TryLinks: an interactive online platform to learn the Links programming language

Nick Wu(s1450445@ed.ac.uk)

4th Year Project Report

Software Engineering

School of Informatics

University of Edinburgh

2018

Abstract

Links is a web programming language under development in Edinburgh aimed at simplifying web development. Conventional multi-tier applications involve programming in several languages for different layers, and the mismatches between these layers and abstractions need to be handled by the programmer, which can lead to costly errors or security vulnerabilities. Inspired to solve this problem, Links was developed to unite these layers.

Links is currently open for anyone to use. However, installing and using Links is non-trivial, making it difficult for new programmers to get started and learn about Links. The goal of this project is to implement a Web-based "TryLinks" system, for 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.

This report outlines the development process of TryLinks. It reviews existing similar software and extracts appropriate requirements and design. Then it covers in detail the engineering decisions and techniques used in building TryLinks. Finally it evaluates TryLinks, and gives future directions of the software.

Acknowledgements

I would like to thank my supervisor, James Cheney, for continuous support throughout the project. Your advice and suggestions have been the greatest help for me. I would also like to thank my friends and family for their understanding and support throughout university, especially my parents for their endless patience and love.

Contents

Chapter 1

Introduction

With around 40% of the world connected to the Internet [38], web development has become one of the most popular developer occupations in the industry. Creating new webpages has grown to include various languages, tools, and frameworks.

Figure 1.1: The 3 Tier Model of modern web development, adopted from [14]

As web applications get more and more complex, they must be split into layers or modules. The most common division is the *3 Tier Model*, shown in Figure 1.1 from [14]. Each tier has a dedicated responsibility and an established protocol to communicate with other tiers.

The 3 Tier Model has merits in that it separates concerns and makes the web application more modular and easy to maintain and extend. But in doing so it makes developing the application harder to learn, since each tier uses a different set of languages and technologies. Also, 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[1] 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 for web or database programming is non-trivial, making it difficult for new programmers to learn enough about Links to decide whether it meets their needs. Also, Links has its own set of syntax, which is similar to some programming languages but still takes a certain period of time to get used to and master. For any new developer who decides to use Links, currently the ramp up time is rather long, with very little help during the process.

The goal of this project is to implement a web-based "TryLinks" system, supporting simple examples similar to those on the Links project website. More specifically, it should support a "playground", which is essentially an interactive window for user to experiment with Links and learn its syntax. Also, the example programs on the official website can be rewritten into a tutorial series, which developers can take and learn more about developing with Links.

Ultimately, this project aims to help interested developers to play with and learn Links easily and thus promote Links as a programming language.

1.1 Summary of work

To restate the goal, TryLinks should help developers interested in the Links programming language to learn about Links syntax, and using Links for web development. Overall, the project is successful in that a functioning software, complied to the specifications, was delivered and deployed on schedule. This software was built from scratch, with the help of many libraries and frameworks. An independent database was created to store encrypted user profiles and their Links programs. User privacy was protected by firewall on the server, and authentication to the database itself.

Conversely, application can also be extended easily in the future. The "playground" can be extended to include more Links features, and the tutorials series can be expanded to arbitrary length.

The project has also brought opportunities for Links itself. During the development process some minor bugs of Links were discovered, and the existing logging protocol of TryLinks can be extended for performance benchmarking for Links.

An evaluation of TryLinks was carried out after a stable version of the software

[1] http://links-lang.org/

was deployed. During the evaluation, developers with no previous Links experience reported overall positive response of using TryLinks as a tool to learn about Links.

1.2 Structure of report

Chapter 2 The background of web development and the Links programming language is discussed, also some existing relevant software is reviewed.

Chapter 3 The functional and non-functional specifications of TryLinks are listed and explained.

Chapter 4 The architecture of TryLinks is discussed and development process of TryLinks is drawn.

Chapter 5 Design and implementation detail of each component is explained and reviewed. Relevant technologies are mentioned and how they integrate with TryLinks are also discussed.

Chapter 6 Both the evaluation methodologies and findings are presented. Reasoning and critical review are also shown.

Chapter 7 The project is concluded, with future directions given.

Chapter 2

Background

2.1 Web programming

Since the advent of the "dot com" boom around 2000, Internet presence has grown to greater and greater significance for individuals, businesses, and organizations. With it the industry of web programming grew as well, into the current mix of big web technology corporations like Google and Facebook, and small consulting or design agencies. For these developers, the tools, and more importantly the fundamental architecture of web application, have evolved massively, into the current stage of 3-tier architecture: server-side programming, client-side programming, and database programming. In this section we briefly review this architecture and corresponding technologies used, and raise some structural liabilities in terms of development and maintenance. For the sake of brevity and clarity, this section only discusses web development on a high level, but references to detailed materials are given when relevant.

2.1.1 Client-server architecture

Nowadays websites leverage increasingly complicated business logic to provide various functionalities to users. In the interest of separating this logic from the display shown to users, the split of Server-side programming and Client-side programming is brought to use [7]. This architecture utilizes a *Request/Response* paradigm. When the user accesses the web page or performs some specific action, such as clicking on a button, a request is generated on the client side, and then sent to the server. The server then carries out all the complicated computations requested, and finally generates a response to send back to the client. The client displays the response and

waits for further interactions from the user.

Server-side programming As mentioned before, server-side programming mostly concerns the logic of the web application. More specifically, the server can either store static files and retrieve them on request, or exposes a CGI (Common Gateway Interface) for client to invoke scripts, or provides a set of endpoints to handle HTTP or HTTPS requests. Usually the server provides functionalities such as user logic (signup, login), and application specific functionalities.

There are a number of programming languages widely used for server-side programming, including PHP, GO, Python, and Java. Frameworks like Node have also led to the use of languages like Javascript for server-side development.

Client-side programming On the other side of the topic, client-side programming has a very different focus: mostly about display of the web pages, and establishing and handling interactions from the user. It is also responsible for communicating with the server when needed, using asynchronous request/response method (Ajax) [2]. There are 3 main components inside client-side development: HTML for the basic structure of the web page, CSS for custom styling for the application, and Javascript for user interactions and communications with server.

Although client-side development can be done with the aforementioned languages, frameworks like Angular and React offer more extensive and interactive functionalities like parameters binding and routing. In addition, due to some of Javacript's criticisms and obscure behaviours, there are also languages such as Elm [18] and Dart [16] present in the realm of client-side programming, both of which compile to Javascript and provides more language safety such as type safety and exception handling.

2.1.2 Database programming

Almost all websites use a database in some way, either to store users' login information, or to store application specific data or files. Usually the database only handles storing and and retrieving data, regardless of its structure, and a Database Management System (DBMS) is layered on top of it and acts as interface between the database and higher layers. Most web applications have a designated database layer which talks to the DBMS and performs various data-oriented operations.

There are many database systems widely used in industry, such as MySQL and Postgres. Their DBMSs offer more than just data operations, also advanced functionalities like event triggers, rollbacks, etc. Structured Query Language (SQL) is

widely used in these databases for the application server to construct custom queries to interact with the database, and the DBMS offers added performance boosting techniques like query caching and indexing.

2.1.3 Difficulties of web programming

Summarizing from the previous sections, it is clear that to be a competent web developer one must learn many different languages and technologies. This makes the barrier of entry of proper web development high and forces many education materials to only teach a subset of all required to be an independent web programmer. Furthermore, the multiplicity of layers increases the chance of error in message passing between layers, and thus making debugging difficult to pinpoint the root cause of bugs. All of these problems and difficulties were present in development of TryLinks.

2.1.4 A step forward

To address the problems and difficulties mentioned above, cross tier web programming languages and tools are introduced. Among them exist Ur/Web [10] and HOP [47], and of course, Links. Since Links is developed at University of Edinburgh, it has the advantage of easily accessible language expertise. Therefore, I decided to focus on the Links programming language and develop TryLinks accordingly.

2.2 The Links programming language

The Links programming language is designed to unify the 3 tiers of modern web programming [14]. Links is a programming language for web applications that generates code for all three tiers of a web application from a single source, compiling into JavaScript to run on the client and into SQL to run on the database. In this section an example of a web page written in Links is given and explained. Note that Links offers more features than the ones used in this example, and a full set of supported features can be found here[1].

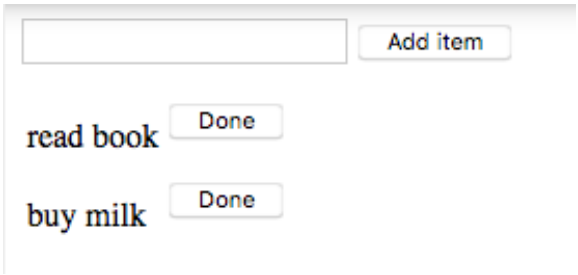
2.2.1 An example in Links

Code Listing 2.1 shows an example Links program. This program compiles to a TODO web application, where a user can insert and remove TODO items. Figure 2.1 shows the compiled web application in action.

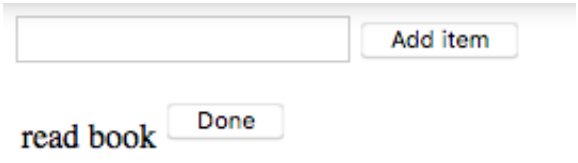
[1] http://links-lang.org/quick-help.html

Listing 2.1: A example Links program

```
var db = database "links";
var items = table "todo" with (name : String) from db;

fun showList() server {
 page
  <html>
   <body>
    <form l:action="{add(item)}" method="POST">
      <input l:name="item"/>
      <button type="submit">Add item</button>
    </form>
    <table>
     {for (item <- query {for (item <-- items) [item]})
        <tr><td>{stringToXml(item.name)}</td>
            <td><form l:action="{remove(item.name)}"
                  method="POST">
                    <button type="submit">Done</button>
                  </form>
            </td>
        </tr>}
     </table>
    </body>
  </html>
}

fun add(name) server {
 insert items values [(name=name)];
 showList()
}

fun remove(name) server {
 delete (r <-- items) where (r.name == name);
 showList()
}

fun mainPage (_) {
  showList()
}
```



(a) TODO items loaded from the database upon accessing the page



(b) Removing a TODO item from the list

Figure 2.1: Compiled web application of the Links example program in Code Listing 2.1

```
fun main() {
  addRoute("",mainPage);
  servePages()
}

main()
```

This example illustrates how Links unifies the 3 tiers of web programming. First a `db` variable is defined to reference the database. In the `showList` function a page template is given. Note that this function is labelled as `server` next to the function name; this is to tell the Links compiler to put the generated code of this function on the server.

The template of the web page is labelled with a `page` keyword. In Links XML, and HTML by extension, can be directly returned by a function, which is done in `showList`. In the template Links specific attributes such as `l:name` and `l:action` are used, to bind value of the input field to a variable and to add a trigger function when the form is submitted respectively. Links provides native event hooks to the DOM directly. In addition, one can interleave Links code in XML, by wrapping it in curly braces. Here a "for" loop is embedded in the template, generating a table entry for each *TODO* item.

Links also offers native interaction with the database. In the "for" loop in the template, each item is drawn from a query of all items in the table `todo` in the database. In addition, there are `add` and `remove` functions which interact with the database as well.

At the end of the program, the `main` function is called, which adds an empty

path to the `mainPage` function, which in turn calls `showList`. Therefore, when the page first loads, the TODO items are retrieved from the database, and also after each alteration.

2.2.2 Current state of art of Links

Links has proven to be harder than average to get started with. At the moment, the Links programming language can be installed by following a step-by-step guide on the Links Github Wiki page [32]. After that, to enable database access, further configuration from [32] must be carried out. Personally this installation process was much more complicated compared to the other programming languages I used. It took two days and some extra help in configuration from a Links expert to finally get Links working normally for me. This inspired TryLinks because a web-based Links platform without extra setup needed would eliminate all the possible complications during the installation process, and can get users started with Links much faster.

Currently Links has a well documented syntax guide [30], which helps developers to get familiar with the basics, and a 6-part tutorial series on Github [31]. However, both of them are procedural and textbook-like, and do not offer a great learning experience. These materials serve as bases for TryLinks, in that the REPL interactive shell incorporated a syntax guide adpoted from [30], and the TryLinks tutorial series was based on [31].

2.3 Learning new programming languages

Nowadays when it comes to learning new programming languages or technologies, about half of developers "take an online course in programming or software development", according to [53] and [54]. For cross tier web programming language, no mature digital learning solutions are available. Therefore a few digital solutions to learn popular programming languages are examined. They serve as an inspiration for defining the functional specifications for TryLinks. Each example is analysed by the following aspects:

- **Main functionality** What does the website offer primarily?
- **User interface** What kind of layout does the website use? How does it present the material for the language?
- **Focus** What part of the language does the website focus on? Is it largely based on learning the syntax or applying the language?

2.3.1 TryHaskell

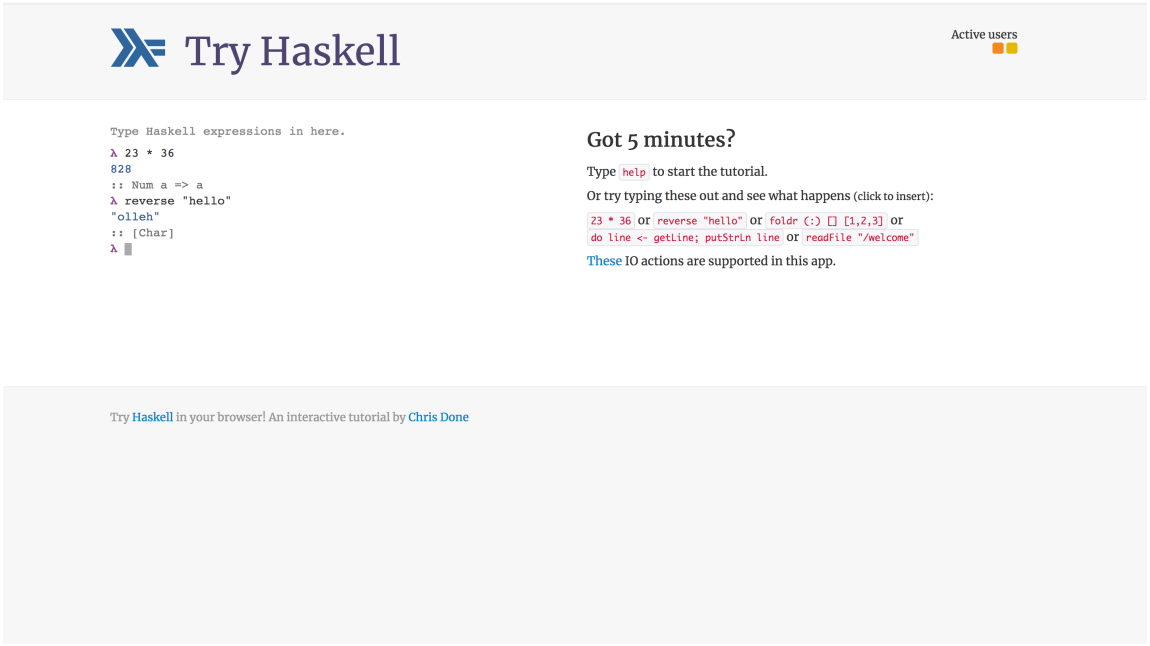


Figure 2.2: Landing page of TryHaskell

TryHaskell (https://www.tryhaskell.org/) is an interactive Haskell shell embedded in a web page, with an added tutorial. Figure 2.2 shows a screenshot of its main web page.

Main functionality TryHaskell primarily features a Haskell shell and a short tutorial by the side. The shell can take Haskell code snippets and evaluate them, giving compile errors when appropriate. There are a limited set of I/O operations supported, likely due to security concerns.

User interface TryHaskell is a Single Page Application (SPA), in other words everything about the application is done in one page load. This means no routing is needed. On the page a minimalist style is adopted: a clear logo on the header, and the main body divided evenly between the shell interface and tutorial description, and some minor footer. This layout immediately directs user's attention to the main body, which is again clearly marked.

Focus The tutorial focuses on the basic data structures and syntax of Haskell, and also some functional programming concepts and examples. The shell itself is unbounded in the sense that user can type any Haskell command[2]. Overall, TryHaskell focuses on getting user on Haskell in a short period, and also giving a somewhat free playground for user to experiment with the language further.

