**КНИГА N2O**

### АНОТАЦІЯ

Друге видання КНИГИ N2O (англ. N2O BOOK Vol. 2 Green Book) визначає формальну специфікацію на програмне забезпечення усіх рівнів моделі Закмана для підприємств ISO-42010, містить широкий спектр прикладів, розказує про складові компоненти та є вичерпним авторським стартовим посібником для курсу навчання розробки технологічних програм для платформи Erlang.

### ПЕРЕДМОВА АВТОРА

КНИГА N2O визначає правила побудови WebSocket сервера, бінарного серіалізатора, протокольних розширень, додатків з використанням веб-фреймворку визначеному формальними протоколами. Промислові версії також підтримують систему управління бізнес-процесами BPMN та ефективне сховище з глобальним простором ключів SpanDB для використання блочних пристроїв NVMe через SPDK.

Нові версії посібників отримали емодзі номенклатуру (кодування підручників Smalltalk-80 Ади Голдберг та Глена Краснера) як і всі бібліотеки системи: КНИГА N2O отримала назву Green Book Vol.2 📗, КНИГА ERP отримала назву Blue Book Vol.3 📘, КГИГА MAD отримала назву Orange Book Vol.4 📙. Перші версії підручників отримали кодування Closed Book 📕 та підтримуватимуться для навчальних потреб та у якості архіву. N2O як і Smalltalk керується принципами мінімалізму та продуктивності, тому таке кодування є символічним.

В квітні 2017 року була випущена остання версія 4.4, яка підтримувала виключно WebSockets та не підтримувала MQTT протокол. Починаючи з першого релізу в жовтні 2013 року, N2O не губив сумісність протягом 4 років, і, ось уже протягом 4-х років не гублять сумісність між собою всі MQTT версії. Можна мене звинувачувати в тому, що випуск оновленої версії КНИГИ N2O 📗 дещо запізнілий, на що у якості виправдання відповім, що хотілося переконатися двома великими впровадженнями, що інтерфейси та протоколи стабілізувалися та не змінюватимуться як мінімум ще протягом тривалого експлуатаційного терміну.

КНИГА ERP 📘 розширилася описом модулів підприємства ERP.UNO, та містить навчальні приклади створення адміністраторів за допомогою веб-фреймворку NITRO для бібліотек BPE та KVS (адміністратор процесів та адміністратор сховища), а також показує загальну методологію дизайну та імплементації модулів підприємства. У той час коли переважна більшість користувачів використовує бібліотеки N2O описані в Vol.2 ми зосереджені на підтримці та покращенні якості модулів підприємства описаних в Vol.3.

КНИГА MAD 📙 тепер розказує не лише про MAD та системні бібліотеки SYN, FS, RT та BASE але і про основи віртуальної машини LAM написаної на Rust Леандро Остера, яка підтримує генерацію WebAssembly для native та web режимів. В другому виданні фокус на головну цільову платформу N2O переміщується з віртуальної LING до LAM. Якщо Vol.2 та Vol.3 зосереджені на прикладних бібліотеках та промислових прикладах для підприємств, то Vol.4 присвячений віртуальному середовищі виконання, його інтерпретатору, інструментальним засобам та системним бібліотекам.

**ВСТУП**

Глава ВСТУП розказує про мотивацію та структуру бібліотек N2O ТЕХ.

**МОТИВАЦІЯ**

N2O сервер, а також [NITRO](https://n2o.dev/ua/deps/nitro/index.html) веб-фреймворк були спроектовані як інструментальні засоби для створення промислових ERP модулів підприємства у складі відкритої платформи [ERP.UNO](https://erp.uno/ua/). Напочатку, N2O був відгалужений, як оптимізована версія веб-фреймворку Nitrogen, створеного Расті Клопгаузом. Хотілося оптимізувати та вдосконалити мінімізований WebSocket-тракт, який не містить синхронного протоколу HTTP взагалі та дозволяє створювати повноцінні асинхронні веб-додатки реального часу. На ньому була створена система управління депозитами в національному банку ПриватБанк. Пізніше N2O був розділений на бібліотеку-фреймворк процесів та протоколів (власне N2O) та бібліотеку-веб-фреймворк NITRO. Бібліотеки N2O та NITRO також отримали можливість роботи не тільки через WebSocket але і через MQTT та через чисті TCP або UDP. Така оновлена версія 5.10 була впроваджена як ядро системи повідомлень для додатку NYNJA з відкритим протоколом і саме їй присвячений друга версія підручника.

**БІБЛІОТЕКИ**

Для забезпечення повноцінної промислової специфікації ERP.UNO, ми розширили набір інструментальних засобів наступними бібліотеками: формальними представленнями презентаційного рівня FORM та системою управління бізнес-процесів BPE. FORM представляє собою декларативну бібліотеку побудови графічних інтерфейсів, а бібліотека BPE підтримує XML файли стандарту BPMN 2.0 та реалізує безпосередню інтерналізацію BPMN семантики у семантику віртуальної машини Erlang.

Ядро бібліотек які реалізують фундацію [N2O.DEV](https://n2o.dev/ua/) (організація SYNRC) для системи управління підприємствами ERP.UNO (організація ERPUNO) у версії 8.0 виглядає наступним чином (у другій колонці кількість рядків після форматування rebar3 format):

* ⭕ [N2O](https://n2o.dev/ua/deps/n2o/index.html) — 930 — сервер протоколів MQTT, WS, QUIC
* 🔥 [NITRO](https://n2o.dev/ua/deps/nitro/index.html) — 3800 — веб-фреймворк Nitrogen Расті Клопгауза
* 💿 [KVS](https://n2o.dev/ua/deps/kvs/index.html) — 600 — бібліотека доступу до KV сховищ
* 🧾 [FORM](https://n2o.dev/ua/deps/form/index.html) — бібліотека конструювання інтерфейсів
* 💠 [BPE](https://n2o.dev/ua/deps/bpe/index.html) — 980 — система управління процесами BPMN 2.0
* ☎️ [RPC](https://n2o.dev/ua/deps/rpc/index.html) — 550 — бібліотека SDK для JavaScript, protobuf, Swift

**ПРИКЛАДИ**

Головні приклади фундації N2O.DEV присвячені наступним темам: MQTT та WebSocket чати для демонстрації веб-фреймворку NITRO, який працює як модуль N2O, приклад REST адаптер до бази даних KVS, та повністю чистий N2O додаток CHAT на основі бібліотеки SYN без використання NITRO:

