Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики Кафедра інформатики

Магістерська робота

освітній ступінь – магістр

на тему **«Розробка бібліотеки для визначення структури та валідацію JSON-документу»**

Виконав: студент 2-го року навчання
Спеціальності
121 Інженерія програмного забезпечення

Бойко Данило Романович

Керівник: Нагірна А. М.,
кандидат фіз.-мат. наук, доцент

Рецензент: Афонін А. О.,
кандидат фіз.-мат. наук, доцент

Магістерська робота захищена
з оцінкою _____

Секретар ЕК _____
«____» _____ 20____ р.

Київ 2024

Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри інформатики,
к.ф-м.н., доц. Гороховський С. С.
_____
(підпис)
«____»_____2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на магістерську роботу
студенту 2-го курсу, факультету інформатики__
Бойко Данилу Романовичу
Дослідити та порівняти існуючі рішення для роботи з JSON-документом та реалізувати бібліотеку для валідації JSON-документа
Зміст ТЧ до магістерської роботи:

Зміст
Анотація
Вступ
1 Аналіз предметної області. Постановка завдання магістерської роботи
2 Теоретичні відомості
3 Розробка бібліотеки для валідації JSON-документа
Висновки
Список використаних джерел

Дата видачі „___” _____ 2023 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Графік підготовки магістерської роботи до захисту

Графік узгоджено **«_____» _____** 2023 р.

№ з/п	Перелік робіт	Термін	Підпис	Дата	Примітка
1.	Отримання теми магістерської роботи	16.10.2023			
2.	Ознайомлення з існуючою інформацією за темою магістерської роботи	20.10.2023			
3.	Розробка плану та структури роботи	01.11.2023			
4.	Ознайомлення з існуючими рішеннями для роботи з JSON-документом	03.11.2023			
5.	Робота з дослідженням про JSON-документ	01.01.2024			
6.	Початок створення власного рішення, валідації JSON-документа	15.01.2024			
7.	Закінчення створення рішення для валідації JSON-документом	01.04.2024			
8.	Початок написання текстової частини	20.04.2024			
9.	Подання проміжної версії текстової частини	10.05.2024			
10.	Остаточне завершення написання текстової частини роботи	15.05.2024			
11.	Створення презентації	20.05.2024			
12.	Захист магістерська роботи	11.06.2024			

Зміст

АНОТАЦІЯ

Робота присвячена розробці бібліотеки для мови Golang. Основна задача бібліотеки – це валідація JSON об’єктів за допомогою JSON-Schema. Бібліотека буде підтримувати всі формати даних, які визначені в JSON-Schema. Буде використано виключно стандарту бібліотеку Golang.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – прикладний програмний інтерфейс;

Бекенд (back-end) – серверна частина програмного забезпечення;

Бекенд-розробник – розробник серверної частини програмного забезпечення;

HTML (HyperText Markup Language) – мова розмітки гіпертексту;

CSS (Cascading Style Sheets) - спеціальна мова стилю сторінок;

HTTP (Hypertext Transfer Protocol) – протокол передачі гіпертексту;

HTTPS (Hypertext Transfer Protocol Secure) – захищений протокол передачі гіпертексту;

JSON (JavaScript Object Notation) – текстовий формат обміну даними;

XML (Extensible Markup Language) – стандарт побудови мов розмітки ієрархічно структурованих даних;

XSD – XML Schema Definition;

DOM (Document Object Model) - специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами;

ПЗ – програмне забезпечення;

REST API (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленого застосунку;

URL (Uniform Resource Locator) – уніфікований вказівник ресурсу;

Фронтенд (front-end) – клієнтська частина програмного забезпечення;

Фронтенд-розробник – розробник клієнтської частини програмного забезпечення;

ВСТУП

Актуальність. Одна з переваг XML [1] над JSON [2] – це наявність валідації на рівні опису даних (XSD [3]). JSON [2] в свою чергу має JSON-документ [4], проте він не розповсюджений достатньо серед користувачів. Більшість просто не знає про його існування, відсутні відповідні рішення для певної мови програмування для роботи з JSON-документом, або такі рішення є застарілими та не актуальним.

Мета і завдання дослідження. Розробка бібліотеки для мови програмування Golang [5] для валідації JSON-документів. Для цього необхідно здійснити аналіз теоретичних відомостей, проаналізувати вже існуючі підходи та методи реалізації таких рішень, знайти інформацію щодо стандарту JSON-документу, розробити бібліотеку для мови програмування Golang для валідації JSON-документів.

Об’єкт дослідження. Бібліотека для мови програмування Golang для валідації JSON-документів.

Предмет дослідження. Використання різних програмних компонентів, характеристика взаємодії їх між собою, розробка бібліотеки що використовує ці рішення є предметом дослідження в цій роботі.

Джерела дослідження. Для отримання релевантної та актуальної інформації щодо предмету дослідження будуть використані електронні ресурси, до яких входить офіційна технічна документація, інформація з проектів з відкритим вихідним кодом, спеціалізовані форуми, блоги великих компаній, результати практичних експериментів.

Наукова новизна одержаних результатів дослідження полягає у реалізації бібліотеки для валідації JSON-документу, яка підтримує всі типи та функції валідації, які описані JSON-документом, сумісна з останньою версію мови програмування Golang.

Практичне значення одержаних результатів. Реалізована бібліотека повинна забезпечити зручні умови для користувачів, а саме: підтримувати всі можливі способи визначення схеми для валідації, підтримувати всі можливі способи визначення цілі для валідації та надавати змістовний опис помилки в разі порушення правил валідації JSON-документу. Ці цілі будуть досягнуті за рахунок того, що бібліотека буде приймати будь-який тип (any) як вхідні параметри, валідувати їх, та повертати змістовну помилку, де буде зазначено, що саме було порушено в процесі валідації.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАННЯ МАГІСТЕРСЬКОЇ РОБОТИ

1.1. Аналіз предметної області

JSON-документ [4] – це декларативна мова для анотування та перевірки структури, обмежень і типів даних документів JSON. Він забезпечує спосіб стандартизації та визначення очікувань для даних JSON.

JSON-документ надає розробникам наступні можливості [4]:

1) Опис структурованих даних: схема JSON дозволяє розробникам описувати структуру, обмеження та типи даних існуючих форматів JSON.
2) Визначення та застосування правил. Дотримуючись обмежень схеми JSON, стає легше обмінюватися структурованими даними між програмами, оскільки вони підтримують послідовний шаблон.
3) Розширюваність: схема JSON забезпечує високу адаптивність до потреб розробників. Спеціальні ключові слова, формати та правила перевірки можна створювати для адаптації схем відповідно до конкретних вимог.
4) Перевірка даних: схема JSON забезпечує валідність даних за допомогою:
    a. Автоматичне тестування: перевірка дозволяє автоматизувати тестування, гарантуючи постійну відповідність даних визначеним правилам і обмеженням.
    b. Покращена якість даних: застосовуючи правила перевірки, схема JSON допомагає підтримувати якість даних, наданих клієнтом, зменшуючи невідповідності, помилки та потенційні вразливості безпеки.

1.2. Вивчення вже існуючих рішень

На даний момент існує одне публічне рішення, яке доступне для всіх користувачів, - це бібліотека користувача «xeipuuv» на вебресурсі GitHub [6]. Ця бібліотека має понад 100 відкритих проблем, які були створенні іншими користувачами, а останні зміни були внесені понад 4 роки тому. Це свідчить про те, що дана бібліотека є не актуальна, розробник бібліотеки перестав підтримувати власну розробку, а користувачі все одно продовжують користуватись даним рішенням та створювати нові проблеми. Про це свідчить остання проблема створена іншим користувачем 4 місяці тому [7].

Також існує ще одне публічне рішення, яке виконує допоміжну роль. Це бібліотека для генерування структур в мові програмування Golang на основі JSON-документу [8]. Ця бібліотека вирішує іншу проблему ніж об’єкт дослідження, проте може стати допоміжним інструментом для користувачів.

1.3. Постановка завдання магістерської роботи

Задачею магістерської роботи є розробка бібліотеки для роботи з JSON Schema використовуючи стандартну бібліотеку Golang. Дана бібліотека буде доступна для використання усім користувачам.

Бібліотека буде підтримувати усі типи та всі основні функції валідації, які передбачені JSON-документом.

Бібліотека буде приймати від користувача будь-який тип (any) як схему та ціль для валідації. Можливі типи для схеми та валідації: рядок (JSON, шлях до файлу або шлях до вебресурсу), масив байтів або будь-яка структура мови Go.

Бібліотека дозволить валідувати JSON об’єкти, структури мови Golang, REST запити (для найбільш популярної бібліотеки – gin).

Бібліотека буде валідувати вхідні параметри всіх функцій валідації.

При невідповідності цілі до схеми валідації, бібліотека поверне вичерпну помилку, де буде зазначено, яку функція валідації було порушено, яке значення було та яке значення очікувалось.

Висновки до розділу 1

Перший розділ присвячено аналізу предметної області. В ньому розглянуто вже наявні рішення, їх недоліки, проаналізовані скарги від користувачів. На основі цих даних було описано завдання магістерської роботи.

