BSc Information Systems and Management – Type Four Project;

A Multiple Choice Online Exam System

Using MySQL, PHP and HTML, with CSS and JavaScript Elements

By Jack Hansard

An online version of this project will be stored at the following address until

31st August 2013

<http://80.68.40.219/boeme/test/main_login.php>

*BSc. Information Systems and Management Project Report, Birkbeck College, University of London, 2013.*

*This report is the result of my own work except where explicitly stated in the text.*

*The report may be freely copied and distributed provided the source is explicitly acknowledged.*

# Contents

[Contents 3](#_Toc355634486)

[Introduction 5](#_Toc355634487)

[1.1 Overview of the Report 5](#_Toc355634488)

[1.2 Development Methodology 5](#_Toc355634489)

[1.3 Software Development Technologies 5](#_Toc355634490)

[1.3.1 Apache 5](#_Toc355634491)

[1.3.3 MySQL 6](#_Toc355634492)

[1.3.2 PHP 6](#_Toc355634493)

[1.3.4 HTML 6](#_Toc355634494)

[1.3.5 JavaScript 6](#_Toc355634495)

[1.3.6 CSS 6](#_Toc355634496)

[Design Analysis 7](#_Toc355634497)

[2.1 Vision 7](#_Toc355634498)

[2.2 Use Cases 7](#_Toc355634499)

[2.2.1 User Stories: 7](#_Toc355634500)

[2.2.2 Use Case Diagram 8](#_Toc355634501)

[2.3 Sequence Diagrams 8](#_Toc355634502)

[2.3.1 Login 8](#_Toc355634503)

[2.3.2 Add New Question 9](#_Toc355634504)

[2.3.3 Delete Existing Question 9](#_Toc355634505)

[2.3.4 Edit Existing Question 10](#_Toc355634506)

[2.3.5 Start Random Exam 10](#_Toc355634507)

[2.3.6 Start Exam on Chosen Topic 11](#_Toc355634508)

[2.3.7 Sit Exam 11](#_Toc355634509)

[2.3.8 View Past Results 12](#_Toc355634510)

[2.4 Communication Diagrams 13](#_Toc355634511)

[2.4.1 The Authentication System 13](#_Toc355634512)

[2.4.2 The Admin Interface 14](#_Toc355634513)

[2.4.3 The Student Interface 14](#_Toc355634514)

[2.5 Database Design 15](#_Toc355634515)

[2.5.1 Entity Relationship Diagram 15](#_Toc355634516)

[Implementation 16](#_Toc355634517)

[3.1 Introduction 16](#_Toc355634518)

[3.2 Database Implementation 16](#_Toc355634519)

[3.2.1 DB\_Connector.php 16](#_Toc355634520)

[3.2.2 PHP Data Objects 17](#_Toc355634521)

[3.2.3 main.php 17](#_Toc355634522)

[3.3 The Authentication System 18](#_Toc355634523)

[3.3.1 main\_login.php 18](#_Toc355634524)

[3.3.2 checklogin.php 19](#_Toc355634525)

[3.3.3 logout.php 19](#_Toc355634526)

[3.3.4 The Safeguard’s 19](#_Toc355634527)

[3.4 The Admin Interface 20](#_Toc355634528)

[3.4.1 adminScreen.php 20](#_Toc355634529)

[3.4.2 addQuestion.php 23](#_Toc355634530)

[3.4.3 updateQuestion.php 23](#_Toc355634531)

[3.4.4 deleteQuestion.php 23](#_Toc355634532)

[3.4.5 Delete/Insert/Update Question Functions 23](#_Toc355634533)

[3.5 The Student Interface 24](#_Toc355634534)

[3.5.1 studentScreen.php 24](#_Toc355634535)

[3.5.2 randomExam.php 24](#_Toc355634536)

[3.5.3 topicExam.php 24](#_Toc355634537)

[3.5.4 examGenerator.php 25](#_Toc355634538)

[3.5.5 exam.php 25](#_Toc355634539)

[3.5.6 answer.php 26](#_Toc355634540)

[3.5.7 results.php 26](#_Toc355634541)

[3.5.8 pastResults.php 26](#_Toc355634542)

[Conclusions 27](#_Toc355634543)

[4.1 Project Summary 27](#_Toc355634544)

[4.2 Recommendations for Future Work 27](#_Toc355634545)

[4.2.1 User Management 27](#_Toc355634546)

[4.2.2 Contextual Drop-Down Menus 27](#_Toc355634547)

[4.2.3 Varying Number of Answers 27](#_Toc355634548)

[4.2.4 Lack of External User Testing 27](#_Toc355634549)

[Bibliography 28](#_Toc355634550)

[Code Listing 29](#_Toc355634551)

# Introduction

Before starting the report, I should mention that the report template I have used is based largely on the example work provided by Birkbeck, specifically the work done by (Amankwah, 2004). Due to the similarity of our project goals I decided utilising the template provided would allow for a clearer framework. I would add that it is only the report style being emulated; any similarities in the coding are purely coincidental.

## 1.1 Overview of the Report

This report covers the analysis, design and implementation of a web-based multiple choice exam system. The report is divided into three distinct sections; design, construction and evaluation. Each of these sections will focus on a different aspect of the project.

The Design section will largely consist of UML diagrams and other design tools such as user stories. I will start with the requirements capture before moving on to demonstrate how I turned the user stories and such into a theoretical model for the exam system.

Following on from this third section I look at how I chose to implement the requirements online. Whilst I will include a paper version of the full code in at the end of the document, I will limit the code within the report itself to highlighting key elements and explaining their purpose.

Finally, I have summarised my thoughts in regard to the projects success and how I think its functionality could be expanded in the future.

## 1.2 Development Methodology

Due to the reasonably small scale of this project, I felt that the use of a formal software development methodology would actually be more of a detriment than a benefit. However I did use a number of software development tools I came across in the last four years, the primary ones being; User Stories, Use Cases, Domain Models and Extended Entity Relationship Diagrams.

The project itself was broken into three distinct stages. A requirements capture phase, in which I aimed to establish exactly what the project aimed to achieve. A planning phase, during which I created the aforementioned artefacts in an attempt to minimize the headaches later during the programming process. Finally the actual putting-together of the website and database system.

Whilst each of these phases had a differing focus, the website approach leant itself to a modular way of thinking due to its nature. As such I was able to narrow my focus down, enabling me to figure out one or two aspects at a time. This in turn meant that while there were no formal iterations such as those found in the United Software Development Process, the development was still iterative in practice.

## 1.3 Software Development Technologies

There are a vast number of different approaches available to a web developer for a project of this type. What I chose to do in the end was to use a variation on the common open-source LAMP stack (Linux-Apache-MySQL-PHP). The difference was that the Apache server runs on a windows machine. Irrespective of this, using a well-known framework meant that any compatibility issues were minimised.

### 1.3.1 Apache

Apache is an open source HTTP server, which according to their website is *“the most popular server on the Internet”* (The Apache Software Foundation). Beyond the compatibility with the LAMP stack, the overriding factors behind my decision was, nothing I would be doing was complicated and the software was freely available.

### 1.3.2 MySQL

Similarly, MySQL according to their website is *“the world’s most popular open source database”* (Oracle Corporation). As with Apache, compatibility and the common availability of the software were the overriding factors in making my decision.

### 1.3.3 PHP

According to their homepage, PHP is *“a widely-used general-purpose scripting language that is especially suited for web development”* (PHP). However although also freely available and compatible these were not the overriding factors for my decision. In my initial proposal, I suggested that I would be using Java, another ‘C-family’ language. I decided that while the project could work using Java, I would instead take the opportunity to learn PHP and LAMP as it provided a more interesting challenge.

### 1.3.4 HTML

HTML stands for Hypertext Mark-up Language and is the *“language that browsers read to render web pages”* (Kyrnin). HTML was not really a choice but a requirement. In addition, any technologies I used had to be compatible with HTML or accessible through the PHP middleware.

### 1.3.5 JavaScript

JavaScript is a client-side scripting language, which, at the time of writing runs in all major web-browsers. Much like Cascading Style Sheets (CSS) its primary function is that of support. In this project, I have used JavaScript to update HTML forms after page load depending on user actions, a typical usage.

### 1.3.6 CSS

Cascading Style Sheets define how the website looks in the browser. HTML defines tags such as <h1> (Heading 1) or <p> (Paragraph) and the CSS document tells the browser how to style the corresponding text. CSS as a standalone document has no function.

# Design Analysis

Within the design analysis section I have made constant use of UML, as described in the book ‘Object-Orientated System Analysis and Design Using UML’ (Bennett, McRobb, & Farmer, 2006). I have therefore assumed a base understanding of the concepts and restricted myself to a brief summary of the UML methods as they are used.

## 2.1 Vision

In the initial project proposal, the main requirements were for two different types of users (Students/Admins), with different options available for each of them. Student requirements were for a multiple choice exam interface, which presented questions from the database. The student should have three main options; an exam consisting of random questions from random topics, an exam consisting of random questions from a specific topic, and, finally, the ability to view their results from past exams.

The Admin interface’s main job was to add new questions and/or edit/delete existing ones. Initially I was planning for different students to be registered to different topics, but this seemed superfluous in the end, therefore the requirement was dropped. Alongside the ability to work with the client-server architecture found online, the above requirements outlined the initial model for this system.

## 2.2 Use Cases

*“A Use Case describes, from a user’s perspective, of behaviourally related set of transactions that are normally performed together to produce some value for the user.”* (Bennett, McRobb, & Farmer, 2006)

I will be using Use Cases as part of this project to formalise my initial requirements.

### **2.2.1 User Stories:**

#### “User stories are an agile requirements approach that helps shift the focus from writing about requirements to talking about them” (Mountain Goat Software)

#### All:

Login User Story: As a user, I want to input a username and password, hit login and then automatically be directed to the correct screen depending on my user type

#### Admin Only:

Add New Question to the System: As an administrator, I want to be able to add a new question to the question bank by inputting all the relevant values and hitting submit question.

Delete Existing Question: As an administrator, I want to be able to give the system a question reference to delete it.

Edit Existing Question: As an administrator, I want to be able to edit a question directly by changing the values and hitting save.

#### Student Only:

Random Exam: As a student, I want to be able to take an exam made up from random questions from random topics

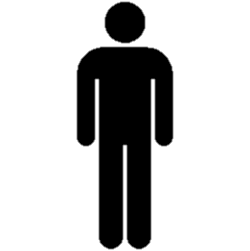
Exam on Topic ‘X’: As a student, I want to be able to take an exam made up from random questions from one chosen topic.

View Past Results: as a student, I want to be able to view my past results, sorted by topic so I know where to focus.

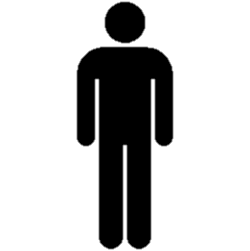
### 2.2.2 Use Case Diagram

Below you can see the use case diagram for the user stories outlined previously:

Admin



Student



## 2.3 Sequence Diagrams

*“Sequence diagrams show an interaction between objects arranged in a time sequence. Sequence diagrams can be drawn at different levels of detail and also to meet different purposes at several stages in the development life cycle.”* (Bennett, McRobb, & Farmer, 2006)

### 2.3.1 Login

checklogin(userID, password)

PHPquery

**Object2**

CheckLogin

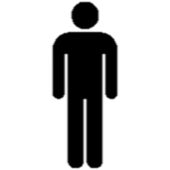
login(userID, password)

Boolean

**User**

checklogin(userID, password)

Boolean



**Object1**

LoginGUI

**Object3**

Database

checklogin(userID, password)

queryResults

login(userID, password)

Boolean

### 2.3.2 Add New Question

addQuestion

PHPquery

**Object2**

AddQuestion

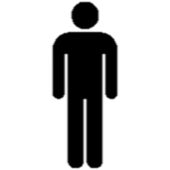
NewQuestionDetails

HTML Form

**User**

addQuestion

PHPQuery



**Object1**

AdminGUI

**Object3**

Database

addQuestion

queryResults

questionAdded

 Boolean

### 2.3.3 Delete Existing Question

deleteQuestion

PHPquery

**Object2**

AddQuestion

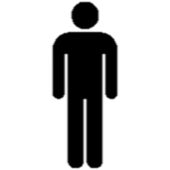
getQuestion(TopicID, QNo)

HTML Form

**User**

deleteQuestion

PHPQuery



**Object1**

AdminGUI

**Object3**

Database

deleteQuestion

queryResults

questionDeleted

 Boolean

### 2.3.4 Edit Existing Question

UpdatedQuestionDetails

 queryResults

**Object3**

EditQuestion

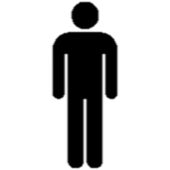
getQuestion(TopicID, QNo)

HTML Form

**User**

getQuestion(TopicID, QNo)

PHPquery



**Object1**

AdminGUI

QuestionDetails

 Query

**Object2**

Database

EditedQuestionDetails()