* 💧 [SAMPLE WS](https://sample.n2o.dev) — 120 — приклад Nitrogen/WS
* 💧 [REVIEW TT](https://review.n2o.dev) — 120 — приклад Nitrogen/MQTT
* ☕ [REST](https://n2o.dev/ua/deps/rest/index.html) — 800 — бібліотека для побудови HTTP API
* 💬 [CHAT](https://chat.n2o.dev) — 80 — ідіоматичний приклад доставки повідомлень

**ІНСТРУМЕНТИ**

Цей посібник більше присвячений бібліотекам N2O та NITRO, та лише незначним чином торкається сховища даних KVS. Пізніше нами було інвестовано ще у наступні бібліотеки, які в основному стосуються створення API та керування пакетами та экземплярами:

* ⚡ [MAD](https://n2o.dev/ua/deps/mad/index.html) — 1420 — бібліотека управління залежностями
* 📁 [FS](https://n2o.dev/ua/deps/fs/index.html) — 320 — бібліотека управління файловою системою
* 🐋 [ACTIVE](https://n2o.dev/ua/deps/active/index.html) — 190 — бібліотека перекомпіляції

Кожна бібліотека розкриватиметься у наступних навчальних посібниках, що вийдуть згодом.

**СЕРЕДОВИЩЕ**

Для забезпечення повного замкненого середовища пропонують наступні заміни бібліотек kernel та stdlib:

* 🚀 [VM](https://n2o.dev/ua/deps/vm/index.html) — 1700 — віртуальна машина середовища виконання
* ⭐ [BASE](https://n2o.dev/ua/deps/base/index.html) — базова системна бібліотека як заміна stdlib
* ⭐ [RT](https://n2o.dev/ua/deps/rt/index.html) — бібліотека середовища виконання як заміна kernel
* 📨 [SYN](https://hexdocs.pm/syn/readme.html) — 1700 — брокер для розподілених систем

Концептуальна модель системи в рамках якої функціонує N2O визначена як обчислювальне середовище, яке складається з процесору подій (N2O), операційного (ETS) та персистентного сховища (KVS). З точки зору обчислювального середовища, ресурси підприємства складаються з глобального сховища та обчислень, які розділяють глобальну адресацію та представляють собою Erlang-процеси (N2O протоколи). Кожен процес PI, може містити певний набір протоколів, будь-який з яких відповідає на певний набір повідомлень. Протоколи N2O визначені на точці підключення повинні не перетинатися, в іншому випадку протокольні модулі можуть перехоплювати та впливати на інші протокольні модулі, які повинні реагувати на той самий тип повідомлень.

**Обчислювальні ресурси N2O, та асинхронні процеси PI**

Усі асинхронні процеси PI запускаються під головним супервізором n2o та індексуються URI ключем разом з типом реактивного каналу реального часу: ws або mqtt. N2O протоколи підключені безпосередньо до веб-сокет точок підключення виконуються в контексті TCP процесів, у даному випадку TCP-сервера бібліотеки RANCH, супервізор ranch\_sup.

> :kvs.all :writer

[ {:writer, '/bpe/proc', 2, [], [], []},

{:writer, '/erp/group', 1, [], [], []},

{:writer, '/erp/partners', 7, [], [], []},

{:writer, '/acc/synrc/Kyiv', 3, [], [], []},

{:writer, '/chat/5HT', 1, [], [], []},

{:writer, '/bpe/hist/1562187187807717000', 16, [], [], []},

{:writer, '/bpe/hist/1562192587632329000', 1, [], [], []}

]

**Персистентні ресурси сховища даних KVS**

Для відображення усіх таблиць (префіксів) які існують в глобальному просторі ключів, скористайтеся системним фідом writer.

> :kvs.all :writer

[

{:writer, '/bpe/proc', 2, [], [], []},

{:writer, '/erp/group', 1, [], [], []},

{:writer, '/erp/partners', 7, [], [], []},

{:writer, '/acc/synrc/Kyiv', 3, [], [], []},

{:writer, '/chat/5HT', 1, [], [], []},

{:writer, '/bpe/hist/1562187187807717000', 16, [], [], []},

{:writer, '/bpe/hist/1562192587632329000', 1, [], [], []}

]

**ЕРЛАНГ ТА СУЧАСНИЙ ВЕБ**

Erlang реалізує недосяжну мрію кожного обчислювального середовища для паралельної та узгодженої конкурентної обробки повідомлень. Так найбільш відомі бібліотеки акторів (Akka, Orleans), які реалізують основні примітиви: процесори та черги, копіюють модель акторів Erlang, зазвичай намагаються також реалізують додатково механізми перезапуску та супервізії процесів подібно до Erlang, проте тільки Erlang забезпечує soft real-time характеристики, завдяки керуванню латенсі з точністю до таймінгу команд віртуальної машини. А з виходом 24 версії в 2020 році, яка почала підтримувати JIT-компіляцію завдяки asmjit, продуктивність та чуттєвість віртуальної машини зросла ще більше.

З формальної точки зору достатньо добре ізольоване середовище віртуальної машини Erlang не тільки забезпечує характеристики реального часу для SMP-планувальника легких зелених процесів, але і обмежує область видимості heap пам'яті виключно для процесів-власників, що унеможливлює вплив відмови певних процесів на глобальний стан віртуальної машини.

Erlang ідеально підходить для побудови високо-навантажених, просто-масштабованих, подійно-орієнтованих, неблокуючих, надійних, постійно-доступних, високо-ефективних, швидких, безпечних та надійних систем обробки повідомлень та розподілених у просторі та часі систем.

**DSL VS ШАБЛОНИ**

З технічної точки зору N2O успішно показує неперевершену досі якість DSL програмування, яку ви не зможете знайти в сучасних веб-фреймворках для мов Erlang та Elixir. За 7 років неперервної еволюції N2O ми переписали кожен з 700 рядків по 30 разів, якшо порахувати через коміти Github. Веб-фреймворк NITRO, сховище KVS, та BERT.JS кодування може забезпечити відображення в веб-браузері повноекранних вертикальних форм з усіма обчислюваними полями зі швидкістю 60 форм в секунду по веб-сокет каналу. А надзвичайно компактна JavaScript бібліотека-компаньйон вміщується в 4 MSS/MTU вікна — саме такий розмір мінімального веб-клієнта з BERT кодуванням, який повністю управляється зі сторони сервера.

