Follows the criteria through with comparison of the different ECMAScript engines are compared:

- ECMAScript compatibility
  - What versions are supported?
- size of runtime in MB
  - This is something we will explore somehow throughout the project, but it's good to see if there are any hints upfront on this point
- ease of integration with a manager and supported management functionalities
  - languages bindings available
  - support for bidirectional interaction, eg callbacks from javascript world to manager's world and vice versa
- Open Source (or not)
  - project health in terms of github momentum
- WASM support
  - This is a nice to have; we don't specifically want it/need it now, but it will be a future need
- ability to precompile ECMAScript
- intelligence within the runtime
  - how much optimization is going on under the hood? (lots for v8)
- support for isolation
  - Can we run different functions within the same runtime which do not have access to each other's data?
- multithreading support
  - Is there support for running in multithreaded mode

Link to table comparing compatibility to standard proposed
https://en.wikipedia.org/wiki/ECMAScript

Raw list of candidates:

- Chakra
- JavaScriptCore
- V8
- Hermes
- JS-interpreter
- Rhino
- duktape
- Njs
- Engine262
- Graaljs
- Spidermonkey
- XS
- mujs

Candidates important information

Chakra	- Will not be supported anymore from Microsoft and the project will be supported by the community. - does not include external libraries provided by other frameworks. Input Output apis must be implemented from the developer (i.e. document.write())
JavaScriptCore	- Basically no information are available - Parallel execution through parallel single threaded execution environments
V8	- Engine developed and proposed by Google
Hermes	- JS runtime environment supporting React Native

Comparison table

	ECMAScript compatibility	**Size of runtime in MB**	**Ease of integration**	**Open Source**	**WASM support**	**Ability to precompile Scripts**	**Intelligence within runtime**	**Support for isolation**	**Multithreading support**	**OS support/architecture support**	**Planned support**
Chakra	5.1 (full) 6.x (partial)		C++, C#, Python, Linux, OS X, CMake	Yes	Yes, but not provided for embedder	Should not be able ?	Simple JIT: low opt. Full JIT: high opt	Should be offered by the Closures	No		Community project
JavaScriptCore	ECMA-262		No integration found	Should be	Yes	Yes	DFG and FTL compilers	Should be offered by the Closures	No		
V8	ECMA-262		C++, C# .NET, Python	Yes	Yes	Yes	A lot	Should be offered by the Closures	No		
Hermes	React Native framework & ECMA-402		C, C++, Python	Yes	No	yes	No JIT precompilation, but ahed compilation	Should be offered by the Closures	No		

	ECMA Script compatibility	Size of runtime in MB	Ease of integration	Open Source	WASM support	Ability to precompile Scripts	Intelligence within runtime	Support for isolation	Multithreading support	OS support/ architecture support	Planned support
Js-interpreter	Limited set of recognized language features		Not found integration	Yes	No		No intelligence	Sandbox each running instance	Yes, multiple instances together		
Rhino	ES6, ES2016+		Java	Yes	Should not be	Yes	Yes	Yes	Yes		
Duktape	E5, Partinal E6, E7		C, C++, Python, Go, Java	Yes	should not be	No JIT compilation	Not much since small footprint		Only one active thread per Duktape instance		

Difference between WASM (web assembly), precompilation and Just-in-time.

Precompilation either to binary -> run directly it or to byte code -> intermediate code representation -> still machine independent

Web Assembly is a target language, such as byte code, that will be further translated in machine executable code. Precompilation means complain hint binary the entire code, so to make it running faster. JIT in added to interpreter so to speed up some part of the execution. (Just in time reduce overhead of an initial recompilation that might be more expensive, optimize some spots when needed)

Protect functions from each other -> same runtime cannot see each others data -> should be supported but at the code level, not offered by the engine itself. At code level exploiting closures. How are they implemented and managed by the JS engine?