Також було описано з якими типами буде працювати бібліотека, які типи та функції валідації буде підтримувати бібліотека, як буде відбуватись обробка помилок та яка інформація буде міститись в помилці.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1. Огляд особливостей Golang для вирішення проблеми

Golang [9] – це мова програмування з статичною типізацією, з компілятором, розроблена Google. Синтаксис мови схожий до C, але має багато переваг відносно мови C. Go має garbage collector, структурну типізацію, безпечна робота при роботі з посиланнями на об’єкти та паралельність у стилі GSP. Цю мову називають часто саме Golang через його колишнє доменне ім'я golang.org, проте офіційна назва Go.

Особливості мови програмування Go [9]:

1) Простота та зручність використання

Основна перевага мови Go – це простота. Синтаксис немає сильних відмінностей від інших мов програмування та легкий для вивчення. Особливо через схожість з C/C++, синтаксис виглядає знайомим.

Хоча йому не вистачає функціональних можливостей інших мов програмування, його область свідомо обмежена, щоб зробити його простим. Крім того, для тих, хто намагається працювати з багатопотоковими програмами, але їм не вдається, Golang є безпечним для таких завдань. Також компілятор цієї мови не дозволяє створення невикористаних змінних. Це означає, що при створенні змінної, яка не використовується, компілятор поверне помилку.

Незважаючи на простоту у використанні, ця мова не підходить для великих проектів (найчастіше ця мова використовується для мікросервісної архітектури, коли є багато невеликих додатків).

Також в цій мові особливе ставлення до помилок, вони вважаються звичайними змінними і це нормально повернути помилку для більшості системних бібліотек. В цій мові немає generics, exceptions та extensibility, override та деяких базових типів (set, замість нього треба використовувати

стороні бібліотеки чи map). Через це доводиться писати більше коду, який погіршує читабельність.

2) Залежність віртуальної машини

Golang компілює файли з кодом у бінарні файли, тому для запуску не потребує віртуальної машини. Також залежності легко керуються, оскільки є спеціальний менеджер залежностей. Виконання без участі віртуальної машини забезпечує високу швидкість. Наприклад, в цій мові програмування будь-які цикли виконуються швидко в порівняні з іншими мовами.

При розробці застосунку використовувалися базові бібліотеки (пакети) та зовнішні бібліотеки. Особливість роботи з пакетами полягає в тому, що дуже легко створювати бібліотеки для інших користувачів. Достатньо опублікувати власну бібліотеку на Git. Цього достатньо, щоб будь-який користувач міг завантажити собі цю бібліотеку та користуватися нею.

Бінарні файли має свої недоліки. Основний з них це великий розмір. Програма «Hello world» буде займати 2МБ пам’яті. Тому багато зусиль було витрачено на оптимізацію та стискання цих файлів. Так наприклад невикористанні методи не будуть включені в двійкових файл.

3) Автоматизація

Golang має такі переваги як: автоматичне оголошення змінних, швидкий час компіляції та збір сміття без затримок. Проте всі описанні переваги можуть стати недоліками, адже не можна сподіватися, що garbage collector запуститься точно тоді, коли це потрібно.

2.2. Огляд особливостей при розробці клієнтської та серверної частин застосунку

Клієнтська частина застосунку (далі по тексу - Фронтенд) і серверна частина застосунку (далі по тексу - Бекенд) є двома найпопулярнішими термінами, що використовуються в веброзробці [10]. Фронтед – це те, що користувачі бачать і з чим взаємодіють, а бекенд – це те, як все працює. Щоб

покращити функціональність вебсайту, кожній стороні необхідно спілкуватися й ефективно працювати з іншою як єдине ціле.

Фронтед — це частина вебсайту, яку користувачі можуть бачити та з якою можуть взаємодіяти, наприклад, графічний інтерфейс користувача (GUI) і командний рядок, включаючи дизайн, навігаційні меню, тексти, зображення, відео тощо. Бекенд, навпаки, є частиною вебсайту, яку користувачі не бачать і з якою не можуть взаємодіяти.

Візуальні аспекти вебсайту, які користувачі можуть побачити та відчути, є фронтедом. З іншого боку, все, що відбувається у фоновому режимі, можна віднести до бекенда. Для фронтеду використовуються мови HTML, CSS і JavaScript, а для серверної частини – Java, Ruby, Python і .Net.

Розробник повинен переконатися, що сайт адаптивний, тобто він правильно відображається на пристроях будь-якого розміру, жодна частина вебсайту не повинна працювати некоректно, незалежно від розміру екрана.

Для побудови фронтенду використовуються наступні мови програмування [9]:

- HTML: HTML означає мову розмітки гіпертексту. Він використовується для розробки зовнішньої частини вебсторінок за допомогою мови розмітки. HTML — це комбінація гіпертексту та мови розмітки. Гіпертекст визначає зв'язок між вебсторінками. Ви можете вивчити цю мову за допомогою курсу Geeksforgeeks Advanced HTML – Self-Paced course і оволодіти поняттями розширеного HTML.
- CSS: каскадні таблиці стилів, які люблять називати CSS, — це просто розроблена мова, призначена для спрощення процесу створення вебсторінок, які виглядають презентабельно. CSS дозволяє застосовувати стилі до вебсторінок.
- JavaScript: JavaScript — це відома мова сценаріїв, яка використовується для створення магії на сайтах, щоб зробити сайт інтерактивним для користувача. Він використовується для

покращення функціональності вебсайту для запуску класних ігор і вебпрограмного забезпечення.

Також існують фреймворки та бібліотеки для розробки фронтенду [9]:

- AngularJS: AngularJs — це фреймворк JavaScript із відкритим кодом, який в основному використовується для розробки односторінкових вебдодатків (SPA). Це фреймворк, що постійно зростає та розширюється, і забезпечує кращі методи розробки вебдодатків. Він змінює статичний HTML на динамічний HTML. Це проект з відкритим кодом, який може бути безкоштовним. Він розширює атрибути HTML за допомогою директив, а дані зв’язуються за допомогою HTML.
- React.js: React — це декларативна, ефективна та гнучка бібліотека JavaScript для створення інтерфейсів користувача. ReactJS — це інтерфейсна бібліотека з відкритим вихідним кодом на основі компонентів, яка відповідає лише за рівень перегляду програми. Він підтримується Facebook.
- Bootstrap: Bootstrap — це безкоштовна колекція інструментів із відкритим кодом для створення адаптивних вебсайтів і вебдодатків. Це найпопулярніший фреймворк HTML, CSS і JavaScript для розробки адаптивних вебсайтів, орієнтованих на мобільні пристрої.
- jQuery: jQuery — це бібліотека JavaScript з відкритим вихідним кодом, яка спрощує взаємодію між документом HTML/CSS, або, точніше, об’єктною моделлю документа (DOM), і JavaScript. Уточнюючи терміни, jQuery спрощує обхід HTML-документів і маніпуляції з ними, обробку подій браузера, анімацію DOM, взаємодію Ajax і крос-браузерну розробку JavaScript.
- SASS: це надійна мова розширення CSS. Вона використовується для розширення функціональних можливостей існуючого CSS сайту, включаючи все, починаючи від змінних, успадкування та вкладеність.

- Flutter: Flutter – це SDK для розробки інтерфейсу користувача з відкритим кодом, яким керує Google. Він працює на мові програмування Dart. Він створює ефективні та гарні нативно скомпільовані додатки для мобільних пристроїв (Ios, Android), вебдодатків та настільних комп'ютерів із однієї кодової бази. Ключова перевага flutter полягає в тому, що плоска розробка стає легшою, виразнішою та гнучкою завдяки інтерфейсу користувача та нативній продуктивності.

Бекенд – це серверна сторона вебсайту. Він зберігає та впорядковує дані, а також гарантує, що все на клієнтській стороні вебсайту працює нормально. Це частина вебсайту, яку ви не можете бачити та взаємодіяти з нею. Це частина програмного забезпечення, яка не контактує безпосередньо з користувачами. Користувачі опосередковано отримують доступ до частин і характеристик, розроблених дизайнерами серверної частини, через зовнішню програму. Такі види діяльності, як написання API, створення бібліотек і робота з системними компонентами без інтерфейсів користувача або навіть систем наукового програмування, також включені в бекенд.

Для побудови фронтенду використовуються наступні мови програмування [9]:

- PHP: PHP — це серверна мова сценаріїв, розроблена спеціально для веброзробки. Оскільки код PHP виконується на стороні сервера, його називають мовою сценаріїв на стороні сервера.
- Java: Java є однією з найпопулярніших і широко використовуваних мов програмування та платформ.
- Python: Python — це мова програмування, яка дає змогу швидко працювати та ефективніше інтегрувати системи.
- Node.js: Node.js — це кросплатформове середовище виконання з відкритим кодом для виконання коду JavaScript поза браузером. Ви повинні пам'ятати, що NodeJS не є фреймворком і не є мовою

програмування. Більшість людей плутають і розуміють, що це фреймворк або мова програмування. Ми часто використовуємо Node.js для створення внутрішніх служб, таких як API, наприклад Web App або Mobile App. Він використовується у виробництві великими компаніями, такими як Paypal, Uber, Netflix, Walmart тощо.