HMTL Form

NewQuestionDetails

HTML Form Data

QuestionUpdated

 Boolean

editQuestion

PHPquery

### 2.3.5 Start Random Exam

**Object2**

CreateExam

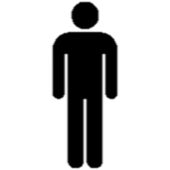
chooseExamLength(N)

Integer

**User**

examLength(N)

Integer



**Object1**

RandomExamGUI

createExam

 PHPquery

**Object3**

Database

startExam

HTML Form Data

returnExam

queryResults

### 2.3.6 Start Exam on Chosen Topic

**Object2**

CreateExam

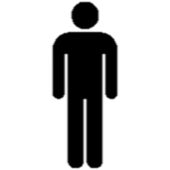
setExamLengthTopic(N, Topic)

Integer, Integer

**User**

examLengthTopic(N, Topic)

Integer



**Object1**

SpecificExamGUI

createExam

 PHPquery

**Object3**

Database

startExam(data)

examData

returnExam

queryResults

### 2.3.7 Sit Exam

**Object2**

Answer

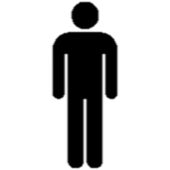
StartExam(data)

examData

**User**

checkAnswer

HTML Form



**Object1**

ExamGUI

**Object3**

Results

DisplayQuestion

queryResults

chooseAnswer

HMTL Form Data

Next Question

HTML Form

finishExam

HMTL Form Data

checkScore

HMTL Form Data

displayScore

HMTL Form Data

writeScoreToDB

PHPQuery

### 2.3.8 View Past Results

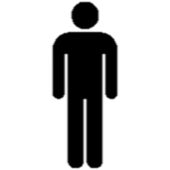
getDetails(userID)

PHPquery

**Object2**

CheckLogin

**User**



**Object1**

LoginGUI

examResults

queryResults

DisplayResults

HTML Table

## 2.4 Communication Diagrams

*“Communication diagrams show an interaction between lifelines (e.g. objects) and the context of the interaction in terms of the links between lifelines.”* (Bennett, McRobb, & Farmer, 2006)

Having covered all of the specified interactions the system is required to make, I will now explain with the aid of communication diagrams how the different pages are going to interact. Due to the complexity of the system, I am going to do this using three diagrams; the first diagram will display the authentication system, the second will show the admin interface and the final diagram will be used for the student interface.

### 2.4.1 The Authentication System

main\_login

<HTML>

checkLogin

<PHP>

Database

<MySQL>

adminScreen

<HTML>

studentScreen

<HTML>

1:POST(Username,Password)

1.1 Query(Username,Password)

1.2 Query(Results)

1.3 Usertype = Student

1.3 Usertype = Admin

1.3 redirect(results.isEmpty)

2:LoggedIn=False

2:LoggedIn=False

3: Usertype = Student

* **Event ID 1** is the login, 1.3 is checkLogin redirecting the user back to the main\_login page if the username/password query returns empty such as when the username / password is missing or an incorrect value is entered
* **Event ID 2** is if a user manually loads the admin/student screen page (by typing in the URL) without logging in first
* **Event ID 3** is a redirect when a student attempts to access the admin page(s)
* **Event ID 2** and **3** will occur on load for every page in the site except main\_login

### 2.4.2 The Admin Interface

adminScreen

<HTML>

Database

<MySQL>

deleteQuestion

<PHP>

1:getTopicList()

2:viewQuestion(Topic, Question)

1.1:returnTopicList(Results)

2.1:returnQuestion(Results)

addQuestion

<PHP>

updateQuestion

<PHP>

3:addQuestion(FormData)

4:deleteQuestion(FormData)

5:updateQuestion(FormData)

3.4:passSuccess/FailureMessage(String)

4.4:passSuccess/FailureMessage(String)

5.4:passSuccess/FailureMessage(String)

3.1:insertQuestion(FormData)

3.2:getQuestion(FormData)

3.3:returnQuestion(Results)

4.1:deleteQuestion(FormData)

4.2:getQuestion(FormData)

4.3:returnQuestion(Results)

5.1:updateQuestion(FormData)

5.2:updateQuestion(FormData)

5.3:returnQuestion(Results)

**Event ID 1** is a dynamic drop down menu, **Event ID’s 2, 3, 4** and **5** are the View, Add, Delete and Update functions.

### 2.4.3 The Student Interface

studentScreen

<HTML>

1.1:getPastResults(userName)

1:goTo(pastResults)

topicExam

<HTML>

pastResults

<HTML>

Database

<MySQL>

2.2:returnPastResults(Results)

randomExam

<HTML>

exam

<HTML>

answer

<PHP>

results

<HTML>

examGenerator

<PHP>

2:goTo(randomExam)

3:goTo(TopicExam)

3.1:getTopicNames()

3.2:returnTopicNames(Results)

2.1:generateExam(formData)

3.3:generateExam(formData)

4:startExam(examData)

2.2:randomExamGen(noQ)

3.4:topicExamGen(noQ, Topic)

2.3:returnExam(Exam)

3.5:returnExam(Exam)

4.1:answerData(formData)

4.2:nextQ

(examData)

4.4:results (examData)

4.3:results (examData)

**Event ID 4 .1** and **4.2** are the iterative cycle, question number is checked on exam submit to see if the end of the exam has been reached. If it has **4.3** is called, otherwise **4.1** and **4.2** repeat. **4.4** writes the exam result to the userID in the database.

## 2.5 Database Design

So far, almost all of the design has focussed on the front end. I am now going to take a moment to explain the database architecture. As mentioned previously I am going to be using four tables; Logins, Student Data, Topics and Questions. Logins and Topics will both be stand-alone tables, whilst Questions and Student Data will both use foreign keys as part of their implementation.

The Logins table will consist of three variables; Username (Primary Key), Password and User Type. The first two are used to login to the system whilst the third is a control variable used to determine administrators (they will have user type ‘1’).

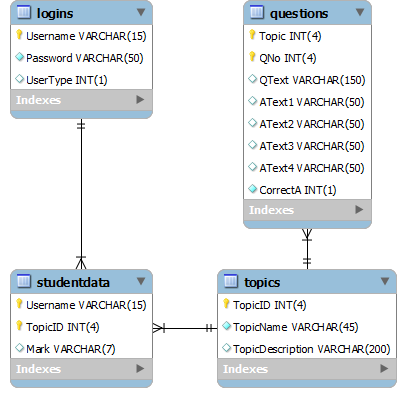
The Student Data table will consist of Username (equal to logins.Username), TopicID (equal to topics.TopicID), and a Mark variable. The first two are used as a combined primary key, whilst Mark + TopicID will be used to record each user’s most recent result after taking an exam in that Topic.

The Topics table consists of TopicID (Primary Key), TopicName and TopicDescription. TopicID is used as the key to allow for two topics with the same name but different purposes. Whilst this has not been setup to occur in the current incarnation of the database, it seemed like a reasonable possibility in the future. Current TopicDescription is unused but could be added into the specific topic exam to provide more information on a topic before the exam.

Finally the Questions table is formed from 8 variables; Topic (equal to topics.TopicID), QNo, QText, AText1-4 and CorrectA. Topic and QNo combine to form the primary key in the database and the Question Reference on the website. QText and AText1-4 form the question and CorrectA denotes which one of answers 1-4 is correct.

### 2.5.1 Entity Relationship Diagram

*“An entity-relationship (ER) diagram is a specialized graphic that illustrates the relationships between entities in a database.”* (Chapple)



# Implementation

## 3.1 Introduction

Having established a theoretical framework around which to build the site, I am now going to explain how I actually coded the system. In order to help structure the section I am going to explore each system separately and follow each user through their journey in the system, including the back-end systems, which have no UI elements.

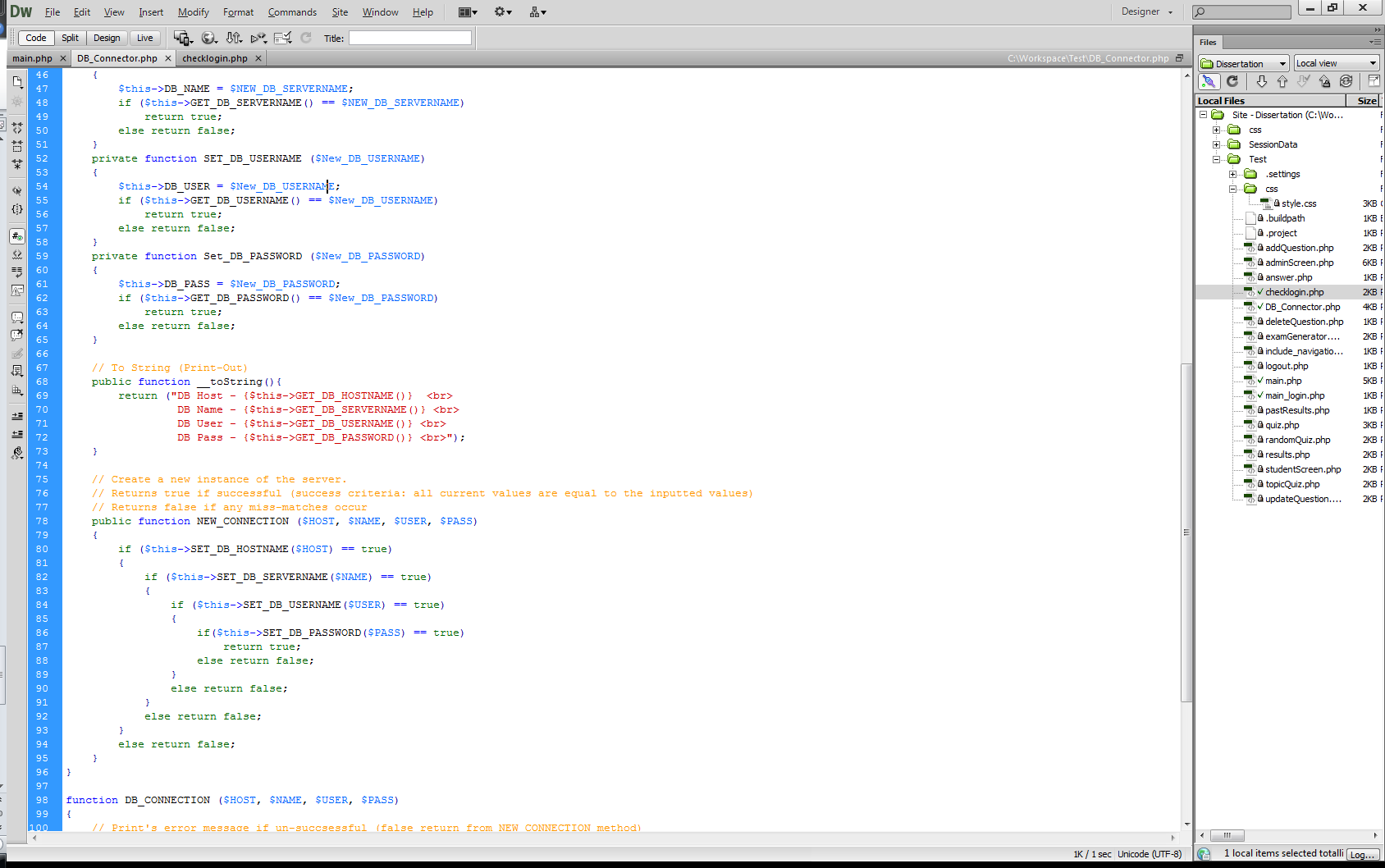
## 3.2 Database Implementation

In this part, I am going to explore the method I use to connect the MySQL database, and some of the query functions. I will not be exploring the actual MySQL database, as it has been implemented identically to the schema on the previous page.

