Hemant Elhence: The topic today is building single page applications; know the ecosystem. Single page applications is a very popular way now to build highly immersive web applications which provide a desktop like user experience, and understanding this entire ecosystem and fully getting some sort of a mental framework. How to pick alternative technologies that go into building single page application is what we'll cover today. I'm delighted to have our own Rohit Ghatol Ghatol our director of engineering to present this topic. Let me do a quick introduction to Rohit Ghatol and then I'll turn it over to him.

Before I introduce Rohit Ghatol let me also just kind of quickly run over … run through the mechanics, the usual mechanics of this webinar. We will have roughly 45 minutes or so of presentation that Rohit Ghatol will go over on this topic of single page applications. Along the way ... I'll act as a moderator, this is obviously a very technical topic; I'll pick and choose some questions to be covered along the way. You guys feel free to type in your questions in the Q&A panel on … go to webinar session. Then at the end we'll still leave some time to take some broader general questions that you guys may have. After the session today we will … we're recording this of course as we do every month but we also make the document available to the attendees and a recorded version will be available for you guys to watch again if you want to go over it or people who could not attend today would get to watch it in the recorded version.

With that let me introduce Rohit Ghatol. Rohit Ghatol is Director of Engineering at Synerzip. He's been with for a number of years and since 2006 he's been the practitioner in the single page application space, so he's really seen this space evolve and has been living and breathing this space. He's … overall he's a technology evangelist and been a Rohit Ghatol Ghatol at various conferences and he's written a book on "Beginning PhoneGap" and in general he tends to dabble in and maintain his cutting edge expertise in areas … all areas related to bleeding edge technologies whether it is mobile or HTML5 or cross platform mobile and so on including the traditional enterprise Java J2EE type technology stack also. Currently he's more enthusiastic about Node.js and Hadoop but I'll let him talk more about him when he … when I turn it over to him.

With that over to you Rohit Ghatol.

Rohit Ghatol: Thanks Hemant Elhence. Like Hemant Elhence just said I'm a technology evangelist. I love technology and like always to look new technology. That's enough about me right now; let's just try to jump on the topic. Before we begin I just want to share how we came up with this topic about SPA Ecosystem. Here in last year we were engaging with a few of our customers and we proposed to them that they should build a new UI, using single page application technology. It was kind of difficult for me to explain to them what really single page application is because it has too many pieces into it. Last year I wrote a blog on SPA Ecosystem. It tended to be a very long blog and this presentation will cover or will summarize some aspects of that blog. After this presentation, if you guys want to dig deeper into each aspect of that or you do not understanding what a concept means, just feel free to visit this blog and it will introduce you to different angles of each of the framework areas.

With that let's begin today's topic of SPA. Like Hemant Elhence said, what is SPA? SPA are the new browser based applications which are fast replacing the fact lines. Imagine email replacing Outlook or imagine Adobe Photoshop coming inside your browser. We're talking about really rich applications which are now appearing in browser and there is no need to install them. All these applications are very rich and immersive. They are able to save the context for the user. You can even bookmark APIs … bookmark URLs on these applications and go back to your particular portion of your application. Also some of these applications can pretty much go offline.

The best part about SPA is that it opens doors for you so that you can build hybrid mobile applications, so while you plan to build your desktop application as an SPA, with a little bit more effort you can always extend it to create a hybrid mobile application. Now most popular example of SPA which there is of course Gmail, Google Hangouts is a very good example of SPA and Facebook and Twitter are almost SPA. Nowadays every new application which is coming up tries to be an SPA if the context is right.

With that we have covered what an SPA is and then we should go to the next topic of why should we learn about the SPA Ecosystem. Now let me give you an example of what typically happens when I talk to people who do not know about SPA and they kind of have a conception … a concept of SPA in their mind which is not circular. If I'm actually talking to a developer and give him or her about SPA and ask him the question, what do you know about SPA? Typically around last year I used to get this answer that I know jQuery and that was my reaction, like no not again. The reason being, yes jQuery is a tiny part of the SPA but it's a very small portion of a very big picture and you'll see that going forward. If somebody claims they know jQuery don't believe that they know SPA because SPA is a much, much bigger thing.

Now having said that, what we should focus on in this webinar is listing popular frameworks on SPA and see where they actually get them, but before we do that we have to go to the classifications. We have to put these frameworks in different areas of SPA and see what they fulfill. This … the main slide for this entire webinar and we'll keep on coming back to this slide as we go ahead.

Now instead of me explaining each portion right now we will just go purposefully and we'll pick one box at one time and go ahead. We'll start at the foundation layer then we will do the presentation layer then we'll touch the large scale application architecture layer followed by the build and deployment layer.

