Software Requirements Specification for

TeamPerformanceViz.

Taller Vertical 2013 - Ingeniería en Sistemas Computacionales

Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Guadalajara

Oracle Mexico Development Center - User eXperience Design Development Team

Table of Contents

Revision History

Revision	Author	Changes
2013.10.06	Antonio Aguilar	▪ Added due date for delivery. ▪ Updated Appendix F. Zip File section.
2013.10.03	Antonio Aguilar	▪ Added many page breaks along the document. ▪ Fixed styles for hyperlinks along the document. ▪ Updated Appendix D. Web API section.
2013.10.02.a	Antonio Aguilar	▪ Added Tooltip Layout/Contents figure. ▪ Added a solid line border to all figures.
2013.10.02	Antonio Aguilar	▪ Added TeamPerformanceViz User Interface Elements figure. ▪ Added ISC logo in the front page.
2013.10.01	Antonio Aguilar	▪ Added references to appendices. ▪ Added header and footer. ▪ Added TeamPerformanceViz User Interface figure.
2013.09.30	Antonio Aguilar	▪ Added Interaction between Client and Server components figure.
2013.09.27	Antonio Aguilar	▪ Added Visualization Challenge logo in the front page.
2013.09.25	Antonio Aguilar	▪ First draft.

Introduction

Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a detailed description of the functionalities of a Team Performance visualization (hereinafter known as TeamPerformanceViz). This document will cover each of the project expected features, as well as its User Interface (UI) and behavior (User eXperience). Hardware, software, and various other technical dependencies will also be covered in this document.

For information about the scope of this project see Product Scope section.

Document Conventions

This document follows certain typographic conventions outlined below:

`Monospace` is used for directory names, filenames, etc. All terms shown in `monospace` are typed literally. It is also used to show the contents of files or the output from commands.

Italic is used to indicate library names, programming language names, technical terms, etc.

Dotted underline is used for intra-document hyperlinks.

Underline is used for external hyperlinks.

ℹ This icon signifies a tip, suggestion, or general note.

⚠ This icon indicates a warning or caution.

ℹ This icon references a User Interface (UI) element.

Readers may be unfamiliar with terminology featured in this document. See

Appendix A. Glossary for a list of terms and their definitions.

Intended Audience and Reading Suggestions

This document is intended for all individuals participating in the TeamPerformanceViz project/hackathon.

Readers interested in a brief overview of the project should focus on the rest of the Introduction section, as well as the

Overall Description section, which provide a brief overview of each aspect of the project as a whole.

Readers who wish to explore the features of the TeamPerformanceViz project in more detail should read on to

System Features section, which expands upon the information laid out in the main overview. External Interface Requirements section offers further technical details, including information on the User Interface (UI) as well as the hardware and software platforms on which the application will run.

Readers interested in the non-technical aspects of the project should read

Other Nonfunctional Requirements section, which covers performance, and various other attributes that will be important to users.

Readers who have not found the information they are looking for should check

Other Requirements section, which includes any additional information which does not fit logically into the other sections.

Product Scope

TeamPerformanceViz is composed of two main components: a client-side application which will run on multiple Tablet and/or Desktop browsers (Google Chrome, Mozilla Firefox and Apple Safari), and a server-side application which will serve the input data (*JSON*) required to populate the client-side.

TeamPerformanceViz is designed to facilitate the visualization of the performance of a team in a quick and innovative way. Potential scenarios include visualizing the performance of a sales, development, or testing team.

Overall Description

Product Perspective

TeamPerformanceViz is an innovative visualization intended to be viewed on Tablet and/or Desktop browsers (Google Chrome, Mozilla Firefox and Apple Safari).

While the visualization is the main focus of the project, there is also a server-side component which will be used to serve the data required to populate the visualization. The scope of the project encompasses both server- and client-side functionalities, so both aspects are covered in detail within this document. Figure 1 illustrates the interaction between the server and client components of the TeamPerformanceViz project.

