Clojure:
Final Project Report

April 18, 2017
Boise State University, CS 354

Wes Gruenberg
Taso Kinnas
Nikki Semmelroth

About Clojure

In October 2007, Rich Hickey made the first public release of Clojure, a new Lisp dialect with some unique characteristics. The release was the product of two and a half years of exclusive work with no outside funding. Hickey's motivation was to create a functional language on a mature platform, designed for concurrency.

While there had been many functional programming languages to date, Clojure was unique in the sense that it runs on the Java Virtual Machine (JVM), thereby inheriting a well-established set of libraries, types, and resources. This means that Java code can be called from Clojure, and likewise, Clojure code can be called from Java. The primary and foremost platform for Clojure is the JVM, but one could use other target implementations as well, including ClojureScript (compiles to Javascript), clojure-py (Clojure in pure Python), rouge (Clojure in Ruby), and CljPerl (Clojure in Perl).

One interesting thing about Clojure is that the development process for the language is community-driven. This means that anyone in the community can submit bug reports or enhancement requests at the Clojure Community website. Cases are created and submitted through the Atlassian JIRA system, and are categorized by project. Those that have signed a Clojure Contributor Agreement can even submit patches for implementation.

Design and Development

Like most other Lisps, Clojure syntax is built on expressions that are parsed into data structures before being compiled. However, because Clojure uses its own data structures, it is not code-compatible with other Lisp dialects. It is the reader that parses the text, not the compiler. And it is the reader that produces the data structures that the compiler will see. The language syntax is defined by the evaluation of data structures such as symbols, lists, vectors, and maps, not in terms of the syntax of character streams or files. In Clojure, functions are first-class objects.

Also like most other Lisps, Clojure features a Read-Eval-Print Loop (REPL) for real-time coding and results. As mentioned above, Clojure can be used in a dynamic way. Running a REPL while concurrently making changes to a project provides immediate feedback. For this reason, the development process can be very fast and luxuriously experimental, since the cost of trying and running is minimal. Though Clojure is compiled, every feature is supported by Clojure at runtime, making it completely dynamic.

When it comes to binding, Clojure uses both lexical and dynamic scoping. The default is lexical scoping, and symbol definition occurs within the code block. The compiler will evaluate the innermost scope and continue up the levels of the scope until the initial declaration of the symbol is found. This is also known as static scoping, and in Clojure is defined by "let." Dynamic scoping can also be created by using the binding macro, but it has its limitations. New bindings can only be created for existing variables, and these bindings are made in parallel of the original.

Why or Why Not Clojure?

There are many reasons why one might choose to use Clojure. Clojure claims to possess all the advantages of a functional Lisp while leaving the disadvantages behind. It has a powerful macro system, but has not inherited the outdated aspects of Lisp, such as no distinction between an empty list and nil. Clojure uses first-class and higher-order functions, while offering its own, unique set of immutable data structures. Additionally, immutability offers protections that can eliminate entire categories of bugs. While some might find immutability to be a nuisance, most swear by the benefits once familiar with programming languages. Clojure is strongly-typed. And most appealing, it is hosted on the JVM it comes equipped with a rich set of libraries, type systems, and memory.

Clojure does have some disadvantages. Error messages can be cryptic and lacking in details, making Clojure difficult to debug. Also, its dependence on the JVM can cause a slow startup. While this is just an inconvenience for single-use manual applications, it could cause some major performance issues if used in a function where this process is done repeatedly. And even though Clojure itself is functional and free of side-effects, real-life Clojure applications will inevitably employ the use of Java libraries, which are less functional and do inherently contain side-effects. Finally, with growing popularity, Clojure is attracting many object-oriented buffs to experiment with Clojure, but many find the functional language syntax difficult to read and write.

Our team members chose Clojure for a couple different reasons. Taso and Nikki had an interest in Clojure for employment purposes. Clojure's growing popularity means that it is also popping up in work environments, either exclusively or in conjunction with other languages, making it a valuable language with which to be familiar. Wes just had a general curiosity about the language and it's capabilities.

Clojure: Getting Started

Getting started with Clojure was no small task. Numerous components were needed before the project could even get off the ground.

