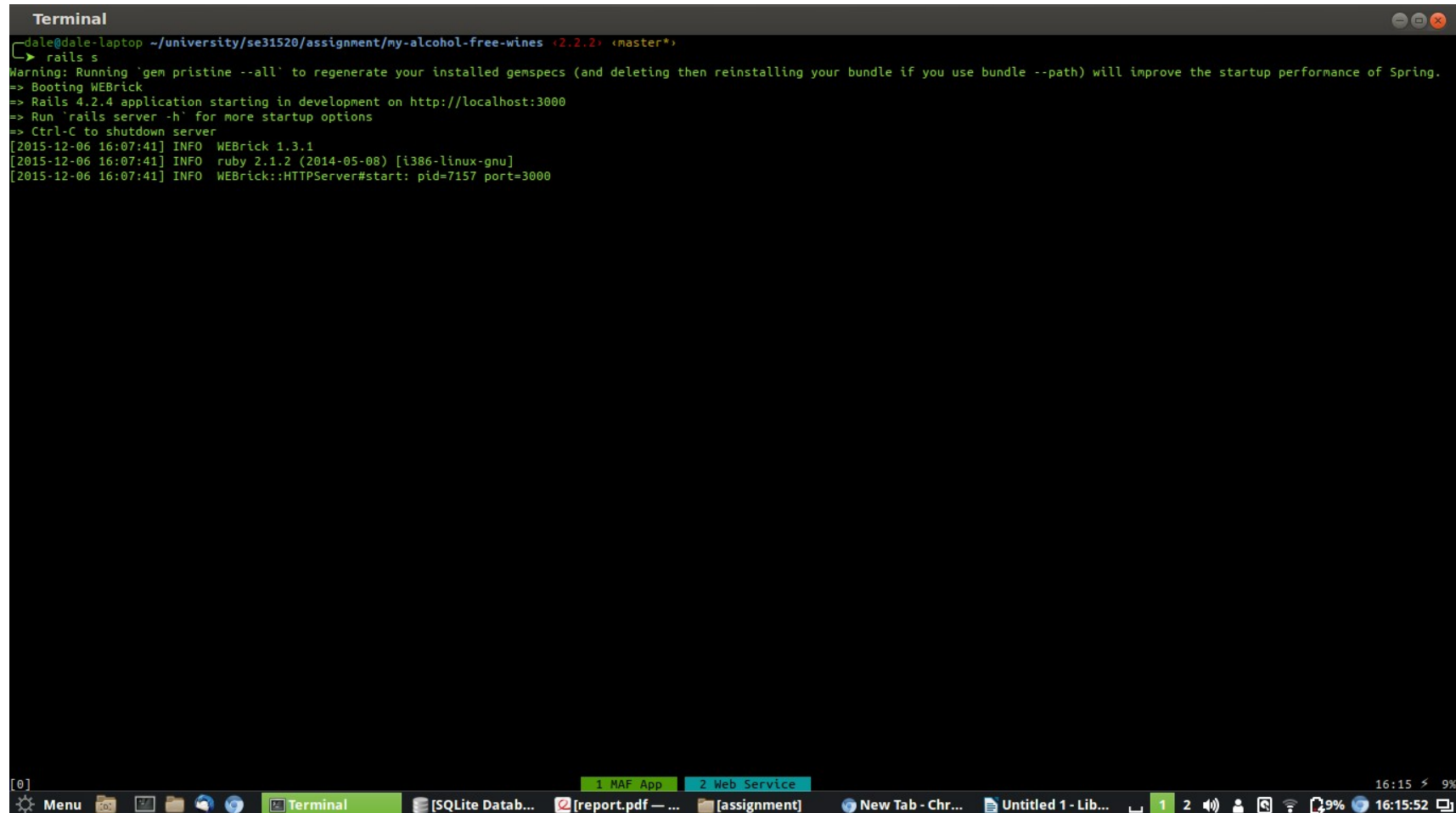


# Screenshots of My Alcohol Free Wines Assignment

Unfortunately I was unable to get any screencasting software working on my machine, so as an alternative, this document provides screenshot evidence of my My Alcohol Free Wines application and the web service running.



```
Terminal
dale@dale-laptop ~/university/se31520/assignment/my-alcohol-free-wines (2.2.2) (master*)
> rails s
Warning: Running 'gem pristine --all' to regenerate your installed gemspecs (and deleting then reinstalling your bundle if you use bundle --path) will improve the startup performance of Spring.
=> Booting WEBrick
=> Rails 4.2.4 application starting in development on http://localhost:3000
=> Run 'rails server -h' for more startup options
=> Ctrl-C to shutdown server
[2015-12-06 16:07:41] INFO  WEBrick 1.3.1
[2015-12-06 16:07:41] INFO  ruby 2.1.2 (2014-05-08) [i386-linux-gnu]
[2015-12-06 16:07:41] INFO  WEBrick::HTTPServer#start: pid=7157 port=3000
```

The screenshot shows a terminal window titled "Terminal" with a dark background. The prompt is "dale@dale-laptop ~/university/se31520/assignment/my-alcohol-free-wines (2.2.2) (master\*)". The user has entered "rails s". The output shows a warning about Spring, followed by "Booting WEBrick", "Rails 4.2.4 application starting in development on http://localhost:3000", and "Run 'rails server -h' for more startup options". Then, it says "Ctrl-C to shutdown server". Finally, it shows three log lines: "[2015-12-06 16:07:41] INFO WEBrick 1.3.1", "[2015-12-06 16:07:41] INFO ruby 2.1.2 (2014-05-08) [i386-linux-gnu]", and "[2015-12-06 16:07:41] INFO WEBrick::HTTPServer#start: pid=7157 port=3000". The terminal window is part of a desktop environment with a dock at the bottom showing icons for Menu, SQLite Datab..., [report.pdf — ..., [assignment], New Tab - Chr..., and Untitled 1 - Lib... The system clock in the bottom right corner shows 16:15 and 9% battery.

