Babylon.js JavaScript Game Development

Raja Naseer Ahmed Khan

B.Sc.(Hons) in Software Development

February 02, 2020

**Final Year Project**

Supervisor: Dr Brian McGinley

Department of Computer Science and Applied Physics

Galway-Mayo Institute of Technology, Dublin Road, Ireland.



Contents

1. **Abstract**
   1. About the Author
2. **Introduction**
   1. Sample Code
   2. Tools
   3. Project Overview
3. **Methodology**
4. **Technology Review**
   1. Babylon vs Three.js
   2. Socket.io vs WebSocket
   3. Node vs Hapi
5. **Results**
6. **Conclusion and Recommendations**
7. **Appendices**
8. **Bibliography**

Table of Contents

[Abstract 4](#_Toc34759229)

[2 Introduction 5](#_Toc34759230)

[2.1 Sample Code 6](#_Toc34759231)

[2.2 Tools: 7](#_Toc34759232)

[2.2.1 Sandbox 7](#_Toc34759233)

[2.2.2 Playground 7](#_Toc34759234)

[2.2.3 Spector.js 7](#_Toc34759235)

[2.2.4 Exporters 7](#_Toc34759236)

[2.2.5 Node Material Editor 7](#_Toc34759237)

[2.3 Project Overview: 8](#_Toc34759238)

[3 Technology Review 8](#_Toc34759239)

[3.1 Babylon vs Three.js 8](#_Toc34759240)

[3.2 Socket.io vs WebSocket 9](#_Toc34759241)

[3.3 Node vs Hapi 10](#_Toc34759242)

[Express Hello World 11](#_Toc34759243)

[4 Methodology 15](#_Toc34759244)

[5 System Design 15](#_Toc34759245)

[6 Conclusion and Recommendations 16](#_Toc34759246)

[7 Appendices 16](#_Toc34759247)

[8 Bibliography 16](#_Toc34759248)

Abstract**.** JavaScript is very popular in web developers, like every programming language JavaScript has pros and cons although it is easy to use language, runs on any device that supports web browser, it is not the fastest language performance wise, it is not compiled hence, hard to read and debug code, vulnerable security wise, different browser may understand different syntax of JavaScript.

In this project, we will be experimenting to make a game using JavaScript engine called Babylon. Which is based on pure JavaScript syntaxes and may not be easy task to achieve as compare to other frameworks and game development engine like Unity3D available. For the sake of learning and experimenting, we would go through step by step process to develop a game which will produce a multi-player mode using Node server, and finally we will conclude our experiences.

**Keywords:** JavaScript, Babylon.js, babylonjs, JavaScript game development, webGL, Node, Express

**About the Author:** I am Raja Naseer Ahmed Khan studying at Galway Mayo Institute of Technology (GMIT), Dublin Road Campus, Galway Republic of Ireland. I am developing this project for my final year of B.Sc. (Hons) in software development.

# 2 Introduction

BabylonJS is an open source JavaScript framework for building 3D games and web graphics with HTML5 and WEBGL.

BabylonJS is developed by Microsoft employees in the year 2016.David Catuhe, a Principal Program Manager for the Window & Devices Group at Microsoft is the main person behind developing BabylonJs and making it a big success.

BabylonJs can run on any browser which supports WEBGL i.e. Internet Explorer 11+, Firefox 4+, Google Chrome 9+, Opera 15+, etc. does have WEBGL support.

Babylon can be downloaded or referenced in the script source section of html page.

BabylonJs has the following type of 3D scenes supported for the development of 3D game and in web graphics:

Babylon supports many 3rd party software for development of 3D animations and graphics i.e. Blender, 3DMax.

