

# Security

---

Validation and sanitization

# Input Validation

---

- In a web app, validation should be carried out on every form element to guarantee that the input is correct.
- Processing incorrect input values can make your application give unpredictable results.
- Risks include
  - SQL Injection
  - Cross-site scripting
  - Buffer overflows
  - Leakage of site internal design through error messages
- Validation for security should always be carried out on the **server** side
  - HTML form attributes and JavaScript validation can be an aid to users but are useless for security

# Whitelists vs Blacklists

---

- **Blacklist validation** is testing an input against a set of unacceptable values
  - Default policy is "accept"
- **Whitelist validation** is testing an input against a set of possible correct values
  - Default policy is "reject"
- Whitelist validation is generally best for security, but tends to be more restrictive and may conflict with user-friendliness
- e.g. EC2 security group *whitelists* specific protocols and ports; all others are blocked

# Validation in Hapi

---

- Modern frameworks have extensive features to support data validation and sanitisation.
- e.g.
  - **joi** for input validation
  - **disinfect** for sanitisation

# Regular Expressions

- `Joi.string().regex()` checks the value provided is a string matching a particular **regular expression**
- Example

```
Joi.string().regex(/^[A-Z][a-zA-Z' -]{3,}$/)
```

[ ] specifies alternative options  
{a,b} at least a; at most b

This pattern checks if the input string starts with an upper case letter and has a minimum of 3 additional characters in total, each of which must be alphabetic, an apostrophe or a hyphen

# Regular Expressions

---

- A **regular expression (regex)** is a sequence of characters that specify a pattern to be matched.
- Very powerful concept as many computing applications involve pattern matching – for example:
  - Search engines
  - Natural (human) language processing
  - Intrusion detection
  - Computer forensics
  - Intelligence gathering (e.g. NSA...)
- A full treatment of regular expressions is beyond the scope of this module
  - Several textbooks just on regular expressions + many online resources

# Regular Expressions

- A useful online regex tester: <https://regex101.com/> (others exist as well)

The screenshot shows the regex101.com website interface. The browser tab is titled "Online regex tester and debug". The address bar shows the URL "https://regex101.com". The website header includes the logo "regular expressions 101" and navigation links for "@regex101", "donate", "contact", "bug reports & feedback", and "wiki".

The main content area is divided into several sections:

- REGULAR EXPRESSION**: The input field contains the regex pattern `[A-Z][a-zA-Z']{3,}`. A status bar indicates "4 matches, 17 steps (~0ms)".
- TEST STRING**: The input field contains the text "Sullivan", "O'Reilly", "McDonald", and "Smith-Garcia". The results show that the regex matches "Sullivan", "O'Reilly", and "Smith-Garcia", but not "McDonald".
- EXPLANATION**: A section for explaining the regex pattern.
- MATCH INFORMATION**: A section for displaying match details.
- QUICK REFERENCE**: A section for quick reference to regex tokens and sequences.

The QUICK REFERENCE section includes a search bar and a list of tokens and sequences:

- All Tokens
- ★ **Common Tokens** ✓
- General Tokens
- ⚓ Anchors
- ⌚ Meta Sequences
- \* Quantifiers
- ⦿ Group Constructs
- ⚫ Character Classes

On the right side of the QUICK REFERENCE section, there is a list of character classes and their descriptions:

- A single c... `[abc]`
- A chara... `^[abc]`
- A charact... `[a-z]`
- A chara... `^[a-z]`
- A cha... `[a-zA-Z]`
- Any single cha... `.`
- Any whitespa... `\s`
- Any non-whit... `\S`
- Any digit `\d`

# RegEx Quick Reference

## Regular Expressions quick reference

[basic](#) | [complete reference](#) | [tips & tricks](#)

.	Any single character	<a href="#">\s</a>	Any whitespace character	<a href="#">( . . . )</a>	Capture everything enclosed
<a href="#">^</a>	Start of string	<a href="#">\S</a>	Any non-whitespace character	<a href="#">(a b)</a>	Match either a or b
<a href="#">\$</a>	End of string	<a href="#">\d</a>	Any digit	<a href="#">a?</a>	Zero or one of a
<a href="#">[abc]</a>	A single character of: a, b or c	<a href="#">\D</a>	Any non-digit	<a href="#">a*</a>	Zero or more of a
<a href="#">[^abc]</a>	A character except: a, b or c	<a href="#">\w</a>	Any word character	<a href="#">a+</a>	One or more of a
<a href="#">[a-z]</a>	A character in the range: a-z	<a href="#">\W</a>	Any non-word character	<a href="#">a{3}</a>	Exactly 3 of a
<a href="#">[^a-z]</a>	A character not in the range: a-z	<a href="#">\b</a>	A word boundary	<a href="#">a{3,}</a>	3 or more of a
<a href="#">[a-zA-Z]</a>	A character in the range: a-z or A-Z	<a href="#">\B</a>	Non-word boundary	<a href="#">a{3,6}</a>	Between 3 and 6 of a



# Sanitization vs Validation

---

- Sanitization tries to achieve similar objectives to validation, but in a somewhat different way
- Output of Validation is usually binary:
  - Valid: input is accepted
  - Invalid: input is not accepted
- Output of Sanitisation is a "cleaned" version of input:
  - Output is filtered input (e.g. certain characters removed or re-encoded)
- Several npm sanitization modules exist - see **disinfect** and **sanitize** for example

# Sanitization examples

Sanitizer	Description	Example input	Example output
Email address	Remove all characters except letters, digits and !#\$%&'*+ -=?^_`{ }~@.[].	<jb@gmail.com> ,	jb@gmail.com
Integer	Remove all characters except digits, + and -	13a	13
HTML entities	Convert reserved characters in HTML to corresponding entities	< > &	&lt; &gt; &amp;
Restrict HTML	Whitelist certain tags: <a href="URL"> <em> <cite> <i> <strong> <b> <sub> <sup> <ul> <ol> <li>	<script>alert(1) </script>	alert(1)