PROJECT ABSTRACT

JSON Schema Validator and Dynamic Form Generator

A client-side JSON Schema toolkit: validate any JSON document against a draft-07-subset schema with path-level error reporting, and render a live HTML form from the same schema. Delivered built-to-order with the web app, report, PPT and viva kit.

01. Title

JSON Schema Validator and Dynamic Form Generator

02. Introduction / Background

JSON is the lingua franca of web APIs, and JSON Schema is how teams contract what a payload must look like — required fields, types, ranges, formats. Students hand-validate payloads by eye and discover schema violations only when an API rejects them. Worse, the same schema that validates data can also generate the form that collects it, a connection rarely taught. This project builds SchemaForge: a client-side validator implementing a JSON Schema draft-07 subset (types, required, properties, items, enum, ranges, patterns, email format, additionalProperties) with precise error paths, plus a form generator that turns any schema into a working HTML form with submit-to-JSON.

03. Problem Statement

Validation is taught as an abstract spec while forms are built by hand, so students never see that one schema can do both jobs. The project addresses this with a single tool where the student writes a schema once, validates documents against it with exact error locations, and watches the same schema render as a form — making the schema the single source of truth, as in professional API design.

04. Objectives

- Validate JSON documents against schemas with path-precise errors (e.g. $.cgpa: above maximum 10).
- Support the practical draft-07 subset: type, required, properties, items, enum, minimum/maximum, minLength/maxLength, pattern, format email, additionalProperties.
- Generate a live form from any schema: enums become dropdowns, booleans checkboxes, nested objects fieldsets, arrays repeatable rows.
- Provide sample schema/document pairs including an intentionally-invalid demo showing error reporting.
- Add a prettify helper and submit-to-JSON on generated forms.
- Deliver the complete student kit: web app, report, PPT and viva Q&A.;

05. Proposed Methodology

Two engines share one schema parser. (1) Validator: a recursive function walks the document and schema in parallel, accumulating {path, message} errors for type mismatches, missing required keys, range/length violations, pattern and email checks, enum membership and forbidden extra properties. (2) Form generator: a recursive renderer maps schema types to inputs — string to text (with minlength/pattern attributes),

number to number inputs with min/max, enum to select, boolean to checkbox, object to fieldset, array to a nested row. Submit serializes the form back to JSON. Views switch via URL hash.

06. System Architecture / Working

On Validate, both panes are JSON-parsed (parse errors reported with the engine message); the recursive validator then checks the document, collecting every violation with its JSON path. A green banner means conformance; a red banner lists each error card. On Generate, the schema is parsed and the renderer builds the form DOM recursively, marking required fields with an asterisk and wiring native HTML validation attributes from the schema constraints. Submit gathers named inputs back into a JSON object shown pretty-printed — demonstrating the round trip from schema to form to data.

07. Hardware / Software Requirements

- Any modern web browser — no installation, no server
- Single HTML/CSS/JS file (vanilla JavaScript, no dependencies)
- Text editor for code walkthrough (VS Code recommended)
- Report: LibreOffice/MS Word; presentation: PowerPoint-compatible slides

08. Expected Outcomes

- Correct validation of documents against schemas with exact, readable error paths.
- Working forms generated from schemas, including nested objects and enums.
- Demonstrated round trip: schema → form → submitted JSON that validates.
- A complete report, PPT and viva Q&A; on JSON Schema, validation and schema-driven UI.

09. Applications

- Designing and testing API request/response contracts in web projects.
- Generating admin/config forms from schemas instead of hand-coding them.
- Teaching data validation as a first-class design activity.

10. Future Scope

- Full draft-07 support: $ref resolution, oneOf/anyOf/allOf, conditional if/then.
- Schema inference: generate a schema from sample JSON documents.
- Export generated forms as standalone HTML files.