

# Group Report

Group 2- Tom Freestone, Blessing Ekpe, Sean Biscoe, Sam Coles

May 2025

Website URL: <http://295group2.sci-project.lboro.ac.uk/IBayStable/LoginPage/main-G02.html>

Username=test

Password=test

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Main contributions . . . . .	2
1.2	Methodology . . . . .	2
1.3	SQL Injection prevention . . . . .	3
1.4	Client-Side evaluation . . . . .	3
1.5	Server-Side evaluation . . . . .	4
1.6	Conclusion of Client-Side and Server-Side technologies . . . . .	4
<b>2</b>	<b>Design</b>	<b>4</b>
<b>3</b>	<b>Database</b>	<b>5</b>
<b>4</b>	<b>Login Page</b>	<b>6</b>
<b>5</b>	<b>Register Page</b>	<b>7</b>
<b>6</b>	<b>Buyer's Page</b>	<b>8</b>
6.1	Buyers Home Page . . . . .	8
6.2	Basket Page . . . . .	10
6.3	View Item Page . . . . .	10
<b>7</b>	<b>Seller's Page</b>	<b>11</b>
7.1	View Items . . . . .	11
7.2	Create Item . . . . .	12
<b>8</b>	<b>Improvements</b>	<b>13</b>
8.1	General improvements . . . . .	13
8.2	Buyer's View Item Page . . . . .	13
8.3	Seller's View Items Page . . . . .	13
8.4	Basket Page . . . . .	13
8.5	Security . . . . .	14
<b>9</b>	<b>References</b>	<b>14</b>

---

# 1 Introduction

## 1.1 Main contributions

Blessing has contributed: Sellers client-side code. Blessing has also done some client-side code for the register and login pages. Sean has contributed: Buyers client-side code. Sean has also done some client-side code for the register and login pages as well as helping set up initial databases. Tom has contributed: Server-side code for all pages. Sam has contributed: Writing the report and set up initial databases.

Note that these are just the main contributions that each person made and not everything they did.

## 1.2 Methodology

Throughout the project, we have used a combination of client-side and server-side programming for each page we have developed. We use HTML to lay out the different elements of each web page such as headers and forms.

CSS is used to style the elements used in our webpage; for example, we have styled the login box of our Login Page to use white text, have a black background and have padding 10px 20px (10px for top and bottom and 20px for left and right.)

We use JavaScript to allow for dynamic programming. This means that we can add functionality to our web pages; with just HTML and CSS, we would be limited to just aesthetics. JavaScript allows us to link client-side programming to server-side programming by calling PHP functions. We use JQuery within JavaScript to create functions which handle user behaviour such as deleting an item from the basket when the user clicks the delete button. We also use AJAX within our JavaScript files to asynchronously interact with the web pages and databases. This means that we can update a web page without the need for the user to reload the page.

PHP is the main server-side language that we use. Users are unable to see this code as it is executed on the server and the results are sent to the web browser. We have used PHP to check if users are logged on for pages which require this such as the basket page. The PHP for this is shown here:

```
// Check if the user is logged in
if (!isset($_SESSION['userId'])) {
    die(json_encode(value: ["error" => "User not logged in."]));
}
```

Figure 1: PHP: logged in check

We also use PHP to write SQL and perform it on the database. After doing this, we can relate the data back to the client-side which can then display it.

### 1.3 SQL Injection prevention

Using PHP can create some vulnerabilities in the website which could allow SQL Injection to occur if not dealt with appropriately. SQL Injection is where a user is able to enter SQL in place of regular text which is then sent to the server and returns data from the database. If the user's SQL statement is treated without any precautions, the statement can do anything to the data in the database such as view it or delete it. An example of an SQL Injection attack is if a user wrote SQL to return all the usernames and passwords in a database. They could then use this to access another users account. For our Ibay website, we have taken measures to prevent SQL Injection. By using prepared statements, SQL injection should not be possible. Prepared statements use templated SQL statements with parameters that have no value; these are labelled with '?'. The database then performs query optimisation on the statement and stores the result but does not actually execute the statement. Once the values of the parameters are confirmed, the database then executes the statement with the parameters taking the values which are now decided. This is useful against SQL injection as the templated statement does not come from the user. Here is an example of prepared statements in our work:

```
// Check if username and password are valid
$membersTbl = "iBayMembers";
$sql = "SELECT * FROM $membersTbl WHERE username = ?
      AND password = ?";
$stmt = $db->prepare(query: $sql);
$stmt->bind_param( types: "ss",var: &$username, vars: &$password);

try{
    $stmt->execute();
}
catch (Exception $e) {
```