Given that this topic is very large and we can talk about this topic all year round. We will just be browsing over these topics at 30,000 feet. We're going to just go over each of these topics, understand what those areas are, what are the key challenges, which frameworks are a good fit and what are the choices … what are the key aspects which you need to consider when choosing these frameworks. We are not going to explain or teach you about each of these areas nor are we going to dig into one of the frameworks at a much deeper level. We're just going to take a glance to that and this seminar will be opening doors to a series of webinars on the SPA side, so just keep that in mind while you are listening to this presentation.

Okay so the first area we'll look at is DOM Manipulation. This is the most basic area of SPA; typically that's what jQuery does. It takes your DOM manipulates it, so basically anything that's in the view you're supposed to use libraries like jQuery or XUI, or things like that. Also even handlings like button clicks and everything are handled in this area. As this is a very trivial area we're not going to dig much deeper.

Let's go into frameworks. jQuery being the most popular framework, another one not jQuery has come 2x which is a lighter version of jQuery removing support for IE 6, 7, 8. Zepto UI is one of the frameworks which came as a lightweight replacement for jQuery. Almost all the APIs of Zepto UI match with jQuery, so it's a good replacement for jQuery if you think your application is being too heavy but given the internet speeds these days that's a non-issue.

Other players are coming up like Snack and $DOM, so that's mostly about the DOM manipulation. It's a very basic area of SPA.

Going forward the second area of SPA is … in the foundation layer is while you manipulate the UI, you need data to manipulate the UI, so simply said Ajax …

Hemant Elhence: Hey Rohit Ghatol let me interrupt and take one question here. You have … could you … if you can go back to the previous page you have introduced this organizing framework for various alternative SPA or various components in the SPA technology landscape. Is this kind of an industry standard where to think about this office, even the bigger one, the presentation layer, the foundation layer and so on or is this something that you have put together based on your experience in how these applications come together?

Rohit Ghatol: That's a very good question Hemant Elhence. Over the last two, three years we have been toying with single page application. In fact since 2006 we have been building single page applications, initially we used technologies like Google toolkit and then we started moving towards JavaScript frameworks. Our first experience was with Sencha and with Backbone and then we moved to Knockout and now we're trying Angular. While we're building things for different clients slowly this pattern emerged out, this our own view of how we've categorized single page applications. In Synerzip we use it as a template for building SPAs.

Hemant Elhence: Okay. All right, thanks.

Rohit Ghatol: Going to the next layer, data access layer Ajax has been around, it's a no-brainer, you call up the APIs and get the data back. The next thing which is there is when you have a richer dashboard output with light charts or you have chart like applications you go for a full duplex communication like web sockets. HTML5 is even using a new concept called server sent events which is many computers using the old technologies like Comet, Reverse Ajax or Hanging GET. These are overall different areas of data access.

Again in the APS jQuery like Ajax there are other APIs which are there. The thing to note is any framework like Sencha, Backbone or Angular will provide their own high level APIs to call REST APIs and fetch data.

When it comes to web sockets and server send events really you don't require a library because these are coming as HTML5 APIs, so most of the modern browsers support them out of the box.

That's what it is about these two low … low level building blocks, DOM Manipulation for UI, data access for fetchable data. When you go beyond that, when you go beyond jQuery which I'll say, you have to think about building your applications which will have certain aspects like models, views, controllers, routers which you'll see these are different components of the applications. Now once you have different components of the applications you have to think in terms of modules. You've got different classes and objects, you're going to put these different classes in different JavaScript files they will have certain dependencies across themselves, so the view will tend to depend on the model, the controller will tend to depend on the view something of those sort of dependencies will come into picture.

Also if you have a bigger application you will have number of these JavaScript files, so you don't want to load all of them up early on, so you might want to do asynchronously loading and you might want to do some kind of a lazy loading. Of course when you're working with JavaScript you have to take care of namespace conflicts. If different classes in different libraries are having similar names then you can have a namespace conflict.

Modules are a good way of taking care of that and typically in that we use a term caller as AMD. Now AMD stands for Asynchronous Module Definition. This is one of the very basic building blocks of single page applications. The confusion is you really have to use AMD, I don't see an alternative where you can't really not be using AMD to build your application . AMD a simple quality and certain speed to your application, so AMD will provide you with reusable modules, it will provide you dependency across your modules, avoid namespace conflicts and so on.

