

Evaluation of a Haskell Web Framework

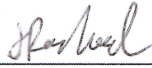
Aston University BSC Computer Science


Junaid Ali Rasheed

April 23, 2018

Final Year Project Definition Form

<i>Student's name</i> Junaid Rasheed
<i>Course</i> Computer Science
<i>Project title</i> Evaluation of a Haskell Web Framework
<i>What is the project about?</i> Comparing a Haskell Web Framework with a more traditional framework. The two frameworks that will be compared are Yesod (Haskell) and Django (Python). A website similar to Twitter will be created with both frameworks and then both websites will be compared. The comparisons will include differences in page load speed, safety, reliability and whether type checking during compile time vs runtime helps reduce bugs, maintainability by comparing the ease of adding a new feature to both websites, and the ease of testing in both frameworks.
<i>What is the project deliverable?</i> Two websites that are functionally identical, one developed using Django, and the other using Yesod. Then a report will be written comparing the reliability, maintainability, speed, safety, and possibly the scalability of both frameworks. The report will evaluate the advantages and disadvantages of making a website using Yesod.
<i>What is original about this project?</i> There has not been any detailed comparisons between a Haskell Web Framework and a Web Framework in a more traditional object oriented language like PHP or Python. This project will provide enough detail to people looking into using a Haskell Web Framework to help them inform their decision.
<i>Timetable showing main stages in work plan</i> End of October: Comfortable with Yesod and Django, create a simple website with both frameworks. Create tests for the simple website. November: Start to create a simple Twitter clone in both frameworks. Ensure tests are created for new features. Debug any errors. End of January: Simple twitter clone finished, users can make posts, follow each other, make 'hashtags' (any word with a '#' preceding itself is linkable to other posts containing the 'hashtagged' word and looking up the word will show all posts containing the 'hashtagged' word in a paginated results view.). Tests created for all features, bugs debugged. End of February: Add a new feature. The feature that planned is a way to send private messages directly to other users. Users will have an area displaying all private messages sent and received and will be able to reply to other people's messages. Add tests for this feature and debug any bugs encountered. End of March: Record and resolve any bugs and errors in the framework, ensure that each framework has a sufficient number of unit tests and that all unit tests pass. Compare both frameworks, with a focus on speed, reliability, and safety. April: Begin and finish the Final Project report, prepare for live demos.

Student's signature  *Date* 19 Oct 2017

Supervisor's signature  *Date* 19 Oct 2017

Contents

1	Introduction	1
1.1	The Chosen Frameworks	1
1.2	The Report	2
2	Background	3
2.1	Similar Previous Work	3
2.2	Real World Haskell Sites	5
3	Preparation	7
3.1	Planning the Website	7
3.2	Learning the Frameworks	7
3.2.1	Learning Django	8
3.2.2	Learning Yesod	8
3.3	The Evaluation	8
4	Deliverable	9
4.1	The Website - Wire	9
4.1.1	The Home Page	9
4.1.2	Authentication	10
4.1.3	User Profiles	11
4.1.4	The Search Page	13
4.2	The Yesod Implementation	14
4.2.1	The Scaffold	14
4.2.2	Defining Routes	14
4.2.3	Database Entities	15
4.2.4	Handlers	15
4.2.5	Templates	17
4.2.6	Tests	18
4.3	The Django Implementation	19
4.3.1	Creating a Project	19
4.3.2	Creating Apps	19
4.3.3	Routes	19
4.3.4	Database Entities	20

4.3.5	Views	20
4.3.6	Templates	23
4.3.7	Tests	24
5	Evaluation	25
6	Conclusion	26
	References	27
	Bibliography	28
A	Project Diary	31
A.1	Meeting 1 - 3rd October 2017	31
A.1.1	Meeting Notes:	31
A.2	Meeting 2 - 12th October 2017	32
A.2.1	Meeting Notes:	32
A.3	Meeting 3 - 19th October 2017	32
A.3.1	Meeting Notes:	32
A.4	Meeting 4 - 24th October 2017	32
A.4.1	Meeting Notes:	32
A.5	Meeting 5 - 10th November 2017	33
A.5.1	Meeting Notes:	33
A.6	Meeting 6 - 16th November 2017	33
A.6.1	Meeting Notes:	33
A.7	Meeting 7 - 23rd November 2017	33
A.7.1	Meeting Notes:	33
A.8	Meeting 8 - 14th December 2017	34
A.8.1	Meeting Notes:	34
A.9	Meeting 9 - 1st February 2018	34
A.9.1	Meeting Notes:	34
A.10	Meeting 10 - 15th February 2018	35
A.10.1	Meeting Notes:	35
A.11	Meeting 11 - 22nd March 2018	36
A.11.1	Meeting Notes:	36
A.12	Meeting 12 - 19th April 2018	36
A.12.1	Meeting Notes:	36
A.13	Project Definition Form	37
A.13.1	14th October 2017	37
A.13.2	15th October 2017	37
A.13.3	19th October 2017	37
A.14	Interim Report	37
A.14.1	18th January 2018	37
A.14.2	19th January 2018	38

A.15 Software Development	38
A.15.1 Yesod Site	38
A.15.2 Django Site	40
A.16 Project Diary Final Report	40
A.16.1 20th April 2018	40
A.16.2 22nd April 2018	41
B Ethics Form	42

List of Figures

2.1	Snap and other frameworks, values are in requests per second © 2011 IEEE	4
4.1	The home page	9
4.2	The login page	10
4.3	The signup page	10
4.4	The profile page for the current user	11
4.5	The profile page for other users	12
4.6	The create message form	12
4.7	The search page	13
4.8	The search results page for messages	13
4.9	The search results page for other users	14

List of Code Blocks

4.1	Yesod URL routes	15
4.2	Yesod Database Entities	15
4.3	GET request handler for current profile page	16
4.4	The message form	16
4.5	POST request handler for current profile page	16
4.6	GET request handler for getting user data	17
4.7	Template file for the search page	18
4.8	Test the profile page	18
4.9	Checking a JSON response	18
4.10	An extract of Django routes	19
4.11	The user entity in Django	20
4.12	Class-based current profile view	20
4.13	Django message form	21
4.14	Function-based create message view	21
4.15	Function-based view for returning user data	22
4.16	Class-based view to search for a given message	22
4.17	Template file for the search page	23
4.18	Django current profile test	24
4.19	Django checking a JSON response test	24

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Chapter 1

Introduction

In this report, we will compare two web frameworks, one written in Haskell and another, more popular framework, written in a typical object oriented language. To perform the evaluation, a functionally identical website was created in both frameworks.

One issue that individuals face when looking into Haskell Web frameworks is the lack of detailed comparisons between these frameworks and more traditional frameworks that these individuals likely have experience in. This report will provide an in-depth look at the advantages and disadvantages of choosing to use a Haskell Web Framework, and will help these individuals come to an informed decision on whether or not a Haskell Web Framework is best for them.

1.1 The Chosen Frameworks

Yesod is a fully featured and modular web framework written in Haskell. Yesod claims to use features of the Haskell language to provide a fast, modular, and type safe web framework. By choosing Yesod as the web framework for the Haskell language, we will be able to determine whether or not Haskell's type safety, referential transparency, and lazy compiling is an advantage or a disadvantage for web developers.

Yesod attempts to ease the web development process by playing to the strengths of the Haskell programming language. Haskell's strong compile-time guarantees of correctness not only encompass types; referential transparency ensures that we don't have any unintended side effects. Pattern matching on algebraic data types can help guarantee we've accounted for every possible case. By building upon Haskell, entire classes of bugs disappear. (Snoyman, 2012, Introduction)

Django is a Python Web Framework. Django, like Yesod, is a "batteries included" web framework, "instead of having to open up the language to insert your own power (batteries), you just have to flick the switch and Django does the rest." Django was chosen as the second framework because it is modular, like Yesod, and Python is one of the most common programming languages today, second only to Node.js. Python is also a dynamically typed language, which helped us evaluate whether or not Haskell's static type checking actually saves time and effort when developing. (George, 2017).

The Django and Yesod web frameworks have a similar set of features. This ensured that we could make a functionally identical site in both of these frameworks with a similar amount of effort. This enables us to make a fair comparison between Django and Yesod, enabling us to come to a conclusion on whether a Haskell web framework may be a good choice for a developer rather than a more tradition web framework.

1.2 The Report

The rest of this report will discuss any pieces of work similar to this project, the process of writing the code for both frameworks, including any preparation that had to be done, and a detailed evaluation of the websites that were produced. The evaluation will compare page load speeds, the reliability and maintainability of the websites, the ease of writing new code, the ease of debugging issues, the features including in each framework's test suite, and whether the static typing and type safety of the Haskell language help or hinder the process of developing a website.

Chapter 2

Background

In this chapter, we will discuss previous work and research pertaining to Haskell web frameworks and any real-world sites built using a Haskell web framework.

