# Development testing

TODO: should I have a sort of “concluding” paragraph under each level 3 subsection?

This chapter will focus on the implementation of the test application described in chapter (TODO: crossref testapp methodology). Each framework has its own subsection where I describe every aspect of the development and thoughts I made during the process. Each section has its own summary, and finally, there is a conclusion giving a nuanced look at what framework solves the different tasks of a real-time application best.

## Socket.IO

Socket.io is a module for Node.js (TODO: nodejs) that provides real-time through pure JavaScript on both server and client. It has been around since 2011 (TODO: last commitpage)[[1]](#footnote-1), and it aims to provide clean and simple real-time across all platforms:

*“Socket.IOaims to make realtime apps possible in every browser and mobile device, blurring the difference between the different transport mechanisms”.*

While it hasn’t reached 1.0 yet (TODO: check), it is used in production code by several companies, and it is considered stable. Perhaps one of the most “famous” applications that use Socket.IO is Trello – the online “Scrum Board”[[2]](#footnote-2).

### Why I chose it

Node.js is increasingly popular, and the idea of using JavaScript on the server is very exciting! Over the past couple of years, there has been a dramatic change in the way developers think of JavaScript (TODO: sources). Therefore, it was only natural that I chose at least one framework that uses Node.js as server.

Though there are several modules for Node that provides real-time (TODO: link to node modules), Socket.IO stands out from the crowd. It seems to have the largest community, as it is frequently featured at conferences and generally mentioned many times in traditional forums like Stackoverflow[[3]](#footnote-3).

Furthermore, Socket.IO feels like more than just a Node module. It has its own homepage (TODO: homepage) with some examples and demos–all presented in a good looking and easy to understand fashion. I feel this gives Socket.IO a more professional impression, which makes it stand out even more from some of the other modules that exist that seem more like something someone threw together in a hurry.

Socket.IO doesn’t have a lot of documentation, but what it has gives users a quick overview of the module and how to use it. The API documentation (TODO: docs) uses code samples, which I find more useful than a so-called “wall of text”. There is also a wiki page (TODO: wiki) to give information beyond the API documentation.

As stated in the quote from Socket.IO’s homepage in the introduction to this section, Socket.IO strives to blur the difference between the different transport mechanisms. WebSockets is the preferred transport, but if the client doesn’t support it, Socket.IO will fall back gracefully[[4]](#footnote-4) to one of the following transports:

* Adobe Flash Socket (TODO: source), which uses, surprise, Flash to establish a TCP socket connection between the client and the server, thus “mimicking” a WebSocket connection.
* Ajax multipart streaming (TODO: source): An alternative streaming technique to the forever frame technique described in section (TODO: crossref AND should I write this in the essaypart?).
* Forever Frame
* JSONP Polling, which is polling with data type set to JSONP. This allows cross domain requests; something that is not allowed in normal HTTP Polling (TODO: same source as multipart).

### Getting started

Having Node.js installed on your computer, installing Socket.IO is done via a simple command to the Node Package Manager (*NPM*)[[5]](#footnote-5). After that you can require it in any JavaScript file in your project (TODO: codelisting).

If you are new to Node.js, the learning curve is somewhat steep. However, this is almost always the case for other frameworks (not just real time frameworks) as well–it is expected that you know how to use the underlying technology.

Still, Socket.IO provides only simple examples, that demonstrates quite simple behavior, on their homepage (TODO: source) and on GitHub (TODO: source). All of these uses just a single HTMLfile, and a single JavaScript file on the server, a case which is quite uncommon in normal web applications. I missed some more information about how to build more complex apps, or at least some more reference to the other frameworks that are used in Socket.IO’s examples (like Express (TODO: source)).

### Coding environment

As JavaScript code is traditionally just the client part of a web application, it is often written in the same editor as the server code[[6]](#footnote-6). That may be why most of the examples I could find in videos throughout the web either use a Linux based text editor like Vim or Emacs or the excellent Sublime Text, a cross platform editor that has become increasingly popular.