### 3.2.1 DB\_Connector.php

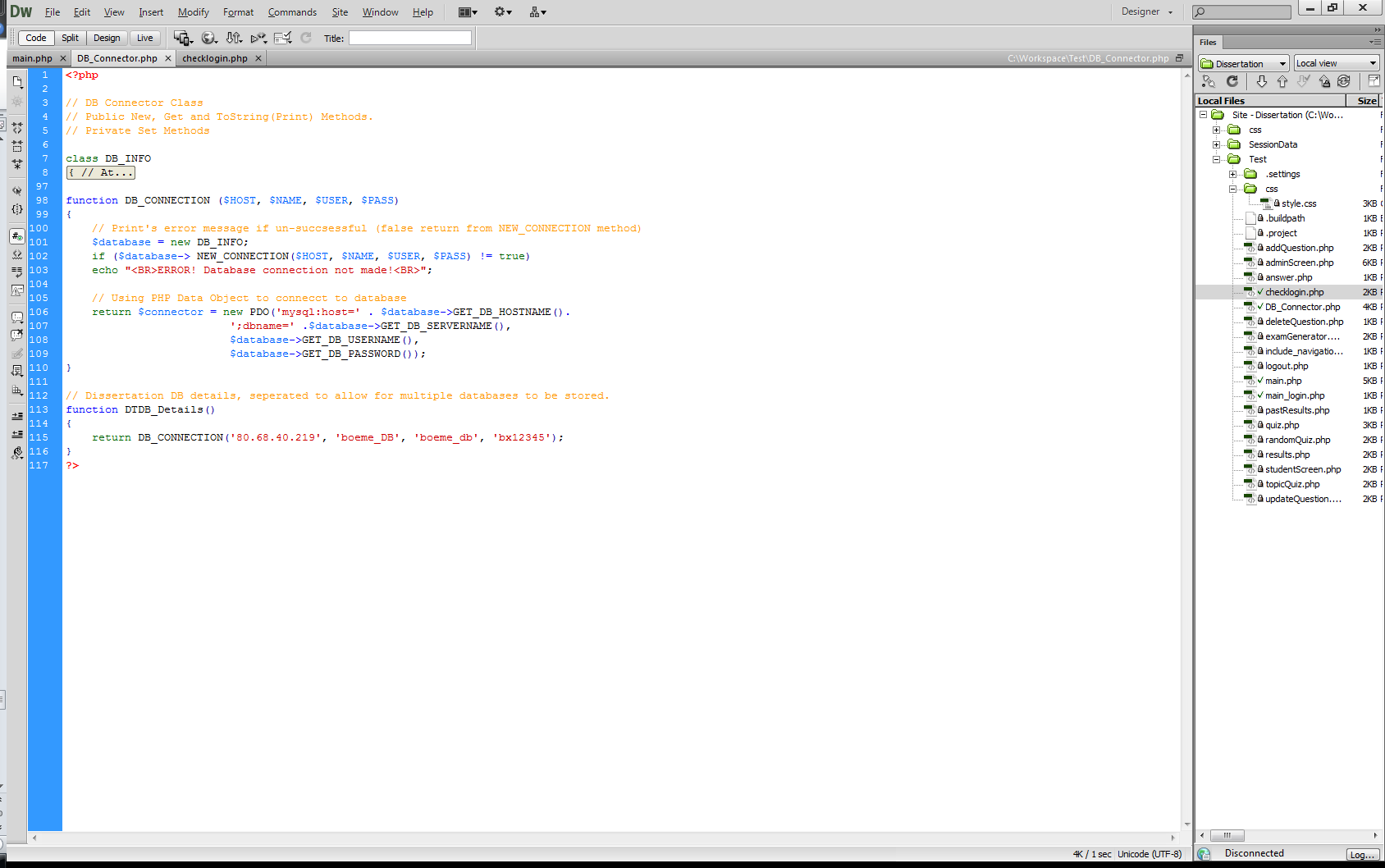
This page is included via “include ‘DB\_Connector.php’;” on every page which needs to connect to the database. I chose not to include it alongside the rest of my generic methods, as there are several pages that make use of functions in main.php that do not need to connect to the database.

DB\_Connector is made-up of one class and two functions. The class is reasonably simple, it has public get and private set functions, in other words; info about the contents is freely available, but only the class itself can edit the details. I have also defined a \_\_toString() function which allows you to echo the database object.



Finally, the DB\_INFO class has a new connection function. This takes an input of four values $HOST, $NAME, $USER, $PASS and, using the private set methods, sets the values of $DB\_HOST, $DB\_NAME, $DB\_USER and $DB\_PASS to the user values. On its own, this class does nothing. However, when called by DB\_CONNECTION…

DB\_CONNECTION is what actually connects to the database. As you can see below, the function is quite simple in execution; it starts by creating a new instance of DB\_INFO. It then sets the database info to the user’s values, returning false and echoing an error to the screen (this should never actually happen). Finally, it creates a new PHP data object called connector, sets it to the values and returns it to whichever page called the function.

****

Finally, we have the DTDB\_Details() function. This function is what every page that wishes to connect to the database actually calls. Inside it has the saved details for whichever database you wish to connect to. In other words $HOST, $NAME, $USER, $PASS. Saving them here means that I only have to change one line of code to connect to a different MySQL database, and only two lines if I wish to connect to a non-MySQL one.

### 3.2.2 PHP Data Objects

There are a number of benefits to using PHP data objects (PDO’s) in this system. Firstly, as mentioned, I only have to change one line of code to change how I connect to the database (source: (.Net Magazine)):

$conn = **new** PDO(‘mysql:host=localhost;dbname=yourdbname’, ’username’, ’password’);

vs.

$conn = **new** PDO(‘pgsql:host=localhost;dbname=yourdbname’, ’username’, ’password’);

This is useful when you consider that the system may not always stay on the current MySQL database and minimising the work needed to change systems is a plus. However, one of the biggest reasons is simply one of security. To send a query using a PDO you have run it through $connector->prepare(‘SQL Query’). This will prevent first order SQL injection (bad user data).

### 3.2.3 main.php

This page is where I have stored (almost) all of the SQL queries. Like DB\_Connector it is inherited when needed via the PHP include function.

function Search ($connector, $select, $from, $where)

This function is called when I need a standard select query for MySQL. In this system, it is usually called by the other functions in the main.php page as they iterate through assigned values to build a table. The only real point of note is that if the Select From Where query returns empty, the function returns -999 instead of a table.

function RQRTGen($noQuestions, $connector)

This function forms one of the core systems for the website. RQRTGen stands for Random Questions, Random Topics Exam Generator. There is a lot of code for a reasonably simple function here. First, it creates an array called $previousResults, and sets the value to zero. It then prepares a query to select all the data from the question table, and executes it. This is used to get a current row-count of the table in order to check that the requested exam length does not exceed the number of questions available. If this is the case then $noQuestions is set to be equal to the total number of questions available. The $count variable is then reset to zero and a list of all available topics ($tList) is created.

Having performed the safety check, it then proceeds to enter a while loop. $count is checked against the requested $noQuestions and the loop iterates until $count is equal to $noQuestions, at which point it breaks. Upon entering the while loop a Boolean operator, $check, is created and set to false.

An inner while loop is the entered and $tList is shuffled and the TopicID of the first topic in the list is then selected. After this a the question number is selected in much the same way, the only difference being that only questions for the previously chosen topic are listed as available options. Finally the $check variable is invoked, calling the QDone function (a simple function that check if the string ($tNo.”, “.$qNo) exists in the $prevResults array, if it does exist it returns true, if not false) to determine its value.

At this point the inner while loop iterates until it finds a question which has not been used already. After this the Search function is called with the recently defined variables $tNo and $qNo as its Where clause. The $count variable is incremented and the process repeats until the outer while loop breaks. After this $results is returned.

function topicList($noQuestions, $connector) & function questionList($noQuestions, $connector)

I have bundled these two together as they both almost identical functions. As you saw in the previous example they are both used to return arrays filled with the information about either a) all available topics or b) all questions for a specific topic. The only particular point of note is the PDO::FETCH\_ASSOC command which is used to fill $results[] with all the information for each $query. This is used because PDO’s don’t react the same way as normal arrays and thus there is a need to process them back into standard two-dimensional PHP arrays.

function RQFTGen($noQuestions, $topic, $connector)

Having explained the previous exam generator in detail I am simply going to explain the differences that this function has. Like RQRTGen before it, RQFTGen is an abbreviation, this time for Random Question Fixed Topic Exam Generator. The only real difference here is that the user has specified the input value for $topic (previously $tNo). As the topic is set beforehand this results in several differences. Firstly, the select statement used to establish the ceiling for $noQuestions uses Topic = ‘$topicID’ as a limiting factor. Secondly, the $qList can be generated outside of the while loop to reduce the number of database calls, and finally the $tList variable does not need to be generated at all as we already know which TopicID we will be querying the database with.

The remaining code in main.php is used either in the admin interface or to display the table data for the getResults page of the student interface. For that reason, rather than explaining it all now, I will explain each in context, as I come across them.

## 3.3 The Authentication System

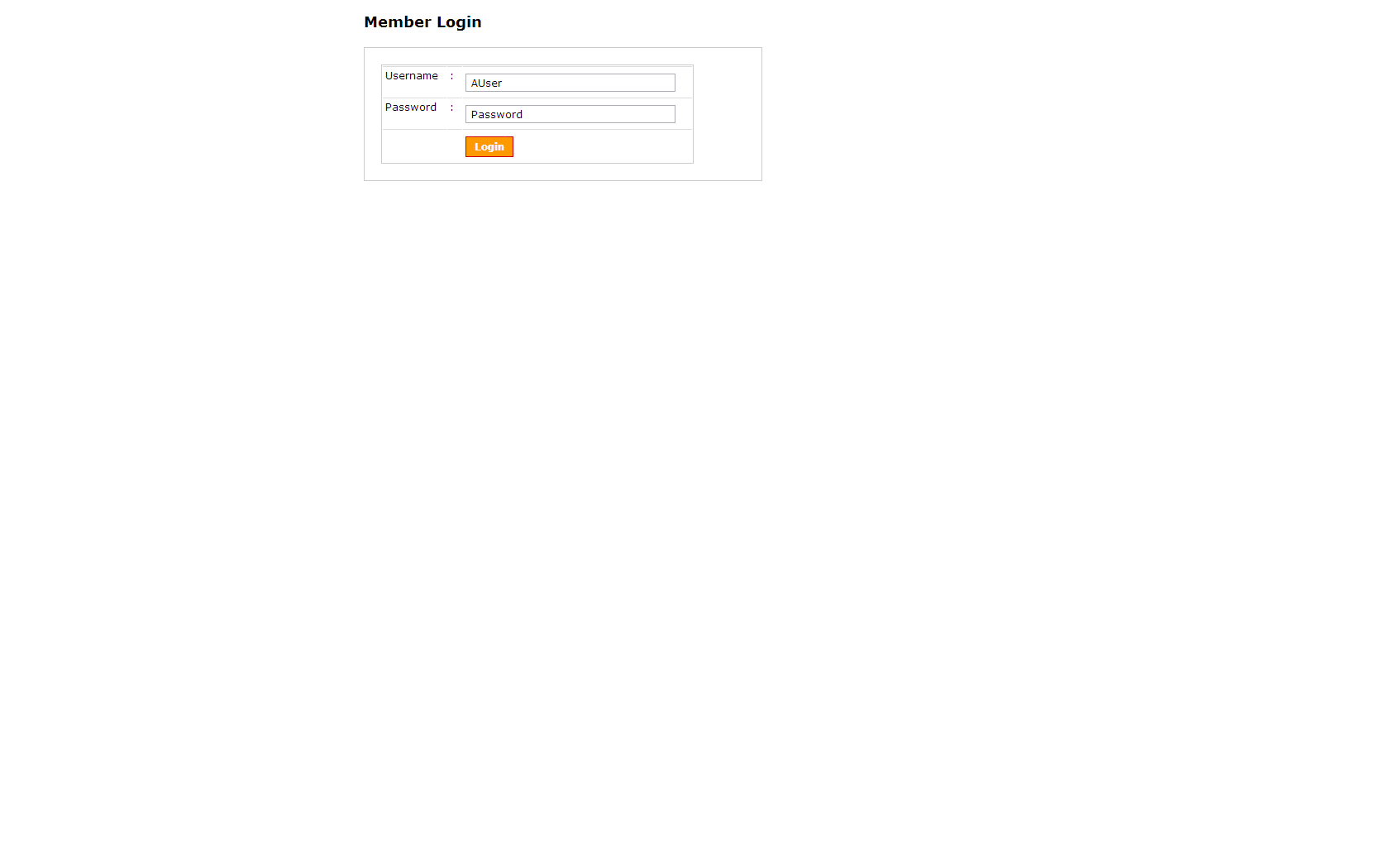
The authentication system can be explained in a two simple concepts:

1. To access the site you have to log in
2. Only admins can access the admin interface

That’s pretty much it, the rest is just coding and session data.

### 3.3.1 main\_login.php

The site gateway is actually just a table contained in a form which posts the inputted username and password to the validator (checklogin.php). As such, there isn’t really anything to say about it other than “Look! Pretty CSS!”



On a more serious note, the only purpose of this page is to act as a gateway to the rest of the site. Every other page in the site has a line of code at the top which redirects the browser to main\_login.php if the username session variable is not set.

### 3.3.2 checklogin.php

The ‘*real’* login page. After starting the session (important for storing variables) and getting the code for the included ‘DB\_Connector.php’, checklogin.php initialises two PHP variables $myusername and $mypassword, setting them to the values posted by the previous page (‘myusername’ & ‘mypassword’). It then encrypts the password using the md5($string) command. This encryption means that I don’t ever need to store the password as plain text, increasing security (somewhat).

Having done this it gets ready to connect to the database ($connector = DTDB\_Details();) and runs the select from logins query, using $myusername and $myencryptedpassword as control variables. Having (hopefully) returned a value checklogin.php proceeds to assign the user a User Type (either zero or one depending if they are an admin or student).

Finally, it counts the rows table rows. If the result is equal to one, which for any valid login it will be, it proceeds to set the Session Variables up and redirects them to the sites other pages. First it sets the $\_SESSION[‘name’] to $myusername and then it sets the $\_SESSION[‘usertype’] to $myusertype.

Now, depending on the user’s user type it will send them to one of two pages. If the $\_SESSION[‘usertype’] is equal to one, i.e. the user is an admin, then it will send them to the admin screen. If the user’s type is anything else, then it will send them to the main student screen.

Finally, if the number of results returned by the earlier SQL query did not equal one, then the user is redirected back to the main\_login screen.