N2O сервер та веб-фреймворк NITRO реалізують концепцію не тільки управління сесіями та каналами, але і усім стеком побудови додатків включаючи UI частину, як це відбувається у таких веб-фреймворках як Erlang Nitrogen, OCaml Ocsigen, Scala Lift, F# WebSharper, а завдяки таким розширенням як FORM та BPE ідеально підходять і для побудови автоматизованих CRM систем.

Це не означає, що за допомогою N2O ви не можете створювати більш класичні та архаїчні додатки у стилі DTL шаблонізаторів, або як це відбувається у таких фреймворках як PHP, ASP, JSP, Rails, тощо. Перші версії NITRO містили в прикладах використання Django Template Library (DTL), проте задля чистоти стеку були прийнято не включати в N2O додаткові шаблонізатори крім NITRO DSL.

**КЛІЄНТ**

ECMAScript клієнтська інфраструктура для усіх рівнів N2O специфікації.

* [UTF-8.JS](https://n2o.dev/ua/deps/n2o/man/utf8.js.htm) — UTF-8 серіалізація
* [IEEE-754.JS](https://n2o.dev/ua/deps/n2o/man/ieee754.js.htm) — Бінарний формат дійсних чисел
* [ZLIB.JS](https://n2o.dev/ua/deps/n2o/man/zlib.js.htm) — ZLIB декомпресія термів
* [ETF.JS](https://n2o.dev/ua/deps/n2o/man/bert.js.htm) — Erlang Term Format серіалізатор
* [N2O.JS](https://n2o.dev/ua/deps/n2o/man/n2o.js.htm) — N2O/WebSocket JavaScript протокольний цикл
* [MQ.JS](https://n2o.dev/ua/deps/n2o/man/mq.js.htm) — N2O/MQTT JavaScript протокольний цикл
* [NITRO.JS](https://n2o.dev/ua/deps/n2o/man/nitro.js.htm) — NITRO/JavaScript біндінги для DOM джерел
* [VALIDATION.JS](https://github.com/synrc/nitro/blob/master/priv/js/validation.js) — NITRO/JavaScript валідації DOM джерел

**ІНШІ МОВИ ПРОГРАМУВАННЯ**

N2O TEX — це формальна філософія та інженерна вправа водночас. Вона обмежує автора бути ефективним та точним не втрачаючи при цьому повноти та функціональності. Це на кшталт внутрішньої дисципліні при проектуванні програмного забезпечення. Ця філософія більше 8 років застосовується на практиці для побудови систем SYNRC, та визначає стандартний мінімальний набір для демонстрації однієї з сучасних моделей реактивного веб-програмування, яка включає: веб-сокет веб-фреймворк з бінарною серіалізацією, пушами та контролем DOM елементів зі сторони сервера. N2O вчить будувати прості та надійні системи на будь-якій мові програмування. Авторські імплементації N2O TEX для функціональних мов програмування:

Мови програмування розділені на чотири рівня як для клієнтської (мобільної та веб розробки) так і для серверної розробки (бекендів системи). Нульовий рівень — тотальні формальні алгебраїчні мови програмування, що забезпечують повноту, функціональність та доведення властивостей програм згідно сучасних уявлень про математичне моделювання та системи залежних типів побудованих на розшаруваннях. Перший рівень — формальні функціональні мови програмування, як правило System F, System Fω які успішно використовуються в промисловості та забезпечують достатньо формальний запис який піддається масштабуванню у великих командах завдяки потужному ядру компіляторів. Другий рівень — неформальні (без формальної операційної чи денотаційної семантики) чи формальних верифікаторів, які проте успішно використовуються в промисловості, можуть бути як з розвиненими системами типів з узагальненими шаблонами та типами сумами, так і однотипними мовами програмування з динамічною типізацією. Третій рівень — мови які погано піддаються масштабуванню у промисловому виробництві (на основі спостережень за власним досвідом).

* Client Tier 3：（JavaScript, Lua）
* Client Tier 2：（Swift, Kotlin, TypeScript）
* Client Tier 1：（UrWeb, OCaml, PureScript）
* Client Tier 0：（Formality）
* Server Tier 3：（PHP, Python, Perl, Ruby）
* Server Tier 2：（Erlang, Elixir）
* Server Tier 1：（Standard ML, Haskell, F#, Rust）
* Server Tier 0：（Coq, Agda, Lean）

Філософія N2O визначає структуру операційних середовищ (runtimes) як операційну систему лямбда-інтерпретаторів які працюють на паралельному обчислювальному середовищі (ядрах процесорів). Кожне з ядер процесорів виконує в нескінченному циклі команди лямбда-інтерпретаторів, переключаючи через певний проміжок часу на потік команд іншого інтерпретатора. Таке визначення дає змогу вбудувати цю структуру у віртуальну машину Erlang.

* Головний процес додатку
* Супервізор додатку
* Проміжні супервізори
* Кінцеві пул процесорів повідомлень

**ПОРАДА**

Кожна мова, на якій реалізовано N2O ТЕХ, має вбудовувати свою філософію максимально природньо та компактно. Якщо потрібен якийсь шар між базовою бібліотекою мови, його можна надати, але, якщо можливо, його слід зменшити до нуля. У деяких випадках деякі частини базової бібліотеки можна замінити кращим аналогом. N2O має надати клієнтську бібліотеку-компаньйон, зазвичай реалізовану на іншому наборі мов клієнта: JavaScript, Swift, Kotlin. Якщо ви все зробили правильно, значення N2O не повинно перевищувати 500 LOC на будь-якій мові.

# ПРИКЛАДИ

### ПРЕРЕКВІЗИТИ

Для розгортання N2O SAMPLE вам знадобиться остання версія Ерланг з пакетного менеджера вашої ОС. N2O SAMPLE сумісний з операційними системами Ubuntu (apt), Windows (WSL), Android (termux), та MacOS (brew), Arch (pacman).

$ sudo apt install erlang elixir build-essential cmake git

Оскільки філософія N2O мотивована мінімалізмом, природнім є використання спартанського сетапу для розробника: ванільні редактори vim, emacs чи mc.

