Improvement of Single Page Application in responsive design using Web API and Angular JS

G.Niranjana, V. Aumugam

Abstract*: In traditional web applications, the communication with the server is initiated by the client by sending a page request. The request is processed by the server and the resulting webpage is sent to the client. For subsequent interactions within the page like a link is navigated or a form is submitted with data, a new request is generated and sent to the server. The server repeats the sequence of action for processing the request and the response is generated by sending a new page to the client. When using Single Page Applications (SPAs), the initial request is processed and the entire page is sent to the client/browser, and further interaction takes place through Aax requests. This means the entire page is not reloaded only the portion of the page that has changed is updated by the browser. This approach increases the response time of the application. Emerging technologies like ASP.NET Web API, JavaScript frameworks like AngularJS and new styling features provided by HTML5 & CSS3 make it really easy to design and build SPAs. Our result shows 10 times improvement in initial loading time and the server response time is reduced to half.

Keywords: ASP.NET Web API, HTML5 And CSS3.

I. INTRODUCTION

For many years traditional web services have been built upon complex backend systems that serve static HTML files to the users. When the Web 2.0 era came around, techniques of dynamic content loading such as Asynchronous JavaScript and XML (AJAX) became popular. These techniques allow the application to fetch data from the server without reloading the entire page. However, most of the page is still initially rendered on the server side. What we are seeing today is a change, now the backend acts as a simple Application Programming Interface (API) and instead puts a lot more responsibility on the client. The client is often built in JavaScript to allow a flexible and interactive user interface. This kind of web application, known as Single-Page Application (SPA) or Single-Page Interface (SPI), radically changes the entire architecture. When the number of components in the client grows the complexity is also increased. This plays a crucial role in system design of a good structure or framework, since JavaScript allows many faults as it is an interpreted and loosely coupled language. Almost all popular sites like Gmail, Twitter and Foursquare are built with SPAs.

In Single page Application all the components required for the website are retrieved in a single page load. This provides the user to get an experience that he works on native application rather than a website. An SPA loads the full page in the initial page load and later upon request the new fragments of the page are loaded from the server. The feature updates are gradually downloaded by the server either as small fragments of the page or complete screen modules in order to avoid downloading unused features. Hence, “states” in a SPA are similar to “pages” in a traditional website. Because "state navigation" in the same page is analogous to page navigation, in theory, any page-based web site could be converted to single-page replacing in the same page only the changed parts result of comparing consecutive pages in a non-SPA. Single-page applications written in JavaScript are becoming more and more popular, but when the size of the applications grows the complexity is also increased. A good architecture or a suitable framework is therefore needed [4].

The aim of this paper is to improve the response of Single Page Application. In order to improve Single Page Application we propose an architecture using Web API and Angular JS frameworks and to design it in responsive mode.

The paper starts with analyzing the number of design patterns suitable for applications containing a graphical user interface. An architecture that targets single-page applications was designed based on the composition of design patters. The architecture was designed to make applications easy to develop, test and maintain. Important features that were considered are initial loading time, data synchronization and search engine optimization. A framework based on the architecture was implemented, tested and compared against other frameworks available on the market.

The framework that was executed was planned to be modular, supports routing and templates as well as a number of different drivers for communicating with a server-side database. The modules were designed with a variant of the pattern Model-View-Controller (MVC), where a presentation model was introduced between the controller and the view. This allows unit tests to bypass the user interface and instead communicate directly with the core of the application. After minification and compression, the size of the framework is less than 20 kB including all its dependencies. This results in a low initial loading time.

Finally, a solution that allows a JavaScript application to be fast, scalable and easy to maintain is obtained.

II. RELATED WORK

With the traditional approach HTML files are rendered with the current content of the database. When data is changed the page must be reloaded to force the HTML files to be re-generated from scratch. Even for sites using AJAX to

dynamically change parts of the page this interaction is the most common case. On the other hand, an SPA allows a more flexible and elegant way of dealing with data. Once the user has loaded an initial slim version of the site all the data can be fetched asynchronously.

Revised Version Manuscript Received on January 25, 2019.