### 3.3.3 logout.php

This is a very simple page, all it does is get the session data, and then delete the session and send you back to main\_login.php. As this is accessed via the link menu at the side of the page (I will discuss this later), from the user’s perspective they hit logout and are the immediately sent to main\_login.php.

### 3.3.4 The Safeguard’s

These are actually very simple in practice. As you just saw, when a user logs in two session variables are set; User Type and Name. Going back to my two initial concepts (must be logged in / only admins can be admins), I can now explain them better

For the first, we just run the following code on page load:

if(!isset($\_SESSION[‘name’]))

header(“location:main\_login.php”);

For the second, we just do this straight after on any pages which only admins should be able to see.

if($\_SESSION[‘usertype’] == 0)

header(“location:studentScreen.php”);

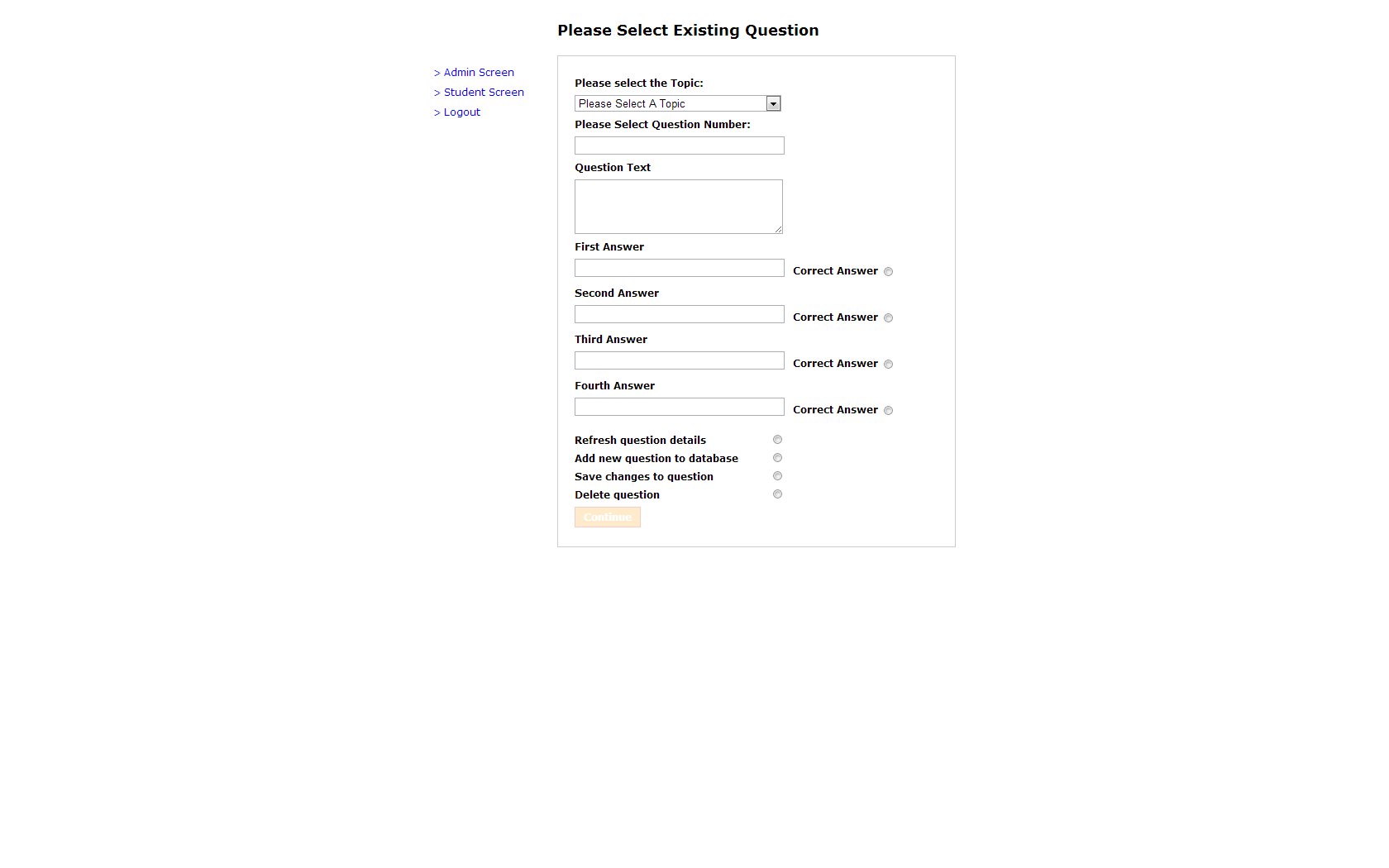
All each of these does is redirect the user to an appropriate page if they shouldn’t be there. As this happens before any of the rest of the script is executed that is all that is needed.

## 3.4 The Admin Interface

Okay, the admin interface falls into four main parts: the front page, adding a new question, editing a question and deleting a question. For the purpose of this report, and to put the code in context, I am going to walk you through each of them in turn.

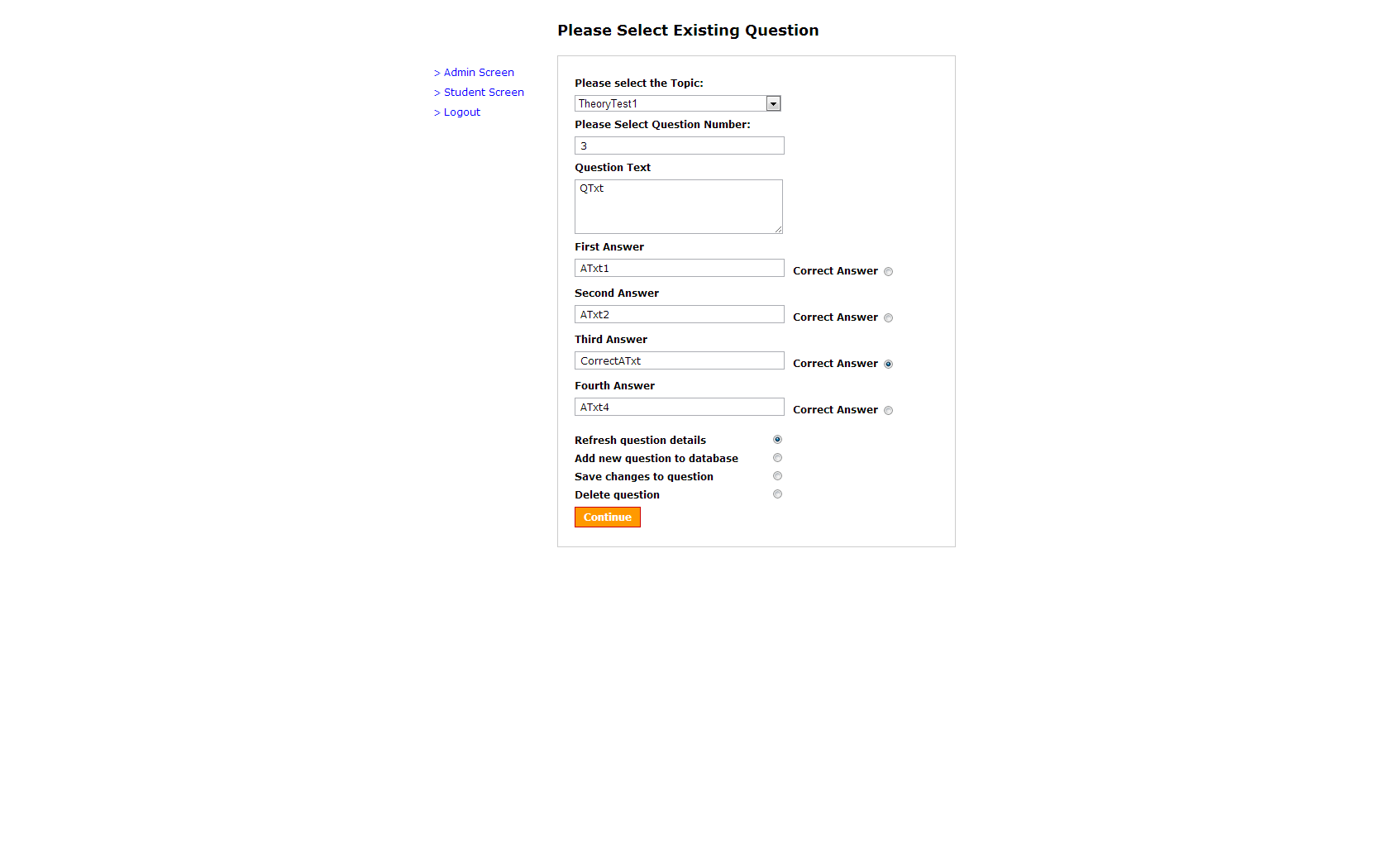
### 3.4.1 adminScreen.php

When you login as an administrator, you are presented with the following screen:

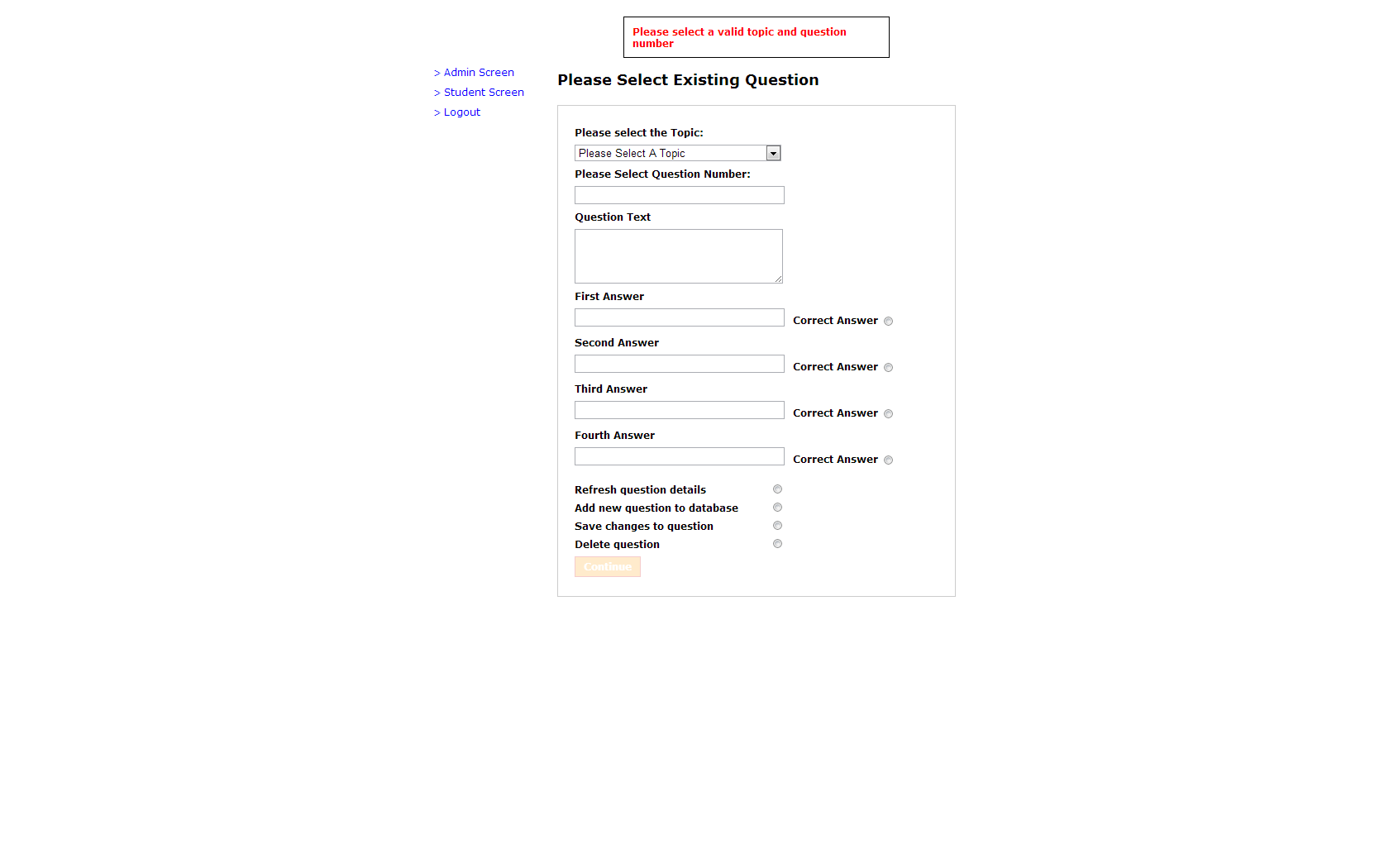


Beyond the creation of the form (which I will get to in a moment), this page is actually quite simple. On the left you have an included navigation interface which is just some hyperlinks stored in a common file to save typing them out every time. The only vaguely complicated part is that they are stored in an ‘if’ statement, used to distinguish between user type zero and one (student and admin), in order to provide them with different links.

The only other part of this page is not actually immediately obvious. Unlike the rest of the options, the refresh question details button actions codes which is all contained within this page (the others link to pages such as addQuestion.php). The way this works is as follows: As an admin you choose a topic from the drop down list and then type in the question number. You then select the refresh question details radio button, which in turn unlocks the continue button. You hit that and the question info is magically populated into the box via the powers of PHP.