$ sudo apt emacs vim mc

### ПРИКЛАДИ

Перша видання книги "N2O: No Bullshit Sane Framework for Wild Web" містила лише ідіоматичний приклад [SAMPLE] використання Nitrogen веб-фреймворку. КНИГА N2O (N2O BOOK Vol. 2) містить розширений набір прикладів [CHAT, REST, REVIEW, SAMPLE, PLM, CRM] який демонструє широкий спектр технічних засобів для автоматизації процесів на виробництві.

Також, перше видання книги містило приклади Erlang синтаксису у той час як КНИГА N2O містить програми на мові Elixir. Незважаючи на це усі бібліотеки є сумісними з обома мовами, а деякі приклади [CHAT,SAMPLE] реалізовані окремо для двох мов одночасно, щоб користувачі могли порівняти синтаксис Erlang та Elixir.

Перевірка працездатності кожного прикладу здійснюється за допомогою:

$ mix deps.get

$ iex -S mix

#### CHAT

Приклад CHAT був створений як ідіоматична мінімальна демонстрація системи доставки повідомлень з in-memory брокером SYN та сховища даних RocksDB. Брокер використовується на стороні сесії та підключення до WebSocket (клієнт), саме в сесійній частині здійснюється підписка та публікація-розсилка повідомлень. Протокол N2O, який реалізує додаток CHAT містить всього дві функції: зберігання повідомлення в KVS/RocksDB сховище з послідуючою розсилкою нотифікації процесу-сесії, та очищення скриньки повідомлень певного користувача.

Брокер SYN це швидка система доставки повідомлень по підпискам (PubSub) побудована на ETS таблицях середовища виконання Erlang. Відомі аналоги для платформи Erlang: gproc, pg, pg2, global; для кожного з них можна реалізувати свій модуль MQ та визначити його змінною mq в налаштуваннях додатку n2o.

**Використані бібліотеки:**

[

{:cowboy, "~> 2.8"},

{:rocksdb, "~> 1.6.0"},

{:syn, "~> 2.1.1"},

{:n2o, "~> 8.8.1"},

{:kvs, "~> 8.10.4"}

]

**Налаштування бібліотек N2O та KVS:**

config :n2o,

port: 8042,

proto: CHAT.Server,

mqtt\_services: [],

pickler: :n2o\_secret,

mq: :n2o\_syn,

upload: "./priv/static",

protocols: [:n2o\_heart, CHAT.TXT]

config :kvs,

dba: :kvs\_rocks,

dba\_st: :kvs\_st,

schema: [:kvs, :kvs\_stream, CHAT]

**Демонстрація використання:**

$ wscat --no-check -c wss://localhost:8042/ws

> HELP < N2O

| SEND

| BOX

| CUT

.

> N2O maxim

< USER maxim

> SEND vlad HELO

< ERROR user doesn't exist.

> N2O vlad

< USER vlad

> SEND maxim OK

< NOTIFY vlad:maxim:1556151953113322286:OK

< ACK "1556151953113322286"

> N2O maxim

< USER maxim

> BOX

< LIST vlad:maxim:1556151953113322286:OK

> SEND maxim this is me

< NOTIFY maxim:maxim:1556152151055371152:this is me

< ACK "1556152151055371152"

> SEND maxim back again

< NOTIFY maxim:maxim:1556152157283311935:back again

< ACK "1556152157283311935"

> BOX

< LIST

vlad:maxim:1556151953113322286:OK

maxim:maxim:1556152151055371152:this is me

maxim:maxim:1556152157283311935:back again

> CUT 1556152157283311935

< ERASED 3

> BOX

< LIST

>

#### SAMPLE

**Використані бібліотеки:**

[

{:cowboy, "~> 2.8.0"},

{:rocksdb, "~> 1.6.0"},

{:n2o, "~> 8.8.1"},

{:syn, "~> 2.1.1"},

{:kvs, "~> 8.10.4"},

{:nitro, "~> 6.11.4"}

]

**Налаштування бібліотек N2O та KVS:**

config :n2o,

port: 8002,

pickler: :n2o\_secret,

mq: :n2o\_syn,

upload: "./priv/static",

protocols: [:nitro\_n2o, :n2o\_ftp],

routes: Sample.Routes

config :kvs,

dba: :kvs\_rocks,

dba\_st: :kvs\_st,

schema: [:kvs, :kvs\_stream]

**Демонстрація використання:**

<main>

<form></form>

<textarea id="message" rows="4" autofocus="true"

placeholder="Just type what you think about this"></textarea>

<button id="upload">Browse</button>

<button id="send">chat</button>

<history id="history"></history>

</main>

У цьому прикладі наведена проста ініціалізація HTML контрольного елементу постбеком login, в процесі якого (після натискання кнопки), на сервер буде відправлений зміст контрольних елементів з іменами user та room. На сервері реакція на цю подію буде у вигляді редіректу на інший контролер сторінки index, з параметром кімнати room.

def event(:init) do room = Sample.Application.room KVS.ensure(writer(id: room)) N2O.reg({:topic, room}) N2O.reg(N2O.sid()) NITRO.update(:upload, upload()) NITRO.update(:heading, h2(id: :heading, body: room)) NITRO.update(:logout, button(id: :logout, postback: :logout, body: "Logout")) NITRO.update(:send, button(id: :send, body: "Chat", postback: :chat, source: [:message])) room |> KVS.all() |> Enum.each(fn {:msg, \_, user, message} -> event({:client, {user, message}}) end) end def event(:logout) do N2O.user([]) NITRO.redirect("/app/login.htm") end def event(:chat) do chat(NITRO.q(:message)) end def event({:client, {user, message}}) do NITRO.wire(jq(target: :message, method: [:focus, :select])) NITRO.insert\_top(:history, message(body: [author(body: user), NITRO.jse(message)])) end def chat(message) do room = Sample.Application.room user = N2O.user() room |> KVS.writer() |> writer(args: {:msg, KVS.seq([], []), user, message}) |> KVS.add() |> KVS.save() N2O.send({:topic, room}, client(data: {user, message})) end $ open http://localhost:8001/app/index.htm

Graphical user interface, application

Description automatically generated

#### PLM

**Використані бібліотеки:**

[

{:kvs, "~> 6.7.7"},

{:n2o, "~> 6.8.1"},

{:nitro, "~> 4.7.7"},

{:cowboy, "~> 2.5.0"},

{:rocksdb, "~> 1.2.0"},

{:chat, "~> 3.8.1"},

{:syn, "~> 1.6.3"},

{:erp, "~> 0.7.17"},

{:bpe, "~> 4.7.5"},

{:form, "~> 4.7.0"}

]