Screenshot 1: The MAF application is running, started using rails s from the application directory

```
Terminal
dale@dale-laptop ~/university/se31520/assignment/service (2.2.2) (master*)
> python ./app.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat

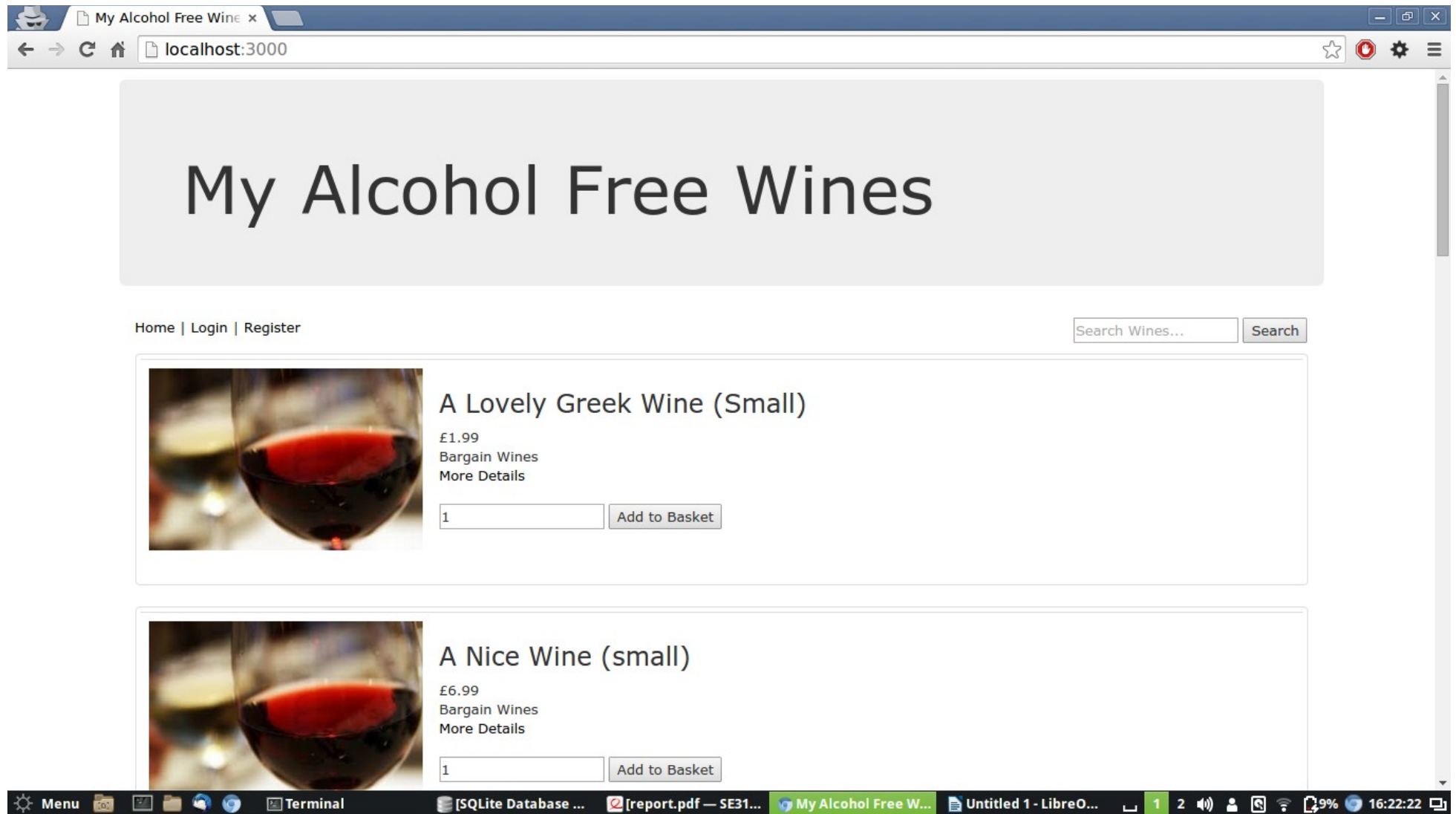
dale@dale-laptop ~/university/se31520/assignment/service (2.2.2) (master*)
>
```

[0]