Figure 2: SQL Injection prevention

### 1.4 Client-Side evaluation

Client-side solutions give users faster response times. If we do not send data to and from the server, any actions the user take will be completed quicker. For example, in our Buyers page, we can click on a category at the top and the webpage will almost immediately scroll towards that category. However, when we use server-side programming such as with our login system, the user has to wait for a response. This is usually less than a second but still not instantaneous. When logging in, the server has to receive the username and password entered by the user. After that, the next web page is loaded.

Furthermore, client-side technology can reduce the traffic on the server. For instance if thousands of users are interacting with the website, the server could

struggle to handle all the requests being sent to it if server-side technology is predominantly used. However, client-side technology uses the user's device to handle its tasks. This would mean that there is less stress on the server and it can handle all its requests.

### 1.5 Server-Side evaluation

Server-side solutions do not allow the user to see the source code. This helps to make a website more secure as the code cannot be studied for vulnerabilities. The server will take inputs from the client-side and only returns an output; no information is given to the user on how the server produces an output for any given input(s). This makes it harder to attack the website as users do not know how the server-side code works.

As mentioned above, server-side technologies usually have slower speeds than client-side.

### 1.6 Conclusion of Client-Side and Server-Side technologies

Both technologies have advantages and disadvantages which is why we would use a combination of the two technologies if we were to undertake another project. In particular, server-side is useful when handling data as it is more secure to store data on the server so we would use server side technologies in this case. As client-side technology has faster speeds, we would use this in all other cases when we do not have to access data from the server.

## 2 Design

We have designed our logo be bold, simple and easily recognisable; its colour scheme is red and black. This will stand out to customers and hopefully keep them engaged on our website. The logo is pictured below:



Figure 3: Logo

We have decided to use a light theme for our IBAY website. This creates a

friendly and welcoming environment for the user. We do however have a dark header on each page with our dark IBAY logo. Below is our favicon to help users know which tab is for the IBAY website. The favicon is displayed on the IBAY tab in the users web browser.

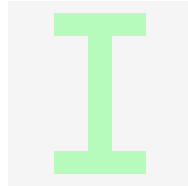


Figure 4: Favicon

We also make use of the colour green across our website. This colour is only used on clickable buttons. This means that the users know they can click anything green and then it will do as it says. For instance, the green 'sell' button takes the user to the buyers page.

### 3 Database

We have created the 3 databases using the SQL provided in the project specification. In addition to this, we have added 'username', 'firstname', 'surname', 'gender' and 'phone number' to the iBayMembers table. The iBayItems table remains the same but with the iBayImage table, we have changed 'imageSize' to 'imageSizeKB.'

The iBayItems and iBayMembers tables were populated using the example data from the cob295db database and then the iBayMembers table was further populated through the use of AI, in particular chatGPT. The iBayImages table was populated by us with the images used originating from the web.

In our files, connection to the database is gained using one file 'connect.php' Other files then call the 'connect.php' file when it needs to connect to the database.