2.3.2 Codecademy

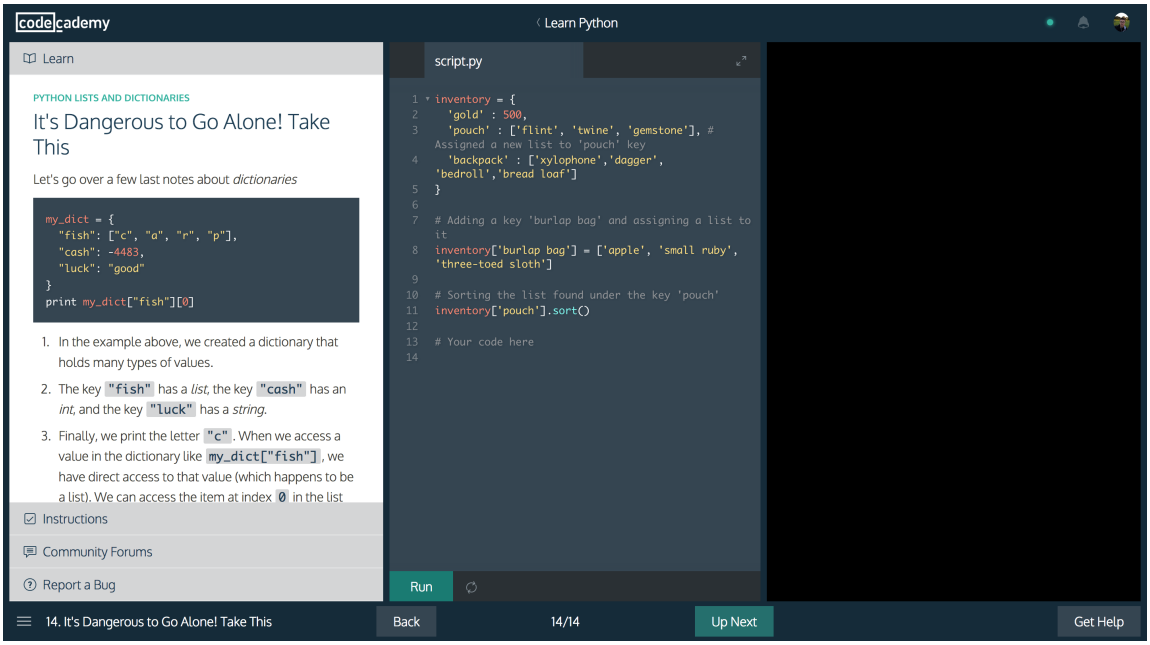


Figure 2.3: An example Codecademy tutorial page to learn Python

Codecademy (https://www.codecademy.com) is a sophisticated online course hub for learning new programming languages and techniques. Figure 2.3 shows a screenshot of a tutorial page of a particular lesson for Python.

Main functionality Codecademy offers a vast selection of languages to learn. For each language, it provides a detailed lesson-based tutorial series. Each lesson shows an example of a particular aspect of the language, and then asks user to practice applying it. It has an evaluation system to determine if the user's code evaluates to

[2] There are some I/O operations unsupported, as mentioned before. But these commands are mostly irrelevant in terms of learning the basics of Haskell.

the expected value. If so, the next lesson is unlocked, otherwise the system attempts to give hints to complete the current task.

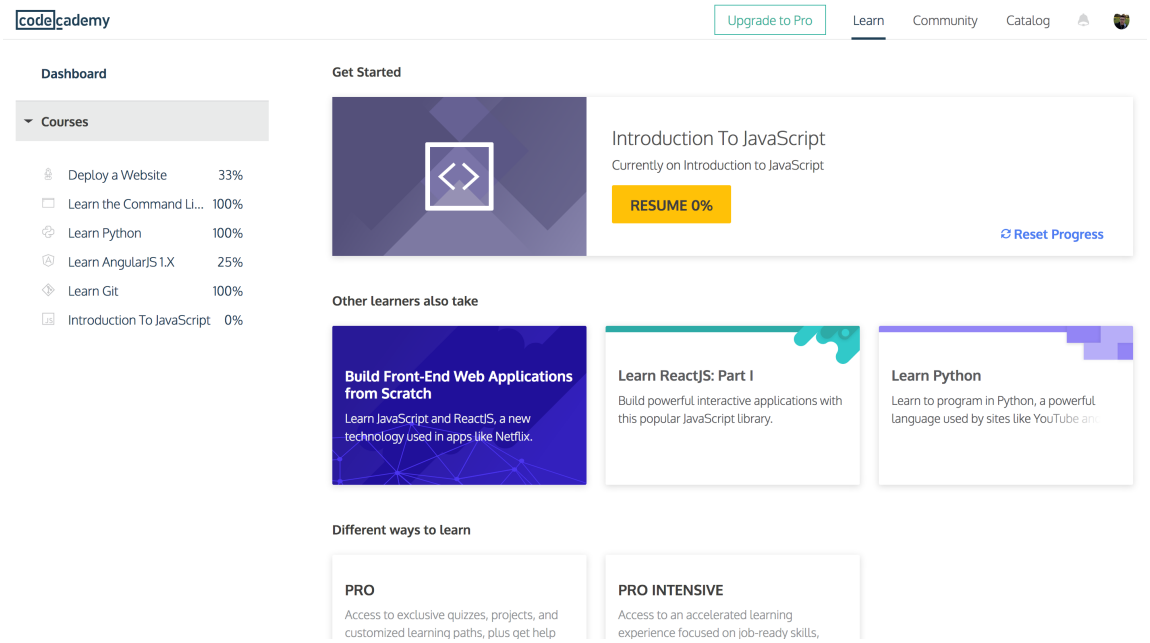


Figure 2.4: An example Codecademy dashboard page

User interface Codecademy resembles a traditional website, with a full set of user logic. Therefore the application is a Multi-page application (MPA), with corresponding pages for each user state. It also has a dashboard page which shows the user's completed courses and current progress on ongoing courses. It also highlights a button to resume last ongoing course for the user, and links to other older courses. Figure 2.4 shows a screenshot of this dashboard.

Focus For the courses to learn new languages, the material focuses mostly on a more comprehensive set of syntax and language features. The practice problems and evaluation system emphasize on getting the exact syntax correct. Overall Codecademy focuses more on the syntax of the language, instead of the practical applications.

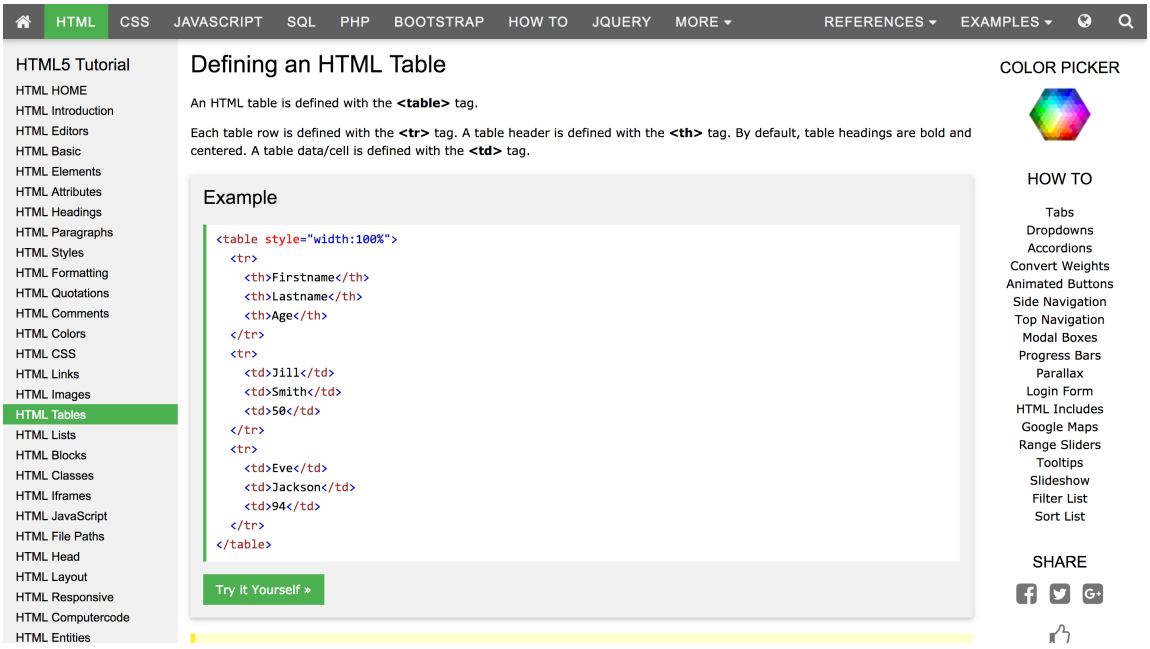


Figure 2.5: An example W3School page to learn HTML

2.3.3 W3School

W3School (https://www.w3schools.com/) is an online repository of chapter-based language guide for mostly web development languages. The style of W3School is close to a digital language textbook, with each chapter explaining in detail one aspect of the language. Figure 2.5 shows a screenshot of a particular lesson for HTML on W3School.

Main functionality W3School offers two major functionalities. First and foremost, the very detailed tutorials of each language feature. For example, in Figure 2.5, HTML `<table>` is introduced. The tutorial shows many examples of tables with different configurations, such as adding borders, adding a caption, etc. For each tunable configuration, the source code is given, and also a hyperlink to a WYSIWYG (What you see is what you get) editor for the user to experiment with. Figure 2.6 shows an example of such source code section.

The other main resource W3School offers is the exercise series. At the end of each chapter, links to some exercises utilizing the introduced language feature is given, as shown in Figure 2.7.

Figure 2.8 shows an example exercise page. Users are to complete the task speci-

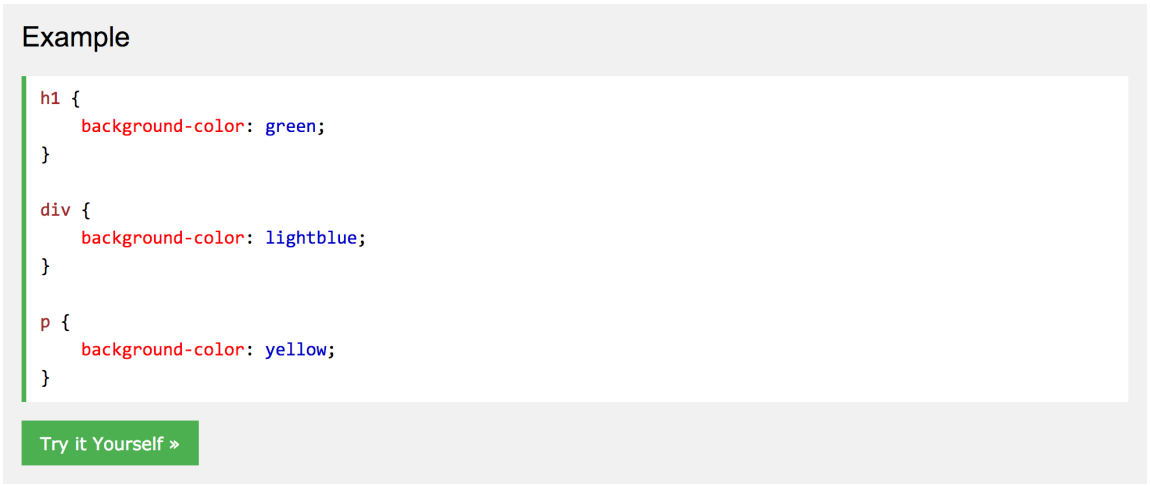


Figure 2.6: An example W3School source code with link to WYSIWYG editor

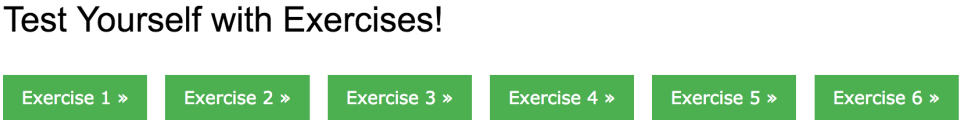


Figure 2.7: An example W3School links to exercise series on a chapter page

fied on the top of the page, by changing the code given. When the code is changed, a new page can be rendered by pressing the "Check Your Code" button. An evaluation system is also in place to inform the user if the change is correct, shown on the upper right corner.

User interface W3School's user interface is a great representative of online programming language documentation pages. It shows a navigation panel of all chapters, and on the main body of the page the current displaying chapter. Each section is clearly separated, with source code highlighted and all relevant buttons shown in a light green color. This helps user to find what's needed quickly. The exercise page resembles the tutorial page for Codecademy, although somewhat less polished and more simplistic with the design and animations.

One criticism about W3School's layout is that in order to maximise page space to accommodate the large amount of material, the editor is not embedded in the chapter page, but rather linked separately. This adds extra jumping between tabs, and potentially breaks the normal reading flow of the chapter. Also the links to exercise series are given at the very end of the page, which sometimes never get

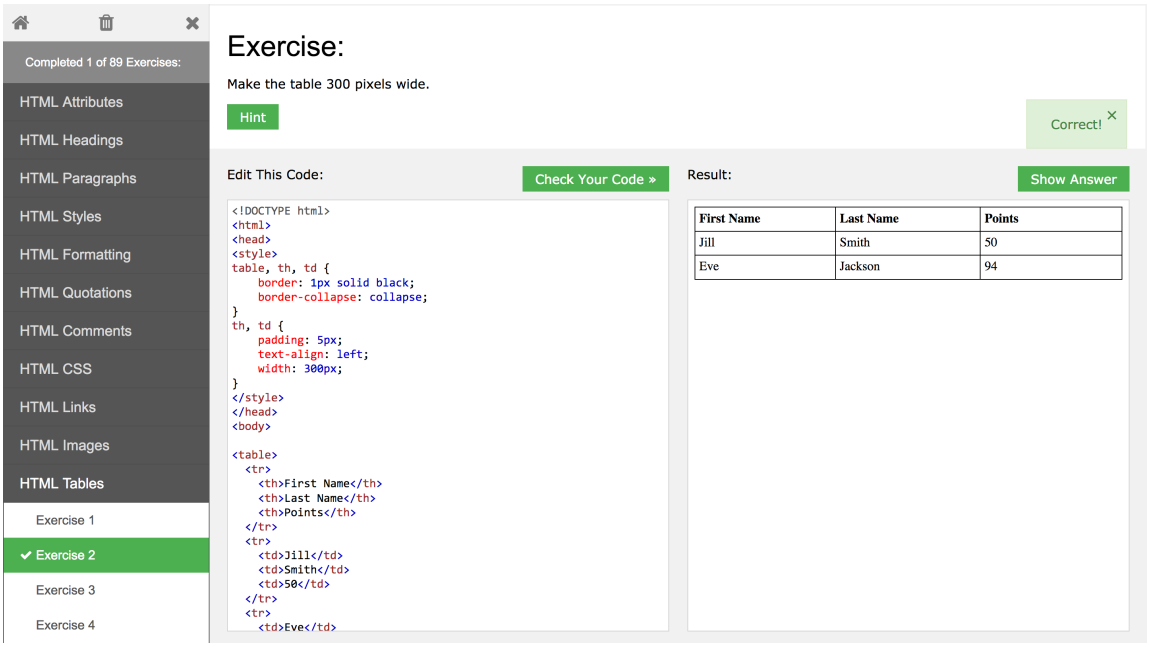


Figure 2.8: An example W3School links to exercise page

scrolled to in long chapters. Also they are in smaller fonts, which undermines their importance.

Focus W3School is very much like a comprehensive documentation of the languages it supports. It focuses on presenting every detail of each language feature, how it is used and the resulting effects. It has high emphasis more on applying the languages features than explaining the syntax.

Chapter 3

Specification

3.1 User Persona

Before drawing out the specific functionalities that TryLinks should provide, it is helpful to think about the audience of TryLinks, their profile as well as what they are looking for. This is termed as *User Persona*, usually used to "highlight the relevant attitudes and the specific context associated with the area of work."[51] Each persona presented has a short description of the fictional character and their characteristics, followed by a *User Story*: a short sentence describing their goal.

This was done at the start of the project, and confirmed with the supervisor to make sure TryLinks is targeted to the right needs. Using these personas, and the existing software features reviewed previously, the functional requirements were derived.

Andrew the Informatics student Andrew is an undergraduate student at University of Edinburgh. After learning Haskell during his first year, he is keen on learning more about functional programming and its applications. He comes across the Links programming language one day when browsing the Informatics department website, and grows interested and wants to get a feel with Links too.

User story: I want to learn a bit about functional programming in a web development context.

Kathy the freelancer Kathy is a freelance developer out and about, and likes to dabble with interesting technologies in her own time, and perhaps use it in a real project. Links is introduced to her by others in the freelance community, and she decides to see if this is a usable technology in the future.

User story: I want to evaluate Links by trying to use it for some simple projects.