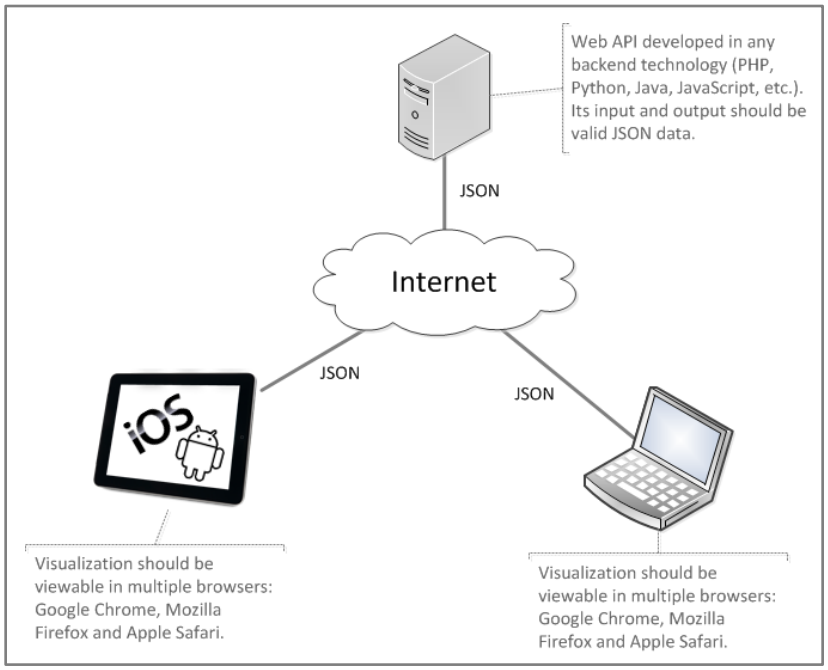

Figure 1. Interaction between Client and Server components

Product Functions

The following list offers a brief outline of the features and functionalities of TeamPerformanceViz. The features are split into two major categories: core features and additional features. Core features are essential to the visualization operation, whereas additional features simply add new functionalities. The latter features will only be implemented as time permits.

Core Features

- HTML Page Scaffolding
- Mountains and Current Progress Displaying
- Scaling
-

- Tooltips Displaying
- Name Label Displaying
- Balloons and Flags Displaying

Additional Features

- Web API Implementation
- Slider Implementation
- Order By Functionality Implementation
- Performance Improvements
- Data Error Validation
- Animations and Transitions Implementation

For more detailed information, see

System Features section.

User Classes and Characteristics

TeamPerformanceViz is meant to offer a simple, quick and innovative way to visualize the performance of a team. Consequently, it has little or no learning curve. Its user interface is as intuitive as possible. Thus any special expertise is necessary.

Although TeamPerformanceViz is mostly intended for sales team managers, it should not only be restricted to this kind of users. TeamPerformanceViz could be used in others not now imaginable contexts.

Operating Environment

The main component of the TeamPerformanceViz project is the visualization, which should be viewed on multiple Tablet and/or Desktop browsers (Google Chrome, Mozilla Firefox and Apple Safari). The visualization may be resource- or graphics-intensive, so ensuring a good performance will be a major concern.

The visualization will, eventually however, be populated with data served from a web *API*. The web *API* can operate on any environment (*Windows*, *Mac OS* or *Linux*) in which an *HTTP* Web Server such as *Apache* (*WAMP*, *MAMP*, *LAMP*, etc.[1]) or *node.js* can be installed. The web *API* can be developed in any backend technology (*PHP*, *Python*, *Java*, *JavaScript*, etc.) as long as its output is valid *JSON* data.

Design and Implementation Constraints

It is fundamental supporting multiple browsers (Tablet and/or Desktop browsers previously mentioned). Since TeamPerformanceViz is designed to be viewed on Tablet and/or Desktop browsers, capturing touch and/or mouse gestures will be a major design consideration. Generating a user interface (UI) which is both effective and usable will pose a difficult challenge. Other constraints such as limited memory and processing resources are also worth considering. TeamPerformanceViz is meant to be quick and responsive (talking in performance terms), even when dealing with large amounts of data so each feature must be designed and implemented with efficiency in mind.

See Software Interfaces section for detailed information about frontend and backend implementation.

User Documentation

The primary goal of TeamPerformanceViz is to facilitate the visualization of the performance of a team through an innovative visualization. This visualization has been designed to be self-explanatory and as simple to use as possible.

Anyways, in the future, end-users may be trained in order to perfectly understand the displayed data. TeamPerformanceViz may also include a *Help* button which could provide a brief explanation of the displayed data. Both things would only be performed and/or implemented as time permits.

[1] Although, some *HTTP* Web Server stack alternatives have been suggested using a database is not a must.

Assumptions and Dependencies

Assumptions

- Teams are composed by 10 members with mixed experiences (from first to fifth semester).
- Team members are familiar with web technologies such as *HTML5*, *CSS3* and *JavaScript*.
- Team members have basic to medium knowledge of backend technologies such as *PHP*, *Java*, *Python*, *JavaScript*, etc. They are able to generate web *API*s (as well as Web Services).
- Some team members will attend a *D3.js* Workshop which will be led a couple of days before the hackathon week (September 4th, 2013 14:30 hrs.). The main purpose of this *D3.js* Workshop is to teach them the basics of the *D3.js* library. If time permits, this Workshop may also include a small *JavaScript* and *SVG* tutorial. Attendants will be in charge of spreading the acquired knowledge to the rest of his/her team mates. After acquiring the basics of the *D3.js* library, teams may need to dive into the *D3.js* documentation and gallery in order to get even more knowledge/experience.
- Teams have at least one tablet device and a PC in which their implementation can be tested.
- Teams will spend a whole week working in this project. In other words, from Monday September 7th, 2013 to Thursday September 10th, 2013 from 9.00 to 18:00 hrs. and Friday September 11th, 2013 from 9:00 to 12:00 hrs.; approximately 35 hours in total.