2.1 Similar Previous Work

Looking through scientific journals, online articles, and blog posts, you can find many individuals documenting their experiences and performing reviews of Haskell Web Frameworks. For example, in the IEEE Internet Computing journal, Collins and Beardsley have written an article giving an overview of Snap. According to the article, snap is a simple web framework written in Haskell where programming is done at a similar level of abstraction to Java servlets. The article instructs the reader on how to install Snap, walks the reader through some sample code, and shows a quick comparison between Snap and other major web frameworks. The comparison is a benchmark of each framework, recording the amount of time it takes for each framework to respond to a request. The benchmark results can be seen in Figure 2.1. (Collins & Beardsley, 2011)

Figure 2.1 shows the results of two benchmarks, one where a server responded to a request by sending the string "pong", and another that records how fast a server can send a 49 kilobyte image. As you can see in the graph, for the file benchmark, the Snap framework was faster than all other frameworks. Snap was only beaten by Node.js in the Pong benchmark when logging was turned on. Turning logging off dramatically increased the performance of Snap, resulting in Snap being 55% faster than Node.js in the Pong benchmark. (Collins & Beardsley, 2011)

So, from the research done by Collins and Beardsley, we can see that Haskell web frameworks can be significantly faster than more traditional web frameworks. This is because Haskell uses the Glasgow Haskell Compiler (GHC). GHC compiles Haskell programs into native machine code, ensuring high performance, especially for concurrent programs such as web servers. Because of this, any Haskell web framework that uses GHC, including Yesod, will serve requests faster than most traditional web frameworks. (Gamari, 2018)

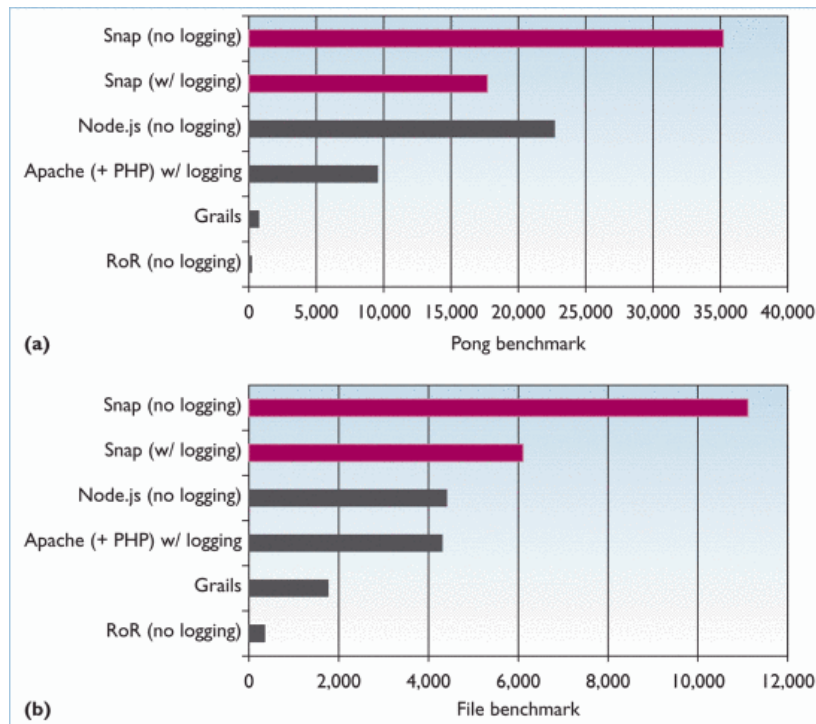


Figure 2.1: Snap and other frameworks, values are in requests per second

© 2011 IEEE

Bajt published a blog post on his website with the title “Comparing Haskell Web Frameworks”. In the post, Bajt provides a quick comparison between some popular Haskell Web Frameworks, including Yesod and Snap. The comparisons include the installation process of each framework, the way they handle routing – i.e. pointing a URL to a piece of code that will produce a response, and the quality of documentation for each framework. (Bajt, 2014)

At the end of his comparison, Bajt found that Yesod was the best framework for his use case. One of the reasons for his choice was because of the great documentation available for Yesod. The creator of Yesod has written a book, *Haskell Programming from first principles*, that is very comprehensive. The book is available for free on the Yesod website. The amount of detail contained in this book is also one of the reasons the Yesod web framework was chosen for this project. (Bajt, 2014; Snoyman, 2012)

In “A Haskell Beginner’s Experience With Yesod”, Picciau discusses his experiences in using Yesod as a beginner to the Haskell programming language. He mentions the depth and thoroughness of the Yesod book when learning Yesod. However, when making an actual website, he came across difficulties when trying to implement features that required the use of functions not in the book. The author had to check the documentation of the functions on Hackage, a Haskell package archive. On Hackage, most functions contain a type signature and normally a one line description. The author mentions how the type signatures would probably be enough for experienced Haskell developers to work out how to use a function but, as a beginner, finding out how a function works using types was much more difficult. Because of this, the author had to spend a lot of time fixing type errors. (Picciau,

2018)

When first starting the project, I personally experienced the same issues described in Picciau's blog post. I found it difficult to understand the type signatures available on Hackage, resulting in spending a lot of time fixing unmatched type errors when trying to use functions not documented in the book. However, as I became more experienced in Yesod and Haskell, these problems became more and more rare as my understanding of Haskell type signatures increased.

2.2 Real World Haskell Sites

Some readers will be concerned about whether or not a Haskell web framework like Yesod is ready for production websites. This is a valid concern considering the relatively small amount of Haskell programmers when compared to mainstream programming languages. Readers will be pleased to know, however, that there are some high traffic sites that are built using the Yesod web framework.

Freckle, previously known as Front Row Education, is an education platform that provides a service to almost 10 million students (Alvarez, 2018). In 2015, Freckle migrated their site to the Yesod web framework and have been using it ever since. Kurilin, the CTO of Freckle, wrote an article on his experience of using Yesod for a high traffic website. (Kurilin, 2015)

In his article, Kurilin states that the reason they chose a Haskell Web Framework was because of the low resource usage and the ability to make quick iterations that the Haskell language gives you. The article also discusses how static typing saves time when writing unit tests. The developers at Freckle did not have to deal with checking for null exceptions, mismatched types, and other common bugs that are annoying to deal with. Spending less time dealing with dynamic typing gives developers more time in implementing their features. The modularity of Yesod also allows Freckle to reuse complex code, reducing potential mistakes by developers, reducing the amount of code that needs to be written, and allowing code to be updated quickly without the need to repeat changes. All of these advantages allow developers to be more efficient and write fewer bugs. (Kurilin, 2015)

However, there are some issues that the Freckle team came across during their migration. Haskell builds are normally quite slow, as all external libraries used have to be compiled during the build process. Kurilin mentions that builds took 5-10 minutes on their most powerful machines. The author does mention that the team could improve their build process by, for example, caching built files. The testing suite caused problems for the team because when a test fails, they could not determine which condition caused the failure when a test block has multiple conditions. The issue, however, was reported to developers behind the testing library and the current version of the testing suite does not have the issue mentioned in the article. The lack of documentation for some functions also caused some frustration to the development team, especially for the more junior developers who could not rely on type signatures. (Kurilin, 2015)

When Freckle switched their main API to Yesod, their CPU usage rose to 95%. This issue did not occur during testing and profiling, in fact, the Freckle team were the first to experience this particular issue. This is one issue when using a relatively niche language like Haskell, you have to be comfortable with the idea that you may be the first person to experience a particular issue. With other popular frameworks, such as Django, any issue you discover has most likely been found and fixed by other members of the community.

Despite these issues, Freckle decided to stick with Yesod due to the advantages of the Haskell

compiler and the fact that the issues they experienced with regards to documentation and build time are improving. And, although the community is small, you can almost always find help by asking on the StackOverflow or Google Groups pages or by visiting the #haskell-beginners IRC channel, an online chat room where developers new to Haskell can quickly and easily get help from more experienced developers.

Chapter 3

Preparation

In this chapter, we will discuss the work that needed to be done before work could be started on developing websites in Yesod and Django.

3.1 Planning the Website

Before any work was done, a plan was created that indicated the features the website should contain to ensure that the features of each framework are able to be fairly tested and evaluated. The website to be created was a twitter clone with the following features: a home page, authentication, a profile page, ability to post a message, ability to post ‘tagged’ messages (any words with a preceding ‘#’ becomes link that leads to a search page), ability to search for messages and users, and an ability to follow other users. All features implemented would also have unit tests implemented using the testing tools available in each framework. After each site is feature complete, a new feature, the ability to message other users, should be implemented.

Implementing these features allows us to test page load speed by navigating to certain web pages. Safety can be tested by analysing how each framework deals with custom user input. We can evaluate how the static type checking and type safety features of Haskell affects reliability when compared to dynamic type checking in Python. Testing all of our implemented features allows us to fairly evaluate the test suites included with each framework. Implementing a new feature once each site is feature complete will also allow us to test the maintainability of each framework.