Jack from Informatics department Jack just got a job in the Informatics department at University of Edinburgh. His is assigned to the Links programming language team which develops the language itself. Before going straight into the codebase, he wants to experiment with Links from an external perspective, which will help him understand the principles of Links and where improvements lie.

User story: I want to experiment with Links, since I will be working on it.

3.2 Functional Requirements

In this section we list the three major functional requirements of TryLinks. These are inspired by both the user personas, and the existing software reviewed.

3.2.1 Interactive Shell

First and foremost, TryLinks should provide an embedded interactive shell, termed as REPL *(Read-Eval-Print loop)* [42]. Users should be able to type in snippets of Links code and run them. This was inspired by *TryHaskell*, with the goal of helping the user get familiar with Links quickly, and as a playground for experimenting with the language.

3.2.2 Tutorials

Since Links is ultimately a web programming language, TryLinks is responsible to teach, at least a subset of, web programming with Links to the audience. Based on the existing approaches, it was decided that a tutorial series would be ideal. In particular, the style from *Codecademy* was adopted, for it best suited the context of TryLinks: a webpage needs to be rendered for each Links program. Users should be able to navigate through the series, and learn an aspect of Links web programming in each tutorial.

3.2.3 User Logic

The final requirement, although implicit, is to support a basic set of user logic. Since TryLinks is responsible of supporting a tutorial series, each user should only see their own tutorials, based on their progress, and have their own Links programs to work with. Specifically, TryLinks should support the following 4 basic user functionalities:

- Sign up
- Login
- Update Info
- Logout

3.3 Non-functional Requirements

TryLinks is fundamentally still a website, which implies a few non-functional requirements commonly assumed. Broadly speaking TryLinks should be a "secure" application, which includes "a set of properties that should be satisfied" [26]. In particular three of these properties and their significance to TryLinks are discussed.

3.3.1 Availability

Availability means data and services can be accessed as desired. In the context of this project, it implies that TryLinks must always be accessible with little disruption. Although Links is, at the moment, localized in the UK, it can potentially be used by anyone around the world. Therefore, TryLinks should also be accessible at any given time.

More specifically, TryLinks should be resistant to maliciously high amount of traffic, known as Denial of Service (DoS) attacks, and maintain normal operations. In addition, TryLinks should be able to detect or prevent malicious use activities, such as entering a spurious Links computation that lead to an infinite loop, or creating many Links processes. These are addressed in Section 5.

3.3.2 Authentication and authorization

Authentication refers to that user or data origin accurately identifiable; authorization suggests that the correct access rights are given to the correct users. These two properties usually go together. In the case of TryLinks, for example, the user possesses a password only known to them, and upon entering the system they can only view their Links programs.

Chapter 4

High-level Design

4.1 Overview

In this section the top level design of TryLinks is discussed, and the relevant technologies are compared and the most suitable ones are listed. Also the development and deployment methodology are presented and explained.

4.2 Software Architecture

The software architecture of a system is boardly speaking in two fold. First the individual components and their properties, and second the relationships among these components [6]. In this section the high level structure of TryLinks is discussed, particularly the top level components and their relationships are examined.

TryLinks adopted the traditional web project paradigm, with a *Database* to hold user info and their files, a *Backend* tier to implement sensitive logic and expose an API (Application Programming Interface), and a *Frontend* that presents the page and handles user interactions. Figure 4.1 shows the software architecture of Links.

It is ironical that TryLinks was developed with the paradigm that Links itself is designed to eliminate. At the start of the design phase, it was proposed that TryLinks should be implemented in Links. This posed a few disadvantages. First of all, Links is still in alpha and not quite mature enough to be leveraged in large web applications. Some of the technologies needed in TryLinks were not supported in Links yet. Secondly, unfamiliarity with Links could dangerously prolong the development process, ultimately leading to reduced functionalities, or incomplete project in the worst case. As a result, Links was not selected to implement TryLinks, although it

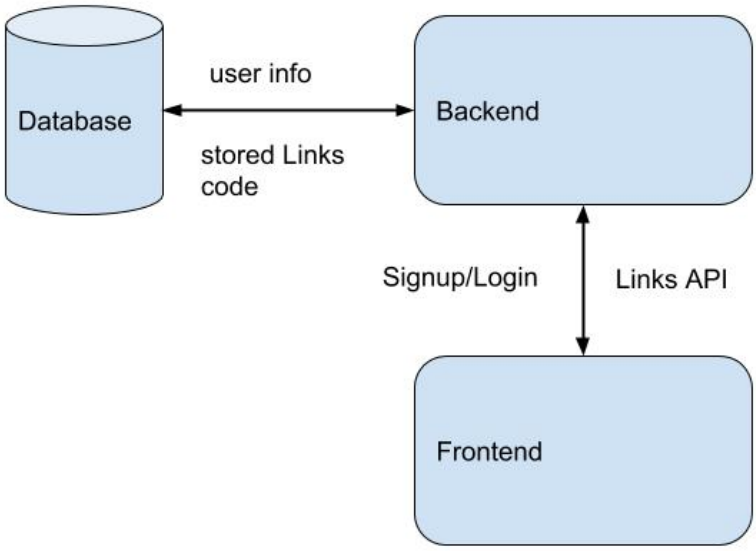


Figure 4.1: Top level architecture of TryLinks

would be interesting to see if a future rewrite of TryLinks in Links is possible. And if so, their respective costs.

4.3 Making Design Decisions

One of the steps of transforming the high level architecture to concrete software design is choosing the right technologies for each component, and tools for development. These design decisions affect the development process in time and effort significantly. In this section the selected technologies for TryLinks are presented, and why they were used over other alternatives are discussed.

4.3.1 Database: relational/non-relational

Relational database stores data in tables of rows and columns [11]. Non-relational database refers to one that represents data in collections of documents [37]. One of the main differences between the two database models is that relational database, based on the Relational Model, requires the stored data to comply with a defined structure, whereas non-relational database has no such restrictions.

In distributed computing, the CAP theorem [24] states that a system can only maintain 2 out of the 3 following properties: consistency (C), availability (A), and network partition (P). Furthermore, since partition of network is innately present, due to the unreliable nature of the network, one must choose one aspect of storage to favour, either consistency or availability.

One of the reasons that non-relational databases have gained so much popularity in recent years is that they "choose to compromise consistency in favour of availability" [27]. Users now expect to be able to view content, even though it might not be up to date yet. However, for developers consistency is more important, in that the latest written version of the source code is expected to be shown and compiled, even though there would be a short period of waiting. In that note, relational database offers DBMS to enforce the ACID properties [28], which guarantees consistency. As a result, relational database was chosen for TryLinks.

Postgres [41] was chosen for implementation of TryLinks, for a couple of reasons. Postgres is supported by Links, which means no adapter for Links program is needed. Also, there is a large community of support for Postgres, both on the official documentation and forum. Lastly, Postgres is free, which keeps the cost of development low.

4.3.2 Backend

There are many frameworks for backend development, a lot of candidates were reviewed for TryLinks, including Flask [21], Laravel [39], and Node.JS [23]. In the context of TryLinks, a large variety of third party libraries were needed, such as WebSocket support, process management, and so on. Node has the advantage of NPM (Node Package Manager) [1], which offers all the needed libraries for TryLinks. Hence, Node.JS was selected for TryLinks implementation.

[1] https://www.npmjs.com/

4.3.3 Frontend

Similar to backend frameworks, there are a plethora of frontend frameworks as well. With such a large pool of candiates, choosing the right one depends more on existing experience. Personally I had used AngularDart [5] for both professional and personal projects, and already had great familiarity over the framework. As a result, AngularDart was selected for development of TryLinks.

AngularDart was initially Google's in-house framework used to develop large scale web applications, such as AdWords [2] and AdSense [3]. It uses the Dart programming language but can compile into plain HTML and Javascript to speed up loading. Also AngularDart offers a Pub package manager [4] similar to NPM, which provides all required libraries for TryLinks.

4.3.4 Version Control

To increase transparency over the development process and facilitate feature planning, version control system was brought to use, specifically git [5]. For code repository hosting, Github [6] was selected, for it is free and offers many useful features such as issue tracking.

4.4 Development Methodology

The development process of TryLinks followed the agile development methodology. Agile [49] is a new methodology of developing software, that focuses on rapid cycles of iteration, with each iteration a more and more functional product. Agile methodology has gained great popularity for both large and small scale software project, and is used now by large corporations such as Facebook [20].

TryLinks was developed in this fashion, specifically with development cycles, targeting incremental completion of features. At the start of each cycle, the goal of the cycle was identified. Throughout the cycle the development process focused only on the goal of that cycle. Finally the goal was evaluated with functional testing

[2] https://adwords.google.com/

[3] https://www.google.co.uk/adsense/start

[4] https://pub.dartlang.org/

[5] Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files.

[6] https://github.com/

and next cycle's goal was built, based on the last cycle. Complying with agile methodology, at the end of each cycle we expect to have a functioning partial product.

Different from traditional agile methodology, TryLinks was developed without continuous integration, which forces each commit of code to be built and deployed into production directly. With TryLinks, deployment was carried out after all essential features were completed. This decision was made because we wanted to focus more on the development of software, and not to be entangled with deployment difficulties until a minimum product was built. As it turned out, having all development locally saved a lot of time in the early cycles, and all important features were completed in time. Also there wasn't too much overhead during the deployment process in the end. In all, even though TryLinks was not developed exactly following agile methodology, the modifications were helpful in delivering the Minimal Viable Product.

4.5 Deployment

Many popular deployment solutions were discussed after a minimal product was completed, but a lot of them didn't suit our needs. Specifically, for TryLinks to function properly the hosting machine must have Links installed, and a lot of the candidates either didn't support it, or cost more than we would like. Eventually, after consultation with the Links development team at Edinburgh, Digital Ocean [7], a VPS (Virtual Private Server) solution, was chosen as the hosting platform. It offers a clean UNIX instance with SSH (Secure Shell) for Links to be installed, and costs only $5 per month. It also promises little down time or disruption.

For the Node server to be always running on the Digital Ocean instance, the *Forever* tool [8] was used. Also server log was streamed to a file for performance analysis and user study.

[7] https://www.digitalocean.com/

[8] https://www.npmjs.com/package/forever

Chapter 5

Detailed Design

This section describes the structure of TryLinks and the rationale underlying its implementation. First the data model of TryLinks is explained, as well as its properties and how TryLinks realized them. Then the backend of TryLinks is discussed. Implementation of the REPL shell and tutorial support are the major focus of this section. Lastly, the user interface design of TryLinks is reviewed. The relevant design and engineering decisions are discussed as well.

5.1 Database

5.1.1 Data Model

A *Data Model* [17] is an abstract structure of the data, and how this piece of data relate to other data. It serves as a conpect of representing the data relevant to the application. It is particularly important in software design, especially in the early phase, in that a stable data model will help establish compatiablity and consistency among the system compoennts [57]. In the discussion of TryLinks, Data Model refers more specifically to *Entity-Relationship Model* [8], as is usually used in relational databases.

In this section we first discuss the "Entities" in TryLinks. In the next section the "Relationship" part is elaborated. TryLinks adopted a straightforward approach with entities, having only two major entities: `LinksUser` and `LinksFile`. Table 5.1a and 5.1b shows the schema of these two entities.

`LinksUser` `LinksUser` contains the basic user profile for TryLinks, including their password. Note that here the password is simply a string of characters and does not

Column	Type
username	character varying
email	character varying
password	character varying
last tutorial	integer

(a) `LinksUser` table schema

Column	Type
data	character varying
tutorial id	character varying
username	character varying

(b) `LinksFile` table schema

Table 5.1: TryLinks data model

assume anything from it. It is the backend's responsibility to apply the necessary security processing. This is covered in Section 5.2.1.

Another point of interest in this table is the `last_tutorial` field. It records the last tutorial that the user visited, so that upon login, it can be queried and the user will be directed to the last visited tutorial. This was for consideration of fluent user experience.

`LinksFile` `LinksFile`'s structure is also simple. Each row refers to a tutorial for a user, and the source code is stored in the `data` field. The `username` here is crucial, since each file should belong to a fixed user. This is important to enforce the relationship between the user and their Links files conceptually, which will be covered in the next section.

5.1.2 Referential Integrity

Referential integrity [44] is a property of data which states references within it are valid. More specifically, referential integrity states that any column declared as a foreign key can contain either a null value, or a value from the a key column in the referenced table. In the context of TryLinks, there was only one foreign key: the `username` column in `LinksFile`, referencing the same column `LinksUser`. Conceptually, there are 2 cases when referential integrity needs to be maintained:

1. When a new row is added in `LinksUser`, a certain number of rows should be added in `LinksFile`, all referencing to the new user.

2. When an existing row is removed in `LinksUser`, a certain number of rows in `LinksFile`, all referencing to that user, should be removed as well.

Fortunately both of these requirements could be satisfied with Postgres; the detail implementations are explained below respectively.

Listing 5.1: SQL Trigger to add Links files when a user is added

```
CREATE TRIGGER init_file_at_signup
    AFTER INSERT
    ON public."LinksUser"
    FOR EACH ROW
    EXECUTE PROCEDURE public.add_files();
```

Listing 5.2: Foreign key definition in `LinksFile`

```
ALTER TABLE public."LinksFile"
    ADD CONSTRAINT username FOREIGN KEY (username)
    REFERENCES public."LinksUser" (username) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE CASCADE;
```

New user added Handling adding Links files was implemented using *SQL Triggers* [59]. A trigger is a special kind of stored procedure that runs when the data in the specified table is modified in some specified way. In the context of TryLinks, a procedure that creates new rows in `LinksFile` was called upon a new row added in `LinksUser`. Code Snippet 5.1 shows the source code of the trigger, in Postgres syntax. Note that the trigger itself does not do anything, but simply calls the `add_files` function, which adds the relevant Links files. The body of this function is omitted for brevity and clarity.

Existing user removed Removing the files when a user is deleted could also be done with SQL triggers, but there was a simpler way to achieve this. When constructing the foreign key constraint on `LinksFile`, a `ON DELETE CASCADE` option could be included, which performed the desired operations perfectly. `ON DELETE CASCADE` means when the referred row is deleted, its effect should "cascade", and delete the rows referring to said row as well. Code Snippet 5.2 shows the definition of the foreign key constraint, with the important configuration.

5.2 Backend

In this section the backend of TryLinks, where most of the important functionalities were realized, is reviewed. Table 5.2 lists all API calls supported. As mention before, the backend was implemented in NodeJS; more specifically, the *Express* [19] framework was used to provide the basic structure of TryLinks backend. It was chosen due to its popularity, and by extension complete documentation and vast help from the community. Express is also very simple to set up, especially in the case of TryLinks because it only needed to expose an API and to serve static content.

HTTP method	url	description
`POST`	`/api/user/signup`	Sign up for new user.
`POST`	`/api/user/login`	User Login. This also allows additional authentication required APIs to be called.
`POST`	`/api/user/update`	Authentication required. Updates user info.
`POST`	`/api/file/read`	Authentication required. Retrieves a tutorial source program.
`POST`	`/api/file/write`	Authentication required. Updates a tutorial source program.
`GET`	`/api/initInteractive`	Authentication required. Initiates the REPL shell.
`GET`	`/api/compile`	Authentication required. Compiles a Links program and serve to user.
`GET`	`/api/logout`	Authentication required. Logs out the current user.