Time Dependencies

Features of TeamPerformanceViz are divided into two groups: Core Features and Additional Features. Core features are crucial to the basic functionality of TeamPerformanceViz. These features must all be implemented in order for the visualization to be useful. Additional features, however, are not critical to the function of the visualization. They are usability improvements and convenience enhancements that can be added later if time permits. Thus, the implementation of these features is entirely dependent upon the time spent designing and implementing the core features. The final decision on whether or not to implement these features should be made during the middle days of the development phase. Hackathon judges/organizers should be notified about the impossibility of implementing these features. Judges/Organizers will indicate to a team lead how to proceed.

External Dependencies

Beyond the use of the *D3.js* library (which is suggested). There are no more external dependencies.

However, it is important to mention, that for a successful implementation of some of the features of TeamPerformanceViz it is crucial to acquire as much knowledge as possible about the *D3.js* library in a limited time frame; even more knowledge than the acquired at the *D3.js* Workshop.

External Interface Requirements

User Interfaces

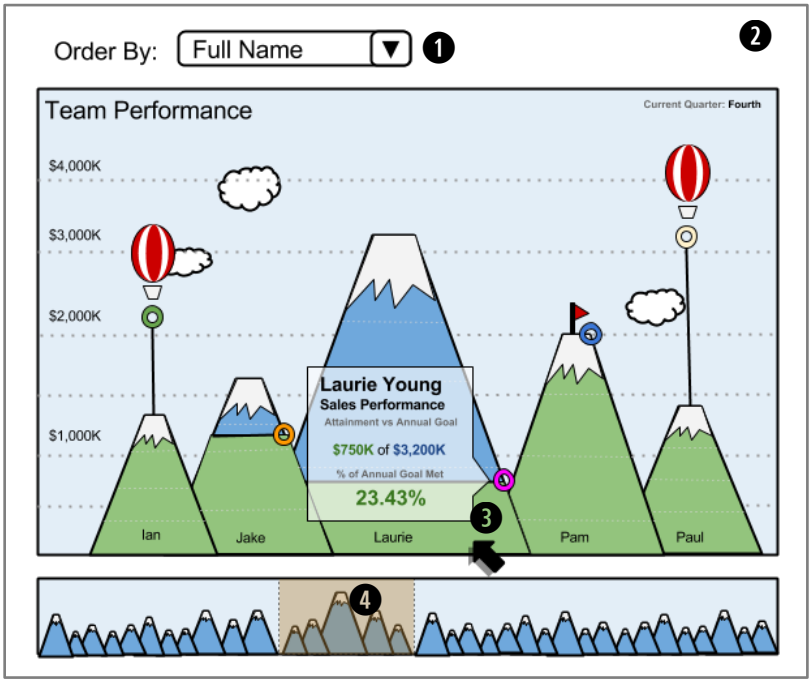

Figure 2. TeamPerformanceViz User Interface

- Figure 2 shows the User Interface (UI) of TeamPerformanceViz.
- TeamPerformanceViz is populated each time:
  - Its hosting web page is loaded or refreshed (talking in more precise technical terms, the visualization is populated when the *DOM* of the hosting *HTML* page has been loaded and parsed so that it is ready to be manipulated: `DOMContentLoaded`).
  - A new option is selected in the *Order By* ❶ combo box.
- Users can start interacting with the visualization once it is loaded and populated:
  - In a quick glance, a visualization ❷ like the one displayed in Figure 2, allows a sales manager to see the performance of his team: it shows which sales representatives made it best/worst; which ones met/exceeded their goals; it also shows quarter goals for each sales representative, etc.
  - Users can click/tap a mountain ❸ to see details of a sales representative such as *Full Name*, *Attainment* and *Annual Goal* amounts, and *Percentage of Annual Goal Met*.
  - Users can drag the *Slider Knob* ❹ (located below the *Chart Canvas*) in order to scroll the *Chart Canvas*. This functionality is useful when the available room of the *Chart Canvas* is not enough to show large teams.
  - See

- System Features section for detailed description about TeamPerformanceViz features.
- See

- Appendix F. Zip File section for assets related with the User Interface (UI).

Hardware Interfaces

TeamPerformanceViz is intended to be viewed on Tablet and/or Desktop browsers.

By now, there is no restriction about which browser or operating system version should be supported. Thus, it is ok to support the latest versions of the Google Chrome, Mozilla Firefox and Apple Safari browsers (for Tablets and/or Desktop).

Additionally, there is no restriction about which tablet should be supported. Thus, it is ok to support any *iPad* tablet as well as tablets with the latest version of *Android*.

Software Interfaces

TeamPerformanceViz frontend must be implemented with *HTML5*, *CSS3* and plain *JavaScript*. The use of the *D3.js* visualization library is suggested. However, any other *MIT/BSD/Apache 2* licensed *JavaScript* charting library can be used as long as it does not depend on any other *JavaScript* library/framework. The use of micro *JavaScript* libraries/frameworks such as *underscore*, *lodash*, *xui*, *zepto*, *handlebars*, etc. is not permitted.

TeamPerformanceViz backend can be developed in any technology/programming language (*Java*, *PHP*, *Python*, *JavaScript*, etc.) as long as the output is valid *JSON* data.

