Evaluation of a Haskell Web Framework

Aston University

BSC Computer Science

Junaid Ali Rasheed

April 27, 2018

Final Year Project Definition Form

Student's name Junaid Rasheed
Course Computer Science
Project title Evaluation of a Haskell Web Framework
What is the project about? 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.
What is the project deliverable? 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.
What is original about this project? 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.
Timetable showing main stages in work plan 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

List of Figures

List of Code Blocks

Abstract

In this report, we have evaluated a Haskell web framework. We did this by comparing two websites, one made in a Haskell web framework called Yesod, and another made in a Python web framework in Django. These websites were functionally identical and were used to support our evaluation by running various experiments including page load speeds and load tests. We also compared the documentation and community support available for both frameworks. After evaluating all of our data, we came to a conclusion that Yesod is a web framework that is ready to be used in a production environment. Yesod is already being used on websites that have millions of users, and has outperformed Django in the various tests that we performed. We found that the type safety of the Haskell language will catch various bugs that may occur when developing a website, but we also acknowledge that some developers prefer the dynamic typing of a language like Python. In conclusion, we would recommend Yesod to developers who are looking for a new web framework and are either experienced in Haskell or are willing to dedicate time to learn the language.

Chapter 1

Introduction

The goal of this report is to Evaluate a Haskell Web framework. To do this, two functionally identical websites were created. One website was created using a Haskell web framework, and another was created using a Python web framework. A number of tests were then performed to compare these websites, supporting our evaluation of the Haskell web framework.

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 programming 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 evaluation 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 the popularity of Python is increasing greatly, being second only to node.js in growth over the last five years. (George, 2017). 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.

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 enabled 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 traditional 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 of the websites, comparison of how the frameworks work under load, the ease of debugging issues, the features included 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 explain the basics of our chosen frameworks, discuss previous work and research pertaining to Haskell web frameworks, and take a look at 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 (Gamari, 2018). Because of this, any Haskell web framework that uses GHC, including Yesod, will serve requests faster than most traditional web frameworks.

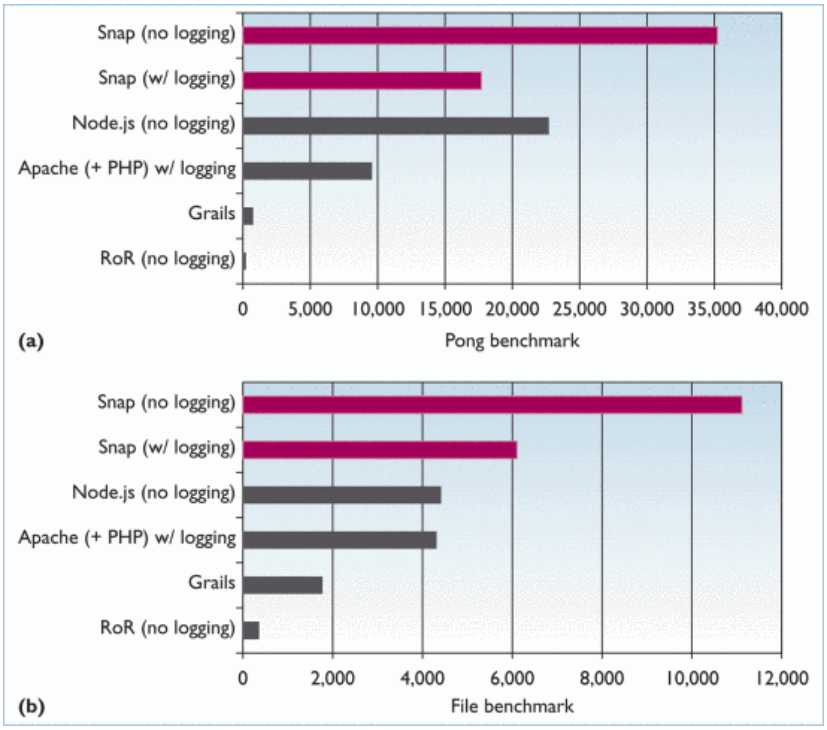


Figure 2.1: Snap and other frameworks, values are in requests per second
(Collins & Beardsley, 2011)

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, *Developing Web Applications with Haskell and Yesod*, 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, founding 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 also 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 who maintain 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 each framework is tested and evaluated fairly. 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. 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 design 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, 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 the next chapter.

Chapter 4

Deliverables

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. After describing the websites and both frameworks, we will discuss the experiments that we ran on our websites, and evaluate the performance of each framework.

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

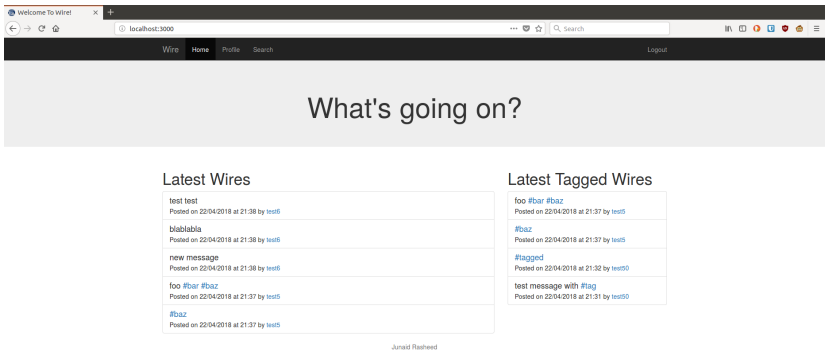


Figure 4.1: The home page

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

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. 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. 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

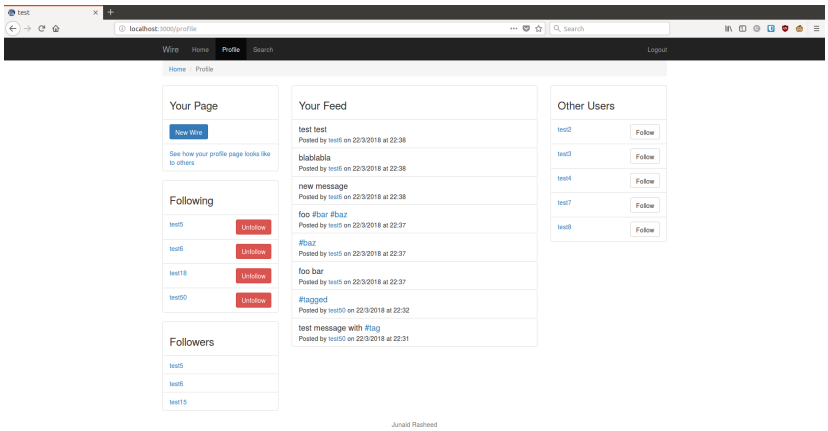


Figure 4.2: The profile page for the current user

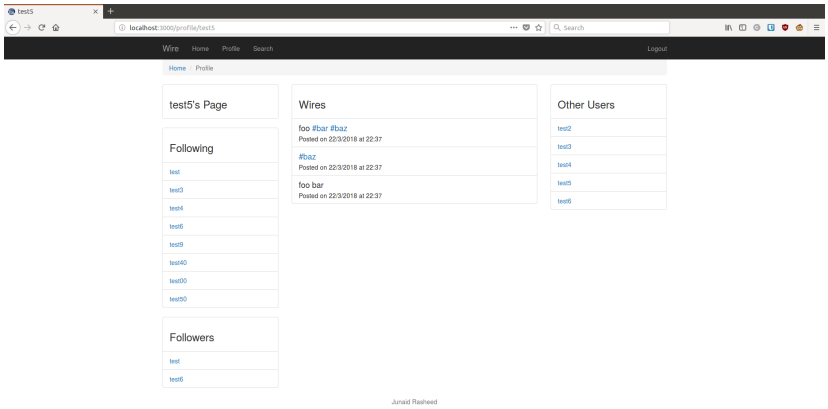


Figure 4.3: The profile page for other users

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 4.2 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 with an input box for the user to enter their message.

Figure 4.3 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

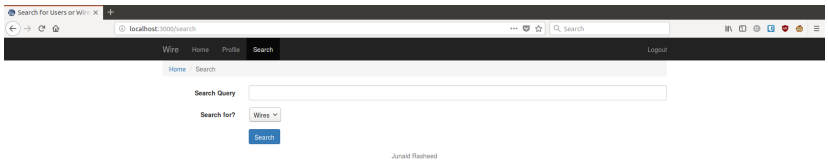


Figure 4.4: The search page

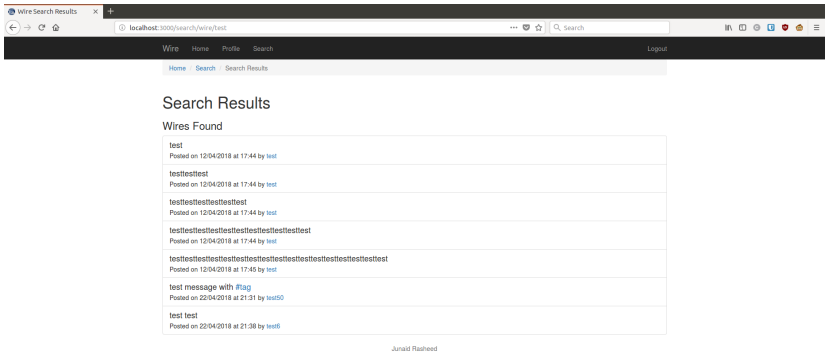


Figure 4.5: The search results page for messages

Figure 4.4 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 Figure 4.5.

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 An Overview

Figure 4.6 gives you a high level overview of how Yesod dispatches a request. By dispatch, we mean taking an incoming request and then generating an appropriate response (Snoyman, 2012, Understanding a Request).

