Web Technologies - "Vent" Online Platform

Zsolt Varga
40212393@napier.ac.uk
Edinburgh Napier University - Web Technologies (SET08101)

Abstract

The 'Vent' platform is a fully functional web platform that utilises current technologies used for the web. The project consists of the client side, which has been made dynamic by the use of the EJS (Embedded JavaScript) templating engine.
The back-end software is a node.js server which provides an API that the front-end utilises. This is done by using several frameworks and packages discussed in this report, and MongoDB - a nonSQL database used for storing data.
During development, equal attention was given to styling, functionality, and user experience. The purpose of this document is to record the process of creating a web platform from scratch and to reflect on the process.

Keywords – coursework, Edinburgh, napier, university, zsolt, varga, web, technologies, front-end, back-end, server, node, node.js, express, javascript, html, css, vent, blog, website, dynamic

1 Introduction

This project was created as a coursework attempt for Edinburgh Napier University's "Web Technologies" module. The assignment for the coursework was to create a fully functional, robust blog-like web platform. Vent adheres to all these requirements and implements many additional features, to demonstrate proficiency in both front-end and back-end web development.

Vent is an online social network resembling a Tumblr/Facebook hybrid, based on the idea of anonymity and free opinion sharing. The platform supports multiple users, each with their own account. Users are able to create vents (individual posts) and save them. Each vent has a category - confession, rant, or ridiculous. Based on these, vents are color-coded. Users can also create groups, where vents are posted. These groups can be looked up in the search page, followed, and unfollowed. Groups and vents can be edited and deleted only by the person who created them. Users can also comment on vents, however, these cannot be deleted or edited, and most importantly, are not anonymous.
All this functionality has been implemented using various technologies on the server-side. The node app uses the express.js framework and several npm packages discussed in this report. The choice of these technologies was strongly influenced by Colt Steele's Udemy course "Web Developer Bootcamp" [1].

2 Software Design

When compared to the first coursework, this project needed considerably more planning and for development choices to be made before implementing anything. This was a bit tricky to do, as the technologies used were also continually learn throughout the development of the project.
The file hierarchy for one, has a significantly different layout than what was planned for it to be (see figure 1). The whole directory contains files and folders used by the app.js node file. Out of these, only the public folder is used to contain all the static content - images, CSS stylesheets and libraries, and scripts.

2.1 Front-end Software

To make the platform dynamic, a choice of a templating engine had to be made to inject data into the HTML. Considering the amount of online resources, the decision came down to using EJS (Jade being the other option). Using EJS, templates were created for each page, along with several partial templates to make the code DRY.

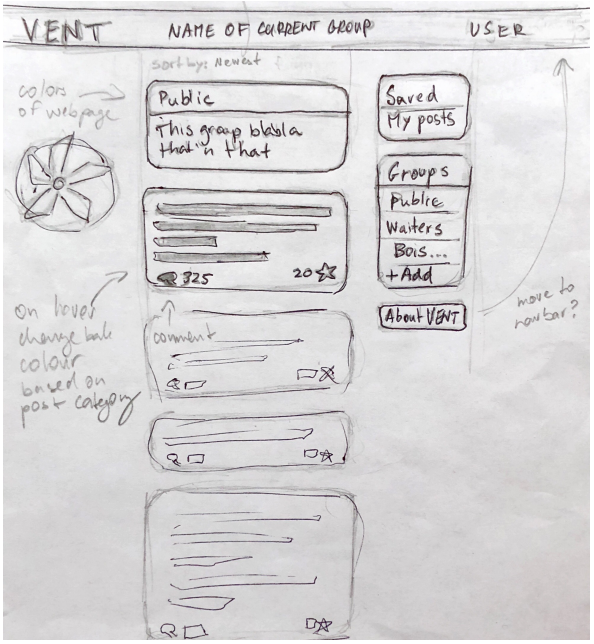

Figure 2: Sketch of the layout of a Vent group

There have only been a few imports for the client-side: the font-awesome library, to include simple icons, and the "Roboto" font provided by Google's web-fonts. The most important one, however, was 'Quill' - an open-source JS library used for rich text editing. [2] By implementing this, the

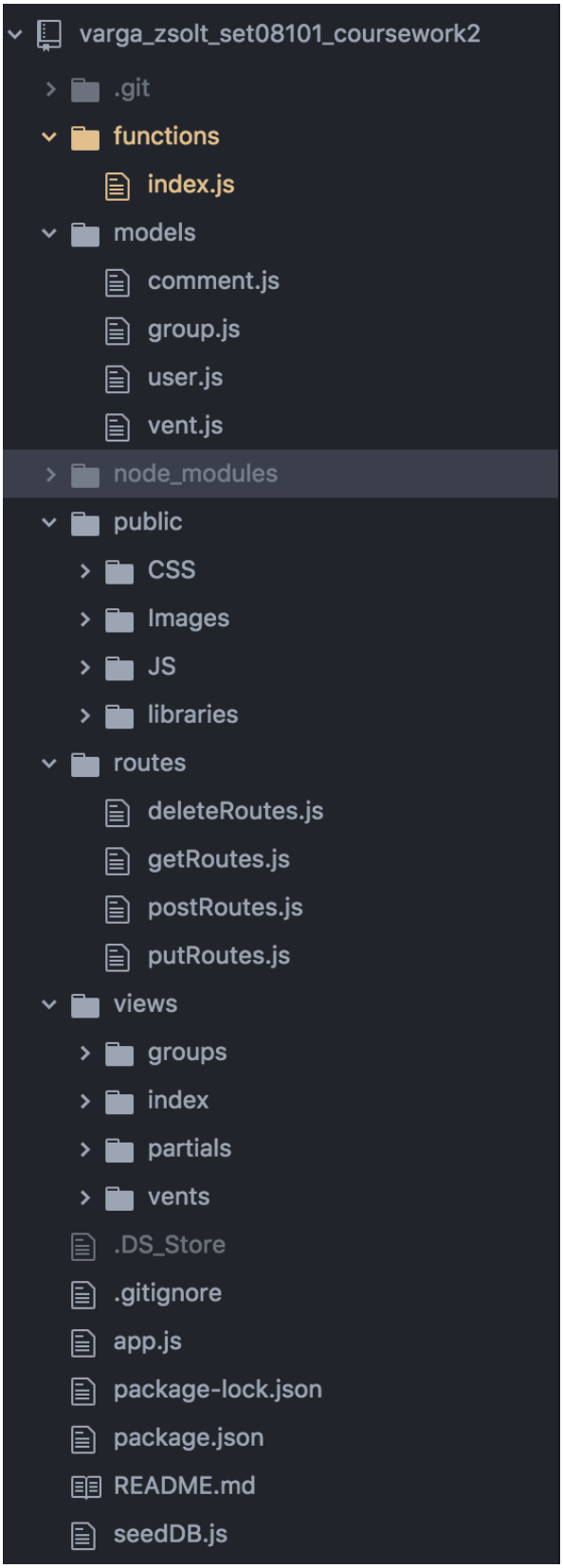

Figure 1: Hierarchy of the project directory

users are able to format their vents and group descriptions. To keep the overall look of the website consistent, only a few formatting options have been granted, the font-family not being one of them (see figures 13 and 14). The rest of the styling has been done manually with plain CSS. This was one of the most important decisions regarding the client-side. Opting to not use any CSS library slowed down the development, but on the other hand, gave complete control over the look and behaviour of content.
Most of the selectors in these stylesheets are classes, and higher specificity selectors are only introduced in very concrete cases, e.g. when only the last div on the page should have a larger bottom margin. Inside the HTML, styling is added by combining the CSS classes. For this purpose, it is important that the CSS classes are atomic enough but not too atomic. This way each tag has about 2-3 classes on average, with very little redundant styling assigned to it.