## 4 Login Page

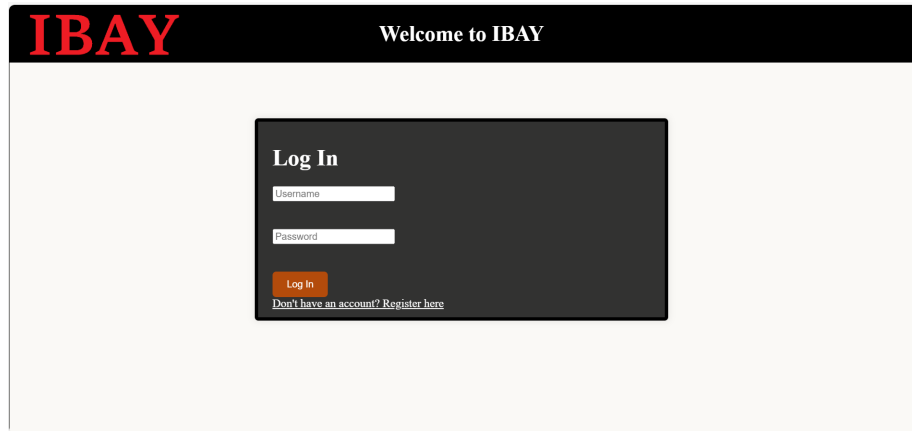


Figure 5: Login Page

On the login page, we have used the HTML file 'main-G02.html' to place the logo in the top left and have a 'Welcome to IBAY' message in the middle. In the centre of the page, we have a 'Log In' form for the user to fill out and a hyperlink at the bottom of that to bring new customers to our register page to create a new account. The login page is the first page which the user will encounter. Each component of this page is styled by using the 'loginPage.css' file. Here is an example of some CSS styling used for the login page.

```
/*Styling for header*/
.header {
  position: absolute;
  top: -20px;
  width: 100%;
  height: 80px;
  background-color: #000;
  color: #fff;
  text-align: center;
  line-height: 80px;
  font-size: 30px;
  font-weight: bold;
  border-bottom: 5px solid grey;
}
```

Figure 6: CSS example code

Once the user has entered a username and password, these are sent to the 'login.php' file using the post method. This allows the 'login.php' file to compare this information with that in the 'iBayMembers' database. In order to do this, we have added SQL to return the rows which have the specified username and password. This allows us to check if the username and password are correct and

that they match. If this is the case, the 'buyersHomePage.html' file is loaded and the user is connected to the buyers page.

## 5 Register Page

Figure 7: Register Page (zoom 80%)

This page is accessed by clicking the link in the Login Page which takes us to the 'signUpPage.html' file. From here, the user can return to the Login Page by clicking the link near the top of the Sign Up box. In this box, there is a form where the user can enter all the information needed to add the new user to the 'iBayMembers' database. There are 2 textboxes for password (the second being for confirmation.) The user is also required to enter their: firstname, surname, email address, address, postcode, phone number, date of birth and gender. We also have buttons at the bottom to allow the user to reset their responses and one for them to sign up once all fields have been filled out. Here is an example of how we made, textboxes, radio buttons and our sign up/ reset buttons.

```
<!-- HTML for textbox for date of birth -->
<label for="dob">Date of Birth:</label>
<input type="date" id="dob" name="dob" value="1990-01-01" required><br><br>

<!-- HTML for radio buttons for gender -->
<label for="gender">Gender:</label>
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male" checked>Male
<input type="radio" name="gender" value="other">Other
<br><br>

<!-- HTML for submit and reset buttons -->
<input type="submit" class="btn" value="Sign Up" id="signUpBtn" name="signUp-Submit" >
<input type="reset" class="btn" value="Reset" id="resetBtn">
```

Figure 8: Buttons example

The HTML file links to the JavaScript file 'signUp.js' where the user's responses are validated against many constraints. In particular, the password is limited so that it must have at least: 8 characters, 1 number, 1 uppercase letter, 1 lowercase letter and one special character. There is one exception to this which is the password 'test' purely for the assessment of our project.

```
//Event listener for password
password.addEventListener("input", function () {
  var passwordVal = document.getElementById("password").value;
  //If password is "test", skip password validation
  if (passwordVal == "test") {
    password.setCustomValidity('');
  } else {
    //Check if password is at least 8 characters
    if (passwordVal.length < 8) {
      password.setCustomValidity("Password must be at least 8 characters");
    } //Check if password contains at least one number
    else if (passwordVal.search(/[0-9]/) < 0) {
      password.setCustomValidity("Password must contain at least one number");
    } //Check if password contains at least one uppercase letter
    else if (passwordVal.search(/[A-Z]/) < 0) {
      password.setCustomValidity("Password must contain at least one uppercase letter");
    } //Check if password contains at least one lowercase letter
    else if (passwordVal.search(/[a-z]/) < 0) {
      password.setCustomValidity("Password must contain at least one lowercase letter");
    } //Check if password contains at least one special character
    else if (passwordVal.search(/[!@#A-Z0-9]/) < 0) {
      password.setCustomValidity("Password must contain at least one special character");
    } else {
      password.setCustomValidity('');
    }
  }
  password.reportValidity();
});
```