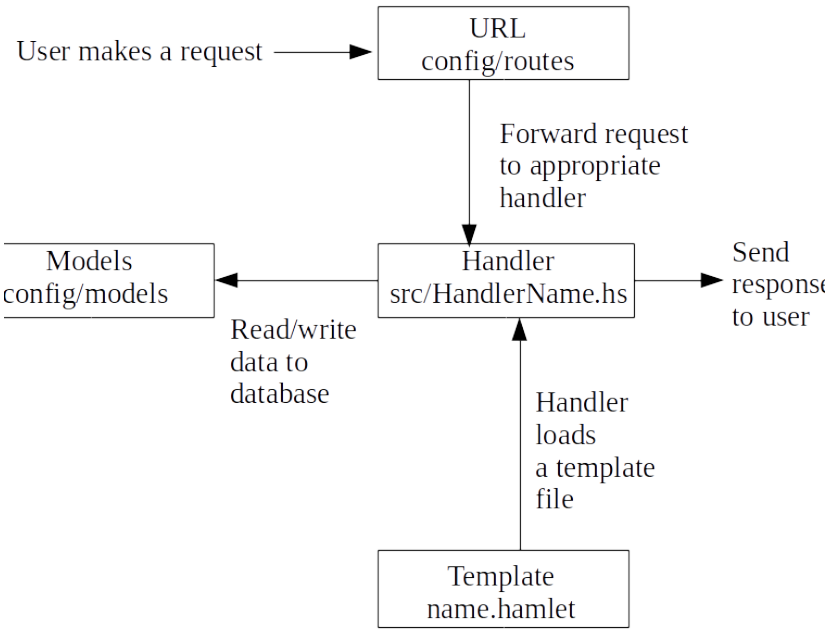


Figure 4.6: An overview of Yesod, from the Request to the Response

In the figure above, you can see that all requests go to the `config/routes` file. This file is used to specify valid routes in an application, and will forward all requests to an appropriate Handler. (Snoyman, 2012, Understanding a Request)

A Handler is a more general term for a web application. Handlers are Haskell files stored in the `src/` directory. These handlers are used to interact with database entities, load an appropriate template file, and then send a response to the user. (Snoyman, 2012, Understanding a Request)

The models file located in `config/models` specifies all the database entities in this web application. Each entry in this file generates a database table and also creates helper functions for you to use in your Haskell code. The handlers normally interact with these entities to deal with certain requests. For example, if the user wants to log in to a website, the request will go to the `Login` handler, which will check if the credentials provided by the user is correct, and then generate an appropriate response.

When a Handler needs to response with a HTML page, a template file is normally used. A template file is a file with syntax similar to HTML that defines the content of a HTML page.

4.2.3 Defining Routes

As explained previously, 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.

```
-- This code block is an extract of the routes file. Each route is split into
-- three parts. The first part is the actual URL route that the user will
-- type. This route can contain a parameter which is mapped to a value from the URL.
-- The second part specifies a Haskell module that will deal with a request
-- from the given URL. The third part specifies the request types valid at this URL.
/profile MyProfileR GET POST
/profile/#Text ProfileR GET

/user UserGetAllR GET
/user-not-following UserGetAllExcludingFollowingR GET
/user/#Text UserGetAllExcludingUsernameR GET
/user/id/#UserId UserGetIdR GET
/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`, is for viewing the profile page of another user. 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/`.

4.2.4 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.

```
User json key -- Specify the name of the entity and some helper functions
  username Text Eq -- Specify entity fields and their types
  email    Text Eq
  password Text
  UniqueUser username
  UniqueEmail email
  deriving Typeable Show -- Some Haskell helper functions
```

Code 4.2: Yesod Database Entities

4.2.5 Handlers

Every URL route in the application points to a handler. Handlers 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, and then loads the template file. The variables that are available in the handler are also available in the template file.

```haskell
-- Loads the 'My Profile' page for the currently logged in user. If someone
-- who is not logged in attempts to access the page, a 404 error will be shown.
getMyProfileR :: Handler Html -- This function will respond to GET requests to the profile
  ↪ page
getMyProfileR = do
  (Entity userId user) <- requireAuth -- Get the current user's data from the database
  let username = userUsername user -- Extract username from the user

  -- Here, we generate a Haskell form to be used in a template file
  (formWidget, formEnctype) <- generateFormPost $ messageForm userId
  defaultLayout $ do -- Load the default layout that wraps around the loaded template
    setTitle . toHtml $ userUsername user -- Set the title of the profile page
    $(widgetFile "currentprofile") -- Load a template file
```

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.

```haskell
-- Here, we create a Bootstrap3 form for the Message entity
messageForm :: UserId -> Form Message
messageForm userId = renderBootstrap3 BootstrapBasicForm $ Message
  <$> areq textField (bfs ("Message" :: Text)) Nothing -- A required text field
  <*> pure userId -- A hidden field containing the userId
  <*> lift (liftIO getCurrentTime) -- A hidden field containing the current time
```

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.

```haskell
-- Create a new wire for the logged in user
postMyProfileR :: Handler Html
postMyProfileR = do
  (Entity userId _) <- requireAuth
  ((result, _), _) <- runFormPost $ messageForm userId -- Get the message form from the
  ↪ post request
  case result of -- Check whether or not the message form is valid
    FormSuccess message -> do
      _ <- runDB . insert $ message -- Insert the message typed in by the user
      setSession "msgrendered" "true"
      setMessage $ renderSuccessMessage "Wire Sent" -- Set a message to be shown to the
  ↪ user
      redirect MyProfileR -- Reload the profile page
    FormFailure errors -> do
      let renderedMessages = map renderErrorMessage errors -- Collect all the error
  ↪ messages
      setSession "msgrendered" "true"
      setMessage $ toHtml renderedMessages -- Set the error messages to be shown to the
  ↪ user
      redirect MyProfileR
    FormMissing -> do
      setSession "msgrendered" "true"
      setMessage $ renderErrorMessage "Form is missing"
      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.

```haskell
-- | Takes in a user id and returns data on the user matching the given id.
-- If no user is found, an empty JSON object is returned.
getUserGetIdR :: UserId -> Handler Value
getUserGetIdR userId = do -- Here, userId is a parameter from the URL
  users <- runDB $ selectList [UserId ==. userId] [] -- Load the user with the given user id from the database
  -- The map in the line below extracts the username from the user object and
  -- stores this in a new object called cleanUsers. This ensures that the
  -- user's email and (hashed) password is not given in the response.
  let cleanUsers = map (\(Entity uid (User uname _ _)) -> (object ["id" .= uid, "username" .= uname])) users
  returnJson cleanUsers
```

Code 4.6: GET request handler for getting user data

4.2.6 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 syntax changes. 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 `^{widgetName}`, which renders the given widget onto the page. Type safe URLs are loaded using `@{routeName optionalParameters}`.

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

Code 4.7: Template file for the search page

4.2.7 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.

```
it "asserts that the current profile page looks right" $ do
  foo <- createUser "foo" "foo@bar.com" "foo"
  authenticateAs foo

  get MyProfileR
  htmlAnyContain "h3" "Your Page"
  htmlAnyContain "h3" "Your Feed"
  htmlAnyContain "h3" "Followers"
  htmlAnyContain "h3" "Following"
  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.

```
it "asserts all users are returned when not authenticated" $ do
  _ <- createUser "foo" "foo@bar.com" "foo"
  _ <- createUser "bar" "bar@bar.com" "foo"
  _ <- createUser "baz" "baz@bar.com" "foo"

  get UserGetAllR

  bodyContains "username"
  bodyContains "id"
  bodyNotContains "email"
  bodyNotContains "password"
  bodyContains "foo"
  bodyContains "bar"
  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 An Overview

This subsection will give you a high level overview of how Django works. Figure 4.7 shows you how Django generates a response from a user's request. In the diagram, you can see that all requests first go to the `urls.py` file. In this file, a request is matched with an entry in the file, and this entry forwards the response to an appropriate view.

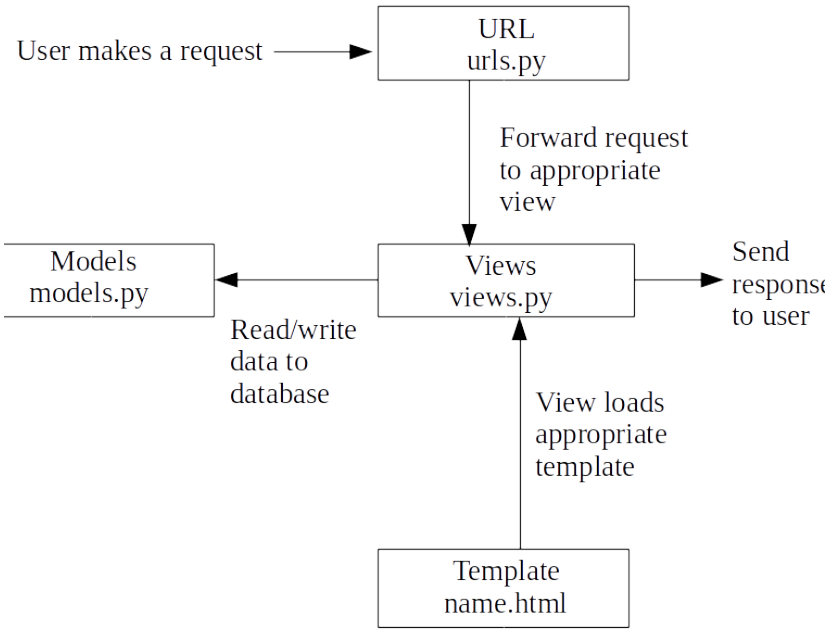


Figure 4.7: An overview of Django, from the Request to the Response

A view is python function that takes in a request and generates an appropriate response. A valid response could be a HTML page, an HTML error, JSON data, etc. To generate an appropriate response, a view may interact with a model, or it may need to load a template file. Views are defined in the `views.py` file. ("Writing views", 2018).

Models, located in `models.py` are used to define database entities in an application. Each entry is written as a class and is mapped to a table in the database. Views interact with these models to read or write data to the database.

Template files are simply files written in HTML that define the content of a page. Views use template files to render an appropriate response if the response is a HTML page.

4.3.4 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:parameterName>` can be used. An example of Django routes can be seen in code block 4.10.

```
# The path function below takes in three variables. The first
# variable is the URL path for the request, the second parameter
# links to a view that will handle a request, and the third
# parameter is the name for the route that can be used in Python code
path('following/<path:username>', views.get_following, name='get_following'),
path('users/<path:user_ids>', views.get_user_ids, name='get_user_ids'),
path('user/id/<int:user_id>', views.get_user_id, name='get_user_id'),
path('search', SearchView.as_view(), name='search'),
```

Code 4.10: An extract of Django routes

4.3.5 Database Entities

Database entities are defined as classes in the `models.py` file within an app. The variables inside each 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).

```
class Message(models.Model):
  message_text = models.CharField(max_length=280)
  created = models.DateTimeField('created')
  user = models.ForeignKey(User, on_delete=models.CASCADE)

  def __str__(self):
    return self.message_text
```

Code 4.11: The user entity in Django

4.3.6 Views

Views in Django are similar to Handlers in Yesod, they are used to create 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.

```
class CurrentProfileView(TemplateView):
  template_name = "wire_profile/current_profile.html"

  def get(self, request, *args, **kwargs):
    """
    Get the current profile if the user is logged in.

    :param request: The current request
    :param args: sent to parent method
    :param kwargs: sent to parent method
    :return: Either redirect to the search page or render the profile page
    """
    if request.user.is_authenticated:
      context = self.get_context_data(**kwargs) # context is a map of objects that can be
    ↪ used in the template
      form = NewWireForm()
      context['form'] = form
      context['user'] = request.user
      return self.render_to_response(context)
    else:
      messages.error(request, 'You must log in to view your profile page', extra_tags='
    ↪ danger')
      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.