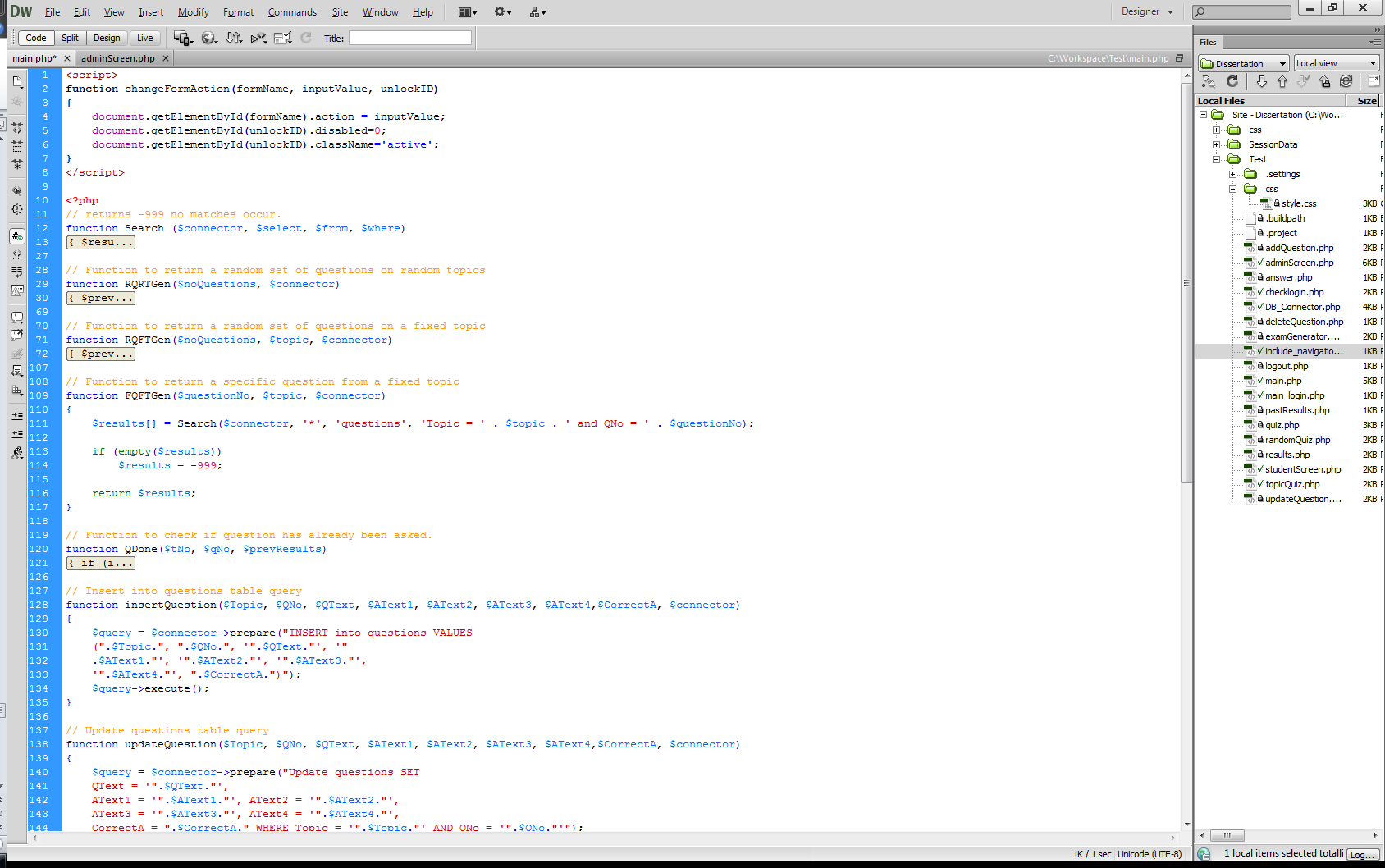
As final user interface element, when you type in a question number that returns blank, you get an error message pop up at the top of the screen:



This code is completely self-contained, as such, it would be very easy to either remove or change into another form such as a pop-up window. Personally, I find such things annoying, and thus prefer the option of a header at the top of the page.

I am not going to explain the actual concept of how form data works in HTML as that is beyond the purview of this document. I will instead examine any parts that are not standard to HTML.

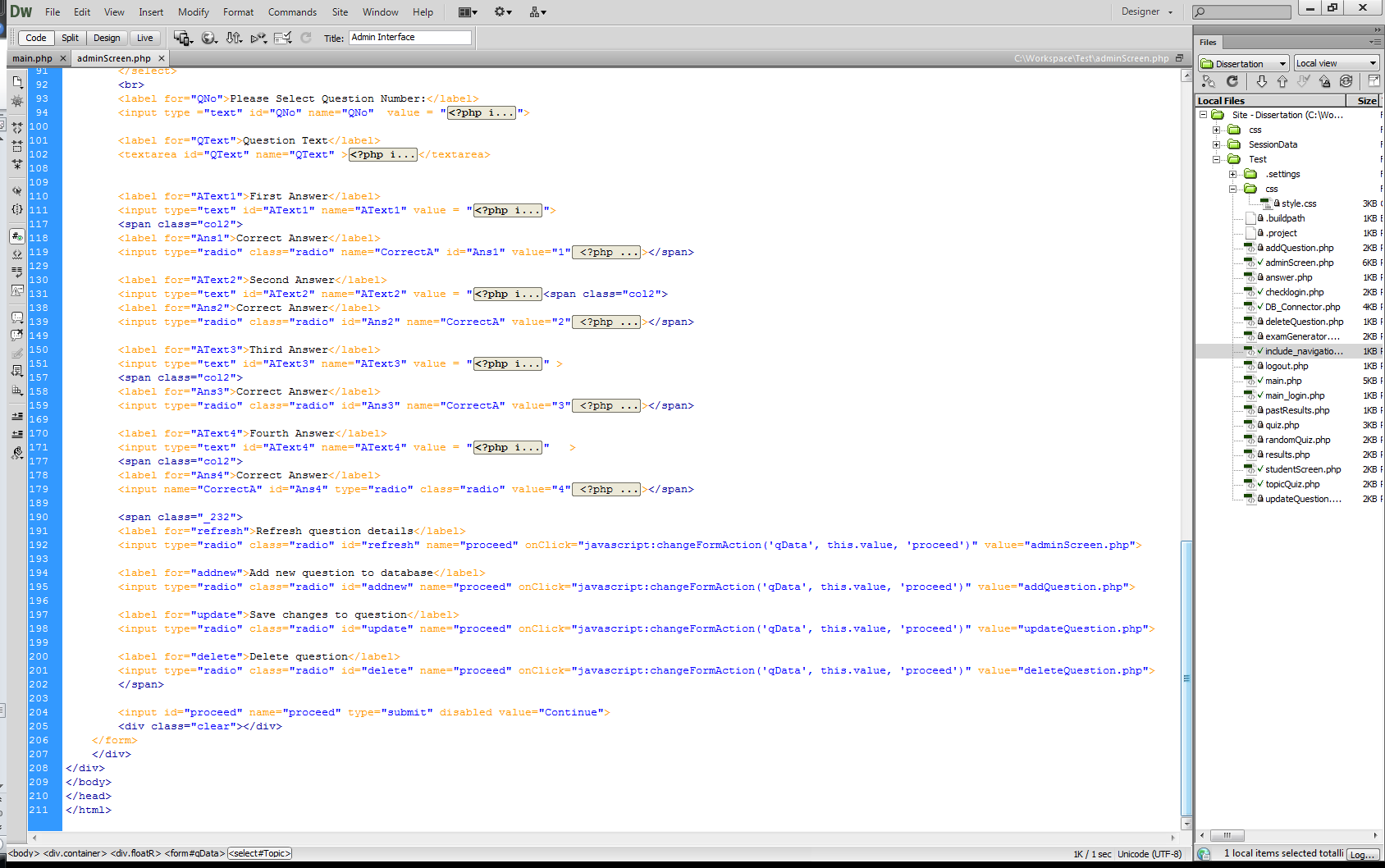
The first thing to note is that by default the form action is ““, in other words it is blank on page load. This would normally mean that when you hit submit/continue at the bottom of the page, nothing would happen. This is where another one of my functions in main.php comes in:



The first thing to note here is that this is a JavaScript function, not a PHP one. Without getting into the differences between client side and server side code, suffice to say that while you can call server-side code as the page is loading, it becomes much more difficult to do after it is finished.

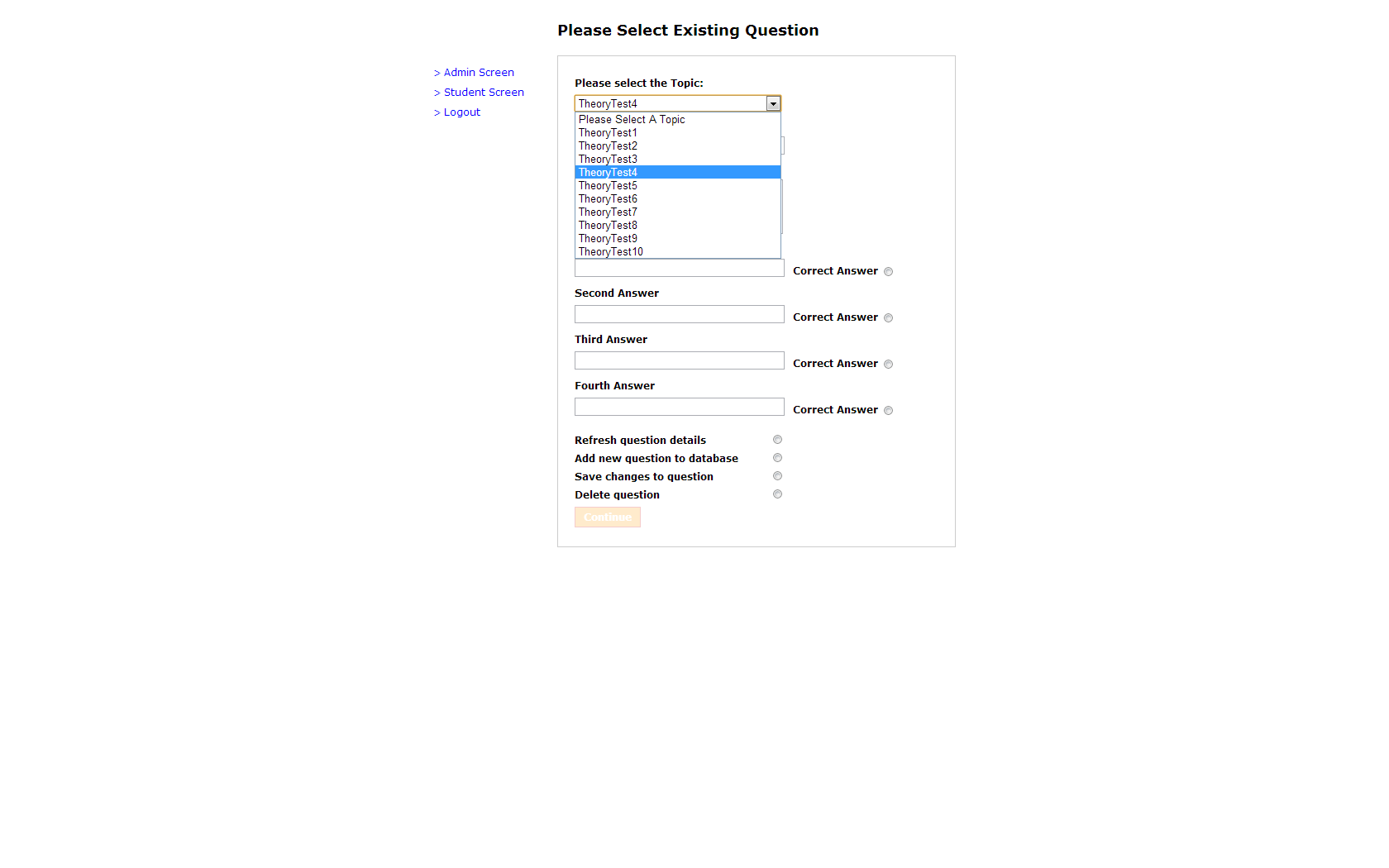
changeFormAction is actually a reasonably simple piece of code once you understand what is happening. It requires three input values to work correctly; the name of the form it is editing the form action of, the value it is changing it to and the unique ID of the button it is setting to active.

If we move on from this function to look at the code for the four radio buttons we will see the following:



Ignoring the <span class=“\_232”> heading, this is just CSS formatting, we can see that each of them has the property onClick=“javascript:changeFormAction(‘qData’, this.value, ‘proceed’)”. Alongside this, they are all given a different ‘value’, which the JavaScript command sends to the changeFormAction function. In this way, clicking on each will update the value of the form to match the value for the radio button.

Moving back up the page to the drop down menu for topic, you will see that clicking on reveals the following



This drop down menu was created on page load by running the topicList function discussed earlier and assigning it to $tList. $tList was then used in an embedded PHP function to echo out the relevant statements using a foreach statement (foreach is a PHP iterator function which will automatically go through every row in an array.)

