**Slide 1 - Babe**

**Slide 2 – Library vs Framework**

In the world of programming, the difference between a library and a framework is often a source of [confusion](https://stackoverflow.com/questions/3057526/framework-vs-toolkit-vs-library) and [debate](https://www.quora.com/Whats-the-difference-between-a-library-and-a-framework).

**Slide 3 - Granies**

What the heck is library or even framework, you ask. Bear with me – cause I have everything you need to get you up and running in this awesome world.

**Slide 4.1 - Library**

We’ll start with libraries.

**Slide 4.2**

Library is “**a collection of implementations of behaviour**, written in terms of a language that has a well-defined interface by which the behaviour is invoked.”

**Slide 4.3**

The principle

**Slide 4.4**

What does it mean? To understand libraries we need to go back to beginnings of libraries. One of the biggest issues back then was **how complicated the code was**.

**Slide 4.5**

Since applications had to be written as a whole, when they got bigger and more advanced, the number of bugs drastically increased.

**Slide 4.6**

So a way to split an application into smaller parts was introduced. The new solution became **a basis for what we now know as libraries**.

**Slide 4.7**

**Slide 4.8**

The next big step was the implementation of so-called a **“KISS Rule”** (KISS= Keep It Simple, Stupid).

**Slide 4.9**

To fulfill it any module or library had to **do one thing and do it well**.

**Slide 4.10**

Combining this with the principle in which every difficult problem can be split into multiple smaller problems, which are then independently solved we get the current library system.

**Slide 5.1 - Framework**

Then there are frameworks

**Slide 5.2**

Framework is described as “**an abstraction, in which software providing generic functionality can be selectively changed** by additional user-written code, thus providing application-specific software”.

**Slide 5.3**

The principle

**Slide 5.4**

Let’s analyze what the previous statement means. The first thing we should do is think about what “abstraction” means. Well, in programming it describes **something that is half finished, when it’s up to an end-user (or end-class) to fill the gaps to complete it.**

**Slide 5.5**

 Another thing to note about “abstraction” is that it already contains information about what those gaps are and how they should be filled.

**Slide 5.6**

Let’s take an iterator as an example. **Iterator** is an abstraction that must have a way of getting current object and moving to next one.

**Slide 5.7**

Similarly, a framework contains a list of rules that need to be fulfilled to make the finished product.

**Slide 6.1 - Library vs Framework**

**Slide 6.2**

Now that we know what frameworks and libraries are, let’s compare them, and find out where and when they can be used.

The first and the most important difference between framework and library is the master — slave model.

**Slide 6.3**

**When using libraries you are the master.** That means you are using the library and have control over its usage. You can create an object and destroy it, as well as manipulate it (based on libraries’ API).

**Slide 6.4**

**In libraries your code is calling a function** and gets a response in return.

**Slide 6.5**

On the other hand, **framework is using your code, so it works as the master**. That means you don’t have control over what will happen and when, which is the responsibility of framework. The advantage is that you have less things to worry about, but you also have less control over what can be done. As I already mentioned, frameworks often tie your hands and prevent you from doing both stupid and useful things.

**Slide 6.6**

**Framework is calling your functions and you are returning responses**. This has a tremendous impact on debugging. Since application flow is independent of programmer, it’s impossible to make breakpoint-based debugging, nor follow the flow of data to find issue. The only solution is to use tools provided to you by the framework itself, while in libraries you can just look at data that goes in and comes out of functions to find source of problems.

**Slide 6.7**

Overall, frameworks are more opinionated and libraries are more flexible. Both patterns of abstraction have their place in the world of programming, and while neither is inherently better, it’s important to determine which is appropriate for the problem you’re solving.

**Slide 7 - Vue**

This presentation examines Vue from an outsiders point of view (no pun intended), and proposes that there are some reasons why Vue is one to take seriously. This is not meant to be a comprehensive look at Vue, but rather just a presentation that “sells it”.

**Slide 8 – What is it**

So, Vue.js is a progressive framework for you to create user interfaces. Core focuses only on the view layer of your application but leaves you free to integrate other libraries and tools, and can be entirely used to build more robust single-page (SPA) applications.

**Slide 9.1 – The Creator and his creation**

The Creator

**Slide 9.2**

**Slide 9.3**

**Slide 9.4**

Evan You were born in a town called Wuxi (China). He arrived in the USA and got his BA in Arts and History of Art and then Master of Fine Arts at Parsons School of Design. However, Evan always had contact with the development.

JavaScript was the language that he always had an interest in because it’s simple and at the same time you can share with the world through the web. Meanwhile, until starting to develop the Vue.js, he worked on Google and the Meteor project. Coincidentally, he used AngularJS in the period that worked on Google. So in the end he just took the best parts out of Angular and used them to create The Holy Grail – I mean, Vue.js.

**Slide 9.5**

The Vue.js project was published on Github in February 2014.

The initial idea was always to create a progressive framework, where the core is primarily composed only to focus on the data links as well as on components. It also had the desire to be something more developer friendly, making anyone who knew HTML, CSS, and JavaScript to learn Vue and use it direct and quick.