Table 5.2: API lists for TryLinks

Figure 5.1 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

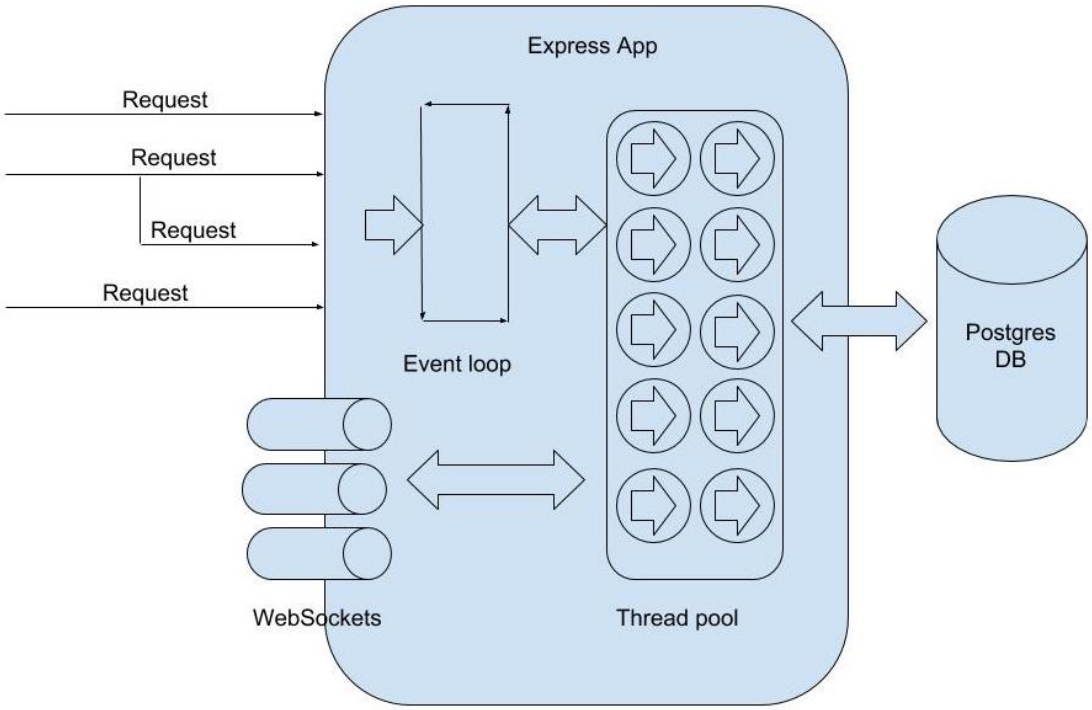


Figure 5.1: TryLinks backend structure based on the *Express* framework

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's different for TryLinks was the interactions with the threads in the thread pool. Apart from the usual interactions with the main event loop and with the database, the individual threads could also create, close, and interact with WebSockets, which can be connected by a client. More detail on WebSocket is given in Section 5.2.1.

In addition to express, TryLinks also leveraged the advantage of NPM, and made use of many third party packages and libraries. The major packages used, and their role in the implementation, are presented in the coming sections.

5.2.1 Technologies and tools

Password Processing Since security is one of the non-functional requirements of TryLinks, password processing became necessary naturally. Storing passwords in plain text is usually considered a great security risk. If an attacker managed to gain access to the database, or obtain a database dump, he/she would be able to obtain all passwords for all users, if the passwords were in plain text. To address this issue, traditionally a *Cryptographic Hash Function* is used. "A *Cryptographic Hash Function* is a function that takes input of random length and generates fix-length codes that are hard to guess" [26].

However, *Cryptographic Hash Function* along still did not provide enough security. It has been shown that some *Cryptographic Hash Functions* such as SHA1 can be reversed [1]. As a result, *salting* was introduced. Figure 5.2 illustrates the role of the salt. A salt is a piece of random data that acts as additional input to the *Cryptographic Hash Functions*. To view the password, one must also obtain the salt, which is hidden from the system and stored elsewhere. Both hashing with *Cryptographic Hash Functions* and salting were utilized in TryLinks.

Session Management When a user logs in once, usually a "web session" [48] is created for that user, which lasts a certain period. Within this web session, all subsequent HTTP requests from the same user are treated as already authenticated and do not need to be verified. This reduces unnecessary network traffic, and enhances user experience.

However, since HTTP is a stateless protocol, where each request is independent of all other requests, to achieve web session functionality a "session management" module must be introduced. This module acts as a link between the authentication and the authorization modules of the web application. Figure 5.3 shows how session management fits into Authentication and Access Control.

In the context of TryLinks, session management was necessary because most of the Links-specific APIs exposed required the user to be logged in, and authenticating for every request was redundant and too slow. With the help of session management, we could determine if a request should be served or not, simply by inspecting the session data. How session management was implemented in TryLinks is covered later.

Interaction with Database As shown in Figure 5.1, one of the communication endpoints the individual threads are responsible is the database. This was achieved, in the case of TryLinks, with SQL (Structured Query Language). One security concern with SQL is that usually it assumes full access to the database tables. Attachers

[1] https://sha1.gromweb.com/

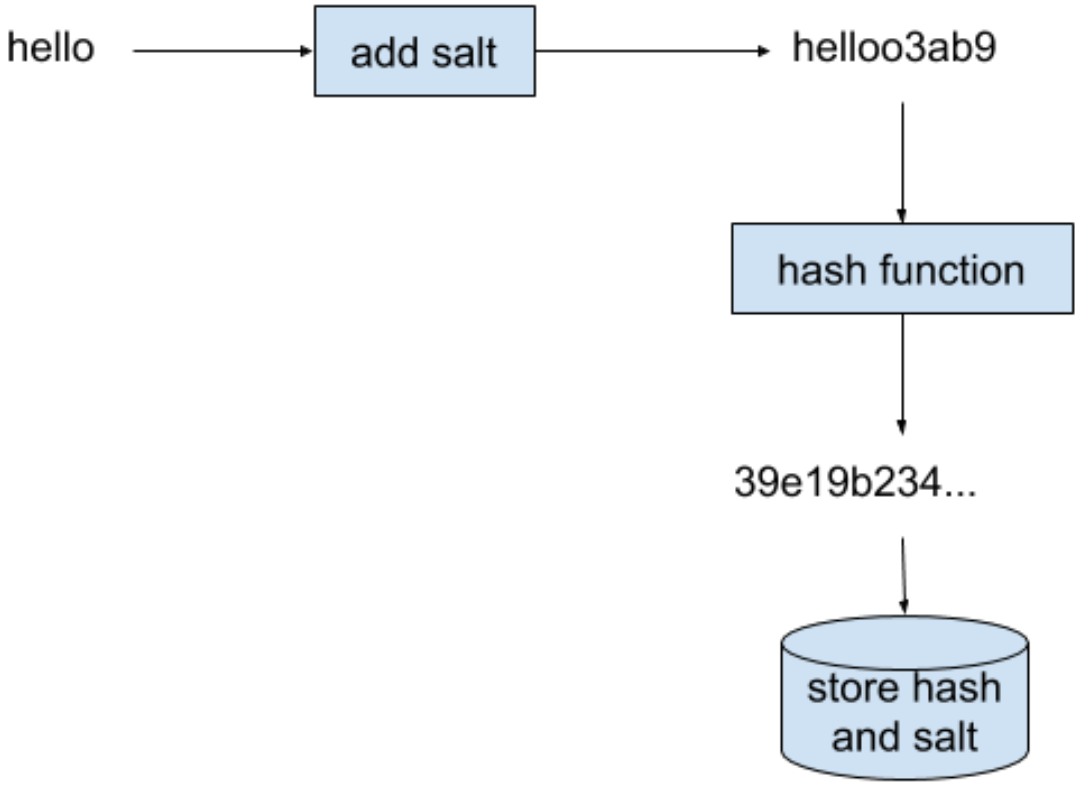
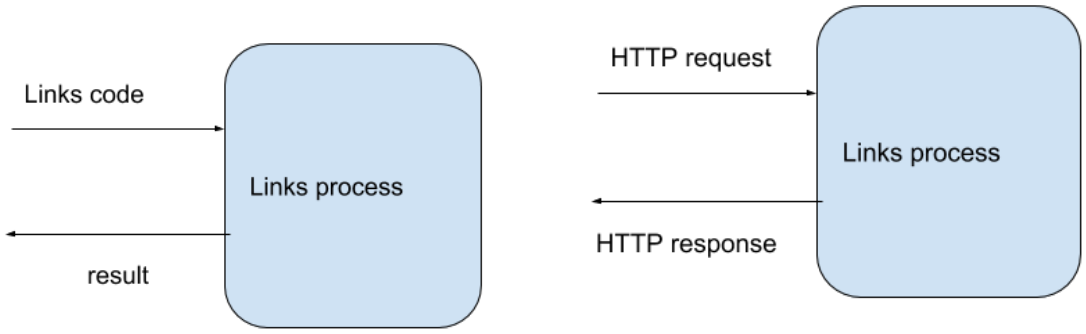


Figure 5.2: An example of salting, adopted from [33]

Figure 5.3: Session Management's role in web applications, from [48]

have been exploiting this vulnerability by adding custom, malicious raw SQL queries in requests and thus executing them in the database. This is known as "SQL injection" [22]. "SQL injection is one of top security concerns for web application",



(a) Links REPL process

(b) Links process that compiles a web application

Figure 5.4: Links process inputs and outputs locally

according to a 2012 survey [58].

To guard against SQL injection, TryLinks adopted the approach of checking and sanitizing input before executing. Typed parameters such as `Integer` and `Date` were parsed before execution, and untyped parameters such as plain string were escaped, in other words wrapped in special delimiters and treated as strictly strings instead of commands.

Child Process A child process usually refers to a process spawned by an existing user process, in our case the express application. This was crucial for TryLinks to implement any Links-specific functionality. Locally, a new Links process can be spawned by using the `linx` command from the command line. It can be treated as a block box, which takes the Links code as input and produces the output to the `stdout` port. In the case of compiling a Links web application, the process would take HTTP requests and send back corresponding responses. Figure 5.4 shows how the Links process works locally.

One alternative considered for implementing Links functionalities on the web was to extend Links to accept complex and dynamic input and compile source code upon change. However, this required changing the Links core library, which was out of scope of this project. Therefore, the child process approach was finally selected.

The built-in `child_process` Node module [9] was used to create and destroy Links processes. By spawning new Links process dynamically within the running Node application, and properly wiring the input and output ports, we could achieve

the REPL through a web interface. Also by spawning a new Links process, we could compile the tutorial source programs and serve it to the user.

Because spawning new processes dynamically is innately vulnerable to DoS (Denial of Service) attacks, where a large number of Links processes are spawned, causing the server to overload and eventually go offline, extra precautions were put in to prevent this from happening. Firstly, spawning new processes could only be achieved from a HTTP call, which checked for authentication first; secondly, we structured the code to allow only one Links process to be running for each user, thus eliminating the case where a malicious user endlessly spawns new processes.

WebSocket The traditional HTTP communication between the client and the server is, in a way, one 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" [56]. 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 5.5 shows how WebSocket facilitated the redirection of input and output, on both the client and server.

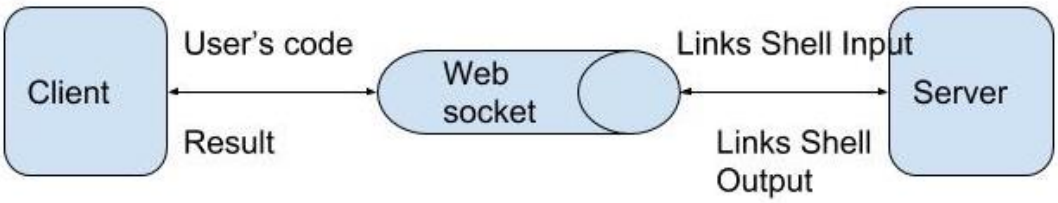


Figure 5.5: WebSocket's role in connecting client and server for Links process

There are many WebSocket modules available in NPM, *Socket.IO* [50] was used in TryLinks for a couple reasons. 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.

5.2.2 Implementation

This section describes the implementation detail of TryLinks backend, based on the tools and techniques discussed above.

User Logic Here each user logic API endpoint is presented. Most of the functionalities were similar to any regular web applications, so they are omitted for brevity.

Sign up A new row was added into `LinksUser` table, and the relevant Links files were added as well. In addition, each Links file was set to a starting template, corresponding to each tutorial.

Login This is a standard login function. Upon entering the correct password, the session data was set.

Update Fields in `LinksUser` was updated, usually the `last_tutorial` field.

File read Given a tutorial index, the stored Links file data for the user was read.

File write Given a tutorial index, the stored Links file data for the user was written to a new version.

Session Management Session management was implemented in TryLinks using the `express-session` module [2]. Code Snippet 5.3 shows how the session was configured. This was done when the application loaded for the first time. Note that a `secret` field must be given as the key to encode session data. This `secret` was also kept hidden to ensure security. Also, the `maxAge` field was set to a week to avoid unnecessary re-authentications.

When the user successfully logged in, the session data was set. In the case of TryLinks, the session data contained the `LinksUser` object. Code Snippet 5.4 shows how the session data was set in login procedure. When requesting access controlled resources, the session data was checked. If it was not set, the API would reject the request and simply respond with a `401` error. Code Snippet 5.5 shows the how the session data is checked in most sensitive API implementations.

[2] https://www.npmjs.com/package/express-sessions

Listing 5.3: `express-session` setup in TryLinks backend

```
app.use(session({
  secret: secret.secret,
  saveUninitialized: true,
  resave: false,
  cookie: {maxAge: 604800000}
}))
```

Listing 5.4: Setting session data during successful login

```
if (bcrypt.compareSync(password, user.password)) {
  req.session.user = user
  res.status(200)
    .json({
      status: 'success',
      message: 'Login successful',
      data: user
    })
}
```

Listing 5.5: Checking session data in sensitive APIs

```
if (!req.session.user) {
  res.status(401)
    .json({
      status: 'error',
      message: 'No authentication. Make sure you have logged
         in'
    })
  return
}
```

REPL Implementation Using the Node `child_process` module and WebSocket, the Links REPL shell was almost trivially implemented. Figure 5.6 shows the basic pipeline of the REPL shell functionality. First of all, the server created a new WebSocket channel, with the username as namespace, when the `api/initInteractive` HTTP endpoint was called, after checking for valid session. In addition, the `api/initInteractive` HTTP endpoint also spawned a new Links child process, and redirected its input and output port to the WebSocket. Finally it responded with the namespace for the client to connected to.

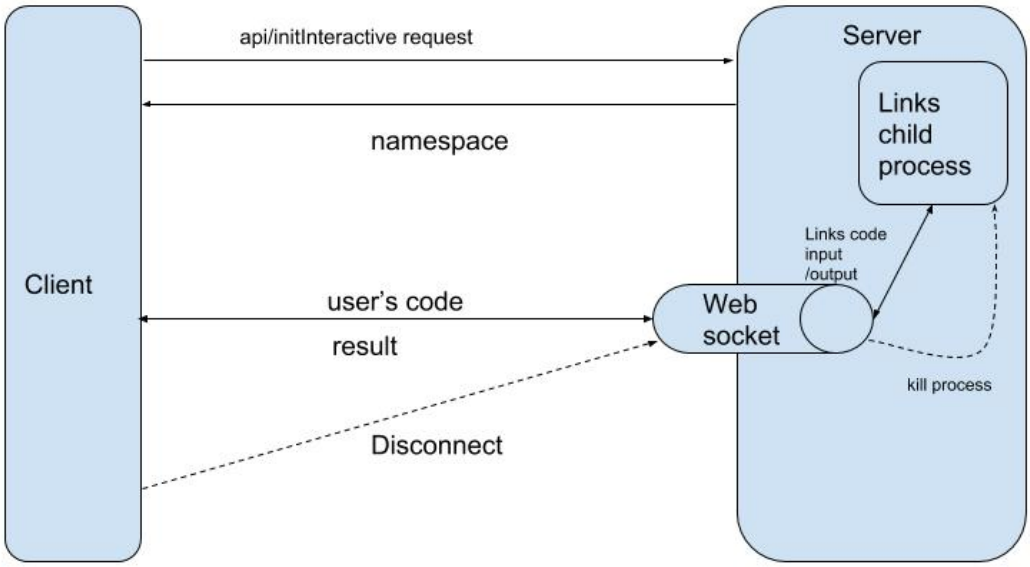


Figure 5.6: TryLinks REPL implementation pipeline

The client then connected to the correct WebSocket. Upon successful connection a welcome message from Links could be read and displayed, which informed the user that the REPL was ready. User inputs were sanitized according to Links syntax [30] to ensure security. In a local Links shell, one could adjust various configuration by using the `@set` command. In TryLinks REPL shell this was disabled, and a special

error message was shown when `@set` was encountered.

When the client disconnected from the WebSocket, usually by leaving the page, a built-in `disconnect` event was sent to the server. The server then disconnected the Links child process from the WebSocket, and sent 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 `child_process` module and WebSocket. Figure 5.7 shows the implementation pipeline of this process.