- *The Java Virtual Machine:* Because Clojure runs on the JVM, it requires that a copy of the Java Development Kit (JDK) to be installed. Luckily, most programmers already have this toolkit installed and ready to go.

- *Leiningen:* This is a build automation and dependency management tool, similar to Maven or Ant. Leiningen is not technically required to run Clojure, but as with any large project, it is extremely useful in building and managing project dependencies.

- *Cursive:* Because Clojure is a functional program built atop a virtual machine, multiple IDEs are needed for ease of development. Cursive is one of the leading IDEs for Clojure, and is available for download to use as a plugin within the IntelliJ IDE. Since Cursive is written almost entirely in Clojure, it allows a seamless integration of all the Clojure components, such as Leiningen support and a Clojure debugger.

- *IntelliJ IDEA:* Because Cursive is built on IntelliJ, it was necessary to use the IntelliJ IDE for this project. Some team members were already familiar with IntelliJ, so that was a big bonus in terms of startup and troubleshooting.

- *Clojure:* Though all the resources above were installed in order to help build and create the project, Clojure itself can be quickly downloaded and started within a terminal for immediately implementation using the REPL. Our team used the most recent version, Clojure 1.8. It was free and easy to download and install. We also used the REPL during the development process within the IntelliJ IDE.

The Project: Structure and Details

Once set up, the team used an application called Luminus to create the basic skeleton of the project. Luminus is a tool that is accessible through Leiningen that creates a bare-bones template for the basics of an HTML project in Clojure. Leiningen generates the template which can then be built upon and developed in IntelliJ through the use of Cursive. We found Luminus to be extremely helpful for getting started with a web-based project, as it provided just enough structure to get started while still requiring the team to develop the Clojure code needed to implement the project.

The basic structure of the project contains the following components:

- *A source directory:* The source directory is the entry point for the web application. It is primarily responsible for starting and stopping the server and defining base routes and controllers for the application. The source files contain layout helpers as well as an optional framework for a database interaction with SQL. This is where all the Clojure code is written to pull in data, normalize/process the data, and create endpoints for data usage.

- *An environment directory*: This is where the environment-specific code is housed. For example, env might contain default environments for running and/or testing, utility namespaces, etc.

- *A test directory:* This directory is for unit testing.

- *A resource directory:* This directory contains all the html logic for the base layout, pages, and errors. This contains all the code to build and personalize our web page.

- *A project file*: This is a file that manages all the project dependencies.

The Project: Business Intelligence Website

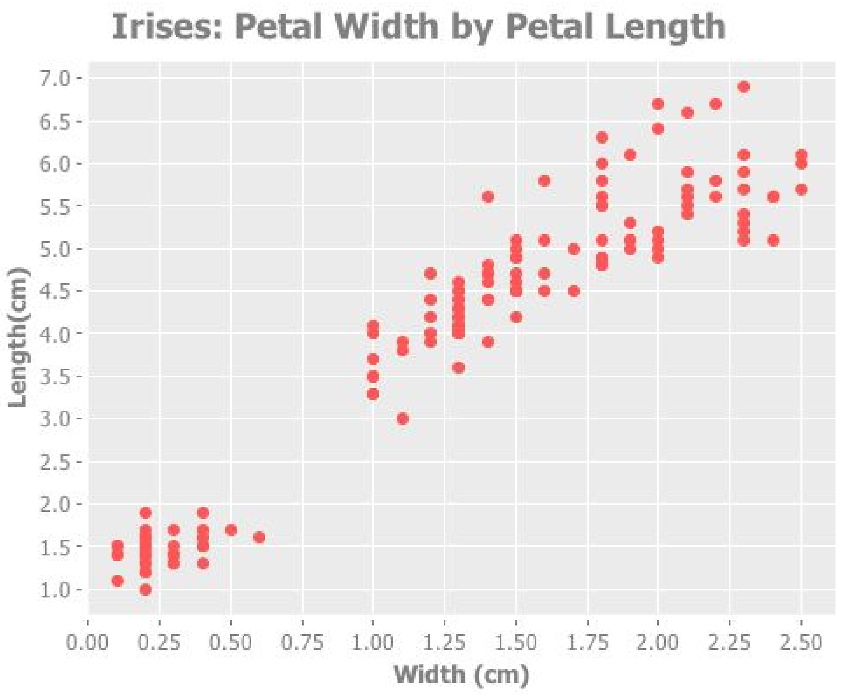
The Clojure project created by our team is a business intelligence website. This project uses Clojure to create a web-based application that can be treated like an interactive document for a given data set. Clojure has established a notable reputation among the data science community. The functional capabilities of Clojure are ideal for pulling in large amounts of data, while the Java libraries can help provide resources to be used for data manipulation and web development. Data can be easily returned to the user with interpretable results in the form of data visualizations.