JSON data required to populate the visualization should be served using one or both of the following approaches:

a) *JSON* file: Contents of the `team-performance.json` file should be served from backend referencing this URL:

```
http://<server>/retrieveTeamPerformanceData
```

b) *JSON* data served by a web *API*: See

c) Appendix D. Web API section for details.

Communications Interfaces

TeamPerformanceViz will have a network server that is web-based and can be developed using any backend technology (*PHP*, *Python*, *Java*, *JavaScript*, etc.). The server exists to serve the data needed to populate the visualization. Data exchanged between frontend and backend should be in *JSON* format. The *HTTP* request required to retrieve the visualization data should be performed once the *DOM* of the *HTML* page in which the visualization is embedded is ready to be manipulated.

System Features

TeamPerformanceViz features are divided into two main categories: Core Features and Additional Features. Core features are essential to the functionality of the visualization. These features must be implemented in order to have a fully-functioning application. Additional features, however, are not completely required for the app to function. They include any features which, if time permits, will be added to the application in order to provide extra functionality.

⚠ TeamPerformanceViz MUST be cross-browser. In other words, all of its features MUST work fine in multiple Desktop and Tablet browsers (Google Chrome, Mozilla Firefox, Apple Safari).

Before listing TeamPerformanceViz features, let's use Figure 3 to describe its UI elements and consequently its functionality:

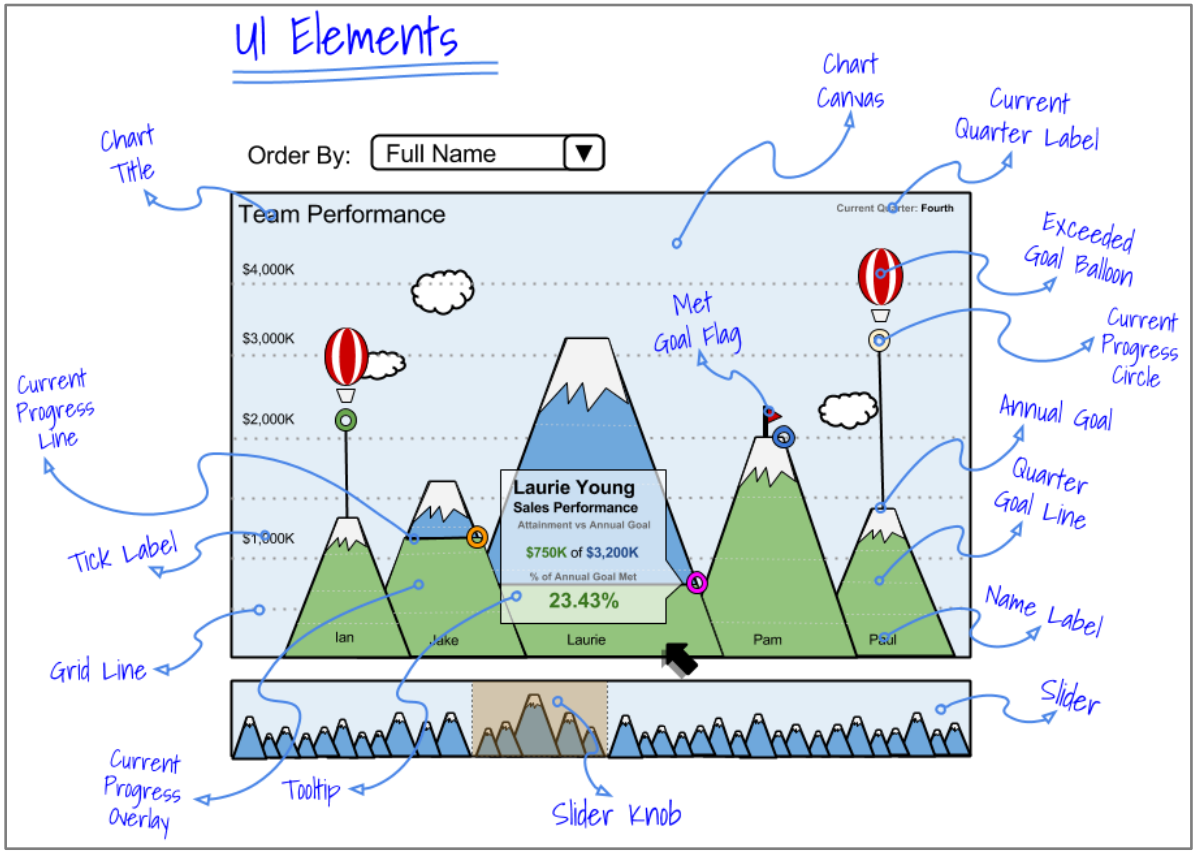

Figure 3. TeamPerformanceViz User Interface Elements

A mountain within the *Chart Canvas* and *Slider* areas is generated for each sales representative. By default (unless a different order is specified), mountains are ordered alphabetically by *Full Name* (using sales representative full name). It is also possible to order mountains by *Percentage of Annual Goal Met* and *Goal Completion Status*. At the end of the day, no matter the specified order, the first mountain overlaps the second one, the second one overlaps the third one and so on. Mountains displayed in Figure 3 are ordered by *Full Name*.

