**SPA GUIDELINE**

Contents

[**1** **History** 3](#_Toc87989373)

[**2** **Introduction** 3](#_Toc87989374)

[**3** **Evaluation and SPA Advantages** 4](#_Toc87989375)

[**3.1** **Generation 1 [HTML static page]** 4](#_Toc87989376)

[**3.2** **Generation 2 [Session]** 4](#_Toc87989377)

[**3.3** **Generation 3 [Ajax]** 5](#_Toc87989378)

[**3.4** **Generation 4 [SPA]** 5](#_Toc87989379)

[**3.5** **Advantages** 6](#_Toc87989380)

[**3.6** **Disadvantages** 6](#_Toc87989381)

[**3.7** **Challenges** 6](#_Toc87989382)

[**4** **System Architecture** 7](#_Toc87989383)

[**5** **Application Architecture** 8](#_Toc87989384)

[**5.1** **Server Side** 9](#_Toc87989385)

[**5.2** **Client Side** 9](#_Toc87989386)

[**6** **Process Flow** 11](#_Toc87989387)

[**7** **Data retrieval approaches** 11](#_Toc87989388)

[**7.1** **Business Logic directly from App server** 11](#_Toc87989389)

[**7.1.1** **Advantage:** 12](#_Toc87989390)

[**7.1.2** **Disadvantage** 12](#_Toc87989391)

[**7.2** **Business Logic via Web server** 12](#_Toc87989392)

[**7.2.1** **Advantage:** 12](#_Toc87989393)

[**8** **Architecture Pattern Justification** 12](#_Toc87989394)

[**8.1** **MVC Pattern** 12](#_Toc87989395)

[**8.2** **Repository Pattern in the server side** 12](#_Toc87989396)

[**8.3** **Mapper pattern in the server side** 13](#_Toc87989397)

[**8.4** **Security** 13](#_Toc87989398)

[**8.4.1** **Authentication/ Authorization** 13](#_Toc87989399)

[**8.4.2** **Data integrity & Confidential** 13](#_Toc87989400)

[**8.4.3** **Instrumentation** 13](#_Toc87989401)

[**8.5** **DI using IOC container in the server side** 13](#_Toc87989402)

[**8.6** **MOCKING FW for Unit testing** 13](#_Toc87989403)

[**8.7** **Validation** 13](#_Toc87989404)

[**8.8** **Performance** 13](#_Toc87989405)

[**8.9** **UI responsiveness** 13](#_Toc87989406)

[**9** **Conclusion** 14](#_Toc87989407)

[**10** **Reference(s)** 14](#_Toc87989408)

# **History**

|  |  |
| --- | --- |
| **Author** | Loyola Stalin Soosai |
| **Reviewers** | Bhargava, Anubha, Gandikota, Sudhakar, Ramados, Prathap |
| **Approver** | Thandalay, Suresh S |
| Approved On |  |
| Changes | 1. Initial version created |

# **Introduction**

A **single-page application** (**SPA**), is a [web application](http://en.wikipedia.org/wiki/Web_application) or [web site](http://en.wikipedia.org/wiki/Web_site) which aims to provide a rich user experience similar to a desktop application. The entire site/pages loaded into the client-side/ browser during the initial request raised by the consumer/user, but subsequent interactions take place through AJAX data request. Routing/ Page navigation is handled by the client-side application using Typescript/JavaScript framework in other words no interaction happens to the server while moving between pages and rendering the HTML content in the specified container/place holder happens at the client-side/browser.

During the initial request from the client [Browser IE, Chrome, Firefox] to the server, the server [Web server] sends back required resources for rendering the whole website including HTML, CSS, scripts [JavaScript]. After the initial request from the client, all the presentation logic is available at the client-side using JavaScript. The client requests the server [Generally it’s an App server] to get the data asynchronously [JSON, XML] based on the demand. In this approach, the server has no role to maintain the UI state [Stateless] and navigation between pages. In some cases, the client interacts with the server to get the HTML content to render the view in the template container.

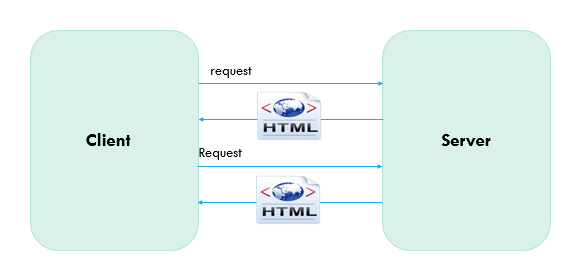
This article summarizes the various best practices and uses case architecture using SPA. This article uses Angular and .NET technology stack as examples however the principles stand out for technology-agnostic by nature.