By planning the website before any work was done, there was a clear vision of what the website should look like. This allowed development to focus on implementing the specified features rather than trying to create new features while developing at the same time, ensuring that functionally identical websites are created in both frameworks that can be fairly compared and evaluated.

3.2 Learning the Frameworks

Even after the planning was completed for both sites, before any development could be done, the basics of each framework must be learned. This ensures that code produced follows the latest standards of each framework, the built-in features of each framework that may help with development

are understood and used appropriately, and code produced is of a high standard, maintainable, and readable.

3.2.1 Learning Django

Because of previous experience with Python and other object oriented web frameworks, Django was learned quickly by going through the official Django tutorials and documentation.

The Django tutorials themselves were straight forward for someone who has experience in Python and other web frameworks. The tutorials walk you through installing Django and creating your own app. In Django, an app resides in a project and is a web application that performs some function. An example of an app could be a web blogging system. A Django project is a collection of apps and configuration settings for a particular website. After completing the Django tutorials, work on the planned website was begun. (“Getting Started”, 2018)

3.2.2 Learning Yesod

Coming from an object oriented background, it was not trivial to start using a functional programming language like Haskell. Before development with the Yesod framework could be started, the Haskell language had to be learned to an adequate level.

To help with learning Haskell, the book *Haskell Programming from first principles* (Allen & Moronuki, 2016) was used. This book walks the reader through learning the Haskell language beginning with the fundamentals. Reading through several chapters of the book gives the reader a basic understanding of programming with Haskell, allowing the reader start learning Yesod itself, referring to the book to understand more advanced concepts as you come across them.

To learn Yesod itself, the book *Developing Web Applications with Haskell and Yesod* (Snoyman, 2012), written by the person who wrote Yesod, was used. The book goes through all of the features of the Yesod framework in an easy to understand manner. After reading the book, development on the planned website using the Yesod framework was started.

3.3 The Evaluation

Once the website was feature complete on both frameworks, a series of experiments were ran to compare the features that we planned to test during the planning phase. The raw data of these experiments can be found in the appendix and an evaluation of these results are discussed in chapter 5.

Chapter 4

Deliverable

This chapter will go through the work that was produced as a result of this project. We will take a look at the website, examples of code for both frameworks, and how we tested both frameworks.

4.1 The Website - Wire

Most of the features that we planned for were implemented in the final version of the website. The website was named ‘Wire’ and users can create an account, post messages, follow other users, see user messages, post tagged messages, and search for other messages and users. The following subsections contain pictures of the website produced. The pictures taken are of the Yesod website but the Django site is functionally identical, with only a few minor styling differences.

4.1.1 The Home Page

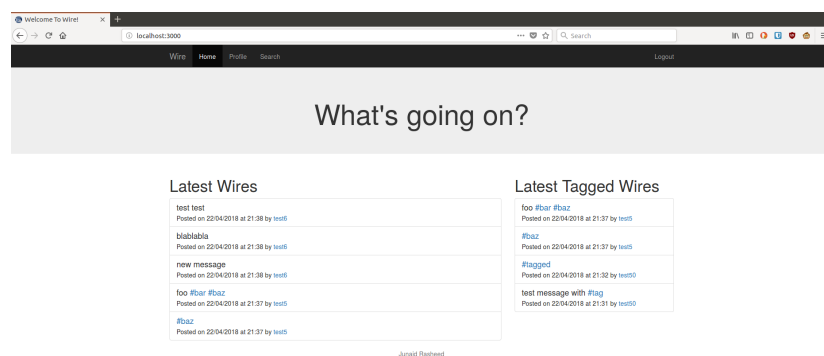


Figure 4.1: The home page

The home page is the first page the user sees when they access the website. The home page contains the latest messages posted by users on the website, with a separate list for tagged messages. Messages that contain tags are links that take the user to the search page. There is a navigation bar at the top of the home page. This navigation bar is present on all pages. When a user is not logged in, the navigation bar allows the user to access the login and signup pages. Once logged in, the user can use the navigation bar to access their profile page or to logout.

4.1.2 Authentication

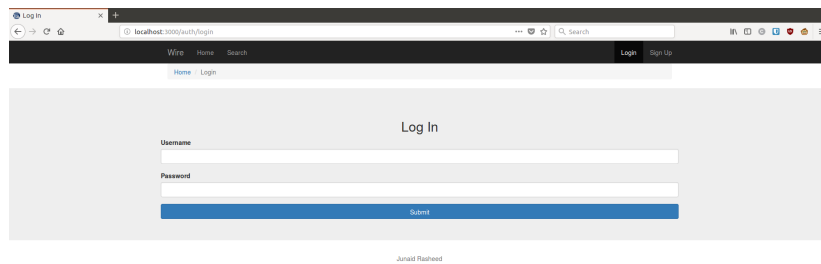


Figure 4.2: The login page

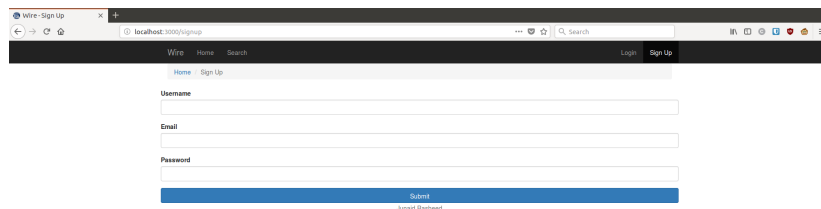


Figure 4.3: The signup page

The authentication functionality of this website includes signing up and logging in using a web page, and logging out using a button in the navigation bar. In figure 4.3, you can see that signing

up requires a username, e-mail, and password. The username and e-mail must be unique. Users are shown a warning message if they try to sign up with details that are already taken. Once a user signs up, their details are stored in the database, with the password being hashed and salted to ensure that it is stored safely.

When logging in, you only need to supply a username and password, as you can see in figure 4.2. This is because the username is a unique identifier, we can use it to determine which user to log in as. When the username and password is submitted, we look up the username in the database, encrypt the password, and see if the encrypted password matches the one stored in the database. If there is a match, the user is authenticated and redirected to the home page. If there is an issue, the user is redirected back to the login page with an appropriate error messages.

Once the user is logged in, they can use the log out button which is present on the top right of the navigation bar. Clicking this button immediately logs the user out and they are redirected to the home page.

4.1.3 User Profiles

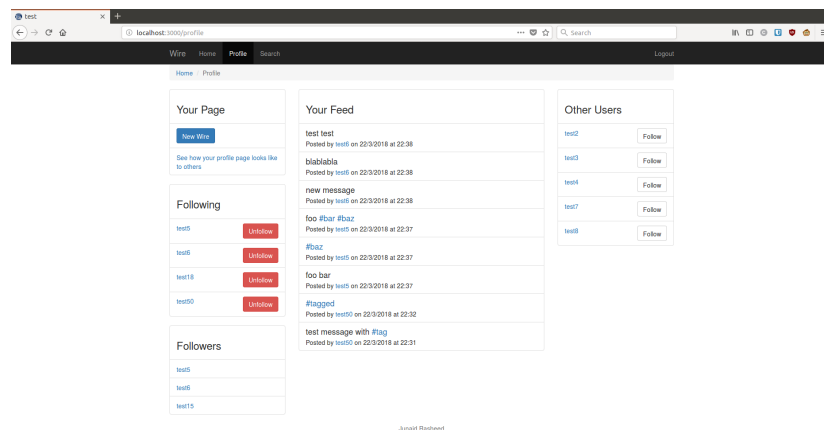


Figure 4.4: The profile page for the current user

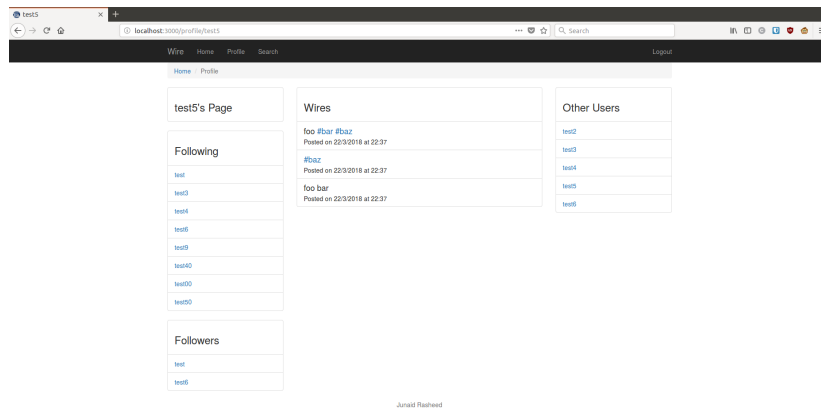


Figure 4.5: The profile page for other users

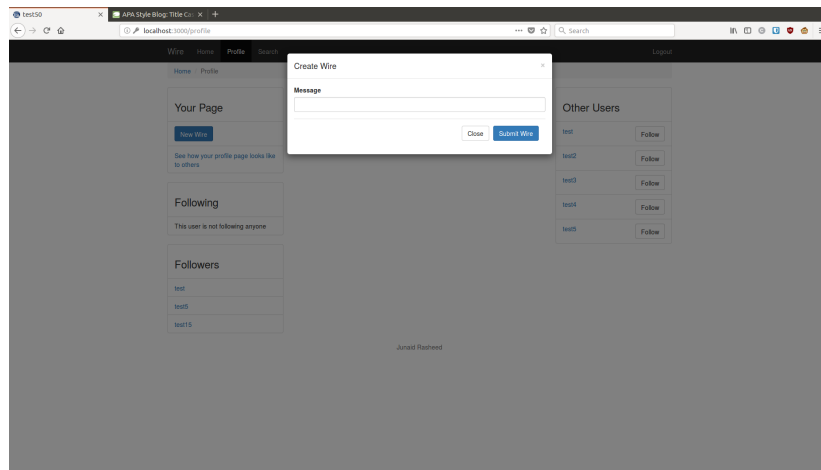


Figure 4.6: The create message form

The profile pages are probably the most complex with regards to logic in the entire website. There are two different profile pages, one for the currently logged in user which only they can see, and another for when someone is viewing the profile page of another user.