Також існують фреймворки та бібліотеки для розробки бекенду [9]:

- Express – Express – це фреймворк Nodejs, який використовується для розробки на стороні серверу. Він використовується для створення односторінкових, багатосторінкових і гібридних вебдодатків. З його допомогою ви можете обробляти кілька різних HTTP-запитів.
- Django – Django — це вебфреймворк на Python, що дотримується шаблону MTV (model-template-views). Він використовується для створення великих і складних вебдодатків. Серед його особливостей — швидкість, безпека та можливість масштабування.
- Ruby on Rails – Ruby on Rails — це платформа на стороні сервера, що дотримується шаблону архітектури MVC (model-view-controller). Він надає такі структури за замовчуванням, як вебсервіси, вебсторінки та бази даних.
- Laravel – Laravel є платформою вебдодатків для PHP. Особливістю, яка робить його ідеальним, є повторне використання компонентів різних фреймворків для створення вебдодатку.
- Spring – ця платформа на стороні сервера забезпечує підтримку інфраструктури для програм Java. Він діє як підтримка різних фреймворків, таких як Hibernate, Struts, EJB тощо. Він також має розширення, які допомагають швидко та легко розробляти програми Java.

2.3. Огляд особливостей комунікації клієнтської та серверної частин застосунку

Існують основні чотири способи комунікації між клієнтською та серверною частинами застосунку: REST API, WebSockets, Server-Side Rendering (SSR) та GraphQL.

2.3.1 REST API

Representational State Transfer (REST) [11] — це архітектурний стиль, який визначає набір обмежень для створення вебслужб. REST API — це простий і гнучкий спосіб доступу до вебслужб без будь-якої обробки. Технології REST зазвичай надають перевагу перед більш надійною Simple Object Access Protocol (SOAP), оскільки REST використовує меншу пропускну здатність, простий та гнучкий, що робить його більш придатним для використання в Інтернеті. Він використовується для отримання або надання певної інформації з вебслужби. Уся комунікація через REST API використовує лише HTTP.

Запит надсилається від клієнта до сервера у формі вебURL як HTTP запит. Після цього відповідь повертається з сервера у вигляді ресурсу, який може бути чим завгодно, як HTML, XML, зображення або JSON. Але зараз JSON є найпопулярнішим форматом, який використовується у вебслужбах.

The Hypertext Transfer Protocol (HTTP) [13] — це протокол прикладного рівня в моделі набору протоколів Інтернету для розподілених, спільних, гіпермедійних інформаційних систем. HTTP є основою передачі даних для World Wide Web (WWW), де гіпертекстові документи містять гіперпосилання на інші ресурси, до яких користувач може легко отримати доступ, наприклад, клацнувши мишею або торкнувшись екрана у веббраузері.

Розробка HTTP була розпочата Тімом Бернерсом-Лі в CERN у 1989 році та підсумована в простому документі, що описує поведінку клієнта та сервера за допомогою першої версії HTTP під назвою 0.9. Згодом ця версія була розроблена, і згодом вона стала загальнодоступною 1.0.

Розробка ранніх HTTP Requests for Comments (RFC) почалася через кілька років у скоординованих зусиллях Internet Engineering Task Force (IETF) і World Wide Web Consortium (W3C), а пізніше робота перейшла до IETF.

HTTP/1 було завершено та повністю задокументовано (як версія 1.0) у 1996 році. Він розвинувся (як версія 1.1) у 1997 році, а потім його специфікації були оновлені в 1999, 2014 та 2022 роках.

Його безпечний варіант під назвою HTTPS використовується більш ніж 85% вебсайтів. HTTP/2, опублікований у 2015 році, забезпечує більш ефективне вираження семантики HTTP "на дроті". Станом на січень 2024 року його використовують 36% вебсайтів і підтримують майже всі веббраузери (понад 98% користувачів). Він також підтримується основними вебсерверами через Transport Layer Security (TLS) з використанням розширення Application-Layer Protocol Negotiation (ALPN), де потрібен TLS 1.2 або новіший.

HTTP/3, наступник HTTP/2, був опублікований у 2022 році. Станом на лютий 2024 року він використовується на 29% вебсайтів і підтримується більшістю веббраузерів, тобто (принаймні частково) підтримується 97% користувачів. HTTP/3 використовує QUIC замість TCP для основного транспортного протоколу. Підтримку HTTP/3 спочатку було додано до Cloudflare та Google Chrome, а також увімкнено у Firefox. HTTP/3 має нижчу затримку для реальних вебсторінок, якщо ввімкнено на сервері, і завантажується швидше, ніж HTTP/2, у деяких випадках утричі швидше, ніж HTTP/1.1.

HTTP визначає методи (іноді їх називають дієсловами, але ніде в специфікації не згадується дієслово), щоб вказати бажану дію, яка має бути виконана на визначеному ресурсі. Те, що представляє цей ресурс, будь то вже існуючі дані чи дані, які генеруються динамічно, залежить від реалізації сервера. Часто ресурс відповідає файлу або виводу виконуваного файлу, що

знаходиться на сервері. Специфікація HTTP/1.0 визначила методи GET, HEAD і POST, а також перерахувала методи PUT, DELETE, LINK і UNLINK у додаткових методах. Проте специфікація HTTP/1.1 офіційно визначила та додала п’ять нових методів: PUT, DELETE, CONNECT, OPTIONS і TRACE. Будь-який клієнт може використовувати будь-який метод, а сервер можна налаштувати для підтримки будь-якої комбінації методів. Якщо метод невідомий посереднику, він буде розглядатися як небезпечний і нереалізований метод. Немає обмежень щодо кількості методів, які можна визначити, що дозволяє вказувати майбутні методи без порушення існуючої інфраструктури. Наприклад, WebDAV визначив сім нових методів, а RFC 5789 [12] вказав метод PATCH. Назви методів чутливі до регістру. Це на відміну від імен полів заголовків HTTP, які нечутливі до регістру.

У HTTP існує п’ять методів, які зазвичай використовуються в архітектурі на основі REST, тобто POST, GET, PUT, PATCH і DELETE. Вони відповідають операціям створення, читання, оновлення та видалення (або CRUD) відповідно. Існують інші методи, які використовуються рідше, наприклад OPTIONS і HEAD.

Методи HTTP [13]:

- GET: Метод GET вимагає від цільового ресурсу передати представлення свого стану. Запити GET мають лише отримувати дані й не мають жодних інших ефектів. (Це також вірно для деяких інших методів HTTP.) Для отримання ресурсів без внесення змін GET є кращим над POST, оскільки до них можна звертатися через URL. Це дозволяє створювати закладки та ділитися, а також робить відповіді GET придатними для кешування, що може заощадити пропускну здатність. W3C опублікував керівні принципи щодо цієї відмінності, кажучи: «Розробка вебдодатків повинна ґрунтуватися на вищезазначених принципах, а також на відповідних обмеженнях».

- HEAD: Метод HEAD вимагає, щоб цільовий ресурс передав представлення свого стану, як для запиту GET, але без даних представлення, вкладених у тіло відповіді. Це корисно для отримання метаданих подання в заголовку відповіді без необхідності передавати все подання. Використання включає перевірку доступності сторінки через код стану та швидке визначення розміру файлу (довжина вмісту).
- POST: Метод POST вимагає, щоб цільовий ресурс обробив представлення, укладене в запиті, відповідно до семантики цільового ресурсу. Наприклад, він використовується для публікації повідомлень на форумі в Інтернеті, підписки на список розсилки або завершення покупки в Інтернеті.
- PUT: Метод PUT вимагає, щоб цільовий ресурс створив або оновив свій стан відповідно до стану, визначеного представленням, укладеним у запиті. Відмінність від POST полягає в тому, що клієнт вказує цільове розташування на сервері.
- DELETE: Метод DELETE вимагає від цільового ресурсу видалити свій стан.
- CONNECT: Метод CONNECT вимагає від посередника встановити тунель TCP/IP до вихідного сервера, ідентифікованого цільовим запитом. Він часто використовується для захисту з'єднань через один або кілька HTTP-проксі з TLS.
- OPTIONS: Метод OPTIONS запитує, щоб цільовий ресурс передав методи HTTP, які він підтримує. Це можна використовувати для перевірки функціональності вебсервера, запитуючи '*' замість певного ресурсу.

- TRACE: Метод TRACE вимагає, щоб цільовий ресурс передав отриманий запит у тіло відповіді. Таким чином клієнт може побачити, які (якщо такі є) зміни чи доповнення внесли посередники.
- PATCH: Метод PATCH вимагає, щоб цільовий ресурс змінив свій стан відповідно до часткового оновлення, визначеного у представленні, включеному в запит. Це може заощадити пропускну здатність шляхом оновлення частини файлу чи документа без необхідності передавати його повністю.

2.3.2 WebSockets

WebSocket [14] — це протокол комп'ютерного зв'язку, який забезпечує одночасні двосторонні канали зв'язку через одне з'єднання Transmission Control Protocol (TCP). Протокол WebSocket був стандартизований IETF як RFC 6455 [15] у 2011 році. Поточна специфікація, яка дозволяє вебдодаткам використовувати цей протокол, відома як WebSockets. Це стандарт, який підтримується WHATWG і є наступником WebSocket API від W3C.