Each mountain (within the *Chart Canvas*) displays the *Goals* and *Current Progress* of a sales representative. *Goals* are represented by horizontal dotted lines which split a mountain into non-proportional slices. Each of these slices represent a *Quarter*. The top of a mountain represents the *Annual Goal* of a sales representative. Additionally, *Current Progress* of a sales representative is represented by multiple UI elements: 1) a colored circle (*Current Progress Circle*) at the right slope of a mountain or at the sky just above a mountain, 2) a green overlay over a mountain (*Current Progress Overlay*), and/or 3) an horizontal solid line (*Current Progress Line*) which starts at the center of the *Current Progress Circle* and ends at the opposite slope of a mountain.

Each *Current Progress Circle* must have a different color asigned dynamically.

If a goal is met by a sales representative then apart of displaying the *Current Progress Circle* at the right slope of a mountain, a flag (*Met Goal Flag*) is displayed at center of the top of a mountain.

If a goal is exceeded by a sales representative then a colored circle (*Current Progress Circle*) is positioned at the sky just above of a mountain (in the exact point achieved by the sales representative). Additionally, two more UI elements are displayed in order to reinforce the representation of an exceeded goal: 1) a balloon (*Goal Exceeded Balloon*) just above the *Current Progress Circle*, and 2) a vertical solid line which starts at the center of the top of a mountain and ends at the *Current Progress Circle* located at the sky.

Each mountain has a label (*Name Label*) at the bottom. This label is in fact the first name of the sales representative. When the first name of a representative is really long then only the first letter of his/her first name should be used.

When a mountain is clicked/tapped a *Tooltip* appears. This *Tooltip* displays the *Full Name*, the *Attainment* and *Annual Goal* ammounts, and the *Percentage of Annual Goal Met* of the sales representative represented by the clicked/tapped mountain. Figure 4 shows how the *Tooltip* should look.

The height of a mountain depends on the *Annual Goal* of a sales representative. On the other side, its width is proportional to its height. In other words, mountains always keep their aspect ratio.

Tick Labels (located at Y axis) and *Grid Lines* depends on factors such as mountains height, location of the *Goal Exceeded Balloons*, etc. They should be updated each time the visualization is populated.

As it name says, the *Current Quarter Label* (located at the top right corner of the *Chart Canvas*) displays the current quarter which is being visualized.

The *Chart Title* (located at the top left corner of the Chart Canvas) can be set programatically.

Since the *Chart Canvas* has limited dimensions (height and width), sometimes it does not have enough room to display all the mountains of a large team. This is where the *Slider* becomes important. The *Slider* contains all the mountains which represent the members of a team. Thus the *Slider* allows end-users to drag left or right the *Slider Knob* in order to visualize all the mountains of a team. The *Slider* works similar to a *scrollbar*. Mountains just behind the *Slider Knob* are displayed within the *Chart Canvas*. By the moment there is no restriction about the maximum number of mountains that the *Slider* can contain. However, this may change in the future if after putting the visualization under usability tests, it is proved that this UI issue requires to be solved.

When a new option is selected in the *Order By* combo box, the whole visualization is repopulated (without refreshing its hosting web page); mountains are ordered in the specified order value.

ℹ As any other spec, this one is not free of errors and/or omissions. If you find an error or something is omitted please let the authors know about it.

Core Features

The following features are ordered by importance:

HTML Page Scaffolding

Generate a valid *HTML5* page to host the visualization: put `<link>` and `<script>` tags in the right place, add a `<div>` which will contain the visualization, add a `<div>` for implementing *Tooltip* functionality, separate *HTML*, *JavaScript* and *CSS* files, generate an appropriate folder structure for the project, etc.

Mountains and Current Progress Displaying

A mountain within the *Chart Canvas* is generated for each sales representative. By default mountains are ordered alphabetically by *Full Name* (using sales representative full name).

Each mountain (within the *Chart Canvas*) displays the *Goals* and *Current Progress* of a sales representative. *Goals* are represented by horizontal dotted lines which split a mountain into non-proportional slices. Each of these slices represent a *Quarter*. The top of a mountain represents the *Annual Goal* of a sales representative. Additionally, *Current Progress* of a sales representative is represented by multiple UI elements: 1) a colored circle (*Current Progress Circle*) at the right slope of a mountain or at the sky just above a mountain, 2) a green overlay over a mountain (*Current Progress Overlay*), and/or 3) an horizontal solid line (*Current Progress Line*) which starts at the center of the *Current Progress Circle* and ends at the opposite slope of a mountain.

Each *Current Progress Circle* must have a different color asigned dynamically.

The height of a mountain depends on the *Annual Goal* of a sales representative. On the other side, its width is proportional to its height. In other words, it is important to keep mountains aspect ratio.

Scaling

Tick Labels (located at the Y axis) and *Grid Lines* should be updated each time the visualization is populated.

