Lisp in Summer Projects Submission

Submission Date	2013-10-15 07:06:30
Full Name	Jonas Enlund
Country	Finland
Project Name	CLJSFiddle
Type of software	web app
General category	development tool
LISP dialect	Clojure
GitHub URL	https://github.com/jonase/cljsfiddle/tree/lisp
Did you start this project?	Yes, all the code is written by me
Project Description	I want to upload a free-form 3-4 page PDF composition.
Upload 3-4 page detailed PDF	cljsfiddle.pdf
Build Instructions	Follow the instructions at https://github.com/jonase/cljsfiddle/tree/lisp#local-install
Test Instructions	There are no tests
Execution Instructions	When the server has started, visit http://localhost:8080
Describe any bugs or caveats	The application will not run on old browsers. I have tested it on Chrome and Firefox 25 (Note, Firefox 24 will not work). It should also work with recent versions of Safari (as reported by users).
Screen shots	cljsfiddle.png
Official	I have read rules and have abided by them. I am 18 years of age or older. I am not living in Brazil, Quebec, Saudi Arabia, Cuba, Iran, Myanmar (Burma), North Korea, Sudan, or Syria.

CLJSFiddle

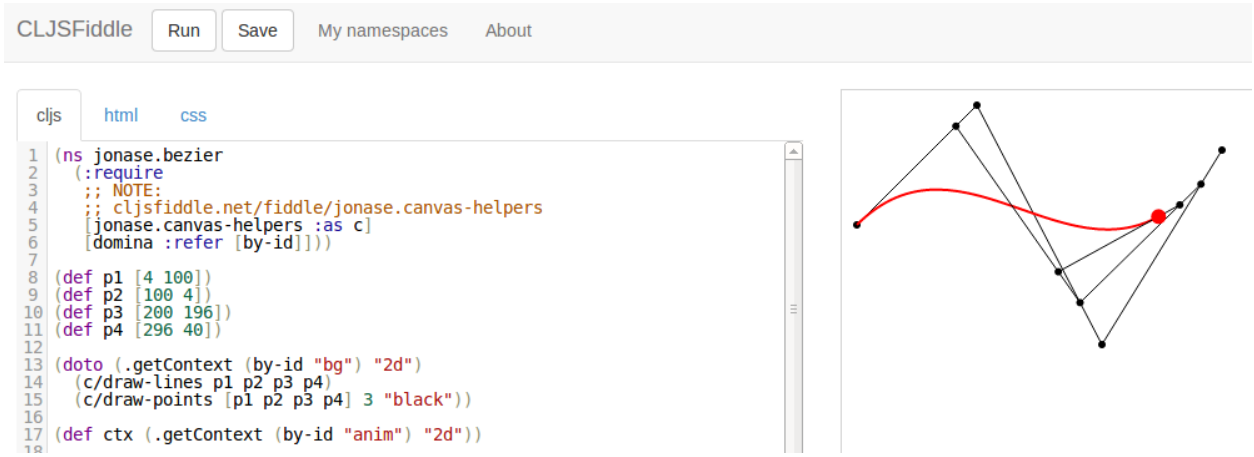
http://cljsfiddle.net

Overview

CLJSFiddle (short for "*ClojureScript Fiddle*") is a ClojureScript[1] playground web application similar to JSFiddle[2] and JSBin[3]. The application lets people write, execute, save and share ClojureScript programs without leaving their web browser. These kinds of live-coding and sharing services have proved valuable to the Javascript front end developer community. The ClojureScript community deserves a similar service.

The target audience is the entire ClojureScript community. New users can go to the site and write their first ClojureScript program, while experienced ClojureScript developers can use the web app for exploratory coding sessions and as a platform to teach and help new users. The web application can also be used for live coding and demos at conferences and other developer events.

A *fiddle* consists of three pieces of user-written source code: ClojureScript, CSS and HTML. The user interface is split in half. The left half has a tabbed pane where the code is written, and the right has a large area where the HTML output of the written program will be displayed.



In order to save fiddles users must log in via their GitHub[4] account. This is required because

[1] https://github.com/clojure/clojurescript
[2] http://jsfiddle.net/
[3] http://jsbin.com/
[4] https://github.com/

CLJSFiddle leverages the Clojure namespace system to give meaningful names to fiddles.

This enforcement enables a strict naming scheme where a namespace must start with the users GitHub name. Apart from the fact that users can't overwrite the work of others (and that it generates nice looking and memorable urls), there is an opportunity for features not available in other similar services: Code reuse across the entire CLJSFiddle code base.

For example, the fiddle jonase.bezier[5] uses code from jonase.canvas-helpers[6] simply by including it in the namespace declaration:

```
(ns jonase.bezier
  (:require [jonase.canvas-helpers :as c] ...))
```

Motivation

I was inspired to create this project because it seemed like both a much-needed service and a relatively challenging project:

- Unlike Javascript, ClojureScript must be compiled before executed on a Javascript runtime.
- ClojureScript is not only a compiler, but also a collection of core libraries (e.g. the immutable collections and the sequence library) as well as the google-closure[7] library. In addition, there are a lot of useful user contributed libraries such as core.async[8], core.logic[9], domina[10], and so forth.
- Clojure programmers are used to interactive and exploratory coding sessions. How can the compilation and delivery of (the possibly huge) libraries be done in a performant enough manner? It is an important objective of CLJSFiddle to retain interactive development in a browser environment.
- How can permalinks be provided when namespaces are used to name fiddles (which can be modified at any time)? How can compiler and library updates be handled without breaking permalinks?