WebSocket відрізняється від HTTP, який використовується для обслуговування більшості вебсторінок. Незважаючи на те, що вони різні, RFC 6455 [15] стверджує, що WebSocket «розроблений для роботи через HTTP-порти 443 і 80, а також для підтримки проксі-серверів і посередників HTTP», що робить його сумісним з HTTP. Для досягнення сумісності рукостискання WebSocket використовує заголовок HTTP Upgrade для зміни протоколу HTTP на протокол WebSocket.

Протокол WebSocket забезпечує повнодуплексну взаємодію між веббраузером (або іншим клієнтським додатком) і вебсервером з меншими накладними витратами, ніж напівдуплексні альтернативи, такі як HTTP, полегшуючи передачу даних у реальному часі від сервера та до нього. Це стало можливим завдяки забезпеченню стандартизованого способу для сервера надсилати вміст клієнту без попереднього запиту клієнта, а також

дозволу передачі повідомлень вперед і назад, зберігаючи з'єднання відкритим. Таким чином між клієнтом і сервером може відбуватися двостороння безперервна розмова. Зв'язок зазвичай здійснюється через TCP-порт номер 443 (або 80 у випадку незахищених з'єднань. Крім того, WebSocket дозволяє надсилати потоки повідомлень поверх TCP. Тільки TCP має справу з потоками байтів без внутрішньої концепції повідомлення. Подібні двосторонні зв'язки між браузером і сервером були досягнуті нестандартизованими способами з використанням тимчасових технологій, таких як Comet або Adobe Flash Player.

Більшість браузерів підтримують протокол, включаючи Google Chrome, Firefox, Microsoft Edge, Internet Explorer, Safari та Opera.

Специфікація протоколу WebSocket визначає ws (WebSocket) і wss (WebSocket Secure) як дві нові uniform resource identifier (URI), які використовуються для незашифрованих і зашифрованих з'єднань відповідно. Крім назви схеми та фрагмента (тобто # не підтримується), решта компонентів URI визначено для використання загального синтаксису URI.

WebSocket вперше згадувався як TCPConnection у специфікації HTML5, як замінник для API сокетів на основі TCP. У червні 2008 року під керівництвом Майкла Картера відбулася низка дискусій, результатом яких стала перша версія протоколу, відомого як WebSocket. До WebSocket повнодуплексний зв'язок через порт 80 був доступний за допомогою каналів Comet; однак реалізація Comet є нетривіальною, і через зв'язок TCP і накладні витрати заголовка HTTP вона неефективна для невеликих повідомлень. Протокол WebSocket спрямований на вирішення цих проблем без шкоди для припущень безпеки. Назва «WebSocket» була придумана Ієном Хіксоном і Майклом Картером незабаром після цього під час співпраці в чаті #whatwg IRC, а згодом автором для включення в специфікацію HTML5 був Ієн Хіксон. У грудні 2009 року Google Chrome 4

став першим браузером, який повністю підтримував стандарт із увімкненим WebSocket за замовчуванням. У лютому 2010 року розробку протоколу WebSocket було передано групами W3C і WHATWG до IETF, автором яких були дві редакції під керівництвом Яна Хіксона .

RFC 7692 [16] представив розширення стиснення для WebSocket за допомогою алгоритму DEFLATE на основі кожного повідомлення.

Захищена версія протоколу WebSocket реалізована у Firefox 6, Safari 6, Google Chrome 14, Opera 12.10 та Internet Explorer 10.

Рукостискання починається із HTTP-запиту/відповіді, що дозволяє серверам обробляти HTTP-з'єднання, а також WebSocket-з'єднання на одному порту. Після встановлення з'єднання зв'язок перемикається на двонаправлений двійковий протокол, який не відповідає протоколу HTTP.

Окрім заголовків Upgrade, клієнт надсилає заголовок Sec-WebSocket-Key, що містить випадкові байти в кодуванні base64, а сервер відповідає хешем ключа в заголовку Sec-WebSocket-Accept. Це призначено для того, щоб запобігти повторному надсиланню попередньої розмови WebSocket проксі-сервером і не забезпечує автентифікації, конфіденційності чи цілісності. Функція хешування додає фіксований рядок 258EAFA5-E914-47DA-95CA-C5AB0DC85B11 (UUID [17]) до значення з заголовка Sec-WebSocket-Key (який не декодується з base64), застосовує функцію хешування SHA-1 і кодує результат з використанням base64.

RFC6455 [15] вимагає, щоб ключ повинен бути одноразовим, що складається з випадково вибраного 16-байтового значення, закодованого в base64, тобто 24 байти в base64 (з останніми двома байтами ==). Хоча деякі прості HTTP-сервери дозволяють подавати коротші ключі, багато сучасних HTTP-серверів відхилять запит із повідомленням про помилку «недійсний заголовок Sec-WebSocket-Key».

2.3.3 Server-Side Rendering (SSR)

Server-Side Rendering [18] — це техніка, яка використовується у веброзробці, яка включає використання сценаріїв на вебсервері, який створює відповідь, налаштовану для кожного запиту користувача (клієнта) до вебсайту. Сценарії на стороні сервера відрізняються від сценаріїв на стороні клієнта, де вбудовані сценарії, такі як JavaScript, запускаються на стороні клієнта у веббраузері, але обидва методи часто використовуються разом. Альтернативою будь-якому або обом типам сценаріїв є те, що вебсервер сам надає статичну вебсторінку.

Сценарії на стороні сервера часто використовуються для надання налаштованого інтерфейсу для користувача. Ці сценарії можуть збирати характеристики клієнта для використання в налаштуванні відповіді на основі цих характеристик, вимог користувача, прав доступу тощо.

Сценарії на стороні сервера також дозволяють власнику вебсайту приховати вихідний код, який генерує інтерфейс, тоді як клієнт- сторонні сценарії, користувач має доступ до всього коду, отриманого клієнтом.

Недоліком використання сценаріїв на стороні сервера є те, що клієнту потрібно робити додаткові запити через мережу до сервера, щоб показати нову інформацію користувачеві через веббраузер. Ці запити можуть уповільнити роботу користувача, посилити навантаження на сервер і запобігти використанню програми, коли користувач відключений від сервера.

Коли сервер обслуговує дані загальноприйнятим способом, наприклад, відповідно до протоколів HTTP або FTP, користувачі можуть вибирати з кількох програм-клієнтів (більшість сучасних веббраузерів можуть запитувати й отримувати дані за допомогою обох цих протоколів).

Netscape представила реалізацію JavaScript для сценаріїв на стороні сервера з Netscape Enterprise Server, вперше випущеним у грудні 1994 року (незабаром після випуску JavaScript для браузерів).

Пізніше на початку 1995 року Фред Дюфрен під час розробки першого вебсайту для телевізійної станції WCVB у Бостоні, штат Массачусетс, використав сценарії на стороні сервера. Технологія описана в патенті США 5835712. Патент був виданий у 1998 році і зараз належить Open Invention Network (OIN). У 2010 році OIN назвав Фреда ДюФрена «Видатним винахідником» за його роботу над серверними сценаріями.

2.3.4 GraphQL

GraphQL [19] — це мова запитів і обробки даних із відкритим кодом для API і механізму виконання запитів. GraphQL забезпечує декларативне отримання даних, де клієнт може точно вказати, які дані йому потрібні з API. Замість кількох кінцевих точок, які повертають окремі дані, сервер GraphQL надає одну кінцеву точку та відповідає саме тими даними, які запитує клієнт. Оскільки сервер GraphQL може отримувати дані з окремих джерел і представляти дані в уніфікованому графіку, він не прив’язаний до жодної конкретної бази даних чи механізму зберігання.

Facebook розпочав розробку GraphQL у 2012 році та випустив його як відкритий код у 2015 році. У 2018 році GraphQL було переміщено до нещодавно створеної GraphQL Foundation, організованої некомерційною організацією Linux Foundation. 9 лютого 2018 року GraphQL Schema Definition Language (SDL) стала частиною специфікації. Багато популярних загальнодоступних API використовують GraphQL як стандартний спосіб доступу до них. Сюди входять загальнодоступні API Facebook, GitHub, Yelp, Shopify і Google Directions API.

GraphQL підтримує читання, запис (зміеу) і підписку на зміни даних (оновлення в реальному часі – зазвичай реалізується за допомогою WebSockets). Сервіс GraphQL створюється шляхом визначення типів із полями, а потім наданням функцій для визначення даних для кожного поля. Типи та поля складають те, що називається визначенням схеми. Функції, які витягують і відображають дані, називаються резольверами. Після перевірки

на відповідність схемі запит GraphQL виконується сервером. Сервер повертає результат, який відображає форму вихідного запиту, як правило, як JSON.

API GraphQL можна протестувати вручну або за допомогою автоматизованих інструментів, які надсилають запити GraphQL і перевіряють правильність результатів. Також можлива автоматична генерація тестів. Нові запити можуть створюватися за допомогою методів пошуку завдяки типовій схемі та можливостям самоаналізу. Деякі з програмних інструментів, які використовуються для тестування реалізації GraphQL, включають Postman, GraphiQL, Apollo Studio, GraphQL Editor і Step CI.

2.4. Огляд особливостей формату даних при використанні REST API

Найбільш часто використовують наступні формати даних: JSON, XML, YAML та Protocol Buffers.