# **Evaluation and SPA Advantages**

The SPA is the 4th Generation application, let’s look at the evaluation of the web application to understand the progress that happens over time.

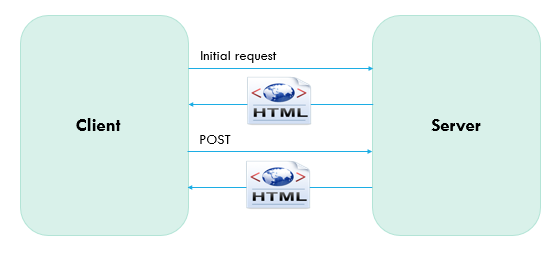
## **Generation 1 [HTML static page]**

In the first generation, the client requests the server and gets only a static HTML page that uses the DHTML approach to render the view at the client side.

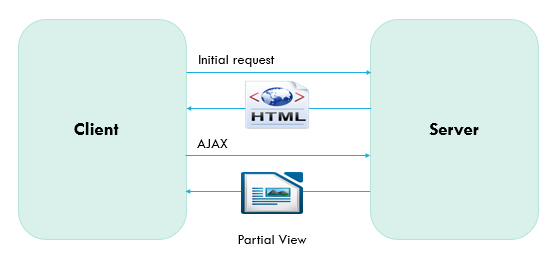


## **Generation 2 [Session]**

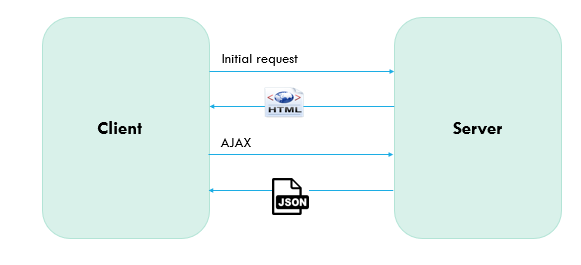
In the second generation, server sends the HTML along with some state machine mechanism [Cookies, token, etc..] to retain the previous state of the user details and every request server sends back the HTML page and reload the entire view to the client for each request.



## **Generation 3 [Ajax]**

In the third generation, the master page loads during the initial request and gets the portion/part of the view from the server and renders the partial view. In this approach the entire page won’t be reloaded, only certain portion of the view will be rendered based on the demand using AJAX requests from the client to server. On navigating between pages, the entire page reloads/redraws at the client side which includes images, JavaScript, CSS, etc...

## **Generation 4 [SPA]**

In the fourth generation, loading the page during the initial request, on subsequent request only the data/HTML will be obtained from the server and bind the value or fill the in container at the client-side. This approach brings up the user experience like the desktop application. When the user navigates between pages, the client wouldn’t make the requests to the server for retrieving the HTML/JavaScript/CSS resources.  


## **Advantages**

1. SPA approach decreases the loading time after the initial requests are made and reduce roundtrip to the server to fetch the static content
2. Most of the processing happens on the client side which prevents the number of requests to the server drastically
3. SPA makes it possible to do offline processing with the help of caching components at the client-side [local Storage, Session Storage, Indexed DB]
4. User responsive application is achieved. Thereby, all the page navigation happens only in the client-side and only the dynamic data [JSON] is required to render the view, which will minimize the network traffic

## **Disadvantages**

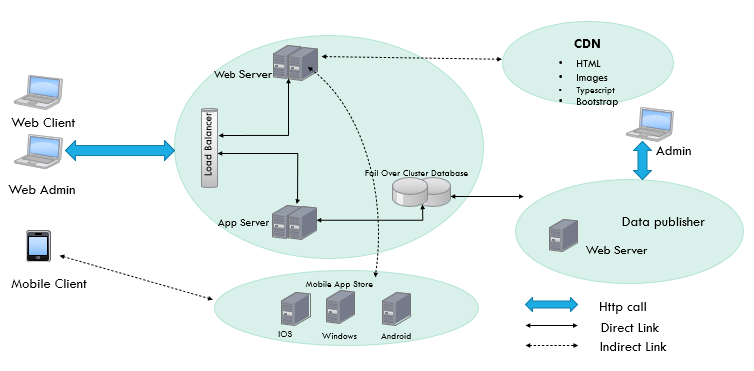
1. Duplication of code that violates the DRY (Don’t repeat yourself principle) i.e. The system must create models both on the server-side and the client-side, etc.
2. Some business logic can be in JavaScript and if the security gets compromised, system breaching might be possible
3. The client must enable JavaScript to run SPA because page navigation occurs on the client browser
4. Initial page load time would be more/higher compared to other generation applications because it must download all the HTML, JavaScript, CSS, images during the initial request. However, the modular approach will be a rescuer to improve the performance to some extent