The rest of the page is reasonably simple with the exception that if a form is value has been posted previously, such as when you choose the refresh question details option, it will automatically fill in the value and thus display the selected question.

The final part of this page is the actual select question code. This function is actioned as follows:

1. A topic and question number is selected, refresh question details is selected and the continue button is clicked
2. On page load PHP checks to see if ‘Topic’ and ‘QNo’ have been posted
3. If they have been posted then $Topic is set to $\_POST[‘Topic’] and $QNo is set to the integer value of $\_POST[‘QNo’]
4. If $Topic and $QNo are both set then $questionList is set to the value of FQFTGen($QNo, $Topic, $connector)
5. FQFTGen gets the question with QNo = $QNo and TopicID = $Topic
6. If $questionlist[0] returns with a value of -999 then a variable $msg is created, populated with the error message and the page is reloaded again with the an error shown at the top, set to the value of $msg.
7. If the array is populated then a foreach statement is run which sets the values of each of the assigned variables to the corresponding values in the $questionList array
8. Finally, the HTML form loads, checks to see if the variables are set, and if they are loads them up as the assigned value.

function FQFTGen($noQuestions, $topic, $connector)

All of the admin pages call this function; this is actually just a convenience function as technically this function could be covered by the earlier search function. The difference here is that a lot of the detail has been hard-coded in. This in turn means that the method calls on the other pages are easier to read, and also helps to separate the search function from the rest of the interface.

### 3.4.2 addQuestion.php

The addQuestion.php page is actually reasonable simple. It starts by getting a database connector ready using the DTDB\_Details method, it then checks to see if all of the required variables have been sent over correctly using the PHP isset method (a simple method that returns true if $variable is != null). Assuming that everything is set then it assigns PHP variables the corresponding HTML variable values (i.e. $Topic = $\_POST[‘Topic’];), if any were unset, then it creates a $msg variable and assigns it an error message.

Having assigned all the variables it now (courtesy of main.php) performs a SQL insert call, of which I will discuss the details in a moment. Finally, it creates the $check variable, assigns it the value of FQFTGen, using the new $Topic and $QNo as control’s, and checks to see if $check equals -999. If it does then it sets the $msg variable to an error message, if it does not i.e. a row was returned as a result, then it sets $msg to a confirmation message.

Finally, it re-directs you back to adminScreen.php, and displays the $msg variable.

### 3.4.3 updateQuestion.php

I am not going to say much on this page as the difference between updateQuestion.php and addQuestion.php are very superficial; the $msg variable’s messages change slightly and updateQuestion(….) is called rather and insertQuestion(…).

### 3.4.4 deleteQuestion.php

The functional way that delete question works is identical to the previous two admin pages, unsurprising perhaps given that they are all designed to check that valid inputs are given, convert the inputs to PHP variables and then input those variables into one of three SQL statements.

In this way the only major difference here is that due to the nature of a delete function, I have to check before I run the query to see if the question exists. This stops me giving out false positives due to attempting to delete a question that was already missing.

### 3.4.5 Delete/Insert/Update Question Functions

function insertQuestion ($Topic, $QNo, $QText, $AText1, $AText2, $AText3, $AText4, $CorrectA, $connector)

function updateQuestion ($Topic, $QNo, $QText, $AText1, $AText2, $AText3, $AText4, $CorrectA, $connector)

function deleteQuestion ($Topic, $QNo, $connector)

Much like the FQFTGen function discussed previously, these three are just formatting functions. By decoupling the insert/update/delete function from the pages I a) gain access to them on other parts of the site and b) make the code on the original pages easier to read. In this case however the normal search function could not cover these function’s purposes as the Select element is hard-coded into Search. Other than convenience, these function’s only purpose is to take a series of pre-defined inputs and convert them into a valid SQL statement.

## 3.5 The Student Interface

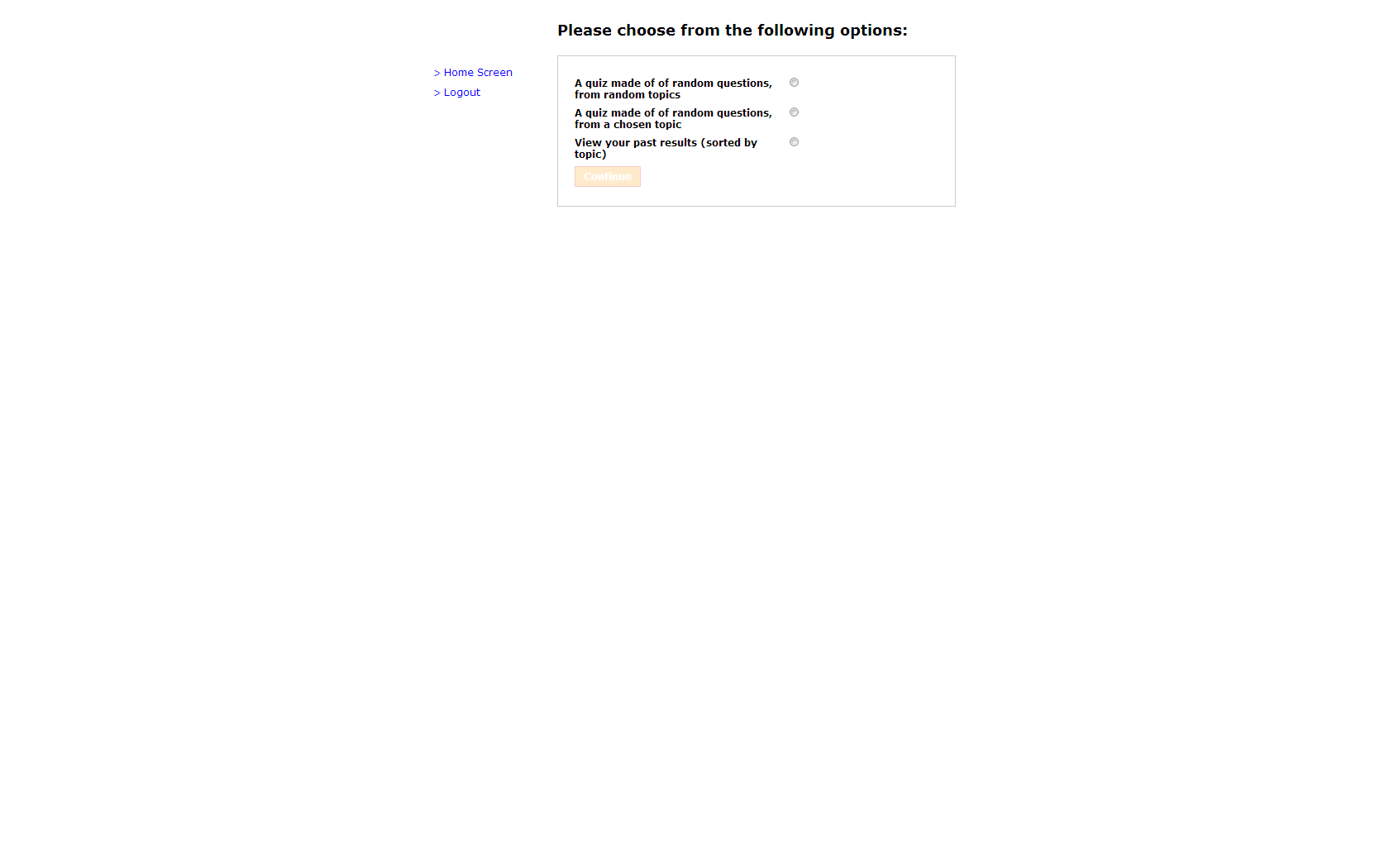
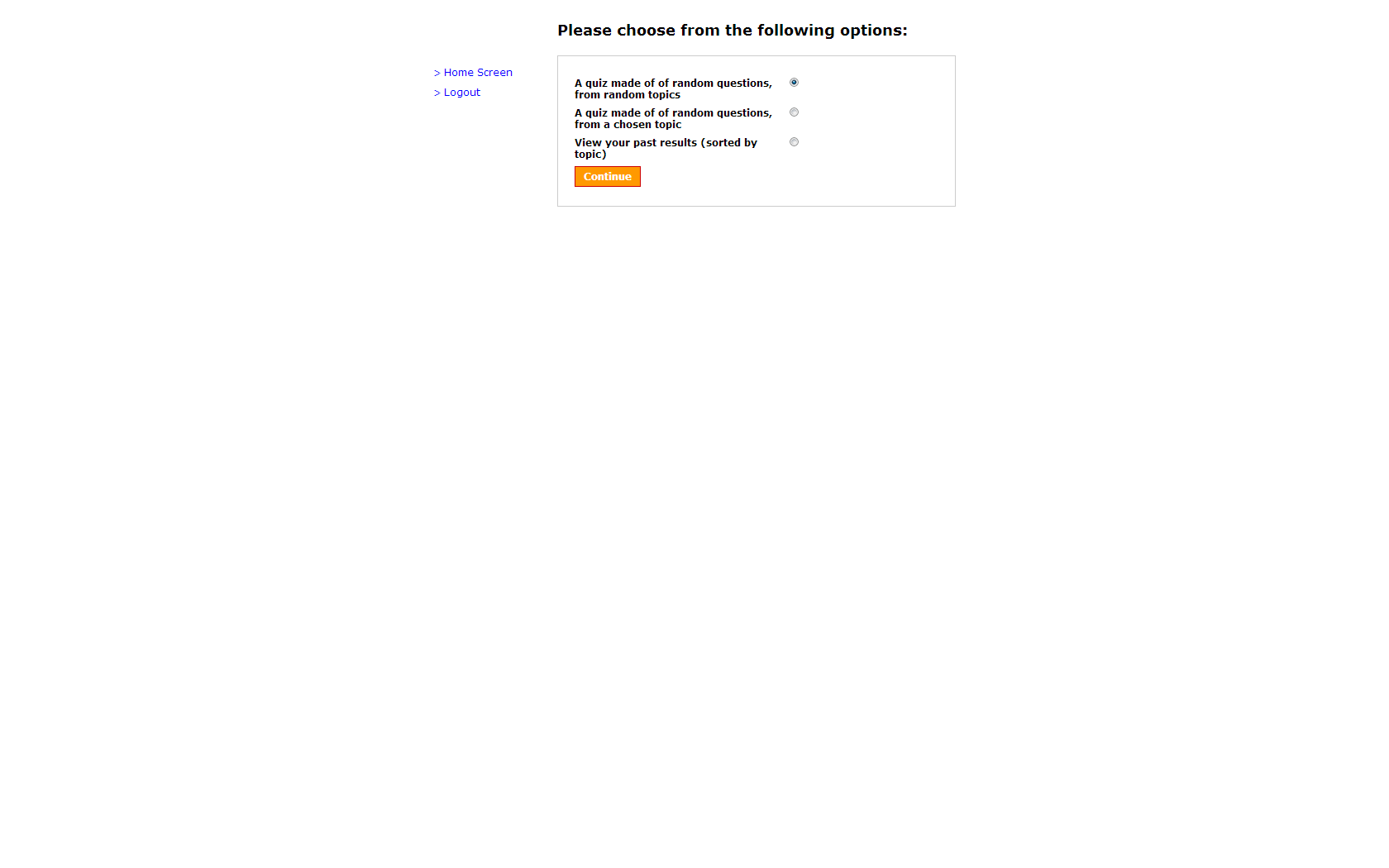
The student interface is the core focus of this project; it can be divided into three distinct categories

1. Choose an exam type
2. Sit an exam
3. View your past results

As I demonstrated earlier in the communication diagrams, once you actually sit the exam, the exam interface is identical irrespective of whether the exam you are sitting is made up from questions on one topic or questions from a random series of topics. For this reason I will first explore the two different ways you can generate an exam, and then move onto examining the exam interface itself.