2.4.1 JSON

JSON (JavaScript Object Notation) [2] — це відкритий стандартний формат файлів і формат обміну даними, який використовує зрозумілий для людини текст для зберігання та передачі об’єктів даних, що складаються з пар атрибут–значення та масивів (або інших серіалізованих значень). Це широко використовуваний формат даних із різноманітним використанням в електронному обміні даними, включно з вебдодатками.

JSON — це незалежний від мови формат даних. Він був похідним від JavaScript, але багато сучасних мов програмування включають код для генерації та аналізу даних у форматі JSON. Імена файлів JSON мають розширення .json.

Дуглас Крокфорд спочатку вказав формат JSON на початку 2000-х. Він і Чіп Морнінгстар надіслали перше повідомлення JSON у квітні 2001 року.

Після того як RFC 4627 [20] був доступний як «інформаційна» специфікація з 2006 року, JSON був вперше стандартизований у 2013 році як ECMA-404. RFC 8259 [21], опублікований у 2017 році, є поточною версією Інтернет-стандарту STD 90 і залишається сумісним з ECMA-404. Того ж року JSON також було стандартизовано як ISO/IEC 21778:2017. Стандарти ECMA та ISO/IEC описують лише дозволений синтаксис, тоді як RFC охоплює деякі питання безпеки та сумісності.

Основні типи даних JSON:

- Number: десяткове число зі знаком, яке може містити дробову частину та використовувати експоненціальний запис E, але не може містити нечисла, наприклад NaN. Формат не має різниці між цілими числами та числами з плаваючою комою. JavaScript використовує формат подвійної точності з плаваючою комою IEEE-754 для всіх своїх числових значень (пізніше також підтримується BigInt), але інші мови, що реалізують JSON, можуть кодувати числа по-іншому.
- String: послідовність із нуля або більше символів Unicode. Рядки розділені подвійними лапками та підтримують синтаксис екранування зворотної косої риски.
- Boolean: значення: будь-яке зі значень true або false
- Array: упорядкований список з нуля або більше елементів, кожен з яких може бути будь-якого типу. Для масивів використовується нотація в квадратних дужках з елементами, розділеними комами.
- Object: набір пар ключ–значення, де ключі є рядками. Поточний стандарт ECMA стверджує: «Синтаксис JSON не накладає жодних обмежень на рядки, які використовуються як ключі, не вимагає, щоб рядки ключів були унікальними, і не надає жодного значення впорядкуванню пар ключ - значення». Об’єкти розділені фігурними дужками та використовуються коми для розділення кожної пари, тоді як у кожній парі двокрапка «:» відокремлює ключ від його значення.

- null: порожнє значення, використовуючи слово null

Пробіли дозволені та ігноруються навколо або між синтаксичними елементами (значеннями та пунктуацією, але не в межах значення рядка). Для цієї мети пробілами вважаються чотири спеціальні символи: пробіл, горизонтальна табуляція, переведення рядка та повернення каретки. Зокрема, позначка порядку байтів не повинна генеруватися відповідною реалізацією (хоча вона може бути прийнята під час аналізу JSON). JSON не забезпечує синтаксис для коментарів.

Ранні версії JSON (наприклад, визначені RFC 4627 [20]) вимагали, щоб дійсний текст JSON складався лише з об’єкта або типу масиву, який міг містити інші типи всередині них. Це обмеження було скасовано в RFC 7158 [22], де текст JSON було перевизначено як будь-яке серіалізоване значення.

Числа в JSON є агностиками щодо їх представлення в мовах програмування. Хоча це дозволяє серіалізувати числа довільної точності, це може призвести до проблем з сумісністю. Наприклад, оскільки не проводиться розрізнення між цілими числами та значеннями з плаваючою комою, деякі реалізації можуть розглядати 42, 42.0 і 4.2E+1 як одне й те саме число, а інші – ні. Стандарт JSON не висуває жодних вимог щодо деталей реалізації, таких як переповнення, недоповнення, втрата точності, округлення або нулі зі знаком, але він рекомендує очікувати не більше двійкової точності IEEE 75464 [23] для «гарної сумісності». Немає внутрішньої втрати точності при серіалізації двійкового представлення числа з плаваючою комою на машинному рівні (наприклад, binary64) у зрозуміле людині десяткове представлення (як числа в JSON) і назад, оскільки існують опубліковані алгоритми, щоб зробити це точно і оптимально.

Коментарі були навмисно виключені з JSON. У 2012 році Дуглас

Крокфорд так описав своє дизайнерське рішення: «Я видалив коментарі з JSON, тому що бачив, як люди використовують їх для зберігання директив синтаксичного аналізу, практика, яка зруйнувала б взаємодію».

JSON забороняє «кінцеві коми», кому після останнього значення всередині структури даних. Кінцеві коми є загальною особливістю похідних JSON для покращення простоти використання.

2.4.2 XML

Extensible Markup Language (XML) [1] — це мова розмітки та формат файлів для зберігання, передачі та реконструкції довільних даних. Він визначає набір правил для кодування документів у форматі, який читається як людиною, так і машиною. Специфікація XML 1.0 Консорціуму World Wide Web від 1998 року та кілька інших пов’язаних специфікацій — визначають XML.

Цілі дизайну XML наголошують на простоті, загальності та зручності використання в Інтернеті. Це формат текстових даних з підтримкою Unicode для різних людських мов. Незважаючи на те, що дизайн XML зосереджений на документах, ця мова широко використовується для представлення довільних структур даних.

Основною метою XML є серіалізація, тобто зберігання, передача та реконструкція довільних даних. Щоб дві різні системи обмінювалися інформацією, їм потрібно узгодити формат файлу. XML стандартизує цей процес.

Як мова розмітки, XML позначає, категоризує та структурно організовує інформацію. Теги XML представляють структуру даних і містять метадані. Теги містять дані, закодовані у спосіб, визначений стандартом XML. Додаткова схема XML (XSD) визначає необхідні метадані для інтерпретації та перевірки XML. (Це також називають канонічною схемою).

IETF RFC 7303 [24] (який замінює старіший RFC 3023 [25]) надає правила побудови типів носіїв для використання в XML-повідомленні. Він визначає три типи носіїв: application/xml (text/xml є псевдонімом), application/xml-external-parsed-entity (text/xml-external-parsed-entity є псевдонімом) і application/xml-dtd. Вони використовуються для передачі необроблених файлів XML без розкриття їх внутрішньої семантики. RFC 7303 [24] також рекомендує надавати мовам на основі XML типи медіа, які закінчуються на +xml, наприклад, image/svg+xml для SVG.

Подальші вказівки щодо використання XML у мережевому контексті містяться в RFC 3470 [26], також відомому як IETF BCP 70, документі, що охоплює багато аспектів розробки та розгортання мови на основі XML.

XML увійшов у загальне використання для обміну даними через Інтернет. Було розроблено сотні форматів документів із використанням синтаксису XML, включаючи RSS, Atom, Office Open XML, OpenDocument, SVG, COLLADA та XHTML. XML також надає базову мову для протоколів зв’язку, таких як SOAP і XMPP. Це один із форматів обміну повідомленнями, який використовується в техніці програмування асинхронного JavaScript і XML (AJAX).

Багато галузевих стандартів даних, таких як Health Level 7, OpenTravel Alliance, FpML, MISMO, базуються на XML і багатих функціях специфікації схеми XML. У видавничій справі Darwin Information Typing Architecture є галузевим стандартом даних XML. XML широко використовується для підтримки різних форматів публікації.

Одним із застосувань XML є передача оперативної метеорологічної інформації (OPMET) на основі стандартів IWXXM.

Коментарі можуть з’являтися в будь-якому місці документа за межами іншої розмітки. Коментарі не можуть розміщуватися перед оголошенням XML. Коментарі починаються з <!-- і закінчуються на -->. Для сумісності з SGML рядок "--" (подвійний дефіс) не дозволяється всередині

коментарів - це означає, що коментарі не можуть бути вкладеними. Амперсанд не має особливого значення в коментарях, тому посилання на сутності та символи не розпізнаються як такі, і немає способу представити символи поза набором символів кодування документа.

Приклад правильного коментаря: <!--коментар-->

2.4.3 YAML

YAML [27] — це спосіб серіалізації даних. Він зазвичай використовується для конфігураційних файлів і в програмах, де дані зберігаються або передаються. YAML націлений на багато тих самих комунікаційних програм, що й XML, але має мінімальний синтаксис, який навмисно відрізняється від Standard Generalized Markup Language (SGML). Він використовує відступи в стилі Python для позначення вкладеності і не вимагає лапок навколо більшості рядкових значень (він також підтримує стиль JSON [...] і {...}, змішаний в одному файлі).

Дозволяються спеціальні типи даних, але YAML нативно кодує скаляри (наприклад, рядки, цілі числа та числа з плаваючою точкою), списки та асоціативні масиви (також відомі як карти, словники або хеші).

Ці типи даних засновані на мові програмування Perl, хоча всі широко використовувані мови програмування високого рівня мають дуже схожі концепції. Синтаксис, орієнтований на двокрапку, який використовується для вираження пар ключ-значення, натхненний заголовками електронної пошти, як визначено в RFC 822 [28], а роздільник документа --- запозичено з MIME (RFC 2046 [29]). Екран-послідовності повторно використовуються з C, а обтікання пробілами для багаторядкових рядків запозичено з HTML. Списки та хеші можуть містити вкладені списки та хеші, утворюючи структуру дерева. Довільні графіки можуть бути представлені за допомогою псевдонімів YAML (подібно до XML у SOAP). YAML призначений для читання та запису в потоках, функція, запозичено з SAX.