```
class NewWireForm(forms.Form):
  message = forms.CharField(widget=forms.Textarea(attrs={'rows': '3', 'cols': '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.

```
def create_message(request):
  """
  Create a message for the logged in user

  :param request: The request sent by the user
  :return: Display the profile page with a relevant message
  """
  if request.method == 'POST':
    form = NewWireForm(request.POST) # Get the form data from the request
    if form.is_valid():
      if request.user.is_authenticated:
        message = form.cleaned_data['message'] # Retrieve the message from the form data
        try:
          Message.objects.create(message_text=message, created=timezone.now(), user=request
↪ .user) # Save the message in the database
          messages.success(request, 'Message created successfully', extra_tags='success')
          return HttpResponseRedirect(reverse('wire_profile:current_profile'))
        except DatabaseError:
          messages.error(request, 'Error creating message, please contact support',
↪ extra_tags='danger')
          return 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 differs 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.

```
def get_user_ids(request, user_ids):
  """
  Get the users with the given IDs in JSON format

  :param request: The request that called this function
  :param user_ids: The user ids to get the users for
  :return: list of users in JSON format
  """
  user_ids_list = filter(bool, user_ids.split('/'))
  user_ids_list = list(map(int, user_ids_list))
  % users = User.objects.filter(pk__in=user_ids_list).values('username')
  return JsonResponse(list(users), safe=False)
```

Code 4.15: Function-based view for returning user data

```
class SearchMessageView(TemplateView):
  template_name = 'wire_profile/search_message.html'

  def get(self, request, *args, **kwargs):
    """
    Render the matching messages for a search message query

    :param request: The current request
    :param args: sent to parent method
    :param kwargs: sent to parent method
    :return: Render the search message results page
    """
    query = self.kwargs['query']
    search_results = Message.objects.filter(message_text__icontains=query).all()
    context = self.get_context_data(**kwargs)
    context['search_results'] = search_results
    return self.render_to_response(context)
```

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

4.3.7 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)

```
{% extends "base/global/base.html" %}

{% load django_bootstrap_breadcrumbs %}
{% load bootstrap3 %}
{% load static %}

{% block breadcrumbs %}
  {{ block.super }}
  {% breadcrumb "Search" "wire_profile:search" %}
{% endblock %}

{% block title %}
  Search
{% endblock %}

{% block content %}
<main>
  <div class="container">
    <div class="row">
      <div class="col-sm-12">
        <form id="search-form" class="inline form-horizontal" role=form method=post action=
    ↪ "/search">
          {% csrf_token %}
          {% bootstrap_form form %}
        </form>
      </div>
    </div>
  </div>
</main>
{% 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.

4.3.8 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.

```
class CurrentProfileViewTest(TestCase):
  # Other tests...
  def test_current_profile_page_for_logged_in_users(self):
    user = User.objects.create_user('testfoo', 'test@test.com', 'test')
    self.client.post(reverse('base:verify'), {'username': user.username, 'password': 'test'
    ↪ })
    response = self.client.get(reverse('wire_profile:current_profile'))

    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.context['user'], user)
    self.assertIsInstance(response.context['form'], NewWireForm)
    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, and ensure that the response content contains the expected data.

```
class GetUserIdsTest(TestCase):
  # Other tests...
  def test_get_user_ids_one_user(self):
    user = User.objects.create_user('foo', 'test@test.com', 'test')
    user2 = User.objects.create_user('bar', 'bar@test.com', 'test')
    user3 = User.objects.create_user('baz', 'baz@test.com', 'test')

    response = self.client.get(reverse('wire_profile:get_user_ids', kwargs={'user_ids': str
    ↪ (user.id) + '/'}))
    response_content = response.content.decode()

    self.assertEqual(response.status_code, 200)
    self.assertIn('"username": "' + user.username + '"', response_content)
    self.assertNotIn('"username": "' + user2.username + '"', response_content)
    self.assertNotIn('"username": "' + user3.username + '"', response_content)
```

Code 4.19: Django checking a JSON response test

4.4 Comparison of Django and Yesod

This section consists of a comparison of both frameworks. A series of experiments were conducted as specified in the plan, the raw results of which can be found in appendix A. These results will be discussed along with features and limitations of both frameworks that stand out.

To ensure that the results we collected were as fair as possible, two Amazon EC2 servers were created. The configuration of both servers were identical, with them having 1GB of RAM and 1 CPU core available. The tools necessary to deploy the website were installed on both servers and both sites were released on the server, allowing testing to begin.

4.4.1 Deployment

Deploying a Yesod onto an Amazon EC2 app is very simple. Yesod has built-in support for a tool called Keter, a Haskell application that can act as a web server. After Keter was installed and configured on the Amazon EC2 server, a Keter binary was created using commands built-in to Yesod. This binary file can simply be transferred onto the server, and Keter will start serving the web application to visitors. (Snoyman, 2012, Deploying your Webapp)

For the Django website, nginx, an open source web server was installed. nginx acts as a reverse proxy to the Django application, which is being run by a tool called Gunicorn ('Green Unicorn'). Gunicorn is a WSGI ('Web Server Gateway Interface') tool that can be used on Unix systems. ("How to use Django with Gunicorn", 2018) This means that Gunicorn can allow nginx to use Django to serve the web application.

4.4.2 Page Load Speed

When testing page load speeds, each page was loaded three times. The time taken for each run was recorded as well as an average. In this section, we will be discussing the average times that were recorded. The raw results can be found in section A.1 in the appendix.

Page load speeds for Yesod are noticeably quicker than Django, with Yesod loading most pages around 200ms faster. For example, on average, the home page takes about 511ms to load in the Yesod framework, and 750ms to load in the Django framework. Creating a new message and then redirecting back to the profile page takes about 679ms in Yesod and 849ms in Django. See table 4.1 for a full list of results.

Table 4.1: Average Page Load Speeds

Page	Average Speed in Yesod (ms)	Average Speed in Django (ms)
Home Page	511.00	753.33
Search Page	517.33	756.33
Login Page	443.67	821.33
Signup Page	490.33	764.00
Creating an Account	504.33	748.67
Logging in to an Account	547.33	722.67
Logging out	510.33	761.33
Current user's Profile Page	617.00	930.00
Other user's Profile Page	651.33	908.67
Creating a Message	679.33	848.67
Search for a Message	513.33	766.33
Search for a User	519.00	756.67

As you can see in table in table 4.1, Yesod consistently outperforms Django. This could be for a number of reasons: Yesod automatically minimises static files like CSS or JavaScript; Haskell is compiled rather than interpreted, and compiled code generally runs faster than interpreted code; the

lazy evaluation of Haskell may give some speed improvements; and the Keter tool used on the server is made specifically for Haskell web applications, which may mean that Yesod and Keter will run faster than Django and nginx.

4.4.3 Load Tests

Load tests were conducting using a tool called RedLine13. RedLine13 is a service that allows users to perform load testing using Amazon EC2 servers. With RedLine13, I was able to send requests from an Amazon EC2 server to the Yesod and Django websites. To keep within the usage limits of RedLine13 and Amazon EC2, three load tests were conducted. Each test had 80 users load a specific page in a short amount of time, typically 25 seconds. The results of these tests can be found in table 4.2.

Table 4.2: Load Testing Page Load Speeds

Page	Yesod (s)	Django (s)
Home	4.96	5.54
Profile 1st Run	5.05	4.94
Profile 2nd Run	4.97	5.13
Average	4.99	5.20

As you can see in the table above, Yesod is around 200-500ms faster than Django when under load. This difference is consistent with the results found in section 4.4.2. The high loading times seen in these tests is probably because the servers are being are not very powerful.

The tests also recorded the total amount of data downloaded for all users. Yesod beat Django by a sizeable margin in this test, with Yesod sending, on average, 6.46MB of data per test and Yesod sending 18.19MB of data. This is because Yesod, by default, compresses all static files. If we load the home page of both websites, Yesod transfers 76.83kB of data which is decompressed to 246.57kB being stored on the disk. Django, on the other hand, transfers 243.49kB of data.

4.4.4 Resource Usage

Resource usage on both servers was measured after running the experiments detailed in section 4.4.2. All the information in this section was obtained by examining output from htop, a process viewer. On the server running Yesod, 109MB of RAM was being used. The server running Django used 125MB of RAM.

To run a web server on the Yesod server, Keter creates two sub-processes. One of these sub-processes loads the compiled code for the website, and the other loads the configuration file for the web server. Altogether, these three processes use about 83MB of RAM in total, with 56MB of RAM being shared.

On the Django server, gunicorn, when executed with the default settings, creates one sub-process. These two processes use around 65MB of RAM and share 19MB of RAM. nginx is not very memory

intensive, using 6MB of RAM, sharing 3MB. In total, the Django server uses 71M of RAM, sharing 22MB.

Yesod is ahead of Django when it comes to resource usage. Even though all the Keter processes use more memory than gunicorn, most of the memory used by Keter is shared, resulting in the overall memory usage in the Yesod server being lower than the Django server. The RAM usage of the Django server is 16MB more than the Yesod server, but this difference is negligible considering the resources available to most servers at this time.

4.4.5 Continuous Integration

For the duration of this project, all programming has been done on a git repository stored on GitHub. Travis CI, a continuous integration tool, syncs with GitHub in order to run tests on the repository every time there is a new commit. The way this works is that every time a commit is pushed to GitHub, GitHub sends the details of the commit to something called a Webhook. Travis gets an update from this Webhook, and executes a file called `.travis.yml` placed in the root of the repository. This file contains instructions telling Travis how to run the tests on the codebase.

Yesod did take longer to run tests when compared to Django. The Django repository took around 2.5 minutes to build and run all the tests. Yesod took around 3.5-4 minutes. For continuous integration, this difference is negligible as any code will be peer reviewed in a real life before being merged in to the main branch. For developing however, Django does speed things along when you're implementing a new feature in a test driven way and are running the test suite multiple times to ensure your feature works and does not unintentionally break other features.

4.4.6 Debugging

When developing, you sometimes make mistakes. When you make these mistakes, ideally, you would want to see an error message that tells you where you made the mistake and some information about the error itself which may help you resolve this mistake. As part of evaluating Yesod and Django, simulated mistakes were made in both frameworks.

The first simulated mistake we made was, when creating a message, try to save the form data object into the database rather than the actual message stored inside this object. The code change in Yesod can be seen in code block 4.20, and the Django change can be seen in code block 4.21.