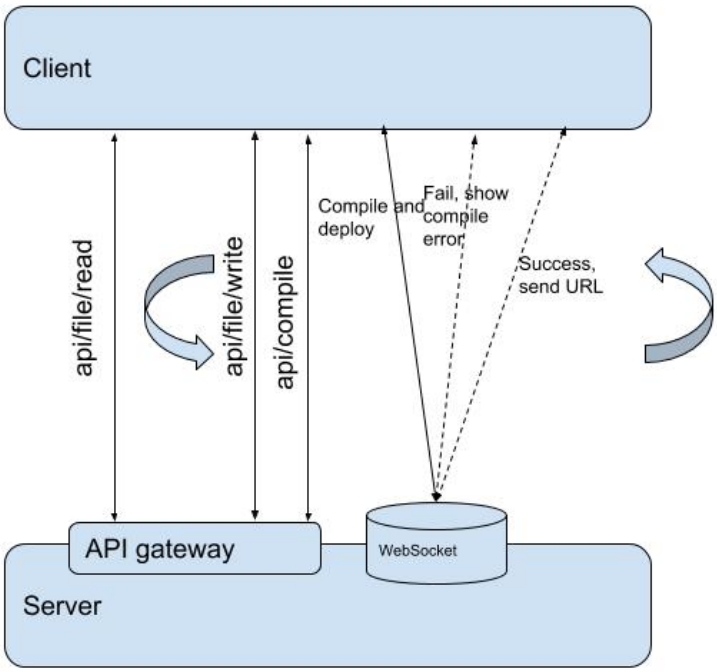


Figure 5.7: TryLinks Links program compilation and deployment pipeline

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 the pipeline loops in itself. The use of WebSocket was to catch compile errors, and enable re-compile functionalities.

5.3 Frontend

5.3.1 Overview

The TryLinks frontend was designed as a simple MPA (Multi-Page Application), with 3 main pages: *dashboard*, *interactive shell*, and *tutorial* page. Figure 5.8 shows the website navigation map. 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 [36] components were used extensively across the frontend to enhance user experience.

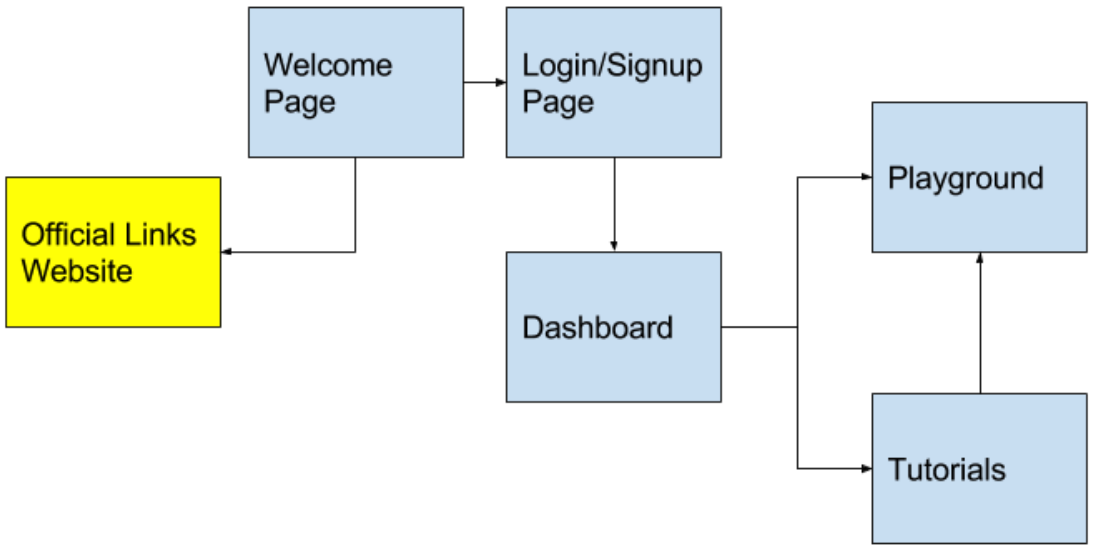


Figure 5.8: Site Map of TryLinks frontend

5.3.2 Angular Dart

Angular Dart [1] is a component based web development framework, loosely following the MVC (Model-View-Controller) pattern. Figure 5.9 illustrates the general architecture of Angular Dart.

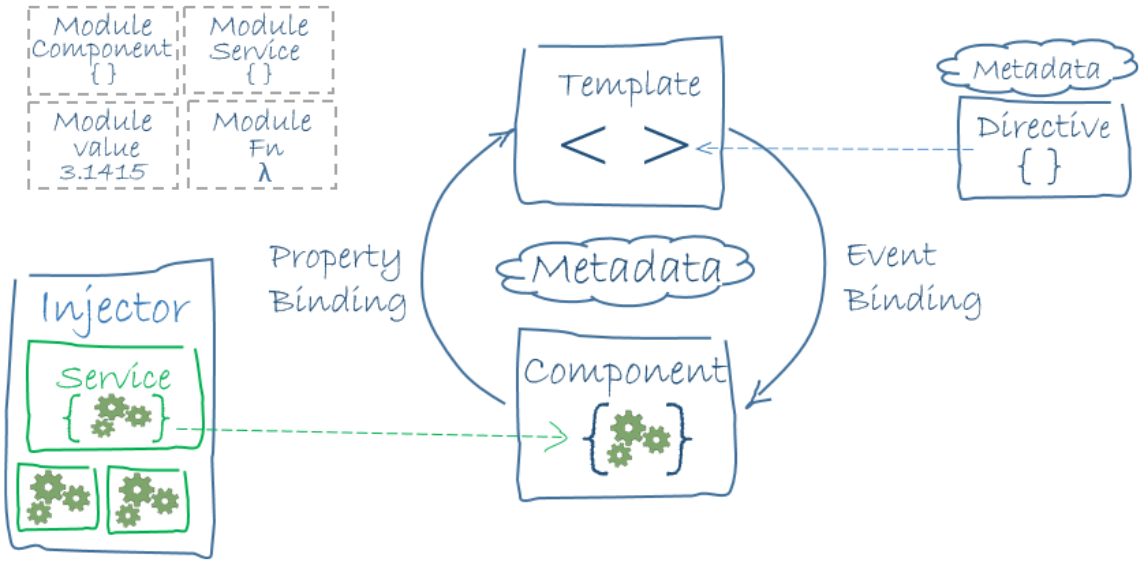


Figure 5.9: Standard architecture of Angular Dart applications, from [1]

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 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" [46] and to "follow the dependency inversion and single responsibility principles" [45].

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 5.10 shows the architecture

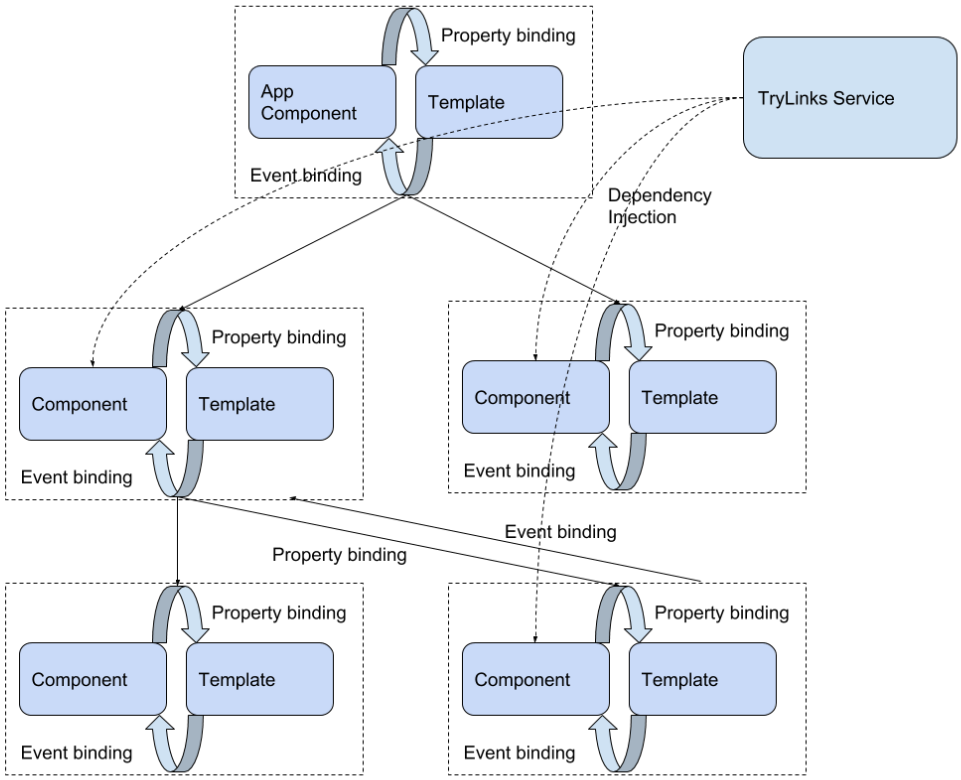


Figure 5.10: TryLinks frontend architecture

of the frontend of TryLinks, slightly tweaked from the standard architecture.

In the coming sections each page of TryLinks frontend is presented. The user interface design choices are explained, as well as the relevant implementation detail. Note that the frontend has been through a few iterations, and only the final iteration is covered here.

5.3.3 Landing Page

Figure 5.11 shows a screenshot of the landing page. The user interface adopted a minimalist design, only showing the essential parts. The Links logo was made intentionally big to be memorable, and the "TryLinks" name was shown on the upper left corner, which is the conventional location for application name. A short description of the Links programming language and the purpose of TryLinks were

given, and 2 animated demonstrations of the main functionalities of TryLinks were previewed. Lastly, a large "Call-to-Action" button was displayed for the user to "enter" the TryLinks application.

Figure 5.11: Screenshot of the landing page of TryLinks

The Landing page is the first page rendered when accessed. The most important role of a landing page is to present the application, and engage users for more interaction. This drove the choice of including demonstrations of the application, and a clearly marked button to invite potential users in.

5.3.4 Signup/login Page

Upon clicking the button on the landing page, the signup/login page is shown. Figure 5.12 shows a screenshot of the signup/login page. A `material-tab-panel` was used to implement the switchable signup/login option. This was so to minimize navigation, as the user would feel more consistent if they do not have to leave the current page. Figure 5.13a and 5.13b show the 2 tabs respectively.

In the spirit of minimizing navigation, as well as maintaining flow, when the user successfully signed up for TryLinks, an overlay dialogue was shown on the same page, which informed the user the signup had been completed, and that they could log in

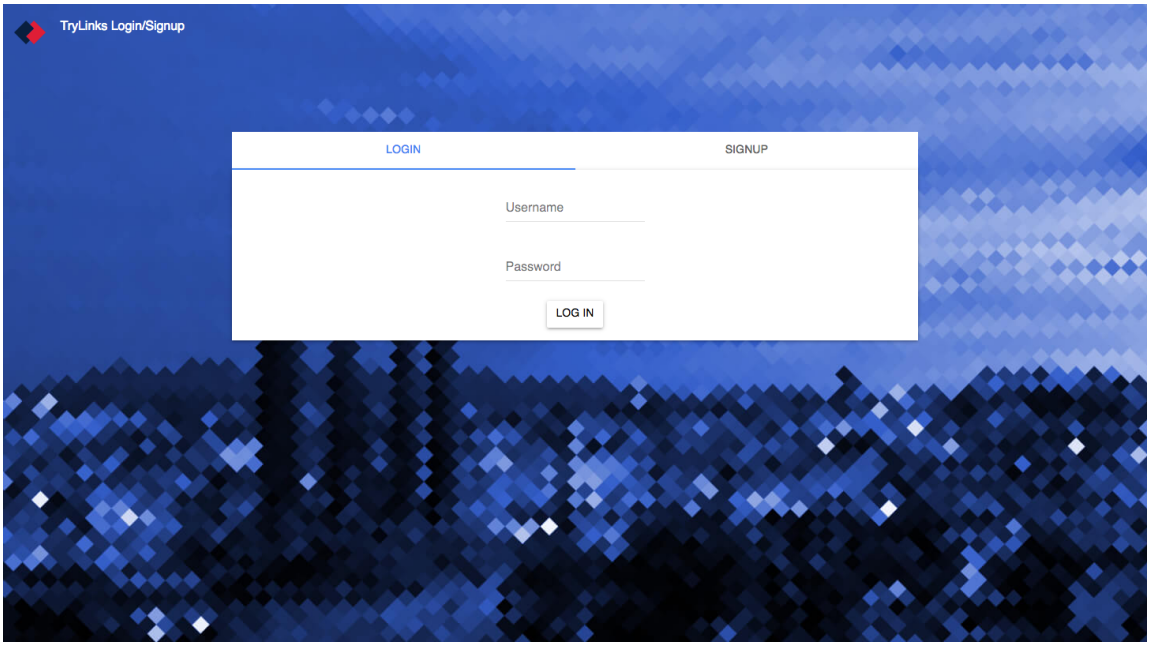


Figure 5.12: Screenshot of the signup/login page of TryLinks

to TryLinks. Figure 5.13c shows a screenshot of this dialogue. With this dialogue the signup process felt more conversational, and the next step for the user was clearly given as well.

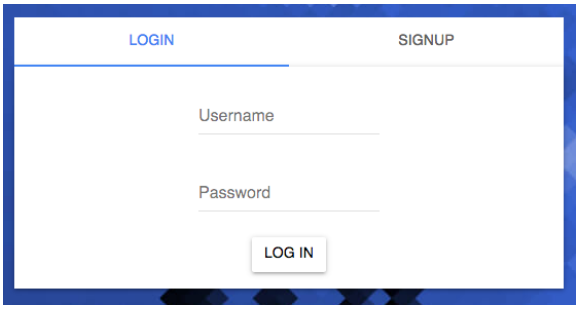
5.3.5 Dashboard

When the user logs in, they are greeted by the dashboard page, which acts as a homepage for navigation. Figure 5.14 shows a screenshot of the dashboard page.

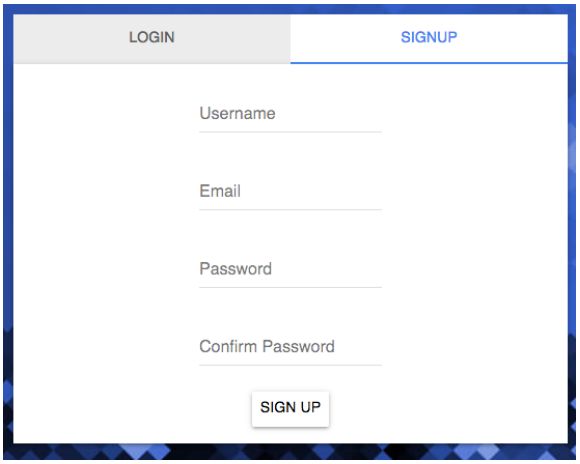
Since TryLinks is a relatively small and simple application, it was possible, and optimal, to establish the main functionalities of the application. This was reflected on the design of the dashboard page. Apart from using the minimalist design principle, the page was made clear to show that there were 2 major functionalities, and what those functionalities were. This helped to establish the core content very quickly, and to direct new user to relevant pages right away.

5.3.6 REPL

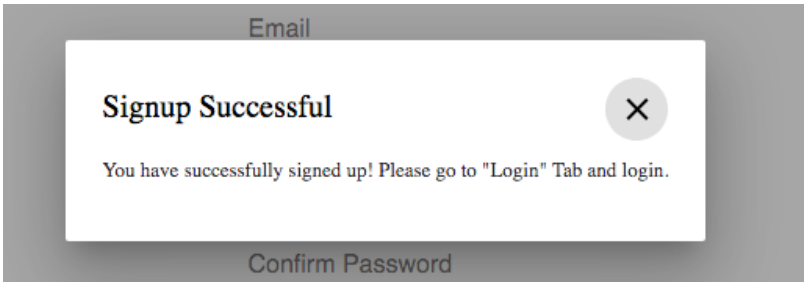
Figure 5.15 shows a screenshot of the interactive REPL shell page. First a short description of page was given, along with the a hyperlink to the official Links syntax



(a) Screenshot of the login tab on signup/login page



(b) Screenshot of the signup tab on signup/login page



(c) Screenshot of the signup successful dialogue on signup/login page

Figure 5.13: Detail components of the signup/login page

Figure 5.14: Screenshot of the dashboard page of TryLinks

documentation. The bulk of the page was a shell-like window, where the user could interact with Links. To give the closest experience as using a local Links shell, the REPL shell was designed as close to a basic terminal as possible, with a few helpful extensions such as automatic scrolling and multi-line input.

First of all, the REPL shell adopted a rich and meaningful colour scheme, to help the user quickly distinguish each type of text. The user's input was marked in `white`, the output of the Links shell was shown in `greenyellow`, a lighter green colour, and the error messages were shown in `red`. This helped the user to quickly establish what was entered, and the kind of the outcome. In addition, colouring had also been utilized in the introduction guide. This is discussed with the rest of the introduction guide.

Similar to *TryHaskell* (discussed in Section 2.3.1), a short introduction guide was added for the user to get a taste of the Links language quickly. This guide was written based on examples from [30]. The guide was shown in `grey`, with the relevant Links code snippets highlighted in `green` to assist identification. Different from the interactive shell in *TryHaskell*, the user must enter manual navigation commands, such as `next tip;` and `go back;`. This choice was made since no evaluation system was built into the TryLinks REPL shell, due to consideration of complexity. In the future it could be added into the REPL shell.

