Vetche Royale developer documentation

# After cloning repository

Make sure to install the libraries the application uses. For now, all you need to type in the WebStorm terminal/console is the following:

npm install express

npm install socket.io

Beware that we may require more libraries in the future.

# Running the application

To run the web app, type the following in the WebStorm terminal/console:

node src/app.js

To run the app in the browser, type this URL in the address bar:

**localhost:3000**

Remember that refreshing the client is different from refreshing the server when making changes. Make sure to press Ctrl + C to stop the server when in the terminal if you have made changes to the server code.

# The typeCheck object

The important convention to follow in the programming of this application is to use getter and setter functions for all classes. JavaScript doesn’t let us be strict with out data types, therefore we use the two following methods in the typeCheck object to enforce this:

static instance(dataType, parameter) {

...

static primitive(testValue, parameter) {

...

The first function takes in the constructor of a class and a value that is to be modified, then with some magic it checks if the value is an instance of that class. If not, it throws an error and terminates the application. The primitive function takes in a primitive JavaScript value (number, boolean, string) and checks if the parameter is of the same type. It performs the same error check. Do NOT use this with classes.

You may ask, how do I use these functions? Well, wherever you have a setter function in a class or anywhere in your code you care about using correct data types. In a setter, just place this function at the top before assigning any class values. This is also good for checking if values ever get assigned to NaN or undefined unintentionally.

# Naming conventions

**“t\_” prefix**

Indicates that a variable is solely used for testing and is to be removed later.

**“\_” prefix**

On all class variables/properties that you do not want to be accessible outside of the class, add an underscore prefix to its name. This does not make the variable private, but this is rather a naming convention reminding you to create getters and setters avoiding errors.

**S and C class prefix**

On certain classes that have the same name both on client and server but have different functionalities, you distinguish them by adding an S (server) or C (client) prefix on the class name. For example, SEntity would be on the server while CEntity would be on the client.

On the same topic, there may be a need for duplicate classes due to how exporting .js files is different on the client and server. So, there will usually be 2 files with the same exact class but with different file names (S or C prefix) and exporting methods in the /shared folder of the source directory.

# Imports and exports, client and server

**Client side:**

In order to get a reference to a class or function from a file you must type this at the top of your file:

import AnyClassName from “./<yourPath>”

Remember to add a “./” before every file path. This is just how JavaScript wants it. Also remember that you can import that class with any name, as long as it is exported with export default as it is defined in that file, but more on that later.

If you want to include multiple things from a file that exports multiple things, type the following. Be careful, as naming here matters:

import {Class1, Class2, ..., ClassN} from “./<yourPath>”

Exporting works a bit differently. You can choose to export only one class or function for that file upon declaration like this.

export default class MyNewClass {

...

This code means that you are dedicating that file to export this class by default, but you can also import it with any name.

If you would like to export multiple things in a file (even along with a default export), type the following:

export function MyFunc(parameter) {

...

export class MyClass {

...

You can also export them after they are defined like this:

export {MyFunc, MyClass}

**Server side:**

**Exporting and importing is a lot simpler (and less intuitive) on the server side. To export a new class just type this at the end of your file:**

**module.exports = MyNewClass;**

**To import it, type this at the start of your file:**

**var MyNewClass = require(“./<filePath>”);**

# **Sources**

This article helps the ping problem: <http://buildnewgames.com/real-time-multiplayer/>