Figure 4.4 is what a logged in user sees if they visit their own profile page. On this page, they can see a list of messages posted by users that they follow, see which users are following them, follow and unfollow other users, and post their own messages. They can also post a new message by clicking on the 'New Wire' button. Clicking this button displays a form which can be seen in figure 4.6.

Figure 4.5 is what the profile page looks like when a user navigates to another person's profile page, or when they click the 'See how your profile page looks like to others' link on their own profile page. This page shows the name of the person who owns the page, the users they follow, other users that follow them, and messages that they have posted.

One feature to note is that the messages and other user information on the profile page is loaded in via AJAX. This ensures that the initial page load is quick, following and unfollowing users does not necessitate an entire page reload, and makes it easier to, if desired, add a feature to automatically update the message area.

4.1.4 The Search Page

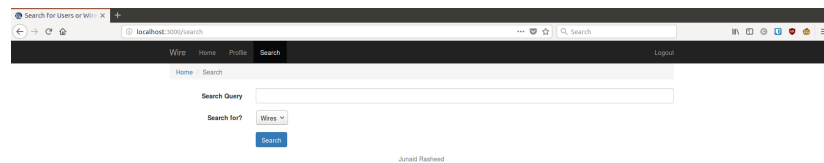


Figure 4.7: The search page

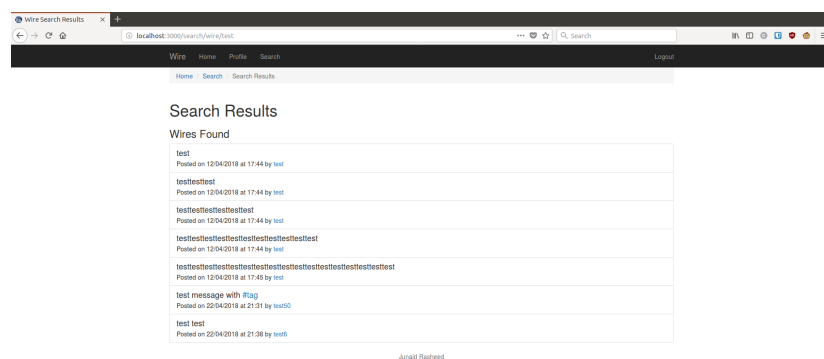


Figure 4.8: The search results page for messages

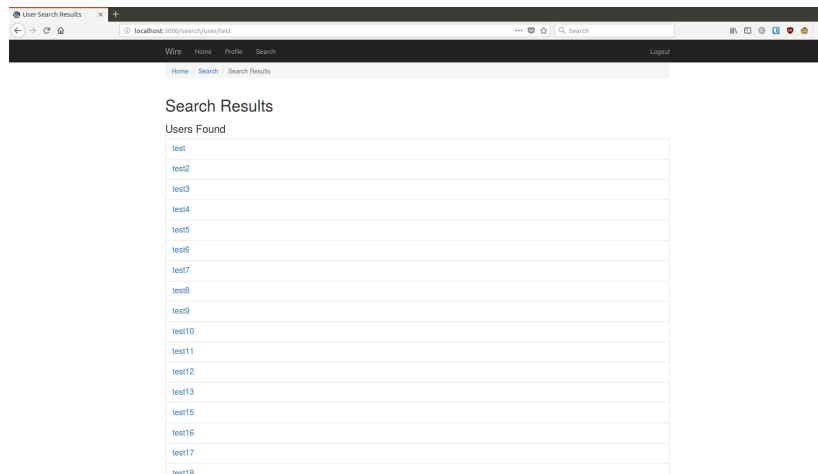


Figure 4.9: The search results page for other users

Figure 4.7 is the page that the user is navigated to when they click the search button in the navigation bar. In this page, they can type in their search query and use the dropdown to specify whether they are searching for users or messages. Once they submit their query, they are redirected to the search results page, as seen in Figures 4.9 and 4.8.

4.2 The Yesod Implementation

In this section, we will briefly discuss how the website was implemented in Yesod. We will take a look at creating database entities, URL routing, handling requests, creating templates, and writing tests.

4.2.1 The Scaffold

When creating a new Yesod site, it is recommended to use the scaffolding tool. The scaffolding tool generates code that sets up the structure of your project. It creates the files needed to connect to a database and launch a website. Sample code is included for developers to see how the framework works. By running the scaffolding tool, it is clear to the developer where source files, configuration settings, templates, and static files should be kept. (Snoyman, 2012, Scaffolding and the Site Template)

The Yesod codebase used for this project was built on top of code generated by the scaffolding tool using the yesod-postgres template, which tells the generated code to be compatible with a PostgreSQL database.

4.2.2 Defining Routes

URL routes are available routes in an application and are specified in the 'config/routes' file. In this file, you specify whether a request from a route is a GET or POST request, the handler that deals with the request, and any parameters that are part of the request. One of the features of Yesod is

type safety in URLs, so you can specify the actual type that a URL parameter should be. If the user tries to navigate to a page with an invalid parameter, a 404 page will be shown. An extract from the ‘config/routes’ file can be seen in code block 4.1 below.

```

1  /profile MyProfileR GET POST
2  /profile/#Text ProfileR GET
3
4  /user UserGetAllR GET
5  /user-not-following UserGetAllExcludingFollowingR GET
6  /user/#Text UserGetAllExcludingUsernameR GET
7  /user/id/#UserId UserGetIdR GET
8  /users/*[UserId] UserGetIdsR GET

```

Code 4.1: Yesod URL routes

In code block 4.1, you can see that there are two routes for the profile page. One, `/profile`, is for users viewing their own profile page and the other, `/profile/#Text`, are for viewing the profile pages of the other users. The `#Text` part of the route specifies that there should be one parameter for this route with the type `Text`. If multiple parameters are provided or a parameter is not of the type `Text`, a 404 error is shown.

Further examples of URL parameters can be seen in the user routes. `/user/id/#UserId` expects a user id as a parameter. If the specified id is not found, a 404 page is shown. `/users/[UserId]` expects a list of user ids, which would look like `/users/1/2/3/4/`.

4.2.3 Database Entities

Database entities are defined in the ‘config/model’ file. In this file, you give a name to the entity you want to create, the names and types of it’s fields, functions that the entity and it’s fields should include, and any fields that are unique, i.e. Cannot be shared with other entities. Once an entity is added to this file, it is created in the database when the codebase is compiled and helper functions are created that can be used when programming. Code block 4.2 contains the definition of the user entity extracted from the ‘config/models’ file.

```

1  User json key
2      username Text Eq
3      email    Text Eq
4      password Text
5      UniqueUser username
6      UniqueEmail email
7      deriving Typeable Show

```

Code 4.2: Yesod Database Entities

4.2.4 Handlers

Every URL route in the application points to a handler. Handlers are located in the `src/handler` directory. They are used to perform any calculations or queries that need to be done and then respond

to a request by rendering a template file, returning a JSON object, or by redirecting to another handler.