Підтримка читання та запису YAML доступна для багатьох мов програмування. Деякі редактори вихідного коду, такі як Vim, Emacs та різноманітні інтегровані середовища розробки мають функції, які полегшують редагування YAML, такі як згортання вкладених структур або автоматичне підсвічування синтаксичних помилок.

З 2006 року офіційним рекомендованим розширенням імені файлу YAML є .yaml. У 2024 році було завершено роботу над MIME-типом application/yaml.

2.4.4 Protocol Buffers

Protocol Buffers (Protobuf) [30] — це безкоштовний міжплатформний формат даних із відкритим кодом, який використовується для серіалізації структурованих даних. Метод включає мову опису інтерфейсу, яка описує структуру деяких даних, і програму, яка генерує вихідний код з цього опису для створення або аналізу потоку байтів, який представляє структуровані дані.

Google розробив Protobuf для внутрішнього використання та надав генератор коду для кількох мов за ліцензією на відкритий код.

Цілі проектування Protobuf наголошували на простоті та продуктивності. Зокрема, він був розроблений, щоб бути меншим і швидшим, ніж XML.

Protobuf широко використовуються в Google для зберігання та обміну всіма видами структурованої інформації. Цей метод служить основою для спеціальної системи remote procedure call (RPC), яка використовується майже для всіх комунікацій у Google.

Буфери протоколів подібні до протоколів Apache Thrift, Ion і Microsoft Bond. Пропонуючи конкретний стек протоколів RPC для використання для визначених служб під назвою gRPC.

Схеми структури даних (так звані повідомлення) і служби описані у файлі визначення proto (.proto) і скомпільовані за допомогою protoc. Ця

компіляція генерує код, який може бути викликаний відправником або одержувачем цих структур даних.

Канонічно повідомлення серіалізуються в двійковий формат, який є компактним, сумісним у прямому та зворотному напрямках, але не описує себе (тобто неможливо повідомити імена, значення або повні типи даних полів без зовнішньої специфікації). Немає визначеного способу включення або посилання на таку зовнішню специфікацію (схему) у файлі буферів протоколу. Офіційно підтримувана реалізація включає формат серіалізації ASCII, але цей формат — хоча й описується сам — втрачає поведінку прямої та зворотної сумісності.

Хоча основною метою Protobuf є полегшення мережевого зв’язку, їх простота та швидкість роблять Protobuf альтернативою орієнтованим на дані класам і структурам C++, особливо там, де в майбутньому може знадобитися взаємодія з іншими мовами чи системами.

Protobufs не мають єдиної специфікації. Формат найкраще підходить для невеликих фрагментів даних, які не перевищують кількох мегабайтів і можуть бути завантажені/надіслані в пам’ять відразу, тому не є потоковим форматом. Бібліотека не забезпечує стиснення з коробки. Формат також не підтримується необ’єктно-орієнтованими мовами (наприклад, Fortran).

Висновки до розділу 2

У другому розділі роботи було розглянуто особливості використання мови Golang – мова надає всі необхідні засоби та вбудовані бібліотеки для всіх завдань та складностей, які можуть виникнути під час розробки, також мова швидко набирає популярність, що забезпечить розповсюдження серед користувачів бібліотеки.

Також було розглянуто особливості взаємодії між клієнтською та серверною частинами застосунку та формат даних для використання в

REST API. REST API та JSON є найчастішим вибором користувачів, отже бібліотека розрахована на найбільшу частину розробників, які займаються саме веброзробкою.

РОЗДІЛ 3. РОЗРОБКА БІБЛІОТЕКИ ДЛЯ ВАЛІДАЦІЇ JSON-ДОКУМЕНТУ

3.1 Опис алгоритму

Весь алгоритм валідації JSON-документу можна поділити на три частини – валідація самого JSON, валідації схеми (структури) та безпосередньо валідація JSON-документу до схеми.

```
func Validate(target any, schema any) error {
    validatedTarget, err := validateTarget(reflect.ValueOf(target))
    if err != nil {
        return err
    }

    validatedSchema, err := validateSchema(reflect.ValueOf(schema))
    if err != nil {
        return err
    }

    err = validate(validatedTarget, validatedSchema)
    if err != (*Error)(nil) {
        return err
    }

    return nil
}
```

Рисунок 3.1 – порядок валідації JSON-документу

3.1.1 Опис алгоритму валідації JSON

Бібліотека підтримує будь-який формат JSON, який користувач може використовувати в свої застосунках – це текст (JSON, шлях до файлу або шлях до вебресурсу), масив байтів або будь-яка структура мови Go.

Для цього використовується базова бібліотека мови Go – «reflect». Ця бібліотека надає можливість отримати тип об’єкту, і виконувати різні дії вже в залежності від його типу.

Якщо користувач хоче провалідувати структуру – для цього використовується базова бібліотека – «encoding/json». Вона використовує

тільки публічні поля структури та теги, які можуть бути додані для кожного поля окремо. Ця бібліотека використовує іншу базову бібліотеку - «reflect», тому не було необхідності написання власного рішення під ці цілі.

3.1.2 Опис алгоритму валідації схеми

Бібліотека підтримує будь-який формат схеми, так само як будь-який формат JSON.

Схема – це структура, яка містить наступні поля.

```
type Schema struct {
    valueType    ValueType
    validateFunc map[string]validateFunc
    properties   map[string]Schema
}

type validateFunc func(any) *Error
```

Рисунок 3.2 – структура схеми

ValueType – це поле, яке визначає відповідний тип для валідації та можливі функції для валідації.

```
type ValueType string

const (
    String  ValueType = "string"
    Integer ValueType = "integer"
    Number  ValueType = "number"
    Object  ValueType = "object"
    Array   ValueType = "array"
    Boolean ValueType = "boolean"
    Null    ValueType = "null"
)
```

Рисунок 3.3 – можливі значення для поля JSON

Схема містить хеш-таблицю, де ключ – це назва функції валідації, а значення сама функція валідації.

Також схема містить іншу хеш-таблицю з полями об’єкту, це поле використовується тільки для типу «object». Ключ – це назва поля, значення – значення поля.

Спочатку визначається тип схеми, далі додаються функції валідації, які присутні в схемі. Якщо тип схеми – це об’єкт, рекурсивно додаються всі поля об’єкту, кожне поле має власну схему.

Кожна функція валідації може приймати різну кількість вхідних параметрів, всі вхідні параметри також проходять відповідну валідацію. Наприклад функція валідації minLength повинна мати значення цілого позитивного числа.

```
v, ok := min.(float64)
if !ok {
    return nil, errors.New("minLength requires integer")
}

if float64(int(v)) != v {
    return nil, errors.New("minLength requires integer")
}
```

Рисунок 3.4 – приклад валідації вхідних параметрів.

Базова бібліотека, яка працює з JSON за замовчуванням використовує число з плаваючою крапкою навіть для цілих чисел, тому спочатку перевіряється чи значення є числом з плаваючою крапкою, а потім чи число є цілим.

Схема вважається сформована, коли схема має правильний тип та всі функції пройшли валідацію вхідних параметрів.

3.2 Валідація тіла HTTP запита

Бібліотека надає можливість валідувати тіло запиту для найбільш розповсюдженого фреймворку gin за допомогою власного «middleware». Він приймає схему та валідує тіло запиту, це допомагає виділити валідацію тіла запиту з самого хендлеру на рівень опису доступних ендпоінтів.

```
func ValidateMiddleware(schema any) gin.HandlerFunc {
    return func(c *gin.Context) {
        var data map[string]any
        if err := c.ShouldBindBodyWith(&data, binding.JSON); err != nil {
            c.AbortWithError(http.StatusBadRequest, err)
        }

        if err := Validate(data, schema); err != nil {
            c.AbortWithError(http.StatusBadRequest, err)
        }

        c.Next()
    }
}
```

Рисунок 3.5 – валідація тіла запита

3.3 Опис доступних валідацій

Бібліотека надає доступ до всіх основних валідацій всіх типів, які передбачені JSON-документом.

3.3.1 Валідації типу String

Перелік функцій валідації для типу String [32]:

1. minLength: Рядок дійсний для цього ключового слова, якщо його довжина більша або дорівнює значенню цього ключового слова. Значення цього ключового слова має бути цілим невід’ємним числом.
2. maxLength: Рядок дійсний для цього ключового слова, якщо його довжина менша або дорівнює значенню цього ключового слова. Значення цього ключового слова має бути цілим невід’ємним числом.
3. pattern: Рядок дійсний для цього ключового слова, якщо він відповідає регулярному виразу, визначеному значенням цього ключового слова.

Значення цього ключового слова має бути рядком, що представляє дійсний регулярний вираз.

