Edward Giles - Criterion C – 993 words

A number of advanced techniques are used in this system, to ensure security, maintainability, and data safety:

1. Database normalisation and foreign-key constraints
2. Hashing passwords before they are stored in the database
3. MySQL prepared statements, to avoid SQL injection attacks
4. QR codes for communication with mobile devices – QR-code library
5. PHP sessions to store user details as they navigate the site

Database Normalisation and Foreign-Key Constraints

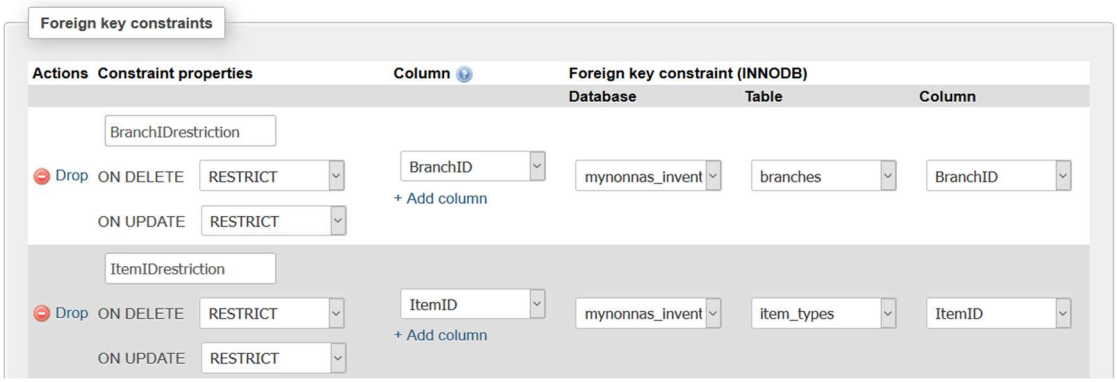
The system's database is normalised so that every piece of data is stored in exactly one location, and SQL JOINs are used to allow all of the relevant data to be retrieved if it is spread across multiple tables. In this system, the below SQL code is used to fetch the data to be displayed on the order-report page. One of the JOIN clauses is highlighted:

```
SELECT `requests`.`BranchID`, `item_types`.`CategoryID`,
`item_types`.`SupplierID`, `item_types`.`DisplayName` FROM `requests`
INNER JOIN `branches` ON `requests`.`BranchID`=`branches`.`BranchID`
INNER JOIN `item_types` ON `requests`.`ItemID`=`item_types`.`ItemID`
WHERE `branches`.`OrdererName` = ?
AND `requests`.`Date` = CURRENT_DATE()
AND NOT `item_types`.`Special`
```

In the `requests` table alone, only the internal stall identifier (`requests`.`BranchID`) is available for the stall that made the request, but the highlighted JOIN clause causes the database server to fetch the stall's details from the `branches` table, giving the query indirect access to the `OrdererName` for that stall, that is, the person responsible for acting on the stall's requests.

The alternative to using a JOIN is to store the `OrdererName` in the `requests` table instead of a separate table. However, one stall can make multiple requests, but each stall only has one `OrdererName`. This strategy would lead to either redundancy (`OrdererName` stored multiple times for the same stall) or data conflicts (two requests from the same stall have different names).

Foreign-key constraints are also used to ensure the data is valid. Wikipedia's definition of a foreign key is a 'field (or collection of fields) in one table that uniquely identifies a row of another table or the same table.' Foreign-key constraints enforce this relationship by making sure that rows are never removed if foreign keys reference them, and stopping rows from being added if they contain bad references.



The first constraint ensures that the values in the `BranchID` column of the `requests` table always match with one of the rows in the `branches` table. This prevents requests from referring to non-existent stalls. The second constraint ensures that requests are always for a valid item, in a similar way.

Password Hashing

To ensure that the user's password is kept safe, passwords are not directly stored in the database. Instead, the password is 'hashed' (processed with a one-way function) so that the password cannot be read directly.

Username	PasswordHash
testorderer1	$2y$10$psdAT.G8me.JwuyUc.N05uu6GSsL8lZYwe7PhNPuSBY...
testorderer2	$2y$10$0q5Tocj.ZLiBhS5XsyQpKuqjjFLKGZ/c9WqZPW8XmPN...
testorderer3	$2y$10$JjoBwpW8c1dl6SuCyvPnCebdFoSdIL.9OxKfhiXerZ3...
testorderer4	$2y$10$ATjLDcr7l1htJaSF7SX3EOpuH2gmpxF9sSukzp/hWBp...