Figure 9: Password Constraints

## 6 Buyer's Page

### 6.1 Buyers Home Page

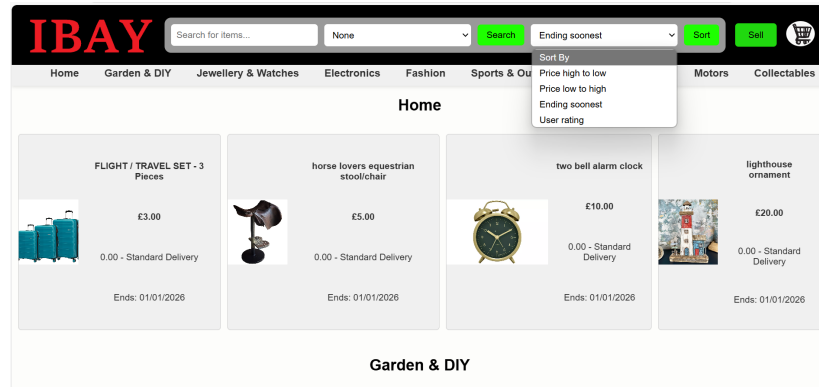


Figure 10: Buyer's Home Page



The HTML file 'buyersHomePage.html' is run from the log in page once the user has logged in. Styling here comes from 'buyersPage.css'. Under the header, we have a navigation bar which contains clickable buttons for all categories of items we have listed on the website. The idea of this navigation bar was given by Sean's family after testing out the website and providing feedback. Once a category is clicked on, the web page jumps to the section of the category selected. In the header, we also have a drop down box, this does not currently work however we have php code which will sort the items by either price (high-to-low and low-to-high), user rating, or end date. To do this, we have SQL statements to order by price, end time or user rating.

```
$sqlItemsSORTSection = "";
if (isset($_GET['sort'])) {
    $sort = $_GET['sort'];
    if ($sort == 'priceLowToHigh') {
        $sqlItemsSORTSection = "ORDER BY price ASC";
    }
    elseif ($sort == 'priceHighToLow') {
        $sqlItemsSORTSection = "ORDER BY price DESC";
    }
    elseif ($sort == 'endingSoon') {
        $sqlItemsSORTSection = "ORDER BY finish ASC";
    }
    elseif ($sort == 'userRating') {
        $sqlItemsSORTSection = "ORDER BY userRating DESC";
    }
    else {
        // Invalid sort option, handle error
        die(json_encode(value: ["error" => "Invalid sort option."]));
    }
}
```

Figure 11: Sorting Items PHP code

There is also a sell button which takes the user back to the seller page. The basket icon takes the user to the Basket Page.

## 6.2 Basket Page

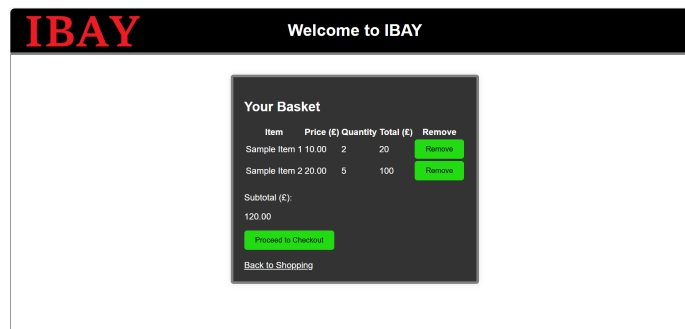


Figure 12: Basket Page

The basket page is not fully complete and will not return the user to the correct page when they click the 'back to shopping' button. We do however have example data of what the basket would look like if it was completed. Here, styling comes from 'basket.css'.

