**AUTOMATION OF REPORTING DASHBOARD**

**Introduction:**

The reporting dashboard is an online web-app tool which helps to monitor the customer projects details and the related financial data generated on the basis of the web-form which is discussed previously. There are several functional enhancements and added facilities which will be discussed in the following sections in details.

**Requirement Details:**

The customer reports and the financial details were usually captured through a master excel sheet, which was maintained and managed by project manager. But due to increasing amount of projects and implementation of automated project details collection, the details maintaining through excel sheet raises difficulty and time consumption.

**Technology Used –**

* **Front End -**  Html, CSS, Bootstrap Framework, JavaScript
* **Back End -** PHP, MySQL, Rest API
* **DataTable -** Bootstrap Data table, X-editable
* **Graphs / Charts** **& Visualization –** AmCharts

**Tools Used –**

* **Text Editor -** Sublime Text, Visual Studio Code, Atom, Brackets
* **API Testing -** Postman
* **Web Browser –** Google Chrome (Latest), Mozilla Firefox(Latest)
* **Local Server –** Xampp, Apache, InnoDb

**Pros. & Cons. Of excel vs web-app**

|  |  |
| --- | --- |
| **Excel Sheet / Desktop App** | **Automated Web-App** |
| **Pros.**   * Not dependent on the Internet | **Cons.**   * Dependent on the internet |
| **Cons.**   * Local data storage (Not secure, No automated backup) * Limitation in automation * Manual data backup * Not easy scalable * Libraries and packages are stored locally * Enormous package or installation size * Optimization is dependent on third party update and time consuming * No data replication (possible manually) | **Pros.**   * Cloud data storage (Secure, automatic backup) * Unlimited possibilities of automation * Automatic server backup * Easy Scalable * Libraries and packages can be accessed via CDN * Minimal package size, No installation required * Optimization is not dependent on any third-party vendor * Multiple server replication |

**Features –**

* **Interactive 3D Graph:** The dashboard is featured with interactive 3D graph which is capable of displaying data classified according to the requirements. There are multiple pie-charts and bar –charts based on ***Project Classification****,* ***Project Status, Subscriptions*** and ***Operations Support Status.*** All of this graphs are interactive and can be modified based on preferences. Graphs are displayed based on the data which is collected from the project manager into the database. The graphs will auto update if any new data has been entered into the database. Here Rest API has been implemented in the backend which is collecting the data through a url in json format and throwing the data directly into AmCharts.
* **Auto Updated Data-Tables:** The data tables which are implemented here are working in the same manner as the graphs. The data is being feed as JSON format using Rest API and as result the data is directly being updated whenever any changes are detected.
* **Master Project Tracker:** The master tracker is the detailed table for projects. There are basically two sections in the master tracker which is implemented as a web app**.**
  + - **View –** The Master tracker view only displays the recorded data collected from the user / project manager. The view section is completely based on bootstrap data table. The details can be modified as the serial number of projects. There’s also a search bar from where the projects can be searched based on any details.

The view section also holds a major functionality for extract the data in specific formats like ***excel, csv, pdf, doc, docx, etc.*** Which is a very important functionality for creating reports or keep tracking of documented data for customer purposes.

* + - **Edit –** In this section the data can be edited which is previously added to the database through the form. This edit section is created by **X-Editable**. The technical details will be discussed later. The data is editable by just on clicking on any data-cell. On click a pop-up will appear where the admin can remove the previous data and type in or select the new updated data. After hitting the enter button from keyboard the data will be placed instead and instantly update the database also using jQuery.
* **Add New Project Details:** The web application consist of a separate section toadd new project details. The form will collect data information from the user and store the data into the database. It’s been already mentioned that the data is being collected from the database using API. So any new data collected using this form will automatically reflect into the master tracker. The data entered through this form is absolutely editable later onwards. The technical features will be explained later.
* **Finance:** The finance section is implemented similarly like the master tracker section, separated by two different sections – **View** & **Edit**.
  + - **View-** The view section helps the user to view and monitor the financial details. All the exporting options are also given into the view section. The view section has search facilities. Using project name and project ID an individual can query a search inside the data-table. The project name and the ID will automatically be collected from the master tracker table as both the table is connected to each other using MySQL foreign key.