Planning the layouts of the pages was one of the very first things during development, and the finished prototype came very close to looking like the sketches. (see figures 2 and 3) Every page has a navbar on the top, and a centered layout for the actual content with a sidebar on the right. The sidebar serves as an easy navigation tool between groups and vents, whereas the navbar provides user actions and the homepage link. (see figure 10).

2.2 Server-side Software

To use node.js with the express framework was a very straightforward decision, considering these are current industry standards and go-to technologies. A major part of the server-side software is the data storing. Even though it was not crucial to use a database for the platform, it was certainly beneficial to learn a new technology. MongoDB was chosen since it's currently fairly common with web developers and has great support for node and express.

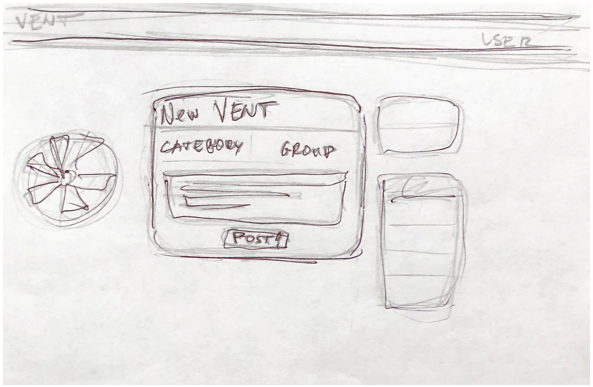

Figure 3: Sketch of a form for creating a new vent

Several packages were used for this node.js project, all of which can be found in the package.json file. This file was created upon initialisation of the project, and provides a handy reference and completely automates the installation of all dependencies.

The packages used are as follows:

- **express**: an industry-standard framework on node.js servers [3]

- **ejs**: templating engine used to inject data into HTML [4]
- **body-parser**: parses the content of put, post, delete (etc) requests into a javascript object [5]
- **method-override**: allows us to place put and delete requests from HTML forms [6]
- **mongoose**: MongoDB modelling tool that builds on the MongoDB package native to express and provides extra functionality [7]
- **express-session**: allows sessions, the ability to have a user logged in [8]
- **passport**: provides user authentication [9]
- **passport-local**: provides the local strategy for user authentication
- **passport-local-mongoose**: mongoose models able to securely store passwords and user information.

3 Implementation

```
var mongoose = require("mongoose");

var ventSchema = mongoose.Schema({
  date: {
    type: Date,
    default: Date.now
    },
  content: String,
  category: {
    type: String,
    enum: ['rant', 'confession', 'ridiculous'],
    default: 'rant'
    },
  favourited: {
    type: Number,
    min: 0,
    default: 0
    },
  comments: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: "VComment"
      }]
});

var Vent = mongoose.model("Vent", ventSchema);

module.exports = Vent;
```

Figure 4: Vent Schema using mongoose for MongoDB

3.1 Data Persistence

Even though MongoDB is a non-relational database it is important to have a good structure in the stored data and avoid duplicates, redundant data, and most importantly missing data. For this reason, data validation has been implemented in the project and to make life easier, this was done with the use of the Mongoose npm package. As can be seen on figure 4, creating schemas and models was very straightforward, and made it possible to keep the data consistent.
The greatest benefit of the non-relational model was with no doubt the ability to nest objects where necessary. We believe the data should never be stored embedded, only as references. What proved to be a great advantage however, was the ability to nest the objects on the go and provide the EJS templates with a single (or very few) objects that can be then easily navigated through with dot notation.

3.2 Templating and User Authorisation

The server software takes advantage of this fact, for example, when accessing the get route for a concrete vent. On this page, we need to display the vent itself, the comments associated with the vent, and the name of the users that wrote the comments. See figure 15. As we only store id's and lists of id's inside an object, we find the vent by the id, populate the list of comments inside the id, and then populate the name of the comment's author. After we populated this data, we pass the single double-nested object to the template (figure 5). Additionally, since we do not actually store the comment data inside the vent, and the user data inside the comment, we can be sure that any changes to the database will update everywhere.

```
router.get("/vents/:ventID", functions.loggedIn, function(req,res){
  var ventID = req.params.ventID;

  Vent.findById(ventID).populate({path: "comments", populate: {path: "user"}}).exec(
    function(err, foundVent){
      if(err){
        console.log(err);
```

Figure 5: Populating a vent object's nested ID's

With this approach, the aim is to make the ejs templates use as little javascript as possible and keep them clear and readable.
The other thing concerning templates, is the ability to take the user's identity into account. For this reason, there are some locals that are used by every route, declared in an app.use statement (figure 6). One of these locals is the req.user object. Based on this object, some controls in the client-side change accordingly. For example, the edit icon on a vent or a group only appears when the user is the owner of this item.

```
app.use(function(req, res, next){
  res.locals.currentUser = req.user;
  if(req.user){
    Group.find({_id: {$in: req.user.groups}}, function(err, followedGroups){
      if(err){
        console.log(err);
      } else {
        Group.find({creator: req.user._id}, function(err, myGroups){
          if(err){
            console.log(err);
          } else {
            res.locals.allFollowed = followedGroups;
            res.locals.allMine = myGroups;
            next();
          }
        })
      }
    })
  } else {
    res.locals.allGroups = null;
    res.locals.allMine = null;
    next();
  }
})
```

Figure 6: app.use statement providing locals used by every route

This, however, still does not prevent someone to use postman or any other HTTP service to send a put or delete request and edit data they should not have access to. As a solution, an

object called functions was created which contains functions used throughout the whole app. As an example, let's take the ownsGroup function. This function is used as a middle-ware in the put and delete routes of a particular group (figure 7).

```
functions.ownsGroup = function(req, res, next){
  if(req.isAuthenticated()){
    Group.findById(req.params.groupID, function(err, foundGroup){
      if(err){
        console.log(err);
      } else {
        if(String(foundGroup.creator) === String(req.user._id)){
          next();
        } else {
          res.render("index/userError", {msg: "You do not own this group."});
        }
      }
    })
  } else {
    res.redirect("/login")
  }
}
```

Figure 7: The ownsGroup middle-ware function used to check if the user owns the group to-be modified

The function first makes sure the user is logged in, then compares the group admin's id with the user's id. If this fails, the user is prompted with an error message, or in the case the user is not logged in, they are redirected to the login page.

```
49        <div class="bubble site!ransp">
50            <div class="bubbleContent">
51                <div class="commentMessage">
52                You do not own this vent.
53            </div>
54            </div>
55        </div>
```

Figure 8: Returned HTML when using Postman to send a PUT request to edit a vent not owned by the user

When the user is not logged in, the only pages that can be accessed are the index page and the login and register pages. The rest of the routes use yet another middle-ware, which simply checks if the user is currently logged in and redirects to the login page if that's not the case.