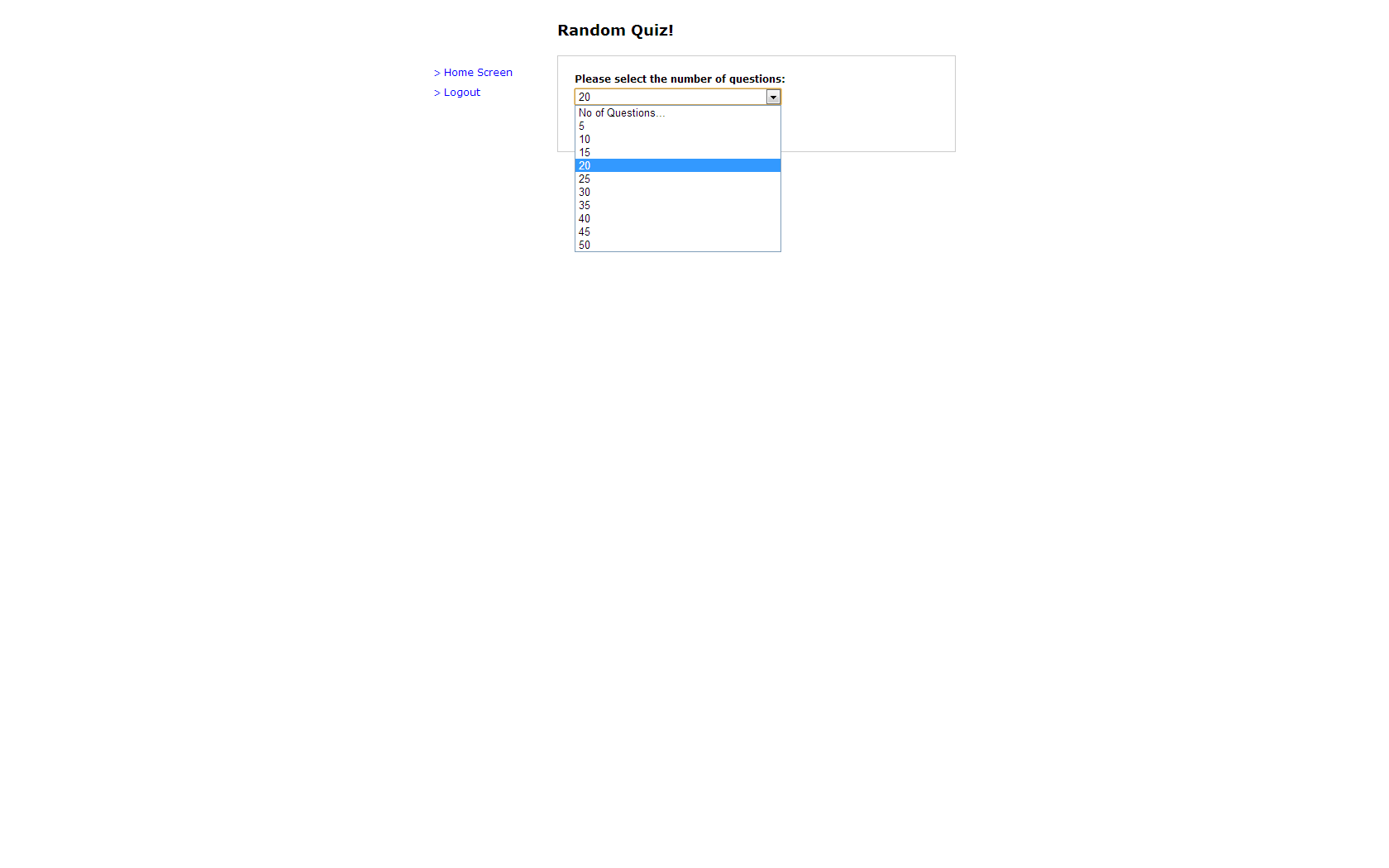
### 3.5.1 studentScreen.php

The student homepage is actually just a portal to one of three other pages. Unlike adminScreen.php, this page doesn’t actually interact with the database. Just like with adminScreen.php the continue button starts locked and becomes active when you choose an option. The code is the same as before so I will move straight onto randomExam.php.

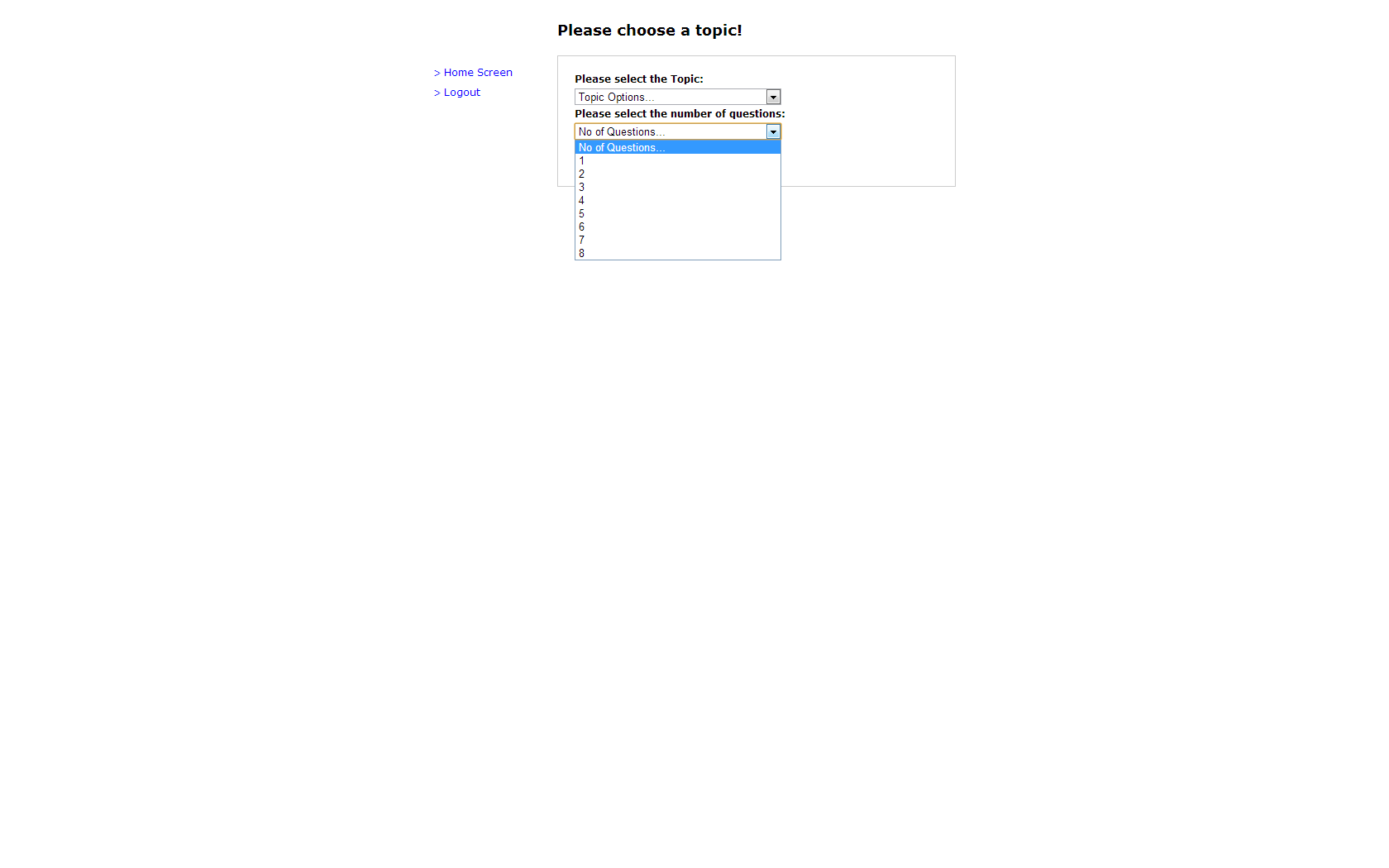
### 3.5.2 randomExam.php

Again, this is a reasonably simple page. Beyond the common locked continue, the main point of note about this page is the drop-down menu. Like the topic dropdown before this has been created using a PHP loop. In this case it is a simple enhance for loop ($x=5; $x<51; $x=$x+5). This repeats the create option HTML function until $x=55, ten times. There is also a hidden field called ‘examType’ on the page which is set to the value ‘randomExam’. This is used by examGenerator.php to establish which sort of exam it should generate.



### 3.5.3 topicExam.php

This page is identical to randomExam.php except that the ‘examType’ hidden field is set to the value ‘topicExam’, there is an additional drop down menu for choosing the topic and the maximum number of questions is eight.



### 3.5.4 examGenerator.php

examGenerator.php is, in essence, a lot of very simple code which is broken up by a series of if statements. The first block of code is there to adjust the exam Length. It starts by checking to see if ‘QNo’ has been posted by the previous page, if it has been set, it performs a second check to see if its value is greater than ‘0’. Only if both of these checks are passed will it set the variables in the middle. As a note, I intentionally performed these checks separately in order to avoid errors when ‘QNo’ is unset and it is compared against zero.

After having checked ‘QNo’ and hopefully set the variables, ‘examType’ is then checked. Here the code splits depending on whether the ‘examType’ is ‘topicExam’ or ‘randomExam’. If a random exam was picked then, after checking ‘Topic’ is set and greater than zero, and the $ExamLength variable is set, it proceeds to create an exam using the RQFTGen method, passing the topic number as a reference. A similar procedure happens if ‘examType’ is ‘randomExam’, the main difference being the exam generation method (RQRTGen).

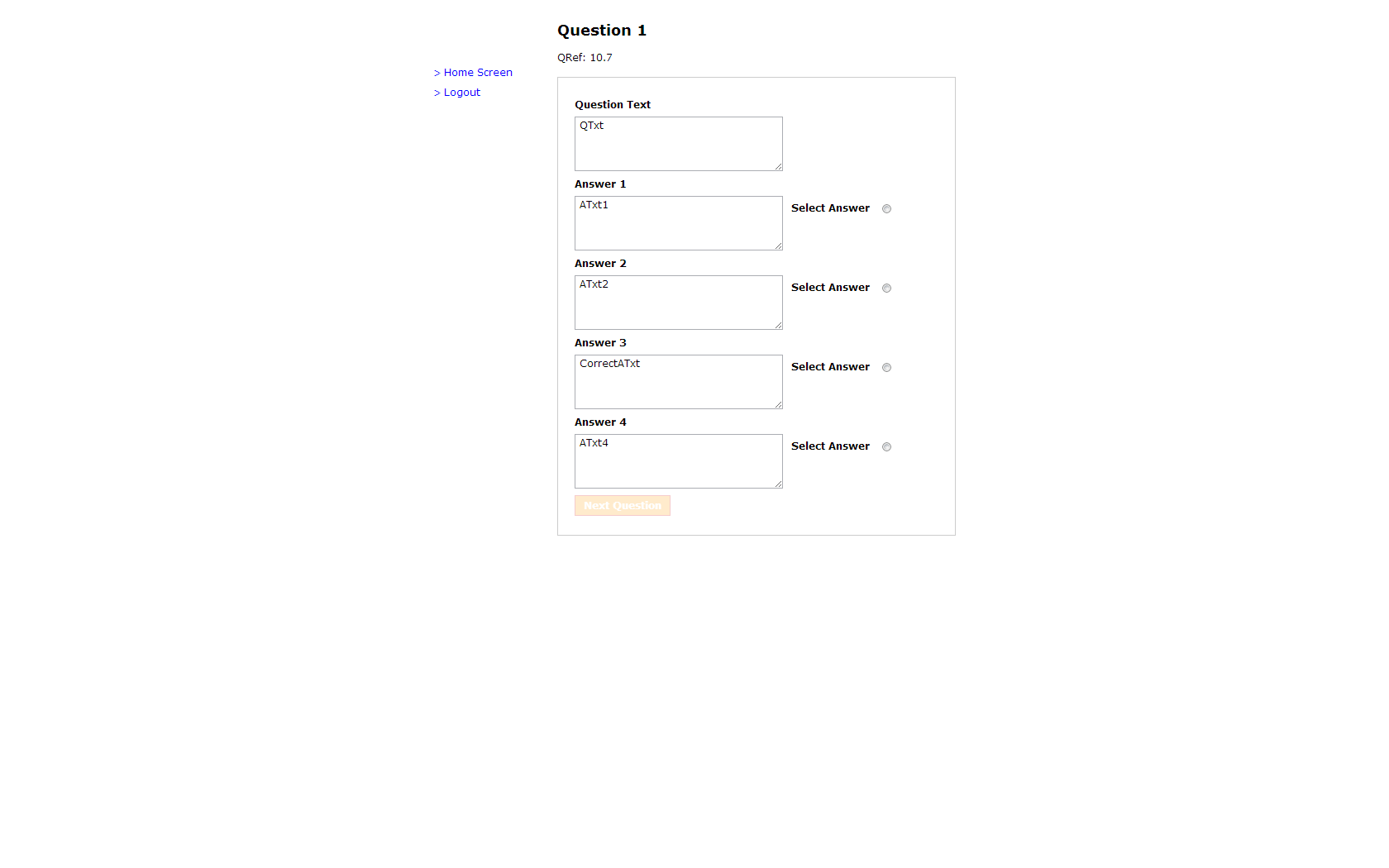
In both cases a number of session variables are set which are then passed on via the browser. The main ones of note are those used by exam.php and answer.php; ‘ExamLength’, ‘QuestionNo’, ‘Answers’ and ‘Exam’. The specific topic exam also sets up two more sessions variables ‘examType’ and ‘Topic’, these are used in the final results.php page as a condition and control variable respectively.

In both cases where an exam is generated the user is sent straight to exam.php to being the test. We now move to the bottom if clause. Due to the exit(); commands given in the exam generators this will only be reached if either the ‘QNo’ variable or ‘Topic’ and ‘QNo’ variables have not been set. The final part of this page will send the user to either topicExam.php or randomExam.php if the ‘examType’ variable is set to one of those values, or to the studentScreen.php homepage if it is either unset or set to a third value.

### 3.5.5 exam.php

This page, and all the other exam pages, will check on load to see if the session variables needed for them to function have been activated. This stops someone accessing them when an exam has not been set. It also has the bonus of recording their progress if they accidentally hit the home button; they can just hit back in their browsers and be right where they left off.

