perspective, a person will see things they normally didn't notice simply by changing visual orientation. Most companies advertise transparency to customers but should also have a desire to do the same for their employees. Transparency allows people to see clearly and straight through complicated issues. It opens the possibility of improvement and engages users to learn more about impactful situations and even areas of interest. Enabling an organization to visualize data puts business decisions in the hands of the individuals. It exposes weaknesses but can also identify strengths. The weaknesses exposed might include areas of improvement for a particular individual, business process or unit. In any capacity, relative visibility of data can highlight problems from many angles. It's not surprising why Human Resource departments like to talk to employees about their experiences because it exposes areas of improvement throughout a company. Perspective offers solutions and can be indicators of larger problems. (SAS, n.d.)

It is useful for businesses to understand the data they store but even more powerful to represent data in a personally meaningful way. In a Forbes article, Dorie Clark stated it well when she said, "it's data visualization that allows Big Data to unleash its true impact". What would happen if data visualization can be distributed and managed at a wider capacity? If a person can extract information and interpret differently then the medium of representation would make a difference. Users want the ability to see data and better understand areas which might impact their job. For example, a loan closer, who finalizes the packaging of loans for a mortgage company might want to see how many loans are waiting for an appraisal to be completed relative to the closing dates. If an appraiser lingers for a longer period of time then underwriting and closing will be impacted. If an appraisal takes too long to complete, then this could be an indicator of an unreliable appraisal company or appraiser. On the other hand, a manager would want to see how long a loan stays in each stage of the loan process. Does it stay in closing for a

long time? How many times does the loan go back to underwriting? These metrics are indicators of potential issues that could affect the loan's closing date but also signal an inefficiency in the process or individual. Decentralizing dashboard creation and enabling simple/flexible graphing options allows anyone within a business the opportunity to improve their own performance, assist in identifying bottle necks within other areas and help expose the intricacies of the business. (Clark, 2014)

Development Need

With the release of ECMAScript 2015 (ES6), JavaScript simplified and enhanced many features but one in particular which is very useful is the modularization capabilities of JavaScript. Modularization allows an application to be designed in a logical and abstract manner. Technological libraries are created and improved continuously and requires an architecture where libraries can be easily replaced. For example, many applications implement some sort of AJAX capability such as building their own library to use jQuery, fetch or axios. Most AJAX frameworks offer a small file size, wide browser compatibility or possibly both but this type of library is highly competitive and improved upon constantly. JavaScript modularization allows for one to create a module which builds all the necessary requests for an application but a separate module which manages AJAX interactions. The structure explained permits an easier replacement of one AJAX library with another, such as *fetch*, by simply building another module to perform the necessary requests using a different library. The architecture explained enables an application to stay abreast of modern and advantageous technology.

The traditional approach to building and deploying a single page application in Salesforce would be to add a reference, either locally or a CDN (content delivery network), to each library within a Visualforce page. Functionality would be added through Visualforce components (i.e.

standard Salesforce) or building custom features through JavaScript. In either case, Visualforce components require server rendering which increases time to present the page, and custom features through JavaScript can become difficult to manage and troubleshoot. One way to resolve these issues is to produce a single page application written and maintained with React and Redux. Putting the burden on the client by implementing React and Redux reduces the load on the server and allows the natural design of React and Redux to build, architect, manage and troubleshoot.

Problem Statement

Customers who use Salesforce have a wide ranging set of needs which can be addressed in many aspects by the software. One challenging area is the ability to build a nice data visualization experience. Salesforce's current data visualization is limited in functionality and tedious to build for non-technical users. Not only is it functionally minimalistic but out-of-date in appearance and technology. Data visualization is a fundamental and, arguably, necessary form of exposing information to assist a business in gaining a competitive edge. The success of a business typically depends on its ability to separate itself from competition by holding something others do not possess. By enabling employees the basic ability to visualize information in a way that uncovers pressing issues or potential problems, employees can assist a business in avoiding harmful circumstances.

This project aims to assist companies who leverage Salesforce to enable their employees the option to visualize data in a way relating most to their responsibilities. Non-technical, and technical alike, will have the ability to choose from Salesforce reports and create graphs and charts. The visualization will not only be built using much more modern libraries and frameworks but will also be easy for administrators to implement and setup. The project will

leverage Salesforce's managed packages for simple deployments and upgrades. Setup will involve importing a file onto the page and identifying an HTML element to bind and load. Styling will be blended as best as possible with Salesforce's existing stylesheets to enable a compatible appearance.

Proposed Solution

Server Application

This project will use Salesforce's Force.com platform to build a graph manager. The manager will leverage Salesforce's Apex REST feature and is further outlined in CHAPTER 3:Server Stack. The manager will supply the client with the saved state of graphs and charts as well as perform all necessary CRUD operations (i.e. create, read, update and delete).

REST API Manager

For the purpose of simplicity and extensibility, the manager will use Apex REST to build an API allowing the client to properly consume CRUD operations from the manager. An object will exist storing all user's charts or graphs and any related metadata. A metadata object will exist to allow the managed package to control the visualization frameworks and types of charts and graphs supported.

- POST / CREATE - The manager will create a graph by receiving a body with a report ID (required), graph ID (required), graph title (optional), and user ID (optional).
    - The report ID is the unique identifier given by Salesforce to associate a report.
    - The graph ID is the unique identifier given by the graph and charting client framework.
    - The graph title is the title being displayed above the chart.

    - The X axis label is the label used to describe the x axis.
    - The Y axis label is the label used to describe the y axis.
    - The user ID is the unique identifier of the user to the chart or graph being stored. The ID can be passed in directly or will be given a user ID based on the context of the user performing the POST.
    - The POST method will return a unique identifier assigned by the object.
- PUT / UPDATE – The manager will accept an ID (required), report ID (optional), graph ID (optional) and graph title for update.
    - The ID is the unique identifier originally assigned in the POST method for creation.
    - The report ID is the unique identifier associated to the Salesforce report.
    - The graph ID is the specific identifier assigned by the graph and chart client framework.
    - The graph title is the title describing the graph or chart.
    - The X axis label is the label used to describe the x axis.
    - The Y axis label is the label used to describe the y axis.
    - The PUT method will return the graph record that was updated.
- GET / RETRIEVE – The manager will accept an 'action' parameter requesting two different lists: a list of graphs created by the manager, or a list of graph types controlled by the managed package.
    - An action of 'retrieveGraphs' will return a list of the requesting user's saved charts or graphs.

    - An action of ‘retrieveGraphTypes’ will return a list of types of graphs and charts supported by the managed package.
    - The GET will return a list of graphs.
- DELETE – The manager will delete a saved chart or graph by accepting a URL parameter of ‘id’.
    - The ID parameter will delete the associated record which was originally created by the manager’s CREATE method.
    - The DELETE will return a boolean value indicating the record was deleted.

A Swagger document has been created to externally document the REST API and provide additional clarity. An overview of the documentation and URL can be seen in APPENDIX L. (Cook, n.d.)

Report and Dashboard API

This project will use Salesforce’s Report and Dashboard API because it provides a medium into Salesforce’s built-in reports. Not only can a list of reports be retrieved but metadata information pertaining to the report, like column names and groupings, can be obtained through this resource as well. Additionally, the metadata retrieved from a report combined with the Salesforce report ID can be used to retrieve the data produced by the report. It’s also important to note that the API respects Salesforce’s security model which means users cannot retrieve reports, fields or data that is not available to their user.

Unit Testing

Apex provides developers with the ability to write unit tests by properly annotating classes and methods. A unit test is, simply, code designed to validate and assert other code. It’s useful for building both common and uncommon test cases which would be encountered in

production. At a minimum, Salesforce requires unit testing to cover 75% of all code across an organization. This means that 75% of written lines of code must have test coverage.

This project will leverage Apex code to build unit tests and cover the various aspects of the chart and graph manager. A mock service will be created which will allow the unit test to mimic a request and response to perform the GET, POST, PUT and DELETE operations inherent of a REST API.

Managed Package

Packages are a containment and deployment mechanism provided by Salesforce to make sharing and managing of components and intellectual property easy. By building a managed package, this project can be distributed to other Salesforce organizations through the AppExchange or directly with a URL. The AppExchange is Salesforce's application store where Force.com software can be distributed.

There are two types of packages available: unmanaged and managed packages. Unmanaged packages are intended for single use such as a templated environment that needs to be deployed in another place, or for one time use. A good example of an unmanaged package's use is to migrate to a new environment. On the other hand, a managed package has the capacity to distribute and manage the contents publicly and provide licensing and intellectual property protection. Managed packages are placed through a rigorous process to be approved. Once approved, Salesforce will host the package through the AppExchange and allow the owner to copyright and protect the application for distribution and resale.

This project will use a managed package to enable the simple dissemination and maintenance of the project should another org use its features. A managed package will allow a

Salesforce administrator to install the project within an org but the owner of the package can deprecate and release upgrades.

Client Application

React

The client side of this project will use React to manage the presentation layer of the application. React provides numerous advantages in building a user interface by being modular and easy to manage, and will house the HTML and CSS. The HTML will be ported from the prototype and should allow for an easy migration. Since graphs will live natively inside of Salesforce, Salesforce's stylesheets will try to override the application's elements. Inline styles will be used to maintain styling regardless of its surroundings. Normally, inline styles are frowned upon because of the control taken away from stylesheets but styling can be managed with React by a process called CSS to JS. A separate file for each React component can be used to manage the styling. This structure promotes style modularity similar to how HTML components are broken into modules.

jQuery

jQuery will be used to assist in performing REST calls and help build the foundation of this project. It will be used to perform various requests to the REST API by performing CRUD operations to Salesforce. When an action, such as "Fetch Graphs", is performed, React will use jQuery to reach out to Salesforce and return a list of graphs.

FusionCharts XT

As the foundation of data rendering and visualization for this project, FusionCharts will be used to build the desired graphs and charts. At a minimum, three types of charts will be primarily used: column, line and pie charts (as shown in Figure 2, Figure 3 and Figure 4). The

project will be using the FusionCharts XT library which is a suite specific to charts and graphs as opposed to other visualization techniques, like maps.

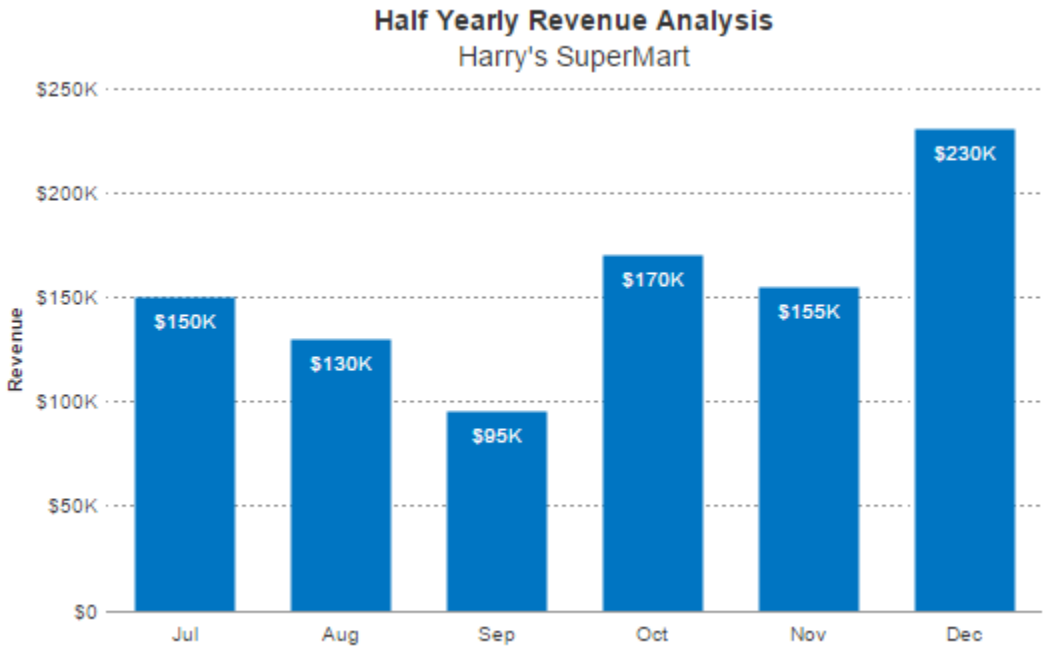


Figure 2: Column Chart

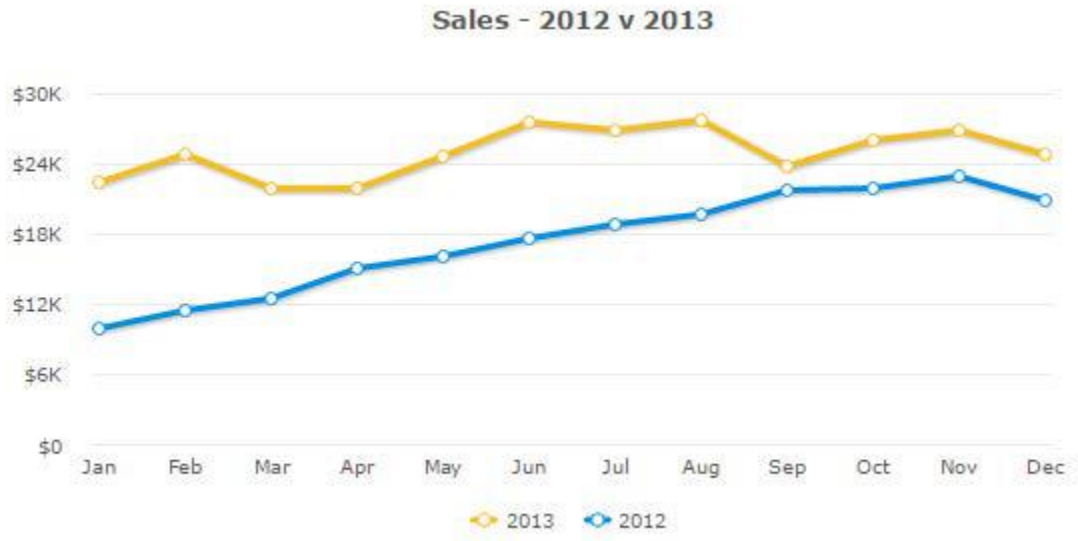


Figure 3: Line Chart

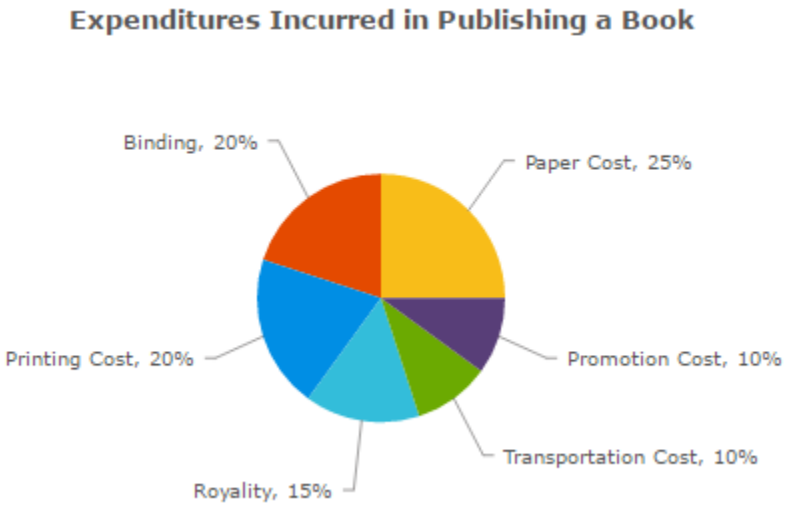


Figure 4: Pie Chart

Build Stack

Webpack

Webpack will be used to gather all the code and libraries from the JSX files and build a single file containing the entire JavaScript application. Since Webpack uses a configuration file to know which files are needed for the build and where to output the file, the example in Figure 5 shows a basic configuration. (Webpack, n.d.)

Figure 5: Webpack Configuration File (example)

```
{
  entry: './src/app.js',
  output: {
    filename: 'bundle.js',
    path: __dirname + '/build'
  }
}
```

NodeJS and NPM

NodeJS and NPM are important for this project as they will assist in helping manage all the dependencies necessary for Webpack to build a deployable file, provide automation in the build process and deploy to Salesforce. NPM will use the configuration specified in a *package.json* file to know exactly which dependencies are needed and the CLI (command line interface) processes to run. An example of a configuration file is provided in Figure 6. Browsenpm.org provides a great interactive package.json example. (NodeJitsu NPM, n.d.)

Figure 6: NPM Package.JSON (example)

```
{
  "name": "module-name",
  "version": "10.3.1",
  "description": "An example module to illustrate the usage of a package.json",
  "author": "Your Name <you.name@example.org>",
  "contributors": [{
```

```
  "name": "Foo Bar",
  "email": "foo.bar@example.com"
}],
  "bin": {
  "module-name": "./bin/module-name"
},
  "scripts": {
    "test": "vows --spec --isolate",
    "start": "node index.js",
    "predeploy": "echo im about to deploy",
    "postdeploy": "echo ive deployed",
    "prepublish": "coffee --bare --compile --output lib/foo src/foo/*.coffee"
  },
  "main": "lib/foo.js",
  "repository": {
  "type": "git",
  "url": "https://github.com/nodejitsu/browsenpm.org"
},
  "bugs": {
  "url": "https://github.com/nodejitsu/browsenpm.org/issues"
},
  "keywords": [
  "nodejitsu",
  "example",
  "browsenpm"
],
  "dependencies": {
    "primus": "*",
    "async": "~0.8.0",
    "express": "4.2.x",
    "winston": "git://github.com/flatiron/winston#master",
    "bigpipe": "bigpipe/pagelet",
    "plates": "https://github.com/flatiron/plates/tarball/master"
  },
  "devDependencies": {
    "vows": "^0.7.0",
    "assume": "<1.0.0 || >=2.3.1 <2.4.5 || >=2.5.2 <3.0.0",
    "pre-commit": "*"
  },
  "preferGlobal": true,
  "private": true,
  "publishConfig": {
  "registry": "https://your-private-hosted-npm.registry.nodejitsu.com"
},
  "subdomain": "foobar",
  "analyze": true,
  "license": "MIT"
}

{
  "name": "module-name",
  "version": "10.3.1",
  "description": "An example module to illustrate the usage of a package.json",
  "author": "Your Name <you.name@example.org>",
  "contributors": [{
```

```
  "name": "Foo Bar",
  "email": "foo.bar@example.com"
}],
  "bin": {
  "module-name": "./bin/module-name"
},
  "scripts": {
    "test": "vows --spec --isolate",
    "start": "node index.js",
    "predeploy": "echo im about to deploy",
    "postdeploy": "echo ive deployed",
    "prepublish": "coffee --bare --compile --output lib/foo src/foo/*.coffee"
  },
  "main": "lib/foo.js",
  "repository": {
  "type": "git",
  "url": "https://github.com/nodejitsu/browsenpm.org"
},
  "bugs": {
  "url": "https://github.com/nodejitsu/browsenpm.org/issues"
},
  "keywords": [
  "nodejitsu",
  "example",
  "browsenpm"
],
  "dependencies": {
    "primus": "*",
    "async": "~0.8.0",
    "express": "4.2.x",
    "winston": "git://github.com/flatiron/winston#master",
    "bigpipe": "bigpipe/pagelet",
    "plates": "https://github.com/flatiron/plates/tarball/master"
  },
  "devDependencies": {
    "vows": "^0.7.0",
    "assume": "<1.0.0 || >=2.3.1 <2.4.5 || >=2.5.2 <3.0.0",
    "pre-commit": "*"
  },
  "preferGlobal": true,
  "private": true,
  "publishConfig": {
  "registry": "https://your-private-hosted-npm.registry.nodejitsu.com"
},
  "subdomain": "foobar",
  "analyze": true,
  "license": "MIT"
}
```

User Interface

Graph Display

A user interface will be created to enable a user the option to visualize a new graph on a page. Upon navigating to the page, if a graph has been previously saved by the user then the

saved graph will be retrieved by the REST API manager and rendered using the graphing library. If a graph has not been saved then an icon indicating a new graph will be shown and allow the user to be prompted with a dialog to create a graph.

Graph Creation

A user interface will be built as shown in APPENDIX F and APPENDIX G to allow a user to create a new graph through five basic steps:

1. Select a Salesforce report
2. Select a graph type
3. Add labels
4. Preview graph
5. Save/Update graph

As illustrated in APPENDIX F, the first and second steps will show a list of items available to the user. A list of Salesforce reports will be presented using the Salesforce Report API to retrieve reports available based on Salesforce's security model. In other words, if the user cannot see the report through Salesforce then it will not be available on the screen. The list of graph types will only be available based on the support by the package. If the package allows for additional graph types to be used then the user will be allowed to render reports based on the available graph types. Lastly, the interface will enable the user to set three basic labels: the graph title, X axis and Y axis. This level of customization will let the user provide additional relevance to the graphs as they see fit.

Additional Features

Sharing graphs and charts

Graph and chart sharing would be a wonderful feature because it would allow users the option to share their chart or graph configuration with co-workers. It supports the idea of an extensible perspective as fellow workers have the ability to view the same data but through their own eyes. For the purposes of this project, the support of shared graphs will not be included. Shared reporting requires testing the security model from the viewpoint of multiple users which adds an additional level of complexity outside the scope of this project.

Additional Charts and Graphs

Other charts and graphs like combination, stacked and bubble charts are not going to be part of the basic requirements for this project. As previously mentioned, the scope of this project will be limited to three categories of charts and at least one chart or graph per category. If additional charts and graphs were supported then the level of complexity and testing would be extended immensely as each chart requires attention to verify its compatibility. However, as support for additional graph types is enabled by the package then distribution to Salesforce environments can be easily introduced through new package versions.

Additional Visualization Frameworks (i.e. high charts, D3, etc…).

The idea of adding additional JavaScript graphing and charting libraries to increase the number of supported options is desired but FusionCharts offers a large number of charts and graphs which covers a wide range of options. To support additional libraries would immensely increase the scope as it adds another dimension of, not only testing, but client side support to the project.

Dashboard

The vision for the project is to allow an end-user to build a dashboard with meaningful graphs or charts. The foundation of the project will allow for the modularization of parts and components which will enable the future of the project to permit multiple charts and graphs. For the initial scope, however, the dashboard concept will not be a focus because it requires an extensive amount of time to verify the user management of the dashboard is fully functional and free of defect.

At some point in the future, the dashboard should allow each graph's height and width to be customized which will give greater purpose to graph arrangement within the dashboard. A maximum number of graphs or charts should be implemented to keep the dashboard manageable by the user but also the application. A recommended start would be 6 total graphs or charts.

Source Control

The version control system that will be used is GIT. GIT takes advantage of a decentralized method of file management and version controlling. Traditional source control systems maintain a central copy of a master branch on a remote server and requires pushing changes to the server for file management and resolution. GIT, however, allows the existence of two types of branches and files: local and remote. Local branches are stored and maintained on a client workstation with the full arsenal of GIT commands (merge, branch, rollback, commit, etc…); and remote branches are typically stored on a remote server. When ready, one may perform a variety of actions such as pushing their local changes to the remote repository; pulling from a child, sibling or parent branch; or merging into parent branch. This project uses GIT through Bitbucket.com to host all server and client files.

Justification

The proposed build process and architecture will require a large amount of initial work and time to setup the project but will enable the implementation of features in the future to be easier and more manageable than a basic application using raw JavaScript (or libraries like jQuery). The build stack will enable a simplified mechanism for creating a single-page application, and the utilization of a modular framework such as React will provide a consistent, predictable and manageable presentation layer. While React enables control over the UI, Redux will have thorough control over the state and allow the implementation of various middleware components, like AJAX, to provide the additional control when needed. The charting framework, FusionCharts, provides an out of the box experience which is highly customizable, attractive and informative. The proposed architecture should enable quick development, application management and problem resolution.