* + - **Edit-** The edit section which was created using JavaScript X-editable is able to update or add new input into the database by simply clicking on the section. Just similar to the master tracker it will store the data into the database, but into a separate table.
* **Upload Section:** The web app has a dedicated section for uploading documents, pdf or pictures. The files uploaded in this section will be stored in a specific folder inside the server storage. The uploaded files are displayed as a list and can be downloaded in the particular format in which the file was uploaded. Files can be chosen from local machine and can be uploaded into the server. This is an added functionality into the web app. The admin can store generated reports or excel sheet in itself into the web-app. So it can be kept as a backup side by side of the local files.

**TECHNICAL SPECIFICATION**

**TECHNICAL TERMINOLOGIES**

Here we will discuss the brief introduction and the perquisites of all the technical terms we have mentioned and used into this project. Before we go into the deep technical details or codes it is required that we discuss the basic terminologies and have a good understanding of the technological standards.

**API Services –**

API (Application Programming Interfaces) is a smart method for implementing a communication between the web components or application. Like Client Server communication. The internet is largely congested with APIs as we are growing smarter and smart applications are being developed. In modern days IoT is mostly based on API services. Smart like autonomous cars, smart home, smart parking this kind of things requires API services to communicate.

But, the question is *why API, what is the specialty in it?*

Here comes the answer. Implementing IoT or any kind of big projects require platform independency. As multiple device platforms are communicating in a system. Big implementation of a company project also requires API. Which removes the dependency of a particular language or skill for the developers.

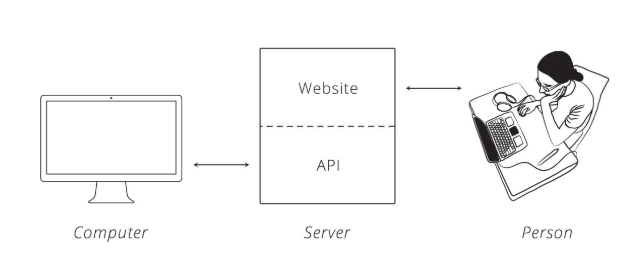
That is why big companies are investing and implementing on API technologies. APIs return faster response and more security into the system. APIs use token authentication for sending and receiving data. So only specific acknowledgement will transfer the data which adds more security to the data itself. It is obvious that securing data is very important while the services are provided to large amount of customers. With so many companies investing in this new venture of business, acquiring a working understanding of APIs has become a relevant choice in careers in the software industry.

While working with servers, which is a typical high configuration computer with no monitor, keyboard or mouse to look into it. The software engineers has to access it remotely to work on them. Servers are used for all sorts of projects. For email servers, for launching website, storing data etc. In a typical way an IT folk just write down the code and copies it to the server, and the server keeps running the program continuously.

Websites are designed to provide required information to the user without ease. Human are extremely capable of taking visual information and combine with experiences to derive meaning. That is why we can look into a form of a website and as soon as we see a little box with a phrase “First Name:” we can easily understand that we are supposed to type in the word which is informally used to identify ourselves.

Yet, what happens is that when we face a very time-intensive task, like copying contact information of thousands of customers from on site to another, it becomes a necessity that we delegate this work to a computer so it can be done automatically, quickly and accurately. Unfortunately, the characteristics that make websites optimal for humans make them difficult for computers to use.

The solution is an API. An API is the tool that makes a website’s data digestible for a computer. Through it, a computer can view and edit data, just like a person can, by loading pages and submitting forms.



***Fig 1:*** *Communication with Server using API*

Making data easier to work with is good because it means people can write software to automate tedious and labor-intensive tasks. What might take a human hours to accomplish can take a computer seconds through an API.

When two systems (websites, desktops, and smartphones) link up through an API, we say they are "integrated." In an integration, you have two sides, each with a special name. One side we have already talked about: the server. This is the side that actually provides the API. It helps to remember that the API is simply another program running on the server. It may be part of the same program that handles web traffic, or it can be a completely separate one. In either case, it is sitting, waiting for others to ask it for data.