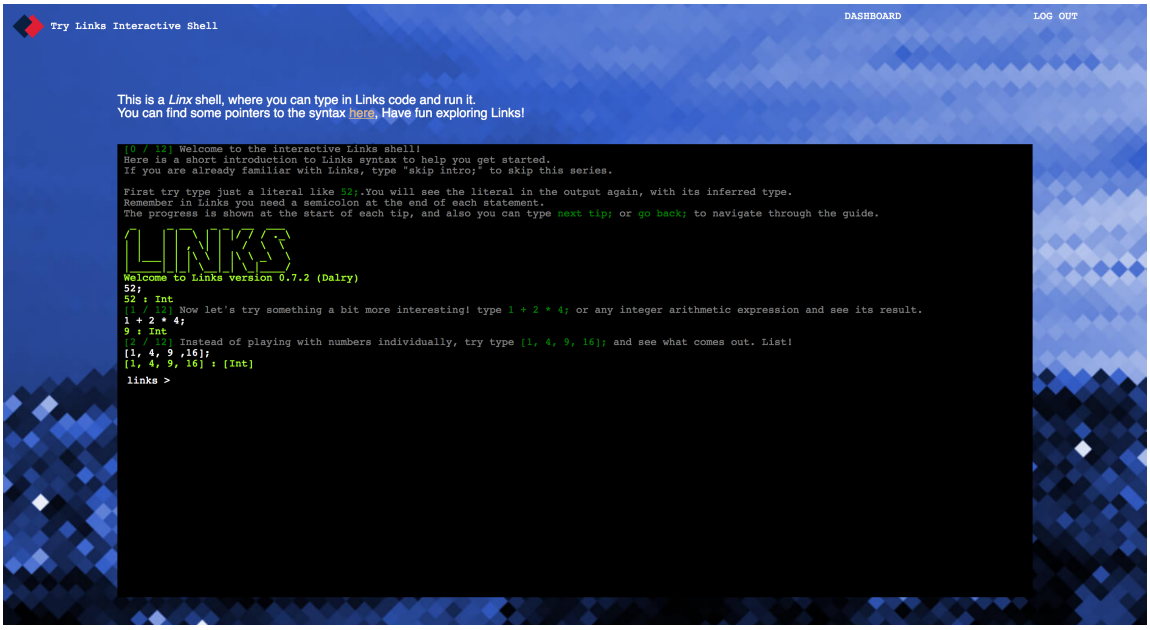
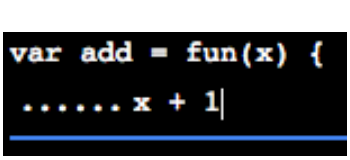


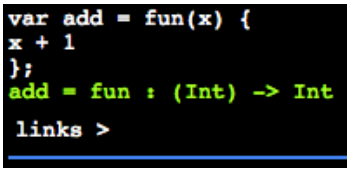
Figure 5.15: Screenshot of the REPL page of TryLinks

Implementation detail Given some Links code snippets in the introduction guide were multi-line, and it is semantically sensible to write long snippets sometimes, it was necessary for the input field to be able to accept multi-line code snippets. When the user hit the enter key but had not yet completed the current statement, which was indicated by the lack of ; symbol, the leading prompt changed to indicate more input was needed, and as the user entered more content, the statement was incrementally constructed behind the scene. When the ; symbol was finally entered, the complete statement was built and then sent to the WebSocket. Figure 5.16 shows an example of entering a multi-line statement, in this case a function, and using the function immediately afterwards.

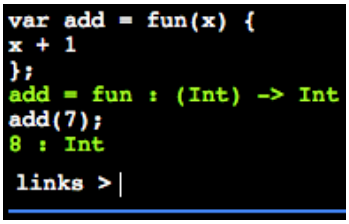
In addition of multi-line statements, TryLinks REPL shell also supported multi-statement input, such as the example shown in Figure 5.17. This was requested in the alpha test, that it was very convenient to be able to define multiple variables in one line. Locally the Links shell does not accept multiple statements in one input; only the first statement is evaluated and the rest is discarded. Therefore, extra logic was added to support this feature. When a line was entered, it was split by the ; symbol into a list of statements. Then each statement was separately sent to the WebSocket for evaluation. Figure shows an example of this feature in action. One caveat of this feature is that it only worked with a single line, and if both multi-line



(a) In the middle of inputting multi-line function



(b) Finished inputting multi-line function



(c) using the function

Figure 5.16: Example of inputting a multi-line function in TryLinks REPL shell

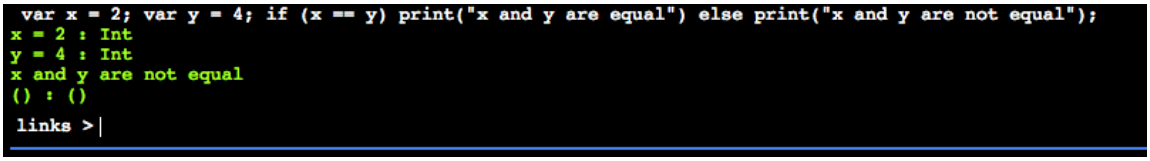


Figure 5.17: Example of inputting a multi-statement code snippet in TryLinks REPL shell

and multi-statement are present in the input, the shell does not function correctly. However, this irregularity is tolerable in that it is rare to input such a complex snippet at one time.

5.3.7 Tutorials

The tutorial page is likely to be where user stays for the longest, therefore it was crucial to design good presentation and interactions. Figure 5.18 shows the one of the tutorial pages. Overall TryLinks adopted the 3-panel tutorial page layout from Codecademy (discussed in Section 2.3.2), with a description panel, an editor panel and a result panel. Navigation to other tutorial pages were also available by clicking the menu button on the upper left corner. The navigation panel was lazily constructed, following Lazy Evaluation principle [29]. Essentially the navigational panel was not computed and rendered during the initial page load, but only when the user requested to navigate to other tutorial pages. This saved time in the initial page load and improved user experience. More information about Lazy Evaluation can be found in [43] and [55].

The content of the tutorial series was written based on [31], but with more descriptive steps that introduce the concept and outline the task for the user. This helped to make the learning experience more interactive. The description panel used a rich text render engine named Markdown [35]. It not only supported various

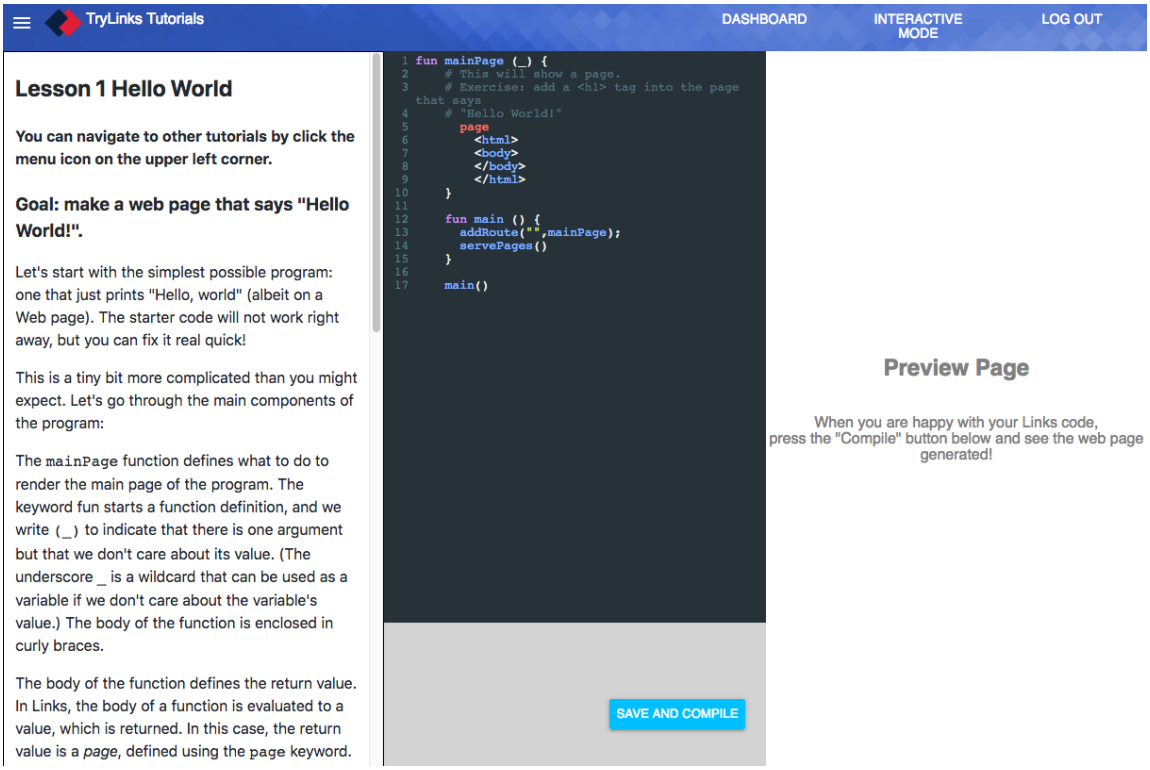


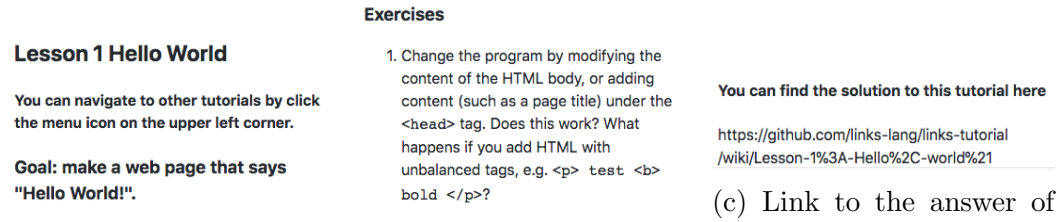
Figure 5.18: Screenshot of the tutorial page of TryLinks

rich-text functionalities like headings and enumerations, but also highlighted code snippets, which was especially helpful in a programming tutorial description.

Context-wise, a reminder of the navigation panel was given and the start of the tutorial, as well as the overall goal of said tutorial. Since there was no evaluation system built into TryLinks, the responsibility of determining if a tutorial is completed was put to the user, hence the goal must be clearly stated for each tutorial. At the end of each tutorial, a series of questions were put for the user to test if the material is understood. Also a hyperlink to the answers were also given. Figure 5.19 shows the aforementioned parts of the tutorial description.

The editor panel of the tutorial page was split into the actual editor and a panel for error messages. The starting code included comments that remind the user of tasks to complete, and the editor supported syntax highlighting custom made for the Links programming language.

The code editor used in the tutorial page was implemented using the *CodeMirror* library [13] [12]. It was selected because it was used previously in LODE (Links

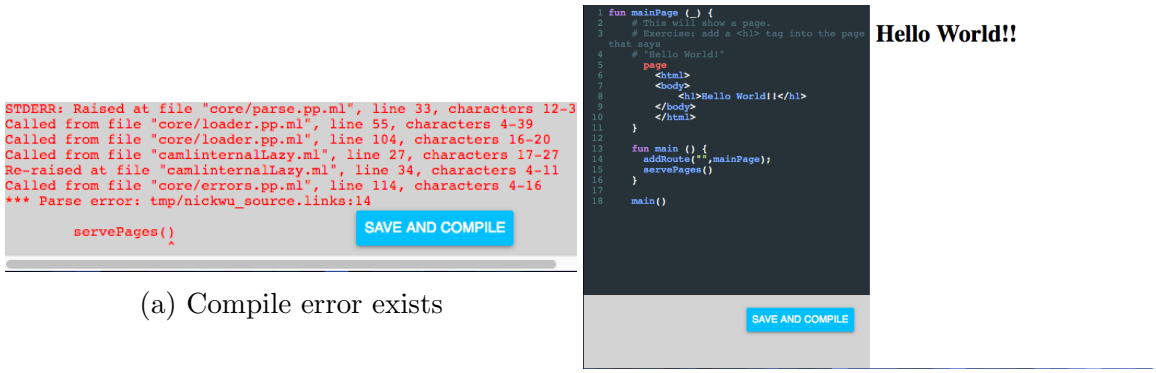


(a) The header of the tutorial

(b) Example of thinking question at the end of tutorial

(c) Link to the answer of the thinking questions

Figure 5.19: Parts of the tutorial description on TryLinks tutorial page



(a) Compile error exists

(b) Compilation successful

Figure 5.20: 2 scenarios of compiling the source code on the tutorial page

Online Development Environment) [4]. *CodeMirror* does not support Links natively, so a custom syntax guide, adopted from [4], was written and added as an extension to *CodeMirror*.

When the "Save and Deploy" button was clicked, the source code was compiled. A new "compile" message was sent through the WebSocket. If there was compile error, the WebSocket would receive an "compile error" message, along with the detail of the error. The error was then shown in the section below the editor. Conversely, when the compilation was successful, a "compiled" message would be received, along with the port where the compiled page was served. The rendered page was subsequently shown on the preview page panel. Figure 5.20 shows examples for both scenarios when the user clicks the "Save and Deploy" button.

5.3.8 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. One notable development technique utilized was the use of environment variable. The service base URL is initiated like this:

```
// trylinks_service.dart
static final serverURL =
      const String.fromEnvironment('Service', defaultValue:
         'http://localhost');
```

As shown the URL is generated by querying the environment. 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. More specifically, during the development process, this `Service` environment could be set by calling `pub serve -DService=http://localhost`. On the other hand, when the application was built and deployed, the URL could changed by calling `pub build -DService=production_server_address`.

5.4 Deployment optimization

Over the last development cycle, several optimizations were applied for production. In this section a few main techniques were presented.

CORS control "CORS (Cross-Origin Resource Sharing) is a mechanism that uses additional HTTP headers to let a user agent gain permission to access selected resources from a server on a different origin (domain) than the site currently in use" [15]. Since the server was run on a VPS which can only be access by IP address, it was more desirable to bind a meaningful domain to it. Therefore CORS was used in TryLinks to allow requests from that domain. In our case, it was a domain owned by the author [3]. After configuring CORS, all APIs were available from the production domain.

Speeding up landing page load Since the landing page is the first page seen by the user when they access TryLinks, it must be rendered quickly to maintain attention. The server achieved this by leveraging browser cache. All content requests

[3] http://devpractical.com:5000

were modified to include a `Cache-Control: public, max-age=604800000` HTTP header which allowed the browser to cache assets such as images and stylesheets. Other techniques such as image compression and asynchronous Javascript loading were also used to minimize the page load size of TryLinks landing page.

After various optimizations, the page size of TryLinks landing page decreased from 2.4 MB to just 1 MB, and with it a jump in loading time was observed. This is covered more in detail in the evaluation section.

Chapter 6

Evaluation

This section describes the various evaluations of TryLinks. First of all, the software's performance benchmarks are recorded and discussed. Then the user study conducted is described and the results are analysed. Lastly, a few known bugs and common complaints of TryLinks are raised.

6.1 Performance benchmarking

Performance benchmarking usually refers to comparing a particular software system to others, based on a established set of metrics. For TryLinks, there were two main metrics evaluated:

1. Initial page loading time of landing page.

2. Times for each HTTP request.

The first metric concerns user experience. Many researches have shown that page loading time affects user experience greatly [52] [3]. Users expect fast loading of the web page, and many of them will turn away if they have to wait 3 seconds for the initial load. In our study, the initial loading time of landing page of TryLinks was compared to those mentioned in Section 2.3, as well as a few other popular sites.

Apart from loading the web page, users also favour the web page to be responsive, which was the purpose of the second item in benchmarking. Since most of the TryLinks functionalities were delivered by HTTP calls, their processing time should be recorded for reference and for benchmarks for future improvements.

6.1.1 Tools

Pingdom [40] is a web service that monitors uptime and measures performance of websites. It uses real Google Chrome instances around the world for testing to provide realistic user experience measurements. This was used to record the page load time of TryLinks and other websites.

For HTTP call times, NPM offered a package called "morgan" [1] to log processing time for each HTTP request, and stream it to a file. About 3000 log entries were gathered during the evaluation sessions, and some extra processing was taken so that the average response time for each HTTP call was computed.