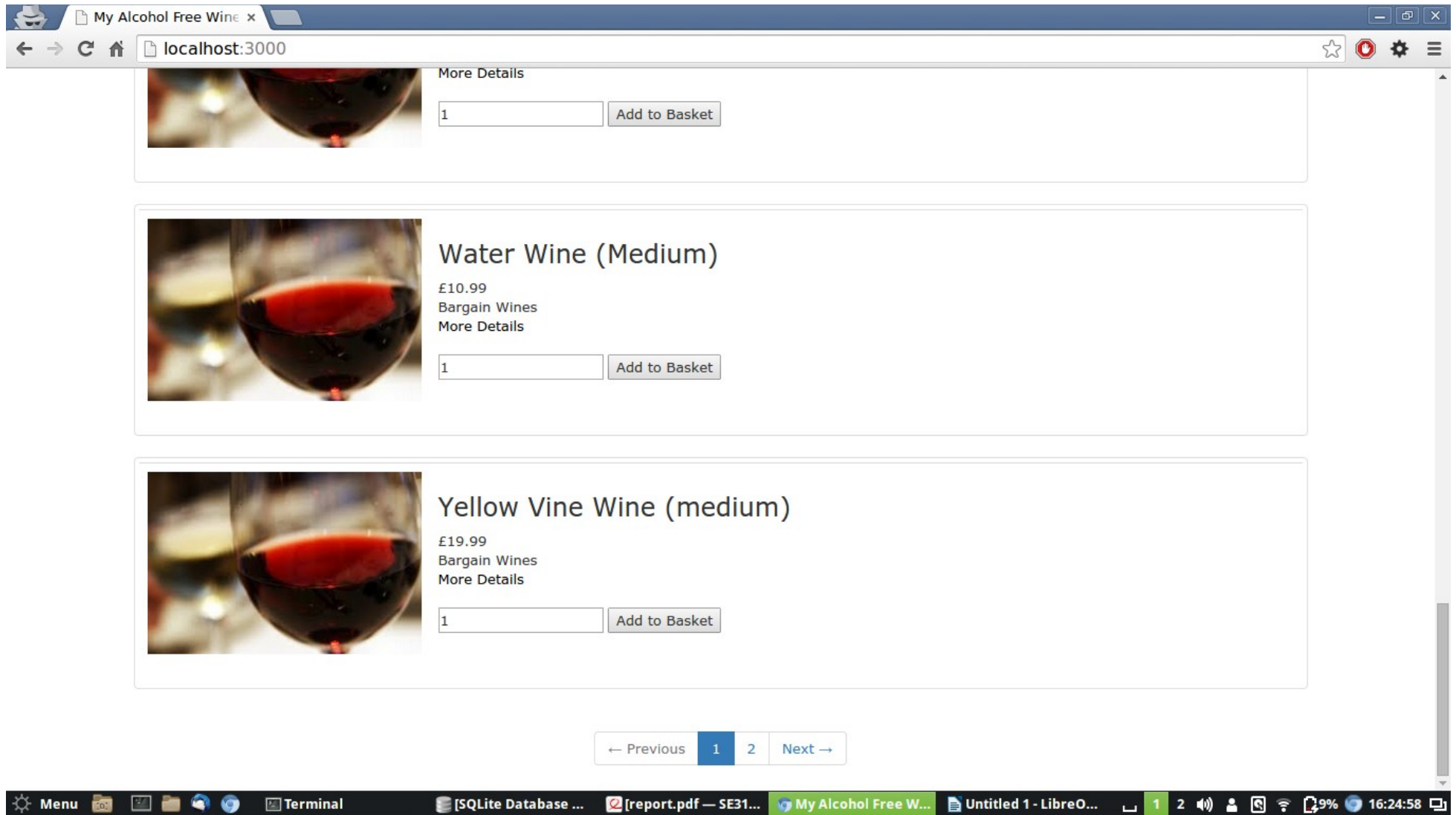
1 MAF App 2 Web Service

Menu [SQLite Database ...] [report.pdf — SE31...] New Tab - Chromiu... Untitled 1 - LibreO... 1 2 16:18 9%

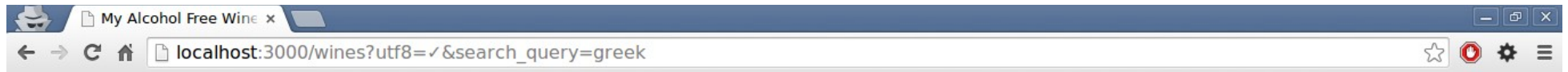
Screenshot 2: The web service is running. As reported in the report, I only have one service. This service is written in Python



Screenshot 3: The wines listings page showing the list of wines. These are retrieved from the web service and if they are not already in the database, they are added. This is done every time the page is visited. The wines are listed in alphabetical order and are paginated (with 10 per page).



Screenshot 4: Shows the pagination of the wines



# My Alcohol Free Wines

[Home](#) | [Login](#) | [Register](#)



## A Lovely Greek Wine (Small)

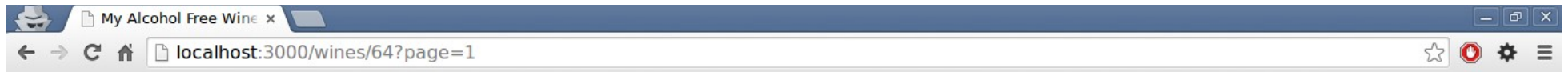
£1.99

Bargain Wines

[More Details](#)



Screenshot 5: Shows the search functionality working. The search function searches for wines that match partially or fully the string searched for. This searches from all data retrieved from the one web service.



# My Alcohol Free Wines

[Home](#) | [Login](#) | [Register](#)



## A Lovely Greek Wine (Small)

£1.99

Lovely and from Greece. Sweet.

**Origin:** Greece

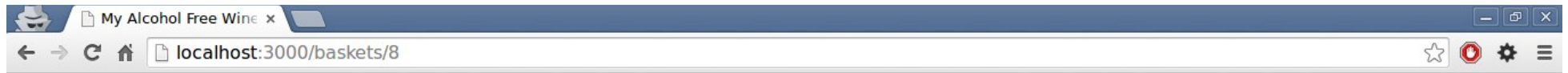
**Supplier:** Bargain Wines

**Grape type:** Chardonnay

[Back to listings](#)



Screenshot 6: Shows the 'Show' page for a particular wine. Shows all the information held on the wine. Also provides a quantity field for the user to specify a quantity of the wine.