The other side is the "client." This is a separate program that knows what data is available through the API and can manipulate it, typically at the request of a user. A great example is a smartphone app that syncs with a website. When you push the refresh button in your app, it talks to a server via an API and fetches the newest info.

The same principle applies to websites that are integrated. When one site pulls in data from the other, the site providing the data is acting as the server, and the site fetching the data is the client.

**The Protocol of The web –**

People create social etiquette to guide their interactions. One example is how we talk to each other on the phone. Imagine yourself chatting with a friend. While they are speaking, you know to be silent. You know to allow them brief pauses. If they ask a question and then remain quiet, you know they are expecting a response and it is now your turn to talk.

Computers have a similar etiquette, though it goes by the term "protocol." A computer protocol is an accepted set of rules that govern how two computers can speak to each other. Compared to our standards, however, a computer protocol is extremely rigid. Think for a moment of the two sentences "My favorite color is blue" and "Blue is my favorite color." People are able to break down each sentence and see that they mean the same thing, despite the words being in different orders. Unfortunately, computers are not that smart.

For two computers to communicate effectively, the server has to know exactly how the client will arrange its messages. You can think of it like a person asking for a mailing address. When you ask for the location of a place, you assume the first thing you are told is the street address, followed by the city, the state, and lastly, the zip code. You also have certain expectations about each piece of the address, like the fact that the zip code should only consist of numbers. A similar level of specificity is required for a computer protocol to work.

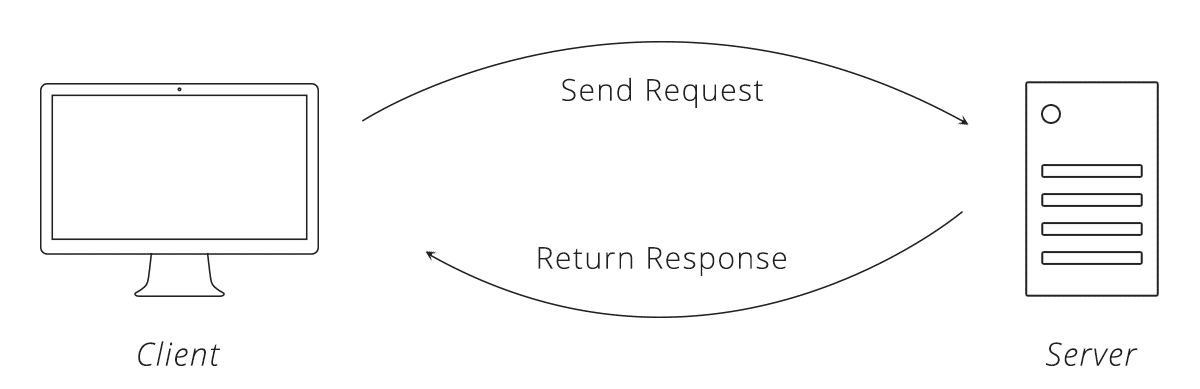
There is a protocol for just about everything, each one tailored to do different jobs. You may have already heard of some: Bluetooth for connecting devices, and POP or IMAP for fetching emails.

On the web, the main protocol is the Hyper-Text Transfer Protocol, better known by its acronym, HTTP. When you type an address like http://example.com into a web browser, the "http" tells the browser to use the rules of HTTP when talking with the server.

With the ubiquity of HTTP on the web, many companies choose to adopt it as the protocol underlying their APIs. One benefit of using a familiar protocol is that it lowers the learning curve for developers, which encourages usage of the API. Another benefit is that HTTP has several features useful in building a good API, as we'll see later.

**HTTP Requests –**

Communication in HTTP centers around a concept called the **Request-Response Cycle**. The client sends the server a request to do something. The server, in turn, sends the client a response saying whether or not the server could do what the client asked.



***Fig:*** *The Request-Response Cycle*

To make a valid request, the client needs to include four things:

1. URL (Uniform Resource Locator)
2. Method
3. List of Headers
4. Body

That may sound like a lot of details just to pass along a message, but computers have to be very specific to communicate with one another.