## 6.3 View Item Page

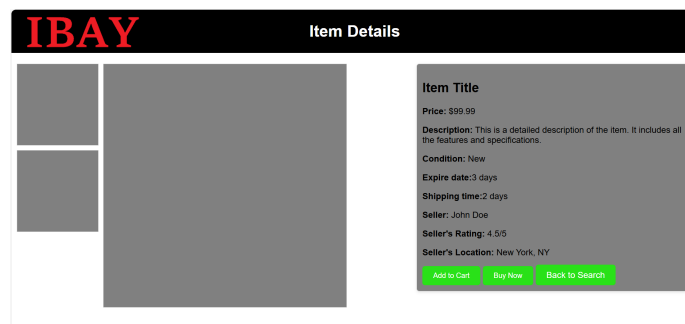


Figure 13: Buyer View Item Page

This page is incomplete as we have not yet gotten the web page to load the item the user is interested in. We do however have a templated page. We have the PHP file 'displayItem.php' which returns the details about the particular item in a JSON format including images.

## 7 Seller's Page

### 7.1 View Items

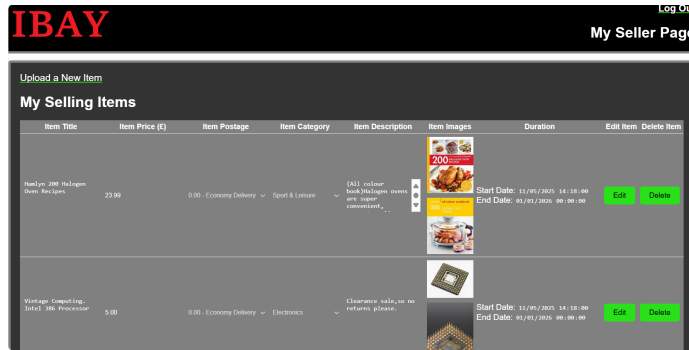


Figure 14: Seller view items (80% zoom)

This page is accessed by clicking the 'sell' button on the buyers page. As pictured above, our Seller's page has the header clearly letting the User know that they are on the Seller Page. Underneath that, is a box containing a link to the item upload page for sellers and the box also contains the items which the current user is selling. The items the user has for sale have the following pieces of information: Item Title, Item Price, Item Category, Item Description, Duration and there are also buttons which allow the user to either edit or delete the item. The HTML file 'ViewItemPage.html' defines the table where the items the user has for sale are listed. This file then calls the 'seller.js' file which checks the items and their information against suitable constraints such as having the end date to be after the start date (code pictured below) and having the price in an acceptable regex format.

```
//Event listener for making sure start date is not greater than end date
$(document).on('change, blur', '#startDate, #endDate', function() {
    var startDate = $('#startDate').val(); //Get the value of the input field
    var endDate = $('#endDate').val(); //Get the value of the end date input field

    //Check if the start date is greater than the end date
    if (startDate > endDate) {
        this.setCustomValidity("Start date cannot be greater than end date"); //If the start date is greater than the end date,
        //set the custom validity message
    } else {
        this.setCustomValidity(''); //If the start date is not greater than the end date, set the custom validity to empty
    }

    this.reportValidity(); //Report the validity of the input field
});
```

Figure 15: seller.js example code

'ViewItemPage.html' also calls the 'SellerView.js' file to populate the items table on the page. This file includes a jQuery function to fetch the item data

from the 'iBayItems' database. The function contains AJAX code calling the php file 'fetchItems.php'. Items belonging to the current user are then added to an array and returned to 'SellerView.js' in a JSON format. Next, using AJAX, we loop through the items and append them to the table and then the table is displayed.