CHAPTER 3: SYSTEMS ANALYSIS

Use Cases

A use case diagram visually represents the relationship of the user to the user interface and can be seen in APPENDIX H. Following the use case diagram are the specific use cases for this project which outlines five primary scenarios for interacting with a graph: create, retrieve, refresh, update and delete. Each of these diagrams can be seen in APPENDIX H.1 to APPENDIX H.5 and help detail the happy, and rainy, paths a user might take.

Each use case is laid out in the same basic manner: a list of primary actors, preconditions, basic flow of events and alternative flows. The list of primary actors provides the reader with the understanding of *who* will be interacting with the system. In some cases, this could be a user, administrator or possibly both. The preconditions provide the various settings needed before the

use case can begin. Preconditions help setup the environment so the reader understands all that is necessary to cultivate the use case. The basic flow of events section explains the “happy” path of the user experience and lists a step by step procedure of the user’s interaction with the system. The “happy” path illustrates what must happen for the best possible user experience. On the other hand, the alternative flows list *possible* deviations from the “happy” path. An example of a use case is provided below in Table 1 and illustrates the variances of the graph creation use case. The same use case can be seen in APPENDIX H.1.

Table 1: Appendix H.1; Create a New Graph

Create a new graph

Primary Actors: Salesforce User

Preconditions: The application has been deployed to a Visualforce page.
The user has access to the Visualforce page containing the application.

Basic flow of events:

1. User has logged into Salesforce using their Salesforce credentials.
2. User navigates to the page where the application is deployed.
3. The application queries the REST API manager for any graph previously created.
4. The user clicks on the button to create a new graph.
5. A modal dialog opens to provide options for creating a new graph.
6. User selects a report.
7. Application queries Report API for metadata if it does not already exist in the local state.
8. Application queries Report API for data if it does not already exist in the local state.
9. Application passes selected report data to chart and graph library for rendering.
10. User selects a graph type.
11. The application reloads the graph with the selected graph type.
12. The user types a value for the graph's header, X Axis or Y Axis labels.
13. The application reloads the graph with the label values.
14. User clicks the create button to save the graph.
15. The application persists the selected report, graph and properties of the graph as a new record in Salesforce.
16. The application closes the modal.

Alternative flows:

1a. User is unable to authenticate with Salesforce.
1a1. Salesforce is locked out/invalid username/invalid password.

2a. The application has not been deployed.
2a1. User is unable to navigate and access application.

6a. No reports exist for the user to see.
6a1. The list of graphs will only have a single value, "Please Select a Report".

7a. A network issue has occurred.
7a1. A notification appears stating, "Unable to retrieve report data."

8a. A network issue has occurred.
8a1. A notification appears stating, "Unable to retrieve report data."

8b. No groupings exist for the report.
8b1. A notification appears stating, "This report must have groupings to display a graph."

9a. The data is invalid.
9a1. The chart and graph library will render with a message stating, "No graph data to display."

11a. The data is invalid.
11a1. The chart and graph library will render with a message stating, "No graph data to display."

13a. The data is invalid.
13a1. The chart and graph library will render with a message stating, "No graph data to display."

14a. An issue occurred during save.
14a1. A notification appears stating, "The graph was unable to save."

State Diagram

The state diagram shown in APPENDIX J illustrates the various states the page can take. Each step is initiated by the user and will interact with the graph manager based on the event that changes the state. Whenever the graph is re-rendered, the application will tell the graphing library to update the rendered view. If a user chooses to delete a graph then the application will remove the graph from the page assuming a successful response from the API. Lastly, an error message is shown to the user should the API return some sort of error.

Screen Mockups

Screen mockups should be created by user interface and user experience (aka UI/UX) professionals because they seek to design an interface that flows naturally, appears vibrant and beautiful, and simple. Mockups are an important step in the systems analysis phase of software design because it collects the ideas, meetings, discussions and documentation into a visual representation available for stakeholders to see and confirm. From mockups, stakeholders can develop an understanding of the user experience and either choose a mockup which meet the business requirements or request additional changes. Mockup illustrations assist in providing a visual contract between the business stakeholders and the development team. For this project, three stages of mockups were built before deciding on a design which was believed to provide a level of simplicity and attractiveness to the user.

Mockup 1

The first design in APPENDIX A shows what the user would see when navigating to the page after previously saving a graph, while APPENDIX B shows a sketch of how a user could be presented with a modal to select a report by filtering Salesforce objects. The next step is to allow

the user to advance to a secondary modal to choose a graph type and preview the graph. The selected properties would be rendered by a graphing framework based on the options selected.

Mockup 2

The second design shown in APPENDIX C helps illustrate a saved graph and the expectation of how the project would look within a Visualforce page. The header shown is a typical Salesforce navigational header and left navigational component seen when an authenticated user logs-in. The background header was captured from the Winter 2016 release. The mockup also depicts a potential feature of creating multiple graphs within the same page (aka dashboards). In the mockup, a "+" sign illustrates the ability of the user to create additional graphs to be added to the dashboard.

Mockup 3

The third, and final, design can be seen in APPENDIX D to APPENDIX F. APPENDIX D is a sketch which shows a concise and simple recommendation for this project. The sketch suggests an interface that is concise and not spread out across several modals. The design allows the user to choose all the desired options and view a preview of the graph without ever leaving the modal. Once ready, the user may choose to save the graph by clicking the button on the bottom right corner. Additionally, the same modal and view will be used to enable the update of a graph.

IDE

This project will use the Sublime Text 3 text editor and the MavensMate Plugin for Salesforce. Sublime is a widely used and supported text editor with benefits like large IDEs but without the system overhead. Sublime has the footprint of a small text editor but has the plugin capacity and community support of developers. MavensMate has multiple plugins ranging from

Sublime Text, Atom to Visual Studios, and the latest version uses a local server on Node.JS and a desktop app to provide a user interface that allows customization of the editor's plugin and local server. The local server helps provide the connection directly to Salesforce. The server, desktop app and plugin architecture has enabled the MavensMate creators the ability to add additional plugins more easily and span across multiple text editors and IDEs. (Sublime Text, n.d.) (Ferrara, n.d.)

Build Stack

NodeJS

As described by the official NodeJS documentation, it is "a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. NodeJS uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices." NodeJS has become increasingly popular because it is fast, scalable and easy to setup. NodeJS executes asynchronously which means it does not wait for a method call to complete before continuing. An event driven architecture allows the server to dispatch a method call and use events to manage call completion. Having been built on Chrome's JavaScript engine enables the library to be very fast as well. Currently, the environment can run on OSX, Microsoft and Linux. (Node.JS Introduction, n.d.)

NPM is the package manager for Node.js and comes distributed with the main installer. Publicized as the largest ecosystem for open source libraries in the world, it is used to install and manage dependencies for applications. NPM can be used to design a build stack to help assist with development consistencies and easy platform configuration. Node.js uses a configuration

file to build environment dependencies by automatically checking and installing both environment and development dependencies. (Node.JS, n.d.)

Node.js and NPM will allow a build stack to be created to manage all the libraries necessary to build the proposed application. React, Redux, FusionCharts, webpack, and superagent are all available in NPM. Each library, while serving its own purpose, is important in assisting the application in performing the appropriate task. For example, React is important in building the UI in a manageable and efficient way while Redux extends React to properly perform capabilities like AJAX calls using the cross-browser and lightweight superagent library.

Another perk to using Node.js is the ability to separate build environments. If a development environment needs a set of debugging tools, then Node.js can control which libraries are packaged for each environment. For example, a debugging tool might be desired in a development environment but not production; whereas a logging tool is needed in production but not development. Node.js provides the ability to differentiate and build as desired.

JSForce

JSForce is a library wrapper available in Node.js that provides access to Salesforce's Force.com Migration Tool. The Force.com Migration Tool is a deployment mechanism which can be used to package and send files to a Salesforce org. By providing the username, password and security token, JSForce can deploy Salesforce files like static resources, classes, pages, labels, metadata and any other file type Salesforce accepts. JSForce provides a simple to use library for making a connection into Salesforce. Being available in Node.js allows it to integrate well into this project's build process. (Mashmatrix, n.d.)

Webpack

Webpack is a module bundler which takes a variety of assets, like JavaScript, images, CSS and fonts and compiles them using a single configuration file. Using Node.js to compile, a "bundle.js" file is created which contains a single instance of all dependencies, code and static assets needed to run an application while excluding unused dependencies from the bundle. (Webpack, n.d.)

One specific benefit of webpack is to reduce the overhead typically associated with managing dependencies and library references by building a dependency map containing the relationships of all assets. Webpack builds the dependencies by finding any "require()" or "import" statements in the code and navigating each dependency path until it reaches the bottom of the tree. This particular benefit is one reason webpack makes itself useful because it reduces the total number of HTTP requests needed to build a page. (Webpack, n.d.)

Another advantage is the ability to separate assets based on environments. In the example with Salesforce, static resources can grow too large if CSS, images and JS code is combined. Having the ability to separate assets is highly valuable on a cloud-based environment like Salesforce keeping in mind the 5 MB limit imposed on static resources. (Ray, 2016)

ESDoc

ESDoc is a very powerful document generator for JavaScript that produces a meaningful and concise HTML document outlining a project's structure and contents. ESDoc can be customized to understand both ES6 and Babel. (Koba, 2015)

Server Stack

AJAX

AJAX provides an application with the ability to asynchronously perform network requests without refreshing the application. This type of call is highly useful when trying to

provide a seamless user experience and a page refresh is not necessary or desired. For the purposes of this project, multiple AJAX calls are made throughout the application starting at the time the page loads and then as various CRUD operations are performed. At the time the page loads, the application performs a series of up to three asynchronous AJAX calls. There are two initial calls made to the REST manager: the first to retrieve a list of graphs saved by the user, and the second returns a list of available graph types. The third call is to Salesforce's Report API with the intention of providing the application with all available Salesforce reports. (TutorialsPoint, n.d.)

The next series of AJAX calls occur when the application requests a CRUD operation or needs to retrieve metadata and data for a report. In each scenario of creating, reading, updating or deleting a graph, an AJAX call will be made to the REST manager to perform the necessary operation. Report metadata is necessary to determine if the report is compatible with the graphing framework and is made whenever the application determines the metadata does not exist locally. The same logical process is examined to retrieve data for the report but, first, requires metadata to be retrieved.

Force.com

The Force.com platform is used by programmers to extend Salesforce's cloud environment. Applications can be built by using the platform to innovate and distribute, or customize and implement. Businesses, for example, who seek to create applications can leverage the Force.com platform and distribute a solution on the App Exchange and allow other businesses to download the app in their environment (i.e. "org"). Applications built on this platform are automatically scaled for performance, secured and backed up to remote servers. Since Salesforce is a cloud environment, all code exists and executes remotely. The platform has

its own server language called Apex, database languages called SOQL and SOSL, and client-side interface named Visualforce.

APEX

Apex is Salesforce's server-side language and was built on the Java platform. Like Java, the language is strongly typed and object oriented. Apex is used in Salesforce to extend business logic and customize the environment to fit the tailored needs of a business. The language has the ability to perform DML (data manipulation language) operations to the database, such as insert, update, read and delete.

Apex can also build, deploy and schedule batch processing. Batch processing allows an Apex programmer to bulk-ify code for the purposes of executing that code on a larger scale. For example, a programmer could build a batch to perform a sequence of synchronization processes on a nightly basis. The batch could be scheduled to run at 11 PM every night with the intention of accessing a remote database and synchronizing data between Salesforce and the remote database. To schedule a batch, Apex uses Cron jobs to initiate and process the scheduler.

Apex classes can be built to provide REST resources enabling custom API (application programing interface) calls. Custom API calls help service a separation and standardization of access into an environment. A custom call would allow a name to be identified as an entry point into Salesforce with, of course, proper authorization and some sort of action performed.

SOQL & SOSL

SOQL is Salesforce's database language and stands for Structured Object Query Language. It is similar to SQL (Structured Query Language) but is built specifically to interact with Salesforce's database model. To perform a query to Salesforce's standard Account, one would write 'SELECT Id FROM Account' which is identical syntax to a SQL statement.

However, one of the differences between the two database languages lie within accessing object relationships. Salesforce does not have the notion of explicit join statements to access a related sObject. Syntactically different but functionally identical statement, one would perform a query to retrieve a list of Account Ids and the names of the associated Contacts by writing the following: 'SELECT Id, Contact__r.Name FROM Account'. Notice the "__r" which indicates to the database language to drill down in the database structure. Another noticeable difference is in selecting fields. SOQL does not allow a user to select all fields within an sObject. Instead, one must select each individual field which is different from SQL because it allows a wild card character (*) to select all fields within a table. Additionally, SOQL does not have the notion of a stored procedure like SQL but, instead, a healthy merging of SOQL queries into Apex code can be done to access data in Salesforce.

Visualforce

Visualforce is Salesforce's HTML rendering engine which takes a page and produces HTML markup. Visualforce provides many useful out-of-the-box features including data tables, forms, input fields and many other standard components to build useful applications. There are many ways to connect to Salesforce data but the most standard and, generally, fastest way to build a Visualforce page is by leveraging Visualforce components within a page. Visualforce does enable custom implementations by allowing one to write custom components when non-standard functionality is needed.

Salesforce Report and Dashboard REST API

The Report and Dashboard REST API is a Salesforce tool that can be used to obtain detailed information about reports and dashboards. The API provides a set of functions for retrieving metadata information which is handy in describing reports such as column information, category data and data groupings. Once obtained, the metadata has to be posted to

the Report API by calling '/services/data/v37.0/analytics/reports/query'. The Report API is not only useful for getting descriptive information about reports and dashboards but also create, update and delete operations. This project does not have a need to create or modify a report but will be using the describe and query functions to obtain information and data.

One of the major benefits to Salesforce is it being a cloud based development platform. This means both hardware and software are ready to be used as soon as an environment is assigned. Salesforce has what is known as a "developer org" which is an environment intended specifically for people who would like to prototype on the Force.com platform without the necessity of purchasing an org.

From session management to field level security (FLS), Salesforce security is ready to go "out of the box". The various levels of security can be implemented at the administrator's discretion. By choosing Salesforce, security is natively handled and allows the developer to focus on building features and solid applications.

Client Stack

A widely known Salesforce contributor by the name of Jeff Douglas provided a great example for creating a basic single-page application using Webpack, Node.JS, React and Reflux. This project adapts Mr. Douglas' example to fit a slightly different build stack but the example provides a tremendous resource for showing how to use React and Redux within the Salesforce context and using a deployment mechanism like JSForce with Node.JS and Webpack. (Douglas, 2015)

JavaScript

JavaScript has become a web standard for developing client-side applications in every major browser. It's flexibility and level of customization allows a programmer to, essentially,

develop in many ways. JavaScript is asynchronous which means each method execution does not wait for the previous execution to complete before moving to the next. The idea of asynchronous execution is very different than server code because server languages, by default, are made to run synchronously. While most server languages allow a programmer to build asynchronous features, it can be tricky, if not impossible, to force JavaScript to run synchronously. With the improvement of both capacity and speed of modern hardware, programmers have begun to offload much of the workload from servers to client rendered applications. Additionally, the shift could be, in part, due to the growing fiscal justification of businesses moving to cloud infrastructures like Amazon Web Services and Salesforce.

I decided to use JavaScript ES6 independently from other frameworks like AngularJS and KnockoutJS because standalone JavaScript offers plenty of functionality for the purposes of this implementation. Angular and Knockout provide better ways for managing JavaScript and the DOM but come at a price – longevity. In many cases, both frameworks bind to JavaScript in a way which makes it difficult to ever convert from one framework to another or revert to raw JavaScript.

jQuery (DOM Library)

jQuery has become the primary standard for accessing and manipulating the Document Object Model (DOM) from within JavaScript. jQuery is a lightweight, versatile library that one loads into the browser and can be used to do things like wild card searching DOM elements, chaining, event binding, AJAX and many other useful features. The library comes in a compressed or uncompressed form which allows a developer implementing the library to use discretion if disk space is a concern. Although jQuery has its own set of useful features, it has

many extensions which increases its usefulness drastically. Each extension can be imported into the browser, just like jQuery or any other script.

jQuery's website boasts support for all major desktop and mobile browsers versions which means Chrome, Edge, Firefox, Internet Explorer (IE), Safari and Opera are all intentionally supported browsers at their latest stable release plus one previous stable release. In fact, a disclaimer is shown on their compatibility page explaining older, or even different browsers besides those listed, might be supported but full functionality is not expected nor tested and supported. To support older browsers, jQuery recommends using jQuery 1.12.

jQuery is a framework that has been around for nearly ten years and continues to build momentum. It is reliable, easy to use and flexible. The alternative to jQuery in most scenarios is to build a custom DOM accessor and manipulator which is not always feasible or even necessary for most applications. However, custom DOM frameworks are generally useful when a dependency to a specific library is required by a project.

React (Presentation Library)

React is a data rendering library allowing programmers to build modular, interactive user interfaces (UI). It uses a series of components as building blocks which can service an entire application or a stand-alone feature. React can use Babel, a JavaScript processor, to simplify and expedite component creation, or entirely by itself. React was built at Facebook and currently maintained by a community of developers and organizations including Facebook and Instagram. It was first released and utilized by Facebook in 2011, and has become an increasingly popular framework to use (Arora, 2016).

At the beginning of 2016, the React website issued a press release stating it will no longer support IE8 beginning in release version 15. Instagram specifies support for Chrome, Firefox,

Safari, Internet Explorer and Opera; while Facebook states the same support but adds Microsoft Edge to the list. Considering the association of both Instagram and Facebook, it's safe to think React supports the same.

Since this application is going to service a repetitive user interface, it makes sense to use React and to modularize the various aspects of the UI into components like buttons, modals, headers and input fields. Its compatibility with the other chosen libraries works well, and should help simplify the maintenance of issues and expansion of features.

Redux (State Management Library)

Redux is a library that allows an application to control the state through a series of actions. Actions dispatch requests to manipulate the state by notifying reducers of changes. This is different than some state machines which allows a direct update of the state. A reducer is intended to modify, or reduce, the state by receiving both the action and the state. In most scenarios, the reducer will check the action dispatched and determine if a change to the state is warranted, otherwise it will return the state passed to it. This type of state machine is called a "predictable state container" because actions can be recorded (Abramov, 2016). If the recorded actions are replayed then the machine will appear at the same location every time.

Another wonderful feature of Redux is the implementation of middleware. Middleware "provides a third-party extension point between dispatching an action, and the moment it reaches the reducer." In other words, the middleware receives the state and dispatched action, and is afforded an opportunity to perform an operation before the reducer receives the action. (Redux Community, n.d.)

In this project, Redux is crucial in managing several reducers within the application's state. The first important reducer is the status of the user interface. The UI's status will provide a

way to know where the UI is at any given point in time. For example, if the user interface is loading then the application should disable the ability to create or edit a graph. Secondly, a reducer will be needed to maintain each graph created by the user, all available graph types, and each report accessible by the user. Lastly, middleware will be needed to manage AJAX requests.

FusionCharts (Chart and Graph Library)

FusionCharts is a charting library for JavaScript which contains over 90 charts and 1,000 maps. It offers compatibility with other JavaScript frameworks like jQuery, AngularJS and ASP.Net. It accepts both JSON and XML, and contains interactive graphs making the library even more useful. The library is offered as free for personal use but is stamped with a watermark until purchased. For the purposes of this project, the evaluation copy will be used. One of the main features that made the library a prominent choice is its export ability. Native to the library is a feature allowing the user to export the generated chart into PDF, PNG, SVG, XLS or JPEG. The library even provides a function for exporting a group of graphs. (FusionCharts, n.d.)

Charting libraries come in a wide variety of options and with many features. FusionCharts was compared to three other charting libraries (Flot, Highchart and D3) and narrowed down as each feature made it more enticing. Almost every library boasts of browser compatibility but most try to render in SVG which can limit the availability in older browsers. FusionCharts offers charts in both SVG and VML. The only downside seems to be the watermark.

Toastr (Notification Library)

Toastr is a non-blocking customizable JavaScript notification library. Toastr comes with a stylesheet providing a very basic and attractive appearance. The library allows the position of notifications to be controlled (ex. Bottom-left, top-center, middle-right). Additional settings like duration, animation and callbacks make the library even more useful. (Ferrell, n.d.)

Superagent (AJAX Library)

Superagent is a lightweight AJAX library available for browsers and Node.JS. The Node.JS version uses the core HTTP module while the browser version uses XHR. If a package manager is used then superagent will use the XHR option. Superagent offers a wide range of features like simple error handling, promises, CORS support and cross-browser support. Figure 7 shows the current browser compatibility. (Vision Medio, n.d.) (Vision Media, n.d.) (Farmer, 2016)

Figure 7: Superagent Compatibility

Deployment

Salesforce Managed Package

Salesforce's managed packages enables code to be deployed from one environment to another. The project proposed will exist inside of a managed package for easy deployment to a destination environment. Included in the package will be static resource(s), classes, custom object(s) and field(s), and custom metadata types. Managed packages allow the package owner to deprecate and version releases as new builds are created.

CHAPTER 4: PROPOSAL

Deliverables

Summary

A managed package will be created to allow a version of this project to be deployed to a Salesforce environment. The managed package will include various components such as:

- Metadata Object
- Custom Object
- Custom Visualforce Component
- REST API Class
- Utility Class (if necessary)
- Static Resource

The REST API will enable RESTful access to any user saved graphs as outlined in CHAPTER 2:Server Application use cases. The static resource will contain the bundle file generated by Webpack and will be referenced within a Visualforce page. Since the static resource contains a compiled version of React and Redux, the user interface will be built into the static file.

The delivered application will allow a user to perform four interactions through this architecture:

- Create Graph
- Update Graph
- Retrieve Graph
- Delete Graph

When creating a graph, the user will have the option to select from one of three graph types: column, line and pie. After selecting a graph type, options will be presented to customize the graph's header, X axis and Y axis. Each selection, or modification, will render a preview of the graph for the user to see. A preview of the graph will allow the user to visualize and make additional changes, if desired. Once ready, the graph selections will be saved and an example will be rendered on the Visualforce page.

Working Prototype

A working prototype will be created to migrate the final flat mockup into an HTML and CSS page using jQuery. A small custom library will connect the HTML page to the REST API manager. The page will be connected to the custom library to help test the page and the library. The intention behind the UI prototype is to quickly build an HTML and CSS replica of the design presented in CHAPTER 3:Screen Mockups (the last mockup). The CSS will be reused by React but the HTML will be broken apart and placed inside React.

Unit Testing

Unit tests will be created within Salesforce to help eliminate potential bugs but also to help with a Salesforce coding mandate. Salesforce requires 75% coverage across the entire code base which also includes managed packages. While 75% is the requirement, this project will, of course, try to achieve 100% coverage. A mock service will be created in Salesforce which mimics the REST API manager in creating, updating, retrieving and deleting graphs. The code coverage will include unit tests for:

- Creating a new graph,
- Updating an existing graph,
- Retrieving a previously created graph,

- Deleting a graph, and
- Retrieving graph types.

The unit tests execute automatically whenever a package is being deployed and can be tested anytime within Salesforce. Additionally, each class can also be examined for coverage by navigating to the class within Salesforce and clicking the button to run the tests. As a note, many Salesforce IDEs provide capabilities for determining overall org wide, individual class and managed package coverages.

Justification / Benefits

Importance of Project

Data is absolutely meaningless without representation. Users who can visualize data are provisioned with the ability to make informed decisions, suggest appropriate courses of action or initiate self-improvement. Many times, the responsibility is placed on managers and upper-level decision makers to visualize data but empowering employees of all levels with the ability to see information as it is relevant promotes a culture of self-accountability and transparency. An employee who can see how they impact a process might take action to improve their performance. A person who spots a bottleneck might suggest a plan of action to correct a problem. The bottom-line is we all see things from our own point-of-view. This can be advantageous if used correctly which is why the project aims to create a small software package with a basic solution which can act as a building block to something even more useful.