**Налаштування бібліотек N2O, KVS, FORM та SCHEMA:**

config :n2o, pickler: :n2o\_secret, mq: :n2o\_syn, port: 8043, ttl: 60\*3, proto: CHAT.Server, nitro\_prolongate: true, mqtt\_services: [:erp, :plm], ws\_services: [:chat], upload: "./priv/static", protocols: [:n2o\_heart, :n2o\_nitro, CHAT.TXT, :n2o\_ftp], routes: PLM.Routes config :schema, boot: [:erp\_boot, :plm\_boot, :acc\_boot, :pay\_boot, :fin\_boot] config :kvs, dba: :kvs\_rocks, dba\_st: :kvs\_st, schema: [:kvs, :kvs\_stream, :bpe\_metainfo] config :form, registry: [ LDAP.Forms.Credentials, LDAP.Forms.Access, BPE.Forms.Create, BPE.Rows.Trace, BPE.Rows.Process, FIN.Rows.Account, FIN.Rows.Transaction, PLM.Forms.Error, PLM.Rows.Payment, PLM.Rows.Investment, PLM.Rows.Product ]

**Демонстрація використання:**

A picture containing table

Description automatically generated

# ПРОТОКОЛИ

### ПРОЦЕСИ

N2O — це вбудовувана бібліотека протокольних циклів обробки повідомлень для WebSocket, MQTT та TCP серверів. До базової функціональності бібліотеки входять: управління процесами; кільця віртуальних вузлів для обробки запитів; сервіси сесій, кодування, шини та кешування.

N2O надає спосіб створення, налаштування і запуску довільних додатків та протоколів всередині деяких хостів, в які N2O може бути включений, таких, як cowboy та emqttd. Кожен додаток також може створювати свої сервісні протоколи-процеси, так, як веб-сторінки створюють WebSocket з'єднання, системи управління бізнес-процесами створюють екземпляри процесів, або як чат-додатки створюють процеси управління кімнатами чи персональними чатами. З N2O все управляється протоколами.

N2O поставляється для роботи в двох режимах: 1) всередині n2o\_mqtt воркерів; 2) всередині cowboy процесів, реалізованих в n2o\_stream. У першому випадку, сервер MQTT використовується між клієнтами та серверними воркерами. У другому випадку, з Erlang процесів задіяні лише клієнти. Ви можете створити власну конфігурацію циклу обробки N2O.

Сам по собі, N2O — це вбудовуваний протокольний цикл n2o\_proto, який ви можете включати у свої продукти. Орім цього, він обробляє кеш та сесії з гнучкими n2o\_pi процесами без обмеження права власності. А також визначає AES/CBC—128 кодування та BERT/JSON декодер.

### ТОЧКИ ПІДКЛЮЧЕННЯ

Перелік типів кінцевих точок підключення протоколів, які підтримуються EMQ, та є доступними для N2O програм: WebSockets, MQTT, MQTT-SN, TCP, UDP, CoAP. Нормальним є використання N2O у якості веб-фреймворку або сервера веб додатків з передачею даних по WebSockets, але для IoT та MQTT програм можна використовувати UDP чи SCTP протоколи, для забезпечення консистентності доставки повідомлень на рівні програми. Використовуючи MQTT в якості транспорту ми розширюємо підтримуваний набір протоколів кінцевих точок підключення.

#### RPC MQTT

N2O надає RPC через механізм MQ для MQTT пристроїв. N2O запускає набір n2o\_mqtt воркерів — n2o\_pi процесів, які слухають певні топіки певної кімнати. Відповіді надсилаються до теми подій, яка автоматичного підписана при старті сесії MQTT.

Цитата 5. MQTT RPC Топіки

actions/:vsn/:module/:client events/:vsn/:node/:module/:client

#### RPC WebSocket

У випадку чистого WebSocket, в N2O реалізовано n2o\_stream як cowboy модуль, що підтримує бінарні та текстові повідомлення.

Цитата 6. Cowboy stream протокол

#binary { data :: binary() }.

#text { data :: binary() }.

### ПРОТОКОЛИ

Хоча всі протоколи додатків в системі вимагають середовище з одним ефектом, або з таким же шляхом обробки помилок, **n2o** визначає єдиний протокольний цикл, як стек протоколів, для всіх додатків.

В базовому варіанті **n2o** містить протоколи NITRO та FTP, що дозволяє створювати працюючі в режимі реального часу веб-додатки, з протоколами на основі бінарних, а також — надійний та продуктивний клієнт для вивантаження файлів, разом з протоколом передачі файлів. Для створення NITRO веб-додатків, вам необхідно підключити nitro в залежності.

#### info(term(),term(),#cx{}) -> #reply{} | #unknown{}.

**info/3** — функція зворотнього виклику (колбек) N2O протоколу, яка викликається при кожному вхідному запиті.

#### HEART

#### NITRO

#### FTP

#### BPE

#### KVS

# ІНТЕРФЕЙС

### СТОРІНКИ

### ЕЛЕМЕНТИ

Nitrogen Elements – це двигун шаблонів HTML для мови Erlang, у якому всі HTML теги рендеряться з Erlang рекордів.

Працюючи з N2O вам взагалі не потрібно користуватись HTML. Натомість ви визначаєте вашу сторінку у вигляді Erlang рекордів, відповідно сторінка генерується, з перевіркою типів, під час компіляції. Це класичний підхід CGI для компільованих сторінок, який надає всі переваги перевірки помилок під час компіляції, та визначає DSL для клієнт- та серверного рендерингу.