3.3 Error Handling

Similarly, with any other error that could occur (non-existent id, wrong url, database crash, etc.) the website handles these with two simple ejs templates (figure 17), making sure in case an error occurs, the user "stays on the website". One is for a non-logged in visitor, and the other is for a logged user. The only difference is the absence/presence of the user's sidebar. Based on the origin of the error, a message is passed into the error template, making it a lesser pain for the user's aesthetic sense.

3.4 Design

When considering what design to use for the website, the purpose of the website was kept in mind. Since Vent is about sharing opinions and creating small communities of people with similar views, the whole layout was sliced into two parts (figures 10 through 17). The left-hand side provides space for the actual viewed content - in most cases, this is a list of vents. A vent is usually a single opinion, therefore most of the elements on the website have a rounded corners to resemble a speech bubble. The right hand side is reserved for the user's sidebar. This is used to easily navigate between followed groups and favourited vents. Every user has a custom combination of followed groups, therefore every sidebar is personal, and only appears when the user is logged in.

The central motive of the site is the vent logo, which is a simplistic ventilation fan, which was chosen for a simple reason. The fan stands for ventilation/venting, and is supposed to "cool down" the user. During scrolling, the fan underneath the sidebar sticks to the top of the page and spins - this is one of the key design elements of the whole website. All imagery (except for the free font-awesome icons) was created for this project by the author.

The chosen colour palette is a very neutral blue. The website itself does not take sides or promote views, hence the calm tone. However, vents by definition should be based on emotions, therefore every vent is colour-coded based on the category. There are three categories - Rant, Confession, Ridiculous - with their colours red, green, and yellow (respectively). These colours appear when the user hovers over a post (figure 12) and also serve as a quick and vague indication of the content of the vent.

Another important decision to make was the typeface. The website ended up using the 'Roboto' font family. This font is a sans-serif type very similar to Arial. In a both figurative and literal way, it is a bold font, so it's great to use for presenting one's opinion. The typeface also comes across as playful, which might remind users that the purpose of Vent is to create a familiar community, rather than a place of conflict.

4 Critical Evaluation

The basic requirements for the coursework were all met, and it is safe to say they were exceeded as well. However, there is still room for improvement. In the following section, we will contrast well developed features with weak spots in the implementation.

4.1 CRUD and RESTful routing

The server software provides a fully functional API that enables the client to utilise all basic CRUD functions. The users are able to create and retrieve accounts, groups, vents, and comments. Out of these, update and destroy capabilities have been added only to groups and vents - leaving some healthy restrictions.
All non-public routes use middle-ware functions to make sure the visitor is authorised to access the path. Mainly the Put and Delete routes had to implement more complicated logic to check if the user owns the content they are about to edit

Possible Improvements
The routes hierarchy in particular, is a little sloppy. There has been an attempt to make the route names uniform, however there are some exceptions to the pattern - mainly with get and post routes. For instance, every get route for vents starts with "/vents". However, the route for retrieving the "new vent" form is as follows: "/groups/:groupID/vents/new". A

possible solution for this instance would be to simple attach the groupID parameter to the end of the route.
It would have been beneficial to plan the routes before implementing any of the server side, and make sure they follow RESTful conventions.

4.2 User Interface

The user interface had to reflect the routes as well, and adjust the visible controls based on what the user is able to access. Therefore, a logged in user can only see the groups they follow in the sidebar, and only have edit/delete controls where the content is theirs.
As both the routes have been secured, and the front-end reflects these restrictions, the user is not misled by UI controls they are not authorised to use.

Possible improvements
Choosing not to use any CSS libraries was a good choice, when we consider the amount of control this decision gave over the styling of elements and layouts. However, this way the website lacks responsiveness that libraries such as Bootstrap provide. Therefore, the site looks not particularly good on a mobile or tablet and this is definitely one of the first improvements to be made in the future.
Similarly, support for other browsers than Google's Chrome has not been considered during development, so there might be some UI flaws when testing in different browsers.

4.3 Data persistence

Using a noSQL database for the website was very useful, since it provides features and data formats that translate easily into the javascript web environment. A decision has been made to not take full advantage of the features MongoDB provides, but to follow some principles of the relational model. Schema objects refer to others exclusively by id's and lists of idâĂŹs (can be thought of as the foreign key). Storing whole objects inside other objects was avoided at all cost, and this way no redundant or duplicate data is stored.

Possible improvements
Again, as this was a new technology learned on the go, thorough planning wasn't possible and resulted in a few small mistakes in the schemas. Namely, the vent schema contains a field called 'favourited', which represents the number of people that saved the vent. This is stored as an actual number, instead of a list of user ID's. This results in unnecessary calculations when favoriting a vent. The problem is an easy fix, but shows how the lack of planning resurfaces in the small details.

4.4 Security

The security of the website has not been considered enough to meet industry standards. One of the biggest risks at the moment is the use of a text formatting library, which may be used for injections of JavaScript. In case of deploying the platform, security and protection of user details should definitely be one of the concerns high up on the list.

4.5 Other features

For future versions of the vent platform, some additional features could be implemented that were perhaps not in the scope of the module or simply too trivial to implement for this version.

- For the vents, there should be a wider spectrum of categories/emotions as opposed to just three.
- Higher security (mainly protection against JS injections)
- User profiles and private messages

5 Personal Evaluation

This project has been very helpful for personal growth and mainly overcoming the faced obstacles taught some valuable lessons. The work-flow of designing the website as well as the back-end API came with some interesting challenges.

Most of the head-scratching was induced by the mongoose package for MongoDB. The way every query needs to contain a call-back function, where the queried object(s) can be accessed, helped understand more of how javascript works. Since the mongo database does not necessarily finish queries in the order they were sent, it was important to utilise these call-back functions and nest queries in the order they had to be executed. This resulted in an odd cascading style of the code, which was certainly a new style of coding to get used to.