**URL**

URLs are familiar to us through our daily use of the web, but have you ever taken a moment to consider their structure? In HTTP, a URL is a unique address for a thing (a noun). Which things get addresses is entirely up to the business running the server. They can make URLs for web pages, images, or even videos of cute animals.

APIs extend this idea a bit further to include nouns like customers, products, and tweets. In doing so, URLs become an easy way for the client to tell the server which thing it wants to interact with. Of course, APIs also do not call them "things", but give them the technical name "resources."

**Method**

The request method tells the server what kind of action the client wants the server to take. In fact, the method is commonly referred to as the request "verb."

The four methods most commonly seen in APIs are:

* **GET** - Asks the server to retrieve a resource
* **POST** - Asks the server to create a new resource
* **PUT** - Asks the server to edit/update an existing resource
* **DELETE** - Asks the server to delete a resource

Here's an example to help illustrate these methods. Let's say there is a pizza parlor with an API you can use to place orders. You place an order by making a POST request to the restaurant's server with your order details, asking them to create your pizza. As soon as you send the request, however, you realize you picked the wrong style crust, so you make a PUT request to change it.

While waiting on your order, you make a bunch of GET requests to check the status. After an hour of waiting, you decide you've had enough and make a DELETE request to cancel your order.

**Headers**

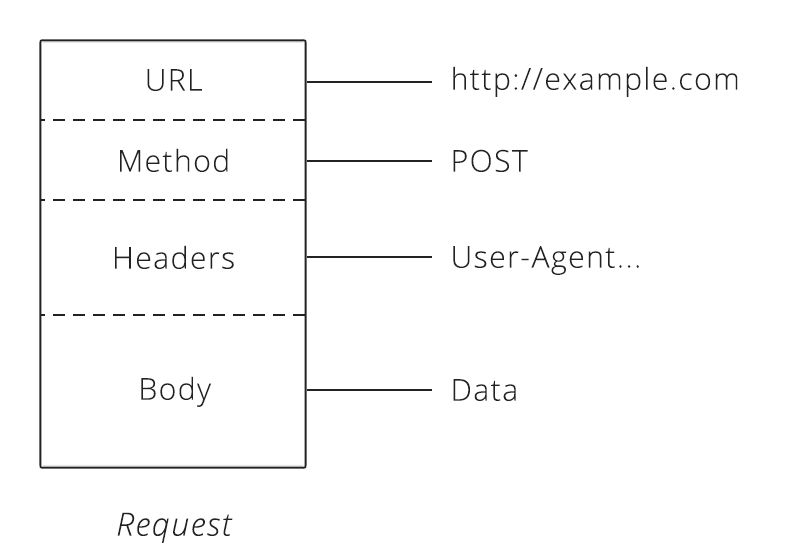
Headers provide meta-information about a request. They are a simple list of items like the **time** the client sent the request and the **size** of the request body.

Have you ever visited a website on your smartphone that was specially formatted for mobile devices? That is made possible by an HTTP header called **"User-Agent."** The client uses this header to tell the server what type of device you are using, and websites smart enough to detect it can send you the best format for your device.

**Body**

The request body contains the data the client wants to send the server. A unique trait about the body is that the client has complete control over this part of the request. Unlike the method, URL, or headers, where the HTTP protocol requires a rigid structure, the body allows the client to send anything it needs.

These four pieces — URL, method, headers, and body — make up a complete HTTP request.



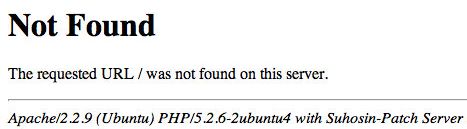
***Fig:*** *The Structure of an HTTP Request*

**HTTP Responses**

After the server receives a request from the client, it attempts to fulfill the request and send the client back a response. HTTP responses have a very similar structure to requests. The main difference is that instead of a method and a URL, the response includes a status code. Beyond that, the response headers and body follow the same format as requests.