Now if your application looks like this like we talked about, your application depends on controller which depends on view or store and ultimately it depends on model. If you're not using something like AMD you will end up writing your code like this or you'll end up putting all the JavaScript files in the HTML page itself. Now that's tricky. Imagine if you happen to have 30, 40 or 50 JavaScript files you had to maintain a certain sequence. You go wrong in a sequence your application doesn't load and that's not really the right way of holding your SPA.

If you move to SPA typically you will move to library like require.js where you will say, hey go ahead and build my first module and that's what you're saying over here require.js, go and load the data main and .js. What require.js really does, it goes through each of these JavaScript files which are nothing but require.js module containing your classes and each of these files start with a header like this. It starts with require controller, so it knows I have to load the controller. The controller says I need the view and the store so that it loads up that and ultimately the store says I need the module, so that's loaded up.

The basic conclusion over here is you have to use AMD, there is no alterative and then there are fewer options which are available when you use AMD. For libraries like Backbone you can go with require.js. Similarly for angular and Knockout you can go with require.js. For much bigger frameworks like Sencha which tend to be their own ecosystem their own version of AMD built inside that. Well there's another library for AMD called common.js but that's commonly for node.js world; that's meant for non-browser worlds.

Finally there is a concept like UMD coming to picture which says that you can have a module which can work on require.js or common.js or straightway in the page.

That was a lot of foundation there. Any questions over here Hemant Elhence?

Hemant Elhence: No, I have not seen any questions. There's one question, let me bring it up. You mentioned the notion of heavyweight application is on the way. Are you going to describe what you mean by heavyweight?

Rohit Ghatol: Heavyweight? I'm not too sure if I used the word. I said … maybe it's a much more bigger application. Sencha tends to be more heavyweight than applications like Backbone because it has a lot of widget coordinates like that. That's what I would talk … refer to as heavyweight if I had to refer to it as heavyweight.

Hemant Elhence: Okay.

Rohit Ghatol: Let us quickly go to the next section and this is the most important section. People typically want to talk about these sections which is choice between frameworks for the MV* layer. Before we head to the MV* let's really understand what really MV* Frameworks mean. Now in the presentation layer you have to worry about your modules, your views, maybe your controllers and things like that. Typically in the UI design pattern world there are these three design patterns MVC, MVVM, MVP. Typically JavaScript both uses MVC and MVVM but generally MV* series that if any library approximates to any of these design pattern we call them MV*. Backbone is MVC but really the controller is embedded inside the view, so it's not pure MVC.

Let's focus on the MV* Frameworks over here. I'm going to take a tiny set of the MV* Framework. This being Backbone which has been there for quite some time. Angular.js, a new one from Google, Knockout a favorite in the Microsoft world. Angular.js again a very known framework.

When you look at the MV* Frameworks there's a categorization happening over there. There are two kinds of frameworks over here. One is an opinionated framework. In this it will be Angular.js will be there Ember.js will be there. Second and the frameworks they're called as un-opinionated are more flexible frameworks in which case Backbone.js will come.

Really your first choice when you're working with MV* Frameworks is where do you want go. Do you want to go the very basic minimalistic framework like Backbone.js which is very highly flexible but you have to do more varying initially to get it started with?

The second thing is an opinionated framework in which they will define pretty much the entire programming construct that the views have to be defined in HTML. They're predefined conventions but then again it's faster for a development cycle. There's a learning curve but then you don't have to worry about that picture. This is a very basic choice which you have you make while you're going to work with the MV* Frameworks.

Let's go over these MV* Frameworks one by one. Backbone.js, one of the first popular MV* Frameworks again very lightweight, un-opinionated. It works with numerous other frameworks. Backbone doesn't define that you have to use a specific libraries. You can pretty much use other libraries as you please. It's been there for a long time so it's got a good ecosystem. You've got Marionette.js which sits on top of backbone and gives you a framework for large scale applications. Chaplin gives you an architecture construct, Thorax gives you an opinionated version of Backbone and Exoskeleton which is very much a lightweight version of the same Backbone, so you can replace Backbone.js with Exoskeleton. It's a very popular choice but like its un-opinionated you have to … there's a learning curve and it takes longer to code for the developer.

Angular.js, is a newer framework and very popular. It is backed by Google and is a good company to support. It also has cool concepts like 2 way Data Binding. Has a very fast development cycle. This is our favorite and it's very, very fast and we are planning to use it for our next SPA.

Next one is Knockout.js and this one is the favorite in the Microsoft world and the reason being it's an MV VM design which the same design used in Silverlight. It also supports 2 way Data Binding, has a much faster development cycle. Generally when I use Knockout in a few on my projects I realized that it's a very good framework when you're building small sized, mid-size applications which is like mid-sized level complexity of applications. In that scenario Knockout tends to be an ideal framework according to me. It does lack basic things like router for switching between different UI schemes and things like that but otherwise it's a lovely framework to work with for small to mid-sized apps and that's my own personal opinion.