The same applies to the very heavy use of javascript objects in a web development workflow. By the end of the project I have noticed that I have grown to love these data structures.

Another part of the work-flow was the utilisation of git and GitHub. It has become a pattern that with every project, new kinds of problems resurface, and I learn a bit more about the way git works. At the very start of the project, it was important learning what .gitignore is, so that npm packages weren't pushed to the online repository for no reason. Another thing was the user configuration on the local repository. For the most part of the project I have been committing changes without having configured the user details, which resulted in the commits not being associated with my GitHub account.

The project has been approached with a very healthy time schedule. This was certainly a beneficial factor which provided enough time to research the technologies as well as implement most of the desired features, resulting in a pleasant development experience in overall.

6 Conclusion

Using current technologies, we built a fairly robust and fully functional web platform, that can be used to share opinions called vents, create groups, comment, follow and unfollow groups, and also edit and delete content owned by the user. The project tries to follow the RESTful routing principles, has all CRUD capabilities, and exceeds the requirements set by the module's assignment for this coursework.
The user experience has been kept in mind when designing the interface, to ensure that the way visitors interact with

the site feels natural and straightforward.
Even though improvements could be made to the security and architecture of the API as well as the client side, a great number of things have been learned during the development of this project.

References

[1] C. Steele, "Web development bootcamp." https://www.udemy.com/the-web-developer-bootcamp/learn/v4/overview. Accessed: 2017-11-14.

[2] J. Chen, "Quill." https://quilljs.com. Accessed: 2018-02.

[3] StrongLoop, "Express." http://expressjs.com. Accessed: 2018-02.

[4] M. Eernisse, "ejs." https://www.npmjs.com/package/ejs. Accessed: 2018-02.

[5] D. Wilson, "body-parser." https://www.npmjs.com/package/body-parser. Accessed: 2018-02.

[6] D. Wilson, "method-override." https://www.npmjs.com/package/method-override. Accessed: 2018-02.

[7] A. Heckmann, "mongoose." http://mongoosejs.com. Accessed: 2018-02.

[8] D. Wilson, "express-session." https://www.npmjs.com/package/express-session. Accessed: 2018-02.

[9] J. Hanson, "passport." http://www.passportjs.org. Accessed: 2018-02.