**Status Codes**

Status codes are three-digit numbers that each have a unique meaning. When used correctly in an API, this little number can communicate a lot of info to the client. For example, you may have seen this page during your internet wanderings:

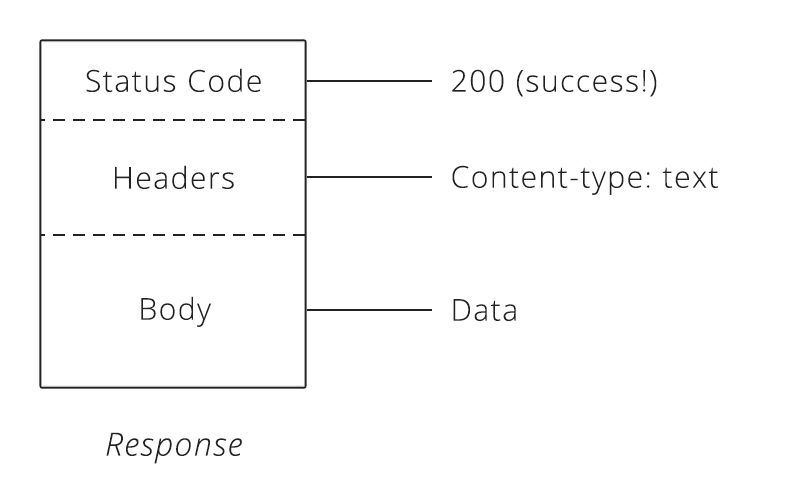


***Fig:*** *Default 404 web page*

The status code behind this response is **404**, which means **"Not Found."** Whenever the client makes a request for a resource that does not exist, the server responds with a 404 status code to let the client know: *"that resource doesn't exist, so please don't ask for it again!"*

There is a slew of other statuses in the HTTP protocol, including **200** ("success! that request was good") to **503** ("our website/API is currently down.") We'll learn a few more of them as they come up in later chapters.

After a response is delivered to the client, the Request-Response Cycle is completed and that round of communication over. It is now up to the client to initiate any further interactions. The server will not send the client any more data until it receives a new request.



***Fig:*** *The Structure of an HTTP response*

**How APIs Build on HTTP**

By now, we can see that HTTP supports a wide range of permutations to help the client and server talk. So, how does this help us with APIs? The flexibility of HTTP means that APIs built on it can provide clients with a lot of business potential.

This versatility in the HTTP protocol extends to other parts of a request, too. Some APIs require a particular header, while others require specific information inside the request body. Being able to use APIs hinges on knowing how to make the correct HTTP request to get the result you want.

**Representation of Data-**

When sharing data with people, the possibilities for how to display the information is limited only by human imagination. A well-designed format is dictated by what makes the information the easiest for the intended audience to understand.

The same principle applies when sharing data between computers. One computer has to put the data in a format that the other will understand. Generally, this means some kind of text format. The most common formats found in modern APIs are **JSON (JavaScript Object Notation)** and **XML (Extensible Markup Language)**.

**JSON**

Many new APIs have adopted JSON as a format because it's built on the popular **Javascript programming language**, which is ubiquitous on the web and usable on both the **front**- and **back-end** of a web app or service. JSON is a very simple format that has two pieces: **keys** and **values**. Keys represent an attribute about the object being described.

Let's see how an example could look in JSON:

{

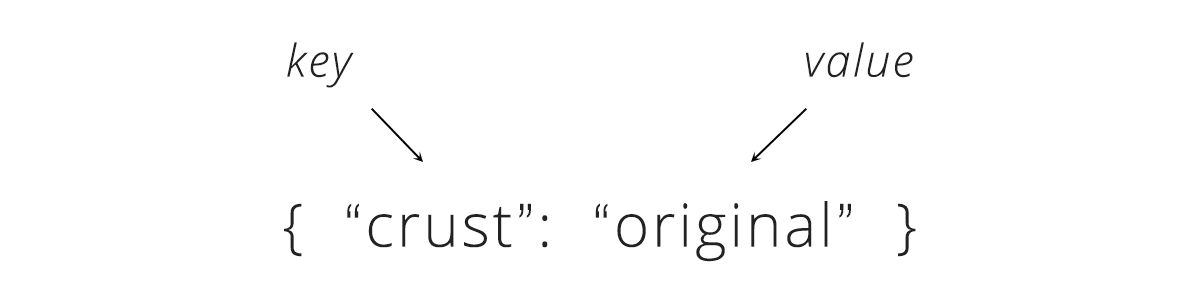
"crust": "original",

"toppings": ["cheese", "pepperoni", "garlic"],

"status": "cooking"

}

In the JSON example above, the keys are the words on the left: toppings, crust, and status. They tell us what attributes the pizza order contains. The values are the parts to the right. These are the actual details of the order.