Backbone seems to be a lot more cryptic in terms you have to code in the view entirely by yourself, it's powerful but I don't like the fact that you have to write jQuery APIs inside the views.

This is how I'll describe the MV* Frameworks which is there and this is a basic choice which you have make when you're doing that.

Hemant Elhence:

Before you go off from this section can you shed some light on this topic that you refer to midsize in complexity versus small. For someone who is building a new application how do they know wither they are midsize or small? What are the characteristics?

Rohit Ghatol: It's to do with how many components do you have in your application versus how much is the development effort for you, that's what I would describe it. Let's say if Gmail is considered as midsized then Adobe Photoshop like application or a full-blown Microsoft office application in a browser I would consider it as large-scale and I would consider Gmail as mid-scale because it only has contacts and emails and nothing more. Think about that analogy when you're looking at mid-sized versus large sized applications.

Hemant Elhence: Okay and can you give me an example of what you would consider small then?

Rohit Ghatol: Small would be something like Twitter. Something very, very basic basically. Like chatting applications or something very, very basic. In fact small sized … let's stick to mid-sized and large sized because small sized is very difficult to describe. I really haven't brought any small sized applications, so let's just cover two sizes.

Hemant Elhence: Okay. All right, sounds good. Thanks.

Rohit Ghatol: If you're working with the MVC, if you want to choose an MV* Framework this is a good start. You can do to MVC.com where almost all the frameworks are mentioned and therefore examples for all the frameworks will fall in the MV* world. If you want to choose which one to go with look at the code examples over here and then you can make your choice. All right, so that was in brief the MV*Framework.

Now typically an MV* Framework also includes templating engine which allows you to take out the view logic out of the view and it also has a routing and history library embedded into that. In case it doesn't have it, you need to understand what really templates are, so for example Knockout doesn't … Backbone doesn't have its own templating library. Backbone says that you can go and use any templating library. What templates really are is new logic put into HTML snippets which you can use. Think about JSPs versus tablets. JSP's when you use templates those are templates. Think of ASPs, those are templates. Now templates also come in the JavaScript world and the whole idea is when you're building an application like a shopping cart application and you're searching for a camera, you don't find it in a section which is conditional which shows up only when no sections are found and so you can have multiple results repeating over here.

All this logic typically goes into your template scripts and which are there and it makes your code very, very clean. There are numerous template engines out there, Underscore being one my favorite. Handlebar being one which is used with Amber.js a lot. Moustache is a much more good-looking template engine whereas if you work with strange frameworks like Angular and Sencha they tend to have their own template engine. Sencha has its own X template engine built in.

You just need to be aware of this when you're using templates. Sometimes template engines are provided with the MV* framework, sometimes you have to use your own.

The same thing was with routers actually. When you're working with routers, understand the concept of router over here. A single page application is typically single page, so that's going to be abc.com. In this case if I click on the tab one it needs to somehow register with itself that it's on tab one, so it puts a hashtag one at the end. Now if I move tab 2 that URL changes but it changes after the hash, so it's all client site changes and if I bookmark this URL and if I come back to that bookmark again the single page application should look at the URL and know that, hey I'm supposed to be on that, so it actually opened up to itself. More over if I'm navigating to the application and if I happen to click the back button it should behave more or less like a normal application and then go from tab 2 to tab 1.

Routers and history basically gives you bookmarkable URLs even for single page application and support for back buttons and forward buttons which are there. Routers and history, you have to make sure that you have the support for them. Without that single page application it won't really feel like normal page applications. Here are a few examples, so again like I said the bigger MV* Frameworks will have support for it like Angular has support, Backbone has support, Sencha has support. Knockout doesn't have support for routers. In case of routers you'd have to pair it, in case of Knockout you will have to pair it up with router or Sammy or some of these frameworks.

That part, so when we combine MV* with template and routers and history that is … these two parts are very close together and typically most of the MV* frameworks provide for almost all three of them. That's about MV*.

Now the next topic which we're going to cover is you've built up your single page application with views, models, controllers, back end web services. Now you want your UI to feel like a modern UI. In that case you go with responsive UI design frameworks. The first thing which this framework provides you is the … allowing your same application to be useable on different screen sizes, so an application on desktop will appear with all the menus on the top, when you move to mobile all the menus are shifted to a dropdown over here. Similarly you can have … on desktop you can afford to have three of these boxes side to side in case of mobile they'll be switched to one view to another.