Code block 4.3 is the handler used to respond to get requests to display the profile page of the current user. The handler ensures the user is logged in, loads the user's details, loads data on people being followed by the current user, and then loads the template file. The parameters that are created and available in the handler are also available in the template file.

```

1  -- Loads the 'My Profile' page for the currently logged in user. If someone
2  -- who is not logged in attempts to access the page, a 404 error will be shown.
3  getMyProfileR :: Handler Html
4  getMyProfileR = do
5      (Entity userId user) <- requireAuth
6      let username = userUsername user
7
8      -- Load messages posted by users followed by the current user
9      follows <- runDB $ selectList [FollowFollowingId ==. userId] []
10     let followIds = map (\(Entity _ (Follow followerId _)) -> followerId)
follows
11
12     (formWidget, formEnctype) <- generateFormPost $ messageForm userId
13     defaultLayout $ do
14         setTitle . toHtml $ userUsername user
15         $(widgetFile "currentprofile")

```

Code 4.3: GET request handler for current profile page

You may have noticed the message form being generated in code block 4.3. This form is defined in Haskell and the source code can be seen in code block 4.4. When defining the form, we give it a user id to specify the user creating a message, we tell the form that there is one input field that is required. Two hidden fields are also included to ensure the form has all the data needed to create a message when it is submitted.

```

1  messageForm :: UserId -> Form Message
2  messageForm userId = renderBootstrap3 BootstrapBasicForm $ Message
3      <$> areq textField (bfs ("Message" :: Text)) Nothing
4      <*> pure userId
5      <*> lift (liftIO getCurrentTime)

```

Code 4.4: The message form

When the user submits the form, the post handler is ran, which can be seen in code block 4.5. In this handler, we use the function `runFormPost` to check whether or not the form is valid. If the form is valid, the message is added to the database and the profile page is reloaded with a success message. If there's an issue with the form, the profile page is reloaded with an appropriate error message.

```

1  -- Create a new wire for the logged in user
2  postMyProfileR :: Handler Html
3  postMyProfileR = do
4      (Entity userId _) <- requireAuth
5      ((result, _), _) <- runFormPost $ messageForm userId

```

```

6      case result of
7          FormSuccess message -> do
8              void $ runDB . insert $ message
9              setSession "msgrendered" "true"
10             setMessage $ renderSuccessMessage "Wire Sent"
11             redirect MyProfileR
12          FormFailure errors -> do
13              let renderedMessages = map renderErrorMessage errors
14              setSession "msgrendered" "true"
15              setMessage $ toHtml renderedMessages
16              redirect MyProfileR
17          FormMissing -> do
18              setSession "msgrendered" "true"
19              setMessage $ renderErrorMessage "Form is missing"
20              redirect MyProfileR

```

Code 4.5: POST request handler for current profile page

If a route contains a URL parameter, the handler must also have a parameter to store the value of the given parameter. Code block 4.6 is the source code for a handler that takes in a user id as a parameter. As you can see, we do not have to check for the type of this parameter or whether it is not null. We specified the type of the URL parameter in the routes file (code block 4.1) so Yesod will perform type checking for us, saving developers time from having to manually deal with invalid types or values.

```

1      -- | Takes in a user id and returns data on the user matching the given id.
2      -- If no user is found, an empty JSON object is returned.
3      getUserGetIdR :: UserId -> Handler Value
4      getUserGetIdR userId = do
5          users <- runDB $ selectList [UserId ==. userId] []
6          let cleanUsers = map (\(Entity uid (User uname _ _)) -> (object ["id" .=
7              uid, "username" .= uname])) users
8          returnJson cleanUsers

```

Code 4.6: GET request handler for getting user data

4.2.5 Templates

The templates used in Yesod are called Shakespearean templates. Shakespearean templates allow you to write type-safe templates that are compiled, helping prevent runtime errors. The syntax for Shakespearean templates are similar to the languages they are based on, with minor changes to the syntax used in the templates. For example, the HTML template language, Hamlet, uses indentation rather than opening and closing tags to denote nesting. Within these templates, you can use Haskell variables, create type-safe routes, and implement conditional and looping logic. (Snoyman, 2012, Shakespearean Templates)

When a template file is loaded in a handler, the file is actually included inside a default file. The default file contains content that is common to all pages. This ensures that code does not need to be

repeated, reducing the chance of mistakes and making it easier to change the layout of the whole site.

A simple template file can be seen in code block 4.7. In this block, you can see how indentation is used to determine nesting. Variable interpolation is done using `#{variableName}`. You can see the form is being loaded using the `^{widgetName}`, which renders the given widget onto the page. Type safe URLs are loaded using `@{routeName optionalParameters}`.

```

1  <main>
2    <div .container>
3      <div .row>
4        <div .col-sm-12>
5          <form #search-form .inline .form-horizontal role=form
method=post action=@{SearchR} enctype=#{formEnctype}>
6            ^{formWidget}

```

Code 4.7: Template file for the search page

4.2.6 Tests

The Yesod test suite allows you to create BDD-style tests. When creating a test, you specify what it should do, create any database entities you need, make a request to a handler, and examine the response to see if the data you received is correct. Code block 4.8 is an actual test from the website. The test creates and logs in as a new user, loads the profile page, and ensures that the resulting HTML contains the text that it should contain. Yesod gives you the ability to use CSS selectors when checking the HTML page given by a response, allowing you to be very specific.

```

1  it "asserts that the current profile page looks right" $ do
2    foo <- createUser "foo" "foo@bar.com" "foo"
3    authenticateAs foo
4
5    get MyProfileR
6    htmlAnyContain "h3" "Your Page"
7    htmlAnyContain "h3" "Your Feed"
8    htmlAnyContain "h3" "Followers"
9    htmlAnyContain "h3" "Following"
10   htmlAnyContain "h3" "Other Users"

```

Code 4.8: Test the profile page

The testing suite also has the ability to check if a JSON response contains the data that we expect. However, you do not have the same helper functions available to you when compared to checking HTML responses. When checking JSON response, you must examine the body of the response itself. You cannot check if a JSON key has a given value. See code block 4.9 below.

```

1  it "asserts all users are returned when not authenticated" $ do
2    _ <- createUser "foo" "foo@bar.com" "foo"
3    _ <- createUser "bar" "bar@bar.com" "foo"
4    _ <- createUser "baz" "baz@bar.com" "foo"
5

```

```
6         get UserGetAllR
7
8         bodyContains "username"
9         bodyContains "id"
10        bodyNotContains "email"
11        bodyNotContains "password"
12        bodyContains "foo"
13        bodyContains "bar"
14        bodyContains "baz"
```

Code 4.9: Checking a JSON response

4.3 The Django Implementation

Now, we will discuss how the Django site was implemented. We will go through Django apps, routes, entities, views, templates, and tests.

4.3.1 Creating a Project

All Django sites require a Django project. A project is a directory that contains all the settings needed for a Django website. This includes database settings, the apps being used, application settings, and Django-specific settings. To create the project, the `django-admin` tool was used. This tool generates the code needed to connect to a database and start a Django site. (“Getting Started”, 2018)

4.3.2 Creating Apps

The code used for the actual web application resides in two Django apps, `base` and `wire_profile`. In Django, an app is a web application that can be a part of a project. These apps contain the URL routes used in the application, database entities, views that respond to requests, templates, and tests. The `manage.py` tool provided by Django was used to create apps. This tool creates a directory with a specified name and a layout of files and directories that is preferred for Django apps. (“Getting Started”, 2018)

4.3.3 Routes

Django routes are specified in the `urls.py` file within an app. The routes specify a URL path, a view that responds to requests from the given path, and a name that is used to refer to a route within code. Django routes can contain URL parameters, like `Yesod`. The valid values for these parameters can be defined using regular expressions or a few built in types like `string` or `int`. To denote a list of parameters, `path` can be used. An example of Django routes can be seen in code block 4.10.

```
1     path('following/<path:username>', views.get_following, name='get_following'),
2     path('users/<path:user_ids>', views.get_user_ids, name='get_user_ids'),
3     path('user/id/<int:user_id>', views.get_user_id, name='get_user_id'),
4     path('search', SearchView.as_view(), name='search'),
```

Code 4.10: An extract of Django routes

4.3.4 Database Entities

Database entities are defined as models in the `models.py` file within an app. In this file, the name of a database entity is mapped to a class in the file. The variables inside this class are used to determine the names and types of entity's fields. You can see an example of a Django model in 4.11. When an entity is created or modified, Django migrations must be created and then ran using the `manage.py` tool. Migrations are used by Django to ensure changes you make to models are executed in the database schema. ("Migrations", 2018).

```
1 class Message(models.Model):
2     message_text = models.CharField(max_length=280)
3     created = models.DateTimeField('created')
4     user = models.ForeignKey(User, on_delete=models.CASCADE)
5
6     def __str__(self):
7         return self.message_text
```

Code 4.11: The user entity in Django

4.3.5 Views

Views in Django are similar to Handlers in Yesod, they are used to determine a response for a given request. There are two main types of views that you can create in Django, class-based views and function-based views. The Django website produced contains a mixture of class-based and function-based views.