There is also an IDE provided by JetBrains[[7]](#footnote-7) under the name of “WebStorm IDE”, which is designed specifically for JavaScript, HTML and CSS. It also has a plugin that allows for Node.js development. This is the environment I chose to use, as it gives good IntelliSense (TODO: explain + crossref), has good syntax highlighting and lets you debug Node.js applications (TODO: figure).

When it comes to debugging, there are a number of options with Node.js applications in addition to WebStorm. Since Node.js is built on Chrome’s JavaScript Runtime (TODO: ref Node’s homepage), that exposes an extensive debugger over TCP, you can build your own debugger. This is exactly what has been done with the Node Inspector (TODO: ref Git). Using this, you can use Chrome’s familiar in-browser debugger to debug your Node.js code (TODO: figure). I actually found this to work better than the WebStorm debugger for certain cases, especially with functions that were used to get back data from a database.

Another option is to use the debugger that comes bundled with Node.js. This is a command line tool that was surprisingly easy and intuitive to use. However, it requires you to write the keyword “debugger”, in your code instead of setting breakpoints (TODO: codelist), so it really only works for simple cases.

### Code structuring

Perhaps the most unfamiliar aspect of JavaScript compared to other languages, is the fact that it is asynchronous. A common pitfall for JavaScript frameworks is to only provide developers with synchronous tools to use. With Socket.IO this is, thankfully, not the case. Socket.IO follows the WebSockets protocol tightly as it provides and event based architecture. While the WebSockets API only provides few, standard events (TODO: crossref), Socket.IO lets you used self named events in addition to the standard WebSocket API events.

With traditional, object oriented, languages like Java and C#, code is structured into separate classes, which are normally given their own files. In Node.js, code is structured through the use of modules. Modules is a natural way of separating code within different domains[[8]](#footnote-8), and it is also a very nice way of separating logic that can be used in other applications. Furthermore, modules can depend on other modules, which makes you able to build your application using modules as small building blocks. This is how Socket.IO is built up–it is a module, but it depends on a lot of other, smaller modules.

This modularization has both good and not so good aspects. The best aspect of the modularity is diversity: If you need some functionality, for instance a module for communication with a MySQL database, you will almost always not just find it, but find many different alternatives. Using modules also allows for an easy way to contribute to Node.js by making your own modules. This allows for a rather rapid growth of the Node.js project.

However, with so many modules, and seemingly little quality control, developers might end up using quite some time just to find a module that fits their requirements. Furthermore, some modules that are displayed on Node.js’s module page (TODO: source), are no longer maintained, which easily can cause problems if you have used it in production code and a bug arises.

Another problem I had regarding Socket.IO, which also applies to other modules, is that they tend to favor the use of other modules. With Socket.IO for instance, most examples requires you to use Express[[9]](#footnote-9). That meant I had to learn more than what I set out to do.

### Serialization

### Simplicity

### Maturity

### API documentation

### Testing

## Lightstreamer

## Play Framework

## SignalR

## Meteor

## Conclusion

1. 0.7 preview was released May 5th 2011 [↑](#footnote-ref-1)
2. <https://trello.com/> [↑](#footnote-ref-2)
3. [www.stackoverflow.com](http://www.stackoverflow.com) [↑](#footnote-ref-3)
4. The fallback happens ”behind the scenes” so that developers do not need to worry about it. [↑](#footnote-ref-4)
5. The command ”npm install socket.io” installes socket.io and all dependencies right into your project. [↑](#footnote-ref-5)
6. If the server code is C# for instance, it is common to use Visual Studio also for the client side JavaScript code. [↑](#footnote-ref-6)
7. [www.jetbrains.com](http://www.jetbrains.com) [↑](#footnote-ref-7)
8. One might for instance put all database logic in one module. If this is a lot of logic, one can split the “main” database module into several, smaller modules, each with a more narrow responsibility. [↑](#footnote-ref-8)
9. <http://expressjs.com/> [↑](#footnote-ref-9)