Typically these applications use a concept called as a 12 column grid where it says that in case of desktop because of large space I'm going to divide the desktop into 12 columns, equal columns and I'm going to say the first box lies in the first four columns, the middle box lies in the middle four columns and the last one lies in the last four columns. While this is true for the desktop role I want to make an exception when I'm going on a mobile world. In a mobile world because all these 12 columns squeeze in very tightly together and I want to say in the mobile world I want to make the first box, not take four columns but taking that twelve columns and because there will be 12 columns the next box automatically just goes below.

That's pretty much the concept of responsive UI design and there are two main frameworks which are available over here which is Twitter BootStrap and Zurb Foundation. We'll spend some time comparing them in the next slide. The name itself suggests what is what, so bootstrap is to bootstrap your application. It provides more of everything, it will provide you with a good looking theme, it will provide you with good looking layouts, it will provide you a rich set of UI elements including items and that's what bootstrap is all about. Pretty much if you start a bootstrap you don't require a UI designer. If you go with Zurb Foundation what they say is that it's a foundation there itself … it will be minimalist, they will have limited UI elements.

That's the choice which you have to make, whether you want more or you want minimalistic. Both of them are mobile first, bootstrap uses pixels for its sizing and Zurb foundation uses EM. The good thing is both of them have a column grid design. Pretty much your UI must have a responsive UI design to work with, that's a given.

We'll quickly just browse through the next few topics 3D and 2D and style sheet language because if you want to …

Hemant Elhence: First of all let's take one question. The question is do you use bootstrap with Angular?

Rohit Ghatol: Well bootstrap with Angular, I haven't really done that. Till now I have used bootstrap with Backbone and Knockout. With Angular I don't think that there should be any issue because typically Angular only provides you things like … some mock ups. In my opinion you can use bootstrap with Angular because bootstrap only provides you structure and CSS so there shouldn't be any problem as far as my current knowledge goes.

Hemant Elhence: Okay. All right, thanks.

Rohit Ghatol: If you're going for 3D and 2D, 2D typically is done using Canvas/SVG and for 3D you have WebGL support. The good thing is even I-10 came onboard with using WebGL. Now all the latest browser versions support WebGL.

In case of 2D, I'd like to categorize this into two worlds. One world is about quick and easy; they have clean charts. Where you know if you want a pie charts or a bar chart, an area chart, some other pretty fine charts. In this case just use libraries like High Charts, Raphael, Flot.js and get the job done.

The second case is of a full blown battle tank like D3.js which is a 2D visualizing framework. It doesn't have three different concepts like line charts and bar charts alone, but what it provides you is a mechanical framework to watch you draw everything. So if you think your needs are immense go with D3JS. If you think you have typical visualization needs like graphs and charts go with the easier frameworks on the left.

Moreover for 3D world WebGL, coding with WebGL directly is not an option, it's very cryptic, very low level, so you have to go with frameworks on the left-hand side which is like Three.js or Babylon. They will give you constructs like shapes, groups and lighting effects, material and so many things.

We'll quickly cover the next topic style sheet languages. We have come a long way from CSS 2.1 to CSS 3.0 and we've got cool effects like gradients coming up, rounded corners, animations, transitions which makes the application look very, very nice and fluid. Even when we come to CSS 3.0 there are some things which are missing in the CSS language itself. One is the ability to group all your similar CSS in a hierarchal structure. I would really want to put all those things in a hierarchical structure so I know that they all belong to one particular group, that's not possible right now in CSS.

Next thing is I cannot use variables. Imagine if you're creating teams and while creating teams if I could declare it with labels for eight different colors and then I use these variables in my team saying that the header is a darker shade of the first variable, the footer is a lighter shade of the second variable then I can really create different teams by just changing those eight variables. Well today if you work with CSS you will have to do it all together.

The last thing is Mixins which is like reusable code snippets. Whenever you code in CSS you end up using hyphen webkit, hyphen transition hyphen, mostly the hyphen transition, hyphen opera, hyphen transition and so on. It's very, very time consuming and you would want to create these Mixins once for the entire application and let your developers use them.

Two popular frameworks to allow these things are SASS and Less and these are the leading frameworks and when it comes to a choice of picking both of them, well they're almost the same, so it's your personal choice as to what to use.

That covers our topic of foundation layer and presentation layer and we are running a little bit slow, we will have to go and look at the large scale application architecture because that's the important topic to cover. I don't mind missing the build and deployments but the next topic is very important. Before we go there I just want to pause for a moment and invite any questions.