G.Niranjana, Department of CSE, SRM University, Chennai, Tamil Nadu, India

V. Aumugam, Department of CSE, SRM University, Chennai, Tamil Nadu, India

This has several advantages. First of all, an initial page is now rendered faster since a smaller amount of data is transferred to the user. This is the case since every request does not have to include the presentation of the data. It also allows data to only be fetched one single time since the client now can act as a small cache memory of the database. All of this flexibility leads to a more responsive interface and in the end to a better user experience. Unfortunately this flexibility also comes with a price, the frontend is now a lot more complex than before. When the frontend grows, testability becomes a critical issue. In order to build a high quality application, the developers must be able to test the components of the system. An architecture that allows a high degree of testability is needed.

Since data is represented both at the client and at the backend side an interface is needed for the communication in between. This interface needs to be implemented, for purpose of fetching, inserting, updating and deleting data. Problems regarding synchronization between user sessions may also arise since the client now acts as a cache memory to the database. When a user modifies data this may affect other users that are online, i.e. concurrency problems can appear. The responsibility of synchronization between users is now put on the developer. All of this leads to a more complex design of the frontend.

When the frontend becomes bigger and more complex it also requires more memory to run. Since a user often only works with a particular part of a system at a time, it would be a waste of memory to have the entire application instantiated. A structure that allows the developer to decide what parts of the system to run would solve this problem. If the design of these parts of the system could also be re-used in other applications, then the efficiency of the development would be greatly increased. Another important aspect of software development is maintenance of the code. In bigger applications several developers are often working collaboratively to build the frontend of a system. Code that is easy to read and understand becomes a key to achieve high efficiency and good quality. The use of design patterns becomes an important part of the system design. Unfortunately, it is a complex choice to make since there are many aspects to consider.

Some criticism has been raised against the performance of SPAs in terms of initial loading time. Due to limitations of the HTTP architecture the initial loading time might be higher when using a JavaScript frontend compared to the traditional approach. This is the case since the client first downloads the initial page and then executes its JavaScript. When this is done the user can be routed to the corresponding page and its content can be loaded. This leads to a higher initial loading time compared to just downloading the plain page from the server. However, once the initial page has been loaded and rendered the JavaScript frontend will be able to perform a lot better. Since the initial loading time is a key factor when a user is deciding to visit a site or not, it has become a crucial aspect of JavaScript-based frontends.

Another problem that is introduced when building a frontend in JavaScript is that if users visit the site and have disabled or cannot run JavaScript, then they will not be able to see the content of the site. Even though only 2% of today's users have

JavaScript disabled this still represents a problem. Crawlers from search engines represent for example a small amount of the total number of users but they are still very important to consider. If a crawler cannot see the content of the page it is impossible for it to analyse what the site is all about. This would lead to a poor page rank at the search engine. For many of today's websites this is extremely important.

The goal of the paper is to come up with a system design of a framework for more lightweight single-page applications. The architecture shall be based on design patterns well suited for web frontends. To give a good view of how design patterns can be used in practice the project clarifies how they can be typically used in a web application. Based on the analysis at the system design stage the framework isimplemented, allowing applications to get a high degree of testability as well as encouraging code re-usage between projects. The initial loading time of the page is considered as an important aspect of the framework and my work aspires to provide a shorter loading time compared to existing frameworks. The paper also proposes a solution to the problem of search engine optimization for single-page applications.

By looking at design patterns suitable for graphical interfaces an abstract idea of how to form the system architecture was developed. This idea was then further refined to fit a structure that was more suitable for the web. Today there already exists a number of JavaScript frameworks, these were analysed so that different ways of approaching the problems could be found. The majority of the existing frameworks are using techniques that are well known, well tested and have a wide browser support. However, the upcoming HTML5 draft [6] contains new exciting techniques introducing new ways of approaching the problems. These techniques were taken into account when designing the architecture of the framework. During the design phase suitable design patterns were selected for the upcoming implementation of the framework. To be able to design the framework with a high degree of testability a test framework for JavaScript was used. It was important to understand how testing typically is carried out and what testing utilities that are commonly used. Once the system design was ready the framework was implemented.

To verify that the project fulfilled its requirements a simple test application was developed. This application was built upon the framework in JavaScript. The purpose was to demonstrate how a developer could use the framework and how the most common problems are solved. Finally the initial loading time and testability were verified by performing a number of tests. The results were then be compared against other competitors.