***Fig:*** *JSON key and value*

If we read a line from left to right, you get a fairly natural English sentence. Taking the first line as an example, we could read it as, "the crust for this pizza is original style." The second line can also be read — in JSON, a value that starts and ends with square brackets ([]) is a list of values. So, we read the second line of the order as, "the toppings for this order are: cheese, pepperoni, and garlic."

Sometimes, it’s required to use an object as the value for a key. Let's extend our pizza order with customer details so you can see what this might look like:

{

"crust": "original",

"toppings": ["cheese", "pepperoni", "garlic"],

"status": "cooking",

"customer": {

"name": "Brian",

"phone": "573-111-1111"

}

}

In this updated version, we see that a new key, "customer", is added. The value for this key is another set of keys and values that provide details about the customer that placed the order. Cool trick, huh?! This is called an Associative Array. Don't let the technical term intimidate you though - an associative array is just a nested object.

**XML**

XML has been around since 1996 [1](https://zapier.com/learn/apis/chapter-3-data-formats/#footnote-1). With age, it has become a very mature and powerful data format. Like JSON, XML provides a few simple building blocks that API makers use to structure their data. The main block is called a *node*.

Let's see what our pizza order might look like in XML:

**<order>**

**<crust>**original**</crust>**

**<toppings>**

**<topping>**cheese**</topping>**

**<topping>**pepperoni**</topping>**

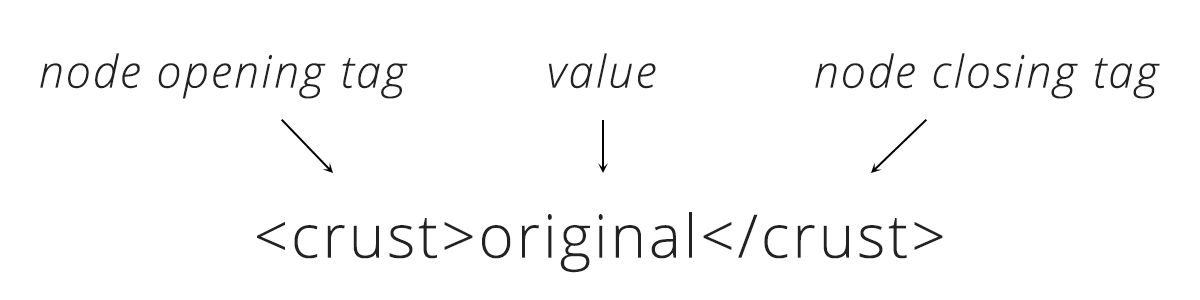
**<topping>**garlic**</topping>**

**</toppings>**

**<status>**cooking**</status>**

**</order>**

XML always starts with a root node, which in our pizza example is "order." Inside the order are more "child" nodes. The name of each node tells us the attribute of the order (like the key in JSON) and the data inside is the actual detail (like the value in JSON).