Hemant Elhence: The only question which I've kept in my pending box here and I'll reserve it for the end because it's kind of a general question, so I think we should keep going and if there's extra time and I'll take the question at the end.

Rohit Ghatol: Sure. In case of large scale application we have to first define what a large scale application is and let me reroute two quotes from Andy Osmani who's a Google engineer working on the HTML5 world. He's the author of Patterns for Large-Scale JavaScript Application Architecture. What Andy Osmani says is to … and his two quotes is, in my view large scale JavaScript apps are non-trivial applications requiring significant developer effort to maintain where most heavy lifting of data manipulation and display falls to the browser. That's how to define large scale application, a lot of components, visual components, a lot of data transition happening and a lot of effort for a developer. He also goes and says is if working on a significantly large JavaScript application, remember to dedicate sufficient time to planning the underlying architecture that makes the most sense. It's often more complex than you may initially imagine.

The whole point over here is you have to focus on architecture a lot when you're building your large scale applications and let's use some names over here. I'll use the name decoupled components or I'll use my favorite name mini applications. For building a large scale application your application has to be divided into components which are decoupled, so one component can live without the other components. Even if one component crashes it should not matter to the other components. If you think your application as a grouping of mini applications which can live by themselves, they are made to communicate with each other to function but they can live by themselves and these components would contain various models, they would contain some various views; these views will be nested in nature, so one view can contain another view.

The most important aspect is they should be proper cleaner. For a large-scale application in the same screen areas, your components will be born and they will end. They will be loaded by the browser; they'll be taken away from the browser, so a proper clean up mechanism so there are no memory leaks and there are no memory issues over there, so it's a very important aspect.

Next thing is as this mini application transitions from one to another one you need to have multiple routers. You really need to have a good architecture for having routers. Third thing is your new applications or components will be communicating to each other in the manner in which they are not aware of each other. A classic example is the Event Bus. One component puts a request … puts an event on an event bus, the second one just picks it up. If possible try to abstract out user of low line libraries, so don't just tie in or marry applications like … frameworks like jQuery or Zepto or any particular router library on any particular template. You might want to change them in the future.

One example of this framework is an upcoming framework called Aura. This is again a framework from Andy Osmani. What he says in that is that there is an Aura Core which takes care of the lifecycle of components. In this case I mentioned the components as a widget in the yellow box, so what he says is the components of widgets, they live in the sandbox. If they want to make any changes to the DOM, they will do it through the sandbox. If they want to communicate with the server they will do it by the sandbox. If they want to communicate with each of them they will do it by the sandbox.

The sandboxes are extended using extensions. Look at this example over here, let's widget 1 wants to find an element on the page, so it won't be using jQuery directly. What it will do is use sandbox. In that case extension DOM extension which we write ourselves will come into the picture and will basically add an extension to the sandbox saying sandbox.DOM. Typically that DOM extension provides you an interface layer, an abstraction layer and jQuery comes in and provides implementation, so tomorrow you can just switch jQuery with Zepto UI and so on.

Classic examples for widget we'll be calling sandbox.DOM.find to look up some elements sandbox.ajax.ajax. Similarly for routers and MVC frameworks you can add extensions. You can pretty much change the user frameworks.

When these widgets want to communicate with each other again as I mentioned they're going to use sandbox to emit messages and they're going to use sandbox to receive messages. That's pretty much the architecture of large scale application over here. The frameworks which I know about are typically Aura JS. Since I was working on Backbone a while back I have read briefly about MarionetteJS which also kind of qualifies for that. There may be other frameworks but the one which is our favorite is Aura JS because it's framework agnostic with the use of extensions.

That was about large scale application architecture. Any questions over there Hemant Elhence?

Hemant Elhence: No, no you can keep going. No, nothing from the section.

Rohit Ghatol: Okay, we'll go over the building deployment tools and you'll quickly browse to that because we're not going to go deeper into these tools, we just want to show what these tools are, what they briefly do and what are the options over there. A popular tool which is coming for building deployments is Yeoman. You want to have conventions while creating your application, while creating your views, models and controllers. The recommendation is to use a tool called as Yo which is a scaffolding tool. You won't typically have a lot of generators for Backbone, for Angular, for Amber I guess. You can just use Yo to write the initial classes and then make changes in those things. Grunt is a tool which is used for automation, so anything required to build which is in case of JavaScript undefined classes, JavaScript files, concatenating the JavaScript files or when you find the JavaScript files and all these things you can use Grunt.

It's a generic tool so you can do pretty much anything with that. The last one is Bower. So these three together are called Yeoman. So Bower gives you dependency management. Let's see if an application needs to use 1.7 version of jQuery 2.2 version of Knockout or 1.4 version. So you just mentioned that I want to use these versions and Bower will go and fetch all those frameworks from the internet and take care of any transit dependencies which is any other framework that is required will also be downloaded.