Architectural Justification

Salesforce provides a stable and pre-built environment that's highly conducive for innovation. Being a multi-tenant, cloud based platform, the overhead in building the environment

is minimal and performance is managed by Salesforce. Security is standard and can be modified to fit the needs of a business. For this project, a REST API could be built to retrieve the graphs and graph types with respect to record and field level security. Salesforce's deployment mechanism enables the easy distribution of code through initial deployments and upgrades by using managed packages. Managed packages provide such a value for Salesforce because of the possibility of extending and selling code created for Salesforce.

JavaScript can be an extremely lightweight and flexible language and, implemented in the right context, can be efficient, fast and scalable. With various improvements in ES6, JavaScript modularization declaration was simplified to help provide additional value. The combination of ES6, React and Redux should help promote an architecture that, once implemented, is easy to improve upon, swap out functional libraries, scalable and manageable. React's modularity of the presentation layer will help create components that are constantly reusable. For example, a button created once in React can be used throughout the application. This feature alone provides ample value because the developer is not having to recreate all the features that come with a button position. Additionally, Redux begins to add more value by bringing the presentation layer to life through state management. Data validation, data retrieval, local and remote logging, and many other features have a better possibility of being used and managed through a library like Redux.

The build stack provides a level of importance that would allow a development team to pick this project up and continue adding to it. Using node enables a simple command to execute and install all the dependencies needed to start developing in the project, and then Webpack takes over when it's time to package all the files. Besides having a Salesforce environment to point to, the configuration is minimal.

This project, coupled with the architecture and build procedure, will provide a great example and prototype for an elaborate single page application implemented inside Salesforce. The benefit behind seeing how an app can be created and managed through an increasingly popular set of frameworks like React and Redux but also using ES6 to build logical and reusable components should encourage and motivate future applications.

Project Plan

Timeline and Estimates

Task Name	Duration	Baseline Start	Baseline Finish
Capstone Project	**71.5 days**	**Mon 1/2/17**	**Mon 2/13/17**
Create and Setup GIT	1 hr	Mon 1/2/17	Mon 1/2/17
Create Managed Package	2 hrs	Mon 1/2/17	Mon 1/2/17
Development	**20.38 days**	**Mon 1/2/17**	**Mon 1/16/17**
Salesforce	**3.75 days**	**Mon 1/2/17**	**Fri 1/6/17**
Custom Object	4 hrs	Mon 1/2/17	Mon 1/2/17
Custom Metadata Object	2 hrs	Mon 1/2/17	Tue 1/3/17
Visualforce Page	2 hrs	Tue 1/3/17	Tue 1/3/17
Apex	**2.75 days**	**Tue 1/3/17**	**Fri 1/6/17**
Build REST API Manager	**2.75 days**	**Tue 1/3/17**	**Fri 1/6/17**
Create POST Handler	**0.75 days**	**Tue 1/3/17**	**Wed 1/4/17**
Store Graph	6 hrs	Tue 1/3/17	Wed 1/4/17
Create GET Handler	**1 day**	**Wed 1/4/17**	**Thu 1/5/17**
Return Saved Graph	4 hrs	Wed 1/4/17	Wed 1/4/17
Return Graph Types	4 hrs	Wed 1/4/17	Thu 1/5/17
Create PUT Handler	**0.5 days**	**Thu 1/5/17**	**Thu 1/5/17**

Update Graph	4 hrs	Thu 1/5/17	Thu 1/5/17
Create DELETE Handler	**0.5 days**	**Thu 1/5/17**	**Fri 1/6/17**
Delete Graph	4 hrs	Thu 1/5/17	Fri 1/6/17
Prototype	**3.13 days**	**Fri 1/6/17**	**Wed 1/11/17**
Convert Mock-up to HTML and CSS	10 hrs	Fri 1/6/17	Mon 1/9/17
Build JavaScript Library	**1.88 days**	**Mon 1/9/17**	**Wed 1/11/17**
Connect to REST API Manager	6 hrs	Mon 1/9/17	Tue 1/10/17
Connect to Salesforce Report and Dashboard API	9 hrs	Tue 1/10/17	Wed 1/11/17
React Graph UI	**3.5 days**	**Wed 1/11/17**	**Mon 1/16/17**
Modal	10 hrs	Wed 1/11/17	Thu 1/12/17
Graph	4 hrs	Thu 1/12/17	Thu 1/12/17
Graph Helper	8 hrs	Fri 1/13/17	Fri 1/13/17
Button	4 hrs	Mon 1/16/17	Mon 1/16/17
Icon	2 hrs	Mon 1/16/17	Mon 1/16/17
Testing	**3 days**	**Mon 1/16/17**	**Mon 1/19/17**
Bug Fixes	3 days	Mon 1/16/17	Thu 1/19/17
Finalize Paper	**18.19 days**	**Thu 1/19/17**	**Mon 2/13/17**
Work on Main Content	3 wks	Mon 1/23/17	Mon 2/13/17
Explain Implementation	2 days	Thu 1/19/17	Fri 1/20/17
Explain Results	3 days	Fri 1/20/17	Mon 1/23/17

Specific dates

November 1st, 2016 : Document Delivery

A draft of the document will be delivered to the committee for their review two weeks prior to the defense proposal.

November 15th, 2016 : Proposal Defense

Propose the project to the committee.

April 7th, 2017 : Document Delivery

The document will be delivered to the committee for their review and approval two weeks prior to the final defense date.

April 21st , 2017 - 10:00 AM : Final Defense

The final defense will be the demonstration of the application with an emphasis on the work and milestones achieved.

Any additional features included will be outlined.

Predictions

Learning Curve

As with any new and uncharted architecture, there will be a learning curve. The developer is very familiar with developing on the Force.com platform but is not well versed in the React library, or Salesforce's Reporting API. Converting flat mock-ups into HTML is not one of the developer's strong points but the user interface is not overly complicated which should help expedite the HTML implementation. The intention behind the jQuery prototype is to quickly build a functional mock-up that interacts with Salesforce. The portability from the raw HTML into React should assist in expediting the conversion to the React library.

React is a new library for the developer but he has read much of the documentation provided by Facebook and navigated through a basic example of the library to assist in solidifying foundational concepts and recommended design approach.

Optional Features

The design decision to use React should help make extending this project with new features relatively simple. With that in mind, hopefully additional features outside of the proposed scope can be fulfilled to increase the overall functionality of the project.

Using React to render the presentation should allow one to build the application with such modularity that every component can be reused and customized. This choice in framework should make extensibility much easier for future features, like a dashboard. Since a dashboard is a group of graphs, a page can be created which allows React to create multiple graphs. The same graph component will be used repeatedly which should make the extensibility relatively easy.

The ability to add additional graph types should be easily obtainable for both the library and through package deployment. By having graph types maintained as meta data types within the Salesforce managed package, the extension of graph types supported by the project can be both added and pushed to subscribing environments. The graphing library supports a wide array of graphs and charts but the limiting factor is supplying the proper data based on the graph type. It is expected that one, maybe two, graphs can be added to help extend the usefulness of the project.

A benefit that is intended to be provided by this project is the ease of installation and setup when attempting to deploy the project to an environment. Salesforce's managed package process should provide the desired ease of deployment. However, deploying a product somewhere is only part of the battle. The other step believed to be achievable in this project is the simplicity in which the package can be configured and presented as usable for a customer.

CHAPTER 5: RESULTS

Summary

Deliverables

The project aimed to provide a set of deliverables which built a REST API, unit testing for the REST API, client application that connected to the REST API and a managed package to deploy the project. In addition, a prototype was to be used to build the basic functionality of the application using basic HTML, JavaScript and jQuery.

A REST API was created which used two namespaces to provide access: graphs and graph types. The *graph* namespace allowed a request to retrieve a list of graphs, delete a graph, update a graph or create a new graph based on a user. The *graph types* namespace enabled a request to, simply, return a list of graph types. The decision to separate graph types from graphs was made during the implementation because it was realized a separation of the two services made more sense as they served different purposes. Additionally, each namespace has a different Apex class to service the request and appropriate unit testing was made.

The client application leverages HTML, CSS, React and Redux to build itself. React modularizes the HTML and CSS into reusable components while Redux helps handle the state of the components. The outcome of these resources produced an application that builds a graph in a repeatable fashion which ultimately renders itself as a dashboard like in APPENDIX D. Each graph is editable as mocked in APPENDIX F and APPENDIX G, and connects to the REST API to perform operations outlined in CHAPTER 2:Proposed Solution.

The managed package neatly collects all the components from both the REST API and the client application into a managed package for distribution to other organizations. The

managed package created was called *ForceGC* to identify it as a graph and chart package on the Force.com platform. The managed package is listed as a beta managed package. Beta managed packages are limited to certain features like pushing updates to subscribing organizations.

Proposal

For the most part, the items listed in the proposal were carried through to implementation. There was only one small design modification made which was believed to improve the customization of the application, and a couple architectural decisions that were needed after it was realized that the internal state of React would not be able to manage such a dynamic application.

Timeline (Expectation vs Reality)

The intention was to start on the prototype January 1st but the prototype wasn't under development until the end of January due to circumstances outside the control of the developer.

Task Name	Duration	Baseline Start	Start	Finish
Capstone Project	**71.5 days**	**Mon 1/2/17**	**Mon 1/2/17**	**Fri 4/7/17**
Create and Setup GIT	1 hr	Mon 1/2/17	Mon 1/2/17	Mon 1/2/17
Create Managed Package	2 hrs	Mon 1/2/17	Mon 1/2/17	Mon 1/2/17
Development	**20.38 days**	**Mon 1/2/17**	**Wed 2/1/17**	**Wed 3/1/17**
Salesforce	**3.75 days**	**Mon 1/2/17**	**Wed 2/1/17**	**Sun 2/5/17**
Custom Object	4 hrs	Mon 1/2/17	Wed 2/1/17	Wed 2/1/17
Custom Metadata Object	2 hrs	Mon 1/2/17	Wed 2/1/17	Thu 2/2/17
Visualforce Page	2 hrs	Tue 1/3/17	Thu 2/2/17	Thu 2/2/17
Apex	**2.75 days**	**Tue 1/3/17**	**Thu 2/2/17**	**Sun 2/5/17**
Build REST API Manager	**2.75 days**	**Tue 1/3/17**	**Thu 2/2/17**	**Sun 2/5/17**

Create POST Handler	**0.75 days**	**Tue 1/3/17**	**Thu 2/2/17**	**Sat 2/4/17**
Store Graph	6 hrs	Tue 1/3/17	Thu 2/2/17	Sat 2/4/17
Create GET Handler	**1 day**	**Wed 1/4/17**	**Sat 2/4/17**	**Sat 2/4/17**
Return Saved Graph	4 hrs	Wed 1/4/17	Sat 2/4/17	Sat 2/4/17
Return Graph Types	4 hrs	Wed 1/4/17	Sat 2/4/17	Sat 2/4/17
Create PUT Handler	**0.5 days**	**Thu 1/5/17**	**Sat 2/4/17**	**Sun 2/5/17**
Update Graph	4 hrs	Thu 1/5/17	Sat 2/4/17	Sun 2/5/17
Create DELETE Handler	**0.5 days**	**Thu 1/5/17**	**Sun 2/5/17**	**Sun 2/5/17**
Delete Graph	4 hrs	Thu 1/5/17	Sun 2/5/17	Sun 2/5/17
Prototype	**3.13 days**	**Fri 1/6/17**	**Sun 2/5/17**	**Sat 2/11/17**
Convert Mock-up to HTML and CSS	10 hrs	Fri 1/6/17	Sun 2/5/17	Tue 2/7/17
Build JavaScript Library	**1.88 days**	**Mon 1/9/17**	**Tue 2/7/17**	**Sat 2/11/17**
Connect to REST API Manager	6 hrs	Mon 1/9/17	Tue 2/7/17	Thu 2/9/17
Connect to Salesforce Report and Dashboard API	9 hrs	Tue 1/10/17	Thu 2/9/17	Sat 2/11/17
React Graph UI	**3.5 days**	**Wed 1/11/17**	**Sat 2/25/17**	**Wed 3/1/17**
Modal	10 hrs	Wed 1/11/17	Sat 2/25/17	Sun 2/26/17
Graph	4 hrs	Thu 1/12/17	Sun 2/26/17	Sun 2/26/17
Graph Helper	8 hrs	Fri 1/13/17	Sun 2/26/17	Mon 2/27/17
Button	4 hrs	Mon 1/16/17	Mon 2/27/17	Tue 2/28/17
Icon	2 hrs	Mon 1/16/17	Tue 2/28/17	Wed 3/1/17
Testing	**21.19 days**	**Mon 1/16/17**	**Thu 3/2/17**	**Fri 3/31/17**
Bug Fixes	3 days	Mon 1/16/17	Thu 3/2/17	Sun 3/5/17
Finalize Paper	**18.19 days**	**Thu 1/19/17**	**Sun 3/5/17**	**Fri 3/31/17**
Work on Main Content	3 wks	Mon 1/23/17	Sat 3/11/17	Fri 3/31/17

Explain Implementation	2 days	Thu 1/19/17	Sun 3/5/17	Wed 3/8/17
Explain Results	3 days	Fri 1/20/17	Wed 3/8/17	Sun 3/12/17

Helpful Design Decisions

Build & Deployment Stack

The build stack proved to be a major benefit to the development process due to the way it assisted the developer in a one click deployment of the client application. A command line interface was used to execute the command "npm run-script deploySF" which used the build stack to package the code, build a zipped file and deploy directly to Salesforce. The dependencies were easily manageable by Webpack and NPM which helped simplify build management. The combination of the two proved to be a major proponent to the build and deployment stack. Additionally, an available Webpack feature that was not used is automated building and deployment. If implemented, the automation is able to detect changes and run the deploy command automatically. This feature might prove handy in future development efforts.

Graphing and Charting Library

The graphing library proved to be a good choice because it was easy to implement, extend and manage. FusionCharts has several additional plugins which can be added to provide extensions into gauges, themes, maps, data formats and many other features making it highly useful. Additionally, a wrapper was available for NodeJS enabling a simple integration into the client design stack and allowing the build stack to easily manage the library as a dependency.

Managed Package

Salesforce's managed packages was a great way to build a package for easy distribution to other Salesforce environments. Through a package URL, this project deployed to several

Salesforce environments with minimal setup. There's no doubt that Salesforce enables innovators and developers an easy medium for building, deploying and managing products on the Force.com platform.

Exceeded Expectation

Graphing Library

The graphing library provided an easy integration and beautiful design enhancements to the rendered graphs. While the default theme was used in the implementation, FusionCharts provides many themes to choose from to help suit the needs of a particular project. The rendering of a graph occurs at various spots throughout the application's life cycle. The majority of the rendering appears when a user makes a selection or change to the graph properties but the rendering was easily controlled through a series of methods and could be accurately predicted with the help of Redux. Lastly, the addition of the 3-dimensional charts was easier than expected. Even though another dimension is introduced, there were no other properties that needed to be set to make the charts functional.

Build and Client Stack

The build and client stack worked very well together and are worth mentioning as an item. The build stack was extremely easy to put together and provided such a simplistic approach at managing the client stack without a complex setup. In fact, once the setup was configured it could be relocated to another machine and development could be continued with a very little amount of lost time. This benefit alone provides an extreme amount of value for a developer especially for one who might jump from project to project.

Below Expectation

Very rarely does a project meet every expectation outlined during the design phase. As such, this section aims to explain the parts of the project where the implementation did not meet the expectation of the original design and planning.

Field Mapping

One desire of the project was to build the application in such a manner that additional graph and chart properties could be added with relative ease. As the implementation phase was underway, it became apparent that additional properties of the graphing library should be potentially broken into several components, serialized in JSON and stored within several fields on the custom object. Instead, the design phase insisted each property having its own field within the database object. The problem with each property having its own field lies in the specificity in the naming and mapping of fields on the client side of the project. Essentially the client is responsible for a large amount of data mapping when it shouldn't care what the database model looks like. Instead, the client should be groups of properties submitted to the REST endpoint and allowing the server endpoint to consume and organize properties. The next version of this project should include design field orchestration because it would enable the easier implementation of: new graphing and charting libraries, loosely coupled client side property handling and more abstract processing (i.e. storing and retrieving) of graph customization.

Multiple Implementations within Environment

Based on the current design, the project could be implemented throughout multiple pages within a single Salesforce environment. However, there is one caveat. Each graph is saved per user within a Salesforce environment which means each additional implementation of the project within an environment would not retrieve graphs and charts per user and per page. Instead, each

implementation within an environment would retrieve all graphs and charts the user has saved previously. In the next version, a modification needed to be made which saved each graph based on the page implemented by the project.

Salesforce Report API

During the proposal phase, it was expected that both standard and custom reports could be retrieved by Salesforce's Report API but this was not possible. The Report API would only retrieve custom reports that were available to the user. This fact was unfortunate but did not hinder the development of the project. It only hindered the full scope of report availability to the end user.

It was also expected that standard reports would provide a basic set of test reports to the end user after the project was deployed because the reports interact well with the standard data provided by Salesforce. Overall the features that were expected would have helped supply a quicker way to test the project within a new Salesforce developer environment. Instead, custom reports had to be created and a free published package named *Populate* on the AppExchange (Northwest Cloud Solutions, LLC, 2016) was used to generate larger amounts of data than what comes standard within a new Salesforce developer environment. In a typical, customer owned Salesforce environment, this would not be the case because custom and personal reports would already exists thus allowing this project to easily pull in those reports.

Design Modifications

User Interface

The original design enabled the selection of a couple parameters with the intention of allowing the user to enable various properties that are built into Salesforce graphs. However, the design and functionality decision was made further into the implementation phase to evolve this feature so a user can manually set the properties of the chart. For example, if a user wants to build a graph showing the total number of expense reports submitted per person and the amount for their department then the X axis can be customized to say "Total: How Many They Reported" and the Y axis could be set to "Total: How Much They Spent". If a label is not selected in the "Customization" section then the label does not appear within the graph.

Architectural

There were only two design modifications made overall. The first was to add Redux to assist React with managing the state of the application. React's intrinsic state manager proved to be difficult to manage and hard to predict. As it turned out, the need from the application was something more extensible and controllable which is where Redux comes into play.

The second modification was to the REST API manager. The original design had one interface interacting with both graphs and graph types. Based on the data passed into a URL parameter, the manager would respond with a list of reports. It was realized during the implementation phase that a more appropriate design would be to separate the GET requests from the RESTful interface and allow one namespace to have access to the graphs stored in the database, and a different namespace to return a list of enabled graph types based on the Salesforce environment.

Redux

React allows a developer to manage the state of the application internally but is limiting. The recommendation for using React's state is to pass from the instantiation point and let the state propagate throughout the components. While useful, AJAX is difficult to manage using React's basic state in situations where a group of AJAX calls must complete before a component renders.

After a re-evaluation, it was determined that Redux provided an additional set of useful tools, namely middleware, which enables a developer to interact with the state of the application after actions. AJAX calls are easily manageable in Redux by intercepting a dispatched action and performing calls based on the actions. While AJAX management is a type of middleware which can be implemented, it is only one example of middleware and the one specifically used in this project. Simply put, middleware is a mechanism to intercept actions and perform additional logic based on the action. Redux provides an example of building a logger middleware which captures and logs the state based on all, or specific, actions. The logger example also shows how one can output state comparisons to the console or, for example, be expanded to log to a remote database for production storage.

Superagent

Originally, the intention for an AJAX library was to use jQuery but the design change to use Redux prompted for the contemplation of a different means for REST communication because jQuery was no longer necessary. Superagent was chosen to provide the communication needed as it was simple to implement and interacts very well with Redux. The library was also smaller in size to jQuery - this consideration alone has packaging and performance implications - and provided better cross-browser support than other compared libraries.

REST API Manager

The REST API built in Salesforce was originally designed to return all lists such as graph types and graphs, but it was determined that graph types would be better fitting in its own class, and thus its own namespace. Moving graph types into its own namespace provides several benefits, namely, a separate service namespace for RESTful requests, extendable for future features and avoid blending service purposes. As an example, a GET call to the "/graphTypes" namespace would yield a list of graph types available to the Salesforce environment, and a GET call to "/graphs" would yield a list of saved graphs, if any, previously created by the user. This type of logical separation simplifies the RESTful interface and enables expansion in the future.

Managed Package Deployment Procedure

The outline below shows the process used to deploy and setup the project to another Salesforce environment.

Create Package (Figure 8)

To begin building a new package, the following must be done:

- Inside Salesforce, as a Salesforce administrator, navigate to "Setup > Create > Packages".
- Click "New", and fill-out the package information. Be sure to click "Managed" to designate the package as a managed package; otherwise, the package will be unmanaged and have various restrictions on deploying code.
- Click "Save". Once saved, the administrator will be redirected to the package where all the necessary components (i.e. objects, fields and resources) can be included within the package. The components included in the package will be deployed to any destination environment.

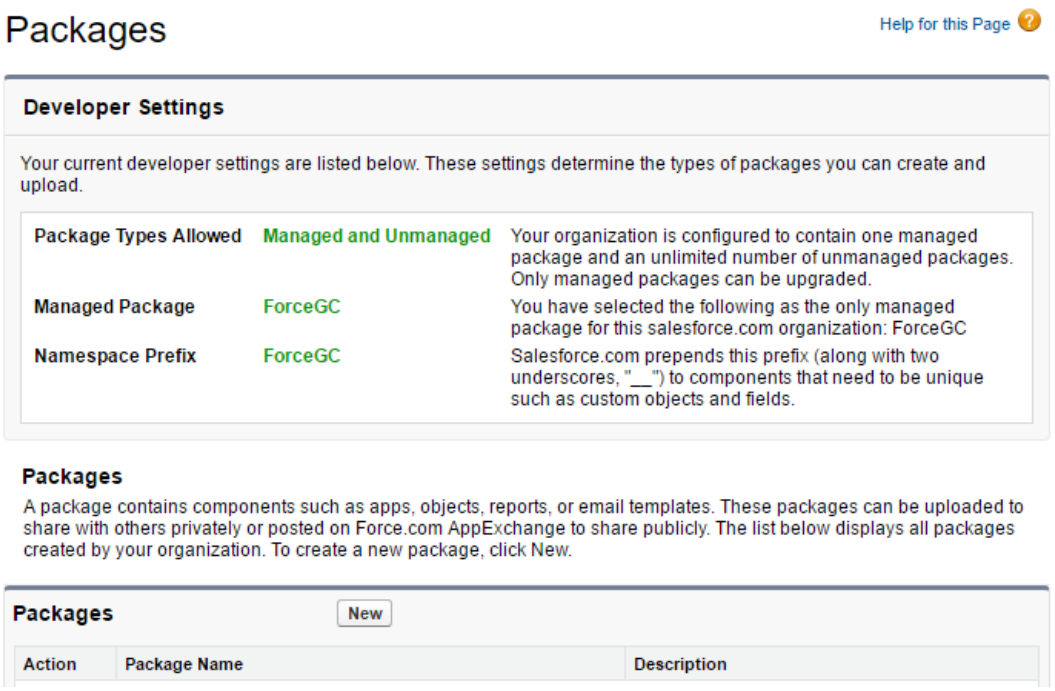
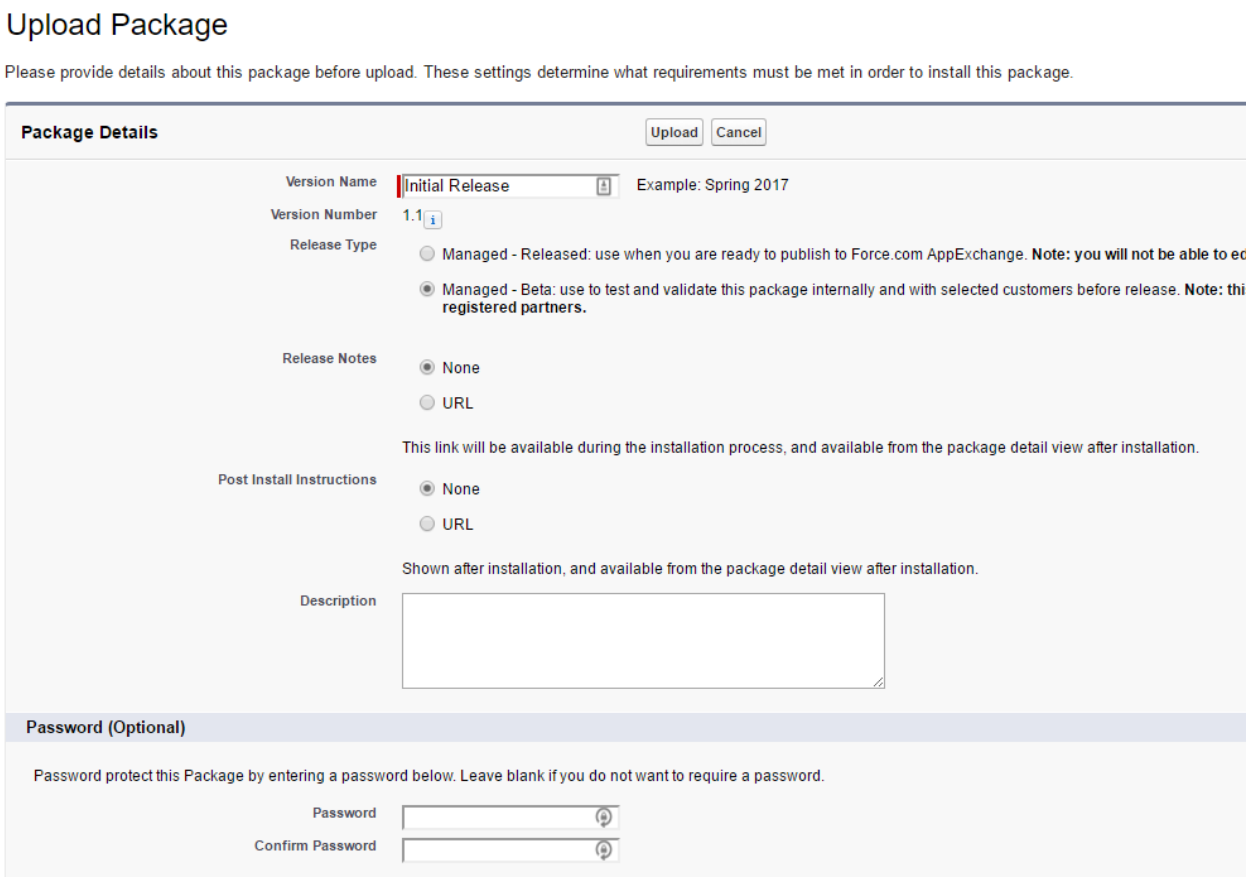


Figure 8: Create New Package

Upload Package (Figure 9)

A screen will appear, as shown in Figure 9, allowing the administrator to specify certain features like the type of managed package (example, managed package for beta testing or managed package for publishing to the AppExchange), release notes, post install instructions, password protection, etc…. Once all of the necessary components are included, click "Upload" to create a new version of the package.