As previously described SPAs introduce new demands regarding architecture and structure on the client side. To understand how these problems can be approached it is important to understand the basics behind how a single-page application works [5,6].

As briefly mentioned in the introduction the initial loading time is a crucial metric to consider. The initial loading time is measured from the time the user has entered the URL in the browser until the page is rendered. Studies show that 25% of the users close a page if it takes more than 4 seconds to load, slow pages are simply less popular. In order to minimize the initial loading time the process can be divided into a number of phases. For SPAs the phases are as follows:

1. A user enters the URL in the browser
2. The page is downloaded
3. The JavaScript is downloaded
4. The JavaScript is parsed
5. The application is initialized
6. A module is loaded
7. If data from the database is needed, it is fetched
8. The page is rendered

The flow for SPAs can be compared to the flow for traditional sites that only got phase 1, 2, and 8. Hence traditional websites will perform better. It is essential that each and every phase in this flow is executed as fast as possible to minimize the total loading time. The developers can take action as soon as the application is loaded and for example present a loading screen. Once the data is fetched the loading screen can be replaced with the real page that the user requested. To keep the file size down it is essential to minimize the time it takes to download and parse the JavaScript. Features that are not used by all applications should not be bundled together with the framework, it would simply make the framework bigger than needed. By using a flexible dependency-handling the framework can be configured so that no unnecessary libraries are included [7,8,9].

III. PROPOSED SYSTEM

Single-page applications written in JavaScript are becoming more and more popular, but when the size of the applications grows the complexity is also increased. A good architecture or a suitable framework is therefore needed.

The objective of this paper is to come up with a system design of a framework for more lightweight single-page applications. The architecture shall be based on design patterns well suited for web frontends. To give a good view of how design patterns can be used in practice this paper clarifies how they can be typically used in a web application. Based on the analysis at the system design stage the framework is implemented, allowing applications to get a high degree of testability as well as encouraging code re-usage between 10 projects. The initial loading time of the page is considered as an important aspect of the framework and my work aspires to provide a shorter loading time compared to existing frameworks. This paper also proposes a solution to the problem of search engine optimization for single-page applications.

Existing System uses the native JavaScript, AJAX and WCF service. WCF service is hosted on a server and the application will retrieve data from the service using the AJAX call and will be updated in DOM.

Drawbacks of the existing system

- Does not have a JavaScript framework to reduce complexity
- Size of the application is big
- Difficulty of testing JavaScript applications
- Synchronization of data between different user sessions
- High initial loading time

Inability of search engines to index JavaScript applications

The paper was limited to only review design patterns that have a clear connection to frontend development, other design patterns were not considered. Since the paper focused on system design of JavaScript frontends, aspects regarding the backend were not included within the scope. The system design of the frontend was not limited to a certain type of backend, any language or design used in the backend shall be able to cope with the frontend as long as they have a common interface for communication. When developing the framework, support for older browsers on the market were not taken into account. The framework can rather be used as a showcase of what is possible to do within a near future.

Proposed system focuses on reducing the complexity, improving speed and Responsiveness of the application. In proposed system, we are using,

1. Angular JS, A JavaScript framework, to reduce the complexity of the code and improve data binding.
2. Web API, ASP.NET Web API is a framework, that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices.

3.1 SYSTEM ARCHITECTURE

The entire system can be divided into three separate components: a backend API, an API driver and the JavaScript application itself. The backend API provides the JavaScript application with data, and the driver takes care of the abstraction in between. By utilizing a driver it becomes possible to replace the backend or frontend independently from each other. The user interacts with the JavaScript application by either providing a URL to the system, or in terms of user input such as clicking and pressing buttons [2]. The application can at any time change the DOM (Document Object Model) to give the user feedback of the provided input actions. The DOM is the browser's representation of the elements on the page and is used when rendering it. An abstract view of the architecture seen from a system-level point of view can be seen in the fig. 1 Black arrows drawn with a solid line represent a hard dependency where communication is done through traditional function calls. Black arrows drawn with a dashed line is used when the communication is done through the observer pattern. At last,

blue arrows represent design patterns that are used in the component that the arrow points at. Many other frameworks do not have the API driver as a bridge between the backend API and the JavaScript application, making them tightly coupled together. For example, to change the backend from using AJAX requests to HTML5 web sockets would in most cases require a redesign of the entire JavaScript application.

