

Databases and Application Development

Author: Tomos Leggett · **Student Code:** 1901852 · **Module Code:**ACCA5022

Table of Contents

Databases and Application Development

- Table of Contents

- Introduction

- Security measures in place

 - Validation

- Security measures needed if this website were to go live

 - Watching out for SQL Injections

 - Protection against XSS attacks

 - Validate on browser and server side

 - Better passwords for the admin section

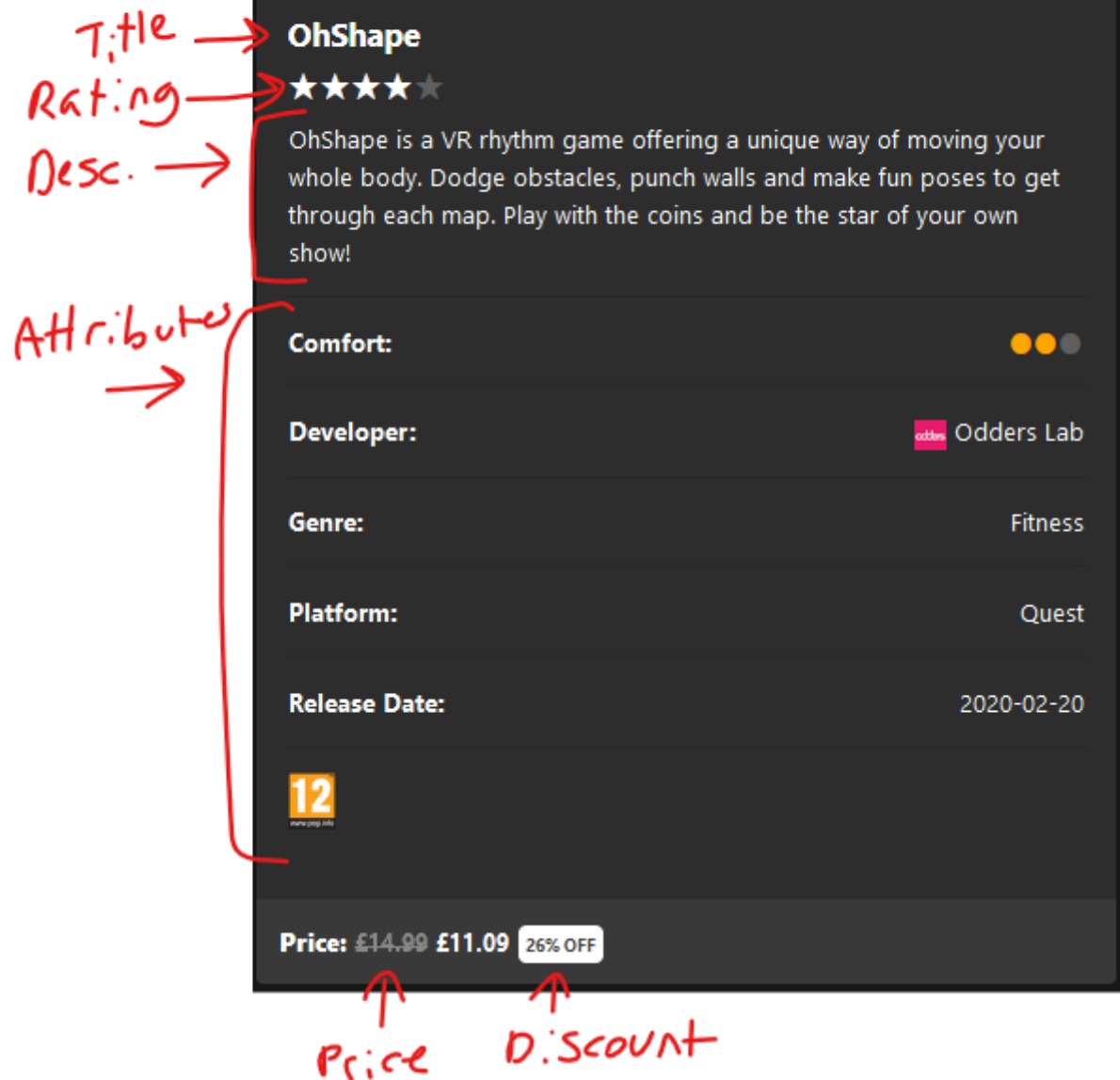
 - HTTPS everywhere on website

 - More protection against SQL injections

 - References

Introduction

The following report entails an analysis of the author's PHP web development project from a security point of view, the document will analyse various basic security measures taken to ensure the security of the website, but will also touch on several additional security measures that will need to be implemented on the website if it were to go live. The project itself is a visual database index of various Virtual Reality games. The website is designed to mimic a traditional digital store page with a search function and various "cards" that list each game accordingly, with a thumbnail and various attributes of the game.



Each card has several attributes followed by a price and an optional discount set by the admin. Admins log in through the admin section, which is guarded by a username/password field.

