ENTWA Project Ideas

*Report: Helping MSc Students find suitable topics of study*

Table of Contents

Introduction 1

Design 2

Homepage Search 2

Register page 2

Applying for an idea 3

Bootstrap 3

Overall flow 3

Implementation and testing 4

Netbeans, Derby and JSF 4

Internationalisation 4

Mapping 5

Unit testing & testing the artefact 5

Persistence, Entities and Derby 5

Application Structure 5

Summary 5

Bibliography 6

# Introduction

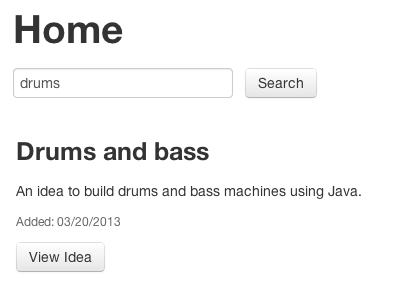
The School of computing currently offers an MSc project ideas database, but it’s implemented in a non-MVC style that does not serve users particularly well. The aim of this project is to create a superior version of the current implementation.

The superior versions main objective is to provide a better experience by offering a search and a simple way to add new ideas. The design will be very simple with clear calls to action to make the user experience positive; this will be achieved by a clean uncluttered interface.

# Design

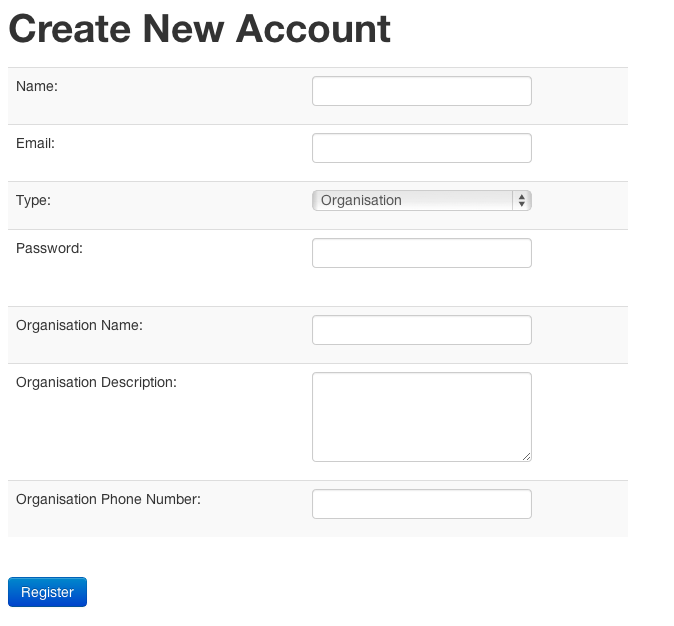
## Homepage Search

The homepage contains a simple search text box that allows users to easily search through the ideas. As this functionality is a key feature, I decided to give it a prominent position on the page. This makes finding ideas fairly straightforward. The search compares the search term against the description and title of the idea via a Java DB SQL query.

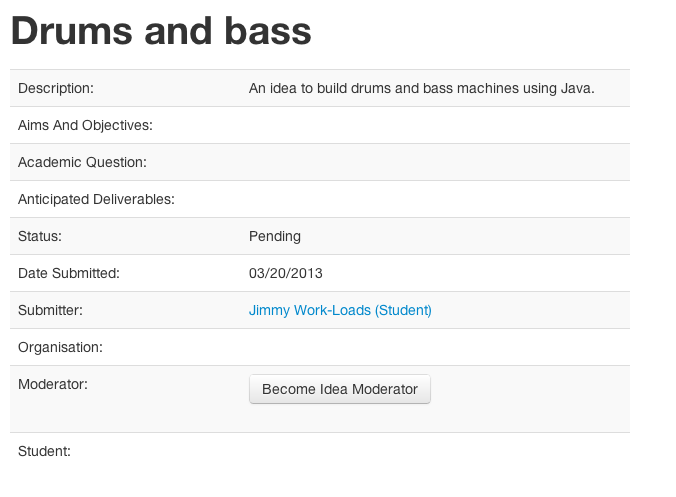
****

## Register page

The page users can obtain an account via is super simple. It has been designed to only request the minimal information required to receive an account. For example, if the user selects an organisation as their user type from the select box, they are presented with more fields that are only related to organisations.



## Applying for an idea

Users should be able to apply to ideas really easily. As a result I made the process of applying for an idea a mere single click of a button. If a user is signed in, they are shown an appropriate “become idea X” as shown in the figure below.   
Once clicked they are associated with that idea.