## **Challenges**

There are some challenges by using this SPA approaches, must be considered

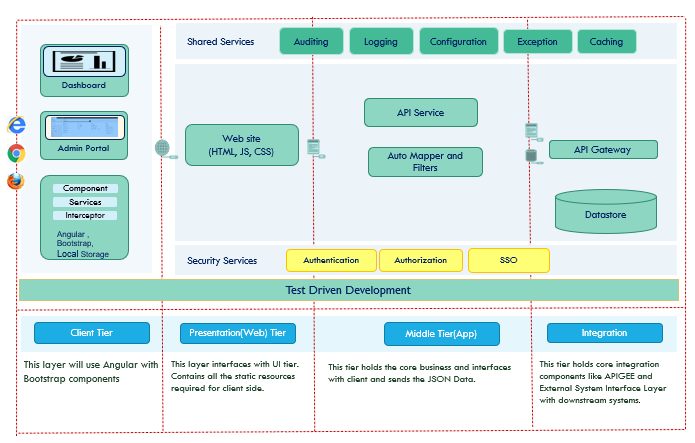
* Search Engine Optimization
* Client/Server code partitioning
* Browser History & Bookmarking
* Analytics
* Speed of initial load

# **System Architecture**



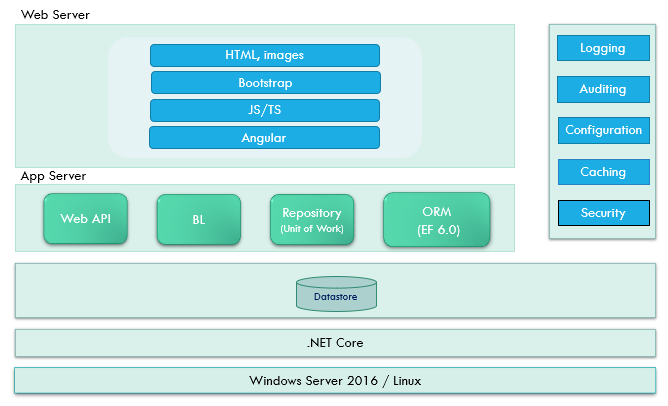
* Web Server responsible for hosting the Website
* REST API Services (App Server) facilitates for handling all the business transactions, authentication, and authorization
* Web/App Server forms clusters and expose the interface via Load balancer [Physical/ Software]
* CDN will be used to load the static content in all the edge locations to improve the performance
* Database and Web/App server must support Failover scenario for the Tier-1/ Tier-2 types of application to achieve high availability and resilience

# **Application Architecture**



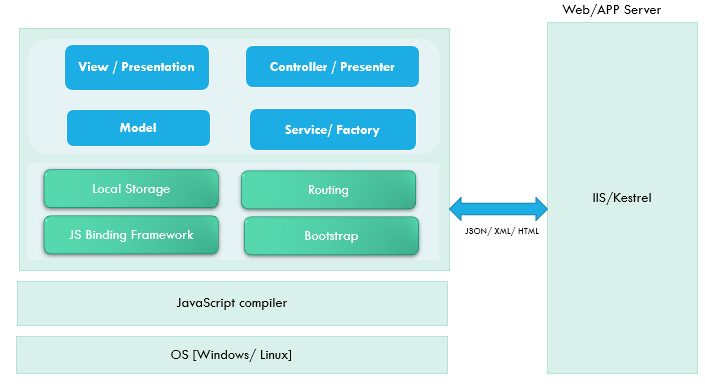
* Client tier will use Angular FW to customize user screen look and feel with a SPA design approach
* RWD design approach will be leveraged for optimal viewing experience for various channel
* The presentation tier holds the static content, this content would be rendered dynamically at the client-side using Angular FW
* WEB API Restful service acts in the middle layer
* API Gateway integration to support the homogeneous environment
* Caching the master data [CDN, Localstorge] to improve the performance
* Test-Driven development [Jasmin, Nunit] improves the quality of the deliverables
* Static code analysis [JLint, Sonarcube] improves the code quality perspective

## **Server Side**



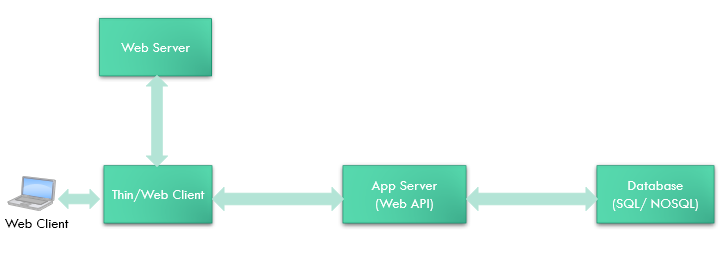
During the initial page request, the server sends the template/master page [Full View] along with all required style sheets, JavaScript, and images. After the initial page is rendered to the client-side, subsequent page navigation would request to the server only for the HTML view [Partial View] in case if the view is not cached on the client-side. WEB API Restful service handles any data retrieval or CURD operation.