# My Alcohol Free Wines

[Home](#) | [Login](#) | [Register](#)

## Your Shopping Basket



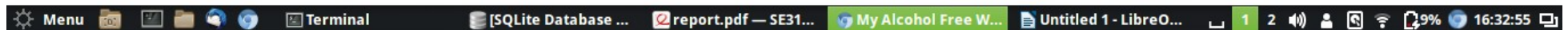
Some Very Nice Wine (small)

1

£89.99

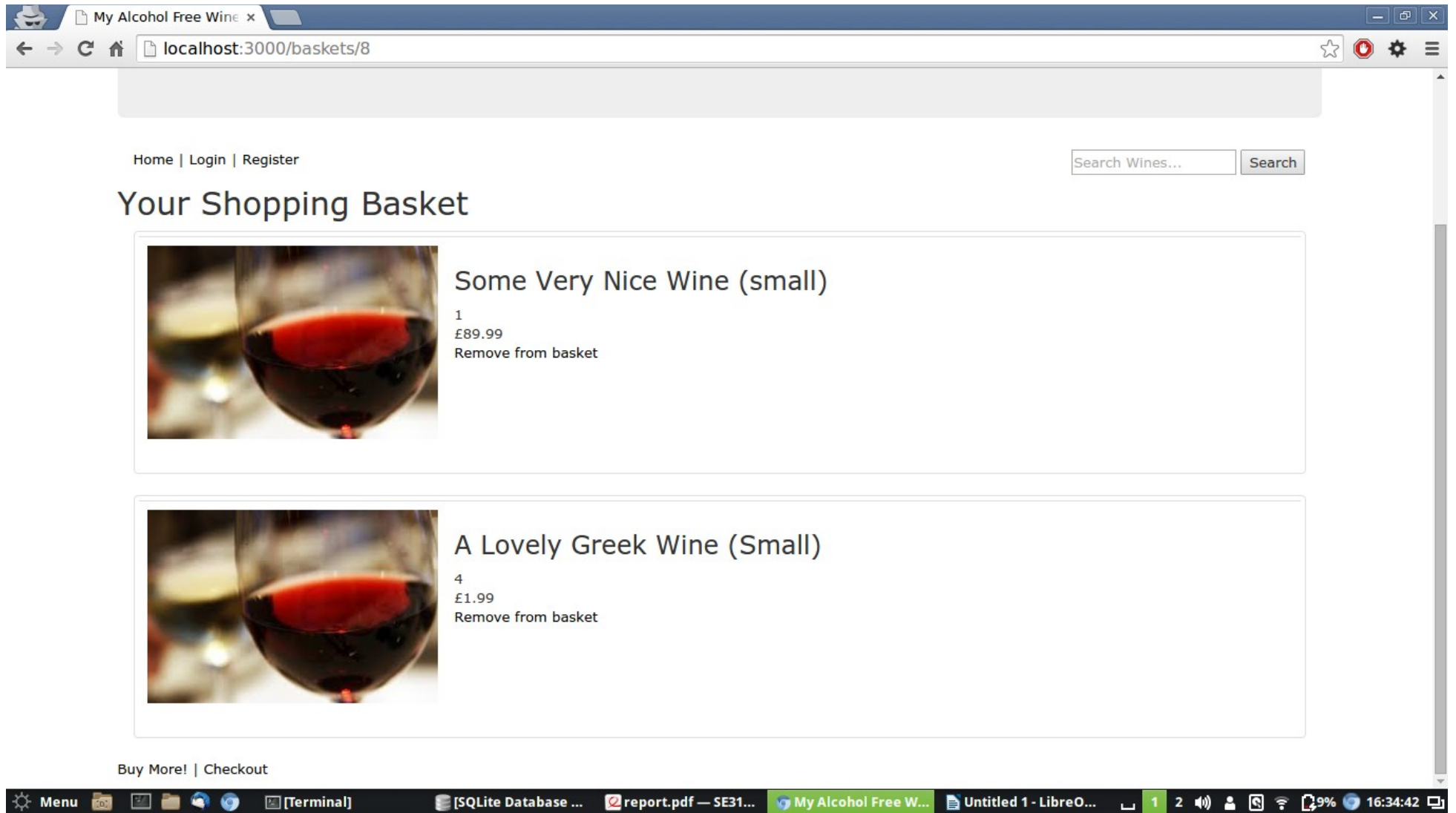
[Remove from basket](#)

[Buy More!](#) | [Checkout](#)