Having received the sessions variables it needs exam.php will then check to see if the exam is over by checking the $examLength against the current $questionNo. If this value is exceeded it will port the user directly to the results.php page, otherwise it create a variable called $control which is initially set to the value ‘1’. This control variable with then be used to make sure that the correct question is printed to the screen.



The HTML form is then populated in the same way as the other forms in this system and upon submit the user’s answer is recorded and passed to answer.php. Like the other forms the Next Question button is unlocked when you click on the radio button, unlike the other forms, the quiz will always go to answer.php as opposed to a dynamically determined value.

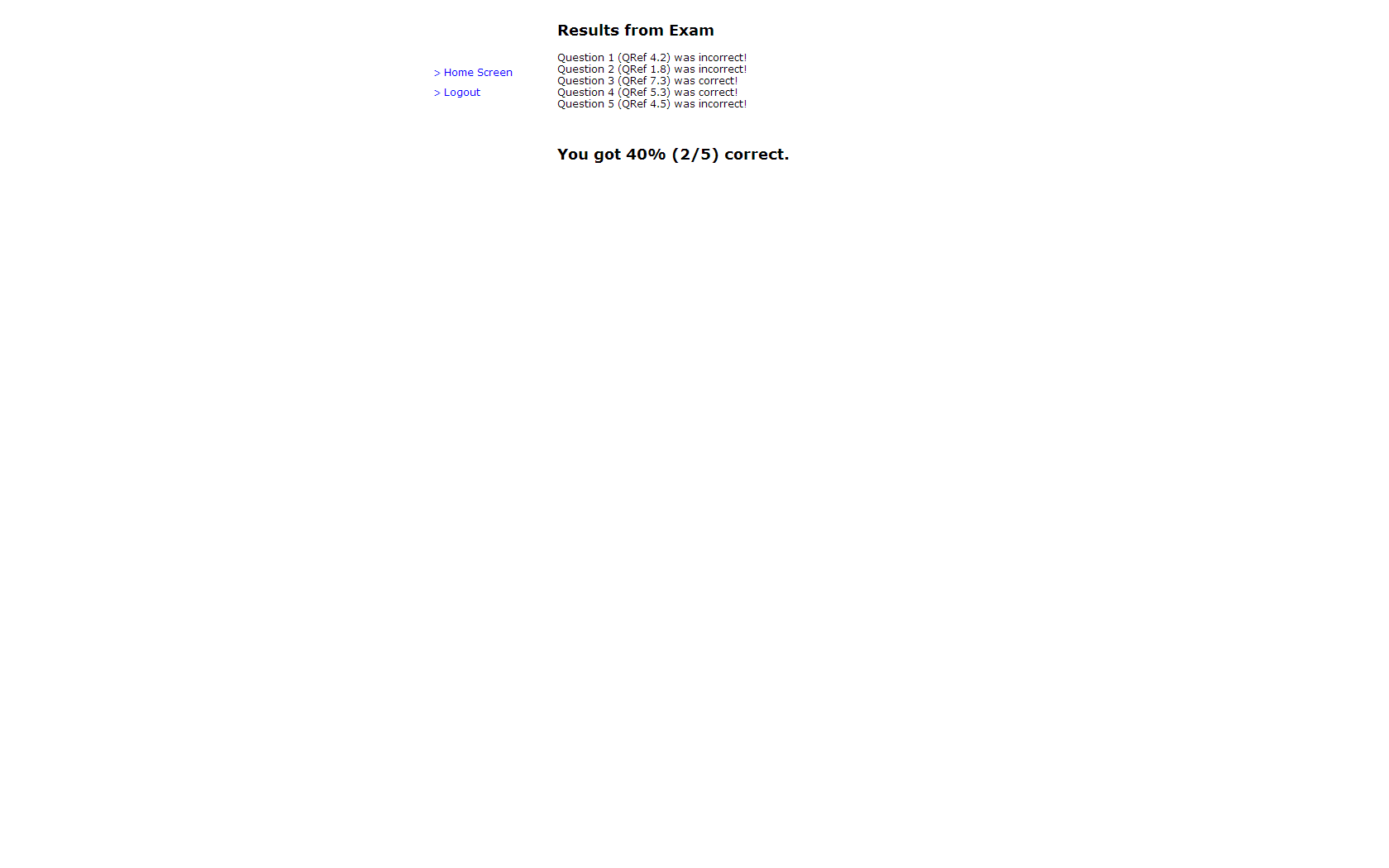
### 3.5.6 answer.php

After checking that the user has an exam registered, the code on answer.php checks to see if they chose a correct answer using (if (isset($\_POST['CorrectA'])). If they did then it stores the last answer in the correct array postion (overwriting ‘0’) and increments the ‘QuestionNo’ session variable and the $QuestionNo PHP variable.

It then checks to see if $QuestionNo is greater than $\_SESSION['ExamLength'], if it is then it sends the user to results.php, if not then back to the exam to take the next question.

### 3.5.7 results.php

The body of this code is made up from two elements. First, we have a foreach loop which is used to print the exam question references (in order) and tell you if you got them correct. The second is an update command which is used to set the student mark in the database (if they sat an exam on a specific topic).

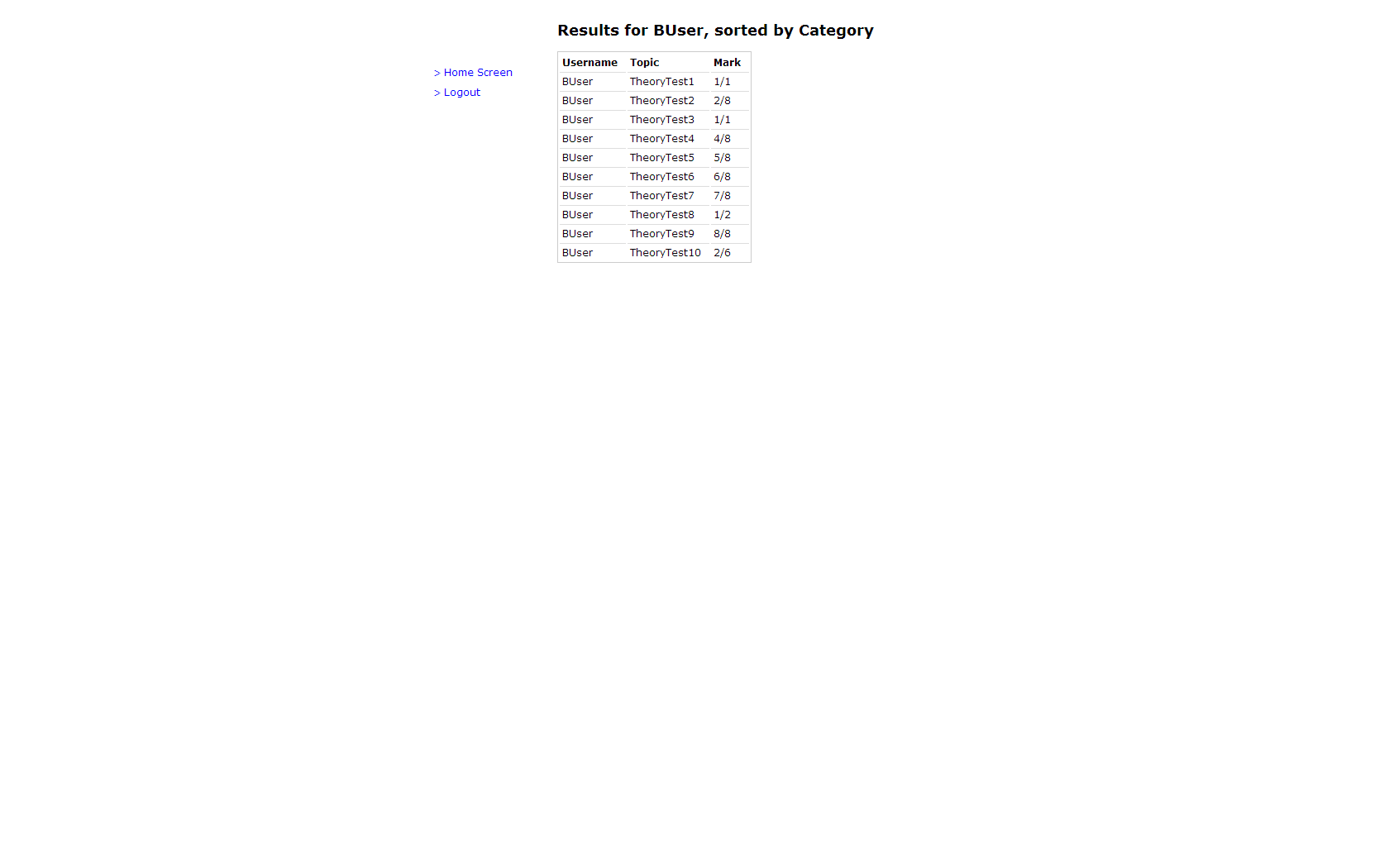


If we examine the print statement, first we can see that it is made up of two echo commands. If the answer the user chose in the exam matches the corresponding value of ‘CorrectA’ then the $noCorrect variable is incremented, and message A is printed, if the value does not match, then message B in printed. After the if statement the $control value is incremented and the process repeats until it every question has been iterated through.

If your exam was random then the process ends and the exam session variables are unset. If your exam was on a specific topic however the sessions runs the query inside the inner if statement and the results are written to the database in the form of x/y.

### 3.5.8 pastResults.php

The main objective of this page is to print the results for a user in a clear, readable fashion. For this reason I created the function PAR ($array). The PAR (Print Array as Readable) function is designed to dynamically create a table and print it when given a two-dimensional array as an input (such as those found in the results table of a PDO query). It does this using two foreach loops, one to print the headers, and one for the array contents. The results of this can be seen below:



# Conclusions

### 4.1 Project Summary

The project proposal for this document was to build the GUI and Core Systems for a multiple choice style exam. A number of core systems were listed and the main requirements were met. The only requirement that has not been met was the ability to register/unregister students from topics as this was deemed unnecessary.

As far as the non-functional requirements listed in the proposal, I was able to add the password protection, but was unable to find a reasonable solution to the question analysis interface. This is an area which could be explored in future iterations of the project.

### 4.2 Recommendations for Future Work

### 4.2.1 User Management

At the moment the package is ‘shipping’ with a predetermined ten users. This is fine for testing purposes but for a real-market application, the ability to add/delete users and the facility to change items such as the user’s name or password could be added.

Additionally the only way to currently make a user an administrator is to change the MySQL database manually (changing a ‘0’ to a ‘1’ in the logins table).

### 4.2.2 Contextual Drop-Down Menus

One of the drawbacks of having to teach myself both PHP and HTML for this project was that although I always had a clear understanding of the websites intended functionality; I struggled converting this into working code. One feature that I was unable to figure out was how to create contextual drop-down menus which interact with PHP correctly. The basic idea was that once a user selected a Topic, rather than then typing in the Question Number, a new drop-down menu would appear, having queried the database for a list of all available questions.

This is, unfortunately, one of the downsides of PHP being server-side and HTML being client-side, it’s easy to get PHP to talk to HTML, but more difficult to get HTML to talk back to PHP.

### 4.2.3 Varying Number of Answers

At the moment the database is locked into a four potential answers model. A feature that may be worth considering is the ability to have more or less potential answers, as well as cases where multiple answers are correct (i.e. 3/5 are correct, pick those three).

### 4.2.4 Lack of External User Testing

My final comment is that due to this system being developed in isolation, there has been no external user testing. While this could be fine, as the GUI is seemingly straightforward, it is likely that there are aspects of the interface which make sense to me but may not be as intuitive to others. For this reason I believe that external testing could be used to further develop the final product.

End

# Bibliography

.Net Magazine. (n.d.). *Using PHP Data Objects*. Retrieved 02 24, 2013, from .Net Magazine: http://www.netmagazine.com/tutorials/using-php-data-objects

Amankwah, F. K. (2004). An Application to Manage Multiple-Choice Exams. London, UK: Birkbeck College.

Bennett, McRobb, & Farmer. (2006). *Object-Orientated System Analysis and Design Using UML, Third Edition.* McGraw Hill.

Chapple, M. (n.d.). *Entity-Relationship Diagram*. Retrieved May 3, 2013, from About.com Databases: http://databases.about.com/cs/specificproducts/g/er.htm

Kyrnin, J. (n.d.). *What is HTML*. Retrieved May 01, 2013, from About.com: http://webdesign.about.com/od/htmlxhtmltutorials/a/what-is-html.htm

Mountain Goat Software. (n.d.). *User Stories*. Retrieved May 02, 2013, from Mountain Goiat Software: http://www.mountaingoatsoftware.com/topics/user-stories

Oracle Corporation. (n.d.). *MySQL:: Why MySql*. Retrieved May 02, 2013, from MySQL Homepage: http://www.mysql.com/why-mysql/

PHP. (n.d.). *PHP: Hypertext Preprocessor*. Retrieved May 01, 2013, from PHP Homepage: php.net

The Apache Software Foundation. (n.d.). *Welcome! - The Apache HTTP Server Project*. Retrieved May 02, 2013, from Apache Homepage: http://httpd.apache.org/

# Code Listing