TryLinks: An interactive tutorial system for a cross-tier Web programming language

Citation for published version:

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings 3rd International Workshop on Programming Technology for the Future Web

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
TryLinks: An interactive tutorial system for a cross-tier Web programming language

Junao Wu, Arek Mikolajczak, James Cheney

1 INTRODUCTION

Web applications are distributed programs. Typically, computation is spread across the server, Web client (browser) and database in the three-tier model. Each tier has a dedicated responsibility and an established protocol to communicate with other tiers. The three-tier model separates concerns and allows for distribution of user-specific computation (for example for UI events) to the client. In doing so it makes developing the applications more difficult, since each tier uses a different set of languages and technologies. Mismatches between these layers and abstractions need to be handled by the programmer, and this can lead to costly errors and security vulnerabilities.

The Links programming language [7] aims to address these difficulties, by using one language and uniting the 3 tiers. In Links, the web application is a single program (whose type-safety can be checked once and for all at a high level) and the implementation takes care of generating appropriate code for the layers comprising the web application.

However, installing and using Links is non-trivial, making it difficult for new programmers to get started and learn about Links. This paper reports on a Web-based “TryLinks” system, allowing anyone to experiment with Links without going through the installation process. TryLinks was designed with two major functionalities: an interactive Links shell that teaches the basic syntax of Links and acts as a playground, as well as a short tutorial series on how Links is used in practical web development. Tutorials can also be created or modified by administrators. We present the design and implementation of TryLinks, and conclude with discussion of lessons learned from this project and remaining challenges for Web-based tutorials for Web programming languages.
Figure 1: TryLinks read-eval-print loop

website. Although TryLinks was developed with Links in mind, TryLinks itself is implemented using conventional Web programming frameworks (Node.js, Angular Dart, Express, WebSockets etc.) Thus, a similar approach should work to support other Web programming languages.

In the rest of this paper, we give an overview of the capabilities of TryLinks, and then present some details of the implementation, focusing on the particular challenges faced in an online tutorial system that can build Web applications. We conclude with a discussion of the strengths and weaknesses of TryLinks, and directions for future work.

2 DEMONSTRATION

TryLinks provides three basic capabilities:

- An interactive mode providing a Links Read-Eval-Print Loop that runs in a web browser.
- A set of tutorials that cover the basics of Web programming in Links.
- An administrator mode that allows new tutorials to be created or edited.

In this section we highlight the main features of these components of TryLinks.

The REPL page (Fig. 1) provides a connection to a Links REPL. Links expressions can be entered in the standard way, and there is a sequence of hints suggesting different expressions the user may enter, starting with basic arithmetic, lists, pattern-matching, anonymous functions, and comprehensions. The REPL supports the full Links language, although some capabilities are disabled for security reasons, and web applications cannot be run in this mode.

The tutorial page (Fig. 2) consists of three frames: a problem description, a Links code editor, and a pane showing the running Links program. The tutorials introduce the main concepts needed to create simple Links web applications: creating static pages, simple forms and formlets, a simple TODO list, querying from a database, using queries and updates to create a persistent TODO list. Finally the system also allows administrators to create or modify tutorial programs and descriptions.

The reader is encouraged to try the system out using the dummy user account testt with password 12345. TryLinks is available at http://examples.links-lang.org/ and the source code is available on GitHub from https://github.com/links-lang/. A demo video is available here: https://tinyurl.com/y7fz44b6.

3 IMPLEMENTATION

Although TryLinks is mostly a conventional three-tier Web application, Links is unfortunately not yet a suitable implementation language for TryLinks. Instead, TryLinks is implemented using a combination of standard Web technologies and is subject to all of the usual complications of multi-language, multi-tier development. Links executables are run as black-boxes and connected to the Web client using WebSockets. In this section we outline the current implementation strategy.