A secure feature of the `password_hash()` function built into PHP is that it generates a random value ('salt') that is incorporated into the hash computation and stored alongside the password hash. This prevents an attacker from seeing whether two users share a password, among other security benefits. When a user attempts to sign in or change their password, their `PasswordHash` is fetched from the database, and the password they entered is verified using the built-in `password_verify()` function:

```
if ($stmt = $conn->prepare('SELECT `PasswordHash` FROM `admin_users` WHERE `Username` = ?')) {
    $stmt->bind_param('s', $username);
    $stmt->execute();
    $stmt->bind_result($server_hash);
    if ($stmt->fetch() === true) {
      if (password_verify($currentPassword, $server_hash)) {
            $correct = true;
      }
    }
    $stmt->close();
}
```

The call to `password_verify()` above does the following:

- Extracts the random salt value stored in `$server_hash`,
- Computes the hash of `$currentPassword`, using that salt, and
- Checks whether the calculated hash matches `$server_hash`.

Prepared Statements

In PHP, the simplest and least secure way to access a database is to use a call to the `query()` method of a database connection (in this case `$conn`). For example, the following code could be used to change passwords, although a more secure solution was actually used:

```
$success = $conn->query("UPDATE `admin_users` SET `PasswordHash` = '" +
$hashedPassword + "' WHERE `Username` = '" + $username + "'");
```

This is susceptible to SQL injection attacks: If data in `$username` is interpreted as SQL commands rather than data, giving anyone who controls the value of `$username` (an attacker) direct access to the database.

In order to prevent this sort of attack from being possible, `prepare()` is used instead of `query()` to create a 'prepared statement' with placeholders that data is inserted into later. Data is inserted using `bind_param()`, which treats the data specially, so it is never interpreted as part of the query. In the version of SQL I am using, a placeholder is denoted with a question mark:

```
if ($stmt = $conn->prepare('UPDATE `admin_users` SET `PasswordHash` = ?
WHERE `Username` = ?')) {
    $stmt->bind_param('ss', $hashedPassword, $username);
```

QR Codes – External Library

The way a stock-taker interacts with the system is by using their mobile device to scan a QR code containing a link to a landing page. This page, when requested from the server, will cause a request to be added to the requests table of the database.

In order to generate QR codes, the system uses a free library called phpqrcode. This library contains a set of functions for generating QR codes with almost any content, in a variety of sizes, and encoding the result in PNG format. It also allows the selection of the amount of error correction that can be embedded in the QR code. The library is invoked in barcode.php:

```
// Force downloading of the QR-code image
$ref_URL =
relToAbs("inspection_landing.php?id=$itemID&inputPIN=$inputPIN");
// Use the phpqrcode library to generate and output the barcode
// http://phpqrcode.sourceforge.net/
include 'assets/phpqrcode/qrlib.php';
header("Content-Disposition: attachment;
filename=\"mynonnas_qr$itemID.png\"");
QRcode::png($ref_URL, false, QR_ECLEVEL_M, $QR_pixel_size);
```

PHP Sessions

PHP has a built-in 'sessions' feature, which allows data to be made persistent across web-page accesses. This is used in my program in order to allow users to sign in once and remain signed in for as long as they use the system, or for a configurable length of time. The function that begins a session is called `session_start()`, and it operates by attempting to read a 'session ID' from a cookie stored on the user's computer (called the session cookie). If there is no session ID, a new one is randomly generated and stored in a new session cookie, with a name that is configurable by `session_name()`. When the script finishes, the information in the session is stored in a file on the server that corresponds to the session ID. The code that manages sessions is stored in session.php:

```
// Read the configuration file
require 'config.php';
// Make sure the name stored in $SID_cookie (from config.php) is used for
the session cookie
session_name($SID_cookie);
// Start or continue the session
session_start();
```

In order to prevent the repetition of code throughout the project, this script is included (using `include/require` statements) by many other parts of the system.

The data in a session can be accessed through a special variable called $_SESSION, which is a collection of key-value pairs. It is used in this system to store the name of the current user, and the permissions they have. The script signin_accept.php loads this data into the session when the user logs in:

```
$_SESSION['username'] = $username;
foreach ($perms as $permName => $permValue) {
      $_SESSION['perm_' . $permName] = $permValue;
}
```