By letting the driver have a well-defined interface the components can be replaced independently of each other. It also allows drivers to be re-used between projects since the interface is the same independently from the application [10].

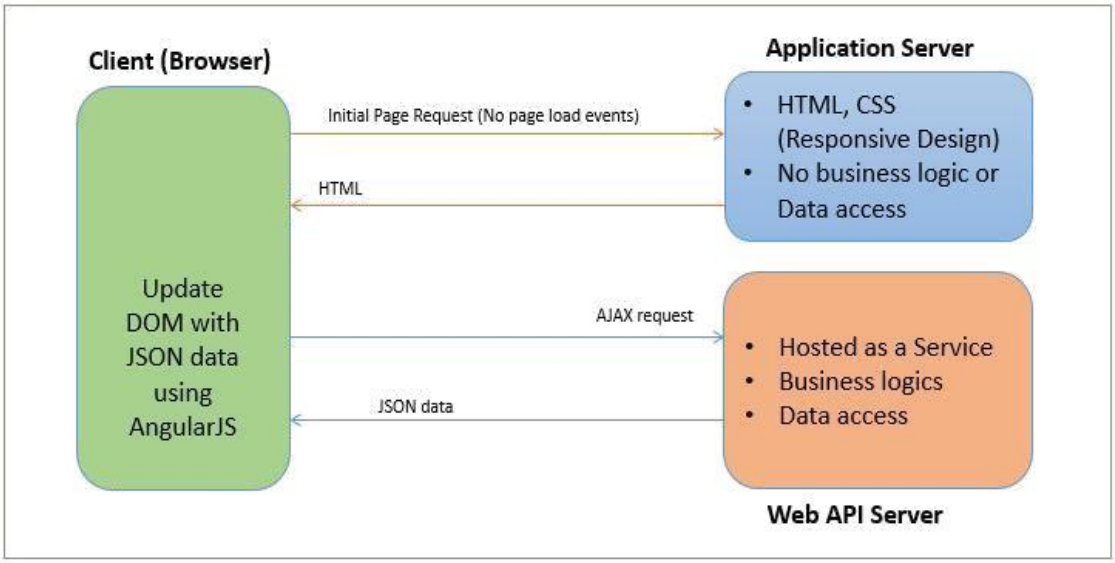

Fig. 1 System Architecture

3.2 WEB API

ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. With Web API content negotiation, we can return data based on the client requests. What we mean is, if the client is requesting the data to be returned as JSON or XML, the Web API framework deals with the request type and returns the data appropriately based on the media type [12]. By default Web API provides JSON and XML based responses [3].

Web API is an ideal platform for building pure HTTP based services where the request and response happens with HTTP protocol. The client can make a GET, PUT, POST, and DELETE request and get the Web API response appropriately [1].

In summary, the Web API is,

- An HTTP Service
- Designed for broad reach
- Uses HTTP as an Application protocol, not a transport protocol

3.3 ANGULAR JS

AngularJS is a JavaScript framework. An extensible web applications can be built using this. This is completely built on HTML, JavaScript and CSS and will not having any other dependency to make it work [14].

Angular is very close to MVC and MVVM but Angular has been declared as MVW pattern where "W" stand for "Whatever" (whatever works for you). AngularJS is an open-source web application framework, maintained by Google and an AngularJS community of developers [13].

- Angular JS built on pure HTML, Javascript and CSS and it does not depend on any server-side technologies.
- This is a very light weight application.
- This works on the SPA (Single Page Application) principle.
- AS in Angular components are separated, so the application can be highly testable. This support TDD development.
- Less development effort, in any project HTML developers are different and they develop HTML pages and provide to the development team, developer then converts all the HTML files to cshtml or ASPX or any other technologies and continues working. In this case, double effort is required for development and testing.
- In AngularJS, development team can continue from the HTML page itself. •

A collaboration diagram of client server communication describes interactions among objects in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from class, sequence, and use case diagrams describing both the static structure and dynamic behaviour of a system.