3.1 TryLinks Server

The backend was implemented in NodeJS using the Express framework. Figure 3 shows the Express backend structure, tuned for TryLinks. The Express application can be divided into two major components: the event loop and asynchronous thread pool. The event loop is a single thread that acts as a handler for incoming requests, and delegates each request to a thread in the thread pool. When a request is handled and a corresponding response is ready, the event loop picks it up and returns it to the sender. Using this structure promises the application is always ready to receive new requests, and handle heavy traffic with high level of concurrency.

What is different for TryLinks is the interactions with the threads in the thread pool. Apart from the usual interactions with the main event loop and with the database, the
Figure 3: TryLinks backend structure based on the Express framework

Figure 4: WebSocket’s role in connecting client and server for Links process

individual threads could also create, close, and interact with WebSockets, which can be connected by a client.

In addition to Express, TryLinks also leverages the advantage of NPM, and made use of many third party packages and libraries. Standard packages for password processing, session management, database interaction, spawning child processes were used. We provide more details of TryLinks’s use of WebSockets because there were a number of complications that needed to be resolved.

WebSocket. The traditional HTTP communication between the client and the server is uni-directional. The client can only request and the server can only respond. This makes it difficult for the server to initiate a message to a client. Unlike HTTP, WebSocket provides “full-duplex communication” [6]. This means both the server and the client can read and write on an established connection.

WebSocket was essential to TryLinks, since the server and the client must send messages back and forth for the Links processes to be dynamic and interactive. Figure 4 shows how WebSocket facilitated the redirection of input and output, on both the client and server.

There are many WebSocket modules available in NPM; Socket.IO [5] was used in TryLinks. It was easy to set up the connection and register callbacks for built-in and custom events, also Socket.IO supported “namespaces” for maintaining one WebSocket channel for each client. However, during development we ran across a bug in the Dart Socket.IO client. After contacting the author, the problem was solved.

REPL Implementation. Using the Node child_process module and WebSocket, the Links REPL shell was almost trivially implemented. Figure 5 shows the basic pipeline of the REPL shell functionality. First of all, the server creates a new WebSocket channel, with the username as namespace, when the api/initInteractive HTTP endpoint is called, after checking for a valid session. In addition, the api/initInteractive HTTP endpoint also spawns a new Links child process, and redirects its input and output port to the WebSocket. Finally it responds with the namespace for the client to connect to.

The client then connects to the correct WebSocket. Upon successful connection a welcome message from Links is displayed, which informs the user that the REPL is ready. User inputs are sanitized according to Links syntax [2] to ensure security. In a local Links shell, one could adjust various configuration settings by using the @set command. In TryLinks REPL shell this is disabled, and a special error message is shown when @set was encountered.

When the client disconnects from the WebSocket, usually by leaving the page, a built-in disconnect event is sent to the server. The server then disconnects the Links child process from the WebSocket, and sends a signal to terminate said process.

Compilation and Deployment Pipeline. The Compilation and Deployment Pipeline refers to the process where the user requests to compile their custom Links program, which
serves a web page. Upon successful compilation, the compiled web page is shown to the user. This functionality was also implemented with the `child_process` module and WebSockets. Figure 6 shows the implementation pipeline of this process.

A common development pattern for web developers starts with editing the source code, then saving and compiling the code, observing the rendered web page, and back to editing. In TryLinks, this pattern drove the design of the compile and deploy pipeline. First of all, the stored source code is pulled from the database. The user makes changes to the source code, then requests to save and compile.

If the compilation is successful, the port to access the web page is sent back to the user. Otherwise, compile errors are shown instead. The user can make further changes and so on. The use of WebSockets was to catch compile errors, and enable re-compile functionality.

### 3.2 TryLinks Client

The TryLinks frontend was designed as a simple MPA (Multi-Page Application), with 3 main pages: `dashboard`, `interactive shell`, and `tutorial` page. This decision was inspired mostly by Codecademy. To give strong consistency with the official Links website, TryLinks adopted the same Links logo and same basic background image. In addition, material design [3] components were used extensively across the frontend to enhance user experience.

