Javascript wrapper for android

Photo: Pixabay (I started my career pretty close to the machine. I learned C and Objective-C to work on the iOS platform. I had a lot of control over the software and I had to learn to manage memory and spend hours or days fixing a problem that was caused by my own negligence and oversight (this was before the ARC). My career grew with a bias toward iOS, and I started exploring Java (and most recently Kotlin) for an Android affiliate.With Android development, memory management had fewer problems due to forgiving the garbage collector. However, I still had a lot of control over the lifecycle of the application, and I had to learn the differences between the two main mobile platforms and from the interface to the functionality that can be accepted. I stayed in my handy bubble, mocking JavaScript developers - mocking the awful performance of web applications. I have refused to touch JavaScript for a long time and have always been reluctant to try a more familiar flavor such as TypeScript. It's still JavaScript, but at least I shouldn't have entered it. I liked my variables and constants to have a clear type. Years of memory management have been on my edge. If you have a car, chances are that you won't get along with the cyclists sharing the road. There is a psychology behind this and it is commonly known as a free rider problem. It took me a long time to realize that the reason I don't like JavaScript developers is because of this psychology. I put years of effort and pain into learning memory management and life cycle application. I thought to myself: JavaScript developers get to share their title without paying the price of memory management or understanding the software lifecycle?!. Photo: News Corp AustraliaAs I grew up and became less content in my career, I started to have my own ideas. My idea was to have a web application with several basic features, such as access control and authentication. This requires several endpoints and resources. I did some research and came across NodeJS. I thought I might as well give it a shot because it looked pretty easy to set up and was well supported. It took me one evening to get this web app up and running. One night. It's a stack I threw together. NodeJS (Server)mLab (Mongo DB)PugJS (Template Language)A few npm set -- save xxx callsA a few hours of hacking around, and I had a functional web application with a functional authentication and access control system. I haven't cared about Redis, Docker, K8s or any of the good stuff yet. My MVP was built. I could test my product. I had a sandbox database with no backups, I had a non-scale product, Digital Ocean. However, I tested this product with real users and realized that it is not as magical as I dreamed. Well, nothing wasted, just found out. I started to learn more about and the unfamiliar package economy that he had. Developing on iOS and Android, you should use several common dependencies like Alamofire and Retrofit. I was expecting a similar level of dependency in JavaScript. I was so wrong. I soon realized why it was called the World Wide Web. It was literally a huge network of dependencies. It was a terrible experience for me. Photo: Pixabay (However, after learning a few rules, such as using a transformer with polyphiles, the most common dependencies and the general ecosystem - I started to get much more comfortable. I was able to create an MVP of any product for a few days. It's an impressive turnaround and it's all thanks to NodeJS.Once the MVP is designed, it's time to invest time elsewhere. I started branching out and taking care of scaling for my specific needs. NodeJS is not a silver bullet. NodeJS is great for some uses, but like any other language, it won't be used just because you're familiar with it. As for the language itself, I used to be disgusted with JavaScript. The main reason for this is that I used to use ActionScript 3.0 some time ago. Both JS and AS3 were ECMAScript so why is AS3 so much nicer than JavaScript? Compatibility. It took me a while to realize how important browser compatibility is to JS. After Babylon began to be accepted, JavaScript began to change rapidly. He made some mega improvements. I really enjoy writing JavaScript now, making sure that outdated code is created under the hood. This post is primarily targeted at people who are mocking JavaScript. I still don't like the idea of native JS apps. However, I understand just how flexible it gives desktop/mobile apps, and it would be hypocritical of me to sit here on VSCode while slating them. I like the idea of hybrid applications that have some basic components written in low-level language, which is the approach a lot of intense desktop web applications do. If you see a developer who hates JavaScript, it's probably because they perceive you as a cyclist taking over the space they think they are entitled to. If you liked this article, please clap and follow me to read my future content. Sign up to get a daily preparation of top tech history! Freeing the brain from all unnecessary work, a good notation releases it to focus on more advanced problems, and in fact increases ... Mental Force - Alfred North Whitehead Programming languages for people, not computers. Computers don't need language except for the machine code. Good programming languages help make it easier for people to reason. This is important because coding is not just a solution to problems. It is also fundamentally about how you think, communicate and understand. On the Internet today, we we one dominant language: JavaScript. JavaScript was created because Marc Andreessen believed that HTML needed glue language, which was easy to use by web designers and programmers