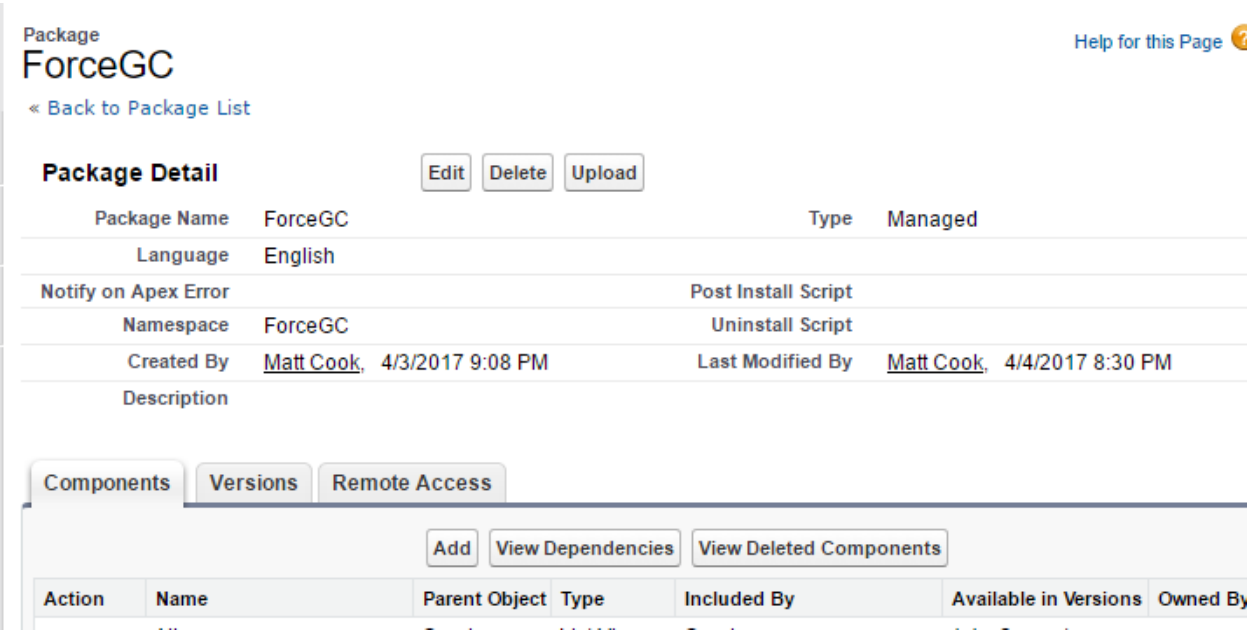
Figure 9: Upload Package Configuration



Add Package Components (Figure 10)

Once the package has been created, all of the components that need to be deployed can be added at this point in the process.

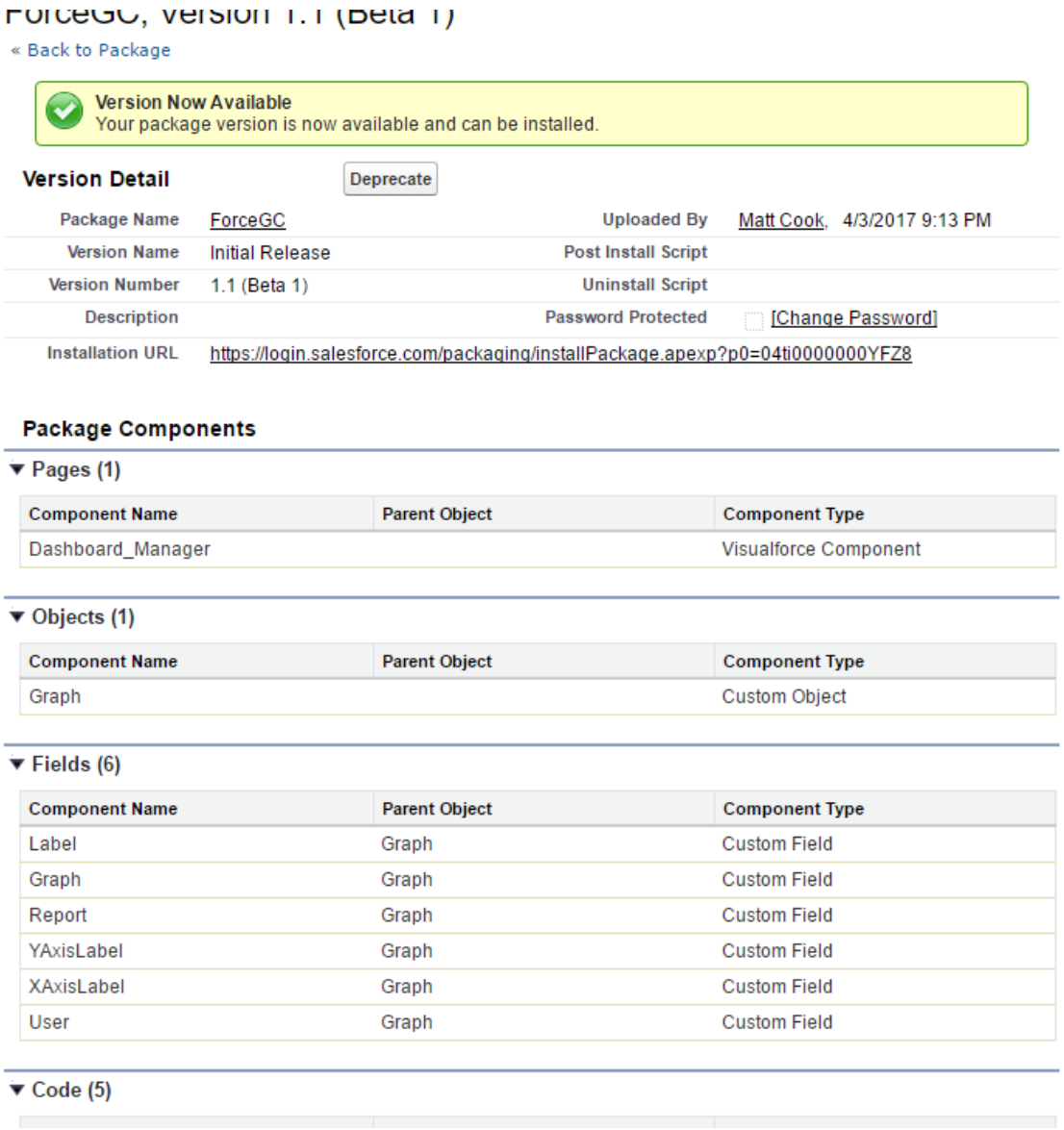
Figure 10: Add Package Components



Copy Package URL (Figure 11)

Once the package has been uploaded, an "Installation URL" is created and shown which is the gateway to deploying packages in other environments.

Figure 11: View of Upload/Built Package



Log into Destination Environment (Figure 12)

For the purposes of development and experimentation, developer orgs are created using the following URL in Salesforce: https://developer.salesforce.com/signup. A Salesforce Developer Org is only intended for development and testing. In this case, we will be creating a new org to test the managed package deployment process.

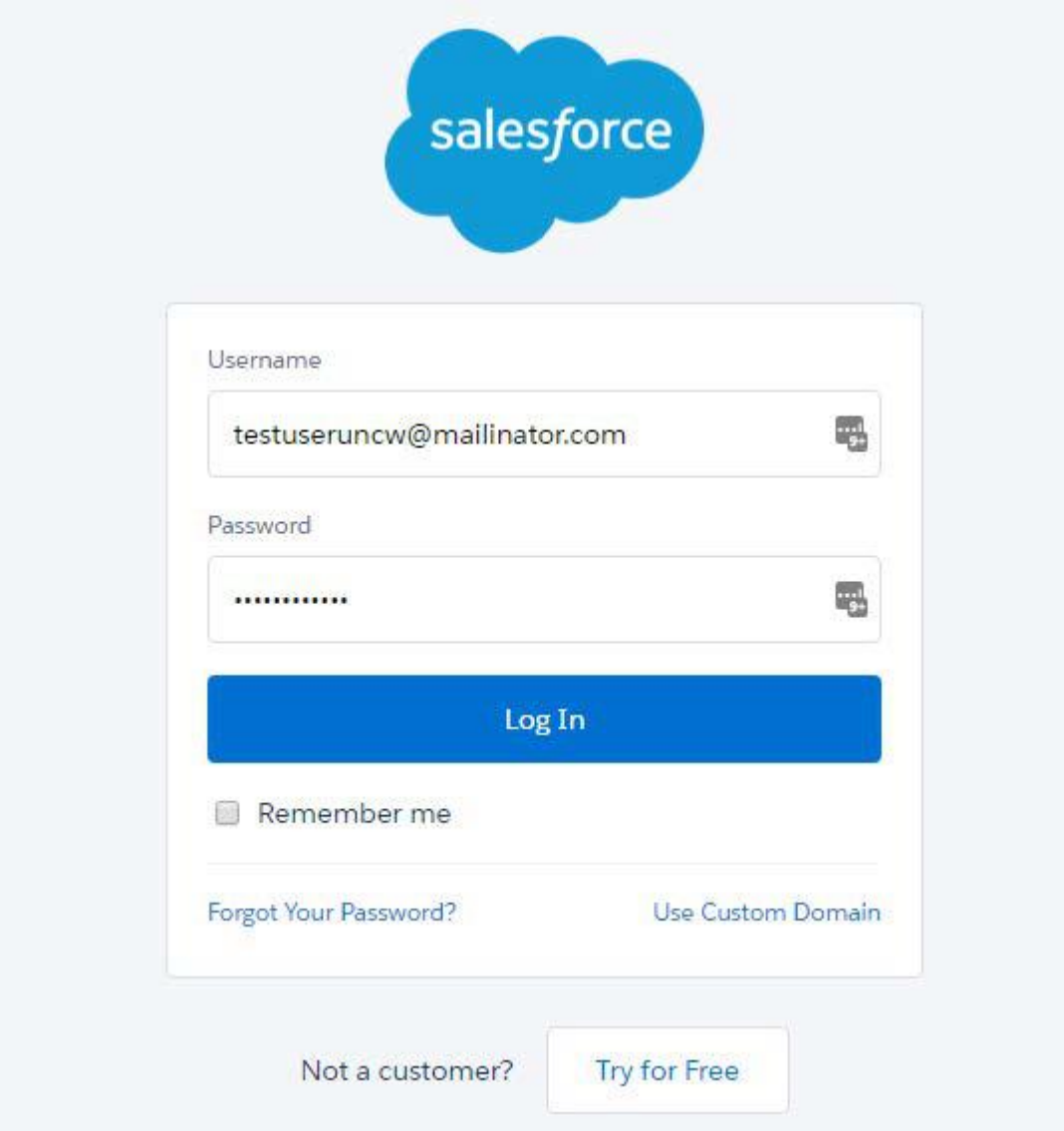


Figure 12: Log Into New Org

Install Package (Figure 13)

The URL must be copied and pasted into the browser because the project is not available on the AppExchange. For an app to be published through the AppExchange, it must be submitted to Salesforce for review and approval. Since submitting and getting an application onto the AppExchange is not part of the scope of this project, we will need to install the managed package through the URL. The image below shows the screen prompting the administrator to select the users who should have access. For the purposes of most installations, "Install for All Users" is acceptable. Next, click the "Install" button to complete the installation of the package.

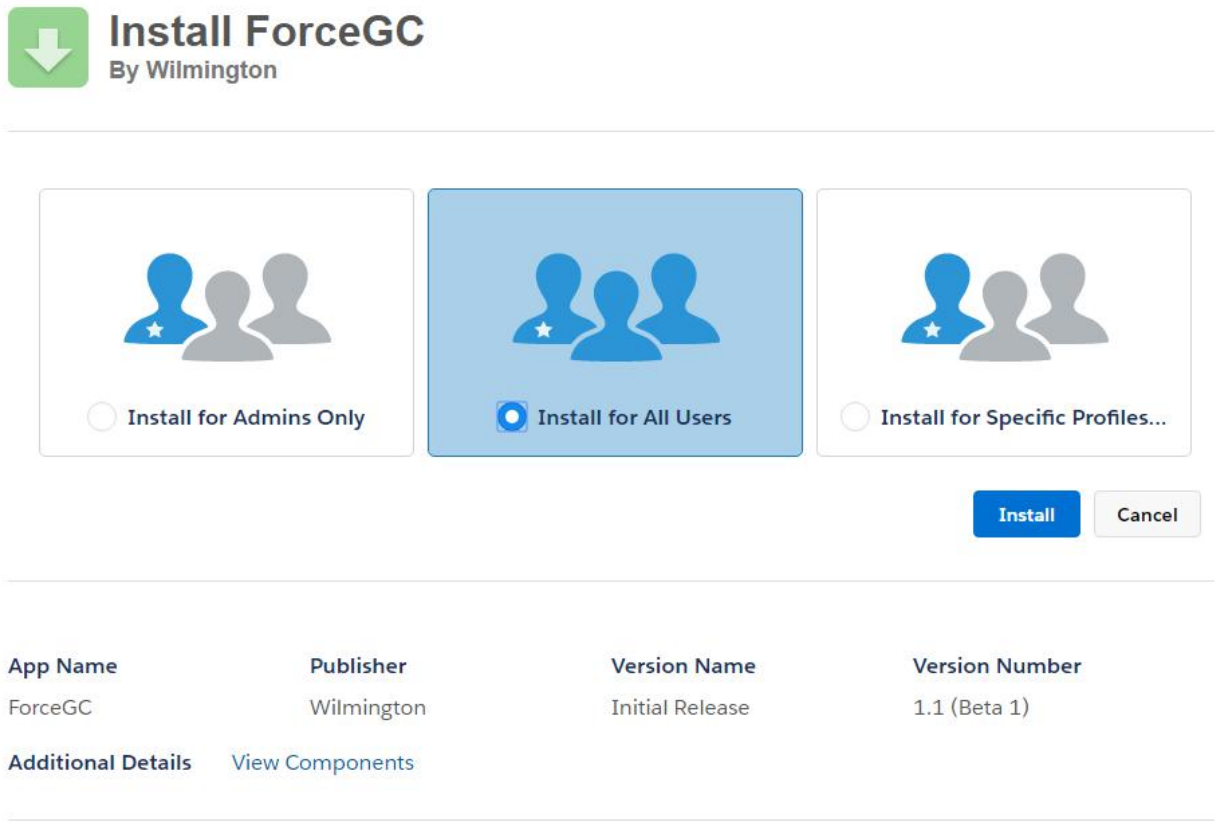


Figure 13: Install Package

Post Deployment Environment Configuration

The following walk through outlines the procedure for configuring an environment to use with the managed package previously deployed.

Create Visualforce Page (Figure 14)

Once the managed package has been successfully installed into a Salesforce environment, the next step is to create a Visualforce page and add the component which loads the app. To create a new Visualforce page inside of Salesforce:

- Navigate to “Setup > Develop > Visualforce Pages”.
- Click the “New” button and give the page a name.

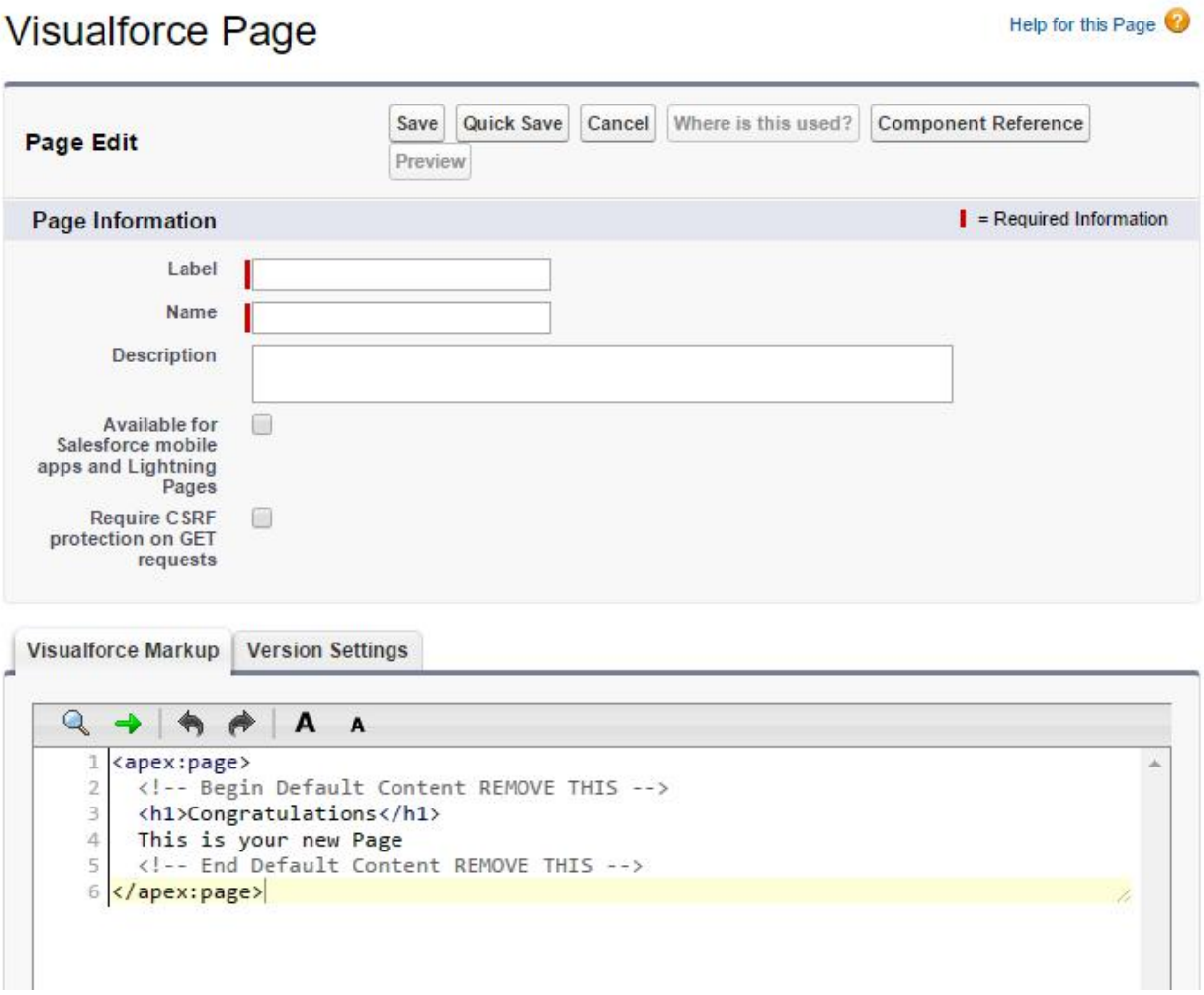


Figure 14: Create New Visualforce Page

Add Component to Page (Figure 15)

Next, add a reference to the component that was added in the managed package. The code appears as follows where the *ForceGC* is the namespace of the package assigned during its creation, and "Dashboard_Manager" references the specific component inside the managed package.

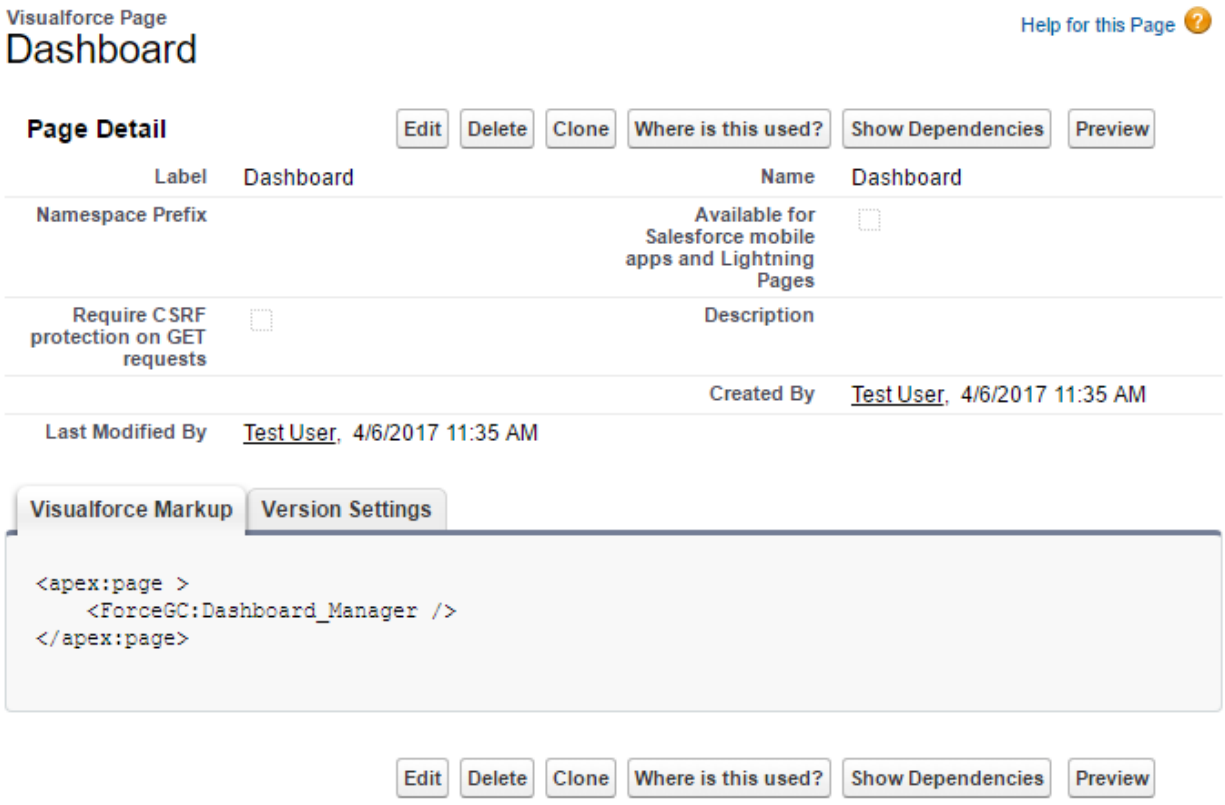


Figure 15: Visualforce Page Content

Feedback

Feedback is a very important aspect of any project as it gathers valuable information from users with the goal of enhancing and improving an idea. A small focus group was organized to assess the basic usability of the project and determine any future improvements suggested by users outside of the project. Each user will receive a set of credentials to a Salesforce environment and be provided with a survey to complete. The survey will help gather the users first impressions of the project but also recommendations through a list of likert items and open-ended questions.

