## Written Test

### Question 1

We have taken over the management of a web application written in ASP.NET C# MVC. The application requires a username and password to login, and the url to access the system is:

<http://made-up-application.com/>

When a user logs into the application they can view their existing details that are stored in the database by going to the following url:

<http://made-up-application.com/customer/details?customerid=1001>

Here is all of the code used in the details action:

public ActionResult details()

{

string database = "Data Source=localhost;Initial Catalog=CustomerDatabase;User Id=sa;Password=password";

string id = Request.QueryString["customerid"];

SqlConnection connection = new SqlConnection(database);

connection.Open();

string query = "SELECT \* FROM Customers WHERE ID = " + id;

SqlCommand cmd = new SqlCommand();

cmd.Connection = connection;

cmd.CommandText = query;

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

ViewBag.Firstname = (string)reader[0];

ViewBag.Surname = (string)reader[1];

ViewBag.Email = (string)reader[2];

ViewBag.Telephone = (string)reader[3];

}

return View();

}

Please list any improvements we think can be made to the above code.

**Answer** :

The given code snippet is vulnerable to several security and best practice issues. Here are the improvements that can be made to enhance security and overall code quality:

1. \*\*SQL Injection vulnerability\*\*: The code directly concatenates the `id` parameter into the SQL query, which makes it susceptible to SQL injection attacks. To prevent this, we should use parameterized queries or stored procedures, which can help mitigate SQL injection risks. Here's an example using a parameterized query:

string query = "SELECT \* FROM Customers WHERE ID = @id";

SqlCommand cmd = new SqlCommand(query, connection);

cmd.Parameters.AddWithValue("@id", id);

2. \*\*Connection string security\*\*: Storing the database connection string directly in the code is not recommended. Instead, we should store it securely, such as using configuration files or environment variables. This helps protect sensitive information, such as the database credentials, from being exposed in the source code repository.

3. \*\*Handling database connections\*\*: It is best practice to properly close database connections after use. In the given code, the connection is not explicitly closed, which can lead to resource leaks. To ensure proper cleanup, use the `using` statement, which automatically disposes of the connection and releases resources:

using (SqlConnection connection = new SqlConnection(database))

{

connection.Open();

// Rest of the code goes here

} // The connection will be automatically closed and disposed of here

4. \*\*Error handling\*\*: The code does not have any error handling mechanisms, which can lead to unhandled exceptions or unclear error messages being displayed to users. Implement proper error handling, such as using try-catch blocks, to handle exceptions gracefully and provide meaningful error messages.

5. \*\*Model-View-Controller (MVC) pattern\*\*: The given code violates the MVC pattern. The details of the customer should ideally be retrieved in the model or a separate data access layer, rather than directly in the controller. Following the MVC pattern improves code organization, separation of concerns, and maintainability.

6. \*\*Input validation\*\*: The code does not perform any input validation on the `customerid` parameter. It is essential to validate and sanitize user inputs to prevent security vulnerabilities, such as code injection attacks or unexpected behavior due to malformed inputs. Ensure we validate and sanitize inputs, ensuring they meet the expected criteria.

By implementing the above improvements, we can enhance the security, maintainability, and overall code quality of the application.

### Question 2

Given a base URL of <https://made-up-api.com/> for a set of RESTful web services, what URL and HTTP verb might we expect to use to carry out each of the following actions:

* Get a list of all customers.
* Get the details of order number 25.
* Delete the customer with an ID of 133.
* Update a single field for all orders.
* Update all fields for the customer with an ID of 1701.

## **Answer :**

Based on the base URL of `https://made-up-api.com/`, here are the expected URLs and HTTP verbs for each of the given actions:

• Get a list of all customers:

URL: https://made-up-api.com/customers

HTTP Verb: GET

• Get the details of order number 25:

URL: https://made-up-api.com/orders/25

HTTP Verb: GET

• Delete the customer with an ID of 133:

URL: https://made-up-api.com/customers/133

HTTP Verb: DELETE

• Update a single field for all orders:

URL: https://made-up-api.com/orders

HTTP Verb: PATCH or PUT

The choice between PATCH and PUT depends on the desired behavior and the API implementation. PATCH is typically used to update specific fields, while PUT is used to replace the entire resource.

