Engineer

validering

å sjekke inndata mot reglene før man stoler på dem

Hva det er

Validering, på engelsk *validation*, er steget der et program holder innkommende data opp mot reglene de skal følge, og sier nei når de ikke holder mål, før noe lagres, kjøres eller sendes videre. Ordet får ulike forledd etter hvilket lag det gjelder: inndatavalidering for skjemafelt og API-forespørsler, skjemavalidering når reglene er skrevet ned som et JSON Schema eller en OpenAPI-beskrivelse, og markup-validering når en nettside sjekkes mot grammatikken i HTML. Inndata som går gjennom, kalles gyldige.

Veiledningen fra OWASP deler arbeidet i to spørsmål. Det syntaktiske spørsmålet er om en verdi har riktig type og format: er dette en dato, er dette et heltall, lar innholdet i forespørselen seg tolke som JSON? Det semantiske spørsmålet er om verdien gir mening for den operasjonen det gjelder. En hotellbestilling trenger to gyldige datoer, og i tillegg en sluttdato som ligger etter startdatoen. Å gjøre strengen «500» om til et heltall avgjør det første spørsmålet og sier ingenting om hvorvidt 500 netter er et akseptabelt opphold.

Markup-validering er den eldste offentlige formen for ideen på nettet. W3C Markup Validation Service beskriver validering som å sjekke et dokument mot grammatikken det sier at det bruker, og sammenligner validatoren sin med lint for C. Dokumentasjonen er nøye med hva det beviser: en gyldig side er ikke nødvendigvis en god side, bare en side et verktøy fant i samsvar med et bestemt sett regler. Den samme forsiktigheten gjelder for alle andre typer. Validering forteller deg at dataene passer reglene du skrev; den forteller deg ikke at det var de riktige reglene.

To nabobegreper blir ofte blandet sammen med den. Sanering endrer inndata for å gjøre dem akseptable, ved å fjerne eller skrive om deler av dem, mens validering godtar eller avviser dem slik de er. Koding og parameterisering gjør

data trygge for stedet de skal til, en HTML-side eller en SQL-spørring, og de trengs fortsatt etter at valideringen har gått gjennom.

Bruk begrepet i en beslutning

Bruk «Hva er validering?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Regler og kontroller som fanger opp dårlige data på vei inn, og et klart svar på hvem som eier hvilket tall og hvor det kommer fra.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Én infrastrukturplan for 14 avdelinger — 14 forskningsavdelinger på én felles infrastrukturplan

Én infrastrukturplan for 14 avdelinger Ta kontakt →

Hvorfor det er viktig

Beslutningen bak validering er hvor et system trekker tillitsgrensen sin, og hva det sjekker der. Hver forespørsel fra en nettleser, hver melding fra en kø, hver fil fra en partner og hver endring i konfigurasjonen fra en kollega kommer med antakelser på kjøpet. Sjekkes de antakelsene ved kanten, kan resten av koden bygge på dem. Hoppes sjekken over, må hver funksjon lenger inn forsvare seg selv, og i praksis gjør de fleste av dem ikke det.

Den andre beslutningen er hvordan reglene skrives. Sjekker skrevet for hånd, endepunkt for endepunkt, er billige i starten og glir fra hverandre etter hvert som systemet vokser, fordi hver av dem er sin egen lille kodebit. Regler som deklareres én gang, som typer eller som et skjema, kan håndheves av et bibliotek og publiseres som dokumentasjon fra den samme kilden, noe som reduserer risikoen for at beskrivelsen og oppførselen glir fra hverandre. Prisen er et rammeverk å lære og standardverdier å forstå, for et skjema håndhever nøyaktig det det sier, og ingenting av det det lar være usagt.

Strenghet er det siste valget. En tillatelsesliste (*allowlist*) beskriver hva som er akseptabelt og avviser alt annet, og OWASP anbefaler det framfor å prøve å kjenne igjen hver ondsinnet streng. Å avvise felt som skjemaet ikke kjenner, fanger skrivefeil og utdaterte klienter tidlig. Å avvise hele forespørselen ved det første problemet, med en tydelig feilmelding, er tryggere enn å fortsette med data som bare delvis er sjekket.

Slik ser det ut i praksis

Et API startet med forespørselsvalidering skrevet for hånd for hvert endepunkt. På det dokumenterte implementeringstrinnet gjorde flyttingen til Huma v2 sjekkene om til typer på tvers av 649 stier og 760 operasjoner. Hvert endepunkt hadde egne struct-typer for inndata og utdata, Huma genererte forespørselsvalideringen fra dem, og en OpenAPI-beskrivelse kom fra de samme typene. To WebSocket-endepunkter sto igjen på ren Fiber, der forespørsel/svar-modellen ikke passet. En sjekk i CI sammenlignet rutene serveren registrerte, med rutene beskrivelsen oppga. Den dokumenterte ruteparitet var 100 %; sjekker deklarert i inndatatypene erstattet den tilsvarende håndskrevne valideringen.