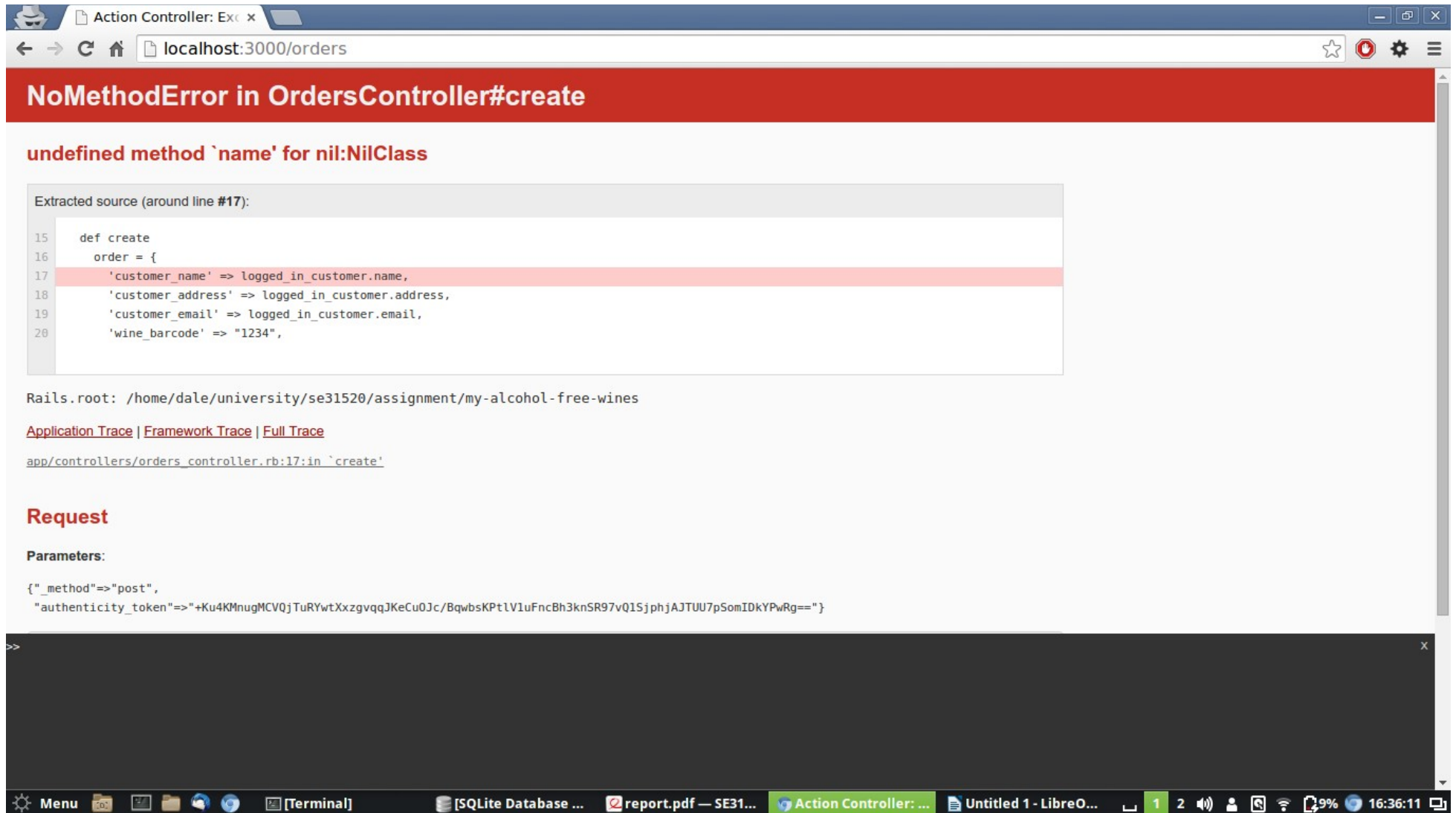
*Screenshot 7: Shows that wines can be added to the basket and that the user can see their basket. This is the only way in which they can see what is in their basket.*