The Root admin login details are:

Username: Username

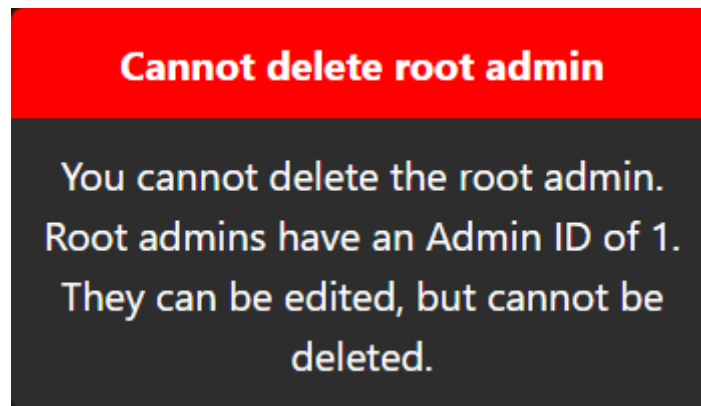
Password: Password

The admin panel allows admins to add, edit or delete items from the database, including games themselves, developers and additional admins. (The root admin cannot be deleted).

Security measures in place

Validation

To ensure correct data is added to the database, a validation system was created on various forms, for example, the prevention of deleting a root admin:



Validating that an admin has logged in:

```
if (@$_SESSION["username"] != "")  
{  
    |
```

and so on.

Security measures needed if this website were to go live

Watching out for SQL Injections

SQL Injections occur when a hacker attempts to manipulate or destroy your database by inputting a valid SQL query into an input box on a web page, which then proceeds to execute said query on your database. For example:

```
1 | DROP TABLE Employees;
```

This will inevitably drop the “Employees” table without your general awareness, causing possible mass destruction of your database, another example is the `OR 1=1` example. Where `OR 1=1` is always true:

```
1 | SELECT * FROM Employees WHERE EmployeeID = 105 OR 1=1;
```

This is completely valid SQL, which means it will return all employees from the database, which may include personal information.

Protection against XSS attacks

XSS attacks allow hackers to execute malicious scripts on websites by injecting the malicious scripts into the website's body itself. To ensure protection against XSS attacks before the website went live the author would need to enable Content Security Policy, which would disable in-line scripts and scripts not running on the author's domain. The author would need to transfer future in-line scripts into separate .js files. [2]

Validate on browser and server side

To ensure maximum security, validation on both client side and server side ensures that the user experience remains efficient and that the browser itself can make decisions based on what data is invalid before executing, but additional validation on the server side can ensure that no validation policies are exploited by the user either by accident or on purpose, such as inputting a NULL value or entering data that is too long for the field on the database. The author would need to use JavaScript and validate more inputs with both PHP and JavaScript [3]

Better passwords for the admin section

The author would be required to set the password for the Root admin on the admin section of the page to something a lot more complex, and enforce much stricter password regulations. The author would need to redesign the entire "admin" table on the database with server-side encryption (i.e. convert all "password" fields into encrypted ciphers, then decrypt said passwords using PHP). This will ensure that if the admin section of the database were to be leaked, the passwords will be unfunctional, and the hacker cannot use these details to log-in to an admin account with malicious intent. [4]

HTTPS everywhere on website

If this website were to go live, an SSL certificate will need to be purchased, and the website itself will need to be hosted in HTTPS. Although this website in particular isn't handling sensitive user information, HTTPS can ensure protection against intruders injecting or tampering with your website's code or appearance, which includes injecting ads that may break user experiences and could cause vulnerability in security. [5]

The author can purchase and setup an SSL certificate and transfer the website into HTTPS. A green lock icon next to the URL will ensure the user that the website itself is secure and safe.

More protection against SQL injections

The best way to protect the website from the many types of SQL injection attacks is by using a web application firewall. A WAF uses a set of rules to determine whether any requests sent server-side are malicious or destructive, and ultimately rules them out. The WAF especially concentrates on GET and POST requests, where SQL injections occur the most. The author will need to install, maintain and configure a WAF program on their backend.[6]

References

- [1] "SQL DROP TABLE Statement", *W3schools.com*, 2021. [Online]. Available: https://www.w3schools.com/sql/sql_drop_table.asp. [Accessed: 16- Jan- 2021]
- [2] "9 security tips to protect your website from hackers", *Creative Bloq*, 2021. [Online]. Available: <https://www.creativebloq.com/web-design/website-security-tips-protect-your-site-7122853>. [Accessed: 16- Jan- 2021]
- [3] "Input Validation on Client-side or Server-side?", *Packetlabs*. [Online]. Available: <https://www.packetlabs.net/input-validation/>. [Accessed: 16- Jan- 2021]
- [4] "The Basics of Web Application Security", *martinfowler.com*, 2017. [Online]. Available: <https://martinfowler.com/articles/web-security-basics.html#HashAndSaltYourUsersPasswords>. [Accessed: 16- Jan- 2021]
- [5] "Why HTTPS matters", *web.dev*, 2015. [Online]. Available: <https://web.dev/why-https-matters/>. [Accessed: 17- Jan- 2021]
- [6] "How to prevent SQL injection attacks", *Ptsecurity.com*, 2019. [Online]. Available: <https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sql-injection-attacks/>. [Accessed: 17- Jan- 2021]