4. format: Ключове слово format дозволяє базову семантичну ідентифікацію певних видів рядкових значень, які зазвичай використовуються. Наприклад, оскільки JSON не має типу «DateTime», дати потрібно кодувати як рядки. Формат дозволяє автору схеми вказати, що значення рядка слід інтерпретувати як дату. За замовчуванням формат є лише анотацією і не впливає на перевірку. Поле формат може мати наступні значення:
    a. date-time
    b. time
    c. date
    d. duration
    e. email
    f. hostname
    g. ipv4
    h. ipv6
    i. uuid
    j. uri
    k. regex

3.3.2 Валідація типу Number

Перелік функцій валідації для типу Number [33]:

1. minimum: Число є дійсним для цього ключового слова, якщо воно більше або дорівнює значенню цього ключового слова. Значення цього ключового слова має бути числом (цілим або з плаваючою точкою).
2. exclusiveMinimum: Число дійсне для цього ключового слова, якщо воно строго більше за значення цього ключового слова. Значення

цього ключового слова має бути числом (цілим або з плаваючою точкою) або логічним значенням.

3. maximum: Число дійсне для цього ключового слова, якщо воно менше або дорівнює значенню цього ключового слова. Значення цього ключового слова має бути числом (цілим або з плаваючою точкою).
4. exclusiveMaximum: Число є дійсним для цього ключового слова, якщо воно строго нижче за значення цього ключового слова. Значення цього ключового слова має бути числом (цілим або з плаваючою точкою) або логічним значенням.
5. multipleOf: Число є дійсним для цього ключового слова, якщо ділення між числом і значенням цього ключового слова призводить до цілого числа. Значення цього ключового слова має бути строго позитивним числом (нуль не допускається).

3.3.3 Валідація типу Integer

Перелік функцій валідації для типу Integer [33]:

1. minimum: Число є дійсним для цього ключового слова, якщо воно більше або дорівнює значенню цього ключового слова. Значення цього ключового слова має бути цілим числом.
2. exclusiveMinimum: Число дійсне для цього ключового слова, якщо воно строго більше за значення цього ключового слова. Значення цього ключового слова має бути цілим числом або логічним значенням.
3. maximum: Число дійсне для цього ключового слова, якщо воно менше або дорівнює значенню цього ключового слова. Значення цього ключового слова має бути цілим числом.
4. exclusiveMaximum: Число є дійсним для цього ключового слова, якщо воно строго нижче за значення цього ключового слова. Значення цього ключового слова має бути цілим числом або логічним значенням.
5. multipleOf: Число є дійсним для цього ключового слова, якщо ділення між числом і значенням цього ключового слова призводить до цілого

числа. Значення цього ключового слова має бути строго позитивним числом (нуль не допускається).

3.3.4 Валідація типу Array

Перелік функцій валідації для типу Array [34]:

1. minItems: Масив є дійсним для цього ключового слова, якщо кількість елементів, які він містить, перевищує значення цього ключового слова або дорівнює йому. Значення цього ключового слова має бути цілим невід’ємним числом.
2. maxItems: Масив дійсний для цього ключового слова, якщо кількість елементів, які він містить, менша або дорівнює значенню цього ключового слова. Значення цього ключового слова має бути цілим невід’ємним числом.
3. uniqueItems: Масив є дійсним для цього ключового слова, якщо елемент не можна знайти більше одного разу в масиві. Значення цього ключового слова має бути логічним. Якщо встановлено значення false, перевірка ключового слова ігноруватиметься.
4. contains: Масив є дійсним щодо цього ключового слова, якщо принаймні один елемент є дійсним щодо схеми, визначеної значенням ключового слова. Значення цього ключового слова має бути дійсною схемою.
5. minContains: Масив є дійсним щодо цього ключового слова, якщо кількість дійсних елементів, перевірених ключовим словом contains, є меншим за значення, визначеним цим ключовим словом. Значення цього ключового слова має бути додатним цілим числом.
6. maxContains: Масив є дійсним щодо цього ключового слова, якщо кількість дійсних елементів, перевірених ключовим словом contains, є більшим за значення, визначеним цим ключовим словом. Значення цього ключового слова має бути додатним цілим числом.

7. items: Масив дійсний щодо цього ключового слова, якщо елементи дійсні щодо відповідних схем, наданих значенням ключового слова. Значення цього ключового слова може бути: дійсна схема json, тоді кожен елемент має відповідати цій схемі або масив дійсних схем json, тоді кожен елемент має відповідати схемі, визначеній у тій самій позиції (індексу). Елементи, які не мають відповідної позиції (масив містить 5 елементів, а це ключове слово має лише 3), вважатимуться дійсними, якщо не присутнє ключове слово AdditionalItems, яке визначає дійсність.

3.3.5 Валідація типу Object

Перелік функцій валідації для типу Object [35]:

1. properties: Об’єкт є дійсним щодо цього ключового слова, якщо кожна властивість, присутня як в об’єкті, так і в значенні цього ключового слова, перевіряється на відповідність схемі. Значення цього ключового слова має бути об’єктом, де властивості мають містити дійсні схеми json. Перевіряються лише назви властивостей, які присутні як в об’єкті, так і в значенні ключового слова.
2. required: Об’єкт дійсний щодо цього ключового слова, якщо він містить усі імена властивостей (ключі), визначені значенням цього ключового слова. Значення цього ключового слова має бути непорожнім масивом рядків, що представляють імена властивостей.
3. dependentRequired: Об’єкт є дійсним щодо цього ключового слова, якщо він відповідає всім залежностям, визначеним цим значенням ключового слова. Значення цього ключового слова має бути об’єктом, де значення властивостей мають бути масивами рядків, що представляють імена властивостей, а об’єкт має містити всі імена властивостей. Перевіряються лише назви властивостей (із значення цього ключового слова), які також присутні в об’єкті.

4. minProperties: Об’єкт є дійсним щодо цього ключового слова, якщо кількість властивостей, які він містить, перевищує значення цього ключового слова або дорівнює йому. Значення цього ключового слова має бути цілим невід’ємним числом. Використання 0 як значення без ефекту.
5. maxProperties: Об’єкт є дійсним щодо цього ключового слова, якщо кількість властивостей, які він містить, менша або дорівнює значенню цього ключового слова. Значення цього ключового слова має бути цілим невід’ємним числом. Використання 0 як значення означає, що об’єкт має бути порожнім (без властивостей).
6. propertyNames: Об’єкт є дійсним щодо цього ключового слова, якщо кожне ім’я властивості (ключ) є дійсним щодо значення цього ключового слова. Значення цього ключового слова має бути дійсною схемою json.
7. patternProperties: Об’єкт є дійсним щодо цього ключового слова, якщо кожна властивість, де ім’я властивості (ключ) відповідає регулярному виразу зі значення цього ключового слова, також є дійсним щодо відповідної схеми. Значення цього ключового слова має бути об’єктом, де ключі мають бути дійсними регулярними виразами, а відповідні значення мають бути дійсними схемами json.

3.4 Опис роботи з помилками

Бібліотека має власно визначену структура, які реалізовує інтерфейс **error**, це єдина умова для того, щоб нова структура могла бути використана як помилка.

Оскільки бібліотека приймає будь-який тип (any) як схему та як ціль, яку буде провалідовано, передбачено всі можливі дії користувача, які можуть призвести до помилки. При таких діях користувач отримає помилку, яка чітко описує, які вимоги були порушені.

Це також стосується не правильних вхідних параметрів. Якщо не вийде провалідувати схему, користувач отримає помилка, яка чітко опише яка функцію валідації має не правильні вхідні параметри та чому.

Якщо було надано валідну схему та ціль для валідації, тоді можливі помилки саме через не відповідність цілі до схеми. В такому випадку помилка буде містити назву функції валідації, яке значення очікується та яке реальне значення містить ціль.

Висновки до розділу 3

У третьому розділі роботи описано алгоритм для валідації схеми, цілі та самої валідації цілі при правильній схемі. Бібліотека працює з будь-яким значенням, щоб надати користувачу легкість роботи. Всі накладні витрати на валідацію цілі та схеми бібліотека виконує сама.

Бібліотека підтримує всі основні функції валідації для всіх доступних типів.

Бібліотека правильно повідомить користувачу про будь-яку помилку, яка виникла в процесі валідації схеми, в процесі валідації цілі чи в процесі безпосередньої валідації цілі відповідно до схеми.

ВИСНОВКИ

У роботі було розглянуто всі особливості, які пов’язані з веброзробкою, які підтверджують актуальність теми, а саме:

- Вибір мови програмування: мова Golang швидко набирає популярність завдяки своїм перевагам саме при розробці вебзастосунку. Це підтверджується збільшенню кількості вакансій відносно загальній кількості на відповідних ресурсах.
- Вибір способу комунікації між клієнтською та серверною частинами застосунку: REST API – найпопулярніше архітектурне рішення, яке описує спосіб комунікації між різними частинами застосунку.
- Вибір формату даних для REST API: JSON є найчастішим вибором серед користувачів та розробників. Цей формат є добре знайомий майже всім розробникам і вважається стандартом. Інші формати використовуються в окремо взяти випадках, коли інший формат має чіткі переваги над JSON та команда має досвід використання даного формату.

В ході виконання роботи було розроблено бібліотеку, яка валідує JSON-документи. Бібліотека підтримує всі типи та всі основні функції валідації, які передбачені JSON-документом. Бібліотека приймає будь-який тип (any), як вхідні параметри. Вхідні параметри можуть бути наступними типами: рядок (JSON, шлях до файлу або шлях до вебресурсу), масив байтів (JSON) або структура, в цьому випадку бібліотека за допомогою стандартних засобів мови перетворить об’єкт на JSON (в цьому випадку будуть використані тільки публічні поля структури та відповідні теги для всіх полів структури). Схема та ціль також проходять відповідну валідацію перед початком валідації самої цілі відповідно до схеми.