So Yeoman is very, very powerful but Grunt tends to be ugly to use. So these are more generic tools and powerful and generic that also means that they can tend to be cumbersome.

Another option for that is Brunch which is basically an extremely better assembler. It's got single focus and it tends to basically build your application for production and it does all the things which are mentioned over here which is JavaScript files prepare them for production similarly for CSS and other files. So an option really is whether you use Yeoman or Brunch, it's very simple if you want more powerful go for Yeoman whether preferred for Grunt it can be in other times, Brunch on other hand is very simple focus, does just one thing. That's all about the build and the deployment tools.

We're going to cover the mobile packaging part because that's the part which is … which comes in as a side effect of building single page applications. You get an opportunity to take your single page applications and extend them to create hybrid mobile applications. So here is what happens to these mobile packaging tools. You take tools like PhoneGap or if you take tools like Trigger.io what they really do is they have a native application planning in which there is an embedded browser and the browser runs a local copy of SPA. So think about an android application opening up that has a browser inside itself and our entire SPA application is right available there in that same application itself so there is no loading of SPA from the internet. The SPA will just communicate with the server to load the data and that's it.

Now what frameworks like PhoneGap and Trigger.io do furthermore is they provide JavaScript APIs if your SPA can consume. So they will provide APIs for let's say the camera, the GPS, contacts, compass and so on so on. So you can create the SPA to detect that okay I'm a hybrid mobile application so for example if I'm on … if I'm an Android application on Android I would get option to directly open the camera and take a picture and upload the picture to Discover so that capability comes into picture.

All these packaging tools they give you options for multiple platforms and if we can look at the first of them, first one being Apache PhoneGap aka Cordova. So PhoneGap used to be owned by Adobe. It can basically … donated to Apache now it's called Cordova. Cordova supports the maximum number of platforms. The last I checked was it supports more than a dozen platforms so obviously it supports Apple iOS, Android, Windows phones and everything. It also has a plugin architecture by which what I mean is tomorrow if you want to add more functionality to your SPA application like you want to do the barcode center you can write your own plugin like you have a camera plugin over here, you can write barcode scanner plugin and just plug it in and SPA kind of assumes that to check for barcodes on some products and tell you hey this store doesn't have the lowest cost, the store across the street has the lowest cost. So you can do all those kind of things using … using the PhoneGap's plugin application.

Also what PhoneGap provides is PhoneGap Build which is the build service in the cloud. So you kind of submit your code to the cloud and it will give you the iOS binary, the Android binary, the Windows 7 binary, and all these binaries so it makes your life very easy when building hybrid mobile applications.

Next in line is Trigger.io which is kind of a newer motion of a cross platform mobile application framework. They only support two platforms iOS and Android but their claim is that they are better than PhoneGap and Cordova in terms of their native U.S. screens so they can give native U.S. screens an application, they tend to be faster in their execution and they have preliminary integration with Facebook and others. They also have concept callers modules like PhoneGap has plugins but what they say is that these modules what they are doing is they are creating a market space around these modules so you can just use readymade modules in your application.

Well, that's about the mobile packaging application. We need to quickly reach a conclusion which is a couple of slides and then maybe open up for a few questions. So the conclusion is when you're using SPA you can classify the frameworks into two categories. One is when the framework is a large framework where it caters to more than one area of SPA. The second thing is the framework only caters to one single area of SPA.

This is an example when a framework only caters to that particular area of SPA. For example BootStrap only gives you Responsive UI, Router.Js only gives you Routers and History. The second example is a framework like Sencha which is a whole ecosystem in itself which will provide you with various options over here. So it will … with Sencha you can even handle packaging which I forgot to mention. Sencha is also … it's on mobile packaging tools these days. That's a choice. Smaller … a group of smaller frameworks versus a bigger framework.

The second choice is going for an Opinionated library or going for an Un-Opinionated data route library. Typically the larger libraries are Opinionated because they give their own way of doing things across the ecosystem so the choice you have to make is really whether you use a larger framework like Sencha or a group of smaller frameworks together which could be Un-Opinionated versus Opinionated. So that choice typically depends on what your product needs like what is the requirement for your product whether your product is a large scale product, whether your product requires mobile version, whether your product requires Responsive UI or it requires charts and graphs and also it depends on your architecture goal whether you are simply interested in development or you're trying to build a platform and you're very much keen on controlling the architecture.