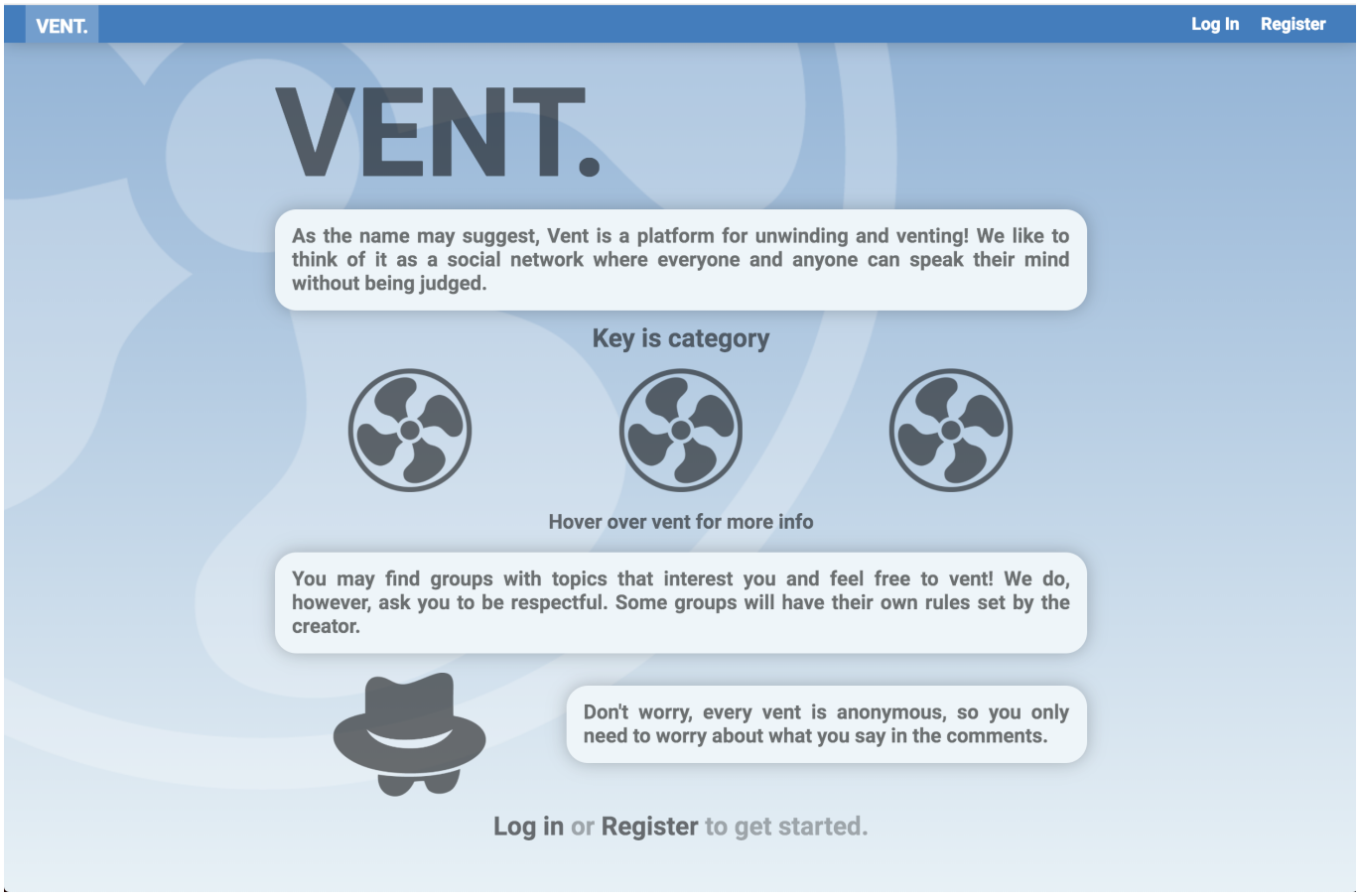

Figure 9: Screenshot of the index page when viewed as a non-logged in visitor

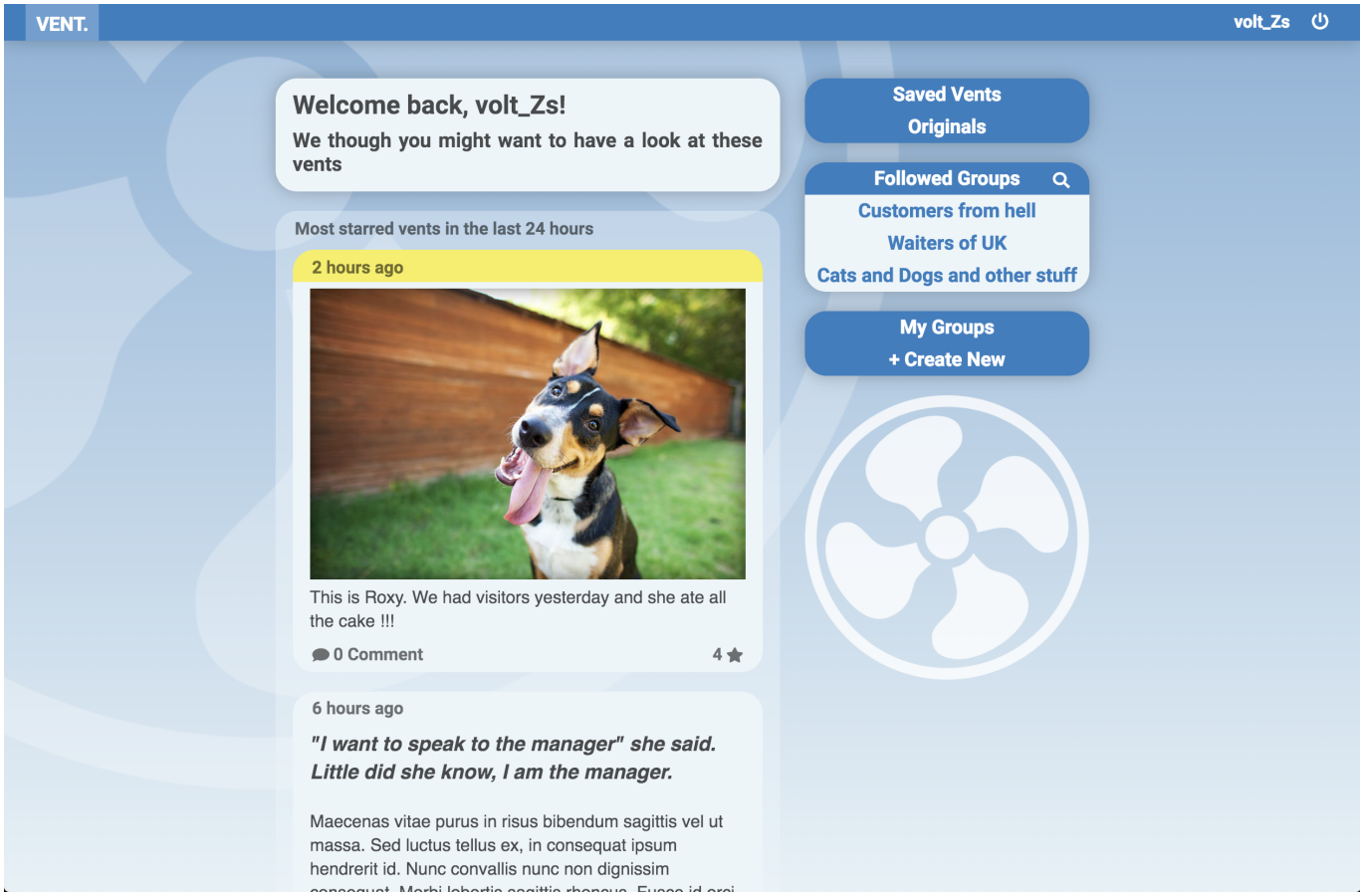

Figure 10: Screenshot of the welcome page after login