## **Client Side**



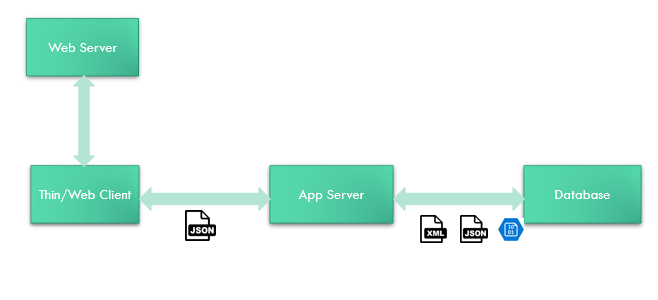
On the client-side also recommended using MV\* pattern. Whenever the user navigates between page, the client-side router fetches the files from the client-side and render the view at the specified placeholder. To render the data, the client should make the HTTP call to WEB API Service and bind the data dynamically at the client-side.

# **Process Flow**



The web/thin client should call the web server only to fetch the HTML content and should call the WEP API service for fetching the data.

# **Data retrieval approaches**



## **Business Logic directly from App server**

### **Advantage:**

* BL can be shared between applications

### **Disadvantage**

* If the App server is shared with the multiple clients, the BL would be shared between all the clients if a security compromise

## **Business Logic via Web server**

### **Advantage:**

* One more level of abstraction for maintaining the granular level of control
* BL will be specific to the application, so other application can’t use the BL

The recommendation/ best practice to keep the BL resides on the App server and specific to the application BL logic resides only on the Web server. The client should interact with the BL via a web server and the web server orchestrates the data retrieval process.

# **Architecture Pattern Justification**

## **MVC Pattern**

Since the View is generated on the runtime, the controller should decide to generate the View-based on each request, the recommendation to use MVC architecture pattern for the Presentation layer. Separation of concern to make the system more testable each component is another driving parameter to choose the MVC on the client-side.

## **Repository Pattern in the server side**

There must be a clear separation between the data access and the business layer, hence the recommendation to the Repository pattern. In case of any changes in the Middle tier, no need to change the WEP API service since the repository behaves as a middleman to bridge between the WEB API service and data service. The repository has exposed an interface and is used by the WEP API service. The Repository pattern decouples the data service and WEP API service.

## **Mapper pattern in the server side**

The business model should not expose to the client-side to achieve the encapsulation and security concerns. The View model sends to the client-side through the WEP API service. The client will get to know the model only what it needs for rendering the data.

## **Security**

### **Authentication/ Authorization**

Based on the user authentication, we provided the role-based authorization. Based on the user’s role the menu and other access will be determined in the system.

### **Data integrity & Confidential**

Most secured details are made by the transport layer security by enabling the HTTPS.

### **Instrumentation**

Logging and Auditing are in place to make the system available for diagnosis purposes. The logging component has a different log level [Off, Warning, Error, Information] for different diagnosis purposes.

## **DI using IOC container in the server side**

Dependency Injection facilitates validating the individual class functionality completely bypassing the mock dependent object, thereby each class should be testable on its own.

## **MOCKING FW for Unit testing**

* NMOQ [Server side]
* Jasmine [JavaScript and Angular]

## **Validation**

* Client-side (using Angular directives)
* Server-side
* Business Rules in the Entity Framework rather than using in the trigger in the database

## **Performance**

* Initialize data once per view and manipulate the data on the client-side using JavaScript
* Caching the master data in the server-side
* Caching the data on the client-side using LocalStorge, SessionStorge, amplify [JQuery]
* Bundling the JavaScript, CSS, and image sprite
* JavaScript minification
* CSS on top and java scrip script on the bottom

## **UI responsiveness**

Bootstrapwe have usedfor the purpose that the website automatically resizes based on the different device screen resolution.

# **Conclusion**

The main advantage of SPA makes our website user experience like a desktop application. The system architecture designed by considering in mind the code should be maintainable, testable, readable, highly cohesive, and loosely coupled. TDD application environment plays an important role to make sure the quality of the deliverable is achieved.

# **Reference(s)**

* <http://msdn.microsoft.com/en-us/magazine/dn463786.aspx>
* <https://angular.io/>
* <http://www.w3schools.com/html/html5_webstorage.asp>
* <http://getbootstrap.com/getting-started/>
* <https://msdn.microsoft.com/en-us/library/ff649690.aspx>
* <https://msdn.microsoft.com/en-us/library/ff649643.aspx>