## 7.2 Create Item

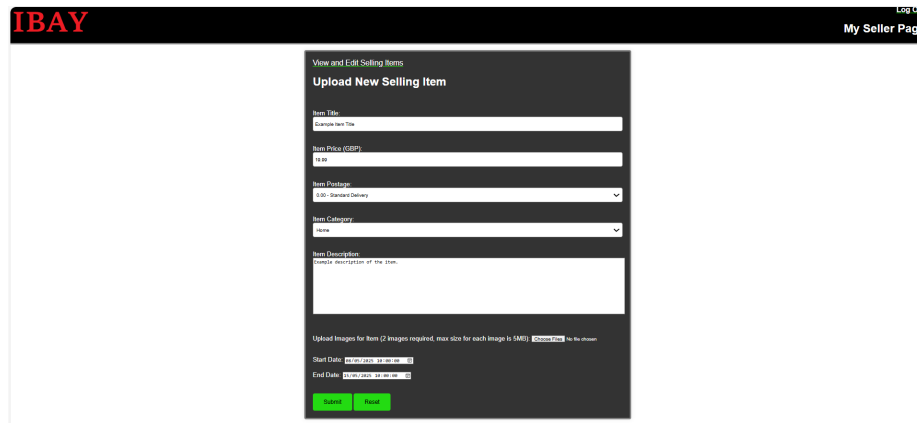
The screenshot shows a web interface for a seller. At the top, there is a black header with the 'IBAY' logo in red and white on the left, and 'Log Out' and 'My Seller Page' in white on the right. The main content area is white and contains a dark grey modal window titled 'View and Edit Selling Items' with a sub-header 'Upload New Selling Item'. Inside the modal, there are several form fields: 'Item Title' (text input), 'Item Price (GBP)' (text input), 'Item Package' (dropdown menu with 'UK - Standard Delivery' selected), 'Item Category' (dropdown menu with 'Home' selected), and 'Item Description' (text area). Below these fields, there is a note about uploading images: 'Upload Images for Item (2 images required, max size for each image is 5MB)'. At the bottom of the modal, there are 'Start Date' and 'End Date' fields with date pickers, and two green buttons labeled 'Cancel' and 'Save'.

Figure 16: Seller upload item (50% zoom)

This page is accessed by clicking 'Upload a New Item' which then links to the 'createItems.html' file. This page is styled using 'sellerPage.css'. We have a link to log out on this page which takes the user back to 'main-G02.html'. Similarly to when we registered a new user, we have used a form to allow the user to give responses for the fields needed to add a new item to the 'iBayItems' database. We have again included elements such as text boxes and buttons but we also have a drop-down menu to allow the user to choose the correct category for their item. Users are also forced to upload 2 images when adding a new item and that the size of each file does not exceed 5MB. This was achieved on the client-side by using the JavaScript file 'seller.js' with the following code:

```

//Event listener for making sure user uploads only 2 files
$(document).on('change', '.imgFiles', function () {
    //Check if the number of files is greater than 2
    if (this.files.length > 2) {
        imgFiles.setCustomValidity("You can only upload 2 files");
    }
    //Check if the number of files is less than 2
    } else if (this.files.length < 2) {
        imgFiles.setCustomValidity("You must upload 2 files");
    }
    } else {
        //Check if the file size is greater than
        if (this.files[0].size > 1048576*5 || this.files[1].size > 1048576*5) {
            imgFiles.setCustomValidity("File size must be less than 5MB"); //Set the custom validity message
        }
        } else {
            imgFiles.setCustomValidity(''); //If the file size is less than 1MB, set the custom validity to empty
        }
    }
});

imgFiles.reportValidity();
});

```

Figure 17: Image upload constraints in JavaScript

## 8 Improvements

### 8.1 General improvements

From any page after the user logs in, they can return to the main buyer's page by clicking the logo however this does not feel as intuitive as it perhaps should be. This could be avoided if there were other clickable elements inside the header of every page exactly like the buyer's page; inside the header we have other clickable elements such as the drop down boxes and a text box for a manual search and also in the navigation bar we have categories the user can shop by. If we were able to create a header for all our other pages like this, this would likely solve the problem and improve the user experience.

We currently only have the ability for the user to log out on our Seller's pages so we would add this to our Buyer's pages as well.

### 8.2 Buyer's View Item Page

Returning the details of the particular item would be a priority to us if we had more time to work on the website as the user should be able to get a more detailed description of the item upon clicking on it.

### 8.3 Seller's View Items Page

The whole page does not fit horizontally so we would adjust the scale accordingly.

### 8.4 Basket Page

Our Basket Page is not fully complete. There is functionality for removing items from the basket and a subtotal for the total cost of the basket however we do

not currently have the functionality of displaying the items in the basket and being able to add items to the basket. We would aim to complete the basket page if we were to continue developing IBAY.

## 8.5 Security

While SQL injection can be a big threat for websites, its not the only method to perform an attack. Other methods include: brute-force and man in the middle attacks. While we have incorporated a method to surpress SQL Injection, these other methods of attacks could be detrimental to IBAY. We would look for ways to prevent these if we had more time. For example, we could implement a maximum number of tries to enter a correct username and password combination; this would make a brute-force attack more challenging.

## 9 References

Documentation for JQuery can be found here: <https://api.jquery.com>