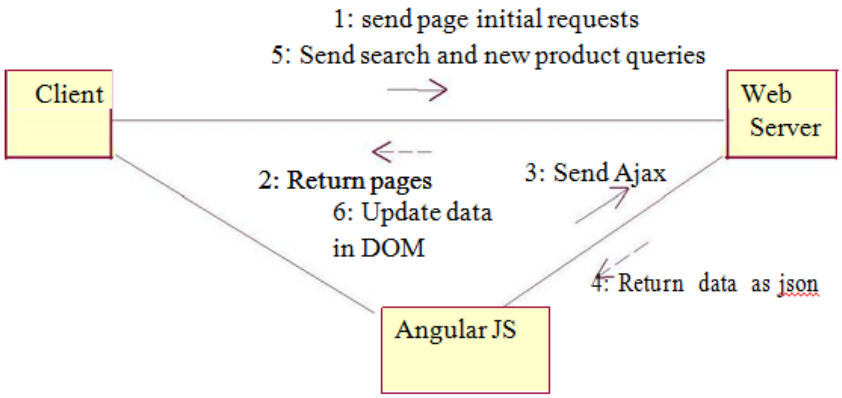

Fig. 2 Client server communication using AngularJS

IV. SYSTEM IMPLEMENTATION

As previously described SPAs introduce new demands regarding architecture and structure on the client side. To understand how these problems can be approached it is important to understand the basics behind how a single-page application works.

There is yet no formal definition that describes what an SPA is. Many developers have their own view of what it exactly means, but Ali Mesbah and Arie van Deursen have stated a quite clear definition in their paper about SPAs:

"The single-page web interface is composed of individual components which can be updated/replaced independently, so that the entire page does not need to be reloaded on each user action."

The definition includes a number of key attributes that help to define an SPA:

- Web interface - Used on the web, focuses on user interfaces.
- Individual components - It is divided into smaller components that cope with each other.
- Updates and replaces - A component can at any time be changed or replaced with another component.
- Reloading - The entire page is never reloaded even though new content may be loaded into some sections.
- User actions - The SPA is responsible for handling user actions such as • input from mouse and keyboard.

When running a JavaScript SPA the browser first makes a request to the web server. The web server will then respond with the JavaScript client including the resources needed to run it. Once the client is transferred it will be initialized and it will be ready to be run. When the user interacts with the client, such as clicking on graphical elements, this may require new data to be fetched from the server. Instead of reloading the entire page the client will instead make a small request to the API on the web server. The API is simply an interface that allows the clients to communicate with the server in a data format that is easy to understand for both the client and server. A typical flow of communication for an SPA can be seen in figure 2.1.

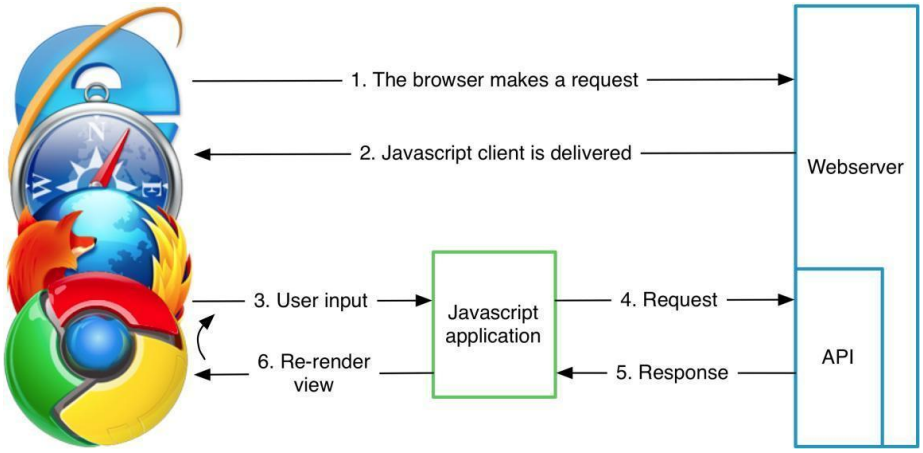

Fig 3 The flow of communication between a browser and the web server

4.1 API COMMUNICATION

In order for the client to dynamically fetch data from the server an API is needed in between. The API describes how data is being transferred, what data format is being used and what methods are available.