Figure 11: Screenshot of the welcome page as a newly registered user

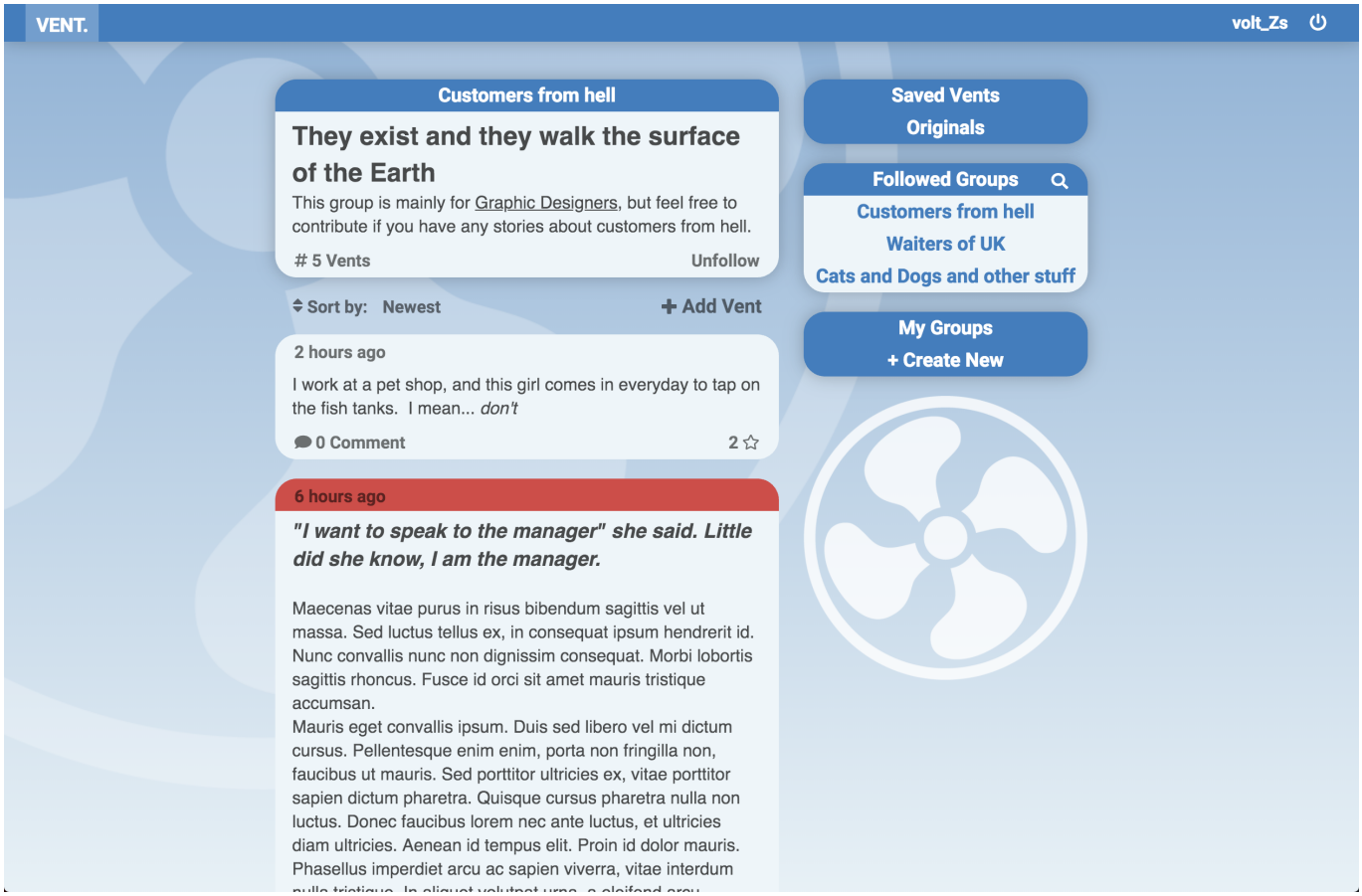

Figure 12: Screenshot of a group - vents sorted by newest first

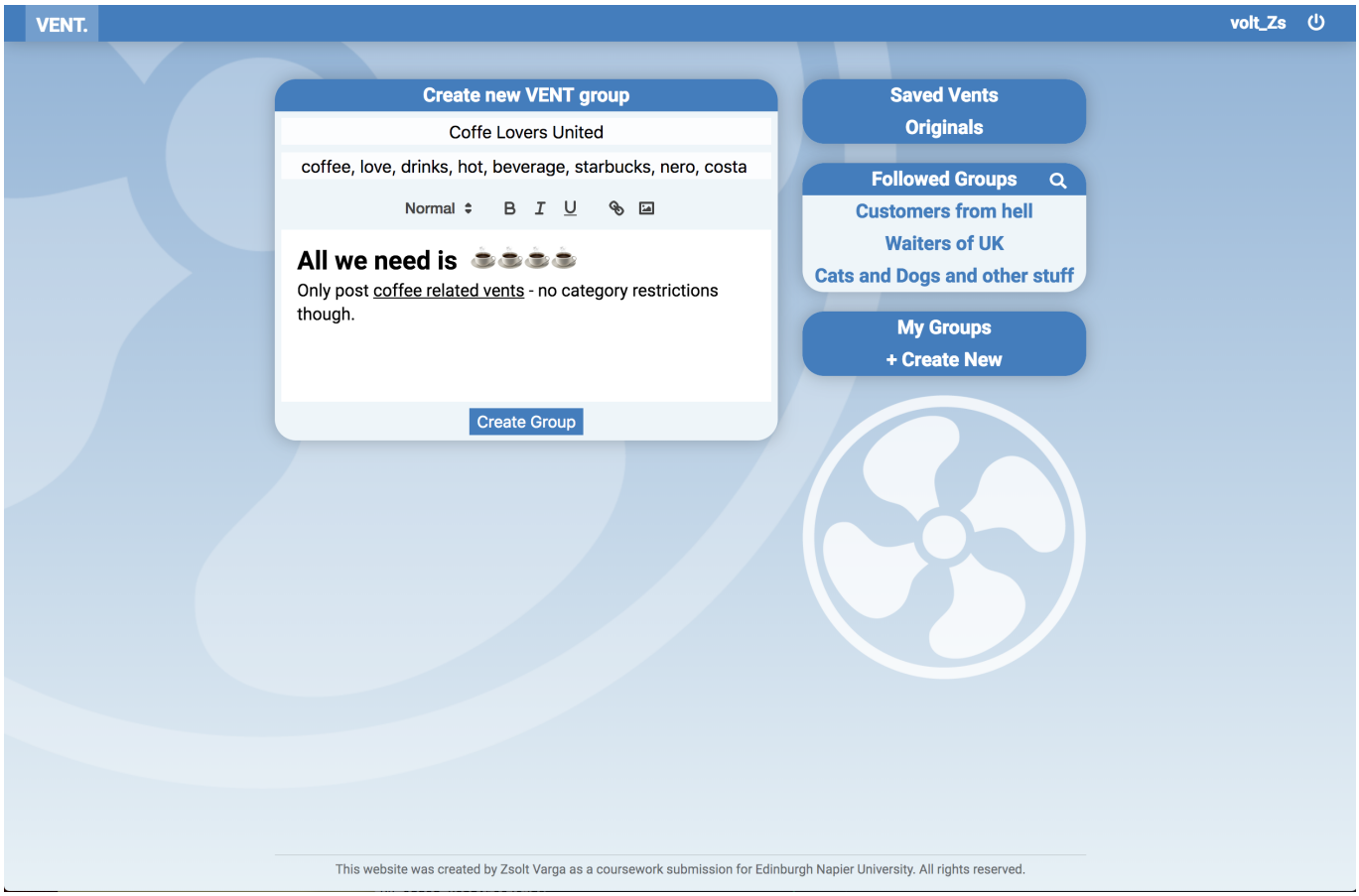

Figure 13: Screenshot of the 'new group' form