Code block 4.12 is a class-based view used to render the profile page for the current user. In this view, we check if the user is authenticated and then render the profile page for the authenticated user. If the user is not authenticated, we redirect them to the home page and show an error message.

```
1 class CurrentProfileView(TemplateView):
2     template_name = "wire_profile/current_profile.html"
3
4     def get(self, request, *args, **kwargs):
5         """
6         Get the current profile if the user is logged in.
7
8         :param request: The current request
9         :param args: sent to parent method
10        :param kwargs: sent to parent method
11        :return: Either redirect to the search page or render the profile page
12        """
13        if request.user.is_authenticated:
14            context = self.get_context_data(**kwargs)
```

```

15         form = NewWireForm()
16         context['form'] = form
17         context['user'] = request.user
18         return self.render_to_response(context)
19     else:
20         messages.error(request, 'You must log in to view your profile page',
21                        extra_tags='danger')
22         return HttpResponseRedirect(reverse('base:home'))

```

Code 4.12: Class-based current profile view

In the current profile view, we load a Django form for a user to create a message, just like we do in Yesod. This form is located in the `forms.py` file. The form, as seen in code block 4.13, is a class where variables map to input names. The Django form only requires one field, the message. No hidden fields are used to determine the user that created a message, this is done in the view itself.

```

1 class NewWireForm(forms.Form):
2     message = forms.CharField(widget=forms.Textarea(attrs={'rows': '3', 'cols':
3     '40'}), label='Message', max_length=280)

```

Code 4.13: Django message form

When the form is submitted, a post request is sent to the function-based view seen in code block 4.14. In this view, the following conditions are checked: whether or not the request type is POST, the validity of the form, and whether or not the user is authenticated. If these conditions are true, the message is created and saved to the database. If not, an appropriate error message is rendered.

```

1 def create_message(request):
2     """
3     Create a message for the logged in user
4
5     :param request: The request sent by the user
6     :return: Display the profile page with a relevant message
7     """
8     if request.method == 'POST':
9         form = NewWireForm(request.POST)
10        if form.is_valid():
11            if request.user.is_authenticated:
12                message = form.cleaned_data['message']
13                try:
14                    Message.objects.create(message_text=message,
15                    created=timezone.now(), user=request.user)
16                    messages.success(request, 'Message created successfully',
17                    extra_tags='success')
18                    return
19                except DatabaseError:
20                    messages.error(request, 'Error creating message, please
21                    contact support', extra_tags='danger')

```

```

19         return
20     HttpResponseRedirect(reverse('wire_profile:current_profile'))
    # ... each else condition renders an appropriate error message

```

Code 4.14: Function-based create message view

The method to retrieve URL parameters in Django differ depending on the type of view you use. For class-based views, URL parameter values are retrieved using the `kwargs` variable available in all class-based views, as seen in code block 4.16. For function-based views, a parameter is added to the function itself, as seen in code block 4.15.

```

1  def get_user_ids(request, user_ids):
2      """
3      Get the users with the given IDs in JSON format
4
5      :param request: The request that called this function
6      :param user_ids: The user ids to get the users for
7      :return: list of users in JSON format
8      """
9      user_ids_list = filter(bool, user_ids.split('/'))
10     user_ids_list = list(map(int, user_ids_list))
11     % users = User.objects.filter(pk__in=user_ids_list).values('username')
12     return JsonResponse(list(users), safe=False)

```

Code 4.15: Function-based view for returning user data

```

1  class SearchMessageView(TemplateView):
2      template_name = 'wire_profile/search_message.html'
3
4      def get(self, request, *args, **kwargs):
5          """
6          Render the matching messages for a search message query
7
8          :param request: The current request
9          :param args: sent to parent method
10         :param kwargs: sent to parent method
11         :return: Render the search message results page
12         """
13         query = self.kwargs['query']
14         search_results =
15         Message.objects.filter(message_text__icontains=query).all()
16         context = self.get_context_data(**kwargs)
17         context['search_results'] = search_results
18         return self.render_to_response(context)

```

Code 4.16: Class-based view to search for a given message

4.3.6 Templates

The Django template language can be used in any HTML, CSS, and JavaScript file. The Django template language can be used to perform variable interpolation, conditional checks, loops, and creating default blocks of code that can be reused in other templates. Rendering these files executes the the template logic that the file contains. (“Templates”, 2018)

```

1  {% extends "base/global/base.html" %}
2
3  {% load django_bootstrap_breadcrumbs %}
4  {% load bootstrap3 %}
5  {% load static %}
6
7  {% block breadcrumbs %}
8      {{ block.super }}
9      {% breadcrumb "Search" "wire_profile:search" %}
10 {% endblock %}
11
12 {% block title %}
13     Search
14 {% endblock %}
15
16 {% block content %}
17 <main>
18     <div class="container">
19         <div class="row">
20             <div class="col-sm-12">
21                 <form id="search-form" class="inline form-horizontal" role=form
method=post action="/search">
22                     {% csrf_token %}
23                     {% bootstrap_form form %}
24                 </form>
25             </div>
26         </div>
27     </div>
28 </main>
29 {% endblock %}

```

Code 4.17: Template file for the search page

Code block 4.17 contains the source code for the template file used to render the search page. In this file, a base template is loaded, which is a full HTML page with the content divided up into a number of blocks. These blocks are then overridden in the search template to define breadcrumbs, set the page title, and write the markup for the main content of the page. Laying out template files like this eliminates repetitive code and keeps the Django template files similar to their Yesod counterparts, reducing the need to design different templates in both frameworks.

4.3.7 Tests

Django tests are contained in the `tests.py` file within an app. Tests are functions within a class. The process for testing in Django is similar to Yesod: we create any needed database entities at the beginning of a test, make a request, and check to see if the response is what we expect. Code block 4.18 is a test where a user account is created, logged in, and then the profile page for the user is loaded. Django does not have the functionality available in Yesod that allows you to test the content of a HTML page using CSS selectors, so instead, we test that the correct template is being loaded with the expected template variables.

```

1  class CurrentProfileViewTest(TestCase):
2      # Other tests...
3      def test_current_profile_page_for_logged_in_users(self):
4          user = User.objects.create_user('testfoo', 'test@test.com', 'test')
5          self.client.post(reverse('base:verify'), {'username': user.username,
6          'password': 'test'})
7          response = self.client.get(reverse('wire_profile:current_profile'))
8
9          self.assertEqual(response.status_code, 200)
10         self.assertEqual(response.context['user'], user)
11         self.assertIsInstance(response.context['form'], NewWireForm)
12         self.assertTemplateUsed(response, 'wire_profile/current_profile.html')

```

Code 4.18: Django current profile test

In Django, we can still examine the content of a response, as seen in code block 4.19. In this block, we make a JSON request decode the response, ensuring that the response content contains the expected data.

```

1  class GetUserIdsTest(TestCase):
2      # Other tests...
3      def test_get_user_ids_one_user(self):
4          user = User.objects.create_user('foo', 'test@test.com', 'test')
5          user2 = User.objects.create_user('bar', 'bar@test.com', 'test')
6          user3 = User.objects.create_user('baz', 'baz@test.com', 'test')
7
8          response = self.client.get(reverse('wire_profile:get_user_ids',
9          kwargs={'user_ids': str(user.id) + '/'})
10         response_content = response.content.decode()
11
12         self.assertEqual(response.status_code, 200)
13         self.assertIn('"username": "' + user.username + '"', response_content)
14         self.assertNotIn('"username": "' + user2.username + '"',
15         response_content)
16         self.assertNotIn('"username": "' + user3.username + '"',
17         response_content)

```

Code 4.19: Django checking a JSON response test

Chapter 5

Evaluation

Chapter 6

Conclusion

References

- Allen, C. & Moronuki, J. (2016, July). *Haskell programming from first principles*. (Cit. on pp. 4, 8).
- Alvarez, H. (2018, April 13). Say hello to freckle education! Retrieved April 22, 2018, from <http://blog.freckle.com/say-hello-to-freckle-education>. (Cit. on p. 5)
- Bajt, A. (2014, February 23). Comparing Haskell web frameworks. Retrieved April 21, 2018, from <http://www.edofic.com/posts/2014-02-23-haskell-web.html>. (Cit. on p. 4)
- Collins, G. & Beardsley, D. (2011, January). The Snap framework: A web toolkit for Haskell. *IEEE Internet Computing*, 15(1), 84–87. doi:10.1109/MIC.2011.21. (Cit. on pp. 3, 4)
- Gamari, B. (2018). The glasgow haskell compiler. Retrieved April 22, 2018, from <https://www.haskell.org/ghc/>. (Cit. on p. 3)
- George, N. (2017, May). *Why Django? the Django book*. Retrieved January 19, 2018, from <https://djangobook.com/tutorials/why-django/>. (Cit. on p. 1)
- Kurilin, A. (2015, April 25). Haskell at Front Row. Retrieved April 22, 2018, from <https://github.com/commercialhaskell/commercialhaskell/blob/master/usage/frontrow.md>. (Cit. on p. 5)
- Getting started*. (2018). Django Software Foundation. Retrieved April 22, 2018, from <https://docs.djangoproject.com/en/2.0/intro/>. (Cit. on pp. 8, 19)
- Migrations. (2018). Retrieved April 22, 2018, from <https://docs.djangoproject.com/en/2.0/topics/migrations/>. (Cit. on p. 20)
- Templates. (2018). Retrieved April 23, 2018, from <https://docs.djangoproject.com/en/2.0/topics/templates/>. (Cit. on p. 23)
- Picciau, L. (2018, January 22). A Haskell beginner's experience with Yesod. Retrieved April 21, 2018, from <https://itscode.red/posts/a-haskell-beginners-experiance-with-yesod/>. (Cit. on pp. 4, 5)
- Snoyman, M. (2012, April). *Developing web applications with Haskell and Yesod*. O'Reilly Media. (Cit. on pp. 1, 4, 8, 14, 17).

