The advantages of cross-platform development are clear: you can write one application and run it on several platforms. Instead of having several applications in various languages, you are able to save a lot of code and expense. The lesser code and languages you use the fewer your bugs, distribution and maintenance effort. I wanted to see how easy it would be to support several platforms with a single language and environment. For this, I chose JavaScript which is gaining more and more impact every day and runs nearly everywhere. I will show you which frameworks I used and how cross-platform development with JavaScript could look like on the basis of some examples.

The agony of choice

There are different options in developing applications for multiple platforms. Xamarin, which uses C# compiled for multiple platforms using Mono, is one great approach. You can write apps and reuse a lot of C# code to support other platforms like desktop, but it lacks the ability to write applications for the web platform. Phonegap, however works well on the web and mobile but features including hardware access can get really time-consuming and you have no possibility to write desktop applications. Besides, it’s missing the native mobile look&feel to which users are used to. If you want to cover all platforms – Android, iOS, Windows Phone, MacOS, Linux, Windows and Browsers – the right JavaScript stack can be the solution. Everyone jokes that there are so many JavaScript libraries out there, but this enables us to find a stack to develop desktop, web and mobile applications with one language and share a lot of code.

Future JavaScript

Some of the reasons why JavaScript is on the rise might be because of more advanced frameworks, libraries and improving standards, so it becomes more and more interesting for serious software development. For using these new standards and use features like promises, arrow functions, classes and much more, you need a transpiler. With this, you can write ES6 and ES7 code and transpile it to current JavaScript afterward. For the examples here I used TypeScript from Microsoft. Unlike another famous transpiler called Babel, TypeScript gives you a static typing system which improves your code quality, makes it more robust and
detects possible code failures before runtime.

Cross-platform stack

**Node.js** and Phonegap were some of the first attempts to use JavaScript across various platforms. Soon promising frameworks like **Electron** for desktop applications and **NativeScript** for developing cross compiled native apps followed. In the following picture, you can see how they fit into the chosen stack. The data is provided by a Node.js web service. Reusable code published as npm module is used by the Aurelia, Electron and NativeScript clients for consuming data from the web service. Let´s have a look at each framework.

![Cross-platform stack diagram]

**NativeScript**
NativeScript is a framework for cross-platform app development developed by **Telerik**, a company known for their UI components for different platforms. NativeScript uses native controls instead of web views and thus solves the problem we had with Cordova/PhoneGap regarding the look&feel and performance. It´s open source and highly extensible. If you want to you are able to use plugins from npm, cocoaPods or gradle directly. NativeScript even has a roadmap where they plan desktop **support for Microsoft UWP and Apple OS X**.
Electron
For bringing your JavaScript application to the desktop, Electron is awesome. It uses Chromium and Node.js and supports Windows, Mac and Linux. Electron is open source and was initially developed for the Atom editor from Github. Many known companies like Microsoft, Facebook or Slack use Electron for their Desktop apps already. Regarding desktop applications, users do not expect native looking applications, but with some effort it is possible to style Electron applications similar to Windows or MacOS applications with libraries like WinJS.

Aurelia
For the Electron part and the web application, I used Aurelia. Aurelia is a newer JavaScript framework that is built on web standards and allows you to write single-page applications with less framework, configuration, and boilerplate code. It is backed by a single company and invented by a person who now works at Microsoft and was in the former Angular 2 team for a while. Aurelia is still less popular but the community is growing every day. We chose it over other frameworks like Angular for several projects already and don´t regret it.

Node.js
The web service is written with Node.js which is really lightweight. This allows me to use JavaScript even on the server-side so you don´t need to translate between server and client side data formats. Node.js uses Chrome´s V8 JavaScript Engine and is developed by a company named Joyent,Inc.

Each of the four frameworks – Aurelia, NativeScript, Node.js and Electron – supports TypeScript. This means we can use typings across the platforms and work with interfaces and classes when using shared code.

In the next blog post, I will show you a little demo application with the chosen stack.
Do you have questions? Feel free to contact me via Twitter @kabaehr or Xing!