Nitrogen Elements, за своєю природою, є примітивними UI елементами управління, які можуть бути використані для побудови Nitrogen сторінок з внутрішнім DSL Erlang-а. Вони компілюються в HTML та JavaScript. Поведінка всіх елементів контролюється на стороні сервера, а весь зв'язок між веб-переглядачем та сервером здійснюється за допомогою WebSocket каналів. Отже, вам не потрібно використовувати POST запити чи HTML форми. Тобто:

#textbox { id=userName, body= <<"Anonymous">> },

#panel { id=chatHistory, class=chat\_history }

згенерує наступний HTML

<input value="Anonymous" id="userName" type="text"/>

<div id="chatHistory" class="chat\_history"></div>

### ПОДІЇ

Nitrogen Actions – це двигун шаблонів JavaScript подій для мови Erlang, у якому всі події та обробники кнопок рендеряться з мови Erlang.

nitro:update(loginButton, #button{id=loginButton, body="Login",postback=login,source=[user,pass]});

Це згенерує наступний HTML:

<input value="Login" id="loginButton" type="button"/>

та JavaScript код:

qi('loginButton').outerHTML='<button id=\"loginButton\" type=\"button\">Login</button>';{var x=qi('loginButton'); x && x.addEventListener('click',function (event){ event.preventDefault(); { if (validateSources(['user','pass'])) { ws.send(enc(tuple(atom('pickle'),bin('loginButton'), bin('b840bca20b3295619d1157105e355880f850bf0223f2f081549dc 8934ecbcd3653f617bd96cc9b36b2e7a19d2d47fb8f9fbe32d3c768866 cb9d6d85700416edf47b9b90742b0632c750a4240a62dfc56789e0f5d8 590f9afdfb31f35fbab1563ec54fcb17a8e3bad463218d6402f1304'), [tuple(tuple(string('loginButton'),bin('detail')),[]), tuple(atom('user'),querySource('user')), tuple(atom('pass'),querySource('pass'))]))); } else console.log('Validation Error'); }});};

### РЕНДЕРИНГ

# СХОВИЩЕ

### ІТЕРАТОРИ

KVS — це бібліотека абстракції на KV сховищами (з єдиним простором ключів), що складається з двох частин: базового API та API управління ітераторами: take, drop, next, prev, append. Основне API в модулі kvs, а ітераторів в kvs\_stream.

#### Призначення KVS

— Надання інтерфейсу абстрагування широкого спектру сховищ;

— Надання зручного Ерланг REPL інтерфейсу для роботи з рекордами;

— Поділ на базовий (get put delete) та розширений інтерфейс ітераторів;

— Набір драйверів (внутрішня база, зовнішня база і файлова система);

— Шари даних: файлова система, ланцюжки повідомлень, банківські транзакції, дерева підписів, трейси бізнес-процесів, блокчейни, системи черг, часові послідовності.

Бібліотека KVS дозволяє або повинна дозволяти зберігати та витягувати будь-якого виду структури надаючи семантику управління курсорами next prev, якою володіють дерева. Тому можна сказати, що це інтерфейс оператора до ланцюжкових та деревоподібних сховищ. У своїй основі KVS підтримує три механізми зберігання ланцюжків: 1) перший спосіб, двозв'язні списки, явний, де next і prev покажчики безпосередньо присутні в даних, підходить навіть для управління деревами; 2) другий, явний, де є тільки next, цей спосіб підходить для списків, багато людей запитують про цей спосіб, але ми його не використовували ніколи та імплементації під рукою немає; 3) і третій, явний спосіб, де записи вбудовуються у існуючий індекс, тобто. btree таблиці, zero-overhead.

В якомусь сенсі перший і другий способи реалізують певний шар поверх KVS, так як kvs\_stream працює з будь-якими сторожами поліморфно, а ось kvs\_st драйвер ітераторів зроблено спеціально для rocksdb.

### ROCKSDB

1> {ok,Ref} = rocksdb:open("hey",[{create\_if\_missing,true}]). 2> rocksdb:put(Ref, <<"/users/1">>,<<"maxim">>,[{sync,true}]). 3> rocksdb:put(Ref, <<"/users/2">>,<<"doxtop">>,[{sync,true}]). 4> rocksdb:put(Ref, <<"/users/3">>,<<"vlad">>,[{sync,true}]). 5> rocksdb:put(Ref, <<"/staff/1">>,<<"vlad">>,[{sync,true}]). 6> rocksdb:put(Ref, <<"/staff/2">>,<<"maxim">>,[{sync,true}]). 7> rocksdb:put(Ref, <<"/staff/3">>,<<"doxtop">>,[{sync,true}]). 8> {ok,I} = rocksdb:iterator(Ref,[]). 9> rocksdb:iterator\_move(I,{seek,<<"/staff/">>}). 10> rocksdb:iterator\_move(I,next). 11> rocksdb:iterator\_move(I,next). 12> rocksdb:iterator\_move(I,next). 13> rocksdb:iterator\_move(I,{seek,<<"/users/">>}). 14> rocksdb:iterator\_move(I,next). 15> rocksdb:iterator\_move(I,next). 16> rocksdb:iterator\_move(I,next). 1> kvs:ver(). {version,"KVS ROCKSDB"} 2> rr(kvs). [emails,id\_seq,it,iter,kvs,reader,schema,table,writer] 3> kvs:join(). ok 4> kvs:put(#emails{id=1,email="maxim"}). 5> kvs:put(#emails{id=2,email="doxtop"}). 6> kvs:put(#writer{id=2}). 7> kvs:put(#writer{id=1}). 8> kvs:all(writer). [#writer{id = 1,count = 0,cache = [],args = [],first = []}, #writer{id = 2,count = 0,cache = [],args = [],first = []}] 9> kvs:all(emails). [#emails{id = 1,next = [],prev = [],email = "maxim"}, #emails{id = 2,next = [],prev = [],email = "doxtop"}] 10> kvs:add(#writer{id=chain,args=#emails{email="maxim@synrc.com"}}). 11> kvs:add(#writer{id=chain,args=#emails{email="vlad@synrc.com"}}). 12> kvs:add(#writer{id=chain,args=#emails{email="doxtop@synrc.com"}}). 13> kvs:all(chain). [#emails{id = 1555244691729330000,next = [],prev = [], email = "maxim@synrc.com"}, #emails{id = 1555244699905648000,next = [],prev = [], email = "doxtop@synrc.com"}, #emails{id = 1555244696660271000,next = [],prev = [], email = "vlad@synrc.com"}]

### MNESIA

# ЛОГІКА

### BPMN

BPE — це бібліотека управління бізнес-процесами стандарту BPMN 2.0 (з підтримкою паралельних гейтвеїв). До 2008 року існували інші стандарти та системи: WWF і BizTalk, Activity, Oracle BPM. BPE — це підмножина BPMN 2.0, який став узгодженим стандартом корпорацій в 2008 В BPE не обов'язково потрібно писати XML, всі ваші процеси можуть моделюватися в мовах Erlang та Elixir, а API настільки природний, наскільки природним є числення процесів та процеси віртуальної машини Ерланг. BPE повністю покриває всю мінімально необхідну спартанську функціональність систем управління бізнес-процесами, мереж Петрі, SADT-діаграм, діаграм активностей UML та FSM-моделей. У контексті процесів BPE у якості змінних можуть зберігатися і передаватися будь-які дані ERP системи які зберігаються в KVS. Кожен процес має свій лог, працює під супервізорами, з повністю персистентним станом, ізольований для vnode і стійкий до збоїв, здатний до міграцій. Є проксі N2O-протокол для веб-сокетів.