***Figure 2.****XML node and value.*

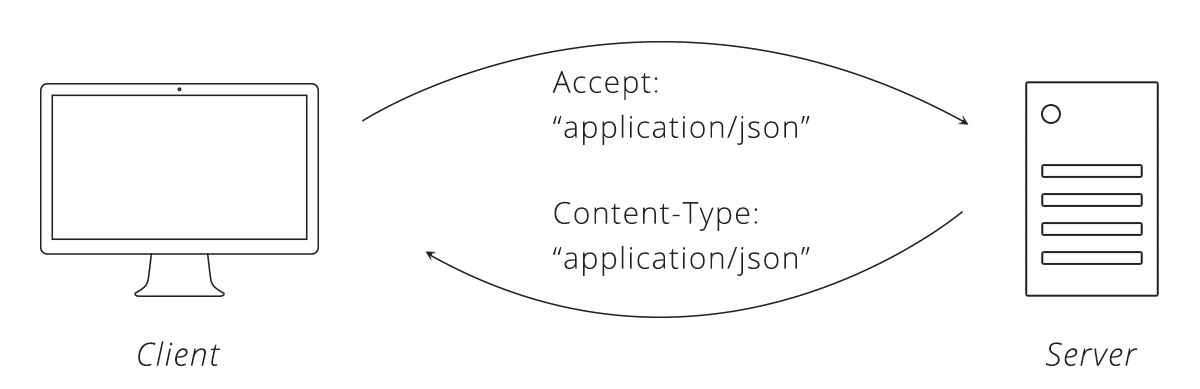
You can also infer English sentences by reading XML. Looking at the line with "crust", we could read, "the crust for the pizza is original style." Notice how in XML, every item in the list of toppings is wrapped by a node. You can see how the XML format requires a lot more text to communicate than JSON does.

**How Data Formats Are Used In HTTP**

When the client sends the Content-Type header in a request, it is telling the server that the data in the body of the request is formatted a particular way. If the client wants to send the server JSON data, it will set the Content-Type to "application/json." Upon receiving the request and seeing that Content-Type, the server will first check if it understands that format, and, if so, it will know how to read the data. Likewise, when the server sends the client a response, it will also set the Content-Type to tell the client how to read the body of the response.

Sometimes, the client can only speak one data format. If the server sends back anything other than that format, the client will fail and throw an error. Fortunately, a second HTTP header comes to the rescue. The client can set the *Accept* header to tell the server what data formats it is able to accept. If the client can only speak JSON, it can set the Accept header to "application/json." The server will then send back its response in JSON. If the server doesn't support the format the client requests, it can send back an error to the client to let it know the request is not going to work.

With these two headers, Content-Type and Accept, the client and server can work with the data formats they understand and need to work properly.



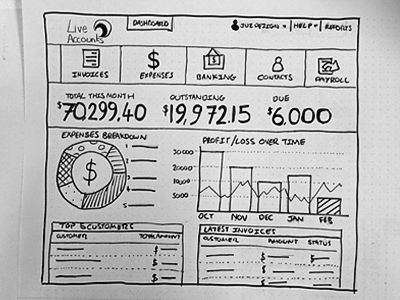
***Figure 3.****Data format headers.*

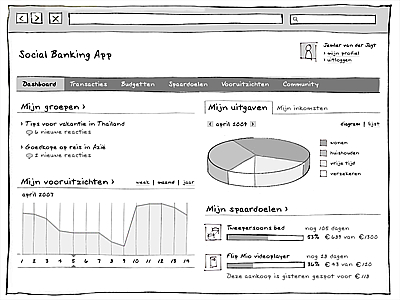
**DESIGN APPROACH AND DETAILS**

**4.1 Design Approach:**

In the beginning stage the requirements were collected from the user. For this case the requirements were gathered from the project manager. The purpose of this dashboard is to make the project manager’s work load lower and to automate stuffs which was previously handled by manual effort.

So first after requirement gathering a thorough research was done by googling up existing dashboard projects and templates. Then suitable hand drawn wireframe sketches were made and those sketches were displayed to the user to make sure that the design or model is fulfilling his expectations. Once the final model was selected the rough hand drawn sketch were converted into a digital wireframe model using wireframe tools. Here I have used MS Paint application and some existing online tools.





***Fig:*** *Dashboard Wireframe Rough Sketch*

A digital model of the rough sketch was created to maintain the standard during the build process. The look and feel must be clearly following the prototype as it is fulfilling the expectation of the customer. By taking the sketch as a model to follow the real application was build, using various tools and web framework.

The same process should be followed when we are doing the same for a desktop application also. We have already discussed that why it’s more preferable to build a web app rather than a desktop app.