Figure 14: Screenshot of the 'new vent' form accessed within a group

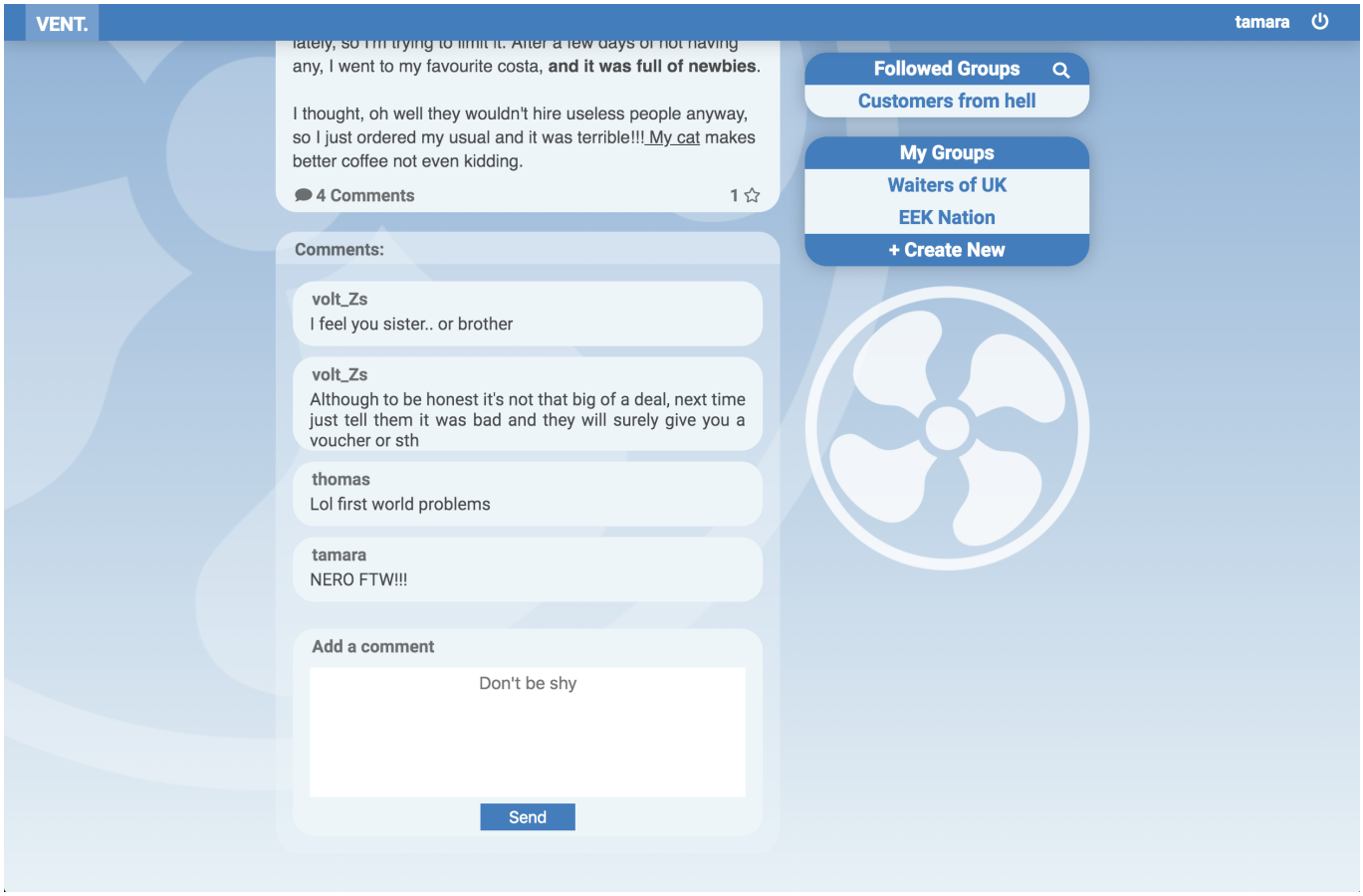

Figure 15: Screenshot of the comments page of a single vent

Figure 16: Screenshot of the search page

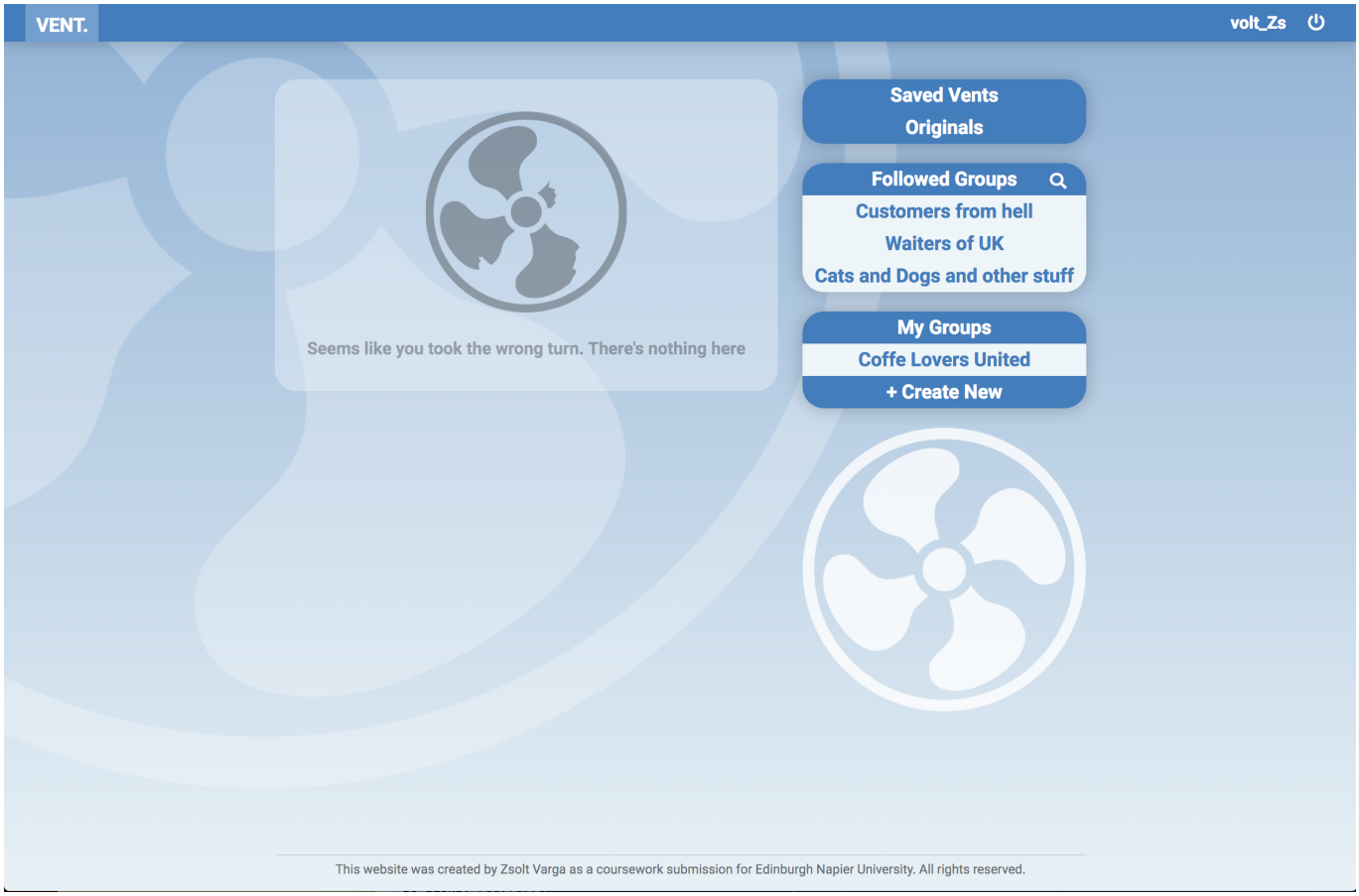