6.1.2 Findings

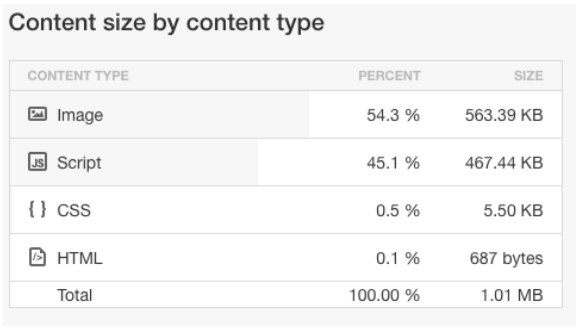
Figure 6.1 shows the result of Pingdom test of TryLinks and comparing websites. All tested websites load under 3 seconds, which is consistent to the research mentioned earlier. As shown in the graph, TryLinks has the second fastest loading among all tested sites. In comparison, TryLinks has relatively more assets than websites such as TryHaskell, but manages to achieve a faster loading time. This is due the various optimization techniques outlined in Section 5.4.

Website	Google PageSpeed Grade	Page Size	Load Time
TryLinks	A - 94	1.0 MB	1.06 s
TryHaskell	A - 90	125.7 kB	824 ms
repl.it	B - 89	3.5 MB	2.08 s
W3School	B - 86	453.8 kB	1.09 s
Codecademy	C - 78	2.4 MB	2.60 s
JsFiddle	C - 76	646.7 kB	1.70 s
Python official online shell	C - 74	727.3 kB	2.96 s

Figure 6.1: Pingdom Test result of TryLinks and other popular websites

In terms of page size, TryLinks ranks number 5 out of the 7 websites tested. Figure 6.2 shows the breakdown of the contents loaded for TryLinks. Images occupy more than half of the contents loaded in size, and among them the 2 demonstration animated GIF files take up the majority. In the original design of TryLinks frontend, these 2 images were not included out of consideration of simplicity. If they were to be removed, then TryLinks would have a much smaller page size. Even though they are added in the final design, it is obvious that they do not slow down the page

[1] https://www.npmjs.com/package/morgan



(a) Each type of content of respective size

(b) Detailed breakdown of the images loaded

Figure 6.2: Loaded contents of TryLinks landing page

loading time dramatically. Also they bring upon value on user experience, so overall this decision is still worthwhile.

In addition to testing page size and loading time, Pingdom also offers Google PageSpeed performance grade [34], which rates the website by a numeric score and a grade from A to F. This offers an overall evaluation based on many factors such as *Speed Score* and *Optimization Score*, and also gives helpful optimization suggestions. By this grade TryLinks achieves the highest score among all tested websites. This is so because a lot of suggested optimizations were taken during productionization.

Last but not least, Pingdom also offers a continuous monitoring service, which tries to connect to the website every 5 minutes and records the loading time. Figure 6.3 shows the report of the last 30 days as of this writing. Overall, TryLinks achieves a satisfactory level of availability. Also, the 4 outages shown were not due to system malfunction, but rather because new source codes with bug fixes were pushed to the server. In addition, on average the loading time for the landing page was actually a lot shorter than measured manually, suggesting TryLinks may have a better loading performance than previously discussed.

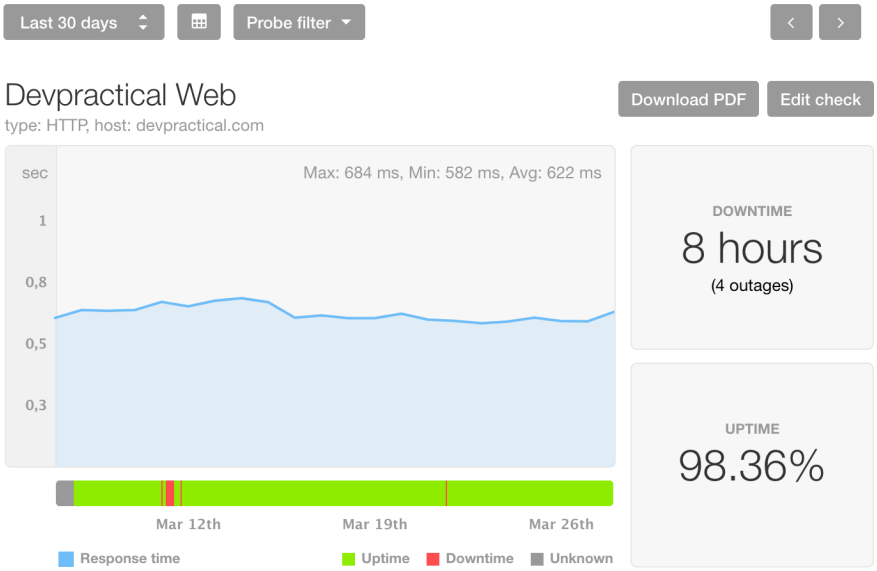


Figure 6.3: Pingdom monitoring report of TryLinks for 30 days

6.2 User Study

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 each finding is reviewed and its reasoning discussed.

6.2.1 Methodology

The TryLinks evaluation session was designed as the following. 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*, which is attached in Appendix A. In it some more detailed description of TryLinks was also given to aid the user.

In terms of evaluation group demographics, 9 people with 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 6.3.

6.2.2 Pre-evaluation Findings

Pre-existing knowledge of functional programming Figure 6.4 shows how much the participants know about functional programming beforehand. Since Links is a functional programming language, it is important to establish existing functional programming knowledge before using TryLinks. Among the tested group, most have at least some experience to functional programming. In theory, this would help them in learning Links, as a lot of the programming concepts are transferable.

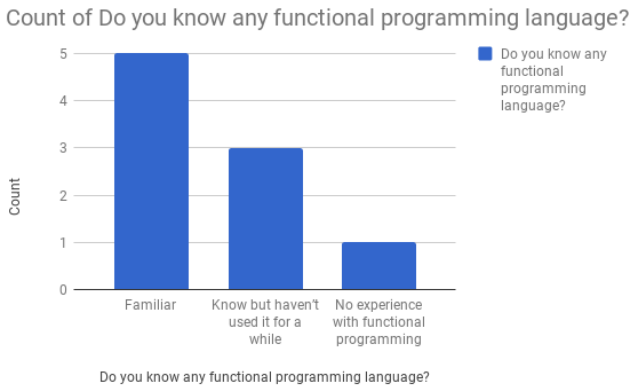


Figure 6.4: Response of previous functional programming experience

Pre-existing knowledge of web development Figure 6.5 shows the how much the evaluation knows about web development beforehand. This is made since Links is mostly used for web programming. Different from the last question, it is shown that although over half of the tested people have web programming experience, there are also a significant amount with little or no experience in web development. This would theoretically make learning Links, especially the tutorials harder, which is confirmed later as well.

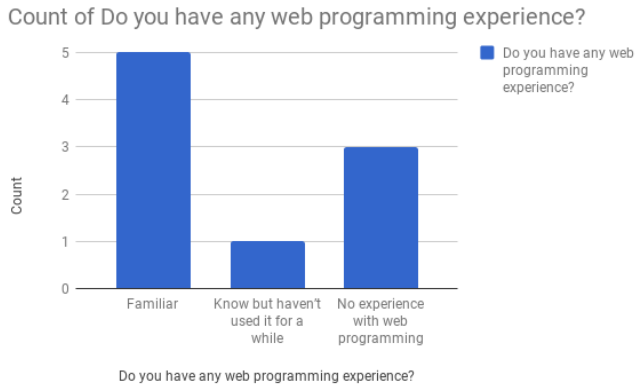


Figure 6.5: Response of previous web development experience

6.2.3 Post-evaluation Findings

Helping to learning Links language After the hour-long session, the first question asked is how hard is Links to learn. This is intended to reflect on the Links language itself on its difficulty to pick up. Figure 6.6 shows the result of this question. Contrary to expectation, no participants find Links to be easy to learn, and the average is skewed towards hard to learn.

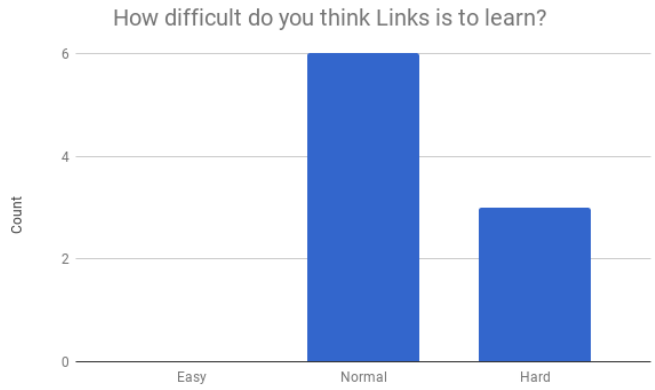


Figure 6.6: Response of difficulty of learning Links

This is so partly because of Links' somewhat unique syntax. Also the error messages emitted by the Links interpreter are sometimes hard to understand. It was suggested that it would be much more helpful if TryLinks offers a "cheatsheet"

of Links syntax readily available on all pages, which could help reinforce the user's memory of Links syntax.

In addition, the number of tutorials the participants finished during the hour is also recorded, as a proxy of the difficulty of learning to use Links for practical web programming. Figure 6.7 shows the responses of this question. Following up from the last point, no participants completed more than 4 tutorials under an hour, and most of them reported that it was hard to progress through the tutorials. This is so partly because the tutorials themselves were not constructed in the best way. One participant commented: "It is a bit abstract, between learning the language Links and the usage in web programming." The transition from using Links in the command line to using it for web programming is so sharp that some users lose the narrative.

However, this was actually a quite positive result, since before TryLinks it was impossible for non-experts to get Links running in an hour, let alone using it in any tutorial. In this regard, TryLinks massively shortened the installation time, and in turn lowered the difficulty of getting started with Links.

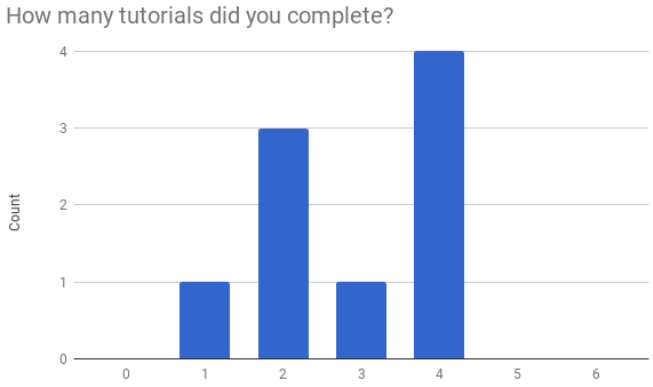
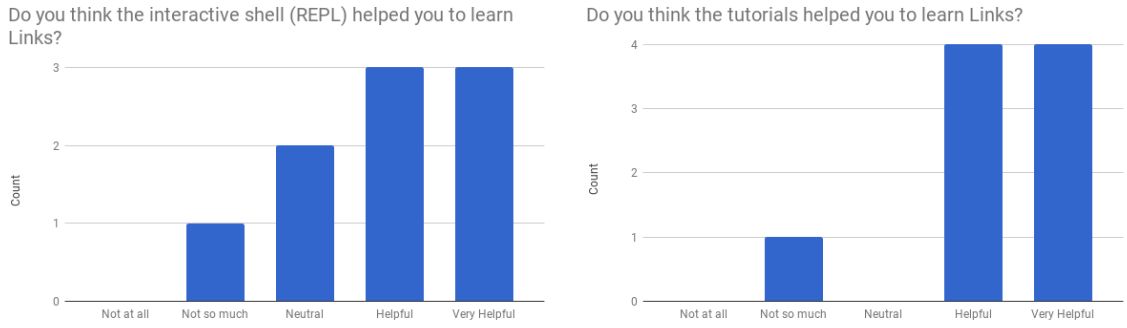


Figure 6.7: Response of how many tutorials the participants completed under an hour

In addition, many users reported that, while debugging the Links programs in tutorials, the error messages from the compiler were obscure and did not offer help in finding root of the error. This is perhaps a flaw of Links itself, that it should be able to give more helpful error messages.

Then the effectiveness of TryLinks, arguably the most important metric, is prompted. Figure 6.8 shows the response of how helpful the 2 main functionalities of TryLinks are in terms of learning Links. It is shown that both functionalities helped the par-



(a) The REPL shell

(b) The tutorials

Figure 6.8: Response of how helpful TryLinks is in learning Links

ticipants to learn Links overall. In comparison the tutorials are more effective in teaching the participants, partly because of their practicality and because the REPL shell was not yet polished for some of the participants due to scheduling clash. With current design of REPL and tutorials, it is expected that more users would find TryLinks helpful in learning Links.

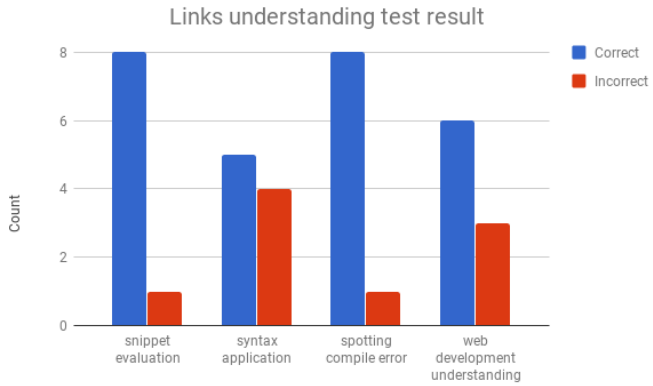


Figure 6.9: Response of Links understanding test

Tested Links language understanding After the session of using TryLinks, a series of questions were asked to test the participants' understanding of Links. Figure 6.9 shows the result of the test. Overall, most participants answered correctly to the tested questions. This in a way proves that TryLinks is effective in teach the

user about Links. Question 2 and 4 were designed to be more open ended, which partly explains why fewer participants answer these questions correctly, compared to question 1 and 3.

6.3 Known bugs and complaints

During the development of evaluation of TryLinks, a few bugs have been found. Some of them were fixed, but others are not in the final version of TryLinks either because the bugs originate from outside of TryLinks, or because of lack of time.

6.3.1 iframe rejection issue

On the tutorial page when the user clicks the button to compile and deploy the Links program, the render page would not load the page. After a manual refresh it would behave normally. This is due to the mismatch of events. Before the the Links process on the server starts, the port is already specified and sent to the client. The client loads the iframe, with no knowledge of the fact that the Links process requires some time to be booted up and functional. Fortunately, on Google Chrome the iframe is refreshed automatically every fixed interval, and on Firefox this can be fixed with a manual refresh on the iframe. It does hurt the user experience in that the user usually does not expect the rendered page to have a connection error.

After the evaluation session this was fixed. The `compiled` message was changed to delay for about 4 seconds before getting sent to the WebSocket. With this change the Links child process would be fully loaded and functional, by the time the `compiled` message is sent to the client. Therefore, the iframe "cannot connect" issue was fixed.

6.3.2 REPL XML rendering bug

In Links XML can be used as regular code, and this has caused the REPL shell to render HTML snippets if they are entered. For example, if `<b>foo</b>;` is entered as a command, a bold "foo" text will be shown in the output. This is caused by incomplete sanitization on the client side. Unfortunately because this bug is discovered rather late, it is not fixed as of this writing.

6.3.3 Links child process start-up time

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.

6.3.4 Lack of evaluation system

Another complaint about TryLinks is that it does not have a built-in evaluation system. This causes the user to have to manually navigate through the introduction guide on the REPL page. Also, during the tutorials it is sometimes not clear what the goal is. This decision was made out of consideration of complexity, but it turns out users expect a built-in evaluation system in place, so that they feel more certain that they have coded correctly. In the future this can be another feature to add.

Chapter 7

Conclusion

TryLinks is a complete 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 overall.

This project outlines a few functional and user interface design of online programming language learning platforms, and in-depth design and implementation techniques in building one. To summarize, the following web development practices are used throughout the development of TryLinks:

- Agile development methodology
- Version Control (Git)
- Independent and portable relational database (PostgresSQL)
- Modularized and extensible API Building (NodeJS, Express)
- Dependency management (NPM, Pub)
- Access Control and Session Management (Express-session)
- Real-time WebSockets (Socket.IO)
- Child process management (Child Process Module)
- A client-optimized language (Dart)
- Responsive Design (Material Design, CSS Grid)
- Client-side Code Editor (CodeMirror)
- Deployment optimization (Image Compression, CORS Control)

7.1 Obstacles encountered and overcome

Throughout the development process, several obstacles were encountered, and finally fixed. One of them was using WebSocket with child process, 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 booted up and functional. Furthermore, in the case of compiling Links program, 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 WebSocket 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, such as *CodeMirror* not accepting custom CSS rules, but rather had to be adjusted using its own API. In addition, coordinating these technologies presented some difficulty as well, for example the coordination among the rendering `iframe`, the compiler error section, and the WebSocket.