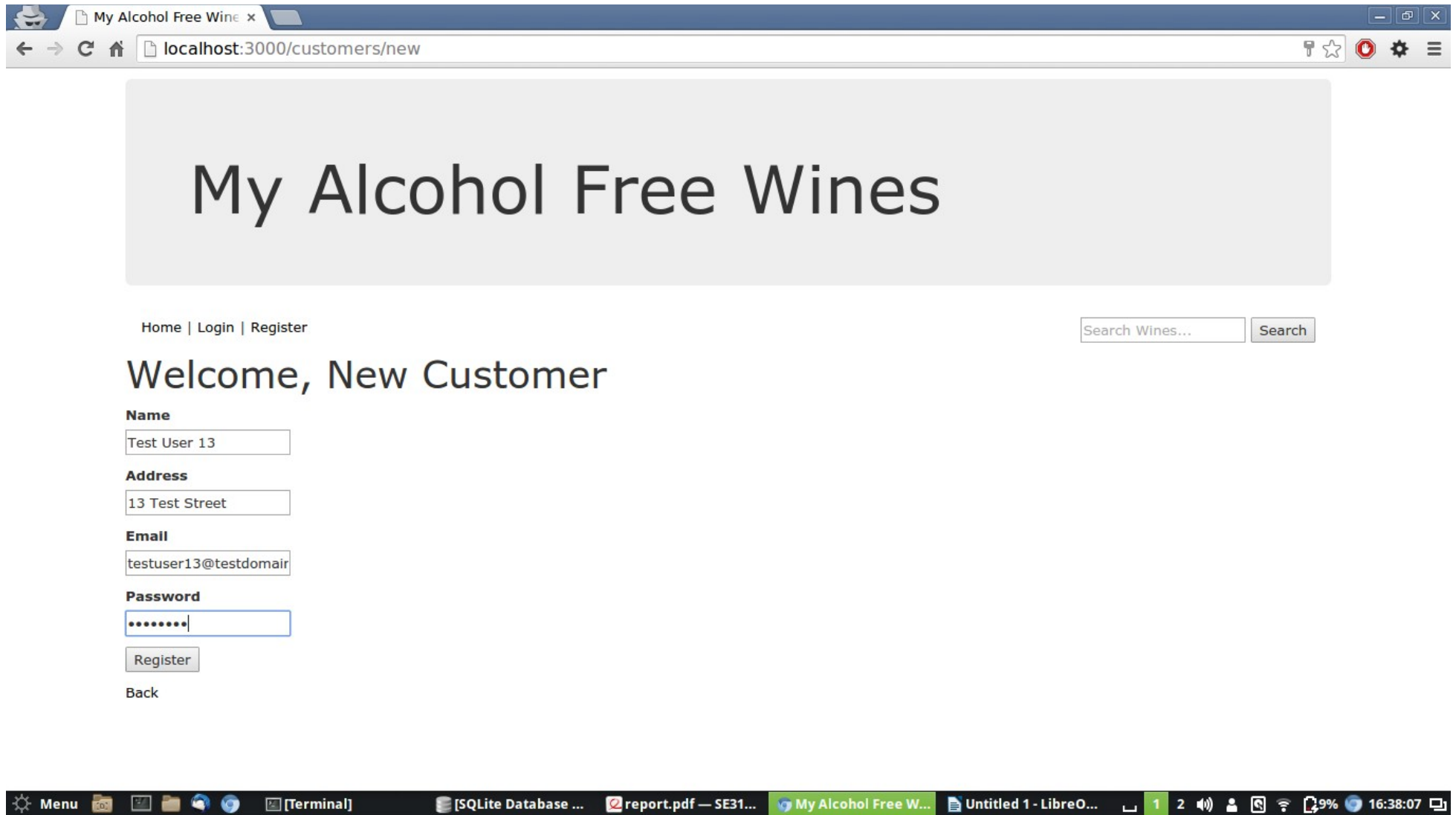


Screenshot 8: Shows that arbitrary quantities of a wine can be added to the basket