Figure 17: Screenshot of an error page as a logged in user

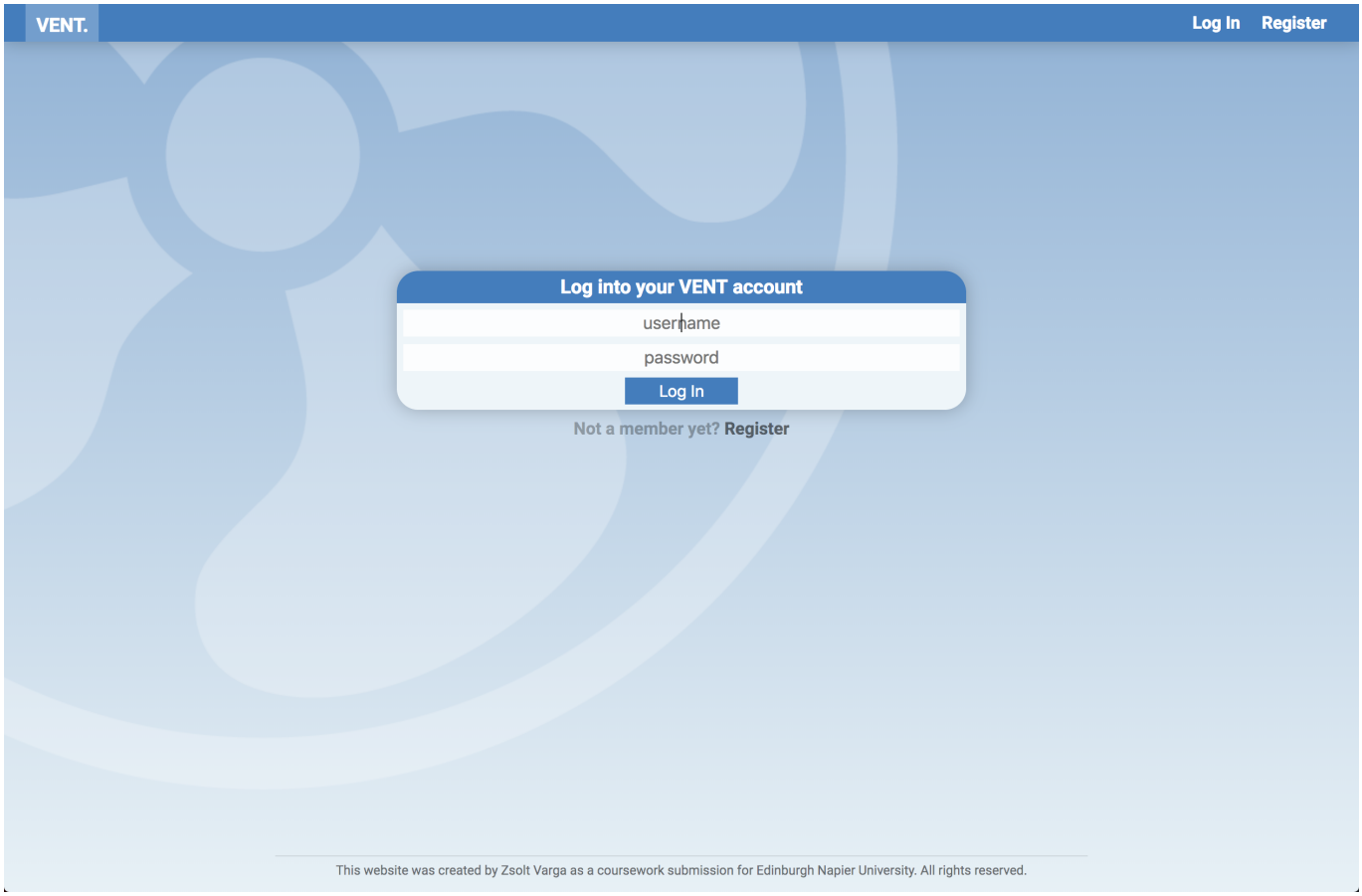

Figure 18: Screenshot of the login page

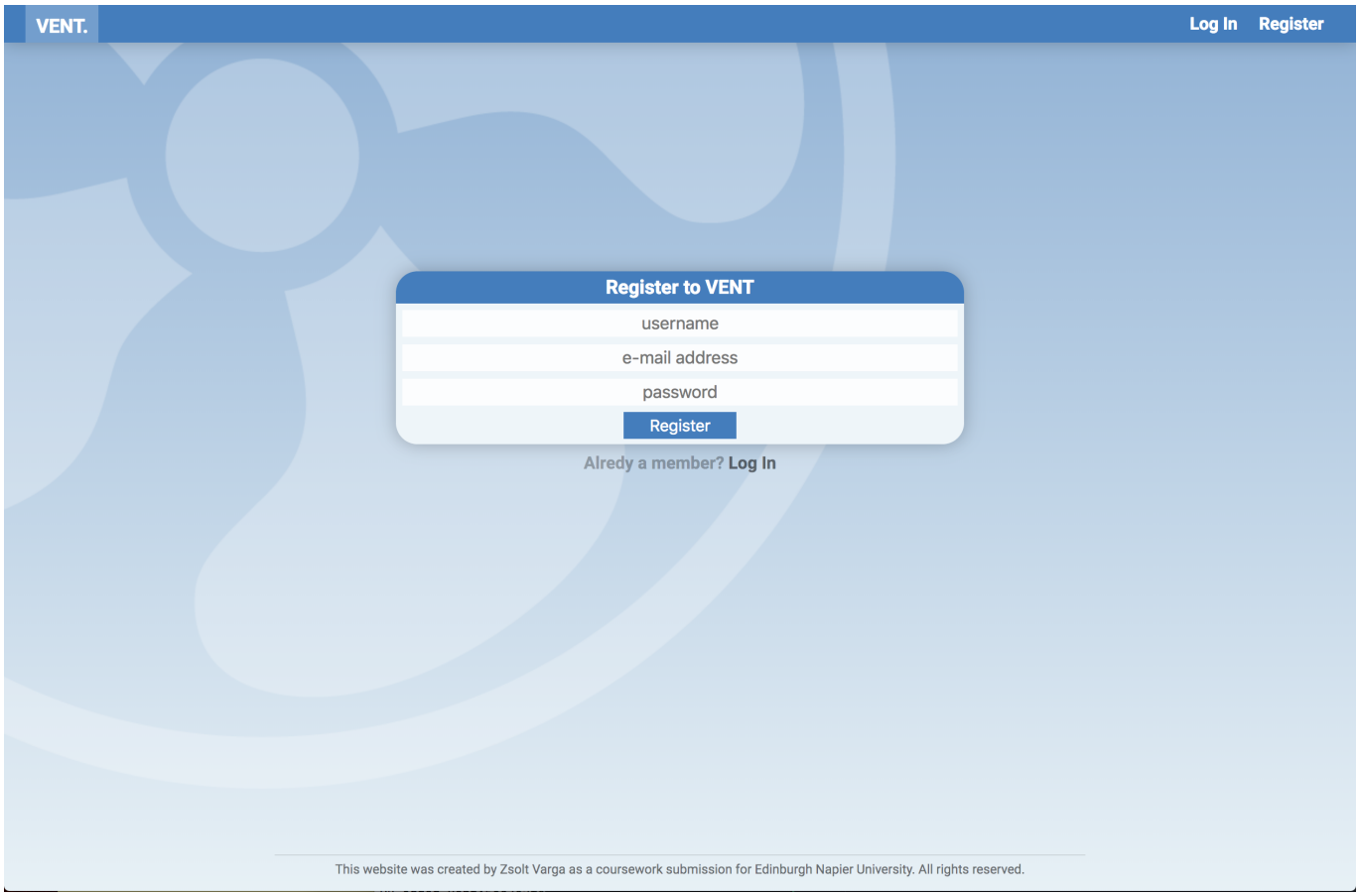

Figure 19: Screenshot of the register page