• Update all fields for the customer with an ID of 1701:

URL: https://made-up-api.com/customers/1701

HTTP Verb: PUT

In the case of updating all fields, it is common to use the PUT method to replace the entire resource with the updated data.

## Practical Test

### Task 1

We should have been sent an accompanying file **students-and-subjects.csv**. This includes example data for a group of 20 fictional students.

For each student there is a record of an id number, first name, surname, title, e-mail address, date of birth, and up to 2 subjects that the student is studying.

Using this csv please design a relational database that would be appropriate for storing all the details in the csv file.

Please provide a database diagram showing the tables we would use in the database, and the relationships between the tables, and accompanying SQL commands that we would use to create the tables and their relationships.

Based on the provided CSV data, I would design a relational database with the following tables: `Students`, `Subjects`, and `Student\_Subjects`. Here's a database diagram showing the tables and their relationships:

|  |  |  |
| --- | --- | --- |
| Students | Subjects | Student\_Subject |
| id | subject\_id | student\_id |
| first\_name | subject\_name | Subject\_name |
| last\_name |  |  |
| title |  |  |
| email |  |  |
| dob |  |  |

The `Students` table stores information about each student, including their ID, first name, last name, title, email, and date of birth. The `Subjects` table contains information about the subjects, with a unique subject ID and subject name. The `Student\_Subjects` table establishes a many-to-many relationship between students and subjects, storing the student ID and subject name for each student-subject combination.

To create these tables and their relationships, you can use the following SQL commands:

-- Create Students table

CREATE TABLE Students (

id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

title VARCHAR(10),

email VARCHAR(100),

dob DATE

);

-- Create Subjects table

CREATE TABLE Subjects (

subject\_id INT PRIMARY KEY,

subject\_name VARCHAR(50)

);

-- Create Student\_Subjects table

CREATE TABLE Student\_Subjects (

student\_id INT,

subject\_name VARCHAR(50),

FOREIGN KEY (student\_id) REFERENCES Students(id),

FOREIGN KEY (subject\_name) REFERENCES Subjects(subject\_name),

PRIMARY KEY (student\_id, subject\_name)

);

These SQL commands will create the necessary tables with the appropriate columns and primary keys. Additionally, the `Student\_Subjects` table includes foreign key constraints to maintain referential integrity between the `Students` and `Subjects` tables.

After creating the tables, you can use SQL INSERT statements to populate them with data from the CSV file.

Note: It's important to consider data types and sizes that match your specific requirements when creating the tables. The example provided assumes basic data types and sizes, but you may need to adjust them based on your application's needs.

### Task 2

Please create a web page that displays a list of all students in the attached spreadsheet. This page needs to display the following information about each student - id, first name, surname, title, and e-mail address.

This web page does not have to connect to a data source, and we can just use the static data provided if we wish.

Please use bootstrap v5.3 (https://getbootstrap.com/) to help we build this web page.

### Task 3

Please create a web page for creating a new student account. This page needs to provide input fields for:

* first name (maximum 25 characters)
* surname (maximum 25 characters)
* title
* e-mail address (maximum 50 characters)
* date of birth (date picker)
* Up to 2 subjects

This web page does not have to connect to a data source to save the data and can just be a web page to allow data to be input and provide client-side validation.

Please use bootstrap v5.3 (https://getbootstrap.com/) to help we build this web page.

### Task 4

The following JSON feed is used to lists courses at the University where students can study some of the course at a partner institute:

<https://www.liverpool.ac.uk/app-data/study-abroad/courses.json>

Please create a web page to list all the courses that are available. Each course name should be a link that when clicked on will then provide a list of the names and countries of partner universities where the course can be studied.

Please use bootstrap v5.3 (https://getbootstrap.com/) to help we build these web pages.