Поточний мій проект безпосередньо пов'язаний із моїм магістерським дипломом. Зараз я можу сміливо заявляти, що працюю за фахом. Тема мого магістрантського диплома на Факультеті Прикладної Математики була "Архітектура та програмування систем управління бізнес-процесами". У цій роботі було досліджено комплекс математичних, лінгвістичних (спеціалізованих мов DSL) та інших забезпечень, що лежать в основі таких систем.

У основі систем управління процесами лежать Мережі Петрі, переходи між станами FSM також відзначені вершинами. Цей математичний апарат передував моделі акторів. У роботі були розглянуті також інші моделі організації бізнес-процесів: Event-Condition-Action моделі (або як зараз можна говорити реакційної моделі), алгоритм RETE та дедуктивні системи правил.

Вже тоді мною були розглянуті всі існуючі на той момент Workflow DSL, як XPDL (WfMC), Skelta, OpenWFE, Shark, J2, jBPM. Створюючи тоді свою систему на .NET 1.0, я об'єднав ECA і Workflow підходи. Але я тоді не знав Erlang. Зараз я реалізував аналогічну за потужністю систему для мов Ерланг та Еліксір. За основу я взяв сучасний BPMN 2.0 CORE стандарт, який поєднав усі існуючі системи на ринку через 8 років після моєї роботи.

### ІНШІ ФОРМАЛЬНІ МОДЕЛІ

Приємним у всій палітрі різних реалізацій моделей робочого процесу є те, що всі вони зведені до одного з двох видів кодування: один є алгебраїчним, а інший — геометричним.

#### Мережі Петрі

Геометричною є мережі Петрі. Карл Петрі ввів його в 1962 році під час дискретного аналізу асинхронних комп'ютерних систем. Будь-яке його графічне представлення можна було б визначити за допомогою формалізму мереж Петрі. Моделювання Петрі в одній із його форм є хорошим доповненням до алгебри процесів, корисної як обчислювальна модель.

#### Пі-числення

Алгебраїчне обчислення - це число Пі, розроблене Робіном Мілнером, який отримав премію Тюрінга за 1) Метамову ML, 2) обчислення для комунікаційних систем CCS (1980), загальну теорію паралельності та 3) теоретичну базу для помічників доведення, Логіка обчислювальних функцій LCF. Модель обчислення процесів є теоретичною основою віртуального середовища інфраструктури Erlang, тому реалізація BPE повністю спирається на Pi-calculus (1999), наступник поняття CCS. Таким чином, забезпечується ефективна обчислювальна модель для реалізації управління процесом робочого процесу.

#### Скінченні автомати

Скінченні автомати Одним із загальновідомих типів обчислення процесу кодування є добре розроблена структура FSM (60-і роки). Ця мова широко використовується майже в будь-якій мові програмування, представленій як основна функція або як бібліотека. Процес визначається з розширенням до машини Тюрінга зі станами, входами, виходами та функціями.

#### SADT діаграми

SADT Наступною мовою (фреймворком), яка використовувалася в (80-х, 90-х) для опису подібних до обчислень процесів визначень за допомогою графічних мереж Петрі та визначень моделі, була SADT, представлена Marca і MacGowan 1988, 1991.

#### Реактивні системи

Реактивні системи Однією з широкого діапазону семантики є реактивні системи, засновані на передачі повідомлень і маршрутизації подій, але також вони можуть бути відомі як FRP функціонального реактивного програмування, що скоріше є набором комбінаторів над потоками. Обидві інтерпретації використовуються в мовах і фреймворках, залежно від участі потоку у визначенні ядра (2010-і).

### СИСТЕМИ ТИПІВ

Введення моделі Pi-числення у гомотопічній теорії типів дозволяю доводити чи спростовувати судження про властивості програм написаних на Пі-численні. Сесійні типи Кохея Хонди дозволяють вбудовувати логіку протоколів на рівні типів в система жирара (система F та система з лінійними типами). У BPE типи змінних відповідають типам документів, які є простими рекордами Erlang, тому в BPE обробка робочого процесу безпечна для типів на етапі компіляції щодо типів документів.

# МОДЕЛЬ

### ERLANG

#### N2O

-type n2o() :: #bert{} | #json{} | #binary{} | #default{}.

-type cx() :: #cx{}.

-type formatter() :: binary | json | bert | text | default | atom().

-type response() :: { formatter(), binary() }.

-record(ok, { data :: term() }).

-record(error { data :: term() }.

-record(reply, { msg :: n2o(), req :: term(), ctx :: cx() } ).

-record(unknown, { msg :: n2o(), req :: term(), ctx :: cx() } ).

-spec start(#pi{}) -> {pid(),term()} | #error{}.

-spec stop(term(),atom()) -> #pi{} | #error{}.

-spec proc(atom(),#pi{}) -> term().

-spec info(term(),term(),#cx{}) -> #reply{} | #unknown{}.

-record(pi, { name :: term(), table :: atom(), sup :: atom(), module :: atom(), state :: term() }).

-record(cx, { handlers = [] :: list({atom(),atom()}),

actions = [] :: list(tuple()),

req = [] :: [] | term(),

module = [] :: [] | atom(),

lang = [] :: [] | atom(),

path = [] :: [] | binary(),

session = [] :: [] | binary(),

formatter = bert :: bert | json | atom(),

params = [] :: [] | list(tuple()),

node = [] :: [] | atom(),

client\_pid= [] :: [] | term(),

state = [] :: [] | term(),

from = [] :: [] | binary(),

vsn = [] :: [] | binary() }).

-type memtable() :: atom().

-spec encode(tuple()) -> binary().

-spec decode(binary()) -> tuple().

-spec session(term(),term()) -> term().

-spec session(term()) -> term().

-spec cache(memtable(),term(),term()) -> term().

-spec cache(memtable(),term()) -> term().

