Design document

# Context

# Каріна Корчова — solo developer.

# Goals and non-goals

Goals:

* Розробити Python-бібліотеку configlib, яка:
* зчитує YAML або JSON конфігураційні файли;
* перевіряє відповідність вмісту схемі (JSON Schema);
* обробляє помилки;
* легко тестується.

Non-goals:

* Підтримка TOML, INI та інших форматів
* Автоматичне завантаження схеми з API
* UI або CLI-інтерфейс

# Subsystems

#### Архітектура:

* parser.py — читає YAML або JSON файл, повертає dict
* schema.py — представляє JSON-схему (може бути dict або окремий клас)
* validator.py — перевіряє, що дані відповідають схемі
* tests/ — unit-тести для кожного модуля
* CI — GitHub Actions запускає тести при push/pull request

Взаємозв'язки:

(parser.py) --> dict --> (validator.py) <== schema from (schema.py)

↓

повертає помилки / ОК

Зв’язування:

* Функції взаємодіють напряму, без dependency injection.
* Вся логіка внутрішня, без сторонніх API.

## Data storage subsystem

## Використання: немає БД. Всі конфіги читаються напряму з файлів.

## Типи даних: JSON / YAML → Python dict

## Схема: теж зберігається в JSON-файлі

Формат на диску:

config.yaml:

host: localhost

port: 8080

schema.json:

{

"type": "object",

"properties": {

"host": {"type": "string"},

"port": {"type": "integer"}

},

"required": ["host"]

}

## Business logic

* load\_config(filepath) — повертає dict
* load\_schema(schema\_path) — читає schema.json
* validate\_config(config, schema) — виконує перевірку через jsonschema.validate

У випадку помилки — викидає ValidationError з описом.

# Alternatives considered

* **Ідея:** написати власний валідатор без jsonschema.  
  **Причина відмови:** дуже складно для навчального проєкту; jsonschema є стабільним стандартом.
* **Ідея:** використовувати тільки JSON, без YAML.  
  **Причина відмови:** YAML зручніший для людей, частіше використовується в DevOps.