Environment Configuration

Available Data

The data installed in each Salesforce environment is a combination of standard data included with developer provisioned environments, and data from a managed package called *Populate* (Northwest Cloud Solutions, LLC, 2016). Populate is a free managed package which installs a large quantity of data. Figure 16 displays a graph showing the data in the focus group's environment and shows the number of records per object (and name of object).

Current Data Storage Usage

Record Type	Record Count	Storage	Percent
Opportunities	1,031	2.0 MB	48%
Contacts	520	1.0 MB	24%
Accounts	261	522 KB	12%
Leads	261	522 KB	12%
Cases	26	52 KB	1%
Campaigns	4	32 KB	1%
Solutions	10	20 KB	0%
System Streaming Channels	2	4 KB	0%
Graphs	2	4 KB	0%
Last Used App	1	2 KB	0%
Tasks	0	0 B	0%
Photos	2	0 B	0%

Figure 16: Number of Records in Focus Group Environment

Available Reports

Following are the available reports created for the focus group. Each report is custom which means it was created by the project organizer. The reports are intended to retrieve a variety of information pertaining to Opportunities, Accounts or Leads. Each object has data available to it as explained in the *Available Data* section of Chapter 5.

- Won Opportunities – returns a list of opportunities that were either lost or won.
- Opportunity Forecast – returns a list of opportunities and their stage (Prospecting, Qualification, Needs Analysis, Value Proposition, Decision Makers, Perception Analysis, Proposal/Price Quote, Negotiation/Review, Closed Won or Closed Lost)

- Accounts by State – returns a list of accounts within each state.
- Leads by Owners – returns a list of leads by their owners.

Available Graphs and Charts

2D & 3D Column Charts (Examples)

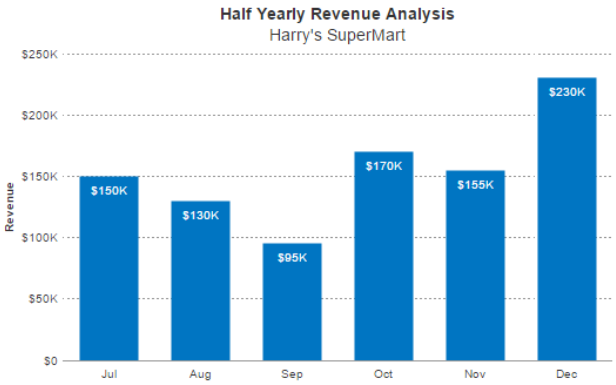


Figure 17: 2D Column Chart

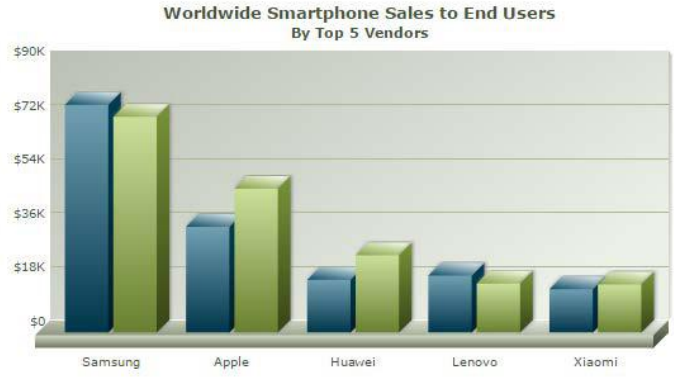


Figure 18: 3D Column Chart

Line Chart (Examples)

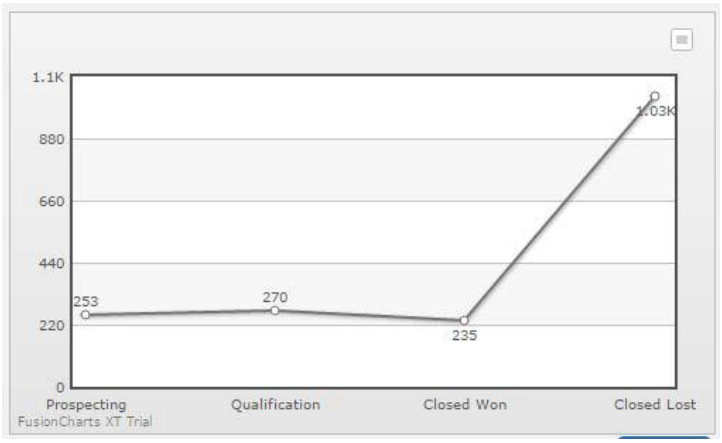


Figure 19: Line Chart

2D & 3D Pie Chart (Examples)

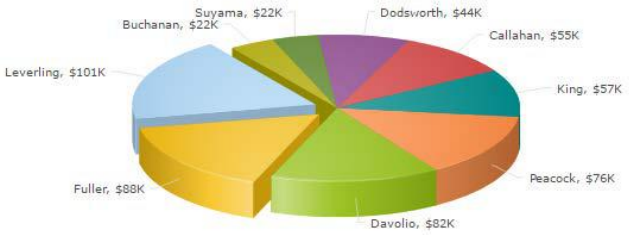


Figure 20: 2D Pie Chart

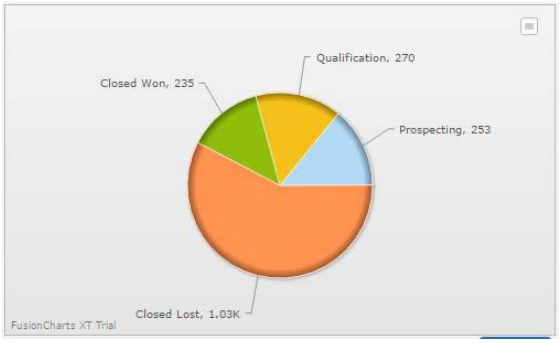


Figure 21: 3D Pie Chart

Additional Features

The features in Figure 22 were enabled for the users to perform basic customization for each graph or chart. Each feature is a text field enabling the user to type appropriate names for each label.

- Header
- X Axis Label
- Y Axis Label

Figure 22: Header Labels UI

Questions Asked

To help assess the basic usability of the application, six likert and six short-answer questions were presented. The questions were distributed and observed through Google Forms for easy access and management. The likert items use five radio buttons where a 1 shows the user *strongly agrees* and a 5 says the user *strongly disagrees*. Likert questions help provide a basic understanding of the user's assessment. Short-answer questions, on the other hand, help provide slightly more detail to the likert questions.

Introduction

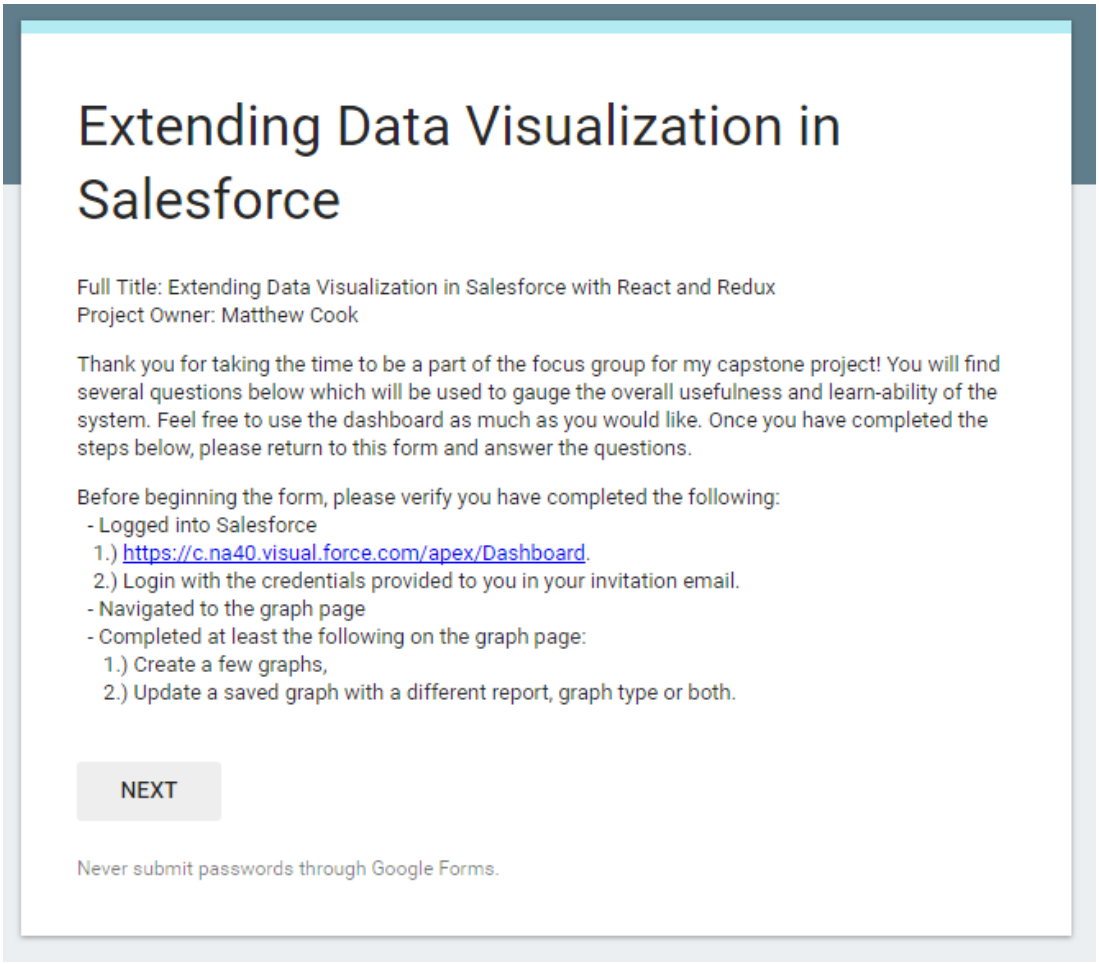
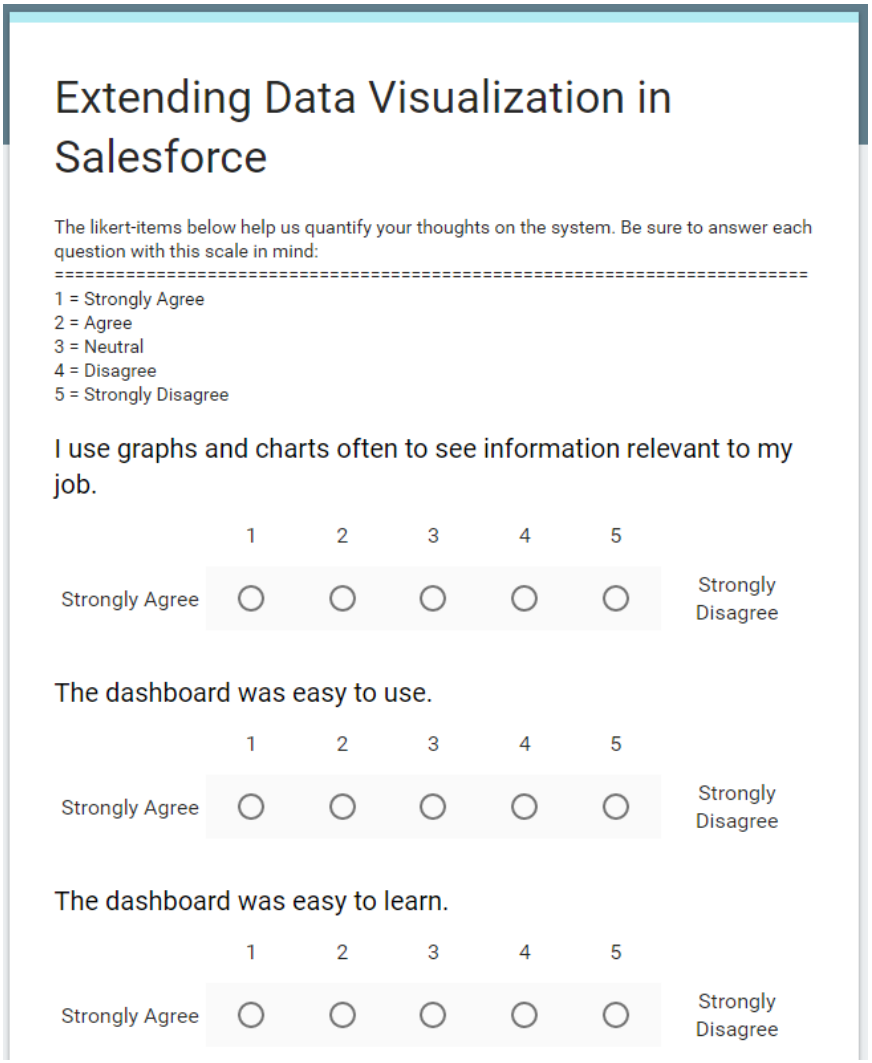


Figure 23: Survey Intro Screen

Likert Items



Extending Data Visualization in Salesforce

The likert-items below help us quantify your thoughts on the system. Be sure to answer each question with this scale in mind:

==========

1 = Strongly Agree
2 = Agree
3 = Neutral
4 = Disagree
5 = Strongly Disagree

I use graphs and charts often to see information relevant to my job.

	1	2	3	4	5	
Strongly Agree	○	○	○	○	○	Strongly Disagree

The dashboard was easy to use.

	1	2	3	4	5	
Strongly Agree	○	○	○	○	○	Strongly Disagree

The dashboard was easy to learn.

	1	2	3	4	5	
Strongly Agree	○	○	○	○	○	Strongly Disagree

Figure 24: Survey Likert Items

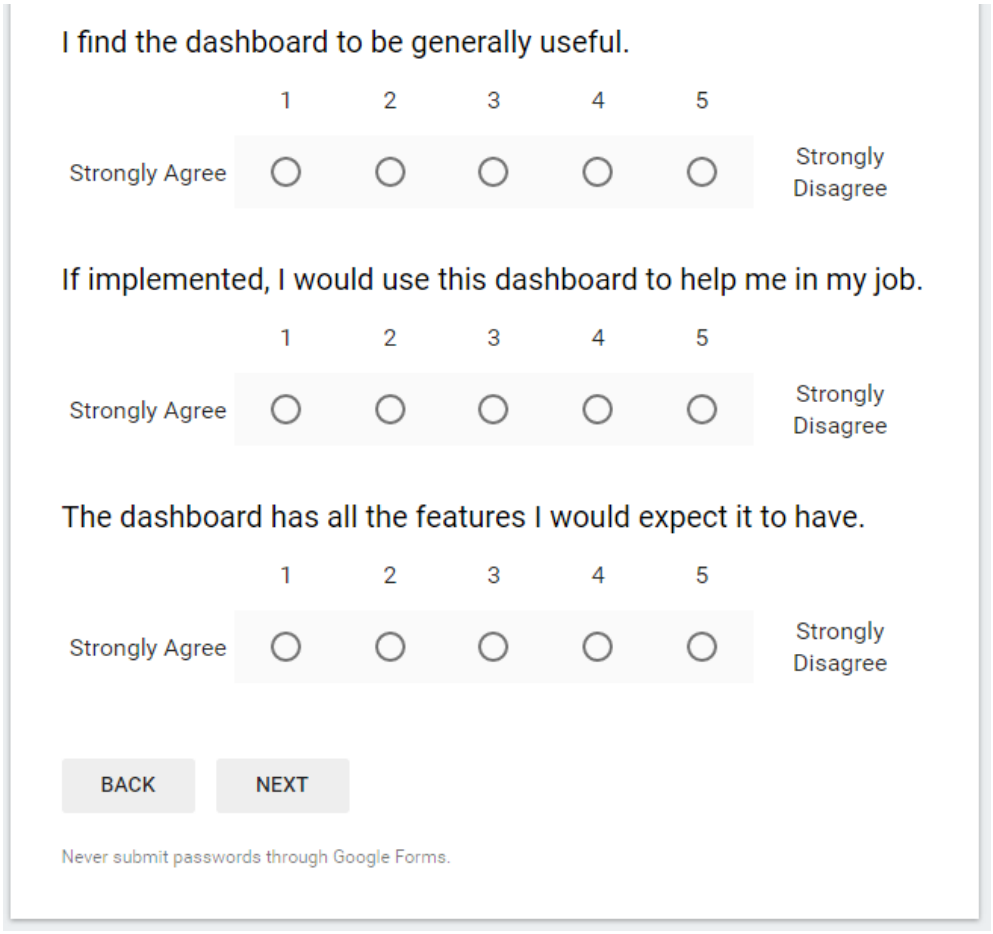


Figure 25: Survey Likert Items (cont.)

Open-Ended Questions

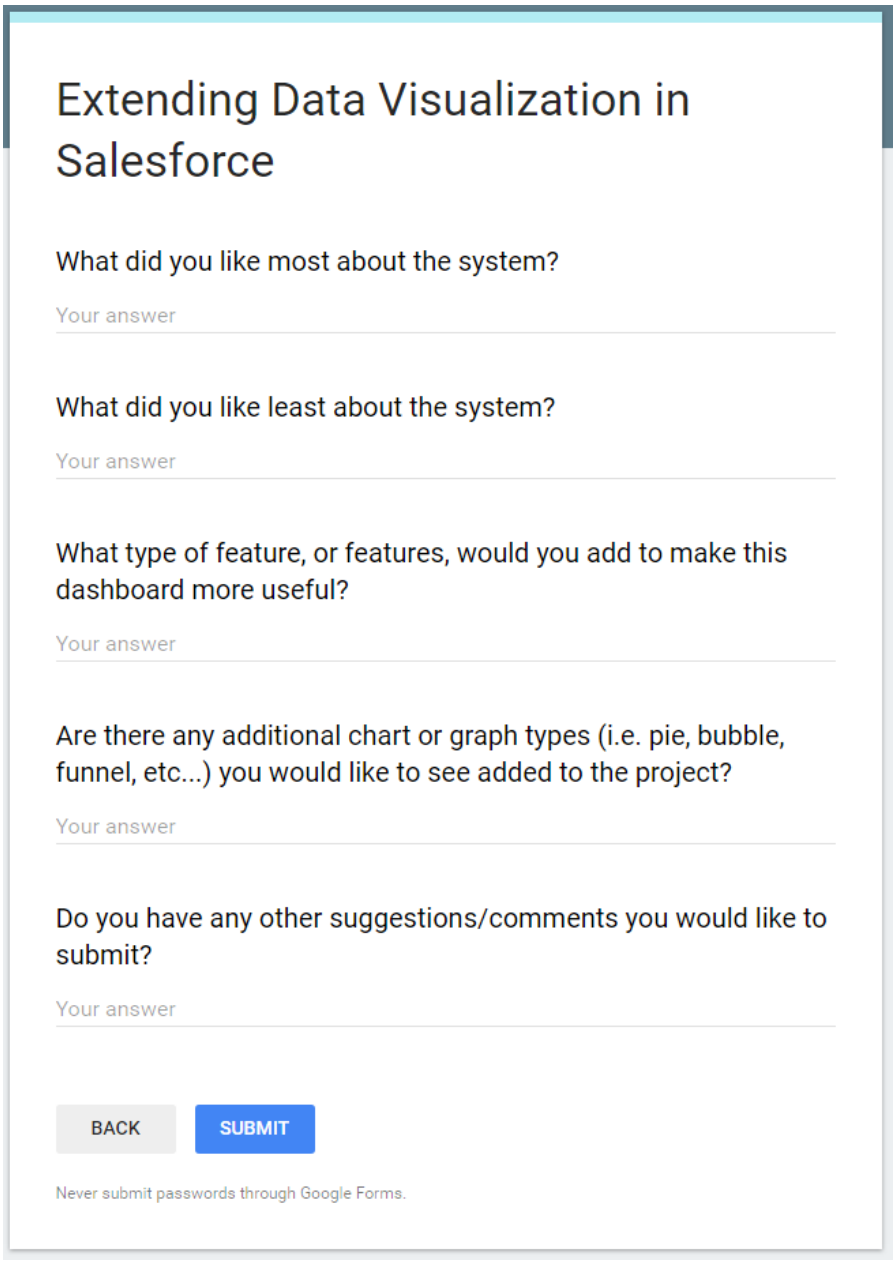


Figure 26: Open Ended Questions

Survey Results

In the survey, five people were asked to log into Salesforce and interact with the application by creating, editing and deleting graphs within the dashboard. Once the users have performed the actions listed, they were asked to fill out the survey and provide their opinions on their experience. The results of the survey can be seen in APPENDIX W through APPENDIX BB and shows the application is generally useful, easy to learn and would be a viable replacement for Salesforce's standard dashboard.

APPENDIX W shows a question presented to the user and requests their opinion on the general ease of use of the dashboard. The users responded by answering every option except "neutral" even though it was expected to see results closely grouped on one end or the other. A wider spread of responses could suggest the users misread the likert scale or did not have a good context of the system. However, it's interesting to see the typed responses in APPENDIX BB because all five users commented on the simplicity and ease of use of the system which indicates all the responses should have been in the ones and twos of the likert scale.

Figure 27 shows responses to an open-ended question that will be very valuable in determining future enhancements to the application. Several users expressed the usefulness of having a feature to allow one to modify the data source "on the spot" without the need of Salesforce reports. Other users suggested having the ability to choose the color, further drill-down and filtering. The graphing library chosen for this project would allow each of these features to be implemented and with some, like color choice, to be completed sooner than the others.

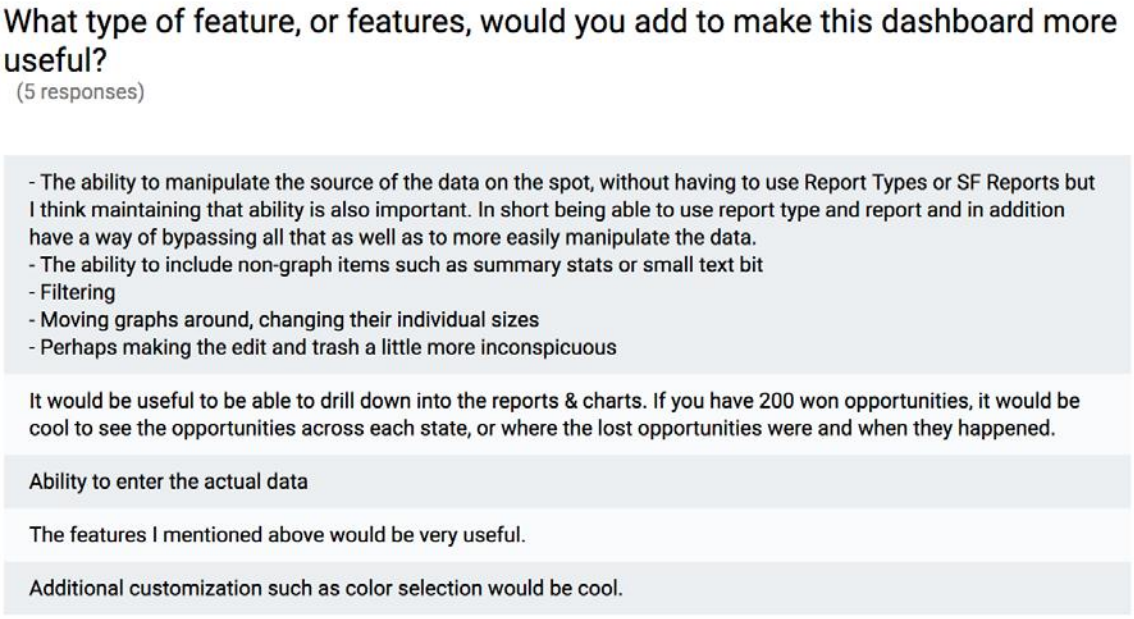


Figure 27: Suggested Features

All of the questions asked in the survey allow the future of the project to be determined and tailored based on user needs or wants. The short answer questions provided a level of feedback which will help the application planners build a roadmap for future releases. The way the questions were asked could provide weight to assist the decision making process of each feature mentioned as well. In other words, the frequency of a feature would provide a greater importance, and thus urgency, to include it sooner on the roadmap.