The project goal is for the site to support three different types of data inputs. The user can choose a CSV from the project's repository or choose a dataset from the Incanter library. The application has the ability to parse CSV files of any number of row or column lengths, but the CSV file must include headers.

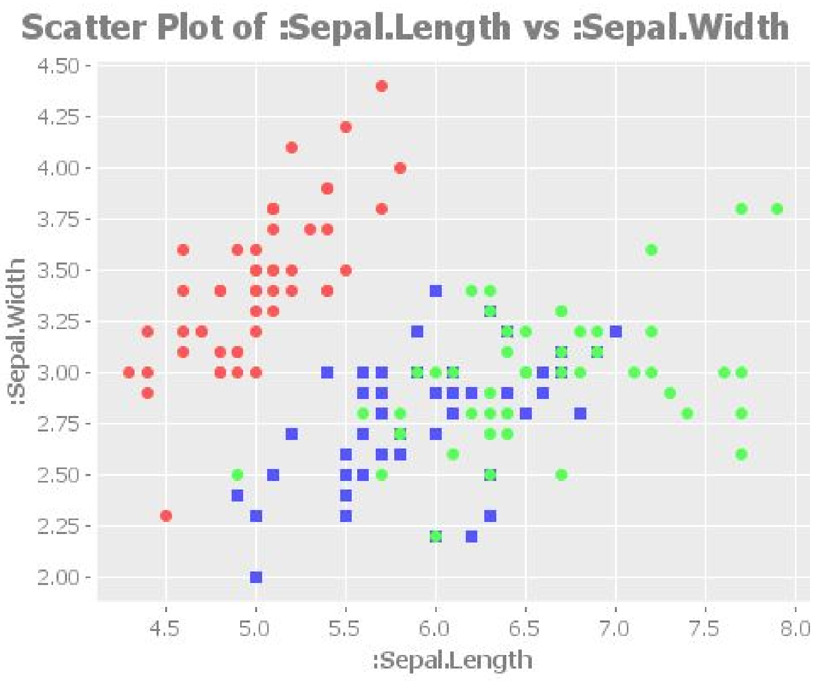
The output of the program is a variety of data visualizations from a library called Incanter. Among other things, Incanter provides various charting and visualization, data visualization function, and statistical functions that we found to be useful for our project. From the web interface, the user is able to choose a dataset, choose a visualization, which the project generates and displays. In addition to our own data, Incanter has some "out of the box" datasets that we utilized for testing and availability.

Included within the project is a well-known set of data called The Iris Dataset. It contains data from 3 different irises' petal and sepal length. Specifically, the columns consist of sepal length, sepal width, petal length, petal width, and species. Below are some visualizations from that data set, as generated by the project.