Lots of support available on www.babylonjs.com

* Draw box, sphere, cylinder, cone, height ground
* Scene can have Cameras and Lights
* Objects can have Meshes, textures and Materials
* Supports Sprites.
* Supports Mesh Intersection and collision detection
* Physics engine plug-in
* Supports Assets/Action Manager
* Supports Solid Particles
* Support for Bones and Skeletons for animation.
* Adding music and sound to the scene

## 2.1 Sample Code

<!-- Following should be in the enclosed in script tag of html page-->

var canvas = document.getElementById(“canvas”);

var engine = new BABYLON.Engine(canvas, true);

var createScene = function(){

var scene = new BABYLON.Scene(engine);

scene.clearColor = new BABYLON.Color3(1, 0.8, 0.8);

var camera = new BABYLON.ArcRotateCamera(“Camera”, 2, 0.8, 10, new BABYLON.Vector3(0, 0, 0), scene);

scene.activeCamera.attachControl(canvas);

var light = new BABYLON.PointLight(“li”, new BABYLON.Vector3(0, 0, 10), scene);

var sphere = BABYLON.Mesh.CreateSphere(“sphere”, 10, 1.0, scene);

return scene;

};

var scene = createScene();

engine.runRengerLoop(function(){

scene.render();

}); [1]

Above code output in browser



## 2.2 Tools:

### **2.2.1 Sandbox**

### **2.2.2 Playground**

The place to try out coding with Babylon.js.

Experimenting and changing any code in the playground and clicking on the Run button will not affect any original code in the playground you currently using. Original code can be restored by refreshing the browser.

You can write the code in JavaScript or Typescript. The playground software compiles the code to JavaScript, in the background, before rendering

### **2.2.3 Spector.js**

Spector is a tool aim to WebGl developers willing to know what happens on their context. It enables capturing all the available information from a frame. You will be empowered to look at your entire commands list with their associated visual states and context information.

You will be able to Explore and Troubleshoot your WebGL and WebGL2 scenes easily. This is compatible with any WebGL capable browser and all the web based 3d engines as well as vanilla scripts

### **2.2.4 Exporters**

### **2.2.5 Node Material Editor**

The Node Material is a new material introduced with Babylon.js v4.1. It lets you create a material based on custom shaders but without having to deal with shader code. All the shader creation will be done using either an UI (the Node Material Editor) or by creating and connecting nodes (the Node Material blocks).

## 2.3 Project Overview:

In this project we will use the said Babylon engine and will make a game which can be run on any browser. Game will be using mouse and keyboard to move around the ground and shoot cannon, laser and bullets. We will also use the socket.io and node express server to run and create multiplayer mode to instantiate multiple player objects. Also, this project will be stored on Heroku.com for hosting online.

# Technology Review

## 3.1 Babylon vs Three.js

Both Three.js and Babylon.js are easy to use libraries for handling of graphics using WebGL animations.

Three.js was created with one goal in mind: to take advantage of web based renderers for creating GPU enhanced 3D graphics and animations. As such, this framework employs a very broad approach to web graphics without focusing on any single animation niche.

This flexible design makes Three.js a great tool for general purpose web animations like logos or modelling applications (great examples can be found [here](http://threejs.org/examples/)).

Where Three.js attempts to bring a wide range of animation features to the WebGL table, Babylon.js takes a more targeted approach. Originally designed as a Silverlight game engine, Babylon.js maintains its penchant for web-based game development with features like collision detection and antialiasing. As previously stated, Babylon.js is still fully capable of general web graphics and animations as evidenced by the demos found on the front page of its [website](http://www.babylonjs.com/).

In the end, these two relatively young frameworks enable web developers to more easily take advantage of the powerful 3D opportunities afforded by WebGL. As such, anyone with an interest in 3D web development should certainly take a closer look at this cutting-edge technology.

## Socket.io vs WebSocket

**WebSocket:**

It is the communication Protocol which provides bidirectional communication between the Client and the Server over a TCP connection, WebSocket remains open all the time so they allow the real-time data transfer. When clients trigger the request to the Server it does not close the connection on receiving the response, it rather persists and waits for Client or server to terminate the request.

* **Key features of WebSocket:**
* WebSocket helps in [real-time communication](https://www.educba.com/real-time-analytics/) between the Client and the web server.
* This protocol helps in transforming to cross-platform in a real time world between the server and the client.
* This also enables the business around the world for real-time [web application to enhance](https://www.educba.com/how-to-build-web-applications-using-mongodb/) and to increase the feasibility.
* The major advantage it stands over an HTTP connection that it provides full duplex communication.
* **Why do we need WebSocket?**
* It provides the full duplex communication which helps in persisting the connection established between the Client and the Web Server.
* It also lives up to the standards and provides the accuracy and efficiency stream events to and from with negligible latency.
* WebSocket removes the overhead and reduce complexity.
* It makes real-time communication effortless and efficient.