Tooltips Displaying

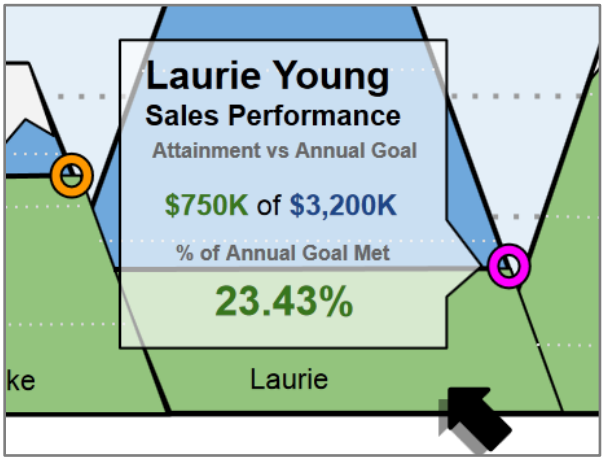

Figure 4. Tooltip Layout/Contents

When a mountain (within the *Chart Canvas*) is clicked/tapped a *Tooltip* appears. This *Tooltip* contains details about the sales representative such as the *Full Name*, *Attainment* and *Annual Goal* amounts and the *Percentage of Annual Goal Met* (See Figure 4).

If a *Tooltip* is visible and a different mountain is clicked/tapped then the original *Tooltip* is hidden and a new one is shown.

Tooltips should increase/decrease its dimensions based on its contents.

It is also fundamental to ensure they are displayed within the visualization container bounds.

Name Label Displaying

Each mountain has a label (*Name Label*) at the bottom. This label is in fact the first name of the sales representative. When the first name of a representative is really long then only the first letter of his/her first name should be used.

Balloons and Flags Displaying

If a goal is met by a sales representative then apart of displaying the *Current Progress Circle* at the right slope of a mountain, a flag (*Met Goal Flag*) is displayed at center of the top of a mountain.

If a goal is exceeded by a sales representative then a colored circle (*Current Progress Circle*) is positioned at the sky just above of a mountain (in the exact point achieved by the sales representative). Additionally, two more UI elements are displayed in order to reinforce the representation of an exceeded goal: 1) a balloon (*Goal Exceeded Balloon*) just above the *Current Progress Circle*, and 2) a vertical solid line which starts at the center of the top of a mountain and ends at the *Current Progress Circle* located at the sky.

Additional Features

The following features are ordered by importance:

Web API Implementation

TeamPerformanceViz backend can be developed in any technology/programming language (*Java*, *PHP*, *Python*, *JavaScript*, etc.) as long as the output is valid *JSON* data.

See

Appendix D. Web API section for more details about how the web API should be implemented.

Slider Implementation

A mountain within the *Slider* is generated for each sales representative. By default mountains are ordered alphabetically by *Full Name* (using sales representative full name).

The *Slider* contains all the mountains which represent the members of a team. Thus the *Slider* allows end-users to drag left or right the *Slider Knob* in order to visualize all the mountains of a team. The *Slider* works similar to a *scrollbar*. Mountains just behind the *Slider Knob* are displayed within the *Chart Canvas*.

Mountains contained within the *Slider* should be properly resized each time the visualization is populated.

Slider Knob should support both mouse and touch gestures in order for the app to work fine in Tablet and/or desktop browsers.

Order By Functionality Implementation

When a new option is selected in the *Order By* combo box, the whole visualization is repopulated (without refreshing its hosting web page); mountains are ordered in the specified order value in both areas: the *Chart Canvas* and the *Slider*.

Performance Improvements

See Performance Requirements section for information about possible performance improvements which can be applied to the app in general.

Data Error Validation

There should be zero tolerance for data errors in the algorithm that generates the visualization. If an error occurs it should be notified to the end-user.

Animations and Transitions Implementation

Admit it; any visualization becomes more compelling when it includes animations/transitions. This one will not be the exception. For now we will try it to keep it as simple as possible. Here is the only animation/transition you will need to implement:

- Mountains Animations/Transitions: Mountains in the *Chart Canvas* should have an animation/transition similar to the titled "*Bar Chart Animated*" on this page.

Other Nonfunctional Requirements

Performance Requirements

TeamPerformanceViz is intended to be viewed on the web then it is important to ensure the best possible performance. Some approaches to achieve this include: compressing *HTTP* requests/responses, minifying assets such as *SVG* files, optimizing images, following code Best Practices, improving *CSS* selectors, etc.

Software Quality Attributes

Code of TeamPerformanceViz will be as maintainable as possible. This means: it will be easy to edit by some other person in the future, it will have a reasonable amount of comments, it will follow Best Practices and as mentioned before, it will be generated having performance in mind.

The user interface of TeamPerformanceViz will be designed with usability as the first priority. The visualization will be presented in a manner that is both visually appealing and easy for the user to manipulate.

To ensure reliability and correctness, there will be zero tolerance for errors in the algorithm that generates the visualization. To maintain adaptability, situations such as when a user loses internet connection or for whatever reason cannot establish a connection with the server will be taken into account.