One of the most common ways of letting the client asynchronously communicate with the server is via AJAX (Asynchronous Javascript and XML). AJAX allows the client to asynchronously fetch and post data to the server after the page has been loaded. However, AJAX has no native support for letting the server push data to the client. The client must always initialize a new connection and then poll the server for new data. To enable the server to push data to the client other techniques can instead be used, e.g. the websocket API in the HTML5 draft which supports bi-directional communication. This will ease the load of the server and allows a faster and more responsive system.

When using a HTTP-based data transportation some kind of architectural structure is needed. The architecture describes how messages are formed so that both parties understand each other, e.g. what methods are available and how data is transferred. It also often includes additional information about how authentication is handled, as well as the flow of communication. Common architectural structures are REST and SOAP. The data format of a message describes the syntactical form of the objects that are transferred. Since HTTP messages are based on ASCII-characters [24]

it is also commonly used when describing data objects. Common data formats used today are JSON, XML and plain text.

4.2 TEMPLATES

In order to separate the presentation layer from the business logic, templates are often used. A template engine compiles HTML from a description written in a template language together with a set of data, it is commonly used for rendering the views in a JavaScript application. When the data is changed the view can simply be re-rendered so that the user sees the correct version of it. A template engine is however a quite complex system. To generate HTML from a description an interpretation of the template language is required. Since these interpretations are done during runtime the performance is a crucial aspect to consider. Some frameworks have their own template engines while others use generic libraries in order to support template rendering.

4.3 DATA BINDINGS

Data bindings are used to bind a set of data to a corresponding view. One-way data bindings allow a view to be automatically re-rendered when its data is changed. Some frameworks, like Angular JS, support a two-way data bindings that also allow the data to be updated corresponding to changes in the view. This is quite a powerful technique since all logic behind a view is now managed by the framework which makes it easier for the developer to get things working. Important to note is that no logic or complexity is removed, it is just moved to the framework. However, a framework must support a wide range of HTML elements even though all of these are not being used by the application. This often affects the performance of the application and two-way data bindings can sometimes become overkill, especially for smaller applications.

4.4 TESTING A JAVASCRIPT APPLICATION

Testing comes in many forms and shapes during a project. During the development phase unit tests are commonly used to drive the development. It is used as a tool by the developers to ensure that all the functionality is kept intact. The use of a test-driven approach to develop software is becoming more and more popular. However, developing JavaScript applications with a test-driven approach can be tricky. The functionality behind a JavaScript application tends to be tightly coupled with the user interface. This is simply the case since one of the most essential purposes of a JavaScript application is to interact with the user, which of course is done via the user interface. To write unit tests that interacts with the user interface is not trivial. It requires the unit test to trigger events on DOM elements and then traverse the DOM and validate its content. If a view is even slightly changed this can break the test even if an acceptance test would still pass. Some architectures allow developers to write unit tests for what is underneath the graphical interface, making the tests easier to write and the result easier to validate. In the end unit testing will only ensure that the code is doing the right thing, not that the system is working as expected. High-level testing, such as acceptance testing, will always be needed to validate the functionality from a user perspective.

V. EXPERIMENTAL RESULTS

Below graphs shows the initial page loading time and script and style file rendering in the browser between the existing and proposed.

