**Assignment 1**

**Masters - Mobile Development**

**Please complete by:** The following Sunday, 6PM (BST, unless markers otherwise notified)

**Assignment Link:** [Click on me!](https://mehdinv.notion.site/Assignment-1-11748a2860574eaca8f95e7f5a2ec5d1)

**Note:** Please feel free to use this document as your place to write the answers!

**Student name:** *Daniela Tripon*

*Please write below*

**Task 1:**

I have installed everything.

**Task 2:**

* What is a transcompiler?

A transcompiler is also named a transpiler or a source-to-source compiler and is a compiler that translates a program’s source code into another language, an older version of the same programming language to a newer version or used for an API that is not backwards compatible. There are other compilers for example Emscripten, which convert codes written in C++ and C to code in JavaScript using LLVM. Other examples 2to3 script (Python 2 to Python 3), Google Web Toolkit (Java to JavaScript based on a certain API), c2go (as the name suggests C to Go), Runtime Converter (PHP to Java).

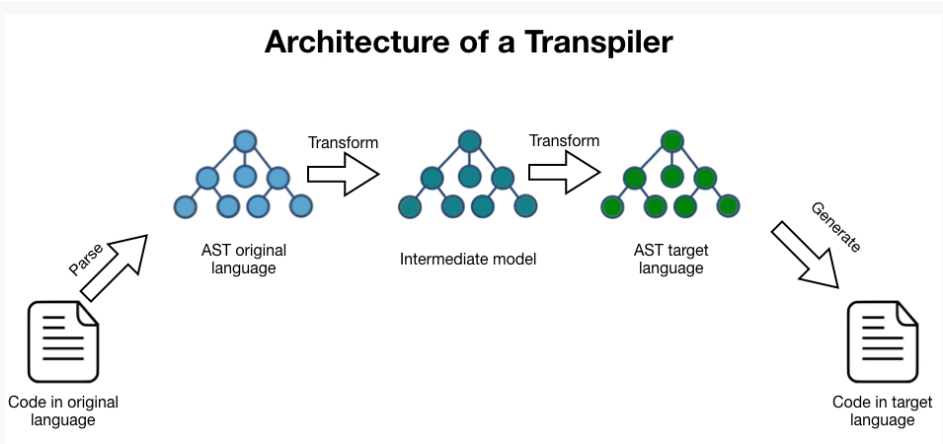


Figure Example Architecture of a Transcompiler

A source-to-source compiler is used between languages that operate at the same level of abstraction and a standard compiler is used to translate source code for higher-level programming languages to lower-level programming languages. In case of API compatibility issues, the compiler will do an automatic code refactoring and allow users to convert code from for example Python 2 to Python 3. Some compilers keep the code to a close form to the original form which makes it easier to debug, while others transform the code to a different form.

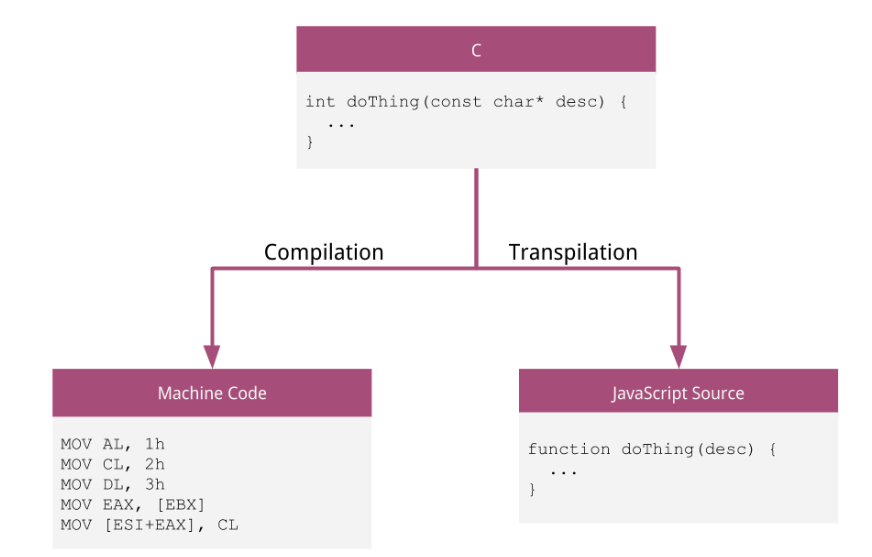


Figure Example Difference between Compilation and Transpilation

* What is the difference between Native, Hybrid and Transpiled approaches for Mobile Development?