who collected components such as images and plug-ins part-time. (1) As front-class developers, we don't have to put up with this. We should not allow ourselves to be limited to one language, especially a language famously cobbled together in 10 days, which is just some good part. While Axel Rauschmeier (for whom I have a lot of respect) may believe it's good for JavaScript to be a mess - I couldn't disagree more. We need languages that help us solve problems, not languages that create more problems for us to solve. The language and idea of words available in the programming language to express your programming thoughts will certainly determine how you express your thoughts and perhaps even determine what thoughts you can express. - Steve McConnelThere is an old joke that goes something like this. In heaven: French chefs, lovers Italians and bankers Swiss.In Hell: cooks English, lovers Swiss and bankers Italians. While we could write about cooking in English, we would lack the vocabulary and paradigms present in French that are essential for cooking. And, in fact, if you know about cooking, you know it's true! There are many central terms for cooking, which are particularly French. Even if you write an English cookbook, you use these French words (e.g. misanza en place, Bain Marie, Kulis, c.k.). The language you work for influences how you think and solve problems. To illustrate this, let's compare how Curry works in Haskell with how it works in JavaScript. We'll look at a simple curry function. Here's the Haskell code: Haskell's features automatically curry. Carring is part of Haskell's paradigm. As a Haskell programmer, I know this. It affects every bit of code I write. The way I think about how to create a function - i.e. solve problems and communicate - involves curry on some level. Here are some JavaScript code that does more or less the same (2): Currying is not implied in JavaScript. If I want to use a curry, I have to implement it. Then I have to use it explicitly. And there is a ton of implementations, not counting the possibility that I could choose to implement it myself. Each implementation has its own costs and advantages. Because I don't have a native curry, I'm going to have to deal with decisions I don't have to make in Haskell. In fact, if I'm trying to do functional programming JavaScript, I may have to make a lot of decisions like this. Every decision my problem is solving, and my code is in several ways: Tackling the fatigue Of the Multiple paradigm (Ramda's way, the Way Of Lodash, the Folktale way, some random NPM module...) more code to manage (more code for reading and understanding) etc. What is trivial and innate in Haskell is proving to be more involved, verbose and alien in JavaScript.To the outcome, we can learn our own version of Whitehead: A good programming language applied to the relevant domain can increase your ability to solve problems. Bad language - or one that doesn't fit well - can make things harder. Languages and problem solvingNation the impact of language on problem solving and communication, it should come as no surprise that programming languages are often created to solve specific problems. A few examples: Ada was written to work with built-in systems and systems in real time. Erlang was designed to write phone apps. ALGOL was designed to clearly describe algorithms. Pascal was written to teach students structured programming. Modern web applications are written to solve a wide range of problems. We have to have a variety of programming languages. We need to be able to work with languages that have been designed for our problem space that fit our way of dealing with problems, or that we want to learn something from. In short: It's time to get rid of just JavaScript.We Can Do This NowEven Although JavaScript is the only language supported by modern browsers, we can work with a wide range of languages to create web applications today. Thanks to the work of many well-supported projects, there are many languages available for compilation on JavaScript. There are even projects that support the use of these languages with popular frameworks, such as React and Vue - for example, Reason-React, PureScript-React, Reason-Vue and others. For a fairly complete list of languages that compile on JavaScript, see Jared Ashkenas' GitHub list. The list includes all .Net, Scala, Haskell, Ruby, Elm, Python, Erlang... on and on. The number of languages on Ashkenas' list is so large that it indicates that it is a companion to Atwood's law: If a programming language exists, someone will write a tool to compile it into JavaScript. Road forward While there are TON languages that compile on JavaScript, this is not ideal. The JavaScript compilation comes with known problems: Debugging a compiled JavaScript can be difficult. If there are tools to help with debugging, many are immature (although debugging Elms very promising). Because of the conceptual differences between languages, some concepts in the same language may not translate well into JavaScript. This can affect performance and understanding. Setting up and performance to your scenarios can be challenging.etc.Innovation always comes with a price. So is stagnation. The cost of stagnation is higher. If we are to make progress in development, we must have more diversity. Someone has to make the way. When there are enough developers who use this language in the web application space, there will be motivation for tool creators, vendors and the community to give their support. Even if we end up with only two or three dominant languages, we will have rich dictionaries and paradigms to work with. NotesSubscribe to get daily overtaking of top tech history! History!