**4.2 Materials and methods:**

For building a web application with the previous mentioned requirements here we have used basic html, css and javascript. To make the things more visually attractive to the customer we have used Bootstrap framework, which is a very popular framework available for free.

**HTML5:**

HTML stands for **H**yper**t**ext **M**arkup **L**anguage, and it is the most widely used language to write Web Pages.

* **Hypertext** refers to the way in which Web pages (HTML documents) are linked together. Thus, the link available on a webpage is called Hypertext.
* As its name suggests, HTML is a **Markup Language** which means you use HTML to simply "mark-up" a text document with tags that tell a Web browser how to structure it to display.

Originally, HTML was developed with the intent of defining the structure of documents like headings, paragraphs, lists, and so forth to facilitate the sharing of scientific information between researchers.

Now, HTML is being widely used to format web pages with the help of different tags available in HTML language.

## Basic HTML Document

In its simplest form, following is an example of an HTML document −

<!DOCTYPE html>

<html>

<head>

<title>This is document title</title>

</head>

<body>

<h1>This is a heading</h1>

<p>Document content goes here.....</p>

</body>

</html>

**CSS:**

Cascading Style Sheets (**CSS**) is a simple mechanism for adding style (e.g., fonts, colors, spacing) to Web documents. These pages contain information on how to learn and use**CSS** and on available software. They also contain news from the **CSS** working group.

CSS are added to the HTML by adding a reference path link to it’s header.

**Bootstrap:**

Build responsive, mobile-first projects on the web with the world’s most popular front-end component library.

Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. Quickly prototype your ideas or build your entire app with our Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery.

We have used Bootstrap Version 3 for building this web App.

**JavaScript:**

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The ECMA-262 Specification defined a standard version of the core JavaScript language.

* JavaScript is a lightweight, interpreted programming language.
* Designed for creating network-centric applications.
* Complementary to and integrated with Java.
* Complementary to and integrated with HTML.
* Open and cross-platform

**jQuery:**

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

**Ajax:**

AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

* Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
* Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
* With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
* XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
* AJAX is a web browser technology independent of web server software.
* A user can continue to use the application while the client program requests information from the server in the background.
* Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
* Data-driven as opposed to page-driven.

**DataTables:**

DataTables is a plug-in for the jQuery Javascript library. It is a highly flexible tool, built upon the foundations of progressive enhancement, that adds all of these advanced features to any HTML table. We can see the huge application of DataTables in this project for implementing the data viewing tables.

**X-Editable:**

This library allows you to create editable elements on your page. It can be used with any engine (bootstrap, jquery-ui, jquery only) and includes both popup and inline modes.

**4.2 Codes and Standards**

We will discuss the web- application part by part. Describing each component and their implementation code.

**Login:**

The login is implemented as simple to load it faster and efficiently.



1. **SCHEDULE, TASKS AND MILESTONES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NO | Task | Start Date | End Date | Status |
| 1 | Understanding Service Request Form (Excel Sheet) | 3rd Sep 2018 | 14th Sep 2018 | Completed |
| 2 | Installing Required Software | 17th Sep 2018 | 21st Sep 2018 | Completed |
| 3 | Brief Review on Understanding/ Clarification of user inputs | 24th Sep 2018 | 28th Sep 2018 | Completed |
| 4 | Prototype or Skeleton of Service Request Form in MS Paint | 1st Oct 2018 | 12th Oct 2018 | Completed |
| 5 | Review of prototype and modifications | 15th Oct 2018 | 31st Oct 2018 | Completed |
| 6 | Designing of Frontend | 1st Nov 2018 | 23rd Nov 2018 | Completed |
| 7 | Review on frontend design | 26th Nov 2018 | 30th Nov 2018 | Completed |
| 8 | Designing of Backend | 3rd Dec 2018 | 14th Dec 2018 | Completed |
| 9 | Review on frontend design | 17th Dec 2018 | 31st Dec 2018 | Completed |
| 10 | Connecting Web form to the Database | 1st Jan 2019 | 18th Jan 2019 | Completed |
| 11 | Final Review | 21st Jan 2019 | 31st Jan 2019 | Completed |