Konfigurasjon trenger den samme behandlingen, og SSH-herdingen på selskapets vert kjører nå validering i tre trinn ved hver kjøring. Kandidatfilen syntakssjekkes før den installeres, den sammensatte konfigurasjonen sjekkes etter installasjonen, og deretter leses daemonens egne effektive innstillinger tilbake og 24 direktiver sjekkes mot dem. Det tredje trinnet finnes fordi en tilgangsrapport i september 2026 fant to innstillinger som automatiseringen hadde skrevet ved hver kjøring, og som daemonen hadde overstyrt. Operativsystemets egne standardverdier ble lest først, og for de to innstillingene er det den første forekomsten som vinner. Som siden sier det: et søk med grep etter et direktiv beviser bare at det ble skrevet.

Markup-validering var del av det dokumenterte implementeringstrinnet i august 2026: HTML-validering var én av sjekkene i en commit-port med 22 lintere uten advarselsnivå, der hver diagnostikk var en feil og inline-undertrykking var forbudt, så en ugyldig side ikke kunne committes ved å merke feilen som akseptabel.

Hvor det går galt

Serveren stoler på det nettleseren allerede har sjekket. Et skjema som avviser en ugyldig e-postadresse i nettleseren, ser validert ut, men forespørselen kan sendes helt uten skjemaet. OWASP regner sjekker som bare skjer hos klienten, som en vanlig felle: valider på serveren, og behandle sjekken i nettleseren som et raskt hint til den som skriver.

Et skjema godtar alt det ikke nevner. I JSON Schema gjør det å liste opp en egenskap den ikke påkrevd, og det forbyr heller ikke andre felt, og med standardvokabularet i 2020-12 blir nøkkelordet `format` samlet inn som en annotasjon, med formatsjekk avslått som standard. Dette skjemaet ser strengt ut:

```
{
    "type": "object",
    "properties": {
        "email": { "type": "string", "format": "email" }
    }
}
```

Det godtar `{}` , `{"email": "nope"}` og `{"emial": "a@example.com"}` . Legg til `required` og `"additionalProperties": false` , og slå på formatsjekk (*format assertion*) i validatoren, eller sjekk formatet i koden. Huma på sin side avviser ukjente felt i forespørselsinnholdet som standard, dokumenterer `readOnly` og `writeOnly` uten å håndheve dem, og overser ukjente spørringsparametere med mindre den får beskjed om noe annet.

Sjekken godkjenner én versjon av dataene, og programmet bruker en annen. En validator som leser den rå forespørselen mens håndtereren leser en dekodet kopi, eller en test som ser på filen på disk mens daemonen bruker en annen innstilling, har sjekket noe systemet aldri bruker. Rådet fra OWASP er å validere den representasjonen som faktisk skal brukes. SSH-arbeidet over er den samme lærdommen i konfigurasjon: bare tilbakelesingen fra daemonen viste hva som faktisk gjaldt.

En sperreliste møter neste variant. Å blokkere kjente farlige tegn avviser legitime inndata, f.eks. en apostrof i et etternavn, og gjør likevel ikke spørringen trygg. Definer hva hvert felt godtar, og avvis resten.

Gyldige inndata behandles som trygge inndata. En streng som har passert hver regel, kan fortsatt ødelegge en SQL-setning eller en side når den limes inn i en. Validering snevrer inn hva som kommer inn; parameteriserte spørringer og koding av utdata beskytter stedet der det havner, og begge deler trengs.

Les mer

- OWASP Input Validation Cheat Sheet — syntaks og semantikk, tillatelseslister, grenser for tolking og vanlige feller.
- W3C Markup Validator documentation — validatorens egen forklaring av markup-validering, med hjelpesider om hva gyldighet beviser, og hva den ikke beviser.
- JSON Schema Validation, draft 2020-12 — nøkkelordene for validering, også hvordan `format` oppfører seg som standard.
- Huma: Request Validation — validering generert fra struct-tagger i Go, med innstillingene for strenge og løse felt.
- OpenAPI Specification — beskrivelsesformatet hvis skjema mange API-validatorer håndhever.

Ordliste

Velg ut fra deres faktiske rammer

Ta med et representativt datasett, spørsmålene det skal besvare og dagens flaskehals. Vi kan avtale målinger av kvalitet, gjennomstrømning og gjenoppretting.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er validering?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/validation/