Overall, the app balances both the ease of use and the ease of learning. The layout and UI of the app will be simple enough that users will take no time to learn its features and interact with it with little difficulty.

Other Requirements

The deliverable for the TeamPerformanceViz hackathon must include its implementation and a PowerPoint presentation. For more information about the deliverable see the Appendix E. Deliverable section.

Appendix A. Glossary

DOM

The Document Object Model (*DOM*) is an *API* for manipulating *HTML* and *XML* documents. It provides a structural representation of the document, enabling you to modify its content and visual presentation by using a scripting language such as *JavaScript*.

JSON

JavaScript Object Notation (*JSON*) is a text-based open standard designed for human-readable data interchange. Derived from the JavaScript scripting language, *JSON* is a language for representing simple data structures and associative arrays, called objects. *JSON* is language independent with parsers available for many languages. The *JSON* format was originally specified by Douglas Crockford. The *JSON* filename extension is `.json`. The *JSON* format is often used for serializing and transmitting structured data over a network connection. It is used primarily to transmit data between a server and web application, serving as an alternative to *XML*.

Visualization

Visualization is any technique for creating images diagrams, or animations to communicate a message.

Web API

A server-side web Application Programming Interface (web *API*) is a programmatic interface to a defined request-response message system, typically expressed in *JSON* or *XML*, which is exposed via the web -most commonly by means of an *HTTP*-based web server.

While a web *API* in this context is sometimes considered a synonym for web service, Web 2.0 web applications have moved away from a service-oriented architecture (*SOA*) with *SOAP*-based web services towards more cohesive collections of *RESTful* web resources. These *RESTful* web *API*s are accessible via standard *HTTP* methods by a variety of *HTTP* clients including browsers and mobile devices.

Appendix B. Table of Figures

Appendix C. Use Cases

Sales Team Performance Visualization Use Case

The User eXperience (UX) team, through an innovative visualization known as TeamPerformanceViz, will demonstrate to Kurt, a sales manager, that there is a better way in which he can see the performance of his team (composed by sales representatives) using a Tablet and/or Desktop browser. This visualization should replace the classical-static-bar-chart that he has been using for years. You, as part of the UX team, will be in charge of implementing a rapid functional prototype for TeamPerformanceViz.

Appendix D. Web API

URL

```
http://<server>/retrieveTeamPerformanceData
```

Request

```
{
  "date": "09-04-2013" | date,
  "limit": false | number,
  "orderKey": "full-name" | "annual-goal-met-percentage" | "goal-completion-status",
  "orderDirection": "asc" | "desc"
}
```

- All parameters are optional. In other words, they can be omitted from the request.
- If a parameter is not specified then its `default` value must be used at backend.
- `date` value should be specified in the following format `"MM-DD-YYYY"`.
- `number` should be any not negative numeric value.
- If `limit` is equal to `false` then maximum 20 items must be served.

Response

Successful Response

```
{
  "success": true,
  "teamPerformance": [
    {
      "name": "Aaliyah Henry",
      "finalTarget": 330000,
      "goalStatus": "missed",
      "data": [
        {
          "type": "target",
          "date": "2013-01-01T23:15:30Z",
          "amount": 0
        },
        …
        {
          "type": "actual",
          "date": "2013-09-04T16:00:53Z",
          "amount": 330000
        }
      ]
    }
    …
  ]
}
```

Non-Successful Response

```
{
  "success": false,
  "errorMessage": "Error Message"
}
```

Appendix E. Deliverable

Deliverable must include:

- A functional prototype of TeamPerformanceViz specified above.
- A PowerPoint[2] Presentation which must be used for demoing the TeamPerformanceViz implementation.

Policies

Deliverable must follow next policies:

Legal

As the disclosure agreement you have signed indicates:

- Code, documents/assets (mockups, red-lines images, ai/doc/pdf/psd/svg files, etc.) **MUST NOT** be shared/hosted/posted on the Internet[3] (and/or any person alien to this project/hackathon) unless you have explicit permission to do so.
- It is **PROHIBITED** to take screenshots/pictures of TeamPerformanceViz and/or its implementation (code, diagrams, documents, etc.) and/or you implementing it and share/host/post them on the Internet[3] (and/or any person alien to this hackathon) without having explicit permission to do so.
- It is **PROHIBITED** to mention/share/post partially or completely any information about TeamPerformanceViz and/or its implementation (code, diagrams, documents, etc.) on the Internet[3] (and/or any person alien to this hackathon) without having explicit permission to do so.

⚠ Always **ASK FIRST** if you are not sure that what you are going to do is legal.