## Bootstrap

One of the methods to improve the visuals of the design was via Bootstrap, a CSS framework. Bootstrap is a “Sleek, intuitive, and powerful front-end framework for faster and easier web development” (Bootstrap); this allowed me to focus on the business logic of the artefact while not having to worry about a poor UI.

## Overall flow

The main site is split into two main parts, the ideas and the people. The artefact was designed to be as simple to use as possible, so as a result it’s very stripped down to its basic functions. The main two tasks as a user needs to perform are either to submit an idea or apply for an idea. As the diagram below shows, a users flow through the site requires minimal effort on their part.

# Implementation and testing

## Netbeans, Derby and JSF

The main development tools were Netbeans as the IDE, Derby (Java DB) as the database engine and JSF with Java EE as the framework.

Initially the artefact was developed using a MySQL database, but soon it was discovered that this could prove an unreliable solution, as it’s not included by default with Netbeans. Instead Derby (Java DB) was used as it comes with Netbeans. Derby works in a similar way to MySQL, but has slightly fewer features and treats variable types differently. For example, its integers are not suitable as an auto increment primary key as they do not count very high.

NetBeans has a wizard to create a JSF skeleton application from the entities (Netbeans, 2013). This feature allowed for a rapid development of the application so more focus could be placed on the complex business logic, such as creating Derby compatible search functions and login services.

One of the downsides of using JSF 2.0 was its inability to make use of HTML5 input elements such as the search & email inputs. It also doesn’t allow new HTML5 attributes such as placeholder out of the box. While there are ways around this issue they proved tricky to implement, for example one solution required adding a library from Omnifaces via faces-config.xml (Scholtz, 2012).

## Internationalisation

One of the stronger features of JSF is its internationalisation feature. In the artefact I specified a string that could be repeated across multiple elements that were stored in a file called “bundle.properties” (Hlavats, 2009). The settings of the application could then be configured to use a different “bundle.properties” file if the browser requests another language.

## Mapping

A feature I did not make use of in Java EE was the ability to map variables from the data source to their entity. This has the advantage of if the data source changes the variable name (it could be from an external API or a database managed by other person), the artefact would not require a big rewrite. Mapping was not implemented because I controlled the database used in the artefact.

## Unit testing & testing the artefact

One of the features I didn’t utilise in Java EE was its ability to do unit testing, which would have made testing and developing individual methods at a time more efficient then the solution I used. My solution involved coding a feature, pressing run in Netbeans, and then running through a user story. While my solution did allow for rapid development, it made finding the root cause of bugs tricky, especially as I discovered Netbeans & Glassfish required restarting every hour or so due to a lack of RAM available on the development machine I used.

## Persistence, Entities and Derby

The derby database was created by the variables used in the entities classes as defined in the persistence.xml file. Initially I found this a difficult concept to understand but once I understood this I took full advantage of this feature to create new tables when I pressed run in Netbeans.

One of the strong advantages of using entities to manage the data in the application was the @pattern option that allowed the entity to be validated against regular expressions via a single line of code as apposed to creating a new method. This allowed for tidier controllers within my artefact.

## Application Structure

I used the scaffolding application structure as generated by Netbeans in my artefact. This allowed me to focus on building the business logic and not kerfuffle with making sure I have my Facades and data sources the correct way around. Below is a diagram of the setup of my artefact (excluding JSF views):

# Summary

One of the key drawbacks in the development of this project was initially using MySQL to design the database of the artefact. If I had researched more I would have discovered Derby to be a more suitable solution, to start from as it had many small variations from MySQL that caused initial confusion. Alternatively I could have set up an externally hosted MySQL host to act as the database, which would have simulated an enterprise environment, more accurately.

In hindsight it could have been more interesting to use a framework such as Spring or Struts instead of JSF as it had documentation that suggested it had more features that would have allowed for more rapid development.

# Bibliography

Bootstrap. (n.d.). *Bootstrap*. Retrieved 03 20, 2013 from Bootstrap: http://twitter.github.com/bootstrap/

Hlavats, I. (2009, 12). *An Introduction to JSF: Part 2*. Retrieved 03 20, 2013 from packtpub: http://www.packtpub.com/article/introduction-to-jsf-2

Netbeans. (2013, 03 20). *Generating a JavaServer Faces 2.x CRUD Application from a Database*. From Netbeans: http://netbeans.org/kb/docs/web/jsf20-crud.html

Scholtz, B. (2012, 06 15). *Adding HTML5 attributes to standard JSF components*. Retrieved 03 20, 2013 from The BalusC Code: http://balusc.blogspot.co.uk/2012/06/adding-html5-attributes-to-standard-jsf.html