**Angular Dart.** Angular Dart [1] is a component based web development framework, loosely following the MVC (Model-View-Controller) pattern. Each Angular Dart component controls a part of screen called a view, which is defined by the component’s binding template. Components nest to form the entire application. To include child components, the parent component must declare them in its metadata, which also contains more information about the component.

**Service.** A service is a broad category encompassing any value, function, or feature that the application needs. A service is typically a class with a narrow, well-defined purpose. Components utilize services by “injecting” the services into themselves. This is called Dependency Injection. Dependency injection “separates the creation of a client’s dependencies from the client’s behavior, which allows program designs to be loosely coupled” [9] and to “follow the dependency inversion and single responsibility principles” [8].

TryLinks was implemented following the best practices of Angular Dart. Each page was well broken down into components; data and event binding were used extensively both intra- and inter-components; a custom service was built to interact with the backend as well as manage session data. Figure 7 shows the architecture of the frontend of TryLinks, slightly tweaked from the standard architecture.

**Interactions with Backend.** As mentioned before, a custom `TryLinksService` was built for interactions with backend. It contained all HTTP calls and a few helper functions that supplied as application global variables, including a configurable parameter for the service base URL. This was helpful in that the URL could be customized without changing the source, which accelerated the deployment process and reduced the possibility of using the wrong URL.

### 4 DISCUSSION

#### 4.1 Performance

Some effort has been made to ensure TryLinks has acceptable performance. Page loading time for TryLinks and several
other language tutorial sites have been measured (with low load) using Pingdom[4], as follows:

<table>
<thead>
<tr>
<th>System</th>
<th>Page Size</th>
<th>Load Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TryLinks</td>
<td>1.0 MB</td>
<td>1.06 s</td>
</tr>
<tr>
<td>TryHaskell</td>
<td>125.7 kB</td>
<td>824 ms</td>
</tr>
<tr>
<td>repl.it</td>
<td>3.5MB</td>
<td>2.08s</td>
</tr>
<tr>
<td>W3School</td>
<td>453.6 kB</td>
<td>1.09 s</td>
</tr>
<tr>
<td>Codecademy</td>
<td>2.4 MB</td>
<td>2.60 s</td>
</tr>
<tr>
<td>JsFiddle</td>
<td>646.7 kB</td>
<td>1.70 s</td>
</tr>
<tr>
<td>Python online shell</td>
<td>727.3 kB</td>
<td>2.96 s</td>
</tr>
</tbody>
</table>

These times measure the front page only, however. Subsequent pages, such as the REPL and Links tutorial pages, have a longer start-up time or delays when a program is re-compiled and the Links executable is stopped and restarted.

During the evaluation with the Links team, it was observed that sometimes the REPL page took about 2 seconds to be loaded and functional. Further investigation showed that this was caused by the Links child process starting up. Although this was technically out of scope of the project, it did affect usability of TryLinks. This may serve as a motivation for Links to refactor the core library to speed up loading time, or to implement a dynamic re-loading functionality.

4.2 User feedback

TryLinks was designed with the goal of helping developers to learn the Links programming language; as a result reviewing this goal was the main focus of evaluation. In this section the evaluation methodology is presented, and the main findings are summarized.

The TryLinks evaluation session was designed as follows. Each user filled out a pre-evaluation questionnaire, which consisted of mostly background questions. Then they launched TryLinks in the browser, and used the software for an hour. At the end, a post-evaluation questionnaire was filled out, which tested the user’s understanding of the Links language, and also included user experience feedback.

The process of evaluation was written into a README. In it some more detailed description of TryLinks was also given to aid the user.

In terms of evaluation group demographics, 9 people with Web or functional programming experience but no previous Links experience were selected. For this group the main evaluation focus was on if TryLinks helped them to learn the Links language. In addition, 2 people from the Links team also participated in the evaluation. For this group the focus was on the organization and presentation of TryLinks, in other words did TryLinks present Links in a way which is easy to learn. Note that the second group’s feedback on learning the Links language is not relevant to the main goal and therefore not included in the findings. However, their critical evaluations are discussed later in Section 4.3.