CHAPTER 6: DISCUSSION

When first starting the project, I thought the prototype effort would allow for much reusability between the user interface and JavaScript. I was able to reuse much of the HTML

designed but the JavaScript was not as transferable as I had hoped. Additionally, I had planned on using jQuery within React but learned jQuery wasn't necessary because of the close binding of each React component to the DOM. jQuery's AJAX calls were also not necessary because of Redux's middleware. I was able to implement middleware to interact with superagent and ES6 to separate the various aspects of making an AJAX call. ES6 enabled the easy creation of modules that are importable into other files.

Predictions Revisited

Learning Curve

There was certainly a learning curve with every client framework used. Prior to this project, I had no experience using React, Redux or superagent. React and Redux are very powerful frameworks especially when they are used together but Redux took me some time to learn because it required changing my mindset. I had to change my outlook on client design and development from the traditional perspective of binding requests almost directly to elements in the user interface to performing requests based on actions generated by the client. Once I was able to grasp the concept, I was able to pick up speed in terms of development. React was rather simple to break down and build the various components into modules which allowed the learning curve to be rather quick.

I'm very comfortable with Salesforce because of my experience which helped expedite my Salesforce learning curve, but the Report and Dashboard API was one that caught me off-guard. The reason it caught me off-guard was because I had expected the API to return a wider list of reports instead of just custom reports created, and shared, by users. There was very little documentation regarding retrieving standard reports which led me to believe it was possible.

The build stack was an area where I had little to no experience. I have used Node and NPM on small, personal projects but not to the extent used here. Combined with Webpack, Node and NPM proved to be very helpful during development because it was quick and compact. Not only was it useful in deployment but also in maintaining dependencies and utilizing ES6.

Optional Features

I was planning on the structure providing the opportunity to implement a dashboard and fortunately that was afforded. Because the application was using React to manage the presentation layer, building a dashboard component and letting React manage the graph component replication was quite easy. I think it's an example of leveraging a library to let it do what it does best – modularize.

Originally, I was planning on having one basic example per category: column, line and pie. While I wasn't able to extend outside of those areas, I was able to add the 3-dimensional version of column and pie. The 3D versions were pushed within a managed package by including it as static data within the appropriate metadata object.

Quick Installation and Setup

Salesforce provides many useful tools and one that, I believe, is instrumental in their success is the packaging capabilities. Managed packages innately provide a surplus of tools that provide simple deployment and management of packages. Everything from initial deployment to updates, managed packages can provide extreme simplicity. This project was able to use managed packages for helping deliver the contents to a destination environment, but the second part of the project is setting it up to be used by end-users. Once the package was deployed, there were very few steps and the process was quick to setup within an environment. Section 0 and 0 outline the specifics for both package deployment and environment configuration respectively.

Limitations

After implementation, there was one dominant limitation that was discovered in the application. The issue was the inability to configure the application across multiple pages within a Salesforce environment to allow each page to function independently. In other words, if two pages were created and they both used the application then a graph created in one would appear in the other. The suggested resolution is to add a unique identifier for each page so the application stores graphs based on the page and the user instead of just the user.

Future Work

Additional Charts and Graphs

I would like to extend the capabilities of this project to include a variety of graphs. I think the availability of graph types will provide a large amount of additional value to the end-user because it will enable them to see the data in a wider variety of ways. For example, country maps, heat maps and bubble charts are a few graphs worthy to be considered for future releases. The downside to adding too many graphs is overwhelming the user with too many options which means a few changes to the user interface might be necessary. It might be feasible to allow the user to filter graphs based on graph types or potentially offer recommendations based on the data selected.

Other Features

A feature that FusionCharts offers which I thought would be a great feature is to export all graphs at once. Due to a lack of time, this feature fell pretty low in priority but I believe it has value and can be added somewhat quickly in the future.

Advanced Graph Customization

Advanced graph customization includes a wide variety of ideas but there are two in particular which I think would prove useful: allow the customization of an individual graph's properties, and extending the capabilities of external data sources. Customizing individual graph properties means enabling a user, who is more advanced technically, to modify more extensive properties offered by the graphing library. For example, if a user wanted to change colors, messages, fill lines, alpha values or padding then a dialog box could appear which allows the user to modify or add properties they know are supported by the library.

As noted in the ***Survey Results***, one of the items submitted in the feedback is to allow the addition of a custom data source. One way to achieve this would be to allow the user to copy and paste their own data into the graph component very similarly to the customizations listed previously. The graphing library will notify the user if the data is not accurate, but one hurdle to consider is storing the information for future retrieval. For example, if the user puts a list of values into the graph component then the data source would need to be retrieved when the user returned to the dashboard.

ACKNOWLEDGEMENTS

First and foremost, I'd like to acknowledge and thank my wife. Her support and motivation provided a level of inspiration throughout the program but also at times when I was most exhausted. She has picked up my slack in areas of life just so I could focus on the program and complete my final project. Thank you.

Drs Kline, Pence and Janicki, thank you for your patience and help throughout this project. Your advice and insight has helped me learn tremendously and hopefully this project reflects that education.

I'd also like to thank my family who provided endless support, and friends who gave assistance and advice. Specifically, Michael Kalmykolv, a UX/UI designer friend, met with me to layout and design the final mock-up used in this project. Michael spent his own time and energy discussing options and building an illustration based on his experience that might make the UI for this project better.

WORKS CITED

Abramov, D. (2016, January 4). *How do you explain the term Predictable State Container in simple words?* Retrieved from Hashnode: https://hashnode.com/post/how-do-you-explain-the-term-predictable-state-container-in-simple-words-ciizdac5300wege53dogz8aqk

Arora, S. (2016, March 5). *JavaScript Frameworks: The Best 10 for Modern Web Apps*. Retrieved from NOETICFORCE: http://noeticforce.com/best-Javascript-frameworks-for-single-page-modern-web-applications

Clark, D. (2014, March 10). *Data Visualization is the Future - Here's Why*. Retrieved from Forbes: https://www.forbes.com/sites/dorieclark/2014/03/10/data-visualization-is-the-future-heres-why/#763bb95d46fa

Cook, M. (n.d.). *ForceGraphManager Specifications*. Retrieved from SwaggerHub: https://app.swaggerhub.com/api/mwcook/ForceGraphManager/1.0.0

Data Pine. (n.d.). *Pricing*. Retrieved from DataPine: https://www.datapine.com/pricing

Douglas, J. (2015, April 3). *Build SPAS with React Reflux for Salesforce*. Retrieved from Jeff Douglas' Blog: http://blog.jeffdouglas.com/2015/04/03/building-spas-with-react-reflux-for-salesforce/

Farmer, A. (2016, February 4). *AJAX/HTTP Library Comparison*. Retrieved from andrewhfarmer: http://andrewhfarmer.com/ajax-libraries/

Ferrara, J. (n.d.). *Mavensmate for Sublime Text*. Retrieved from GitHub/joeferraro/MavensMate: https://github.com/joeferraro/MavensMate-SublimeText

Ferraro, J. (n.d.). *Mavensmate Docs*. Retrieved from GitHub/joeferraro/MavensMate: https://github.com/joeferraro/MavensMate/tree/master/docs

Ferrell, T. (n.d.). *Toastr Notification Library*. Retrieved from GitHub: https://github.com/CodeSeven/toastr

FusionCharts. (n.d.). *Tech Specs of FusionCharts Suite XT*. Retrieved from FusionCharts: http://www.fusioncharts.com/tech-specs/

Hollar, K. (2016, July 07). *Top CRM Software*. Retrieved from Capterra: http://www.capterra.com/customer-relationship-management-software/#infographic

Koba. (2015, June 28). *Write a documentation React and ES6 project by ESDoc*. Retrieved from en.blog.koba04: http://en.blog.koba04.com/2015/06/28/esdoc-documentation-for-react-and-es6/

Live Oak Bank. (2017). *About Us*. Retrieved from Live Oak Bank: https://www.liveoakbank.com/

Mashmatrix. (n.d.). *JSforce Home Page*. Retrieved from JSforce: https://jsforce.github.io/

Nadaq. (2006, 06 23). *SALESFORCE COM INC (CRM) IPO*. Retrieved from Nasdaq: http://www.nasdaq.com/markets/ipos/company/salesforce-com-inc-159140-36918

Nasdaq. (2015, 07 23). *LIVE OAK BANCSHARES, INC (LOB) IPO*. Retrieved from Nasdaq: http://www.nasdaq.com/markets/ipos/company/live-oak-bancshares-inc-803316-78750

Node.JS. (n.d.). *About Node.JS*. Retrieved from Node.JS: https://nodejs.org/en/about/

Node.JS Introduction. (n.d.). Retrieved from TutorialsPoint: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

NodeJitsu NPM. (n.d.). *package.json: An Interactive Guide*. Retrieved from browsenpm.org: http://browsenpm.org/package.json

Northwest Cloud Solutions, LLC. (2016, 12 21). *Populate*. Retrieved from Salesforce AppExchange - Populate: https://appexchange.salesforce.com/listingDetail?listingId=a0N3A00000EO5smUAD

Ray, A. (2016, April 09). *Webpack: When To Use And Why*. Retrieved from blog.andrewray.me: http://blog.andrewray.me/webpack-when-to-use-and-why/

Redux Community. (n.d.). *Redux Middleware*. Retrieved from Redux: http://redux.js.org/docs/advanced/Middleware.html

Salesforce. (n.d.). *Introducing the Force.com Platform*. Retrieved from Salesforce: https://developer.salesforce.com/docs/atlas.en-us.fundamentals.meta/fundamentals/adg_intro.htm

Salesforce. (n.d.). *Salesforce Connect*. Retrieved from Salesforce: https://help.salesforce.com/articleView?id=platform_connect_about.htm&type=0

SAS. (n.d.). *Data Visualization - What it is and why it matters*. Retrieved from SAS: https://www.sas.com/en_us/insights/big-data/data-visualization.html

Sublime Text. (n.d.). *Sublime Text 3 Download Page*. Retrieved from sublimetext.com: https://www.sublimetext.com/3

TutorialsPoint. (n.d.). *What is AJAX?* Retrieved from TutorialsPoint: https://www.tutorialspoint.com/ajax/what_is_ajax.htm

Vision Media. (n.d.). *Superagent Documentation*. Retrieved from GitHub: https://github.com/visionmedia/superagent

Vision Medio. (n.d.). *Browser and Node Versions*. Retrieved from GitHub: https://visionmedia.github.io/superagent/#browser-and-node-versions

Webpack. (n.d.). *Comparison with other bundlers*. Retrieved from Webpack.js: https://webpack.js.org/guides/comparison/

Webpack. (n.d.). *Configuration*. Retrieved from Webpack: https://webpack.github.io/docs/configuration.html

Webpack. (n.d.). *Webpack - Getting Started*. Retrieved from Webpack.js: https://webpack.js.org/guides/get-started/

Wikipedia. (2017, April 02). *Salesforce.com*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Salesforce.com

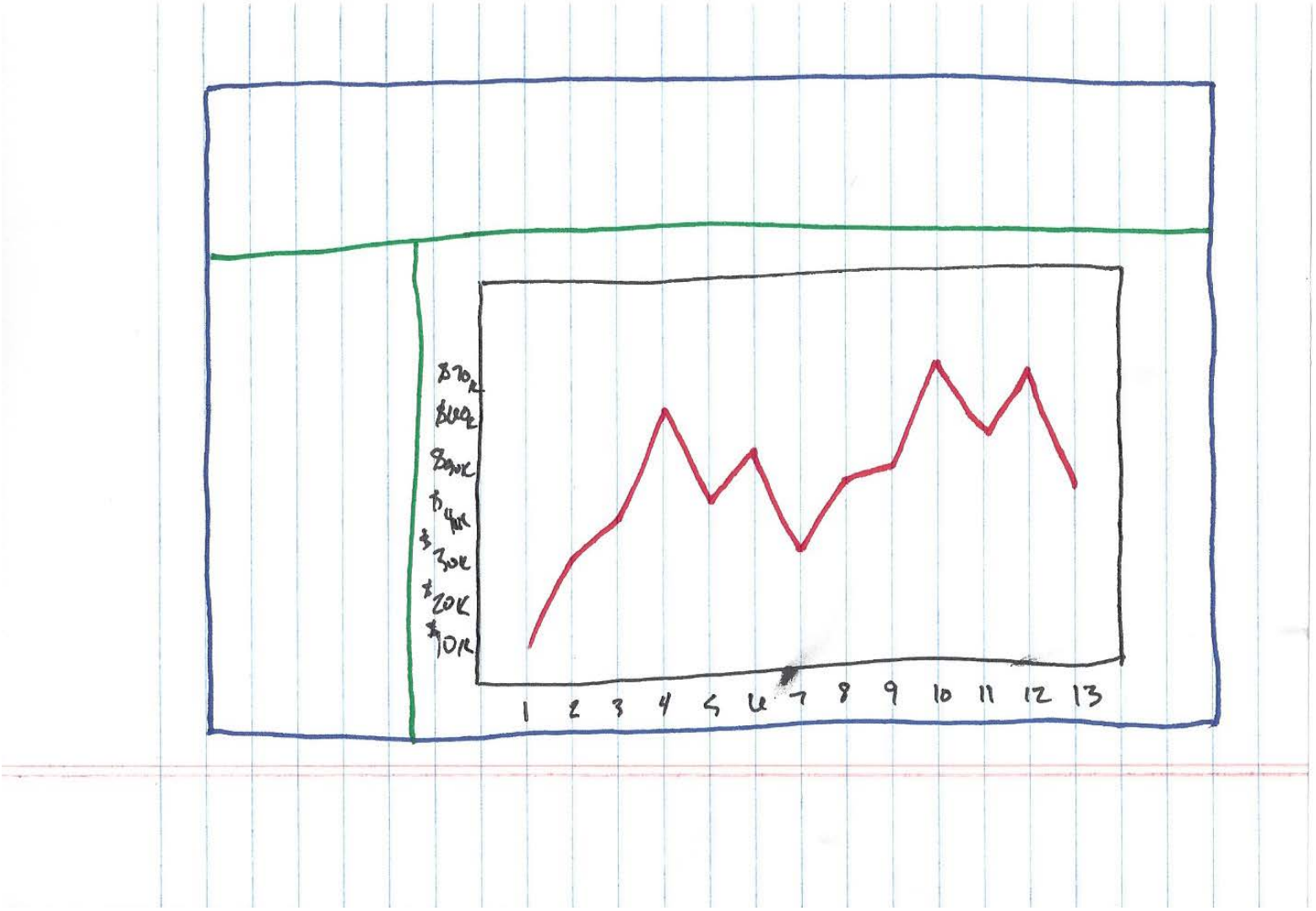
Wikipedia. (2017, April 02). *Salesforce.com#work.com*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Salesforce.com#Work.com