**Socket.io**

It is a library which enables real-time and full duplex communication between the Client and the Web servers. It uses the [WebSocket protocol](https://www.educba.com/websocket-vs-rest/) to provide the interface. Generally, it is divided into two parts, both WebSocket vs Socket.io are event-driven libraries

* Client Side: it is the library that runs inside the browser
* Server Side: It is the library for Node.js
* **Key features of Socket.IO:**
* It helps in broadcasting to multiple sockets at a time and handles the connection transparently.
* It works on all platform, server or device ensuring the equality, reliability, and speed.
* It automatically upgrades the requirement to WebSocket if needed.
* It is a custom real-time transport protocol implementation on top of other protocols.
* It requires both libraries to be used Client side as well as a server-side library.
* IO works on work-based events. there are some reserved events which can be accessed using the Socket on server side like Connect, message, Disconnect, Ping and Reconnect.
* There are some Client based reserved events like Connect, connect- error, connect-timeout and Reconnect etc.
* **Why do we need Socket.IO:**
* I handle all the degradation of your technical alternatives to get full duplex communication in real time.
* It also handles the various support level and the inconsistencies from the browser.
* It also gives the additional feature room support for basic publish infrastructure and thinks like automatic reconnect.
* Currently, AFAIK is the most used one and easier to help with vanilla web sockets.

## Node vs Hapi

**Hapi vs Express**

Both frameworks have more in common than they have differences. However, there are some key differences you should consider if you’re choosing between them for a project.

[Hapi](https://hapijs.com/) (short for (Http)API, pronounced “happy”) is a newer framework that abstracts the existing Node API. [Express](https://expressjs.com/) is older and more established. Express code looks and feels more like native Node.

Hapi has more in the core

There are some cases where Express needs middleware to perform a task that Hapi handles internally. Forms processing is a good example.

**Hapi** parses forms data and exposes it on the request object. **Express**, by contrast, needs the body-parser middleware to offer the same functionality.

Express is closer to Node

Express is somewhat less opinionated than Hapi, in the sense that it is less abstracted from Node. Both frameworks are extensible and adaptable. However, Express “feels” more like a native Node application. Hapi provides more abstraction from Node. Long-time Node developers may prefer Express for its familiarity, or they may appreciate the abstractions provided by Hapi.

Hapi uses plugins, and Express uses middleware

Express uses [*middleware*](https://expressjs.com/en/resources/middleware.html) to provide developers access to the request/response pipeline. Developers have access to Node’s req and res request/response objects. An Express application “chains” middleware together to act on requests and responses. Each middleware component has a single, well-defined job to do, keeping concerns isolated within each component.

Hapi, by contrast, uses [*plugins*](https://hapijs.com/plugins) to extend its capabilities. Plugins are configured at runtime through code. There are a wide variety of Hapi plugins, for capabilities including routing, authentication, logging, and more. There is usually a Hapi plugin for every Express middleware component, making Express and Hapi more or less equal regarding capabilities.

### Express Hello World

Here is a bare-bones Express application:

var express = require('express');

var app = express();

app.get('/', function (req, res) {

res.send('Hello World!');

});

app.listen(3000, function () {

console.log('Example app listening on port 3000!');

});

It’s short and sweet and gets the job done. The app.get function defines a handler for the / request, which returns the text “Hello World!”. The handler takes req (request) and res (response) parameters. The last line starts the server.

Here’s what the same sample looks like with a little middleware:

var express = require('express');

var cookieParser = require('cookie-parser');

var app = express();

app.use(cookieParser());

app.get('/', function (req, res) {

console.log("Cookies: ", req.cookies);

res.cookie("greeted", "true").send('Hello World!');

});

app.listen(3000, function () {

console.log('Example app listening on port 3000!');

});

This example includes the cookie-parser Express middleware, which is used to read and write cookies from requests. The middleware extends the res object with methods and properties related to cookies.