```
1 (Entity userId _) <- requireAuth -- get the user id
2 ((result, _), _) <- runFormPost $ messageForm userId -- get the form data
3 case result of
4   FormSuccess message -> do -- if it's a valid form, get the message
5     -- _ <- runDB . insert $ message -- original line, insert message
6     _ <- runDB . insert $ result -- new line, insert form data
```

Code 4.20: Yesod Code Change

```
form = NewWireForm(request.POST) -- get the form
if form.is_valid():
  if request.user.is_authenticated:
    message = form.cleaned_data['message'] -- form.cleaned_data is a map of form values
    try:
      # Message.objects.create(message_text=message, ..) # original line, store the message
      Message.objects.create(message_text=form.cleaned_data, ..) # changed line 1, store
    ↪ form values
      # Message.objects.create(message_text=form, ..) # changed line 2, after previous line
    ↪  passed, store form object
```

Code 4.21: Django Code Change

For Yesod, the simulated error caused a compilation error, with the exception message complaining about mismatched types, as seen in code block 4.22. For Django, however, no exception was thrown, even when submitting the message. Further investigation showed that Python was converting the form data into a string, and then saved this string in the database. Because the message was saved in the database, the tests, which at the time only checked the amount of objects in the message table, passed. This test was later amended to check the actual content of the message, and failed appropriately when the mistake was reintroduced.

```
- Couldn't match type 'PersistEntityBackend (FormResult Message)'
with 'SqlBackend'
arising from a use of 'insert'
- In the second argument of '(.)', namely 'insert'
In the expression: runDB . insert
In a stmt of a 'do' block: _ <- runDB . insert $ result
```

Code 4.22: Yesod Exception Message

In the second test, we simply misspelled a variable name. This would normally be caught by most editors but it would be useful to see the error message produced as a result of a particularly common mistake. This mistake was done in the piece of code that returns data for recommended users in JSON format. This is used as part of an AJAX request to display recommended users to the user on the profile page. See code block 4.23 for the Yesod change and code block 4.24 for the Django change.

```
Entity userId user <- requireAuth
followers <- runDB $ selectList [FollowFollowerId ==. userId] []
-- See: https://stackoverflow.com/questions/36727794/haskell-persistent-reusing-selectlist
let followingIds = map (\(Entity _ (Follow _ followingId)) -> followingId) followers
users <- runDB $ selectList [UserUsername !=. userUsername user, UserId /<-. followingIds]
  ↪ [LimitTo 5]
let cleanUsers = map (\(Entity uid (User uname _ _)) -> (object ["id" .= uid, "username" .=
  ↪  uname])) users
-- returnJson cleanUsers -- original line
returnJson cleanUser -- new line
```

Code 4.23: Yesod Code Change

```
if request.user.is_authenticated:
follow_query = Follow.objects.filter(follower_id=request.user)
users = User.objects.filter().exclude(id=request.user.id).exclude(username=
  ↪ excluded_username)\
  .exclude(followed_user__in=follow_query).values('username')[:5]
# return JsonResponse(list(users), safe=False)  # original line
return JsonResponse(list(user), safe=False)  # new line
```

Code 4.24: Django Code Change

The change caused a compilation error in Yesod. The exception message complained about the misspelled variable not being in scoped, and actually recommended the correct variable that should have been used. For Django, the recommended users section did not load in the Profile page. Checking the network tab of the web developer tool built into the browser showed that the AJAX request responded with a 500 error. Loading the URL used in the AJAX request displayed an exception, with a stack trace and a message that said "name 'user' is not defined".

```
Variable not in scope: cleanUser
Perhaps you meant 'cleanUsers' (line 19)
```

Code 4.25: Yesod Exception Message

4.4.7 Documentation

The book, *Developing Web Applications with Haskell and Yesod*, will teach you almost all the features of the Yesod framework. The book has a lot of detail of how every feature in Yesod works, including templates, database entities, routing, and deployment. If the reader is familiar with Haskell, the book will give you the base knowledge needed in order to start developing a Yesod website.

The problem with Yesod is that outside of topics in the book, the documentation of libraries used is, most of the time, not very detailed. Most of the documentation for these libraries are a couple of lines explaining what a function does and a Haskell type signature. For more experienced Haskell programmers, most of the time, this is all you need to figure out how to use a function. You can also examine the source code if you need more information on how a function works. However, for developers who are new to Haskell, examples and detailed explanations like the one given in the book are invaluable. This resulted in a lot of time being spent towards the beginning of the project trying to fix errors that would be trivial for more experienced developers. As the project progressed further, the documentation and type signatures provided by external libraries became easier to understand as knowledge of Haskell increased.

The Django project also has excellent documentation. There are tutorials for beginners to get started with Yesod, tutorials for deploying Django websites, and documentation on almost all functions in the framework. Most of the documentation also contain usage examples which are invaluable for beginners who want to easily see how a function works. This is a definite advantage that Django has over Yesod. The Yesod book does have great usage examples but the documentation for external libraries is lacking.

4.4.8 Community

Django has millions of users. This means that any issue you come across, someone else has most likely already encountered and solved. This means that a lot of the time, when you come across an issue, you can just search for the error message and come across forum posts discussing solutions for the exact same error that you are having.

Yesod, on the other hand, has a much smaller community. It is likely that you are the only person experiencing a certain issue. This was certainly the case for when Freckle migrated to Yesod, as discussing in chapter 2. Because of this, beginners will likely find it harder to solve issues with Yesod when compared to Django.

Although the Haskell community is small, experienced developers are almost always available to help beginners with issues if they make a post on an appropriate forum. In fact, with Yesod, you often see Michael Snoyman, the person who wrote the framework, answering questions posted by beginners. Developers can also join the #haskell-beginners IRC channel where they can speak to an experienced developer about their issues in real time.

4.4.9 Recommendations

After running all of our tests, we can see that a relatively niche framework like Yesod can keep up with a giant like Django. If you're an experienced Haskell developer and are looking for a web framework to use for your project, then Yesod is a great choice. The documentation for Yesod itself is excellent, it is being used in the real world on websites that get millions of visitors, and the community is great when you are experiencing an issue.

Performance-wise Yesod is faster and less resource intensive than Django. This is most likely because of the fact that Haskell is a compiled language, and compiled languages are generally faster than interpreted languages, like Python. If your primary concern when it comes to choosing a web framework is performance, then Yesod is a great choice.

For Haskell beginners or people who have no experience in Haskell, you should only decide to use Yesod if you are willing to dedicate a lot of time to learn Haskell and the framework itself. Yesod is a nice framework to use if you're an experienced Haskell developer, but it can be frustrating for beginners. You will find it hard to understand why your code won't compile, some of the documentation of external libraries is poor, and you will not understand how a lot of the advanced Haskell features being used in Yesod work. If you need to find a web framework to start working on right away, then Yesod is not for you. If you are willing to learn Haskell and the framework itself, then Yesod is a great choice. The Haskell community is friendly towards beginners and will be willing to help you if you are experiencing issues, and developing a Yesod application is a great way to develop your Haskell knowledge.

As shown in the tests where we simulated common development errors, Yesod is very robust. It will not compile at all if there is a mismatched type, saving you from having to write tests to detect when a mismatched type occurs. Because Python uses type coercion, Django is more flexible in this regard. This may save time when developing but it may cause unexpected errors as seen in our tests. This is a matter of personal preference, some developers will prefer the flexibility that Python's dynamic types give you, and others will prefer the robustness of Haskell's static types and type safety.

In conclusion, Yesod is a production ready framework. It is used in the real world, can keep up

and sometimes outperform other popular frameworks, has great documentation, and has a helpful community.

Chapter 5

Evaluation

In this chapter, we will discuss the websites that we created and the quality of our comparison of Yesod and Django. We will look at the methods that we used to test each framework, whether or not these methods could be improved, and recommendations for further comparisons

5.1 The Websites

The websites that were created were functionally identical. These websites were usable, fully functional, and responsive. The websites allow users to create accounts, interact with forms, have their own profile pages, and search for other users or messages. These features are used by many websites in the real-world and by implementing these features, we were able to realistically evaluate both frameworks.

There is one feature that was discussed in the project plan but was not included in the final implementation of our websites, and that was the messaging feature. This feature was planned to be implemented towards the end of development and would have evaluated the difficulty of making a large change to both frameworks. Due to time constrains, it was decided to abandon this feature and instead, run some measurable tests to quickly and reliably obtain some concrete data.

5.2 Site Hosting

Both of the websites were hosted on Amazon EC2 servers located in the same place with identical hardware specifications. This ensured that the only differing factor in our tests was the framework being used, giving us fair results. There is, however, one issue with the servers used, they were very underpowered, with 1GB of RAM and 1 CPU core. This may have been a limiting factor when measuring the performance of the frameworks, affecting our results. It would have been ideal to use a powerful server for our tests, but choices were limited with no budget available to host a more powerful server.

5.3 Testing Methods

This section will go through each experiment that was performed on the frameworks. We will discuss the merits of the test, the reliability of our results, and any way we could improve the test.

5.3.1 Page Load Speed Tests

These tests were performed on identical servers and were repeated three times. The time taken to load the page was taken from Chromium's developer tools under the network tab. An average was measured and this value was used for the actual comparison. This ensured that the results we measured were reliable and accurate. Measuring page load speeds is also an excellent way of measuring the performance of a web framework, as keeping page load speeds as low as possible is very important in order to ensure visitors do not get annoyed at having to wait long periods of time to load a page.

The reliability of results could have been improved by performing more repeats of our experiments. We could have measured the load time of specific AJAX requests to see if any stand out. Using servers with different specifications would also have been useful in order to see how more or less powerful hardware affects page load speeds.

5.3.2 Load Tests

Load tests were performed using a free service called RedLine13. With RedLine13, we were able to use a free Amazon EC2 instance to perform load tests. Unfortunately, these instances were limited to 80 users so we were not able to perform a large load test over a number of hours. We also only repeated this experiment three times in order to stay within Amazon's usage limits.

The results that we obtained from this experiment confirmed what we found in the Page Load speed experiment, Pages in Yesod load 200-500ms faster than pages in Django. This experiment also resulted in high page load times of around 5 seconds. This is most likely because of how weak the hosting servers are. Using a powerful server would have been very useful to see how each framework works under very heavy loads.

This experiment also told us that Yesod requests send much less data than Django due to the way Yesod minimises all static files by default. Using less data is a very desirable feature for web frameworks, as website visitors may be on limited plans and would prefer not to use websites that send a lot of data to their devices.

5.3.3 Resource Usage

This value was only measured once after the page load speed experiments were complete. Because of this, this value is not very reliable. Repeating our measurements and calculating an average would definitely increase the reliability of this result. It would have also been useful to measure this value every hour for an extended period of time, say, 24 hours. This would tell us how memory usage varies over time, and would tell us if there's any bug in the framework that may cause memory leaks.

The value that we recorded did tell us that resource usage is very similar in both frameworks, with Yesod coming out on top. This is a useful value for readers to know as some users will be mindful of resource usage when choosing a web framework for their project.

5.3.4 Continuous Integration

The values we got for this were from Travis CI's web interface. Travis CI was first integrated with the Yesod repository. It was added to the Django repository at a later date. This meant that there are much more build times in the Yesod repository compared to the Django repository. It would have been useful to have a similar number of builds to allow us to calculate a reliable average for build times.

The value itself may be useful for developers who are concerned with long build times, but builds are completed in under five minutes for both repositories. This informs readers that testing in both frameworks does not take a significant amount of time, which would be a concern for developers who adopt a test driven methodology.

5.3.5 Debugging

The simulated errors we performed in this test proved how the Haskell compiler can save the developer a lot of time when developing a website. In the first test, the compiler caught a bug where a new message was being saved using the form data itself rather than the message from the form data. This same bug was not caught by the Python interpreter. In fact, Python automatically converted the form data, which was a map, into a string, and stored the value in the database. A test case had to be added to ensure that this would be caught if a similar mistake would occur in the future.

The second test, misspelling a variable name, also highlighted how the Haskell compiler can help developers quickly figure out how to fix their mistakes. The Haskell compiler printed a message stating the actual variable name that was misspelt. In this case, the Python interpreter also printed an appropriate exception message which mentioned that the variable used was undefined.

5.4 Further Comparisons

A refactor towards the end of development would have been very useful in our comparison. It would have given us information on how the Haskell compiler can guide you during a refactor. In theory, if you start the refactor as a small change and then compile your program, the Haskell compiler should guide you on where further changes need to be made. For example, if we decide to add a new field to the message entity, the Haskell compiler should tell us where the message entity is being used, helping us efficiently refactor our program.

Under the free usage limits of Amazon Web Services, available EC2 servers are not very powerful and the monitoring tools are only updated every five minutes. Because of this, only very limited load testing could be performed, with a maximum of 80 users. Monitoring tools are also only updated every five minutes under the free plan, making it infeasible to monitor load usage during a test. These tests did tell us the amount of time it took to load a page and the total amount of data transferred, but it would have been useful to know how the web frameworks handle longer load tests. Longer load tests would have also told us how each framework manages resources during extended periods of load. These tests may have identified issues such as memory leaks if they were able to be conducted.

Chapter 6

Conclusion

In this report, we have discussed how we evaluated a Haskell web framework. We compared two frameworks, a Haskell framework called Yesod, and a Python framework called Django. We gave a high-level explanation of how the two frameworks worked and explained the preparation needed before development could be started in both frameworks. We also discussed the process of creating a website in both frameworks

Once the websites were complete, we performed tests to compare each framework. These tests were used to: evaluate the performance of each framework, compare how the language features of each framework help or hinder web development, and test the resource efficiency of each framework. After performing these tests and discussing the results, we wrote a section that discussed who we would recommend the Yesod framework to. Overall, Yesod is a production ready framework that is ready to be used (and has been used), in the real world.

Personally, this project has greatly improved my own academic and development skills. I gained a lot of knowledge of the Haskell programming language by getting some hands on experience with Yesod, a framework that uses a lot of advanced features of Haskell. By writing this report, my academic research and writing skills have greatly improved, as I had to perform a lot of research on existing scientific journals about Haskell web frameworks. I have also made some contributions to open source Haskell projects, and have become more interested in the academic side of Computer Science.

References

Allen, C. & Moronuki, J. (2016, July). *Haskell programming from first principles*. (Cit. on p. 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, 20)

How to use Django with Gunicorn. (2018). Retrieved April 24, 2018, from https://docs.djangoproject.com/en/2.0/howto/deployment/wsgi/gunicorn/. (Cit. on p. 27)

Migrations. (2018). Retrieved April 22, 2018, from https://docs.djangoproject.com/en/2.0/topics/migrations/. (Cit. on p. 21)

Templates. (2018). Retrieved April 23, 2018, from https://docs.djangoproject.com/en/2.0/topics/templates/. (Cit. on p. 24)

Writing views. (2018). Retrieved April 27, 2018, from https://docs.djangoproject.com/en/2.0/topics/http/views/. (Cit. on p. 21)

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, 13, 14, 18, 27, 31).

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

Amazon Web Services. (2018). Retrieved April 27, 2018, from https://aws.amazon.com/

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/

How to deploy with WSGI. (2018). Retrieved April 24, 2018, from https://docs.djangoproject.com/en/2.0/howto/deployment/wsgi/

How to use Django with Gunicorn. (2018). Retrieved April 24, 2018, from https://docs.djangoproject.com/en/2.0/howto/deployment/wsgi/gunicorn/

Migrations. (2018). Retrieved April 22, 2018, from https://docs.djangoproject.com/en/2.0/topics/migrations/

Password Iterations (PBKDF2). (2018). Retrieved April 23, 2018, from https://helpdesk.lastpass.com/account-settings/general/password-iterations-pbkdf2/

Redline13. (2018). Retrieved April 27, 2018, from https://www.redline13.com/blog/

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/

Writing views. (2018). Retrieved April 27, 2018, from https://docs.djangoproject.com/en/2.0/topics/http/views/

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-experiance-with-yesod/

Rouse, P. (2018). Yesod auth hashdb github repository. Retrieved April 23, 2018, from https://github.com/paul-rouse/yesod-auth-hashdb

Snoyman, M. [Micahel]. (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

Experiments

All experiments were ran on an Amazon EC2 t2.micro instances. These are cloud servers with 1GB of ram and 1 CPU core. These servers were setup with Ubuntu and the tools needed to run the websites. Both servers were located in the US (East).

A.1 Page Load Speed

Each page of the website was loaded three times with an average being given. The result recorded is the time it takes to load all the files on the web page. This value is taken from the 'Finish' value located at the bottom of the Chromium developer console in the 'Network' tab. These experiments were done with caching turned off. Times were measured in milliseconds. The experiments also started with a fresh website, i.e. no users or messages. For redirects, the HTML values is recorded for the page that started the redirect.

Table A.1: Home Page load speed

Run	Yesod Page	Django Page
1	504	750
2	512	754
3	517	756
Average	511	753.33

Table A.2: Search Page load speed

Run	Yesod Page	Django Page
1	541	739
2	510	768
3	501	762
Average	517.33	756.33

Table A.3: Login Page load speed

Run	Yesod Page	Django Page
1	401	767
2	413	843
3	517	854
Average	443.67	821.33

Table A.4: Signup Page load speed

Run	Yesod Page	Django Page
1	509	775
2	479	739
3	483	778
Average	490.33	764

Table A.5: Create an account load speed

Run	Yesod Page	Django Page
1	510	728
2	500	751
3	503	767
Average	504.33	748.67

Table A.6: Log in to an account speed

Run	Yesod Page	Django Page
1	514	643
2	570	760
3	558	765
Average	547.33	722.67

Table A.7: Logout load speed speed

Run	Yesod Page	Django Page
1	496	770
2	525	750
3	510	762
Average	510.33	761.33

Table A.8: Current user's profile page

Run	Yesod Page	Django Page
1	560	931
2	660	936
3	631	923
Average	617	930

Table A.9: Creating message 'test'

Run	Yesod Page	Django Page
1	680	811
2	667	806
3	691	929
Average	679.33	848.67

Table A.10: Other profile page with three messages

Run	Yesod Page	Django Page
1	670	936
2	643	931
3	641	859
Average	651.33	908.67

Table A.11: Search for message ‘test’, three results

Run	Yesod Page	Django Page
1	514	781
2	502	773
3	524	745
Average	513.33	766.33

Table A.12: Search for user ‘test’, five results

Run	Yesod Page	Django Page
1	524	774
2	516	748
3	517	748
Average	519	756.67

A.2 Resource Usage

For Yesod, a Keter bundle is deployed on the server. Keter is a deployment manager written in Haskell and Yesod has built in support for Keter. On the Django server, nginx is used, a popular open source web server. On the server, a Django server is created, and nginx uses the Django server to deal with incoming requests. All values in this section were acquired through htop.

On the Yesod server, after running all the tests in the previous section, total RAM usage was 109MB. Keter’s main process and sub-processes used about 83MB of RAM, with 56MB being shared. The Django server’s RAM usage was at 125MB of RAM, with Django and gunicorn using around 65MB of RAM, sharing 19MB, and nginx using 6MB of RAM, sharing 3MB. This gives a total of 71MB of RAM being used on the Django server, with 22MB of RAM being shared. Htop sreenshots are included below.

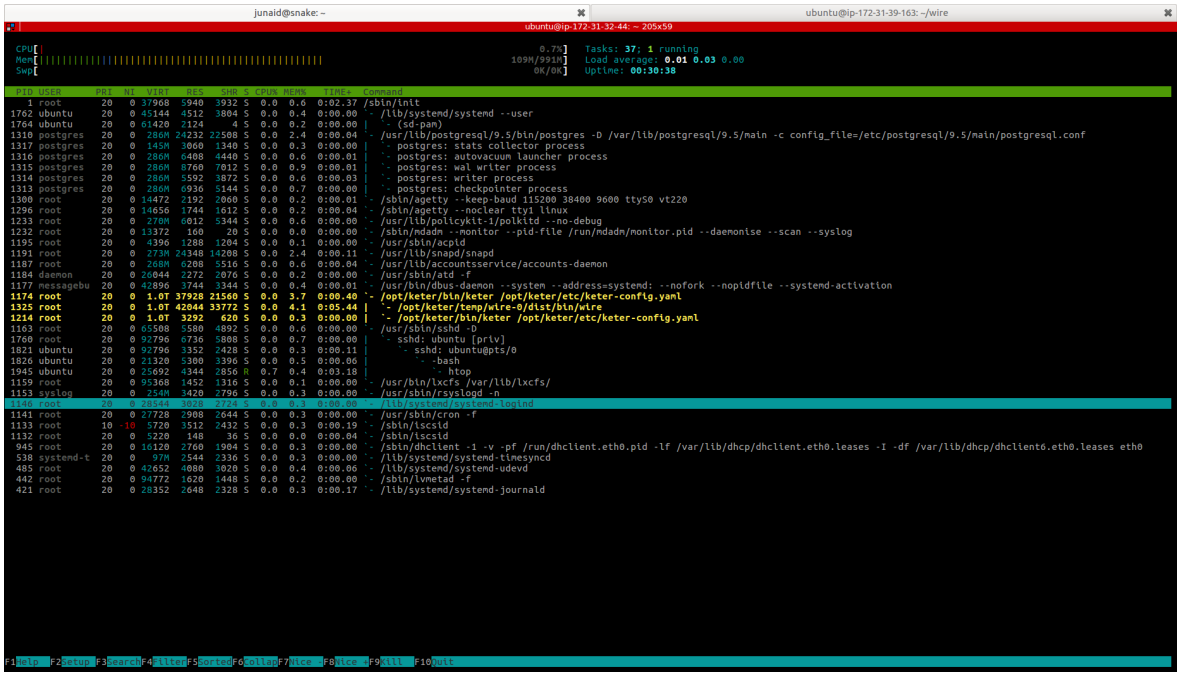


Figure A.1: Yesod htop output

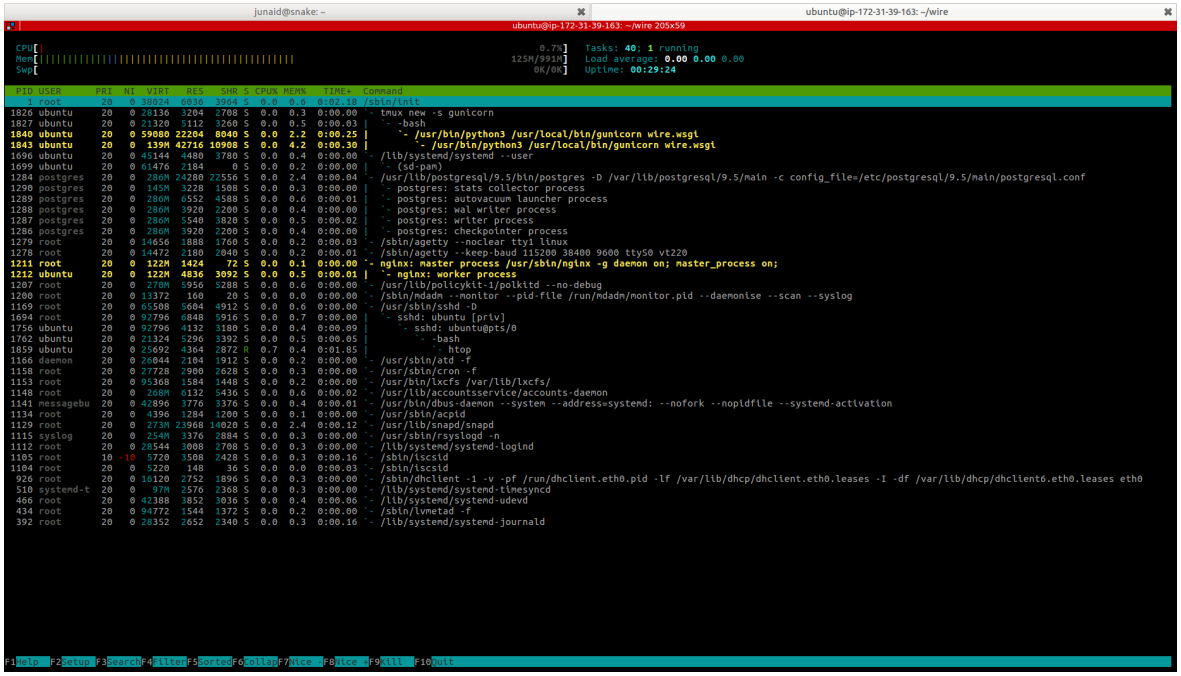


Figure A.2: Django htop output

A.3 Continuous Integration Build Times

Both frameworks are stored on a git repository on GitHub. Whenever there is a new commit, Travis CI, a continuous integration tools, starts to build both frameworks and run tests. The tool will tell you whether or not tests have passed. Django builds take around 2.5 minutes. Yesod builds take around 3.5-4 minutes. It should be noted that the first Yesod build took 32 minutes. This was because the first build had to compile all the library files in the Yesod project. Once these are compiled, they are cached, allowing them to be used for future builds.

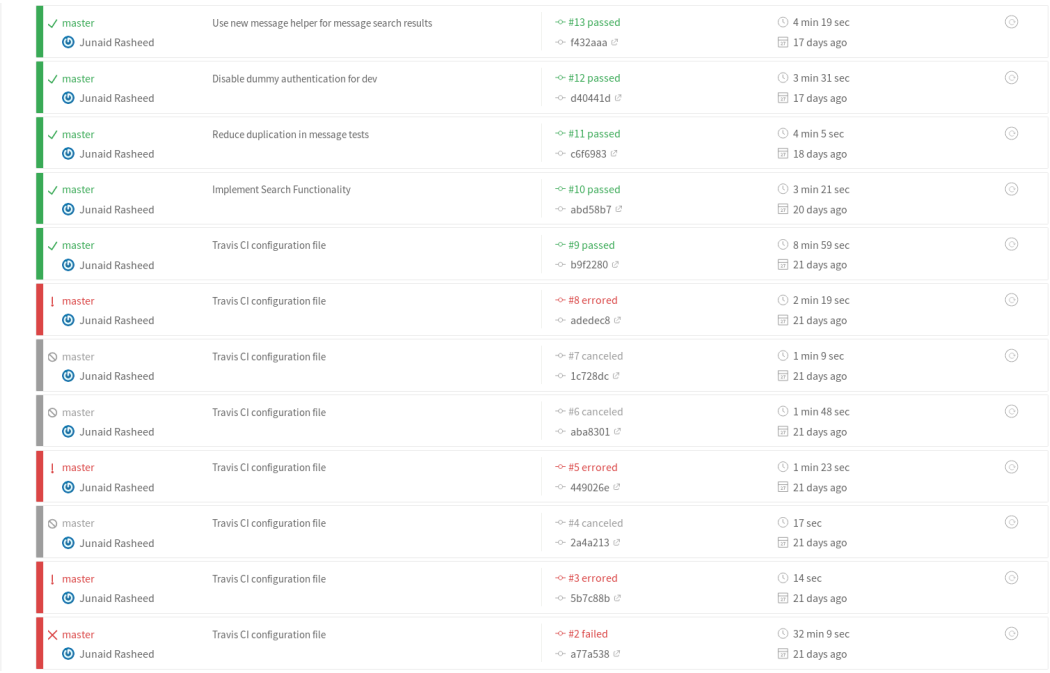


Figure A.3: Yesod Travis build times

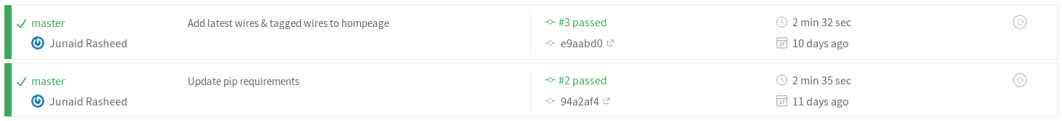


Figure A.4: Django Travis build times

A.4 Load Tests

Load tests were ran using RedLine13 and Amazon EC2 servers. Load tests consisted of 80 users loading a specified page. Three tests were conducted. In one test, the home page was loaded. In the two other tests, the profile page for a user who posted three messages was loaded. Results can be found in 4.2

Table A.13: Load Testing Page Load Speeds

Page	Yesod (s)	Django (s)
Home	4.96	5.54
Profile	5.05	4.94
Profile	4.97	5.13
Average	4.99	5.20

Table A.14: Load Testing Data Received

Page	Yesod (MB)	Django (MB)
Home	6.28	17.99
Profile	6.76	18.74
Profile	6.34	17.84
Average	6.46	18.19

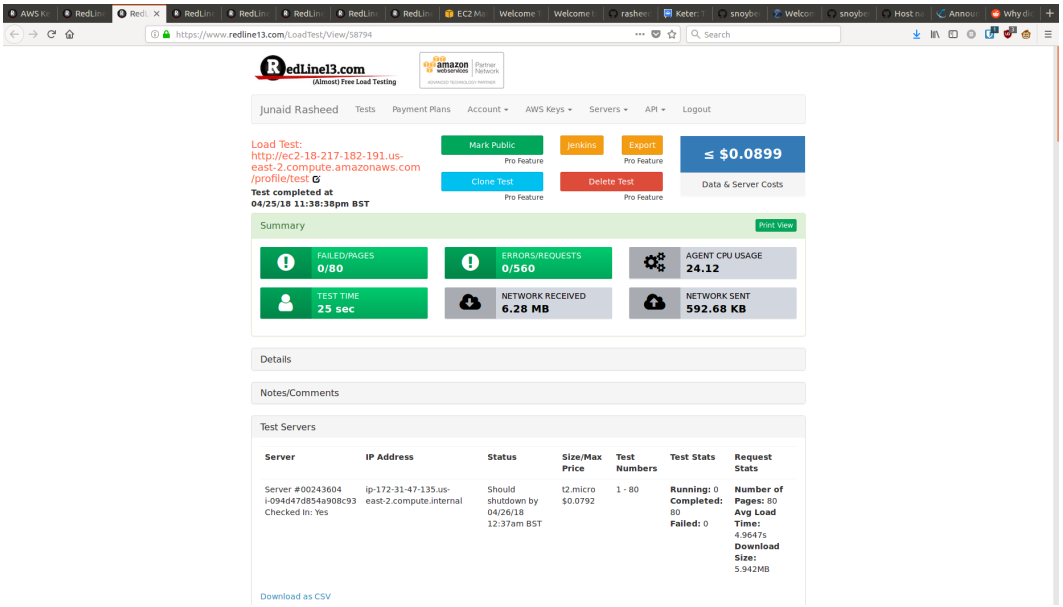


Figure A.5: Yesod Load Test 1

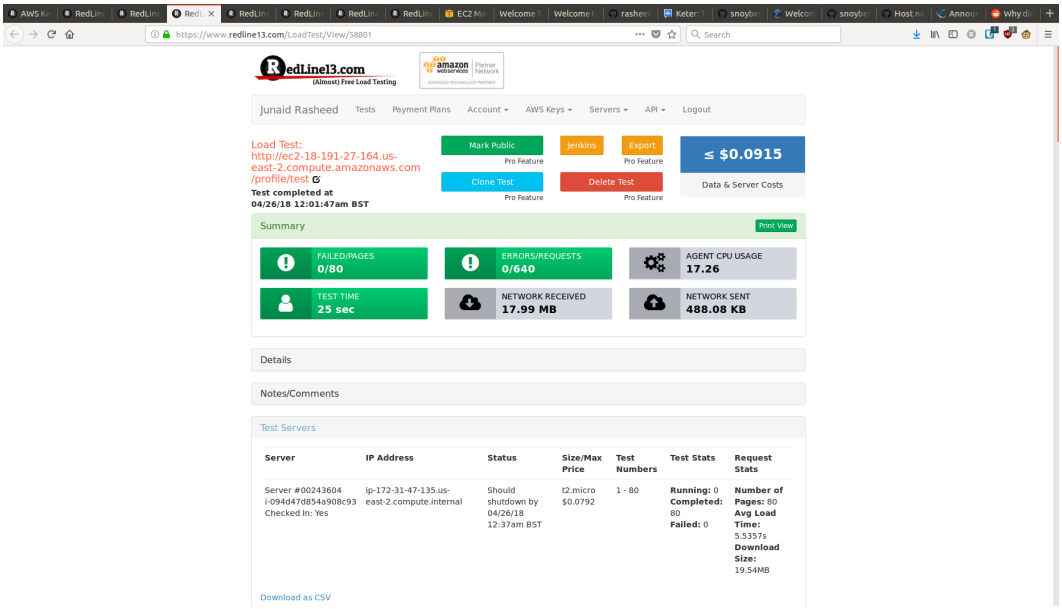


Figure A.6: Django Load Test 1

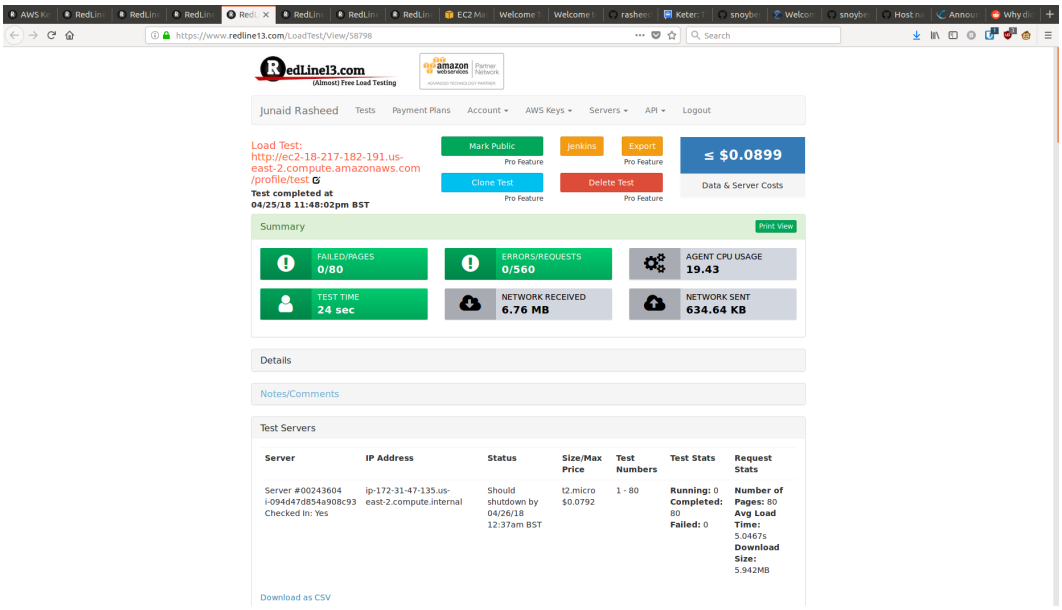


Figure A.7: Yesod Load Test 2

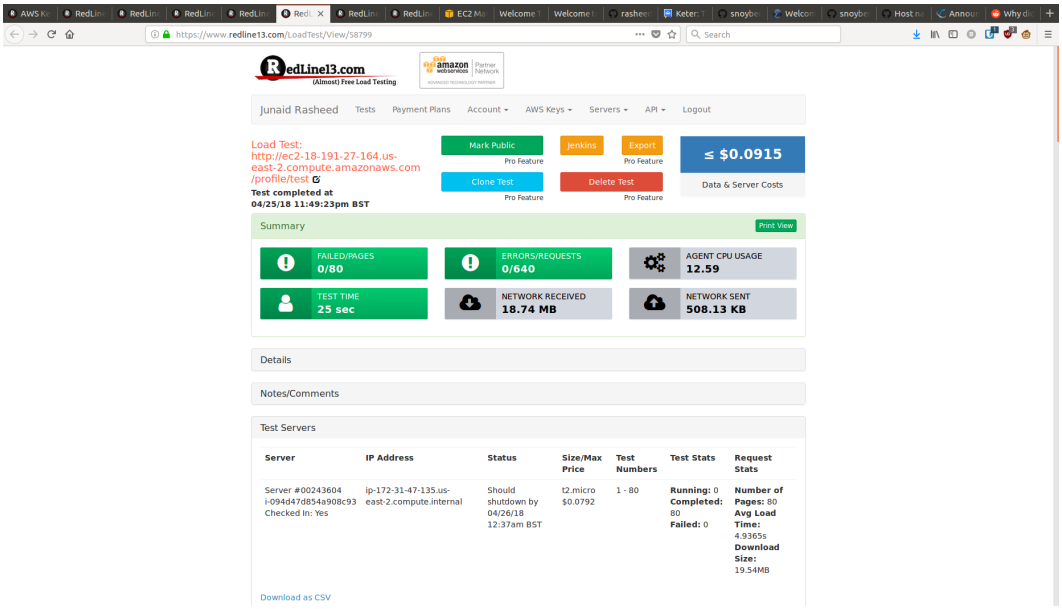


Figure A.8: Django Load Test 2

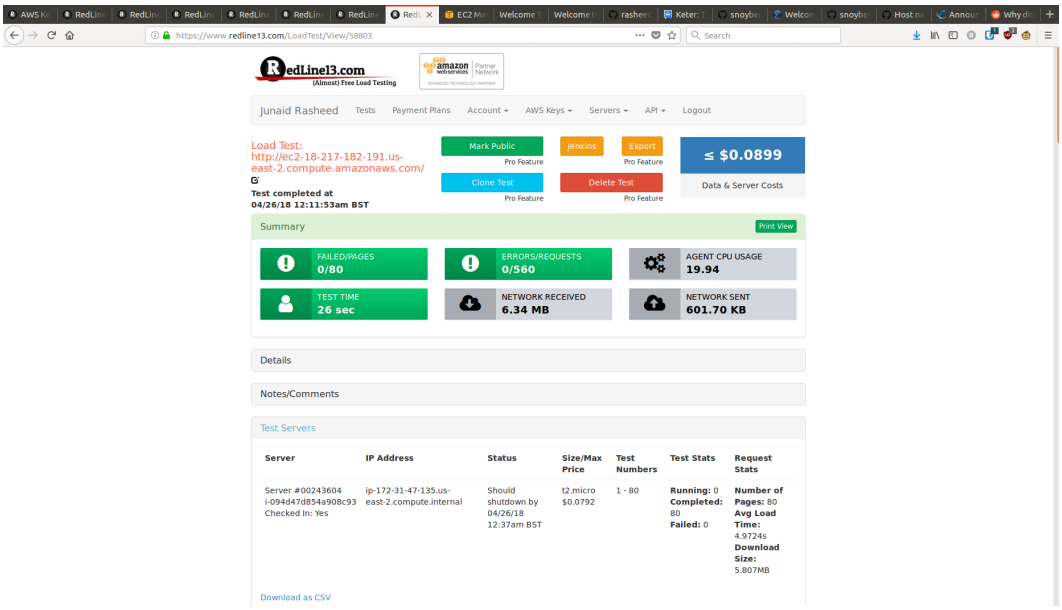


Figure A.9: Yesod Load Test 3

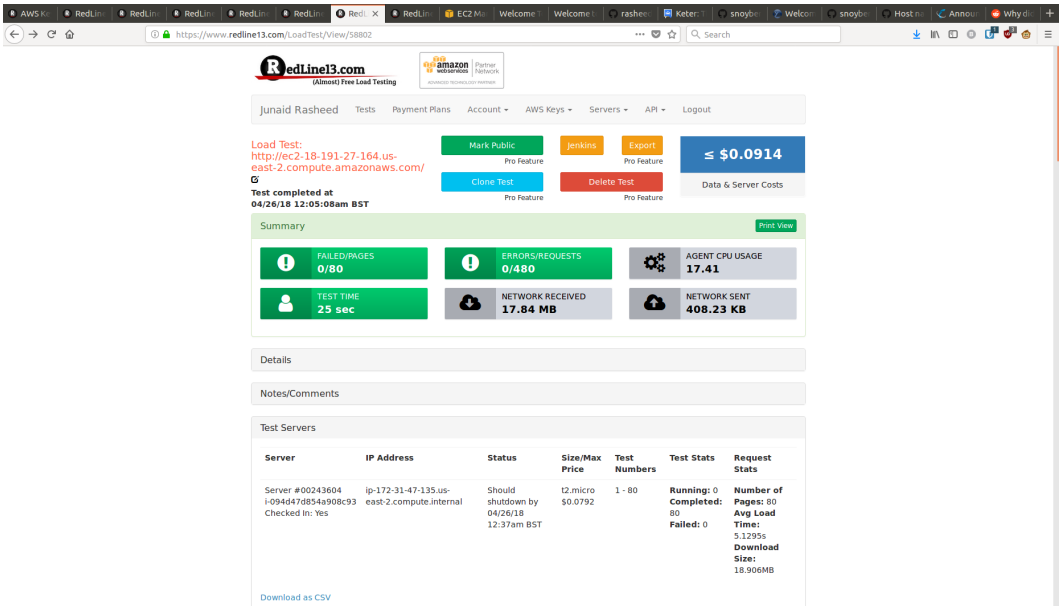


Figure A.10: Django Load Test 3

A.5 Introducing Realistic Errors

For these tests, we purposefully made an error that could realistically happen. After we made the error, we see the error messages that each framework gives us.

A.5.1 Test 1

When creating a message, try to save the form results rather than the message extracted from the results into the database.

Yesod Results: Did not compile, unmatched type error. Tests can't be ran as site did not compile. Django Results: No error thrown even when submitting new message form. Message not created. Database entry added, form data was converted to string and added to Database. Tests passed because they were checking the database count. This test was fixed to check the database content as well.

```
(Entity userId _) <- requireAuth
((result, _), _) <- runFormPost $ messageForm userId
case result of
  FormSuccess message -> do
    -- _ <- runDB . insert $ message -- original line
    _ <- runDB . insert $ result -- new line
```

Code A.1: Yesod Code Change

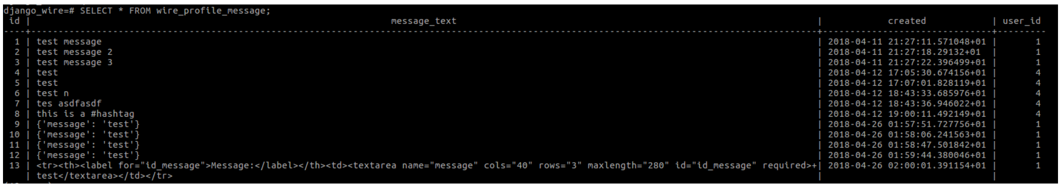
```
form = NewWireForm(request.POST)
if form.is_valid():
  if request.user.is_authenticated:
```

```
    message = form.cleaned_data['message']
    try:
      # Message.objects.create(message_text=message, .. # original line
      Message.objects.create(message_text=form.cleaned_data, .. # changed line 1
      # Message.objects.create(message_text=form, .. # changed line 2, after previous line
    ↪ passed
```

Code A.2: Django Code Change

```
- Couldn't match type 'PersistEntityBackend (FormResult Message)'
with 'SqlBackend'
arising from a use of 'insert'
- In the second argument of '(.)', namely 'insert'
In the expression: runDB . insert
In a stmt of a 'do' block: _ <- runDB . insert $ result
```

Code A.3: Yesod Exception Message



```
django_wire=# SELECT * FROM wire_profile_message;
 id |                                   message_text                                   |            created            | user_id
----+-----------------------------------------------------------------------------------+-------------------------------+---------
  1 | test message                                                                      | 2018-04-11 21:27:11.571048+01 |       1
  2 | test message 2                                                                    | 2018-04-11 21:27:18.29132+01  |       1
  3 | test message 3                                                                    | 2018-04-11 21:27:22.396499+01 |       1
  4 | test                                                                              | 2018-04-12 17:05:30.674156+01 |       4
  5 | test                                                                              | 2018-04-12 17:07:01.828119+01 |       4
  6 | test n                                                                            | 2018-04-12 18:43:33.685976+01 |       4
  7 | tes asdfasdf                                                                      | 2018-04-12 18:43:36.946022+01 |       4
  8 | this is a #hashtag                                                                | 2018-04-12 19:00:11.492149+01 |       4
  9 | {'message': 'test'}                                                               | 2018-04-26 01:57:51.727756+01 |       1
 10 | {'message': 'test'}                                                               | 2018-04-26 01:58:06.241563+01 |       1
 11 | {'message': 'test'}                                                               | 2018-04-26 01:58:47.501842+01 |       1
 12 | {'message': 'test'}                                                               | 2018-04-26 01:59:44.380046+01 |       1
 13 | <tr><th><label for="id_message">Message:</label></th><td><textarea name="message" cols="40" rows="3" maxlength="280" id="id_message" required>+| 2018-04-26 02:00:01.391154+01 |       1
    | test</textarea></td></tr>                                                         |                               |
```

Figure A.11: Django Message Table Values

A.5.2 Test 2

Simply misspell a variable name. Most IDE tools catch this anyway but error messages are useful to see. The misspelling was done in the code that returns recommended users on the profile page. This code returns data in JSON format.

Yesod: Did not compile, variable not in scope error. Compiler recommended actual variable.

Django: AJAX request returned a 500 error. Loading the page shows the debug output. Exception thrown with message "name 'user' is not defined". Tests failed.

```
Entity userId user <- requireAuth
followers <- runDB $ selectList [FollowFollowerId ==. userId] []
-- See: https://stackoverflow.com/questions/36727794/haskell-persistent-reusing-selectlist
let followingIds = map (\(Entity _ (Follow _ followingId)) -> followingId) followers
users <- runDB $ selectList [UserUsername !=. userUsername user, UserId /<-. followingIds]
  ↪ [LimitTo 5]
let cleanUsers = map (\(Entity uid (User uname _ _)) -> (object ["id" .= uid, "username" .=
  ↪  uname])) users
-- returnJson cleanUsers -- original line
returnJson cleanUser -- new line
```

Code A.4: Yesod Code Change

```
if request.user.is_authenticated:
follow_query = Follow.objects.filter(follower_id=request.user)
users = User.objects.filter().exclude(id=request.user.id).exclude(username=
  ↪ excluded_username)\
  .exclude(followed_user__in=follow_query).values('username')[:5]
# return JsonResponse(list(users), safe=False)  # original line
return JsonResponse(list(user), safe=False)  # new line
```

Code A.5: Django Code Change

```
Variable not in scope: cleanUser
Perhaps you meant ‘cleanUsers’ (line 19)
```

Code A.6: Yesod Exception Message

```
NameError at /get-recommended-users/test
name ‘user’ is not defined
```

Code A.7: Django Exception Message

Appendix B

Instructions

This section will give you instructions on how to run the frameworks on your own Machine. These instructions are for Ubuntu 16.04. The codebase is included on a CD attached to the physical copy of this report. I can also give you access to the GitHub repository if you email me at `rasheeja@aston.ac.uk`.

B.1 Yesod

Install postgresql 9.5

```
sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt-get -y install postgresql postgresql-9.5 postgresql-client postgresql-common
↪ libpq-dev
```

Create the database

```
sudo -u postgres psql
CREATE USER wire WITH PASSWORD 'wire';
CREATE DATABASE yesod_wire OWNER wire;
CREATE DATABASE yesod_wire_test OWNER wire;
```

```
cd wire-yesod/yesod/wire
curl -sSL https://get.haskellstack.org/ | sh
stack build yesod-bin cabal-install --install-ghc
stack build
```

Run the development webserver

```
cd wire-yesod/yesod/wire
stack exec -- yesod devel
```

You should now be able to access the site at `localhost:3000`

B.2 Django

Install postgresql 9.5

```
sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt-get -y install postgresql postgresql-9.5 postgresql-client postgresql-common
↪ libpq-dev
```

Create the database

```
sudo -u postgres psql
CREATE USER wire WITH PASSWORD 'wire';
CREATE DATABASE django_wire OWNER wire;
CREATE DATABASE django_wire_test OWNER wire;
```

Install pip and the necessary python libraries

```
sudo apt-get -y install python3-pip
sudo pip3 install --upgrade pip
sudo pip3 install psycopg2 Django django-bootstrap3 django-bootstrap-breadcrumbs
```

Make database migrations and run the development webserver

```
cd wire-django/django/wire
python3 manage.py migrate
python3 manage.py runserver
```

You should now be able to access the site at `localhost:8000`

Appendix C

Project Diary

C.1 Meeting 1 - 3rd October 2017

C.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

C.2 Meeting 2 - 12th October 2017

C.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

C.3 Meeting 3 - 19th October 2017

C.3.1 Meeting Notes:

Carry on with the Haskell Programming from First principles book
Have some planning for the twitter clone ready

C.4 Meeting 4 - 24th October 2017

C.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

C.5 Meeting 5 - 10th November 2017

C.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.

C.6 Meeting 6 - 16th November 2017

C.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.

C.7 Meeting 7 - 23rd November 2017

C.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.

C.8 Meeting 8 - 14th December 2017

C.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.

C.9 Meeting 9 - 1st February 2018

C.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.

C.10 Meeting 10 - 15th February 2018

C.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 experineced

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

C.11 Meeting 11 - 22nd March 2018

C.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

C.12 Meeting 12 - 19th April 2018

C.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
Think about separating language and 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.

C.13 Meeting 13 - 24th April 2018

C.13.1 Meeting Notes:

Entity relation diagram.
Redo tests on EC2 instances.
For research reports

- State objective, hypothesis, search question.
- Methodology, development (testing, iterative development, conntinous integration)
- Mention websites as explicit deliverables that support the comparison.
- Comparison methodology, what do frameworks give you to support TDD.
- Evaluation methodology, deploy, page load speeds, load speeds, documentation.

Background chapter, expand on Yesod and Django components. Some diagrams would be nice.

C.14 Meeting 14 - 26th April 2018

C.14.1 Meeting Notes:

For the networking test where you mention that Django uses more data, use the network tab on the browser to confirm this. If you load the site in Yesod, the data loaded is lower and then it's compressed.

Fix lstlisting, set language before code blocks.

When Yesod throws errors, say runtime error, with a compiler message.

texcel, you can use macros. Could make font for code smaller. Reformat the code blocks to be easier to read. Any case against the types. Early profile page was one request/view. This meant that there was some extra Haskell code that had to be written.

Last section, fix typo 'Yesod being less resource intensive than Yesod.', wrong your.

Small section that talks about how you made a functional, actually usable, site. Site is realistic, which gives us a useful way to evaluate the frameworks. This could be before the testing method.

Amazon monitoring is only available in five minute intervals (free usage limit).

A refactor at the end would have been useful to evaluate at the end. Does the more rigid structure of the Haskell program help with refactoring? Haskell compiler would guide you on what needs to be changed after a small change.

Conclusion Summarise the entire report, half a page, a side.

Get rid of some of the more uninteresting website images.

A useful diagram would explain the architecture of each framework.

C.15 Project Definition Form

C.15.1 14th October 2017

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

C.15.2 15th October 2017

Tutor feedback implemented

C.15.3 19th October 2017

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

C.16 Interim Report

C.16.1 18th January 2018

Interim report started and then finished

C.16.2 19th January 2018

Sent Ethics form to tutor
Proof-read and submitted the interim report

C.17 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.

C.17.1 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 profle 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 uses on your feed.

C.17.2 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.

C.18 Project Diary Final Report

C.18.1 20th April 2018

Started write-up of the final report

C.18.2 23rd April 2018

Run experiments on Yesod and Django sites, as planned. Completed first draft and sent to tutor for feedback

C.18.3 24th April 2018

Received in-person feedback from tutor, Started implementing feedback.

C.18.4 26th April 2018

Sent next draft to tutor, received more feedback, completed third draft to review before submission.

C.18.5 27th April 2018

Applied finishing touches to the report. Sent small sections to tutor for quick feedback. Submitted the report.

Appendix D

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 re ected upon the ethical implications of the project and/or its potential ndings, 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 

_____