First database security steps like [input validation](https://www.esecurityplanet.com/trends/security-threats-in-html5/), sanitization, prepared statements, and stored procedures are essential

**Only accept well formatted email address in form field**

**(Client-Side Validation)**

**☑ an example of validation**

**Don’t Trust Client-Side Validation or User Inputs**

The client side is absolutely not a secure layer to validate forms.

**Allow-list Input Validation……**

The best way to validate email addresses is to perform some basic initial validation, and then pass the address to the mail server and catch the exception if it rejects it.

Spaces are allowed if they are enclosed in quotes – Exploit!!

**Client-side databases allows for many safety and privacy vulnerabilities, its better to implement securer practices instead.**

**Using Cookies to validate input on the client-side**

**(Client-side)**

Cookies can be easily modified, added and deleted by users and should be treated as untrusted user input.

They are just as prone to XSS and SQL injection vulnerabilities as any other user input. “Cookie poisoning attacks”!!

Further, unless you're using SSL, cookies are just as prone to sniffing as GET or POST data in a request.

**IndexedDB(Client-side)**

A large-scale, NoSQL storage system. It lets you store just about anything in the user's browser which is not encrypted and is open to XSS attacks.(Client-side)

IndexedDB may be insecure since security was not considered in its specification.

While using IndexedDB properly can dramatically improve user experience, using it incorrectly or failing to handle error cases can lead to broken apps and unhappy users.

[A vulnerability in the IndexedDB API of the Safari 15 browser](https://fingerprintjs.com/blog/indexeddb-api-browser-vulnerability-safari-15/), disclosed by the team at FingerprintJS

Since client storage involves many factors outside of your control, it's critical your code is well tested and properly handles errors, even those that may initially seem unlikely to occur.

**Server-side databases allow for many safety and privacy features and benefits, it’s better to implement a more secure practice.**

# **Parameterized/Prepared statements in PHP**

# **☑ example of sanitizing**

**In web development to sanitize means that you remove unsafe characters from the input.**

To prevent SQL Injection vulnerabilities in PHP, use PHP Data Objects (PDO) to create parametrized queries (prepared statements).

A prepared statement or a parameterized statement is used to execute the same statement repeatedly but with different specified parameters, creating with high efficiency and protect against SQL injections.

The prepared statement execution consists of two stages: prepare and execute.

SQL Injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces

[**validator.js**](https://github.com/mikeerickson/validatorjs)

**☑ An example of validation**

The validatorjs library simplifies data validation in JavaScript.

Basic input validation put in place with validatorjs.

**Service** **Workers**

Service workers only run over HTTPS, for security reasons.

Most significantly, HTTP connections are susceptible to malicious code injection by [man in the middle](https://developer.mozilla.org/en-US/docs/Glossary/MitM) attacks, and such attacks could be worse if allowed access to these powerful APIs.

In Firefox, service worker APIs are also hidden and cannot be used when the user is in [private browsing mode](https://support.mozilla.org/en-US/kb/private-browsing-use-firefox-without-history).

They are used to handle network requests, cache assets, and manage push notifications, among other things.

[Why Service Workers are a Threat to Modern Browsers](https://sweetcode.io/why-service-workers-are-a-threat-to-modern-browsers/)

[Abusing the Service Workers API](https://www.akamai.com/blog/security/abusing-the-service-workers-api)

## **How to Prevent an SQL Injection**

The only sure way to prevent SQL Injection attacks is input validation and parametrized queries including prepared statements.

The application code should never use the input directly.

The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes.

It is also a good idea to turn off the visibility of database errors on your production sites.

Database errors can be used with SQL Injection to gain information about your database.

If you discover an SQL Injection vulnerability, for example using an Acunetix scan, you may be unable to fix it immediately.

For example, the vulnerability may be in open source code.

In such cases, you can use a web application firewall to sanitize your input temporarily.

To learn how to prevent SQL Injection attacks in the PHP language,

see: [Preventing SQL Injection Vulnerabilities in PHP Applications and Fixing Them](https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-in-php-applications/).

To find out how to do it in many other different programming languages, refer to the [Bobby Tables guide to preventing SQL Injection](http://bobby-tables.com/).

**Primary Defences:**

* **Use of Prepared Statements (with Parameterized Queries)**
* **Use of Properly Constructed Stored Procedures**
* **Allow-list Input Validation**
* **Escaping All User Supplied Input**

**Additional Defences:**

* **Enforcing Least Privilege**
* **Performing Allow-list Input Validation as a Secondary Defence**

Procedure is a method/way of completing a certain task in line with a policy

**Answers for email**

Only accepting well formatted email address in form field – validation

Using validator.js in JavaScript to validate form data - validation

Prepared statements or parameterised statements in PHP - sanitization

**Questions for Email**

* In the style guide typography - font-size can be identified as ems - is it ok to use rem instead of em?
* Do we design a logo?
* For the typography - do we have to use whats in the style guide

e.g Title – 20px, heading 1 - 18px , heading 2 16px or do we create our own?