APPENDICES

APPENDIX A

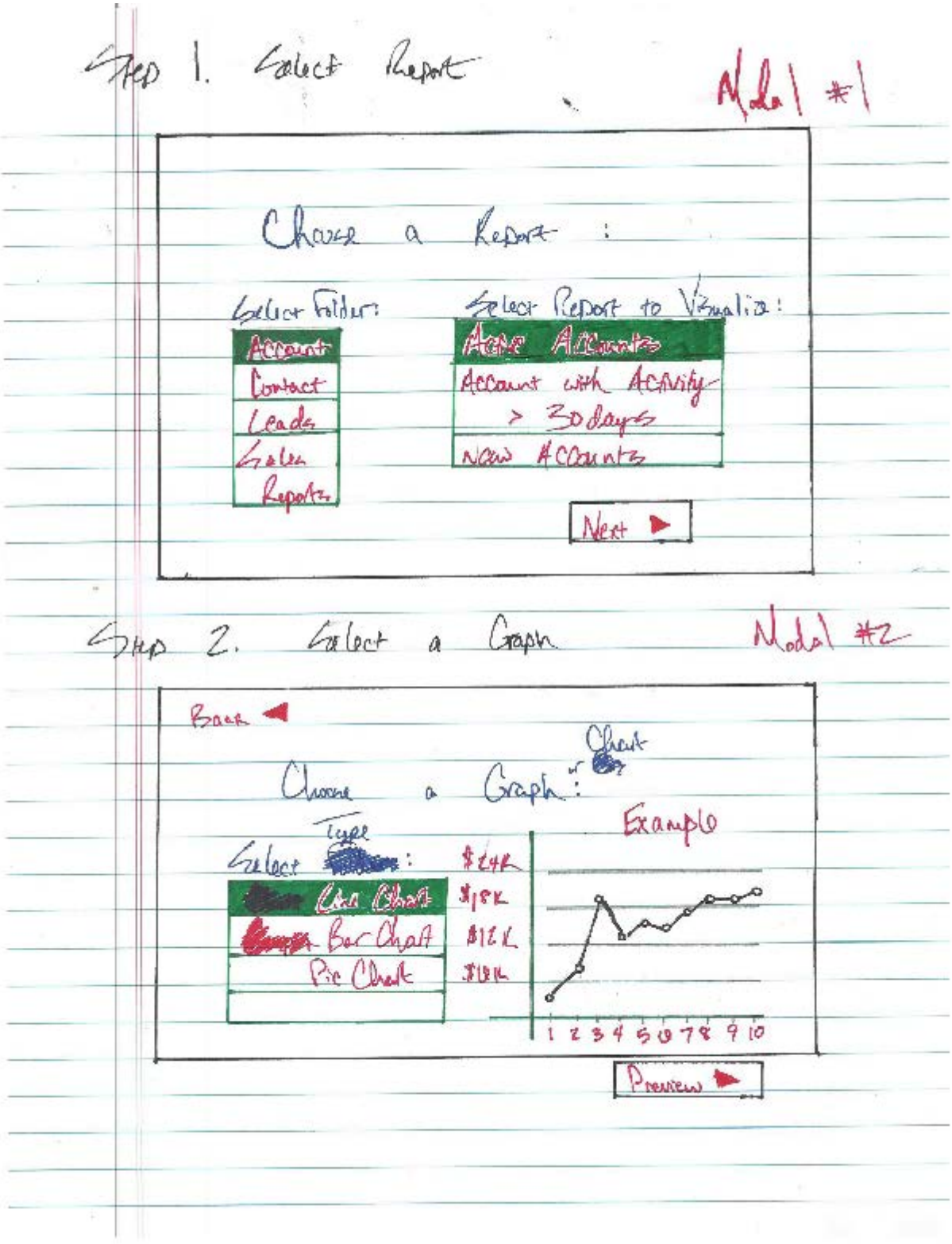
APPENDIX B

UI Mock Version 1, Graph Preview Sketch



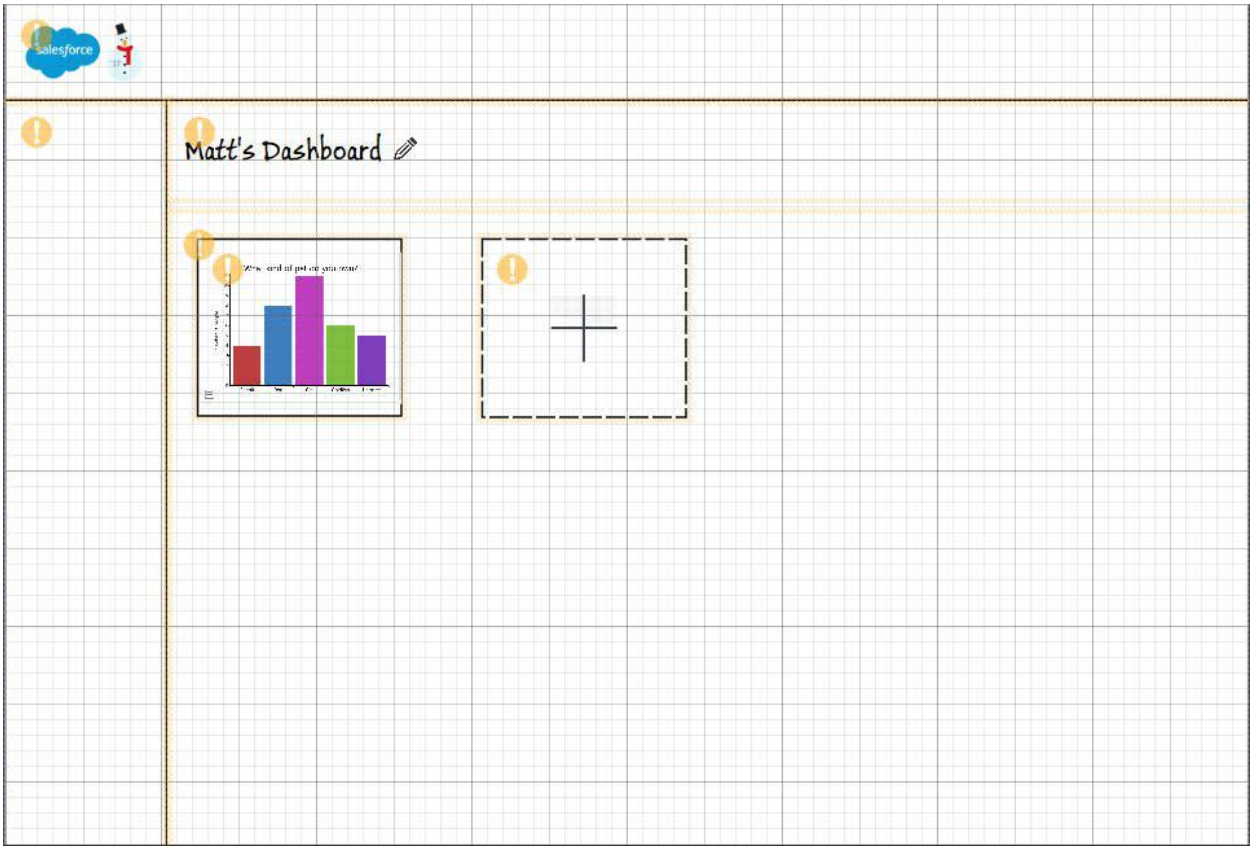
APPENDIX C

UI Mock Version 1, Graph Creation Sketch



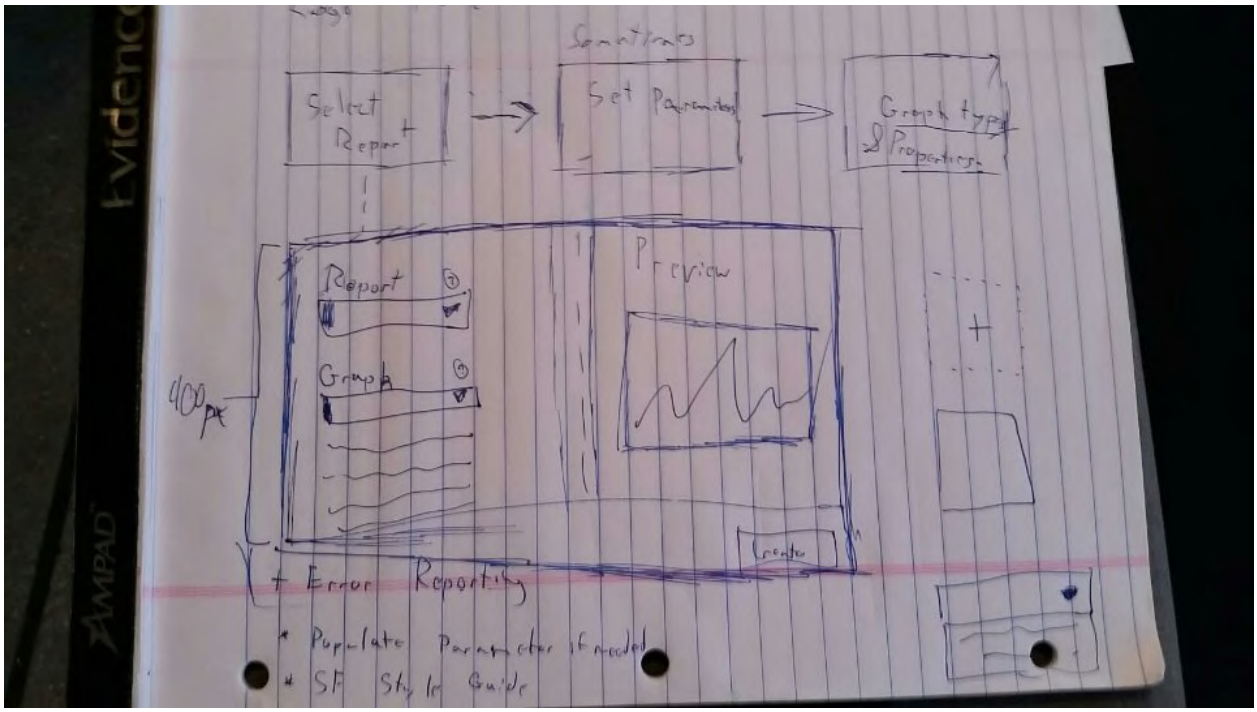
APPENDIX D

UI Mock Version 2, Microsoft Expression Illustration



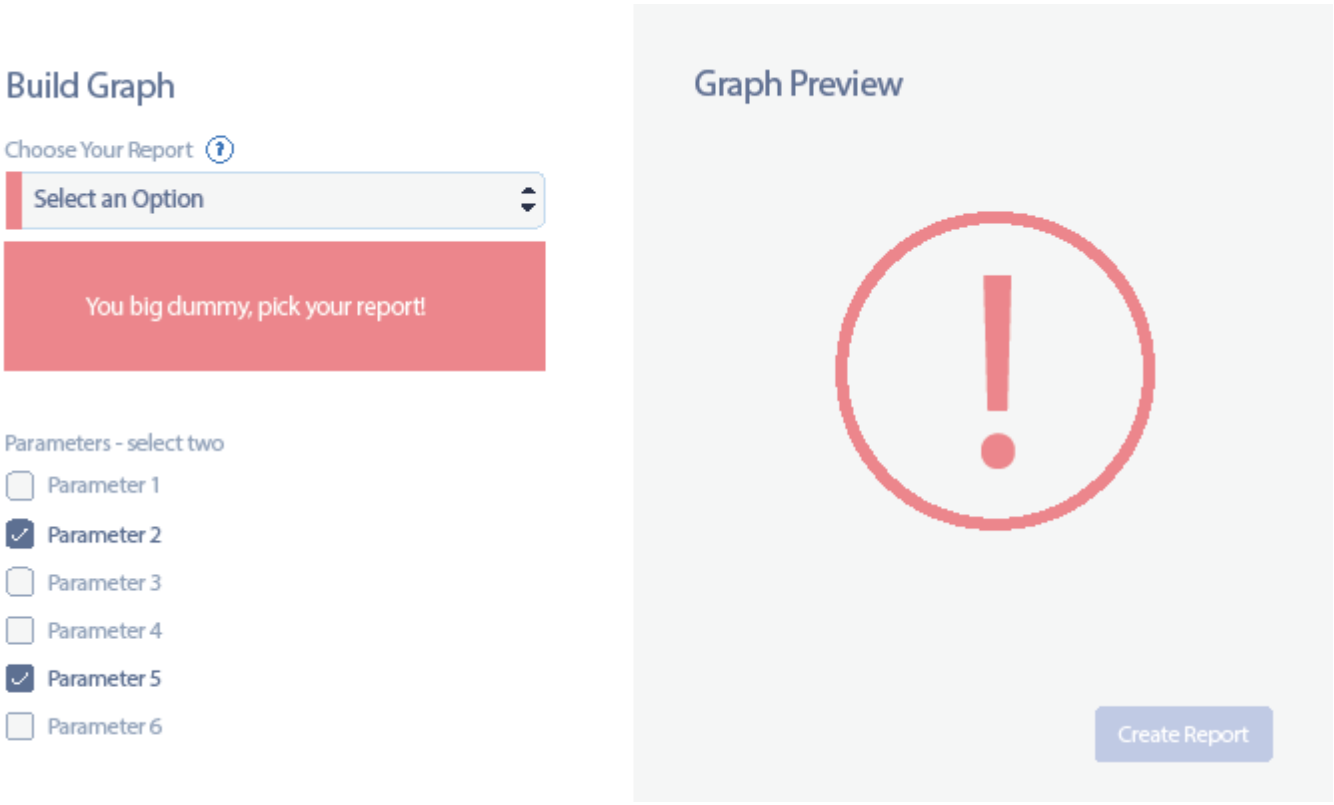
APPENDIX E

UI Mock Version 3, Sketch



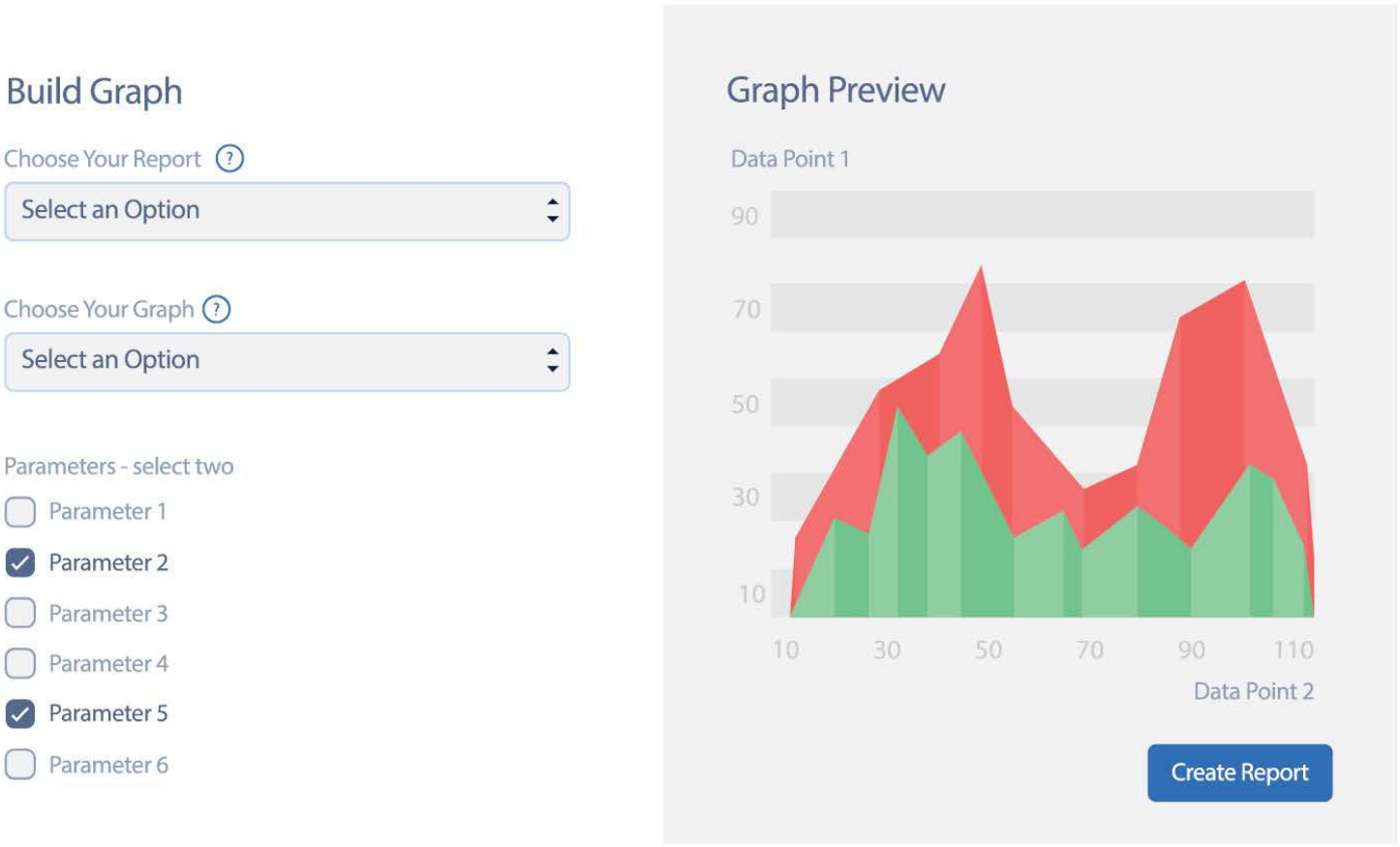
APPENDIX F

UI Mock Version 3, Page 1



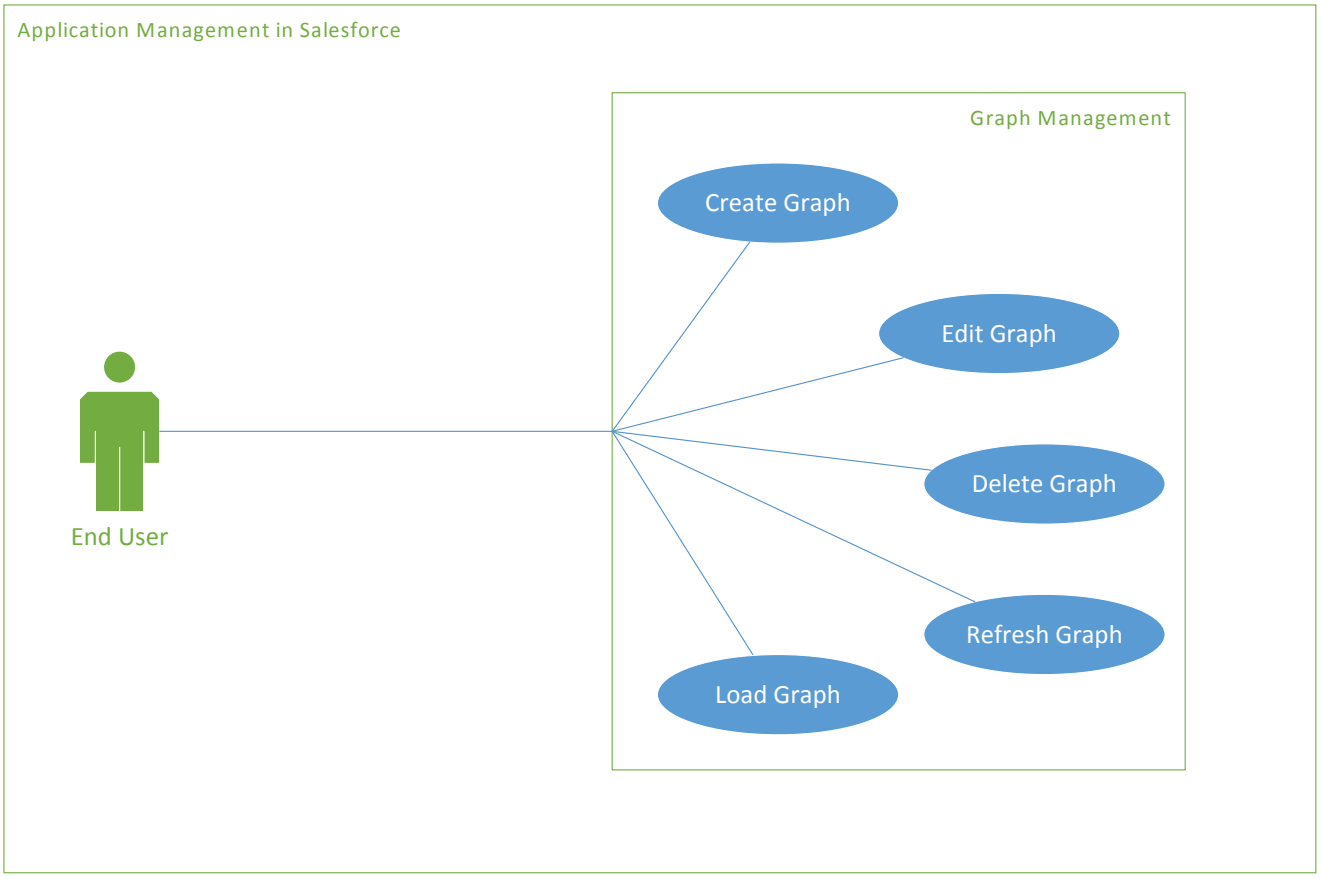
APPENDIX G

UI Mock Version 3, Page 2



APPENDIX H

Use Case Diagram



APPENDIX H.1

Use Case Description: Create New Graph

Create a new graph

Primary Actors: Salesforce User

Preconditions: The application has been deployed to a Visualforce page.
The user has access to the Visualforce page containing the application.

Basic flow of events:

1. User has logged into Salesforce using their Salesforce credentials.
2. User navigates to the page where the application is deployed.
3. The application queries the REST API manager for any graph previously created.
4. The user clicks on the button to create a new graph.
5. A modal dialog opens to provide options for creating a new graph.
6. User selects a report.
7. Application queries Report API for metadata if it does not already exist in the local state.
8. Application queries Report API for data if it does not already exist in the local state.
9. Application passes selected report data to chart and graph library for rendering.
10. User selects a graph type.
11. The application reloads the graph with the selected graph type.
12. The user types a value for the graph's header, X Axis or Y Axis labels.
13. The application reloads the graph with the label values.
14. User clicks the create button to save the graph.
15. The application persists the selected report, graph and properties of the graph as a new record in Salesforce.
16. The application closes the modal.

Alternative flows:

1a. User is unable to authenticate with Salesforce.
1a1. Salesforce is locked out/invalid username/invalid password.

2a. The application has not been deployed.
2a1. User is unable to navigate and access application.

6a. No reports exist for the user to see.
6a1. The list of graphs will only have a single value, "Please Select a Report".

7a. A network issue has occurred.
7a1. A notification appears stating, "Unable to retrieve report data."

8a. A network issue has occurred.
8a1. A notification appears stating, "Unable to retrieve report data."

8b. No groupings exist for the report.
8b1. A notification appears stating, "This report must have groupings to display a graph."

9a. The data is invalid.
9a1. The chart and graph library will render with a message stating, "No graph data to display."

11a. The data is invalid.
11a1. The chart and graph library will render with a message stating, "No graph data to display."

13a. The data is invalid.
13a1. The chart and graph library will render with a message stating, "No graph data to display."

14a. An issue occurred during save.
14a1. A notification appears stating, "The graph was unable to save."

APPENDIX H.2

Use Case Description: Edit a Graph

Edit a Graph

Primary Actors: Salesforce User

Preconditions: The application has been deployed to a Visualforce page.

A graph has been saved for the user by the application.

Basic flow of events:

1. User has logged into Salesforce using their Salesforce credentials.
2. User navigates to the page where the application is deployed.
3. The application queries the REST API manager for any graph previously created.
4. The user clicks on the edit icon to modify the graph.
5. A modal dialog opens to provide options for editing the graph.
6. User selects a report different than previously selected.
7. Application queries Report API for metadata if it does not already exist in the local state.
8. Application queries Report API for data if it does not already exist in the local state.
9. Application passes selected report data and graph type to chart and graph library for rendering.
10. User selects a graph type different than the one previously selected.
11. The application reloads the graph with the selected graph type.
12. The user updates the value for the graph's header, X Axis or Y Axis labels.
13. User clicks the update button to store the changes.
14. The application persists the selected report, graph and properties of the graph to the existing record in Salesforce.
15. The application closes the modal.

Alternative flows:

1a. User is unable to authenticate with Salesforce.

1a1. Salesforce is locked out/invalid username/invalid password.

2a. The application has not been deployed.

2a1. User is unable to navigate and access application.

6a. No reports exist for the user to see.

6a1. The list of graphs will only have a single value, "Please Select a Report".

6b. The report was deleted.

6b1 The list of graphs will only have a single value, "Please Select a Report".

7a. A network issue has occurred.

7a1. A notification appears stating, "Unable to retrieve report data."

8a. A network issue has occurred.

8a1. A notification appears stating, "Unable to retrieve report data."

8b. No groupings exist for the report.

8b1. A notification appears stating, "This report must have groupings to render."

9a. The data is invalid.

9a1. The chart and graph library will render with a message stating, "No graph data to display."

14a. An issue occurred during the update request.

14a1. A notification appears stating, "The graph experienced an issue during the update."

APPENDIX H.3

Use Case Description: Delete Graph

Delete a Graph

Primary Actors: Salesforce User

Preconditions: The application has been deployed to a Visualforce page.

The user has access to the Visualforce page containing the application.

A graph has been saved for the user by the application.

Basic flow of events:

1. User has logged into Salesforce using their Salesforce credentials.
2. User navigates to the page where the application is deployed.
3. The application queries the REST API manager for any graph previously created.
4. The user clicks on the icon to delete the graph.
5. The application persists the selected report, graph and properties of the graph as a new record in Salesforce.

Alternative flows:

1a. User is unable to authenticate with Salesforce.

1a1. Salesforce is locked out/invalid username/invalid password.

2a. The application has not been deployed.

2a1. User is unable to navigate and access application.

5a. An issue occurred during the delete request.

5a1. A notification appears stating, "The graph was unable to be deleted."

APPENDIX H.4

Use Case Description: Load a Saved Graph

Load a Saved Graph

Primary Actors: Salesforce User

Preconditions: The application has been deployed to a Visualforce page.

The user has access to the Visualforce page containing the application.

A graph has been saved for the user by the application.

Basic flow of events:

1. User has logged into Salesforce using their Salesforce credentials.
2. User navigates to the page where the application is deployed.
3. The application queries the REST API manager for a graph previously created.

Alternative flows:

1a. User is unable to authenticate with Salesforce.

1a1. Salesforce is locked out/invalid username/invalid password.

2a. The application has not been deployed.

2a1. User is unable to navigate and access application.

3a. An issue occurred during the fetch request.

3a1. A notification appears stating, "The graph was unable to load."

APPENDIX H.5

Use Case Description: Refresh Graph

Refresh Graph

Primary Actors: Salesforce User

Preconditions: The application has been deployed to a Visualforce page.
The user has access to the Visualforce page containing the application.

A graph has been saved for the user by the application.

Basic flow of events:

1. User has logged into Salesforce using their Salesforce credentials.
2. User navigates to the page where the application is deployed.
3. User clicks the refresh icon.
4. The application queries the REST API manager with the current graph ID.

Alternative flows:

1a. User is unable to authenticate with Salesforce.

1a1. Salesforce is locked out/invalid username/invalid password.

2a. The application has not been deployed.

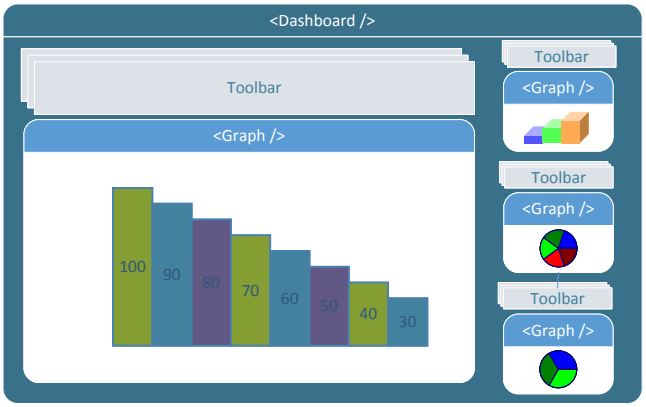
2a1. User is unable to navigate and access application.

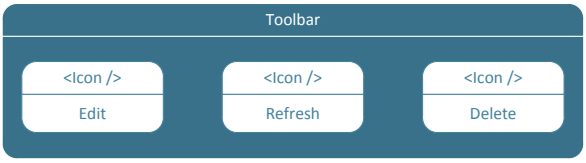
4a. An issue occurred during the fetch request.

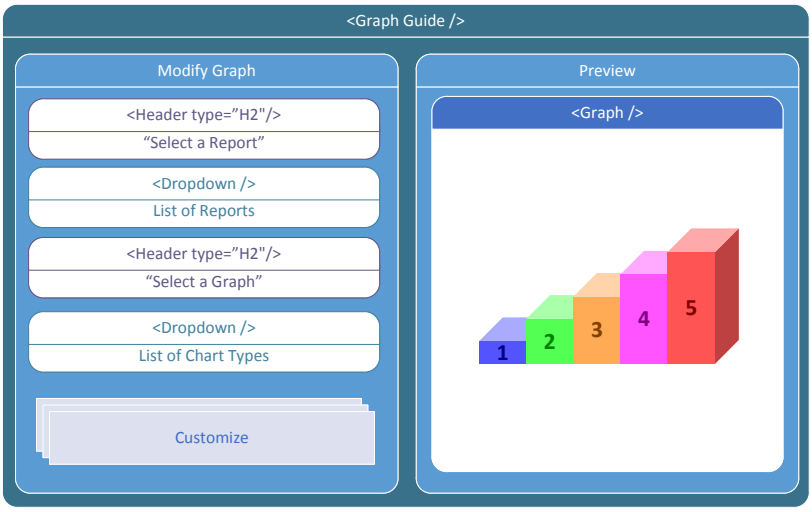
4a1. A notification appears stating, "The graph was unable to refresh."