4.3 Limitations

During the development and evaluation of TryLinks, a number of bugs and user interface problems were fixed. We discuss issues that have not been easy to fix, due to limitations in Links, in the conventional web programming frameworks we are using, or both.

Lack of evaluation system. TryLinks does not have a built-in evaluation system. Instead, users need to read the feedback in the introduction guide on the REPL page to determine whether their responses are correct. Likewise, during the tutorials it is sometimes not clear what the goal is and this is not automatically checked (aside from checking whether the code compiles). Our users would have appreciated a built-in evaluation system, so that they feel more certain that they
have coded correctly. While testing that REPL entries match the expected script is conceptually straightforward, it would require parsing Links results and/or modifying the underlying REPL code. Testing that the submitted solution to an interactive, Web tutorial example meets a specification would likewise require approximating Links program equivalence somehow, which is also nontrivial.

**Extensibility and admin mode.** Another limitation we observed ourselves is that it is not ideal to have a fixed set of tutorials. Adding a tutorial or its supporting documentation then requires manually adding entries to the underlying database. As mentioned earlier, we have experimented with an “admin mode”, in which certain users have the privilege to add new tutorials or edit the initial state of existing ones. However, this is still somewhat limited: there is a single list of tutorials, and new tutorials can only be added at the end. Allowing for multiple groups of tutorials, and for rearrangement of the order of the tutorial list, would make this feature more useful.

**Login requirement.** Currently, TryLinks requires the user to create an account and log in before accessing the REPL or tutorial settings. This is because the Links system runs on the server so starting a new Links process for each access to the REPL or tutorial page is infeasible. Requiring users to log in enables us to limit the number of processes created per user, so that one user cannot deny access to others or crash the system. It also means that we can save users’ partial progress and they can return later. However, many other language learning sites provide at least a basic REPL or tutorial that can be accessed without any login credentials. This is accomplished by providing a stripped-down REPL implementation that can run on the client side. For Links, this has not been implemented but tools such as js_of_ocaml might facilitate this process.

5 CONCLUSIONS

TryLinks is a functional online platform for everyone to try out the Links programming language. Initially planned as an experimental project, TryLinks has received positive feedback from its target audience.

Throughout the development process, several obstacles were encountered, and finally fixed. One of them was using WebSocket with child processes, especially for the “compile and deploy” functionality. It was confusing, at first, to have to register callbacks to the child process input and output ports when the WebSocket first connects, even though the process itself might not have been boot up and functional. Furthermore, in the case of compiling Links programs, when the child process just starts and does not give any output, it is impossible to distinguish whether the process is still booting up, or the compilation has been successful, due to Links core library implementation. This issue forced a complete rewrite of the pipeline, which integrated WebSockets and carefully mapped the sequences of outputs to each scenario.

Another great difficulty was to implement the tutorial page. Being the most complicated page in the whole site, it utilized many technologies and modules. Creating the layout was difficult due to the peculiarities of these modules. In addition, coordinating these technologies presented some difficulties as well.

Last but not least, the first version of TryLinks was not very user friendly, and the initial alpha test reported many problems with the site’s usability. Also bugs were found throughout the system, from trivial ones such as spelling mistakes in the tutorial descriptions, to critical ones such as the rendering port overlapping issue. It took some time to process the comments and rethink about the site. Fortunately, most of these problems were fixed before the evaluations and the final version of TryLinks turned out to be much better.

TryLinks shows that it is possible, using a combination of conventional Web technology and libraries, to provide a Web-based interface for Web programming. However, TryLinks’s design also highlights shortcomings in both conventional Web development and cross-tier languages such as Links. In the future, it would be interesting to extend Links-style cross-tier programming to enable writing a Web tutorial for Links in Links itself.

Acknowledgments. This work was supported by ERC Consolidator Grant Skye (grant number 682315).

REFERENCES