These were the questions that motivated me to create this project.

---

[5] http://cljsfiddle.net/fiddle/jonase.bezier
[6] http://cljsfiddle.net/fiddle/jonase.canvas-helpers
[7] https://developers.google.com/closure/
[8] https://github.com/clojure/core.async
[9] https://github.com/clojure/core.logic
[10] https://github.com/levand/domina

Methodology

CLJSFiddle uses ClojureScript on the client and Clojure on the server. The Datomic[11] database is used for storage.

The Client

The client side of the application is pretty straight forward. CodeMirror[12] is used as the code editor and the resulting program is run in a sandboxed HTML5 iframe[13]. ClojureScript code is sent to the server for compilation. The result of compilation is a Javascript program as well as a list of files the program depends on. The dependency files and Javascript string are put into script tags and bundled together with the HTML and CSS to produce the final output HTML page, which is then added to the iframe's srctag attribute. At this point the result is shown to the user.

The Server

When developing locally, developers use tools like git to manage the history and versioning of their programs. CLJSFiddle also needs to keep all of the history in order to support permalinks. Datomic is a database that accumulates information, and the past is always accessible. Queries are performed against an immutable view of the database at a specific point in time. This turns out to be an excellent match for CLJSFiddle. In CLJSFiddle everything is stored in Datomic. This includes all user-saved fiddles, the ClojureScript standard libraries and some of the most popular user contributed libraries. In addition, the entire Google Closure library is stored in Datomic.

When a new version of ClojureScript, Google Closure, or some other library is released an import script (written in Clojure) can be run which figures out which files has changed relative to the current state of the database and transacts new or updated files into the database. History is preserved so if a fiddle depends on some old library code, any permalinks to that fiddle are unaffected by the update.

Fiddles are treated the same way as library code: when a user saves a fiddle, the system figures out which files have actually changed and only it transacts changes into storage. This means that library code and fiddles are stored side by side which simplifies dependency resolution.

---

[11] http://www.datomic.com/
[12] http://codemirror.net/
[13] http://www.html5rocks.com/en/tutorials/security/sandboxed-iframes/

Dependency Resolution

When ClojureScript source code arrives from the client in order to be compiled by the ClojureScript compiler, it is not enough to simply compile the file and send the Javascript source code back to the client. The generated Javascript will most likely depend on several libraries with their own set of dependencies and maybe even other fiddles. In addition, the client may have requested a permalink which means that the dependency resolution will have to be done against some previous state of the system.

Datomic provides all the features to solve these problems. The program grabs the value of the database at the point in time requested by the client. A ClojureScript source transaction is built and temporarily added to the database. The dependencies are calculated with a simple topological sort algorithm via depth-first search (against the immutable database value) starting with the namespace of the user-supplied source. The result is a sequence of namespaces in dependency order. The SHA-values of the source code corresponding to the namespaces are sent back to the client. The client will use these SHA values to request the actual Javascript in separate requests.

Performance

The obvious performance bottleneck from the user's perspective is the fetching of all the Javascript dependencies. This can be mitigated with client-side HTTP caching. Since the client doesn't get the dependencies as a list of namespaces but instead as a list of SHA-values, the system can cache all the dependencies on the client. The first time the user runs a program on CLJSFiddle it will feel pretty slow as it will probably have to fetch large portions of the ClojureScript core library as well as a few Google Closure libraries. The next time the user runs a fiddle (after some edit to the source) only one round trip to the server is needed in order to compile that single file.

When a new version of a library or a fiddle is added to storage it will be assigned a new SHA which will be sent to the client upon request.

Conclusion

The web application has been up and running since late September, and it appears to work as expected. There are many opportunities for enhancements which I will explore in the coming months:

- The user interface design in particular is somewhat lacking. Line numbers from error messages could be used to highlight the offending code. Structural editing support (i.e.

paredit) should be implemented for the CodeMirror code editor.
- The extensive permalink support is not exposed enough in the UI, it's only mentioned in the 'about' pages.
- A read-eval-print-loop (REPL) could be supported either via a separate console or provided as keyword shortcuts that first send expressions to the server for compilation and the resulting Javascript to the iframe for evaluation.
- Support more browsers. The client uses relatively new features of the HTML5 spec which means the site will not run correctly on older browsers.

In conclusion, I'm really happy with how the system turned out. The edit/compile/run cycle is at least as fast as local development which was a primary goal of mine. It has been well received by the ClojureScript community and I look forward maintain and enhance the system in the future. The use of Datomic and the rest of the Clojure stack has been an absolute pleasure to work with as usual.

Jonas Enlund
October, 2013