Бібліотека правильно обробляє всі можливі помилки, які можуть виникнути на будь-якому етапі валідації. Під час валідації схеми, також відбувається валідація всіх вхідних параметрів функцій валідації. При не правильних вхідних параметрів, бібліотека поверне помилку, де буде чітко зазначено, яка функція має не правильні вхідні аргументи та які саме аргументи вона очікує отримати. Під час валідації цілі до коректної схеми, бібліотека може повернути помилку, де буде зазначено, яке поле не пройшло валідацію, яка функція не пройшла валідацію, яке значення було отримано та яке значення очікувалось.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. XML [Електронний ресурс] https://en.wikipedia.org/wiki/XML
2. JSON [Електронний ресурс] https://en.wikipedia.org/wiki/JSON
3. XSD [Електронний ресурс] https://en.wikipedia.org/wiki/XML_Schema_(W3C)
4. What is JSON Schema? [Електронний ресурс] https://json-schema.org/overview/what-is-jsonschema#what-is-json-schema
5. Standard library – Golang [Електронний ресурс] https://pkg.go.dev/std
6. An implementation of JSON Schema for the Go programming language [Електронний ресурс] https://github.com/xeipuuv/gojsonschema
7. major api improvement [Електронний ресурс] https://github.com/xeipuuv/gojsonschema/issues/374
8. A tool to generate Go data types from JSON Schema definitions. [Електронний ресурс] https://github.com/omissis/go-jsonschema
9. Go (programming language) [Електронний ресурс] https://en.wikipedia.org/wiki/Go_(programming_language)
10. How to Connect Front End and Backend [Електронний ресурс] https://www.geeksforgeeks.org/how-to-connect-front-end-and-backend/
11. REST API Introduction [Електронний ресурс] https://www.geeksforgeeks.org/rest-api-introduction/
12. RFC5789 [Електронний ресурс] https://datatracker.ietf.org/doc/html/rfc5789
13. HTTP [Електронний ресурс] https://en.wikipedia.org/wiki/HTTP
14. WebSocket [Електронний ресурс] https://en.wikipedia.org/wiki/WebSocket
15. RFC6455 [Електронний ресурс] https://datatracker.ietf.org/doc/html/rfc6455

16. RFC7692 [Електронний ресурс] https://datatracker.ietf.org/doc/html/rfc7692
17. Universally unique identifier [Електронний ресурс] https://en.wikipedia.org/wiki/Universally_unique_identifier
18. Server-side scripting [Електронний ресурс] https://en.wikipedia.org/wiki/Server-side_scripting
19. GraphQL [Електронний ресурс] https://en.wikipedia.org/wiki/GraphQL
20. RFC4627 [Електронний ресурс] https://datatracker.ietf.org/doc/html/rfc4627
21. RFC8529 [Електронний ресурс] https://datatracker.ietf.org/doc/html/rfc8529
22. RFC7158 [Електронний ресурс] https://datatracker.ietf.org/doc/html/rfc7158
23. IEEE 754 [Електронний ресурс] https://en.wikipedia.org/wiki/Double-precision_floating-point_format
24. RFC7303 [Електронний ресурс] https://datatracker.ietf.org/doc/html/rfc7303
25. RFC3023 [Електронний ресурс] https://datatracker.ietf.org/doc/html/rfc3023
26. RFC3470 [Електронний ресурс] https://datatracker.ietf.org/doc/html/rfc3470
27. YAML [Електронний ресурс] https://en.wikipedia.org/wiki/YAML
28. RFC822 [Електронний ресурс] https://datatracker.ietf.org/doc/html/rfc822
29. RFC2046 [Електронний ресурс] https://datatracker.ietf.org/doc/html/rfc2046
30. Protocol Buffers [Електронний ресурс] https://en.wikipedia.org/wiki/Protocol_Buffers
31. JSON Schema Reference [Електронний ресурс] https://json-schema.org/understanding-json-schema/reference#json-schema-reference

32. JSON-Shema string [Електронний ресурс] https://json-schema.org/understanding-json-schema/reference/string
33. JSON-Shema Numeric types [Електронний ресурс] https://json-schema.org/understanding-json-schema/reference/numeric#numeric-types
34. JSON-Shema array [Електронний ресурс] https://json-schema.org/understanding-json-schema/reference/array
35. JSON-Shema object [Електронний ресурс] https://json-schema.org/understanding-json-schema/reference/object

ДОДАТОК 1 – Приклад помилок при валідації

```
Testing duration
error duration.error.txt failed to validate format; got: 20, expected: duration
successful duration.txt
Testing email
error email.error.txt failed to validate format; got: asd, expected: email
successful email.txt
Testing ipv4
error ipv4.error.txt failed to validate format; got: 127.0.0, expected: ipv4
successful ipv4.txt
Testing ipv6
error ipv6.error.txt failed to validate format; got: 2001:db8:68, expected: ipv4
successful ipv6.txt
Testing time
error time.error.txt failed to validate format; got: 20:20:aa, expected: time
successful time.txt
Testing uuid
error uuid.error.txt failed to validate format; got: 3e4666bf-d5e5-4aa7-b8ce-cefe41c7568, expected: uuid
successful uuid.txt
Testing maxLength
error maxLength.error.txt failed to validate maxLength; got: 4, expected: 3
successful maxLength.txt
Testing minLength
error minLength.error.txt failed to validate minLength; got: 2, expected: 3
successful minLength.txt
Testing pattern
error pattern.error.txt failed to validate pattern; got: (888)555-1212 ext. 532, expected: ^(\([0-9]{3}\))?[0-9]{3}-[0-9]{4}$
successful pattern.txt
```

```
go run cmd/main.go
Testing integer
Testing exclusiveMaximum
error exclusiveMaximum.error.txt failed to validate exclusiveMaximum; got: 10, expected: 10
successful exclusiveMaximum.txt
Testing exclusiveMinimum
error exclusiveMinimum.error.txt failed to validate exclusiveMinimum; got: 10, expected: 10
successful exclusiveMinimum.txt
Testing maximum
error maximum.error.txt failed to validate maximum; got: 10, expected: 11
successful maximum.txt
Testing minimum
error minimum.error.txt failed to validate minimum; got: 10, expected: 9
successful minimum.txt
Testing multipleOf
error multipleOf.error.txt failed to validate multipleOf; got: 3, expected: 2
successful multipleOf.txt
Testing number
Testing exclusiveMaximum
error exclusiveMaximum.error.txt failed to validate exclusiveMaximum; got: 10.5, expected: 10.5
successful exclusiveMaximum.txt
Testing exclusiveMinimum
error exclusiveMinimum.error.txt failed to validate exclusiveMinimum; got: 10.5, expected: 10.5
successful exclusiveMinimum.txt
Testing maximum
error maximum.error.txt failed to validate maximum; got: 10.6, expected: 10.5
successful maximum.txt
Testing minimum
error minimum.error.txt failed to validate minimum; got: 10.2, expected: 10.5
successful minimum.txt
Testing multipleOf
error multipleOf.error.txt failed to validate multipleOf; got: 10.2, expected: 0.5
successful multipleOf.txt
Testing object
Testing dependentRequired
error dependentRequired.error.txt failed to validate dependentRequired; got: , expected: c
successful dependentRequired.txt
Testing maxProperties
error maxProperties.error.txt failed to validate maxProperties; got: 3, expected: 2
successful maxProperties.txt
Testing minProperties
error minProperties.error.txt failed to validate minProperties; got: 1, expected: 2
```

```
Testing patternProperties
error patternProperties.error.txt failed to validate patternProperties; got: float64, expected: string
successful patternProperties.txt
Testing properties
error properties.error.txt failed to validate properties; got: float64, expected: string
successful properties.txt
Testing propertyNames
error propertyNames.error.txt failed to validate minLength; got: 1, expected: 2
successful propertyNames.txt
Testing required
error required.error.txt failed to validate required; got: , expected: b
successful required.txt
Testing slice
Testing contains
error contains.error.txt failed to validate contains; got: [string], expected: integer
successful contains.txt
Testing items
error items.error.txt failed to validate minimum; got: 0, expected: -2
successful items.txt
Testing maxContains
error maxContains.error.txt failed to validate maxContains; got: 3, expected: 2
successful maxContains.txt
Testing maxItems
error maxItems.error.txt failed to validate maxItems; got: 3, expected: 2
successful maxItems.txt
Testing minContains
error minContains.error.txt failed to validate minContains; got: 1, expected: 2
successful minContains.txt
Testing minItems
error minItems.error.txt failed to validate minItems; got: 2, expected: 3
successful minItems.txt
Testing uniqueItems
error uniqueItems.error.txt failed to validate uniqueItems; got: elem: 1 duplicated, expected: unique
successful uniqueItems.txt
Testing string
Testing format
Testing date
error date.error.txt failed to validate format; got: 2018-11-aa, expected: date
successful date.txt
Testing date-time
error date-time.error.txt failed to validate format; got: 2018-11-13T:20:39+00:00, expected: date-time
successful date-time.txt
```