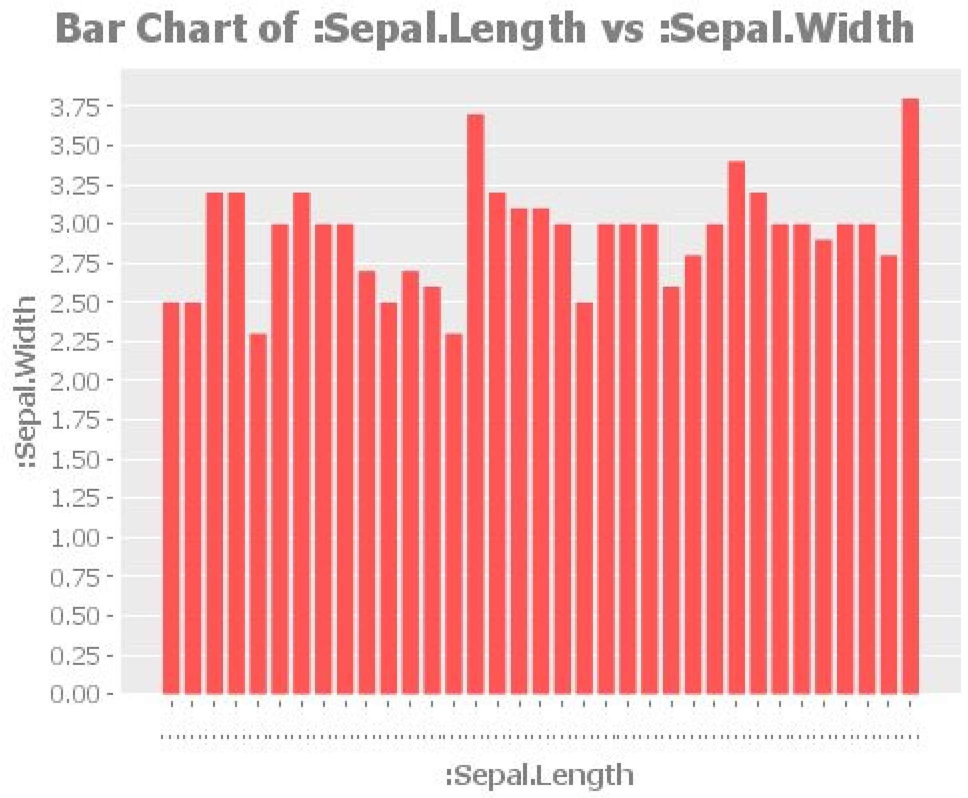
Data from our iris-data.csv:



A scatter plot visualization from iris-data.csv:



Bar graph from iris-data.csv:



Concluding Remarks/Reflection

Everyone certainly found the project challenging. As noted before, just getting the environment setup was a more difficult experience than setting up other programming environments. Once we were up and going, the development of the website seemed comparable to development with other languages we have worked in. There were some various pain points as we worked through the development process. Particular pain points that stuck out:

- adjusting to functional syntax for structuring functions and making function calls
- passing arguments from HTML to Clojure functions
- creating dynamic functions to minimize overall code (never fully achieved this)

We would all use Clojure again for processing data. The wealth of tools and community support make it a great language for data manipulation. Overall, the consensus was that choosing Clojure as the language and the direction of building a basic data driven analytics website was good.

Bibliography

Borges, Leonardo. "Why You Should Use Clojure for Your next Microservice." *Atlassian Developers*. Atlassian, n.d. Web. 3 Apr. 2017. <https://developer.atlassian.com/blog/2016/03/why-clojure/>.

Carper, Brian, and Christophe Grande. "1. Down the Rabbit Hole." *Clojure Programming*. By Chas Emerick. Sebastopol, CA: O'Reilly Media, 2012. 1-3, 20-23. Print.

"Clojure." *Wikipedia*. Wikimedia Foundation, 25 Mar. 2017. Web. 2 Apr. 2017. <https://en.wikipedia.org/wiki/Clojure>.

Hickey, Rich. "API Documentation." *Clojure*. Rich Hickey, 29 Mar. 2017. Web. 1 Apr. 2017. <https://clojure.org/api/api>.

Higgenbotham, Daniel. "Clojure for the Brave and True.". Web. 26 Mar. 2017. <http://www.braveclojure.com>.

McBride, Chris. "Lexical vs Dynamic Scope in Clojure." Blog post. *The Data Point*. RJMetrics, 24 Sept. 2014. Web. 04 Apr. 2017. <https://blog.rjmetrics.com/2012/01/11/lexical-vs-dynamic-scope-in-clojure/>.

Rochester, Eric. *Clojure Data Analysis Cookbook*. Birmingham: Packt Publishing, 2013. Print.

Sotnikov, Dmitri. *Web Development with Clojure*. 2nd ed. Dallas: The Pragmatic Programmers, 2016. Print. 1-3, 20-23. Print.

"What Are the Downsides of Clojure?" Review. Blog post. *Quora*. N.p., n.d. Web. 3 Apr. 2017. <https://www.quora.com/What-are-the-downsides-of-Clojure>.

"Your First Application." *Luminus*. MIT License, 2017. Web. 4 Apr. 2017. <http://www.luminusweb.net/docs>.