Native apps are built using specific programming language for that platform, for example for iOS or Android and requires the app to be built in 2 programming languages. Hybrid approach uses a single common codebase that is packed to run on both platforms and is the most popular choice since you can write code for both iOS and Android.

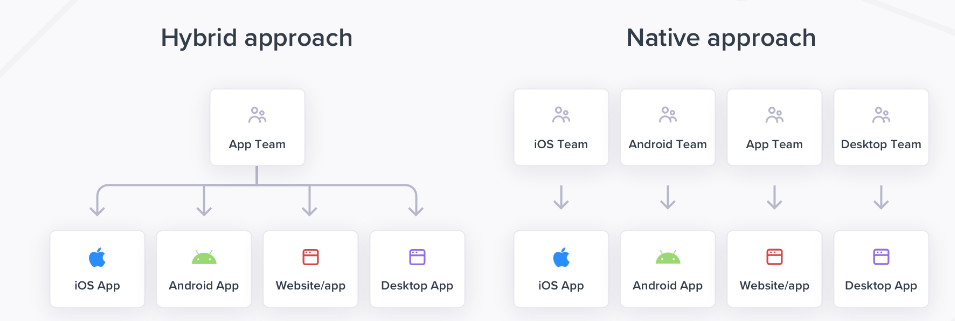


Figure Difference between Hybrid and Native approach

**Native**

An iOS native app is coded using Swift or Objective-C and an Android app is coded using Kotlin or Java. If the apps are written for that platform, for example Android, the downsides are that the app can run only on that platform, you need two development teams with specific skills, and cost more since you need to develop it twice (iOS and Android). Companies that use React Native based on JavaScript frameworks are Shopify, Skype, and Tesla.

|  |  |
| --- | --- |
| Pros | Cons |
| Native behaviour that is familiar for a user since the app is developed for it specifically. | The cost can be double since it needs development and maintenance for 2 platforms. |
| Faster load time. | A lot more time to develop for the same reason as above. |
| Better performance and speed for AR and 3D graphic-heavy games. | Only 7% of developers code for both platforms and for this reason, we need two teams as there are not enough versed developers. |
|  | Improvements and customisation are tricky as they need to be done on both platforms, following different design patterns and deploy codes separately. |

A popular framework for it is Xamarin by Microsoft. The downside is that UI follows platform specific regulations, so the app needs to be adapted to specific platform. Companies that use it are UPS and BBC good food.

**Hybrid**

At core, it uses well known technologies HTML5, CSS and JavaScript. It runs on both platforms using a single codebase and makes use of native plugins. It has access and uses same capabilities as the native app such as Face ID, file storage, fingerprint, camera, and saves time and costs to have it developed and maintained. The app is built in separate packages that are sent to Apple App Store or Google Play, and runs is a web view that is wrapped in a web browser.



Figure Hybrid model example diagram

Nowadays is the popular choice over native tools, and big names like Netflix or Instagram chose this approach. Another benefit is the facile scaling, reduced time, and sources for development (~ 75% less cost than native), and easy to maintain.

A popular framework for it is Ionic and approximately 15% of the apps in store were built on it. Companies that use it are Home Depot, Burger King, and NHS.

|  |  |
| --- | --- |
| Pros | Cons |
| Fast to develop, consistent across devices and platforms. | Can have poor performance and speed for AR and 3D graphic-heavy games because of the overhead of the webview. |
| Cheaper to develop comparative to native apps. | Relies on plugins the choice of plugins used is very important. |
| High number of developers competent in HTML, CSS, and JavaScript. |  |
| Adaptable and easy to maintain. |  |

**Transpiler**

Web experience is different comparative to the native app, and when we need to render webview components at 60fs, not all mobile phones can perform good. For this reason, the alternative is the transpiler which compiles the source code to a required source code and increases performance and usability.