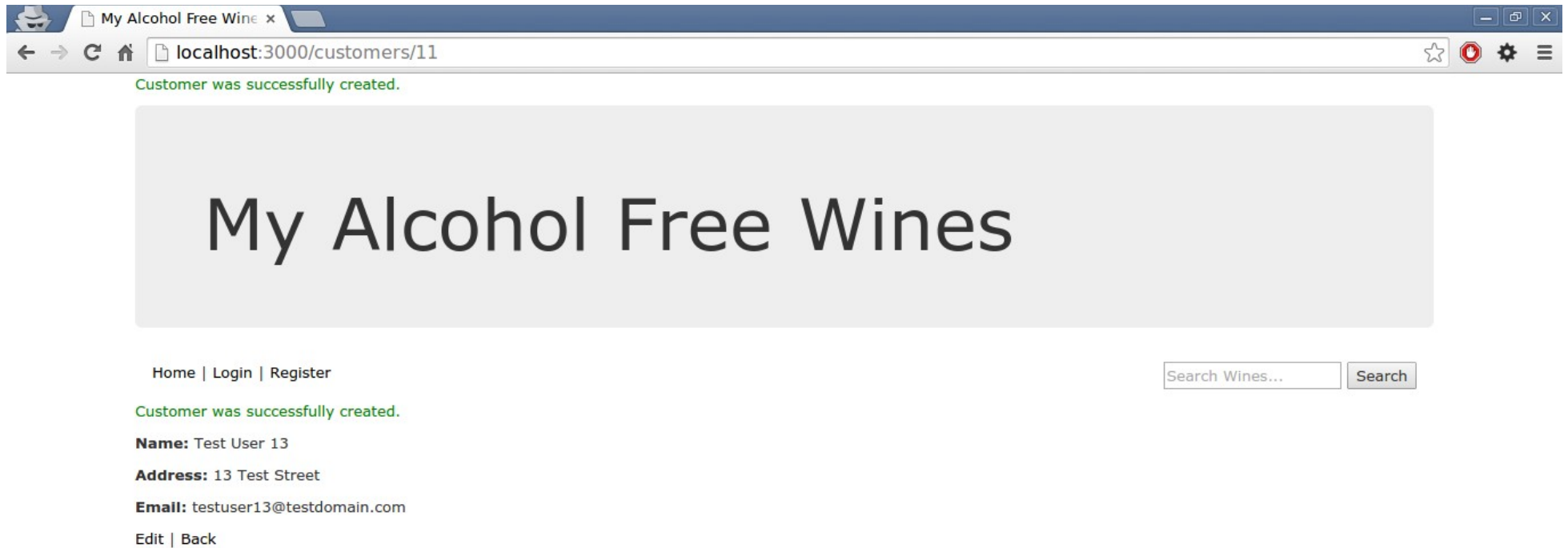




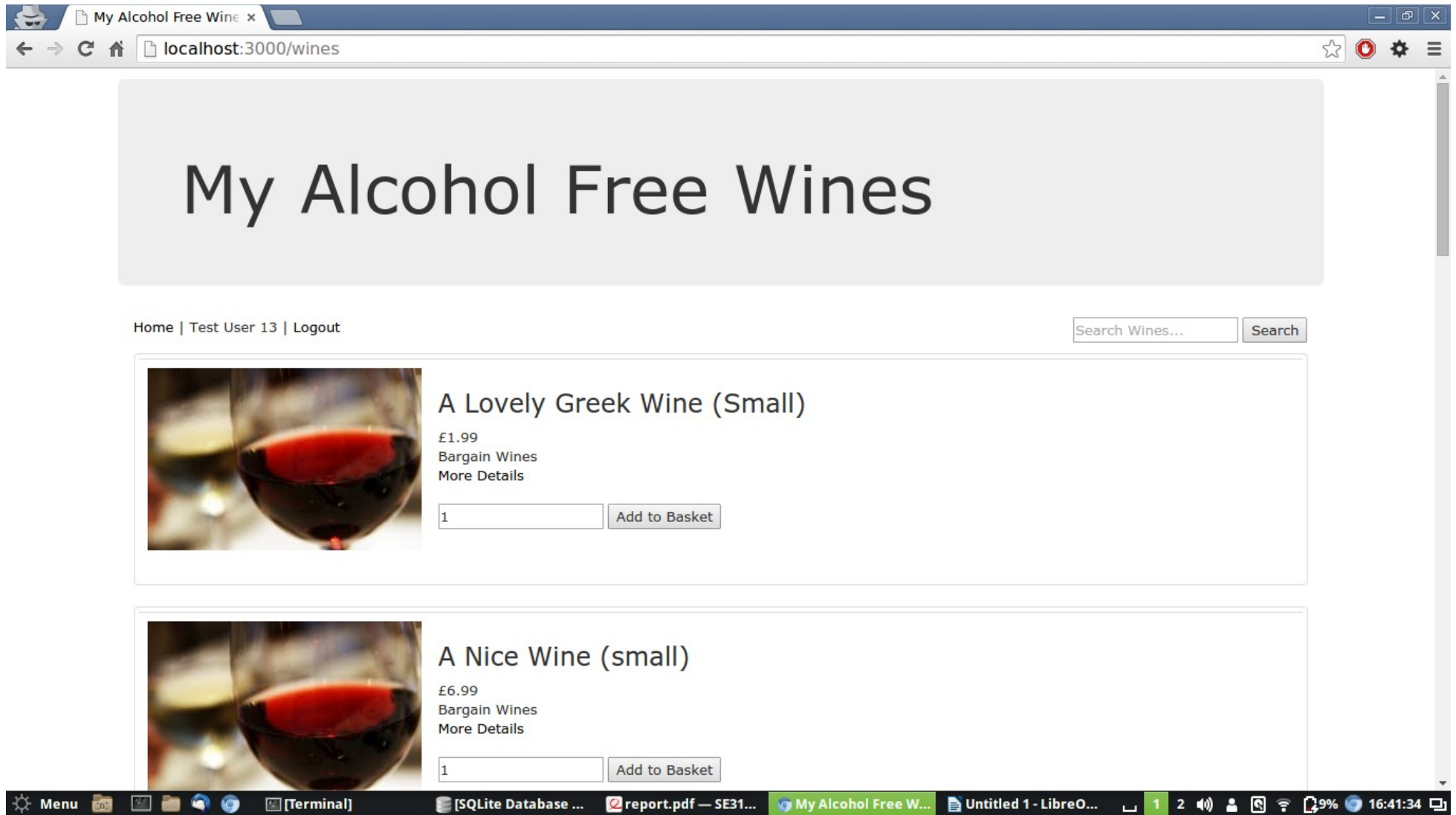
Screenshot 9: Shows what happens when Checkout is selected from the basket without logging in. Unfortunately I didn't get time to fix this, and so this requirement was not met.



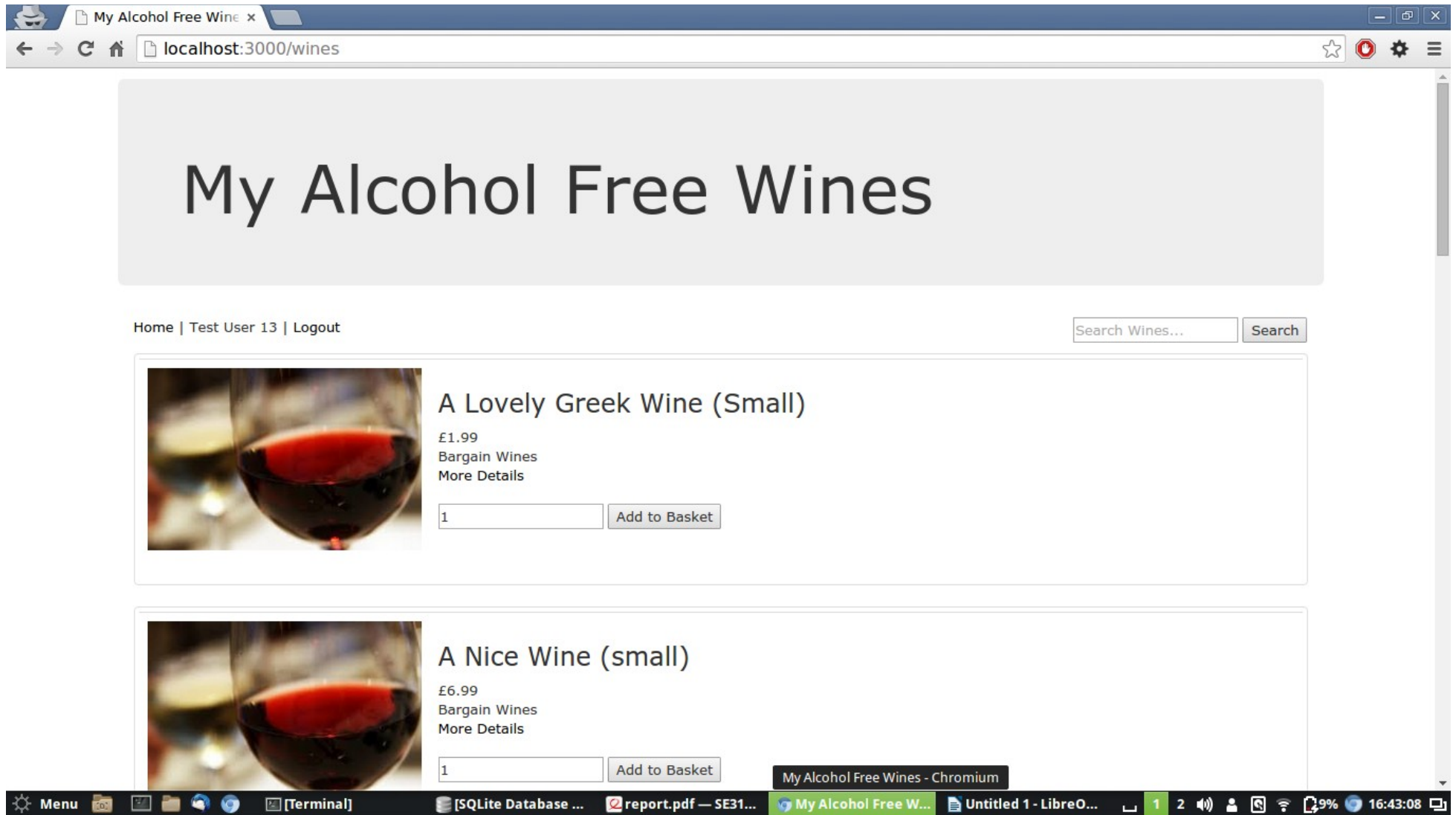
Screenshot 10: Shows the registration screen for new customers