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. When I looked at TryLinks from a outsider's point of view, the system had a lot of dissatisfactions. Fortunately, most of these problems were fixed before the evaluations and the final version of TryLinks turned out to be much better.

7.2 Reflection

In all, most of the tools and techniques worked well, as expected. The modified agile development method carried some risks, since no functional product was put into production until a fairly function-complete product was built. There were some problems encountered during deployment, most noticeably the configuration of WebSocket took roughly 4 days to clear out. In retrospect, it might be worth setting up a continuous integration process, which would uncover the potential problem that would block deployment.

Another reflection was that when choosing third party libraries, it is better to

choose the ones with vast support, either from the author or the developer community. This was especially clear during the development of Links-specific backend with `child process`. There was a lot of confusion about the asynchronous aspect of NodeJS. Fortunately the community was extremely helpful with detailed tutorial and diagrams. On the other hand, the *Socket IO* Dart library was less maintained, and eventually caused a problem during production. Only after directing contacting the author and requesting a fix the problem was solved.

Last but not least, user experience should have been taken a lot more seriously during the development process. Being the developer of a particular software would sometimes blind one from noticing how hard to use it is. In the case of TryLinks, the first round of alpha test revealed many problems, not about the functionalities, but the usability of the software itself. After fixing the problems raised, TryLinks received a lot more positive feedback, as shown in the evaluation. If I were to implement TryLinks from scratch again, getting feedback at every round of development would have a lot more priority.

7.3 Final features

The final features of TryLinks include:

1. Full User Logic (signup, login, etc)
2. Interactive Links REPL shell
3. 6-part tutorial series of Links in web development

Referring back to the goal and the specifications of the project, a functioning software was delivered on schedule, with the designated functionalities. In this regard the project was successful. Furthermore, TryLinks received positive feedback overall during the evaluation. This was a big step from the previous state, where the installation process alone took more than an hour. Also, the evaluation confirmed the hypothesis that it is better to learn new programming languages in an interactive environment, compared to the old-fashioned textbook style.

7.4 Accessing the software

TryLinks is currently accessible here [1], hosted on the author's personal domain. The source code of TryLinks can be found here [60]. To extend the project, users can

[1] http://devpractical.com:5000/web

create a pull request. This is the process of proposing code changes into a repository. If approved by a "collaborator" [2], it would be merged into the codebase. People can also fork [3] the repository and start developing their own version of TryLinks.

7.5 Future work

7.5.1 Better Editor Support

One difference between the TryLinks editor on the tutorial page and a local editor is that it is missing many great editor features. For example, the TryLinks editor does not support auto-completion, on-the-fly static type checking, and customizable key bindings. Also the editor window itself, being the most important panel on the tutorial page, cannot be resized. In the future hopefully these can all be added into the TryLinks system.

7.5.2 Evaluation System

One of the greatest source of confusion reported in using TryLinks is that users do not always know that the end goals are, due to the lack of an evaluation system. This was purposely left out in the design on TryLinks due to complexity, but it can and should be integrated into TryLinks in future iterations.

7.5.3 Pipeline Refactor

The most complicated part of the implementation is the pipeline for REPL shell and the "compile and deploy " functionality on the tutorial page. It is also worryingly fragile and has caused many bugs throughout the construction. A future refactor of these two APIs would help eliminate the current bugs of TryLinks, and at the same time prevent new bugs from emerging.

7.5.4 Better Security

Security is one of the non-functional requirements of TryLinks. Although a fair amount of work has been done to ensure TryLinks is secure, there are many more

[2] A collaborator is a person with read and write access to a repository who has been invited to contribute by the repository owner [25]

[3] A fork is a copy of a repository. Forking a repository allows the ability to freely experiment with changes without affecting the original project [25]

upgrades in security that need to be added. For example, all TryLinks API are in HTTP instead of HTTPS, which is innately less secure. Also, in the tutorial series, all users are sharing one database, which is a concern of privacy. These can be addressed with more engineering on the infrastructure and more sophisticated database partition.

Also, a few security vulnerabilities are still present with the REPL shell. A user can potentially read arbitrary files from the sever, depending on the permission of the Links child process. Also a user could also potentially write programs that go into an infinite loop, and use up system resources. These can be addressed with more input sanitation, and setting a timeout for the REPL output. In other words, if a command take too long to process, the child process should be terminated and restarted.

7.5.5 Integration with Links

TryLinks is developed as an independent project from Links, and therefore not included in the official Links website. Some have suggested that TryLinks can be integrated into the Links official domain. This requires many collaborations and discussions and is clearly out of scope of this project. However, it is a promising direction for TryLinks.

Furthermore, if TryLinks were to be integrated into Links, it would be desirable to implement a "administrator" mode, where an administrator can add new tutorials into the tutorial series. Current this requires changing the source code and re-compile the entire application.

Another integration would be with LODE [4], and allowing users to create their own workspace to create and develop their custom Links applications, not just the ones in the tutorial series.

Appendix A

TryLinks Evaluation README

A.1 Introduction

Welcome to the TryLinks evaluation session, where you will try to learn about the Links language by using the TryLinks application.

Before we begin, please tell us about yourself over **here**.

Now launch TryLinks at http://devpractical.com:5000/web. You should see a welcome page. Click the Login/Sign up button and sign up with an account.

After you signed up you should be able to login with the username and password you chose.

A.2 REPL

Before we use Links for actual web programming, you must know the language syntax a bit. If you have experience with Haskell this should be relatively easy. Click the "REPL" button on the left of the homepage and you should see a Links shell. In this section we are going to learn some basics about Links language. In the REPL shell an short guide to Links syntax is included, feel free to take a look and try the given examples.

A.3 Tutorials

Now that you have learned the syntax of Links, let's jump into some actual coding! Launch the tutorials by clicking the "Tutorials" button on the right on homepage.

There are 6 tutorials in total, each harder than the last one. You can find the description on the left, the editor in the middle. The start code is given to you already, so you only need to change a few places here and there.

When you are happy with your code, press the "save and compile" button. If your code compiles correctly, you should be able to see the web page rendered on the right. If you see something like "page can't load", just press the "try again" button and it should work just fine. This is a known bug with iframe, which has little to do with the project.

If your code has compile error, the error will be shown below your editor and there won't be a rendered page. Change your code according to the error message, and press "save and compile" again. If your code is correct, the error should go away and you should see the rendered page.

At the end of each tutorial, there are a few further thinking exercises. Try to do them and feel free to ask the organizer for the answers if you are interested!

Don't worry if you don't finish all 6 tutorials in time. If stuck feel free to ask the organizer.

A.4 Finishing up

At the end of the evaluation, please give us some **feedback**. Thank you very much to make TryLinks better!

Bibliography

[1] *About AngularDart*. URL: https://webdev.dartlang.org/angular (visited on 03/16/2018).

[2] *Ajax*. URL: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX (visited on 03/16/2018).

[3] *Akamai Reveals 2 Seconds as the New Threshold of Acceptability for eCommerce Web Page Response Times*. URL: https://www.akamai.com/us/en/about/news/press/2009-press/akamai-reveals-2-seconds-as-the-new-threshold-of-acceptability-for-ecommerce-web-page-response-times.jsp (visited on 03/16/2018).

[4] Carl Andersson. "LODE: An Online IDE for Links in Links". In: *University of Edinburgh, Undergraduate Honours Project* (2008).

[5] *AngularDart*. URL: https://webdev.dartlang.org/angular/ (visited on 03/16/2018).

[6] Len Bass. *Software architecture in practice*. Pearson Education India, 2012.

[7] Alex Berson. *Client/server architecture (2. ed.)* McGraw Hill series on computer communications. McGraw-Hill, 1996. ISBN: 978-0-07-005664-0.

[8] Peter P. Chen. "The Entity-Relationship Model - Toward a Unified View of Data". In: *ACM Trans. Database Syst.* 1.1 (1976), pp. 9–36. DOI: 10.1145/320434.320440. URL: http://doi.acm.org/10.1145/320434.320440.

[9] *Child Process*. URL: https://nodejs.org/api/child_process.html (visited on 03/16/2018).

[10] Adam Chlipala. "Ur/Web: A Simple Model for Programming the Web". In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. 2015, pp. 153–165. DOI: 10.1145/2676726.2677004. URL: http://doi.acm.org/10.1145/2676726.2677004.

[11] E. F. Codd. "A Relational Model of Data for Large Shared Data Banks". In: *Commun. ACM* 13.6 (1970), pp. 377–387. DOI: 10.1145/362384.362685. URL: http://doi.acm.org/10.1145/362384.362685.

[12] *codemirror — Dart Package: A Dart wrapper around the CodeMirror text editor*. Apr. 2016. URL: https://pub.dartlang.org/packages/codemirror (visited on 03/16/2018).

[13] *CodeMirror: CodeMirror is a versatile text editor implemented in JavaScript for the browser. It is specialized for editing code, and comes with a number of language modes and addons that implement more advanced editing functionality.* URL: http://codemirror.net/ (visited on 03/16/2018).

[14] Ezra Cooper et al. "Links: Web Programming Without Tiers". In: *Formal Methods for Components and Objects*. Ed. by Frank S. de Boer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 266–296. ISBN: 978-3-540-74792-5.

[15] *Cross-Origin Resource Sharing (CORS)*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (visited on 03/16/2018).

[16] *Dart programming language*. URL: https://www.dartlang.org/ (visited on 03/16/2018).

[17] *Data model*. Feb. 2018. URL: https://en.wikipedia.org/wiki/Data_model (visited on 03/16/2018).

[18] *Elm: a delightful language for reliable webapps.* URL: http://elm-lang.org/ (visited on 03/16/2018).

[19] *Express - Node.js web application framework*. URL: https://expressjs.com/ (visited on 03/16/2018).

[20] Dror G. Feitelson, Eitan Frachtenberg, and Kent L. Beck. "Development and Deployment at Facebook". In: *IEEE Internet Computing* 17.4 (2013), pp. 8–17. DOI: 10.1109/MIC.2013.25. URL: https://doi.org/10.1109/MIC.2013.25.

[21] *Flask: web development, one drop at a time*. URL: http://flask.pocoo.org/ (visited on 03/16/2018).

[22] Jeff Forristal. *NT Web Technology Vulnerabilities*. URL: http://phrack.org/issues/54/8.html#article (visited on 03/16/2018).

[23] Node.js Foundation. *Node.js*. URL: https://nodejs.org/en/ (visited on 03/16/2018).

[24] Armando Fox and Eric A. Brewer. "Harvest, Yield and Scalable Tolerant Systems". In: *Proceedings of The Seventh Workshop on Hot Topics in Operating Systems, HotOS-VII, Rio Rico, Arizona, USA, March 28-30, 1999*. 1999, pp. 174–178. DOI: 10.1109/HOTOS.1999.798396. URL: https://doi.org/10.1109/HOTOS.1999.798396.

[25] *GitHub Glossary - User Documentation*. URL: https://help.github.com/articles/github-glossary/ (visited on 03/16/2018).

[26] Michael T. Goodrich and Roberto Tamassia. *Introduction to computer security*. Pearson, 2014.

[27] Katarina Grolinger et al. "Data management in cloud environments: NoSQL and NewSQL data stores". In: *J. Cloud Computing* 2 (2013), p. 22. DOI: 10.1186/2192-113X-2-22. URL: https://doi.org/10.1186/2192-113X-2-22.

[28] Theo Härder and Andreas Reuter. "Principles of Transaction-Oriented Database Recovery". In: *ACM Comput. Surv.* 15.4 (1983), pp. 287–317. DOI: 10.1145/289.291. URL: http://doi.acm.org/10.1145/289.291.

[29] Paul Hudak. "Conception, Evolution, and Application of Functional Programming Languages". In: *ACM Comput. Surv.* 21.3 (1989), pp. 359–411. DOI: 10.1145/72551.72554. URL: http://doi.acm.org/10.1145/72551.72554.

[30] *Links Syntax*. URL: http://links-lang.org/quick-help.html (visited on 03/16/2018).

[31] *links-tutorial: Tutorial examples for getting started with Links*. URL: https://github.com/links-lang/links-tutorial (visited on 03/22/2018).

[32] *Links wiki home*. URL: https://github.com/links-lang/links/wiki (visited on 03/22/2018).

[33] Quick Heal Technologies Ltd. *Password Security: A dash of 'salt' and little of 'hash' to go please!* June 2012. URL: http://blogs.quickheal.com/password-security-a-dash-of-salt-and-little-of-hash-to-go-please/ (visited on 03/16/2018).

[34] *Make the Web Faster — Google Developers*. URL: https://developers.google.com/speed/ (visited on 03/16/2018).

[35] *Markdown*. Mar. 2018. URL: https://en.wikipedia.org/wiki/Markdown (visited on 03/16/2018).

[36] *Material Design*. URL: https://material.io/ (visited on 03/16/2018).

[37] *NoSQL*. URL: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page (visited on 03/16/2018).

[38] *Number of internet users (2016) - internet live stats*. URL: http://www.internetlivestats.com/internet-users/ (visited on 03/16/2018).

[39] Taylor Otwell. *Laravel - The PHP Framework For Web Artisans*. URL: https://laravel.com/ (visited on 03/16/2018).

[40] *Pingdom Tools: website speed test*. URL: https://tools.pingdom.com/ (visited on 03/16/2018).

[41] *PostgreSQL:the world's most advanced open source database*. URL: https://www.postgresql.org/ (visited on 03/16/2018).

[42] *Read–eval–print loop*. Feb. 2018. URL: https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop (visited on 03/16/2018).

[43] John C. Reynolds. *Theories of programming languages*. Cambridge University Press, 1998. ISBN: 978-0-521-59414-1.

[44] Peter Rob and Carlos Coronel. *Database systems - design, implementation, and management (2. ed.)* Boyd and Fraser, 1995. ISBN: 978-0-7895-0052-6.

[45] Niko Schwarz, Mircea Lungu, and Oscar Nierstrasz. "Seuss: Decoupling responsibilities from static methods for fine-grained configurability". In: *Journal of Object Technology* 11.1 (2012), pp. 1–23. DOI: 10.5381/jot.2012.11.1.a3. URL: https://doi.org/10.5381/jot.2012.11.1.a3.

[46] Mark Seemann. *Dependency Injection is Loose Coupling*. Apr. 2010. URL: http://blog.ploeh.dk/2010/04/07/DependencyInjectionisLooseCoupling/ (visited on 03/16/2018).

[47] Manuel Serrano, Erick Gallesio, and Florian Loitsch. "Hop: a language for programming the web 2.0". In: *Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, Oregon, USA*. 2006, pp. 975–985. DOI: 10.1145/1176617.1176756. URL: http://doi.acm.org/10.1145/1176617.1176756.

[48] *Session Management Cheat Sheet*. URL: https://www.owasp.org/index.php/Session_Management_Cheat_Sheet#Introduction (visited on 03/16/2018).

[49] Panagiotis Sfetsos and Ioannis G. Stamelos. *Agile Software Development Quality Assurance*. IGI Global, 2007.

[50] *Socket.IO: FEATURING THE FASTEST AND MOST RELIABLE REAL-TIME ENGINE*. URL: https://socket.io/ (visited on 03/16/2018).

[51] Mads Soegaard and Rikke Friis Dam. *Encyclopedia of Human-Computer Interaction*. Interaction Design Foundation, 2013.

[52] *Speed Matters*. June 2009. URL: https://research.googleblog.com/2009/06/speed-matters.html (visited on 03/16/2018).

[53] *Stack Overflow Developer Survey 2017*. URL: https://insights.stackoverflow.com/survey/2017 (visited on 03/16/2018).

[54] *Stack Overflow Developer Survey 2018*. URL: https://insights.stackoverflow.com/survey/2018/?utm_source=Iterable&utm_medium=email&utm_campaign=dev-survey-2018-promotion (visited on 03/16/2018).

[55] David A. Watt and William Findlay. *Programming language design concepts*. Wiley, 2004. ISBN: 978-0-470-85320-7.

[56] *WebSockets*. URL: https://developer.mozilla.org/en-US/docs/Glossary/WebSockets (visited on 03/16/2018).

[57] Matthew West. *Developing high quality data models*. Morgan Kaufmann, 2011.

[58] *White Paper Imperva Web Application Attack Report*. July 2012. URL: https://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed4.pdf (visited on 03/16/2018).

[59] *Working with Triggers*. URL: https://msdn.microsoft.com/en-us/library/sdk3bcyw.aspx (visited on 03/16/2018).

[60] Nick Wu. *TryLinks: An online platform for everyone to try the Links Programming Language*. URL: https://github.com/NickWu007/TryLinks (visited on 03/16/2018).