The benefit of it is that we can use for example a text written in HTML react native component and then translate the code in a TextView for Android. We still need to follow platform specific regulations and sometimes some things don’t get transpiled correctly. In other words, it translates the code in a different programming language or a different version. For example, TypeScript code will be translated to JavaScript code.

Examples of platforms that use this: Xamarin, Nativescript, and the well-known React-native.

The entire idea behind using it is to allow and support cross-platform apps.

In game industry, we have Flutter which is an application engine built on a text rendering engine (for e.g. HarfBuzz) and a graphic stack (for e.g. Skia) and which develops a runtime on these as long as they run correctly. A downside can be that not applications have common API for different platforms and if we look at cross-platforms approaches, there could always be issues.

* What is a bridge (in React Native)?

An iOS native app is coded using Swift or Objective-C and an Android app is coded using Kotlin or Java. A bridge is the connection created between JavaScript and the native language of an application and supports two-way communication between these. Most likely we use it at the production level, in case for example we need to reuse Java library elements, we can use React Native and React Bridge to eliminate the need to implement it again in JavaScript.

In React Native, we use asynchronous serialized grouped messages exchanged between Native modules and JavaScript over the React Bridge carrier layer. Please see architecture diagram for React Native structure below.

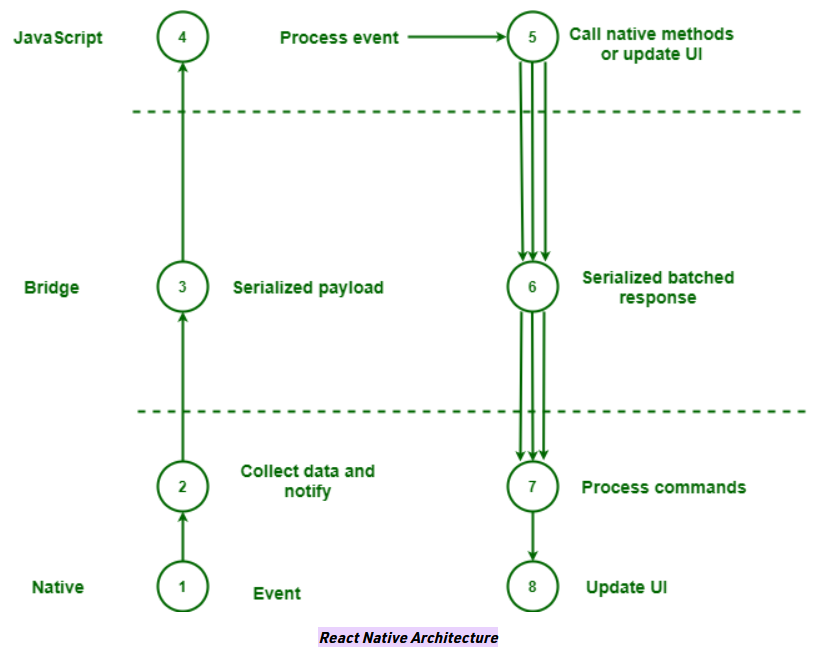


Figure React Native Architecture

In this diagram we can see the React Native, React Bridge, and JavaScript layer. When an incident starts in the native layer (UI/ the main thread), its data is serialized and sent to React Bridge where data is processed and forwarded to JavaScript layer (JavaScript runtime).

* Why do we develop for mobile platforms? What are some reasons you think we've done this, especially since computers already exist and are considered accessible for many?
  + *Feel free to argue against the question or disagree - simply just make a few points why you think mobile development from off in our world*

**Benefits**

* Mobile apps allow users from different platforms access to the app globally, to reach different marketplaces such as Apple App Store, Google Play, and Blackberry.
* Enhances business accessibility since apps can perform even without the internet.
* Most of the activities users carry out nowadays are over the mobile phones: reading books, playing songs or games, emails, shopping, social media 🡪 on-the-go service.
* Through a mobile app, businesses promote brand awareness.
* Mobile apps can keep track of users interests and based on these the committee can plan loyalty programmes.
* Reduces cost that comes with renting the premises. Same store, same products, cheaper to maintain.
* Allows users to share feedback on social media if the app has it integrated.
* Integrated payment services into the app. These are secured, fast, and user-friendly.
* Are great for marketing on the go and use an intuitive support.
* Support push notifications to multiple users at the same time and increase the customer engagement.
* Saves time when is coming to purchasing offering on-touch access to products, no matter where the user is.
* Why do we write our React Native code in JSX? What is JSX as well?