Screenshot 11: Shows the page that the user is led to after registration. This page isn't on the requirements. If I was to be given more time, I would remove the ability to see this page and lead the user back to the wines page.



Screenshot 12: Shows that users can log in after registration. Login link disappears and the logout link appears.



Screenshot 13: The page the user is led to after checking out their order successfully (they must be logged in to do this)

```
Terminal
^C$
dale@dale-laptop ~/university/se31520/assignment/service (2.2.2) (master*)
> clear

dale@dale-laptop ~/university/se31520/assignment/service (2.2.2) (master*)
> python ./app.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
127.0.0.1 - - [06/Dec/2015 16:22:14] "GET /wines/ HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2015 16:22:18] "GET /wines/ HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2015 16:25:58] "GET /wines/ HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2015 16:30:45] "GET /wines/ HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2015 16:32:43] "GET /wines/ HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2015 16:34:29] "GET /wines/ HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2015 16:34:30] "GET /wines/ HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2015 16:37:38] "GET /wines/ HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2015 16:41:00] "GET /wines/ HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2015 16:41:30] "GET /wines/ HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2015 16:42:55] "POST /orders/ HTTP/1.1" 201 -
127.0.0.1 - - [06/Dec/2015 16:42:55] "GET /wines/ HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2015 16:44:29] code 400, message Bad HTTP/0.9 request type ('\\x16\\x03\\x01\\x01\\x1e\\x01\\x00\\x01\\x1a\\x03\\x03\\x91i\\xcb\\x7f\\xed\\xed\\x9a\\x02\\x01\\x00\\xc6\\xa2j\\x9c\\x03\\x00\\xb45\\x83\\xda')
127.0.0.1 - - [06/Dec/2015 16:44:29] "i\\x01\\x00\\xc6\\xa2j\\x9c\\x03\\x00\\xb45\\x83\\xda" 400 -
127.0.0.1 - - [06/Dec/2015 16:44:45] "GET /orders/ HTTP/1.1" 200 -

dale@dale-laptop ~/university/se31520/assignment/service (2.2.2) (master*)
> curl http://localhost:5000/orders/
{
  "orders": [
    {
      "customer_address": "13 Test Street",
      "customer_email": "testuser13@testdomain.com",
      "customer_name": "Test User 13",
      "quantity": 5,
      "wine_barcode": "1234"
    }
  ]
}
dale@dale-laptop ~/university/se31520/assignment/service (2.2.2) (master*)
>
```

Screenshot 14: Evidence showing that the POST request containing the order was received by the service. Notice that the Customer details are exactly the same as the ones I entered when I logged in, but the product details are wrong. I couldn't work out how to get these to send to the service, so I hardcoded these into the MAF application. While it wasn't a requirement to be able to issue a GET request to the service for orders, I implemented this anyway for debugging and testing purposes.

```
Terminal
127.0.0.1 - - [06/Dec/2015 16:51:58] "GET /wines/ HTTP/1.1" 200 - [0/386]

    "customer_name": "Test User 13",
    "quantity": 5,
    "wine_barcode": "1234"
  }
]
}%
dale@dale-laptop ~/university/se31520/assignment/service (2.2.2) (master*)
➤ curl http://localhost:5000/wines/
{
  "wines": [
    {
      "barcode": "091234",
      "bottle_size": "medium",
      "company": "Bargain Wines",
      "grape_type": "Chardonnay",
      "id": 1,
      "image_url": "wine.jpg",
      "is_vegetarian": true,
      "long_description": "Description of some great wine",
      "origin_country": "Germany",
      "price": 6.99,
      "short_description": "Some Great wine"
    },
    {
      "barcode": "091235",
      "bottle_size": "large",
      "company": "Bargain Wines",
      "grape_type": "Chardonnay",
      "id": 2,
      "image_url": "wine.jpg",
      "is_vegetarian": true,
      "long_description": "Description of some wine",
      "origin_country": "Germany",
      "price": 5.99,
      "short_description": "Some wine"
    },
    {
      "barcode": "091245",
      "bottle_size": "small",
      "company": "Bargain Wines",
      "grape_type": "Red",
      "id": 3,
      "image_url": "wine.jpg",
      "is_vegetarian": false,
      "long_description": "This is made from French grapes",
      "origin_country": "Germany",
      "price": 6.99,
    }
  ]
}
```

1 MAF App 2 Web Service 16:52 9%

Menu [Icons] Terminal [SQLite Database ...] report.pdf — SE31... My Alcohol Free W... Untitled 1 - LibreO... 1 2 [Icons] 16:52:13

Screenshot 15: Evidence that a list of the wines can be retrieved with a GET request issued to the /wines url of the web service.