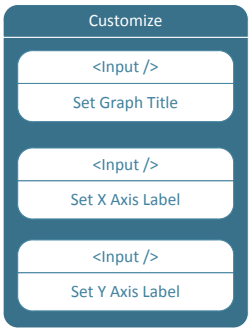
APPENDIX I

Container Diagram



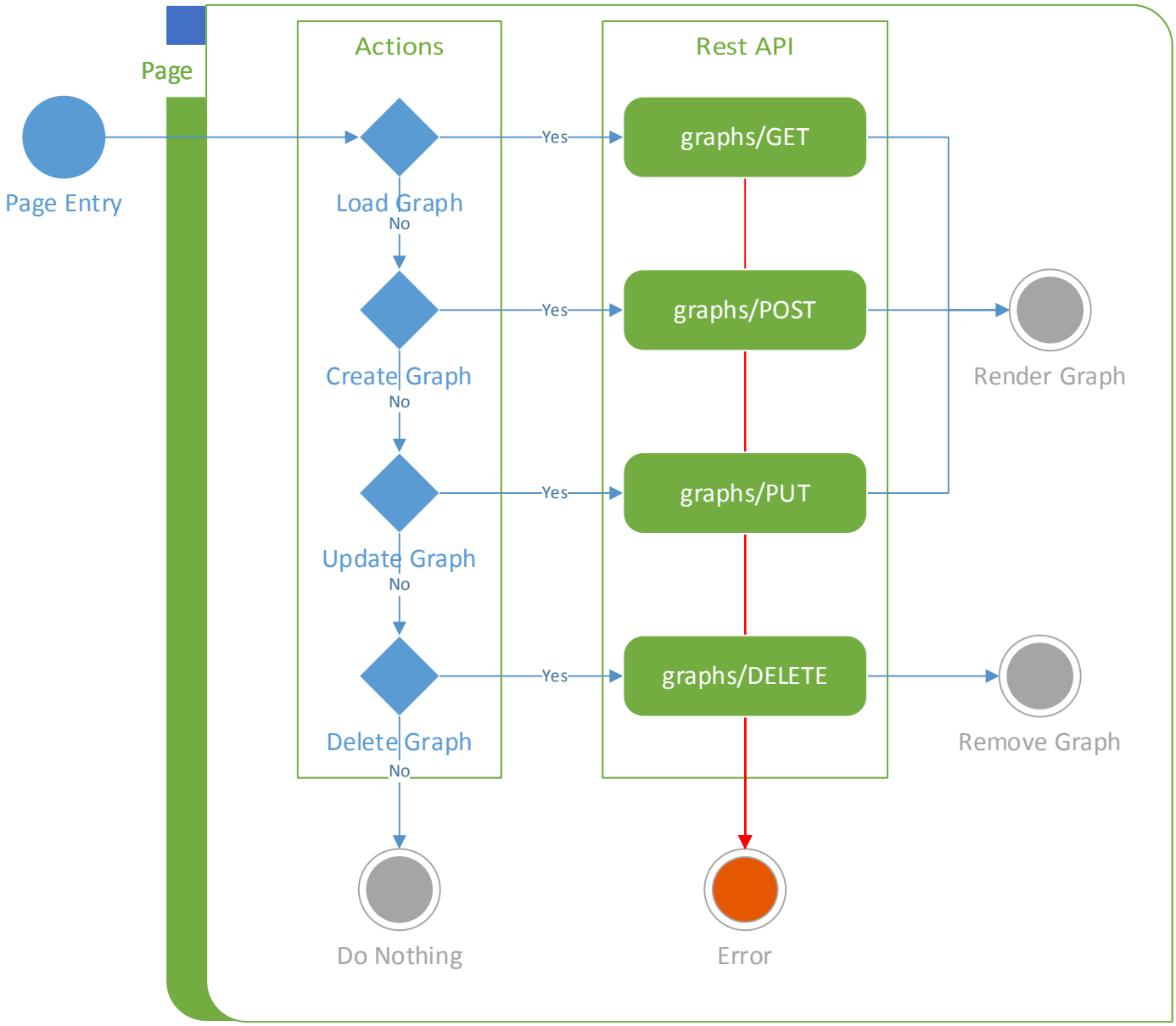






APPENDIX J

State Diagram



APPENDIX K

AJAX/HTTP Library Comparison

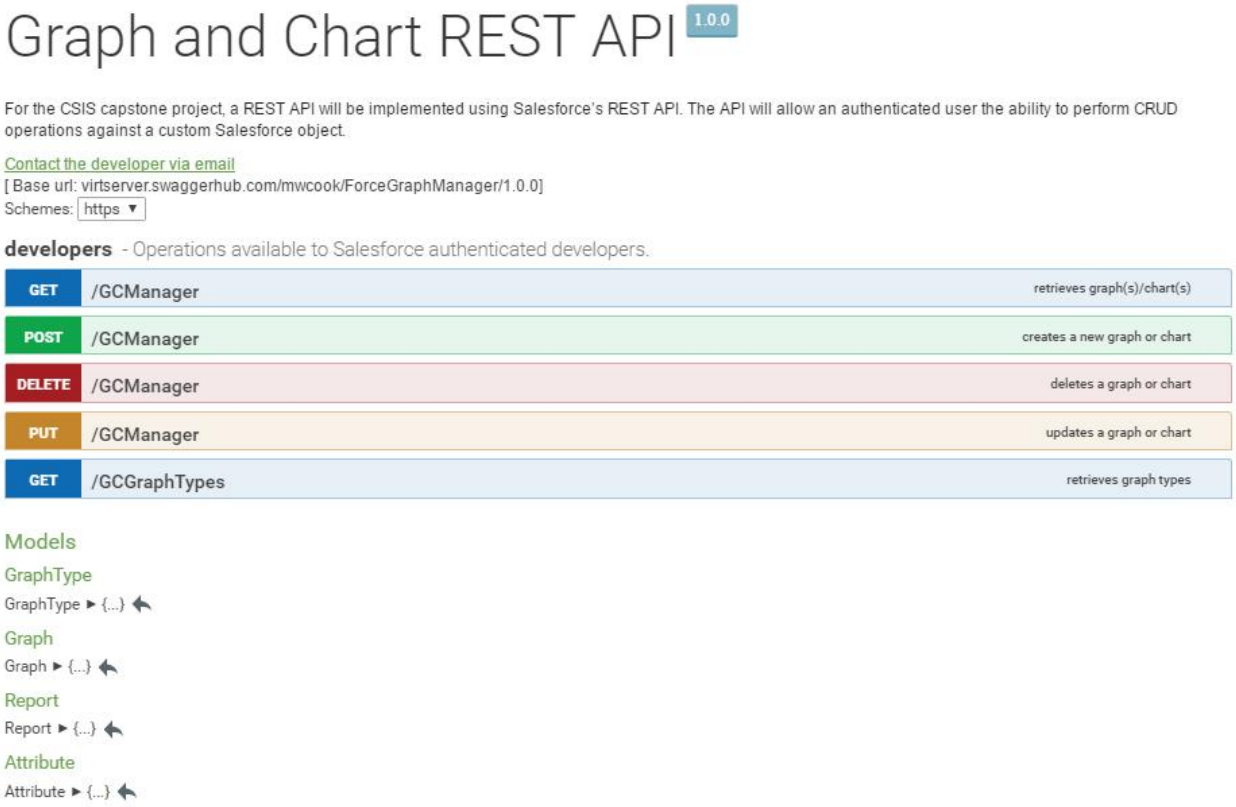
	Support			Features				
	Chrome & Firefox[1]	All Browsers	Node	Concise Syntax	Promises	Native[2]	Single Purpose[3]	Formal Specification
XMLHttpRequest	✓	✓				✓	✓	✓
Node HTTP			✓			✓	✓	✓
fetch()	✓			✓	✓	✓	✓	✓
Fetch polyfill	✓	✓		✓	✓		✓	✓
node-fetch			✓	✓	✓		✓	✓
isomorphic-fetch	✓	✓	✓	✓	✓		✓	✓
superagent	✓	✓	✓	✓			✓	
axios	✓	✓	✓	✓	✓		✓	
request			✓	✓			✓	
jQuery	✓	✓		✓				
reqwest	✓	✓	✓	✓	✓		✓	

[1] **Chrome & Firefox** are listed separately because they support `fetch()` : caniuse.com/fetch
[2] **Native:** Meaning you can just use it - no need to include a library.
[3] **Single Purpose:** Meaning this library or technology is ONLY used for AJAX / HTTP communication, nothing else.

http://andrewhfarmer.com/ajax-libraries/

APPENDIX L

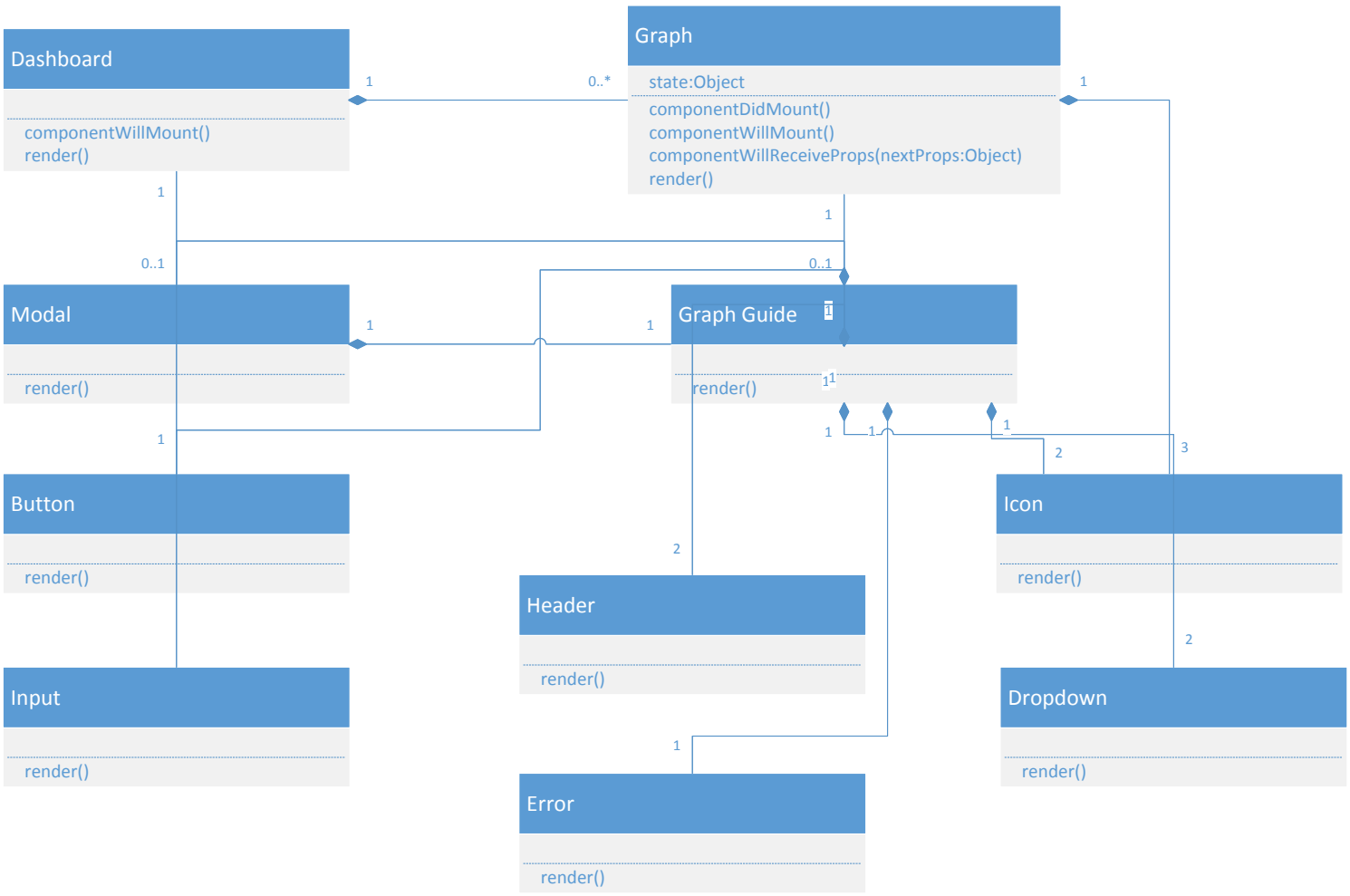
Swagger REST API Documentation



https://app.swaggerhub.com/api/mwcook/ForceGraphManager/

APPENDIX M

Class Diagram



APPENDIX N

Dashboard.jsx

```
import Dashboard from 'Documentor/React_Redux/components/Dashboard.jsx'
```
public class | source

Dashboard

Extends:
react~Component → Dashboard

The Dashboard component is the parent component to the application. It holds a list of all the graphs and the ability to add/edit graphs.

Decorators:
connect(mapStateToProps, mapDispatchToProps)

Method Summary

Public Methods	
public	componentWillMount() The function to execute before the component mounts.
public	render(): ReactComponent Renders the component.

Public Methods

public componentWillMount() source

The function to execute before the component mounts.

public render(): ReactComponent source

Renders the component.

Return:
ReactComponent The data to render inside the component.

APPENDIX O

Graph.jsx

```
import Graph from 'Documentor/React_Redux/components/Graph.jsx'
```
public class | source

Graph

Extends:
react~Component ⟶ Graph

The graph class renders each individual chart or graph saved or previewed by the user.

Decorators:
connect(mapStateToProps, mapDispatchToProps)

Member Summary

Public Members	
public	state: Object Adding an instance of a FusionChart to the graph state.

Method Summary

Public Methods	
public	componentDidMount() The function to execute after the component mounted.
public	componentWillMount() The function to execute before the component mounts.
public	componentWillReceiveProps(nextProps: object) The function will execute when the properties are expected to update.
public	render(): ReactComponent Renders the component.

APPENDIX P

Graph.jsx (cont.)

Public Members

public state: Object source

Adding an instance of a FusionChart to the graph state.

Public Methods

public componentDidMount() source

The function to execute after the component mounted.

public componentWillMount() source

The function to execute before the component mounts.

public componentWillReceiveProps(nextProps: object) source

The function will execute when the properties are expected to update.

Params:

Name	Type	Attribute	Description
nextProps	object		The new set of properties being passed to the component.

public render(): ReactComponent source

Renders the component.

Return:

ReactComponent The data to render inside the component.

APPENDIX Q

GraphGuide.jsx

```
import GraphGuide from 'Documentor/React_Redux/components/GraphGuide.jsx'
```
public class | source

GraphGuide

Extends:
react~Component → GraphGuide

```
The GraphGuide class builds the guide to assist a user with creating or modifying an existing graph.
```

Decorators:
connect(mapStateToProps, mapDispatchToProps)

Method Summary

Public Methods	
public	render(): ReactComponent Renders the component.

Public Methods

public render(): ReactComponent source

Renders the component.

Return:
ReactComponent The data to render inside the component.

APPENDIX R

Modal.jsx

```
import Modal from 'Documentor/React_Redux/containers/Modal.jsx'
```
public class | source

Modal

Extends:
react~Component ⟶ Modal

The Modal is a component used to display an overlay modal.

To use, render the Modal tag with a body. The body will be passed via "this.props.children".

Method Summary

Public Methods	
public	render(): ReactComponent Renders the component.

Public Methods

public render(): ReactComponent source

Renders the component.

Return:
ReactComponent The data to render inside the component.

APPENDIX S

Header.jsx

```
import Header from 'Documentor/React_Redux/containers/Header.jsx'
```
public class | source

Header

Extends:
react~Component ⟶ Header

The Header component is a generic component used to build HTML headers (i.e. H1, H2, H3, etc...).

Method Summary

Public Methods	
public	render(): ReactComponent Renders the component.

Public Methods

public render(): ReactComponent source

Renders the component.

Return:
ReactComponent The data to render inside the component.

APPENDIX T

Button.jsx

```
import Button from 'Documentor/React_Redux/containers/Button.jsx'
```
public class | source

Button

Extends:
react~Component → Button

A generic button component.

Method Summary

Public Methods	
public	render(): ReactComponent Renders the component.

Public Methods

public render(): ReactComponent source

Renders the component.

Return:
ReactComponent The data to render inside the component.

APPENDIX U

Icon.jsx

```
import Icon from 'Documentor/React_Redux/containers/Icon.jsx'
```
public class | source

Icon

Extends:
react~Component ⟶ Icon

Icon component used to render icon fonts.

Method Summary

Public Methods	
public	render(): ReactComponent Renders the component.

Public Methods

public render(): ReactComponent source

Renders the component.

Return:
ReactComponent The data to render inside the component.

APPENDIX V

Dropdown.jsx

```
import Dropdown from 'Documentor/React_Redux/containers/Dropdown.jsx'
```
public class | source

Dropdown

Extends:
react~Component → Dropdown

The Dropdown component is a generic dropdown.

Method Summary

Public Methods	
public	render(): ReactComponent Renders the component.

Public Methods

public render(): ReactComponent source

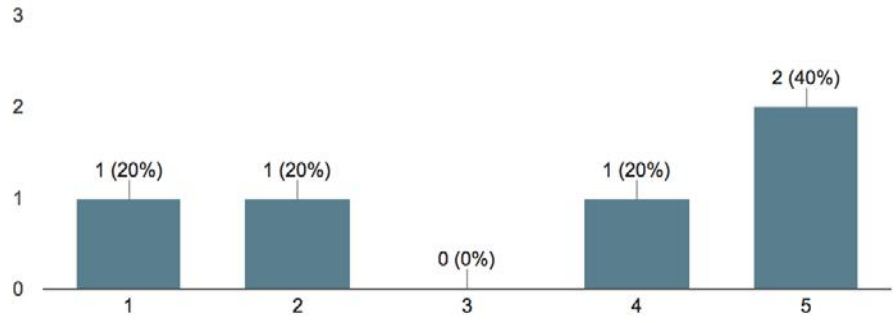
Renders the component.

Return:
ReactComponent The data to render inside the component.

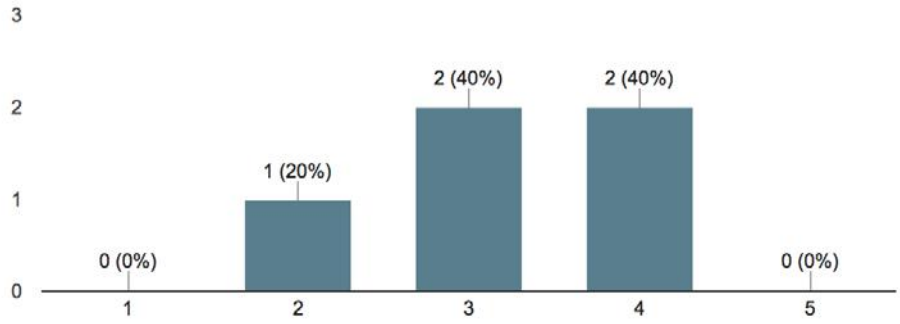
APPENDIX W

Likert Question 1 & 2

The dashboard was easy to use. (5 responses)



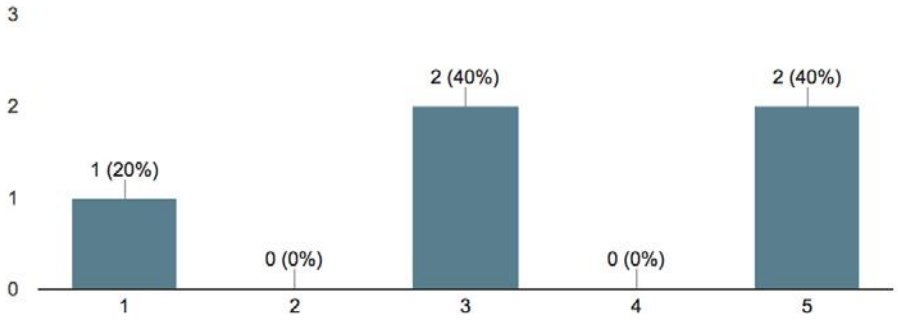
The dashboard has all the features I would expect it to have. (5 responses)



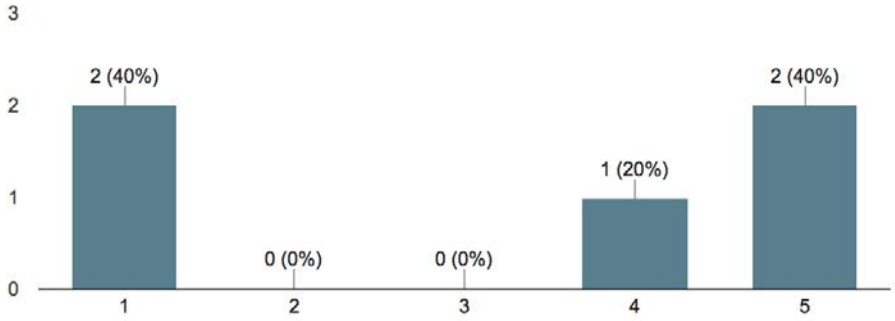
APPENDIX X

Likert Question 3 & 4

I find the dashboard to be generally useful. (5 responses)



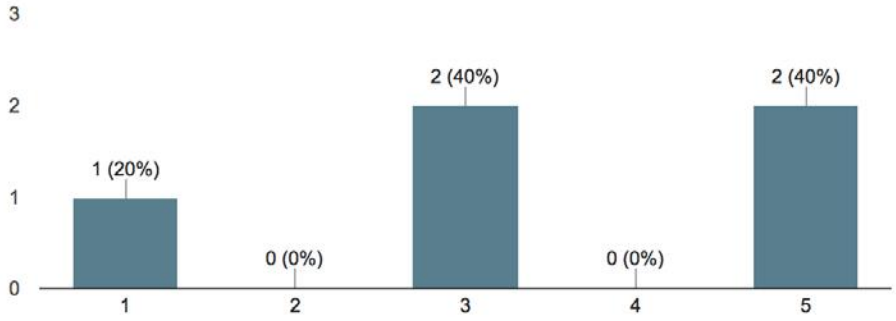
The dashboard was easy to learn. (5 responses)



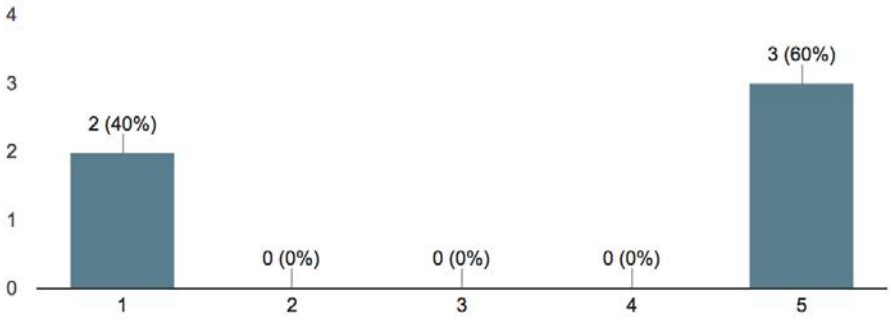
APPENDIX Y

Likert Question 5 & 6

If implemented, I would use this dashboard to help me in my job. (5 responses)



I use graphs and charts often to see information relevant to my job.
(5 responses)



APPENDIX Z

Open Ended Question 1 & 2

What did you like least about the system? (5 responses)

In appearance it is still very similar to SF Dashboards
I didnt like how the 3D pie chart was interactive (you can rotate the chart to view segments) but the rest of the charts were static.
Was not easy for me to figure what I was supposed to do
It didn't have some features you can get in SF dashboards like plotting additional values, sub-groups, etc.
The preview box for the chart looked out of place. It was larger than the original container.

Are there any additional chart or graph types (i.e. pie, bubble, funnel, etc...) you would like to see added to the project?
(5 responses)

- HeatMap Chart - Bubble Charts - CountryMap with bubble overlay - Tables - Summary Stats - Sparkline within tables
No.
NA
Funnel, bar, and donut
3D Funnel Chart would be a fun addition!

APPENDIX AA

Open Ended Question 3 & 4

Do you have any other suggestions/comments you would like to submit?

(4 responses)

I would be interested to see more about the back-end as I am sure, there are tons of cool stuff happening in the back-end.

A small bug I may have noticed were the question mark help icons on the edit did not seem to work.

This has potential, I think more needs to be done to differentiate it from the current way SF handles dashboards and reporting. Graphically it most definitely looks better than SF graphs, but I think focusing on how to give more control to the user in the area of manipulating the data that powers these graphs could be a way to significantly set this project apart.

When it comes to performance it seem to be working really smoothly, I'd be curious to see what large amount of data do to rendering time and overall dashboard loading.

For a first basic run, I'd say that this looks pretty good. I am most definitely curious to see where you take that project next.

I hope the feedback helps, keep up the good work!

The ability to upload an xls or csv that would generate some sort of summary report would be nice -- if the data mapped into the fields correctly. If you had the ability to drag/drop each chart into a different location on the dashboard that would be awesome.

Maybe an overview page with directions

It's a very good prototype. If you added the standard features from SF dashboards, this would be a great replacement.

APPENDIX BB

Open Ended Question 5 & 6

What did you like most about the system? (5 responses)

The ease of use, straight forward and overall design is pleasing to the eye
The system is intuitive & easy to learn; It asks for an input, and I end up picking the report type, graph type, and title customization.
Point and Click
Easier and quicker to use than a standard Salesforce dashboard
It was simple and intuitive!

What type of feature, or features, would you add to make this dashboard more useful?

(5 responses)

- The ability to manipulate the source of the data on the spot, without having to use Report Types or SF Reports but I think maintaining that ability is also important. In short being able to use report type and report and in addition have a way of bypassing all that as well as to more easily manipulate the data. - The ability to include non-graph items such as summary stats or small text bit - Filtering - Moving graphs around, changing their individual sizes - Perhaps making the edit and trash a little more inconspicuous
It would be useful to be able to drill down into the reports & charts. If you have 200 won opportunities, it would be cool to see the opportunities across each state, or where the lost opportunities were and when they happened.
Ability to enter the actual data
The features I mentioned above would be very useful.
Additional customization such as color selection would be cool.