So with that we had the sprint of single page application ecosystem and that's how I conclude the topic, we had to go very, very fast because a lot of topics to be covered at 30,000 feet. Now I'll open up for questioning.

Hemant Elhence: Thanks Rohit Ghatol. So there are a number of questions. So let me encourage people to add any more questions in the Q&A panel and audience while you're doing Rohit Ghatol can you just make a quick introduction to Synerzip for those who are new to this webinar series that we do once a month.

So just go on to the next page. So I'll go through a few highlights of Synerzip in a nutshell. We are a software development partner for typically small to mid-sized venture backed cutting edge technology companies. We help them with the full software development work on ongoing basis and Rohit Ghatol if you read out all the bulletin here. For every client we put together a high performance team that is tailored to handle the kind of work they need to do whether it is a single page application cross platform, mobile or any technology work we just assemble the right team and it's a dedicated team for every client and given our expertise on the technology front and agile development practices we are able to actually significantly reduce risk for our clients in building software.

We have enough experience under our belt that we can navigate the technology roadmaps and process risks of software development and really help them deliver on time on budget and more importantly meet the market requirements. And in addition we are able to provide a significant cost advantage to our clients because we have a dual shore paying structure with the team in India and the team in U.S. and consistently we are able to provide a 50% cost advantage compared to all local US teams that the client may need to have.

Finally we … in addition to the expertise and the cost advantage and reduced risk we're still able to provide the flexibility that the client needs that any time in their lifecycle to make their India team into a captive operation we help them to do that and facilitate in fact to do that. So that way working with also our clients gets to deliver their software better, cheaper and have their long term flexibility to have ownership of the team.

With that let's move quickly next page which gives a glimpse of a number of our clients. You maybe recognize some of these clients players or typical clients for our work software companies, typically small to mid-size, a few large ones in there but that's the nature of our work.

So that's enough about Synerzip. Let's come back to the topic at hand so I'm going to take a few questions. I don't think I will be able to take all Rohit Ghatol but I'll take a few. So the first one I'll take is this question related to healthcare industry. So the question is in healthcare industry they have to provide Legacy explorer support for example IE8, so do you have any experience with SPAs in IE8 or any kind of caveats to provide that kind of Legacy support such as for SPA?

Rohit Ghatol: So SPAs are typically meant for modern browser itself but what happens is they have frameworks like in HTML5 world. So the various … what you should be asking is how do you build applications for SPA applications for older versus newer browsers at the same time. Actually if you go to HTML5 tools you will see that HTML5 has a number of frameworks which help you … let me just quote that particular part of the blog. Just quickly I'll go there and talk about it. So, yeah, in HTML5 you got two kinds of things, at design time you can go to frameworks like HTML5 please and figure out what features are supported by which browsers so you can plan your SPA accordingly, so you can plan for IE8 I cannot use these features but for IE9 and 10 I can use these features. So you have to have some decisions made at design time when you're planning your product.

The second thing is that at runtime you can use frameworks like modernizer to kind of degrade your application gracefully while using an IE. So you can have SPA application build in such a way that it falls back to a basic version when you're using IE8 but when you're on Chrome, Mozilla or IE11 it gives you the maximum advantage. So that's quite possible using … using SPAs but it's definitely more work.

Hemant Elhence: Okay. Let's take another question which is the role … does the role of a developer evolve in this SPA landscape? So the question is, is it more like the developer has to be integrating tools and understanding how they are packaged rather than the conventional roles the developers used to have?

Rohit Ghatol: That's an advantage when you go to a tool like Yoeman. If you're using a tool like Yeoman it extracts developer from all the caveats of packaging all these frameworks. So I would go with AngularJS generator it would do everything required to get AngularJS product with Grunt tooling, with Bower tooling, in fact even things like CSS languages so it will even give me options whether I want to use SaaS and things like that. But on the second hand a developer does have to step up when he's working with SPA and that's the area of Responsive UI. So typically SPA teams are smaller … in small teams but the developer also has to take care of the low end field and making sure the application feels modern and works like that. The developer can illustrate frameworks like BootStrap to give a good looking SPA since day one without having a need of a UI designer.

Hemant Elhence: We have more questions but let me just make some choice and I'll take one final question. You mentioned module dependency using AMD. How does that work with the dependency tool such as Yeoman?

Rohit Ghatol: Oh yeah, so what we're talking about here is Yeoman dependencies is mostly about libraries like with JavaScript libraries you want to use and typically all these JavaScript library nowadays they come as AMD libraries. If you have a jQuery library you will have an AMD version of the jQuery library, if you have a Knockout library you will have an AMD version of a Knockout library something of that sort.