#### KVS

-spec seq(atom() | [], integer() | []) -> term(). -spec count(atom()) -> integer(). -spec dir() -> list({'table',atom()}). -spec ver() -> {'version',string()}. -spec leave() -> ok. -spec destroy() -> ok. -spec join() -> ok | {error,any()}. -spec join(Node :: string()) -> [{atom(),any()}]. -spec modules() -> list(atom()). -spec cursors() -> list({atom(),list(atom())}). -spec tables() -> list(#table{}). -spec table(Tab :: atom()) -> #table{} | false. -spec top(#reader{}) -> #reader{}. -spec bot(#reader{}) -> #reader{}. -spec next(#reader{}) -> #reader{}. -spec prev(#reader{}) -> #reader{}. -spec drop(#reader{}) -> #reader{}. -spec take(#reader{}) -> #reader{}. -spec feed (term()) -> list(). -spec load\_reader (term()) -> #reader{}. -spec writer (term()) -> #writer{}. -spec reader (term()) -> #reader{}. -spec save (#reader{} | #writer{}) -> #reader{} | #writer{}. -spec add(#writer{}) -> #writer{}. -spec append(tuple(),term()) -> any(). -spec remove(tuple(),term()) -> integer(). -spec put(tuple() | list(tuple())) -> ok | {error,any()}. -spec get(term() | any(), any()) -> {ok,any()} | {error,not\_found}. -spec delete(term(), any()) -> ok | {error,not\_found}. -spec dump() -> ok. -spec start() -> ok. -spec stop() -> ok. -spec index(term(), any(), any()) -> list(tuple()). -record(writer, { id = [] :: term(), count = 0 :: integer(), cache = [] :: [] | {term(),term()} | {term(),term(),term()}, args = [] :: [] | term(), first = [] :: [] | tuple() } ). -record(reader, { id = [] :: [] | integer(), pos = 0 :: integer() | atom(), cache = [] :: [] | {term(),term()} | {term(),term(),term()}, args = [] :: [] | integer() | term(), feed = [] :: term(), seek = [] :: term(), count = 0 :: integer(), dir = 0 :: 0 | 1 } ). -record(id\_seq, { thing = []::term(), id = 0 :: integer() } ). -record(it, { id = []::[] | integer() } ). -record(ite, { id = []::[] | integer(), next = []::[] | integer() } ). -record(kvs, { mod = kvs\_mnesia :: kvs\_mnesia | kvs\_rocks | kvs\_fs, st = kvs\_stream :: kvs\_stream | kvs\_st, cx = []::term() }).

#### BPE

-type bst() :: binary() | string() | atom(). -type procId() :: [] | integer() | {atom(),any()}. -type histId() :: [] | integer() | {atom()|string(),any()}. -record(hist, { id = [] :: histId(), container = feed :: [] | atom(), feed\_id = [] :: any(), prev = [] :: [] | integer(), next = [] :: [] | integer(), feeds = [] :: list(), name = [] :: [] | binary(), task = [] :: [] | {atom()|string(),any()}, docs = [] :: list(tuple()), time = [] :: term() }). -type tasks() :: #task{} | #serviceTask{} | #userTask{} | #receiveTask{} | #beginEvent{} | #endEvent{}. -type events() :: #messageEvent{} | #boundaryEvent{} | #timeoutEvent{}. -record(process, { id = [] :: procId(), container = feed :: [] | atom(), feed\_id = [] :: [] | term(), prev = [] :: [] | integer(), next = [] :: [] | integer(), name = [] :: [] | bst(), feeds = [] :: list(), roles = [] :: list(), tasks = [] :: list(tasks()), events = [] :: list(events()), hist = [] :: [] | term(), flows = [] :: list(#sequenceFlow{}), rules = [] :: [] | term(), docs = [] :: list(tuple()), options = [] :: term(), task = 'Created' :: [] | atom(), timer = [] :: [] | reference(), notifications = [] :: [] | term(), result = [] :: [] | binary(), started = [] :: [] | calendat:datetime(), beginEvent = [] :: [] | atom(), endEvent = [] :: [] | atom()}).

#### NITRO

-spec render(list(#element{})) -> binary(). -spec wire(list(#action{})) -> []. -spec insert\_top(atom(), list(#element{})) -> []. -spec insert\_bottom(atom(), list(#element{})) -> []. -spec update(atom(), list(#element{})) -> []. -spec clear(atom()) -> []. -spec remove(atom()) -> []. -record(init, { token = [] :: [] | binary() }). -record(pickle, { source = [], pickled=[], args=[] }). -record(flush, { data = [] :: [] | term() }). -record(direct, { data = [] :: [] | term() }). -record(ev, { module = [] :: [] | atom(), msg = [] :: [] | term(), trigger= [] :: [] | atom() | binary(), name = [] :: [] | atom() | binary() }).

### ANDERS

#### module N2O-KVS-BPE where import BASE axiom pickle : Binary -> Binary axiom depickle : Binary -> Binary axiom encode : Π (k: U), k -> Binary axiom decode : Π (k: U), Binary -> IO k axiom reg: Π (k: U), k -> IO k axiom unreg : Π (k: U), k -> IO k axiom send : Π (k v z: U), k -> v -> IO z axiom getSession : Π (k v: U), k -> IO v axiom putSession : Π (k v: U), k -> v -> IO v axiom getCache : Π (k v: U), Atom -> k -> IO v axiom putCache : Π (k v: U), Atom -> k -> v -> IO v axiom start : PI -> IO Sup axiom get : Π (f k v: U), f -> k -> IO (Maybe v) axiom put : Π (r: U), r -> IO StoreResult axiom delete : Π (f k: U), f -> k -> StoreResult axiom index : Π (f p v r: U), f -> Atom -> v -> List r axiom next : Reader -> IO Reader axiom prev : Reader -> IO Reader axiom take : Reader -> IO Reader axiom drop : Reader -> IO Reader axiom save : Reader -> IO Reader axiom append : Π (f r: U), f -> r -> IO StoreResult axiom remove : Π (f r: U), f -> r -> IO StoreResult axiom start : Proc -> IO Sup axiom stop : String -> IO Sup axiom next : ProcId -> IO ProcRes axiom amend : Π (k: U), ProcId -> k -> IO ProcRes axiom discard : Π (k: U), ProcId -> k -> IO ProcRes axiom modify : Π (k: U), ProcId -> k -> Atom -> IO ProcRes axiom event : ProcId -> String -> IO ProcRes axiom head : ProcId -> IO Hist axiom hist : ProcId -> IO (List Hist)