Bibliography

- Allen, C. & Moronuki, J. (2016, July). *Haskell programming from first principles*.
- Alvarez, H. (2018, April 13). Say hello to freckle education! Retrieved April 22, 2018, from <http://blog.freckle.com/say-hello-to-freckle-education>
- Bajt, A. (2014, February 23). Comparing Haskell web frameworks. Retrieved April 21, 2018, from <http://www.edofic.com/posts/2014-02-23-haskell-web.html>
- Collins, G. & Beardsley, D. (2011, January). The Snap framework: A web toolkit for Haskell. *IEEE Internet Computing*, 15(1), 84–87. doi:10.1109/MIC.2011.21
- Gamari, B. (2018). The glasgow haskell compiler. Retrieved April 22, 2018, from <https://www.haskell.org/ghc/>
- George, N. (2017, May). *Why Django? the Django book*. Retrieved January 19, 2018, from <https://djangobook.com/tutorials/why-django/>
- Kurilin, A. (2015, April 25). Haskell at Front Row. Retrieved April 22, 2018, from <https://github.com/commercialhaskell/commercialhaskell/blob/master/usage/frontrow.md>
- Django Documentation. (2018). Retrieved April 21, 2018, from <https://docs.djangoproject.com/en/2.0/>
- Getting started*. (2018). Django Software Foundation. Retrieved April 22, 2018, from <https://docs.djangoproject.com/en/2.0/intro/>
- Hackage. (2018). Retrieved April 20, 2018, from <https://hackage.haskell.org/>
- Haskell Wiki. (2018). Retrieved April 21, 2018, from <https://wiki.haskell.org/>
- Hoogle. (2018). Retrieved April 20, 2018, from <https://www.haskell.org/hoogle/>
- Migrations. (2018). Retrieved April 22, 2018, from <https://docs.djangoproject.com/en/2.0/topics/migrations/>
- Stackage. (2018). Retrieved April 20, 2018, from <https://www.stackage.org/>
- Templates. (2018). Retrieved April 23, 2018, from <https://docs.djangoproject.com/en/2.0/topics/templates/>
- The Haskell Tool Stack. (2018). Retrieved April 21, 2018, from <https://docs.haskellstack.org/en/stable/README/>
- Yesod Cookbook. (2018). Retrieved April 21, 2018, from <https://github.com/yesodweb/yesod-cookbook>
- Yesod Google Group. (2018). Retrieved April 21, 2018, from <https://groups.google.com/forum/#!forum/yesodweb>
- Yesod StackOverflow Page. (2018). Retrieved April 21, 2018, from <https://stackoverflow.com/questions/tagged/yesod>

- Picciau, L. (2018, January 22). A Haskell beginner's experience with Yesod. Retrieved April 21, 2018, from <https://itscode.red/posts/a-haskell-beginners-experience-with-yesod/>
- Snoyman, M. [Michael]. (2018). Yesod git repository. Retrieved April 21, 2018, from <https://github.com/yesodweb/yesod>
- Snoyman, M. [Michael]. (2012, April). *Developing web applications with Haskell and Yesod*. O'Reilly Media.
- Snoyman, M. [Michael]. (2018). Yesod web framework for Haskell. Retrieved January 19, 2018, from <https://www.yesodweb.com/>
- Staub, C. (2011). *A user interface for interactive security protocol design* (Diploma Thesis, Eidgenössische Technische Hochschule Zürich, Department of Computer Science). doi:10.3929/ethz-a-007554462
- Watson, J. (2018, December 8). How does the Yesod web framework compare to more mature frameworks such as Rails and Django? Retrieved April 21, 2018, from <https://www.quora.com/How-does-the-Yesod-web-framework-compare-to-more-mature-frameworks-such-as-Rails-and-Django-Is-Yesod-stable-enough-to-develop-a-production-website-What-do-you-lose-by-going-with-Yesod-rather-than-Rails-or-Django-What-do-you-gain>

Appendices

Appendix A

Project Diary

A.1 Meeting 1 - 3rd October 2017

A.1.1 Meeting Notes:

Books

Real World Haskell

Haskell from first principles (haskellbook.com)

Web application development with Haskell and Yesod (out of date)

Frameworks / Tools

Haskell Servant package

Snap is alternative to Yesod

ghcjs haskell to js

haskell stack tool

hackage is like npm. Stack can use hackage.

Stackage is like stack on top of hackage

Use the latest LTS version of haskell from stackage

Atom could be useful with their plugins, compare with plugins available for code

ghc-mod available for haskell in atom, helpful when developing

ide-haskell, linter

There is a Haskell plugin for intellij which may work. Good because I would be familiar with the IDE.

Comparing the two frameworks

- Maintainability
 - Make a change to both
- Performance

- Scalability - could use tools, hard to do on your own
- People say Haskell is easier to write code with, less time debugging, once learnt
 - We could test this. How much the type checking helps. The different tools available
 - Can't use line by line debugging

Plan for next meeting

Do as much as possible for now

Come up with rough project definition form

Go through some haskell tutorials, haskellbook.com is recommended

A.2 Meeting 2 - 12th October 2017**A.2.1 Meeting Notes:**

Look into getting GHC mod compile on save

Get the project proposal doc ready for next week

Learn Django and get it installed on the laptop

Make a basic page in Django and Haskell

A.3 Meeting 3 - 19th October 2017**A.3.1 Meeting Notes:**

Carry on with the Haskell Programming from First principles book

Have some planning for the twitter clone ready

A.4 Meeting 4 - 24th October 2017**A.4.1 Meeting Notes:**

Set up a basic homepage in Yesod and Django. Do this over the weekend.

Have a play around with the yesod site that's provided to see what you can focus on.

Carry on with the book

Setup Docker/Vagrant if you have time at the end, for instructions on setting up the repo

Topics important for yesod

- Quasi quotes, provided by yesod
- Yesod Typeclass could be useful to know

A.5 Meeting 5 - 10th November 2017

A.5.1 Meeting Notes:

I've created the homepages in both yesod and django. I've used tests in django to test a basic app not related to the project

Next week, I want to ensure both home pages are the same and to create tests in both frameworks. I want to progress more through the yesod and haskell book. Create User models in both yesod and django and create tests for them.

A.6 Meeting 6 - 16th November 2017

A.6.1 Meeting Notes:

I've created the homepages in yesod and django and ensured that they both have the same content and styling.

For django, I have added the functionality to allow users to create accounts and log in. I have added unit tests for this and they all pass.

For yesod, I have added the latest version of jquery and bootstrap to the project. I have tried to complete the user account functionality but I am blocked. I am trying to import yesod-auth-hashdb but cannot figure out how to do it. There is some documentation showing how to edit the cabal file but this is overwritten during the build, I believe the data comes from package.yml. Editing package.yml causes strange errors when I try to build the project but I don't think I am doing it in the correct manner. Need to figure out how to edit the package.yml, edits would result in errors on my computer.

For next week, I want to fix the weird error and get some tests up.

Things to try to resolve the error, try to reproduce it on normal ubuntu. If you can't resolve it, report it to yesod.

A.7 Meeting 7 - 23rd November 2017

A.7.1 Meeting Notes:

I've resolved the random error we had last week.

I've imported hashdb and have added functionality for users to create accounts and login on the yesod site.

Yesod forms rely on bootstrap 3, so downgraded from bootstrap 4 (beta) to 3.

For next time...

I want to figure out how to concatenate a Text data variable in Yesod. Have to figure out how to deal with overloaded strings?

Finish the user authentication functionality. Show appropriate messages and add extra validation to the yesod form (unique user and email, min and max length of fields).

Create tests for the user authentication functionality.

Change the forms on Django to use their form model rather than a HTML form. This will let me compare the pros and cons of Django's and Yesod's forms.

If there is time, add functionality to allow users to post messages. These messages should be saved in the database so that the user can see all the messages they've posted when they log in.

The user post message page should use ajax so when they post a message, the part of the div will just reload rather than the whole page.

A.8 Meeting 8 - 14th December 2017

A.8.1 Meeting Notes:

On the yesod site:

Have some tests working

Users can post messages, be signed up, see other users messages

Have some tests working, this is WIP

For next time. . .

Get Django messages working

Try to get ajax working on both sites, see <https://www.yesodweb.com/blog/2013/02/ajax-with-scaffold>

Interim report plan

- Intro
- Explain the choices of yesod and django
- Do some initial comparisons of the site
- My experiences with developing on both sites, what I found easy and hard on the different frameworks.
- Advantages and disadvantages of both frameworks.