Vue is a growing framework, and the numbers of users who are joining are impressive.

**Slide 10.1 - Let’s get down to business**

**Slide 10.2**

Vue allows structural flexibility and makes reuse of components easy in your application.

**Slide 10.3**

In short, it does not get on your way. You have the power to write and structure your application however you want it. This freakish feature makes it suitable for building big and highly scalable web applications.

**Slide 11.1 - Focus**

**Slide 11.2**

**Slide 11.3**

Vue has a narrow use case: building user interfaces for the web. The creators know that there are already plenty of libraries out there to help you iterate arrays or handle promises and HTTP requests, so you won’t find any of that duplicated in Vue.

**Slide 11.4**

This focus allows Vue to avoid the bloat of other frameworks. But just as importantly, the minimal surface areas of its API allows its creators to focus more on improving Vue than maintaining it.

**Slide 12.1 - Simplicity**

**Slide 12.2**

Vue code has simplicity built into its design.

One of the most important things you need to know about VueJs before building your first VueJs app is the Vue Instance.

Every Vue application has a root instance called Vue.

A Vue instance can be created by issuing new Vue() with optional objects which can contain template, data etc.  
In another word, we can see Vue instance as an intermediary between your data and view.

**Slide 12.3**

Look at this “hello world” example.

**Slide 12.4**

And you’re ready to start. No polyfilling, no transpiling, no wrestling with a bundler.

**Slide 12.5**

The el: #myApp part tells Vue to render the app inside the DOM element with the id of myApp. The data object is where you place you the data you want to use in your app. We’ve added only one key here, message, which we’re referencing to in our HTML like this: {{ message }}.

**Slide 12.6**

Even those with only a basic understanding of HTML and JS — perhaps designers and juniors on a team — can get a sense of how Vue code works. For more experienced developers, this simplicity allows immediate productivity.

**Slide 12.7**

*“I .. really care about the approachability part of Vue, which is rooted in the belief that technology should be enabling more people to build things”*

[*Evan You, creator of Vue.js — Between The Wires*](https://betweenthewires.org/between-the-wires-evan-you-cb56660bc8a4#.8ju7v068b)

**Slide 13.1 - Flexibility**

**Slide 13.2**

If you want to write a quick and easy app that will run straight from the browser, Vue has got you covered. If you want to build a more sophisticated app with ES6, JSX, separate component files, routing, bundling etc, then Vue can deal with that too.

**Slide 13.3**

Vue is focused regarding what it can do for you, but not opinionated about how you do it.

**Slide 13.4**

For example, if you have a preferred method for writing your templates, Vue lets you do it in any of these ways:

* Write your template in an HTML file
* Write your template in a string in a Javascript file
* Use JSX in a Javascript file
* Make your template in pure Javascript using virtual nodes

This flexibility makes it easy to switch to Vue because React developers, Angular developers, or developers new to JS frameworks would all find Vue’s design familiar.

**Slide 14.1 - Copying competitors**

**Slide 14.2**

A lot of what Vue is getting right is what its predecessors already got right. Just ask an Angular developer.

**Slide 14.3**

Vue’s directives copy Angular’s right down to the syntax: to bind an input to application data in Vue you use *v-model*just like Angular’s *ng-model*.

**Slide 14.4**

As a relatively late mover, Vue has been able to copy what works in other frameworks and avoid what doesn’t. Vue’s state management library Vuex has obvious inspiration from Elm.

**Slide 14.5**

Vue’s components are very similar to Polymer’s customer elements.

**Slide 14.6**

In its latest version, Vue is now rocking a virtual DOM à la React.

**Slide 14.7**

Vue is earning its place by being a great solution if not an original one.

**Slide 16.1 - Where Vue Is Lacking**

**Slide 16.2**

Before you dive into Vue consider these potential drawbacks:

* Vue is young – so it doesn’t yet have the support and wide spread adoption of it’s fellow frameworks. There are far more resources for React, from NPM modules to Stack Overflow answers.
* While Vue is being picked up by large organisations like Alibaba and Baidu, some more examples of high traffic websites would question Vue’s integrity at scale.
* Maybe there’s such a thing as being *too* flexible. An Ember app will always look like an Ember app. But Vue’s multitude of options might become a burden when used in larger projects involving many developers.

**Slide 17.1 -** Documentation

**Slide 17.2**

Incredible. Just incredible. Nothing more to add.

**Slide 17.3**

Evan and the team have put so much effort on making the Vue website a real learning tool instead of just a bucket of information. Most of the sections are well thought, providing tips and actually useful information.

**Slide 17.4**Even if you don’t realise all of its power in the beginning, I am sure that when you start building a bigger project and facing some difficulties, you will be amazed by the fact that you can usually find all the answers within the documentation.

**Slide 18** – The End

Vue is this simplicity and transparency, the community is growing a lot, and I hope I have given you a summary of what I know based on my experience with Vue.