Implementation

- Your TeamPerformanceViz implementation must run well on multiple Tablet and/or Desktop browsers (in order of importance): Google Chrome, Mozilla Firefox and Apple Safari.
- Frontend must be developed in *HTML5*, *CSS3* and plain *JavaScript*:
  - The use of *JavaScript* libraries/frameworks such as *jQuery*, *Dojo*, *YUI*, *Ext*, *Sencha*, *MooTools*, *Prototype*, *script.aculo.us*, *Backbone*, *Angular*, *Ember*, etc. is not permitted (sorry ☹).
  - The use of micro *JavaScript* libraries/frameworks such as *underscore*, *lodash*, *xui*, *zepto*, *handlebars*, etc. is not permitted (sorry once again ☹).
  - The use of the *D3.js* library is suggested. However, the use of any other *MIT*/*BSD*/*Apache 2* licensed *JavaScript* charting/visualization library such as *Processing.js*, *Raphael.js*, *Paper.js* etc. is permitted as long as it does not depend on any other *JavaScript* library/framework.
- Backend can be developed in any technology/programming language you choose (*Java*, *PHP*, *Python*, *JavaScript*, etc.) as long as the output is valid *JSON* data. *JSON* output must have a structure similar to the

[2] We know there are better alternatives (like Prezi). However, we MUST not break legal policies.
[3] With Internet we mean: GitHub, Bitbucket, Heroku, nodejitsu, Amazon AWS, Stack Overflow, bl.ocks.org, JSFiddle, CodePen, JS Bin , Facebook, Twitter, Google+, Google Drive, Dropbox, LinkedIn, Instagram, Pinterest, Blogger, WordPress, etc.

one within the `team-performance.json` file contained within the zip file received by email by your team lead.
- Code MUST include the correspondent Open Source licenses.
- Code MUST include your names and team id/name.
- Code must be as clean and maintainable as possible. Have in mind that someone else will edit it in the future. He/She will really appreciate it.
- Ensure the best possible performance for your TeamPerformanceViz implementation. As users, we all hate waiting for something to be downloaded.
- The due date for delivering your code and presentation is Friday September 11th, 2013 12.00 hrs.

Presentation

A PowerPoint[2] presentation MUST be generated to demo your TeamPerformanceViz implementation.

> ℹ Have in mind that judges will be UX people. Said that, a good presentation is as important as a good implementation. Try to make us choosing your implementation by leading an awesome presentation.

- The whole team must be present during the presentation. We will randomly ask questions to any of your team mates during the Q&A (Questions and Answers) phase.
- Ideas should be clear and concise. We always try to follow the KISS principle.
- Text should be readable.
- No restrictions about amount of slides, colors or themes.
- Presentation must include your names (of course), team id/name and a picture of the whole team.
- Presentation must include an explanation about your implementation.
- Presentation must not take more than 10 minutes.

Judging Criteria

The following criteria will be used to judge competition entries. Keep these criteria in mind as you develop your project:

Criteria		Description	Points
Implementation		Various aspects of the implementation will be judged. A good implementation should respond satisfactorily to most of the questions listed below.	**80**
	Completion and Requirement Compliance	• Percentage of completion. • Does the implementation comply with the requirement?	40
	User eXperience	• How does the app feel (in general) in a Tablet and/or Desktop browser? • Are touch/mouse gestures properly supported? • Animations/Transitions are smooth? • Does the UI comply with the requirement (red-lines)?	15
	Code Maintainability	• Is the code well formatted? • Is the code easily understandable (Hungarian notation, appropriate variable names, etc.)? • Is the code well commented? Does it contain unnecessary comments? • Does the code contain Open Source licenses? • Does the code contain magic numbers? Global variables? Feature detection? Media queries? • Which Best Practices were applied to code? • Is the visualization easy to skin?	10
	Cross-Browser Compatibility	• Does the app work fine in Tablet and/or Desktop browsers?	10

		• Team is able to demonstrate code for making the app to work fine on multiple browsers. • What was the most challenging cross-browser issue the team had to solve?	
	Performance	• What did the team does to achieve a good performance?	3
	Errors Survival	• Did the app survive to data errors?	2
Presentation		A good presentation should contain all of these features: • The project display completely covers all aspects of the work. • Ideas and actual work are clearly explained. • Team members explained their work and answered questions to my satisfaction. • Team members clearly worked as a team in that they could all answer questions about any part of the project. • Team members can demonstrate that they have acquired new knowledge during this competition. • The visual presentation was appropriate, pleasing to look at, and uncluttered. • Presentation took no more than the specified time.	**20**
			Total: 100

Appendix F. Zip File

Your Team Lead has received a zip file with the following contents:

- `assets` folder. Contains assets (such as *SVG* files) required for implementing TeamPerformanceViz UI.
- `team-performance.json` file. *JSON* file which should be used to generate/populate the TeamPerformanceViz. This *JSON* file should also be used as a base for implementing TeamPerformanceViz backend.
- `TeamPerformanceViz Software Requirements Specification.pdf` file. This specification document.

Appendix G. Links

D3.js - Data Driven Documents
http://d3js.org

Datavisualization.ch Selected Tools
http://selection.datavisualization.ch

Open Source Software: Restricted vs. Permissive Licenses
http://www.iplawforstartups.com/open-source-software-restricted-vs-permissive-licenses/

JavaScript Best Practices
http://dev.opera.com/articles/view/javascript-best-practices/

Feature detection is not browser detection:
http://www.nczonline.net/blog/2009/12/29/feature-detection-is-not-browser-detection/