A.9 Meeting 9 - 1st February 2018

A.9.1 Meeting Notes:

Worked mainly on the Django site. I have the messages working and have began comparing features between two sites such as

- The implementation of Handlers/Routes
- The way you can pass variables to templates
- How Haskell's 'maybe' reduces the number of errors you need to catch
- the ways you can implement AJAX in both frameworks

In the near future, refactor the messages implementation to use AJAX for retrieval of messages and creating new messages. This refactoring will help compare the ease of modifiability of both of these frameworks.

Whenever you come across a difficult error, try to compare the process of debugging in both frameworks.

Remember to focus on using different parts of the framework than just implementing new features on the site.

Try to resolve the textarea problem. If you can't send a screenshot of the error.

A.10 Meeting 10 - 15th February 2018

A.10.1 Meeting Notes:

Created AJAX functionality for getting messages

Resolved issue with using single template for profile by declaring the form stuff even if `isCurrentUse` is false, this is fine

What needs to be done

Add more tests this weekend for both frameworks. Does Haskell's type checking mean we need fewer tests compared to Python?

What to look at for evaluation

Evaluate:

- The ease of writing tests
- In python, you need lots of testing because there's no static type checking
 - Does this mean you need less tests in Haskell
 - Does this mean tests are easier to write in Haskell, or in Python because tests are more important in Python so they'd be easier to use
- What types of tests are important in the haskell world
- Some tests are unneeded for Haskell
- Is it cheaper to build a bullet proof app in Haskell or Python, maintainability? etc

Amount of users in Django makes it easier to find problems that others have experienced

Amount of users in Django means more tools for Django but this is improving on the Haskell on the side

The Yesod book written by the creator of Yesod is pretty good

Some of the problems written by people using Yesod are more detailed, users are probably more experienced in the programming world? more academic?

Evaluate the ease of adding a new feature after the site is complete

Evaluate how quick it is to debug something

Built in compiler and type checking in Haskell is very useful when debugging
Evaluate page load speeds, is Python slower because it runs the interpreter every time?
Scalability if you can
Some quantitative data?
Explain to the reader why some things make a big difference

- How long it takes to get a reliable application
 - Length of tests? Number of tests? Instances where types catch important errors? These are very useful
 - Times where the type checking or other similar features got in your way?
 - * E.g. profile page was easier to code in Python, this was because of Haskell checking the scoping

A.11 Meeting 11 - 22nd March 2018

A.11.1 Meeting Notes:

Implemented type safety for some URLs

- How much does it help with testing / debugging?
- Does this reduce the amount of code you need (to check parameter types, etc.)

Joins cannot be done with Yesod Persistent (alternative pseudo SQL available) but restructuring logic may be more efficient than joins.

Plan for the holidays and beyond

- Finish the functionality of both websites by the end of week 2 of the holidays
- Produce a report by the end of week 3 of the holidays
- Get feedback and produce another version of the report the first week back from the holidays
- Get more feedback and produce a final version of the report the second week after the holidays

A.12 Meeting 12 - 19th April 2018

A.12.1 Meeting Notes:

Report draft, hand in by this weekend.

Book in a meeting on Tuesday to discuss the report.

Simulate mistakes and see how the haskell compilers help

Argue against how the compiler may make you write unnecessary code

- Haskell version, I couldn't do something like if not null, ignore the block
- You can try to use an undefined and force an exception
- You can do something like "Error - (loc) shouldn't happen"

Type safety saved a lot of time with checking variable types
But people like python because of the lack of type checking
It may be faster to ignore types when developing but it could cause problems later on
In Yesod, there's types for entity Ids, and if the entity Id does not exist, it will 404
Think about separating language vs framework
For this report, it's more useful to focus on the framework, but think about anything to do with the language
For example, static type checking is to do with Haskell rather than Yesod
In the yesod-test package, you can use HTML selectors. That makes testing static content easy.
Django, you can test HTML page but selectors are not implemented. (If html page contains this string).
If you have time after getting the report done...
Load testing, deploy both apps to the same environment. Is there a tool for it. Send multiple queries from multiple machines, time responses.
If no tool, you can try requests from just your own machine. If you can't get it on Heroku, set up a VM.
Test the ease of deployments for both frameworks. Test the actual production mode, not dev mode.

A.13 Project Definition Form

A.13.1 14th October 2017

First draught written up and sent to tutor via email for feedback

A.13.2 15th October 2017

Tutor feedback implemented

A.13.3 19th October 2017

Tutor and I signed form. Form is submitted electronically via Turnitin

A.14 Interim Report

A.14.1 18th January 2018

Interim report started and then finished

A.14.2 19th January 2018

Sent Ethics form to tutor

Proof-read and submitted the interim report

A.15 Software Development

This section records the development lifecycle of the Django and Yesod sites. More detailed commit messages can be found by running a *git log* on the git repo for both sites.

A.15.1 Yesod Site**1st November 2017**

Created a basic homepage and a vagrantfile.

23rd November 2017

Implemented Login and Signup functionality

11th December 2017

Created a profile page

14th December 2017

Implemented message creation functionality and added some tests

15th February 2018

Modified profile page to retrieve messages using AJAX

19th March 2018

Implemented functionality to show other registered users on the profile page

20th March 2018

Basic functionality to follow users implemented

21st March 2018

Refactor profile page and follow functionality to use type safe URLs

22nd March 2018

Added functionality to allow users to follow eachother through normal usage of the site. Previously, users had to type in a URL to follow a user

31st March 2018

Refactored profile page. Separated pages used for logged in and anonymous users, reducing complexity. Refactoring also increased use of type safe URLs in the profile page and when posting messages.

1st April 2018

Updated stack resolver to latest point release for current resolver, ensuring we have the latest compatible updates for our packages. Fix some bugs and add tests.

2nd April 2018

Added tests for all route handlers and integrate tests with Travis CI.

4th April 2018

Implemented search functionality

5th April 2018

Tested the search functionality

6th April 2018

Added latest posted messages to the homepage and used a previously created datatype to help display posted messages on the frontend.

7th April 2018

Added hashtag functionality and fixed some bugs on the profile page. Hashtagged messages appear on the homepage.

10th April 2018

Added yesod prefix to database names so that the Yesod server can be ran the same time as the Django server.

11th April 2018

Removed unneeded visit button on profile page.

13th April 2018

Improved ordering of messages posted by other users on your feed.

A.15.2 Django Site**9th November 2017**

Created the homepage and a vagrantfile.

14th November 2017

Implemented login and signup functionality

15th November 2017

Refactored authentication functionality to use Django's built in models rather than custom models.

31st January 2018

Implemented profile page and message creation functionality.

10th April 2018

Downgrade Django bootstrap version to bootstrap 3. This is the version used by Yesod so using version 3 reduces the amount of template work to do which does not provide much insight into the benefits of either framework.

Implemented breadcrumbs and a base template file that all other template files use.

11th April 2018

Added AJAX functionality to the profile page.

12th April 2018

Completed the profile page and added search functionality

13th April 2018

Added tests and fixed any bugs discovered. Latest message and hashtagged messages added to the homepage.

A.16 Project Diary Final Report**A.16.1 20th April 2018**

Started write-up of the final report

A.16.2 22nd April 2018

Completed report and sent to tutor for feedback

Appendix B

Ethics Form

Ethics form for student projects

SEAS group: Computer Science

Project title: Evaluation of a Haskell Web Framework

Supervisor name and email: Michal Konecny m.konecny@aston.ac.uk

Ethics questions

Please answer Yes or No to each of the following four questions:

1 - Does the project involve participants selected because of their links with the NHS/clinical practice or because of their professional roles within the NHS/clinical practice, or does the research take place within the NHS/clinical practice, or involve the use of video footage or other materials concerning patients involved in any kind of clinical practice? **No**

2 - Does the project involve any i) clinical procedures or ii) physical intervention or iii) penetration of the participant's body or iv) prescription of compounds additional to normal diet or other dietary manipulation/supplementation or v) collection of bodily secretions or vi) involve human tissue which comes within the Human Tissue Act? (eg surgical operations; taking body samples including blood and DNA; exposure to ionizing or other radiation; exposure to sound light or radio waves; psychophysiological procedures such as fMRI, MEG, TMS, EEG, ECG, exercise and stress procedures; administration of any chemical substances)? **No**

3 - Having reflected upon the ethical implications of the project and/or its potential findings, do you believe that that the research could be a matter of public controversy or have a negative impact on the reputation/standing of Aston University? **No**

4 - Does the project involve interaction with or the observation of human beings, either directly or remotely (eg via CCTV or internet), including surveys, questionnaires, interviews, blogs, etc?

Answer "no" if you are only asking adults to rate or review a product that has no upsetting or controversial content, you are not requesting any personal information, and the adults are Aston employees, students, or your own friends. **No**

Student's signature: Junaid Rasheed

Supervisor's signature: Michal Konečný