JSX is the abbreviation for JavaScript XML and allows us to write HTML like code in JavaScript and use it in React based applications. It doesn’t require to use the createElement() or appendChild() syntax and it simply adds the element to the DOM (Document Object Model).

When we write JSX code, we pass the code in { } braces. We can write a property, a variable, or any JavaScript expression, which will be executed and get the result returned. A large block of code is wrapped in round braces ( ). Any JSX code must be wrapped in an element or if we want to add two or more elements, we need to wrap them in a parent element (for e.g. a div) or a fragment (<> </>) which will reduce the number of nodes used in DOM. In case the code was not wrapped correctly or if there are any errors with the element, JSX will throw an error.

Conditions:

* All elements in JSX must be closed otherwise it will throw an error.
* “class” is replaced with “className”.
* if statements are not allowed inside JSX statements, only outside, or use a ternary expression as an option.

Benefits of using JSX includes:

* We can use JavaScript functions and methods to build React components,
* Style the components with CSS,
* Add events to the components,
* We can create accessible and semantic markup,
* We reduce code duplication,
* We can use conditionals (for e.g. to hide or show elements).
* Based on these components, we can add flexibility and dynamicity to our components and allow the app to change and grow.

Create native apps

In simple words, React Native tells the app what the UI should look like and is used to build user interfaces. It is a combination between React and native development and is used to develop iOS and Android apps.

It supports the creation of specific components versions for certain platforms, using one codebase only. This way, a team can maintain two or more platforms by using only React. It is great for user experience as this feels natural. View, Image, and Text are agnostic native components that map based on the platform’s native UI used.

Native code is wrapped in React components and interact with native API via JavaScript and React UI and is great for cross-platforms. This allows new teams or existing ones to develop native apps faster. Another benefit is the fast refresh which renders the app instantly.

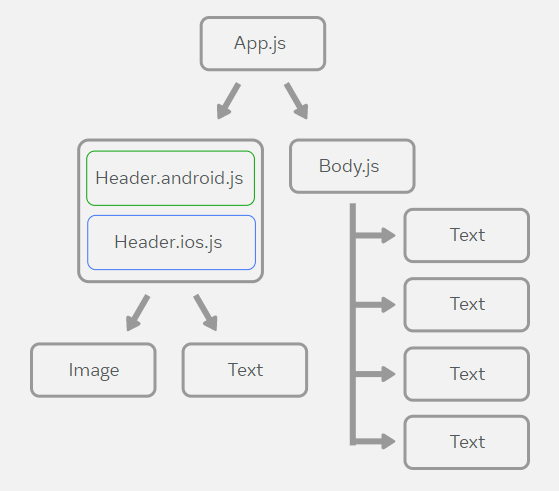


Figure Cross Platform example

* Why do we create components? What's the point? Why not just create the entire code logic in one file instead of what is effectively mini lego parts we're combining together?

We can create custom components that will display according to the platform used (iOS or Android). They can be class or functional component (the last one is much preferred as is easier to use with hooks).

React core components:

* Specific to iOS: ActionSheet
* Specific to Android: BackHandler, DrawerLayoutAndroid, PermissionAndroid, ToastAndroid
* Basic components: View, Text, Image, TextInput, ScrollView, StyleSheet
* List Views: FlatList, SectionList
* User Interface: Button, Switch
* Others: Alert, Animated, StatusBar, Modal, Dimensions, ActivityIndicator, Linking, RefreshControl, KeyboardAvoidingView.

Components are built as independent pieces of code that are reusable and return HTML. This way the code is easy to use, read and maintain, we can adapt components based on the needs of the application, easy to handle errors, and enhances the app performance.

A code that is for example 1500 lines long, is very hard to read and debug. If it is a large project, with lots of files, where the code is very long, it is almost impossible to keep track and to resolve problems. By using components, a functional or class components deals with a certain problem only which makes it easy to debug in case of errors. Also, when we are testing a component only, is easy. Components are maintained in folders and subfolders, following a structured way for easy maintenance. This way, we know exactly where to locate the component or the error.