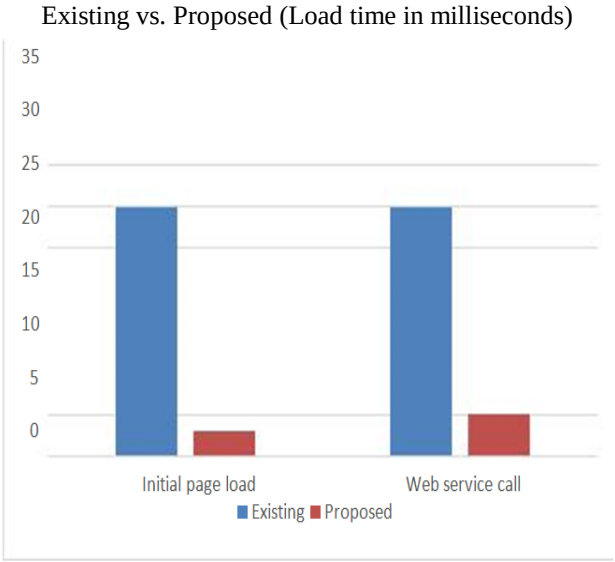

Fig.4. Page initial load time

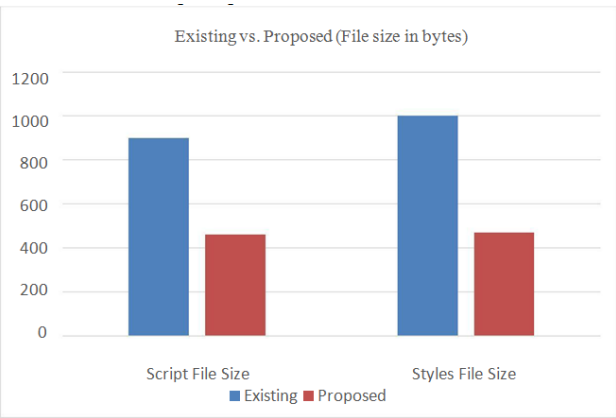

Fig 5. File size comparison

Below table shows the comparison between Existing and Proposed in terms of Initial page load time, Web service call time, Script and Style loading and file sizes.

	Existing	Proposed
Initial Page Load Time	30-35 milliseconds	3-5 milliseconds
Web Services Call Time	35-40 milliseconds	4-6 milliseconds
Scripts Loading Time	10-15 milliseconds	10-15 milliseconds
Overall Script File Size	About 1 KB	Less than 500 bytes
Styles (CSS) Loading Time	10-15 milliseconds	10-15 milliseconds
Overall Styles File Size	More than 1 KB	Less than 500 bytes

Table 1 comparison between Existing and Proposed methods

VI. CONCLUSION

As explained above SPAs are great for all websites that involve a high degree of user interactivity, like websites that are more like applications rather than traditional web pages. This can be also be used to target mobile devices in place of websites. Thus SPAs serves an alternative to native mobile applications. In case a mobile application is written as a web application there will one version to be maintained. In case if a native application is developed then there could be one version for each operating system thereby resulting in higher development and maintenance costs. Also, the user experience will be a lot better in a native application since it always will be able to perform better than a web application.

REFERENCES

1. Yanyan Lu, Haiyan Wu, Yingxue Wang , " Web application performance analysis based on comprehensive load testing ", IET International Conference on Wireless Mobile and Multimedia Networks Proceedings (ICWMMN 2006), 2006, p. 386 - 386
2. Ye Zhou, Yang Ji, " Design of rest APIS for the exposure of IMS capabilities towards Web services", IET International Conference on Communication Technology and Application (ICCTA 2011), 2011, p. 526 - 530
3. L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, P. Spoletini, " Validation of web service compositions ", IET Software, Volume 1, Issue 6, 2007 , p. 219 - 232
4. Ali Mesbah, Arie van Deursen, "Migrating Multi-page Web Applications to Single-page Ajax Interfaces" Report TUD-SERG-2006-018 2nd revision.
5. http://www.w3schools.com/angular, "Angular JS Tutorial" w3schools.com.
6. http://www.asp.net/web-api, "Web API tutorial" Microsoft asp.net site.
7. http://www.asp.net/single-page-application, "Single page application tutorial"
8. http://www.w3schools.com/bootstrap/, "Bootstrap Tutorial" w3schools.com.
9. https://en.wikipedia.org/wiki/Single-page_application, "About Single page application" Wikipedia.org.
10. http://www.w3schools.com/ajax/, "Ajax tutorial" w3schools.com.
11. https://msdn.microsoft.com/en-us/library/hh833994(v=vs.108).aspx, "Web API tutorial" Microsoft MSDN site.
12. http://blogs. msdn. com /b /martinkearn /archive /2015 /01/05 /introduction -to-rest-and-net-web-api.aspx, "Introduction to REST and .net Web API" Microsoft MSDN site.
13. https://angularjs.org/, "Angular JS Introduction" angularjs.org.
14. https://docs.angularjs.org/api, "Angular JS documentation" docs.angularjs.org.
15. Madhuri A. Jadhav Balkrishna R. Sawant, Anushree Deshmukh, "Single Page Application using AngularJS", International Journal of Computer Science and Information Technologies, Vol. 6 (3) , 2015, 2876-2879

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication