Haskell Development

a) Command Line Programs

a.i. Data Visualization - Weather Project

This project was about two datasets, one which included 266 global temperatures from 1750 until 2015 and the second one contained 271 temperatures of Milan collected between 1743 and 2013. To plot the moving averages I created the Haskell program **weatherComparator** which reads in both CSV files. The moving averages of both regions are calculated based on an interval of 10 years. If a year had no temperature then this row was dropped. The calculated moving averages were written to a new CSV file. The Haskell program plotted the CSV file which showed the moving average global temperatures vs. the moving average temperatures of Milan.

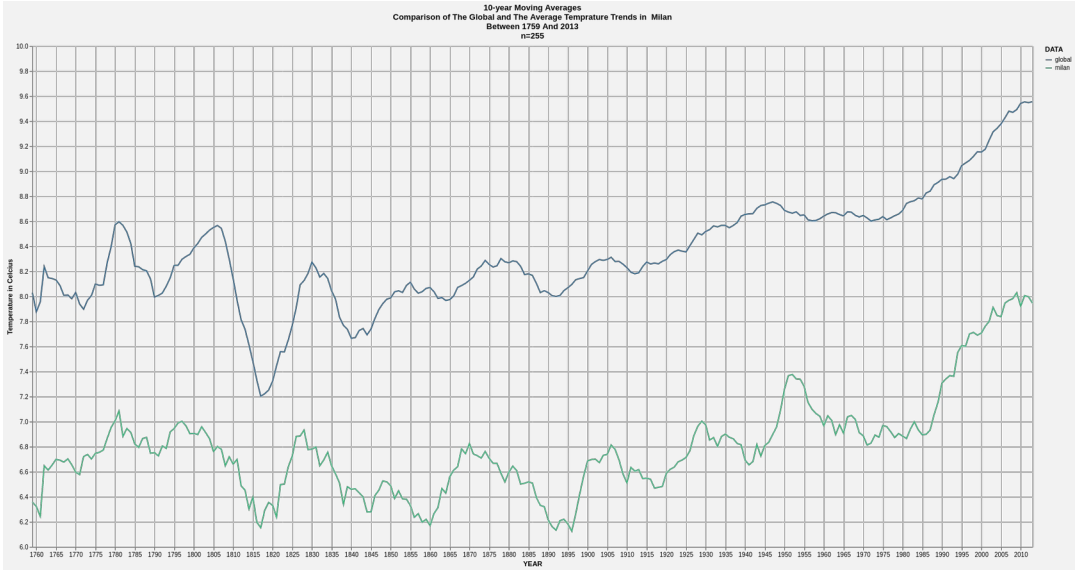
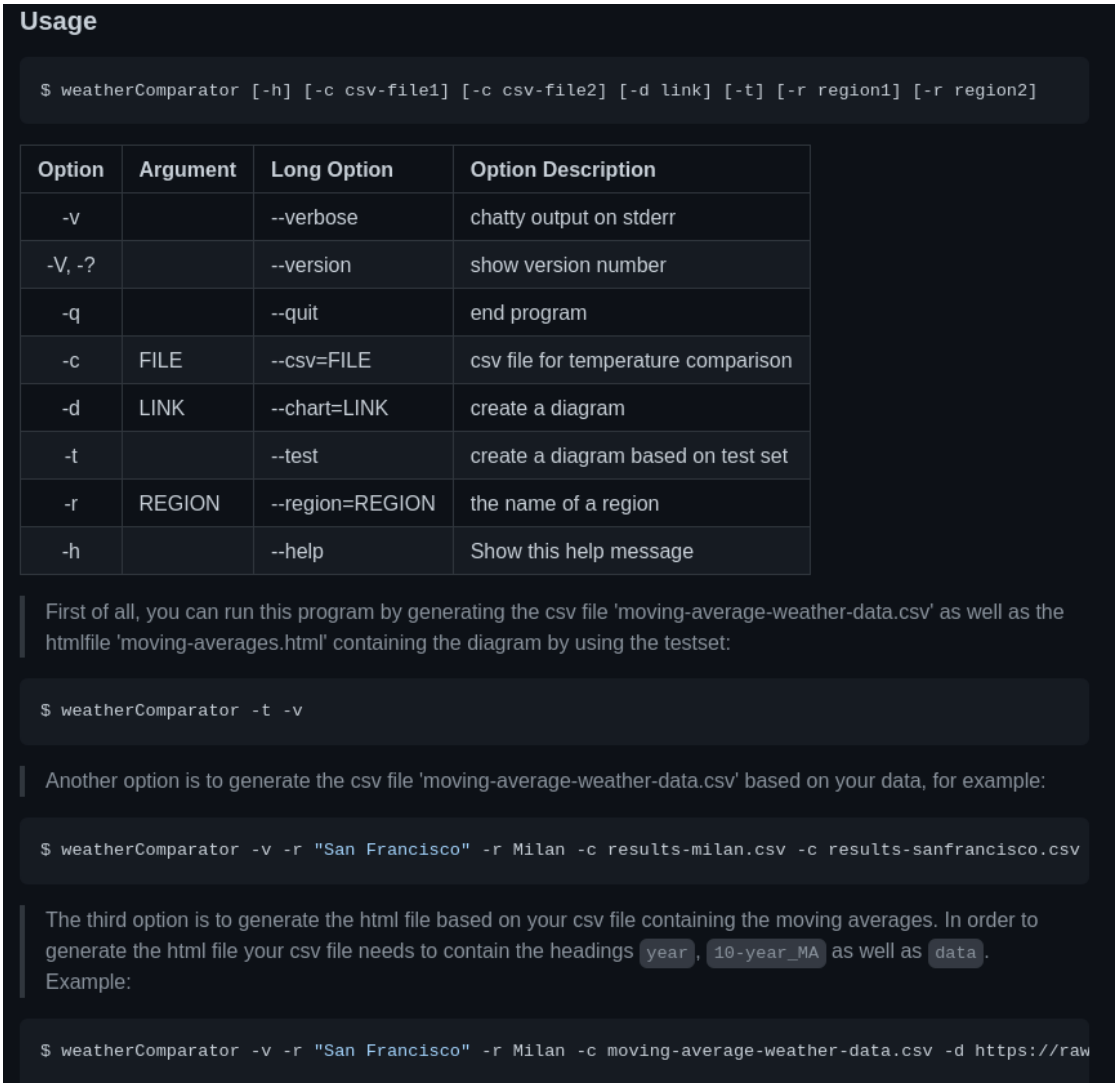


Figure i: Plot produced with the Haskell library Vega Lite

The **weatherComparator** is a typical command line tool. It can either calculate the moving averages of two regions, plot the datasets or perform both tasks one after another.



Usage

```
$ weatherComparator [-h] [-c csv-file1] [-c csv-file2] [-d link] [-t] [-r region1] [-r region2]
```

Option	Argument	Long Option	Option Description
-v		--verbose	chatty output on stderr
-V, -?		--version	show version number
-q		--quit	end program
-c	FILE	--csv=FILE	csv file for temperature comparison
-d	LINK	--chart=LINK	create a diagram
-t		--test	create a diagram based on test set
-r	REGION	--region=REGION	the name of a region
-h		--help	Show this help message

> First of all, you can run this program by generating the csv file 'moving-average-weather-data.csv' as well as the htmlfile 'moving-averages.html' containing the diagram by using the testset:

```
$ weatherComparator -t -v
```

> Another option is to generate the csv file 'moving-average-weather-data.csv' based on your data, for example:

```
$ weatherComparator -v -r "San Francisco" -r Milan -c results-milan.csv -c results-sanfrancisco.csv
```

> The third option is to generate the html file based on your csv file containing the moving averages. In order to generate the html file your csv file needs to contain the headings `year`, `10-year_MA` as well as `data`.
> Example:

```
$ weatherComparator -v -r "San Francisco" -r Milan -c moving-average-weather-data.csv -d https://raw
```

Figure ii: Running a Haskell program from the command line

a.ii. UK Weather Data

Another Haskell program involved querying and updating temperature figures for a list of UK weather stations. This program runs in GHCI mode which means that the Haskell file containing the code can be executed as a script by just loading it under GHCI. It has an interactive menu showing you all the available tasks of this program. Its main purpose is to read in the average temperatures of the UK weather stations stored in a file and display the data on a screen.

Name	North	East	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Heathrow	51.5	-0.4	8.4	9.0	11.7	15.0	18.4	21.6	27.6	23.4	20.2	15.8	11.5	8.8
St Austell	50.3	-4.8	9.5	9.6	11.3	13.3	16.2	18.1	20.6	20.4	18.5	15.0	12.3	10.2
Baltasound	60.7	-0.8	-0.4	6.3	7.4	9.2	11.2	13.2	15.1	15.4	13.6	10.9	8.5	7.0
Ronaldsway	54.1	-4.6	-1.0	8.4	9.4	11.5	14.3	16.5	18.2	18.2	16.6	13.8	11.1	9.2
Solvent	50.8	-1.2	-4.0	8.7	11.0	13.9	17.1	19.6	21.6	21.6	19.4	15.7	11.9	9.2
Greenwich Park	51.5	0.0	-2.0	9.2	12.1	15.4	18.6	21.4	23.8	23.3	20.3	15.8	11.6	8.8
Mumbles Head	51.6	-4.0	-8.7	-0.5	-2.0	12.4	15.2	17.8	19.6	19.7	18.0	14.7	11.5	9.1
Valley	53.3	-4.5	-8.4	-8.4	9.8	12.1	15.0	17.2	18.8	18.8	17.3	14.3	11.3	9.1
Hunstanton	52.9	0.5	7.0	7.4	9.8	12.6	16.0	18.8	21.3	21.3	18.3	14.5	10.3	7.6
Durham	54.8	-1.6	6.9	7.8	9.9	12.5	15.4	18.0	20.2	19.9	17.4	13.5	9.6	0.0
Monks Wood	52.4	-0.2	7.6	8.4	11.0	14.1	17.2	20.0	22.6	22.5	19.5	15.2	10.7	7.8

Figure iii: Displaying the temperatures in a table

What's more, this Haskell command line tool is able to group the temperatures based on months displaying the temperatures of all weather stations for each month in a single graph.

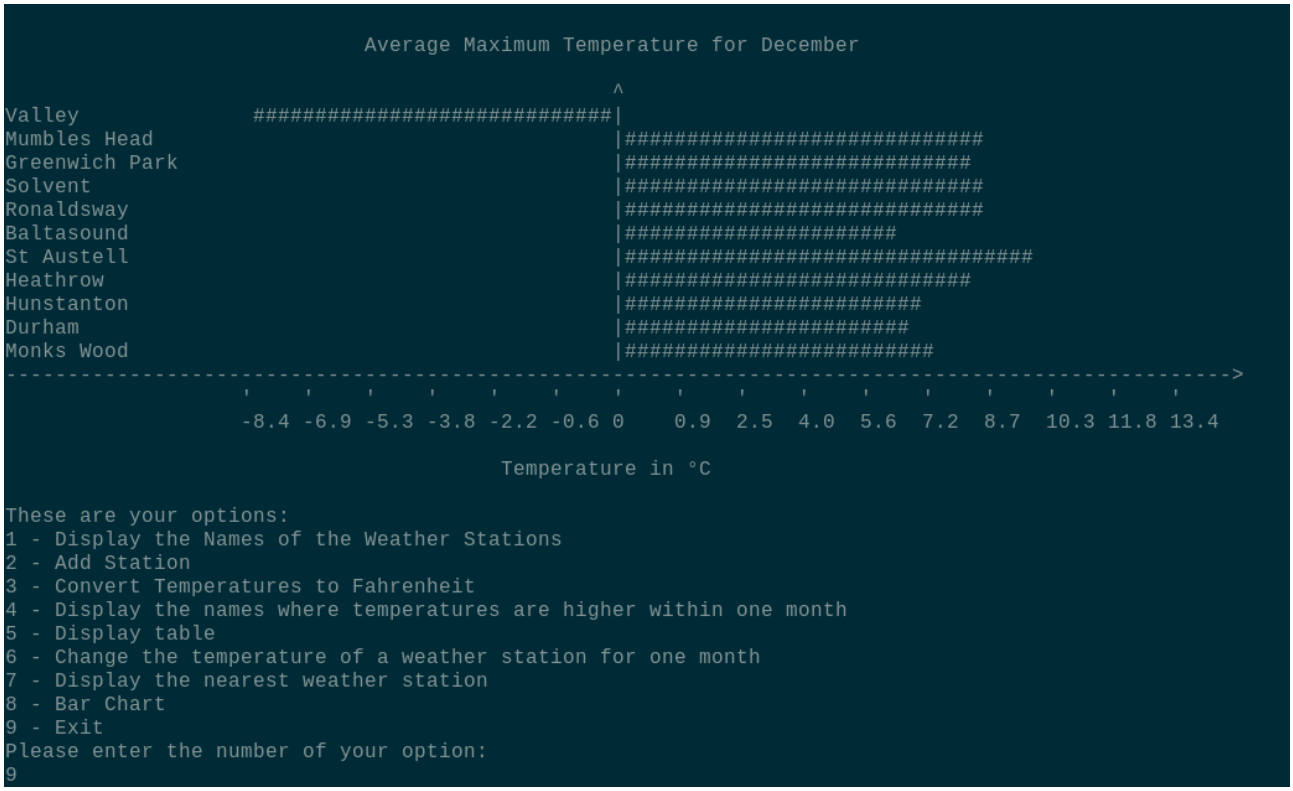


Figure iv: The animated bar charts display the temperatures of a single month

a.iii. Errorbars, Shadedplots, Dataplots

For this project, I created three functions used for plotting different kinds of diagrams. This was achieved by using the Haskell library gnuplot which depends on the command line tool gnuplot. I implemented the functions in such a way so the user could adjust the width of the line together with its label, color and use different datasets. Additionally, I made it possible for the user, to plot several lines on a single diagram. The diagram itself was additionally adjustable with regards to title, x-axis and y-axis labels as well as the x-tics and y-tics.

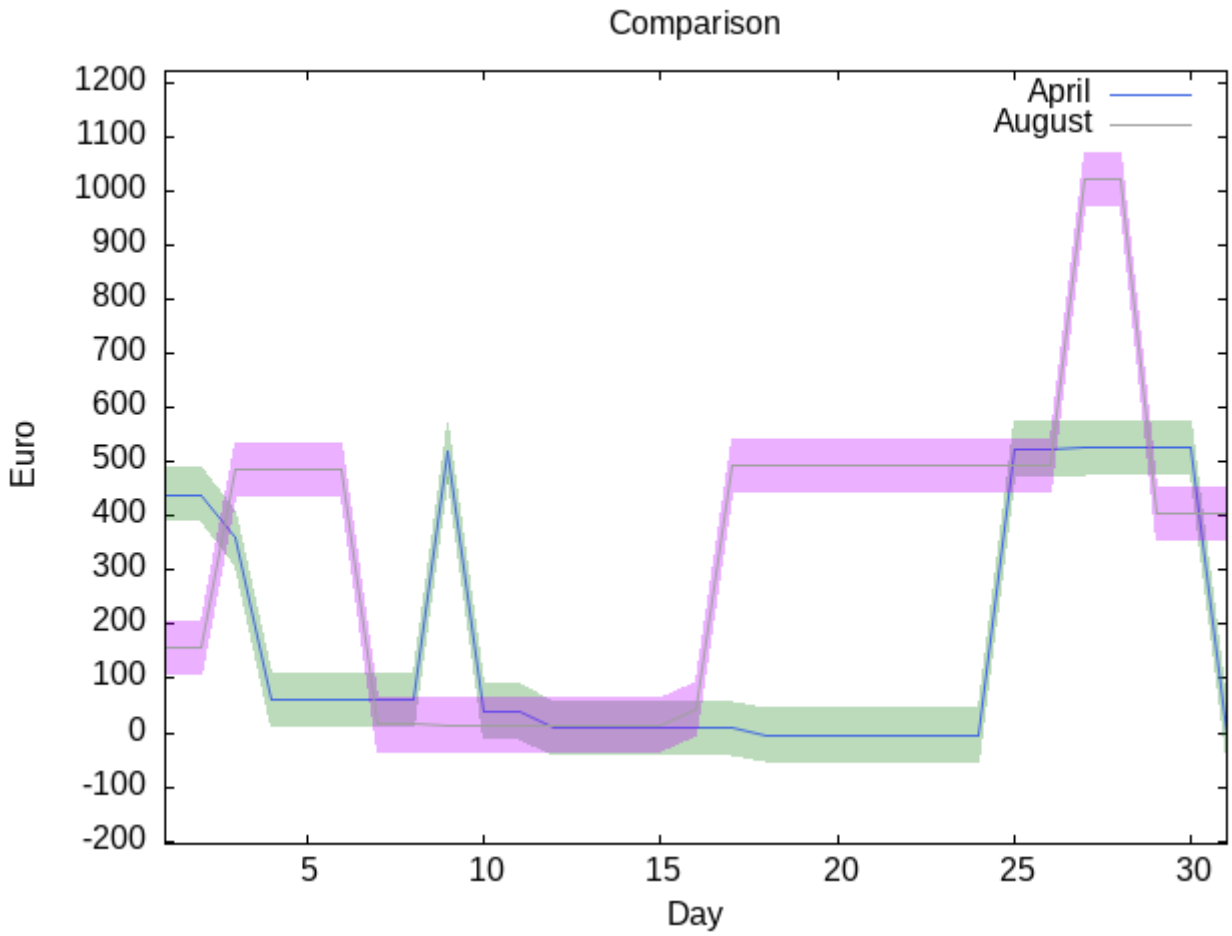


Figure v: Shadedplots realised with gnuplot

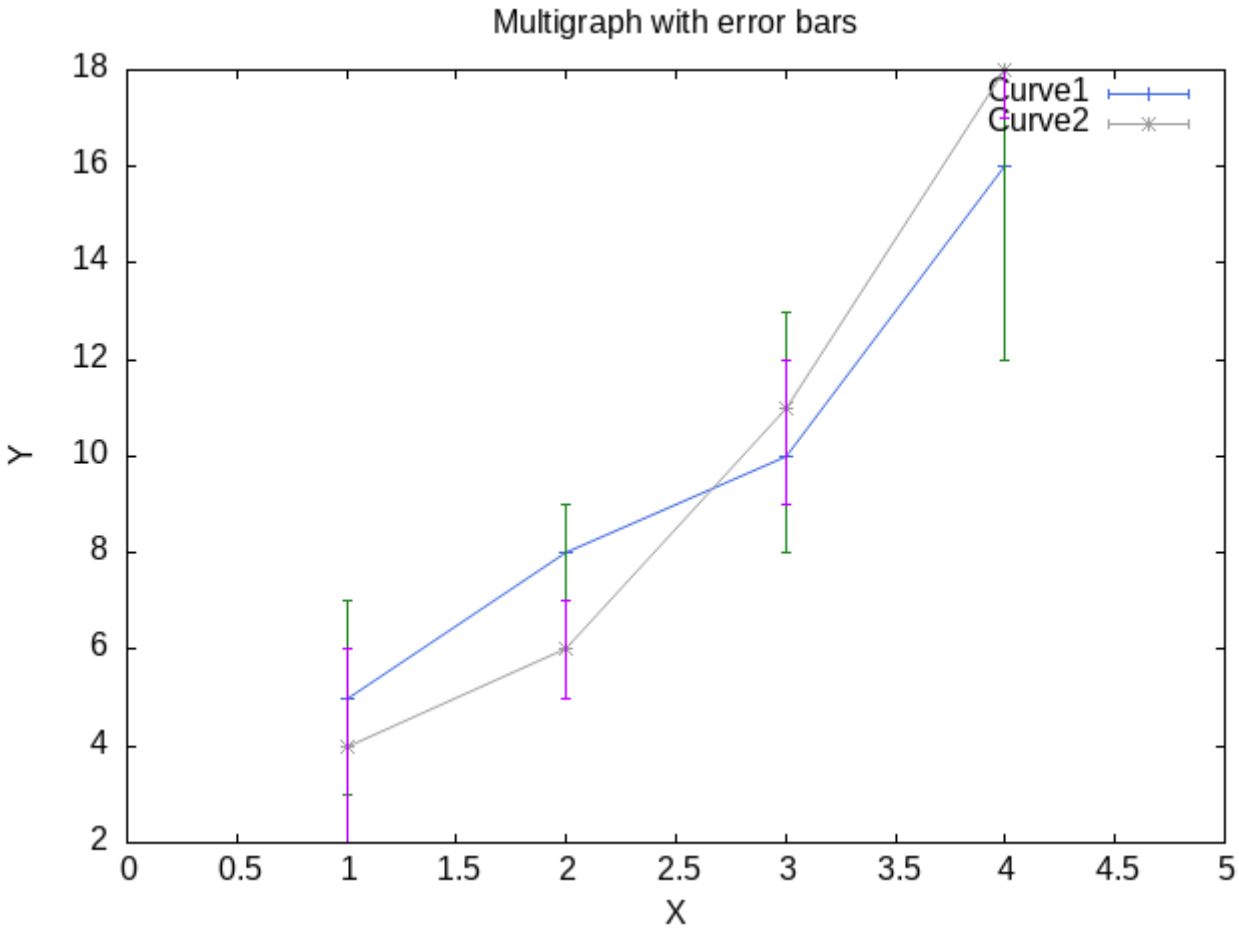


Figure vii: Multiple errorbars realized with gnuplot

b) Web Applications

I can create web applications using the Haskell framework **yesod**. It's a framework combining Haskell as the main back-end language with JavaScript, CSS and HTML as the front-end languages. It supports a variety of libraries which makes it pretty versatile. Under yesod the data can be sent to the server via the GET, or POST method or by using Ajax.

In one of my Haskell projects I adapted a yesod web application in which I made MySQL support available. Additionally, I inserted a search form used for looking through the imported bookmarks. Not only that, I also created a complex query which was basically a full text search using the Haskell library called **Esqueleto**. With Esqueleto I can virtually implement all MySQL functions by myself if they are not provided by the library. This ensures type safety and readable code. What's more, the entered text of the user is parsed using the library **Parsec** which is widely known in the Haskell development. With Parsec, I was able to parse pretty complex phrases.

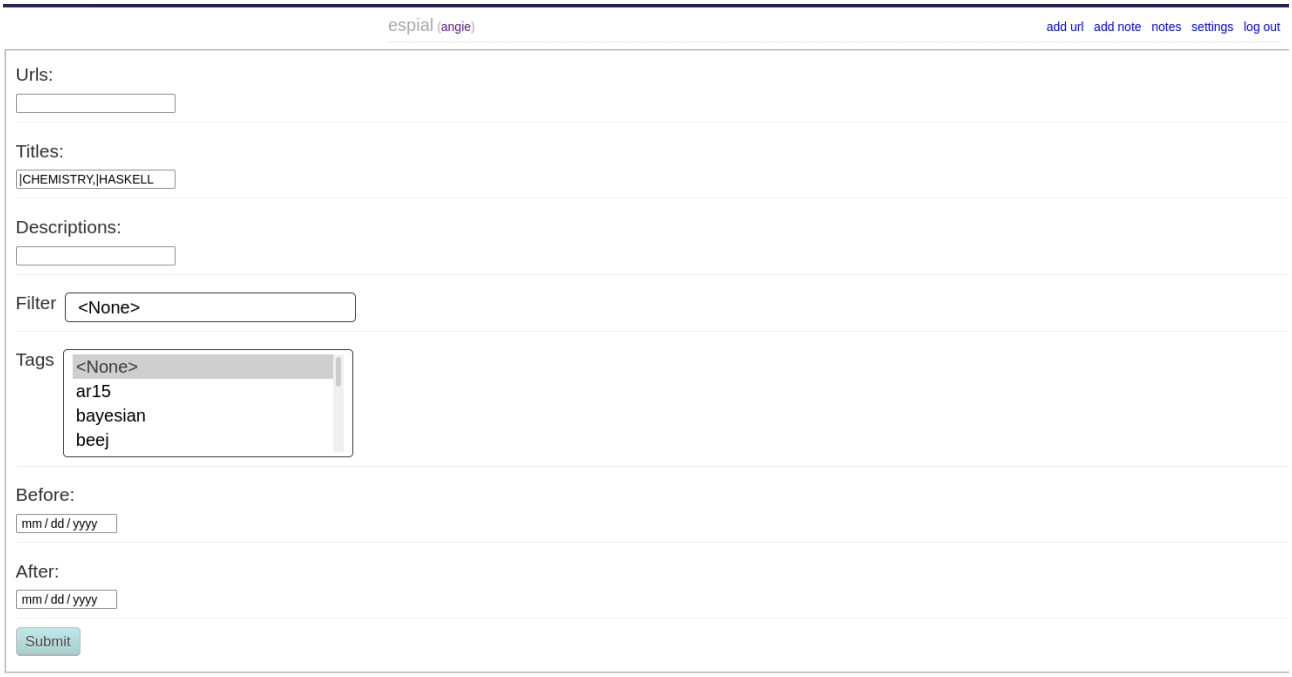


Figure viii: This search form sends the data via the POST method to a Haskell web server.

As for as the search results, yesod provides a pagination module which makes it easy to go back and forth between the search results pages.

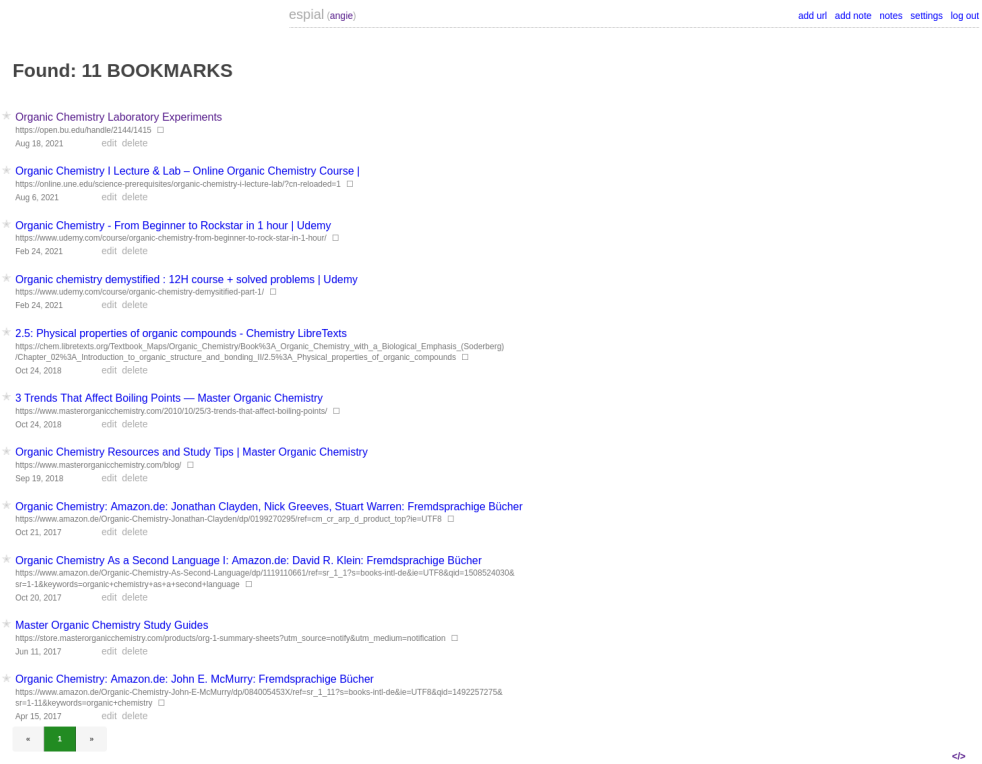


Figure ix: Using the Yesod Pagination module, viewing the search results are realised without javascript.

c) GTK Programs

Apart from web applications, I can implement programs using a graphical user interface. In Haskell, you'll certainly prefer the GTK library because it's well supported in Haskell. Using the GTK library, I can handle events. Usually, the GTK programs are implemented in an imperative style. It's advantage over web apps is that it doesn't require as much PC resources.