Hapi Hello World

'use strict';

const Hapi = require('hapi');

// Create a server with a host and port

const server = Hapi.server({

host: 'localhost',

port: 8000

});

// Add the route

server.route({

method: 'GET',

path: '/hello',

handler: function (request, h) {

return 'Hello World!';

}

});

// Start the server

async function start () {

try {

await server.start();

}

catch (err) {

console.log(err);

process.exit(1);

}

console.log('Server running at:', server.info.uri);

};

start();

Hapi is a little more verbose than Express. Hapi applications center around the server object. The behavior of the application is configured by setting properties of that object. In contrast, the Express app has app.get, and Hapi exposes a server.route collection.

Now, let’s take a look at how cookies are handled:

'use strict';

const Hapi = require('hapi');

// Create a server with a host and port

const server = Hapi.server({

host: 'localhost',

port: 8000

});

server.state('myCookie', {

ttl: null,

isSecure: true,

isHttpOnly: true,

encoding: 'base64json',

clearInvalid: false, // remove invalid cookies

strictHeader: true // don't allow violations of RFC 6265

});

// Add the route

server.route({

method: 'GET',

path: '/hello',

handler: function (request, h) {

const cookie = request.state.myCookie;

console.log(cookie);

return response('Hello World!').state('myCookie', { greeted: true });

}

});

// Start the server

async function start () {

try {

await server.start();

}

catch (err) {

console.log(err);

process.exit(1);

}

console.log('Server running at:', server.info.uri);

};

start();

Hapi includes cookie-handling in its core—we don’t even need a plugin to set and read cookies. The cookie is configured using the server.state property, then read and written in the GET handler for the / route.

Express and Hapi are generally equally capable. Their differences are mostly philosophical. Some applications may benefit from the abstractions provided by Hapi. Other apps may be better off with the “close to the metal” way of Express.

# Methodology

# System Design

# Conclusion and Recommendations

Although it was great learning experience to develop game with pure JavaScript, awe learned lots of critical syntaxes and logics, but I would like to suggest that using any plain JavaScript engine is good for learning but production and ease of development is far reaching. Hence, I would recommend using the Unity or some framework which has support available for the game development.

# Appendices

# Bibliography

|  |  |
| --- | --- |
| [1] | D. C. a. D. Rousset. [Online]. Available: https://www.babylonjs.com/. |
| [2] | K. SUN, "Analysis of JavaScript Programs: Challenges and Research Trends," *ACM Computing Surveys, Vol. 50, No. 4, Article 59.,* p. 36, 2017. |
| [3] | A. B. S. V. J. D. P. C. M. H.C. Vazquez, "Slimming javascript applications: An approach for removing unused functions from javascript libraries," *Information and Software Technology, Elsevier,* p. 12, 2019. |
| [4] | K.-I. D. K. N. D. T. Ioannis K. Chaniotis, "Is Nod.js a viable option for building modern web applications? A perfomance evaluation study," *CrossMark Computing,* p. 23, 2014. |
| [5] | D. M. Noury Bouraqadi, "Test-driven development for generated portable Javascript apps," *Science of Computer Programming, Elsevier,* p. 16, 2018. |
| [6] | A. Liu, "JavaScript and the Netflix User Interface," *Communications of the ACM,* 2014. |
| [7] | D. E. U. K. Berkan Uslu, "Artificial intelligence library for html5 based games: DignityAI," *Sakrya Universitesi Fen Blimleri Enstitusu Dergisi, 21(1), 2017, 1-15,* 2017. |
| [8] | S. W. Crown, "Improving Visualization Skills of Engineering Graphics Students Using Simple JavaScript Web Based Games," *Journal of Engineerng and Education,* 2001. |
| [9] | P. F. Navarro, "3D Programming with WebGL and Babylon.js for Beginners," Zenva Academy, [Online]. Available: https://academy.zenva.com/product/3d-programming-with-webgl-and-babylon-js-for-beginners/?zva\_src=babylonjsdoc. |