Politechnika Świętokrzyska
Wydział Zarządzania i Modelowania Komputerowego

Kierunek studiów:
Inżynieria danych

Sławomir Luściński

Materiały dydaktyczne do przedmiotu

KOMPUTEROWE WSPOMAGANIE ZARZĄDZANIA PRODUKCJĄ

opracowane w ramach realizacji Projektu
„Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
FERS.01.05-IP.08-0234/23

Kielce, 2025

Spis treści

Wprowadzenie

Celem tego opracowania jest przedstawienie kluczowych zagadnień związanych z Internetem rzeczy, jako rozszerzenia treści wykładu „Komputerowe wspomaganie zarządzania produkcją", realizowanego w ramach kierunku Inżynieria danych na Politechnice Świętokrzyskiej. Materiał został przygotowany z myślą o studentach, którzy chcą pogłębić swoją wiedzę na temat nowoczesnych narzędzi cyfrowych wspierających zarządzanie i integrację systemów, ze szczególnym uwzględnieniem technologii Internetu rzeczy.

Publikacja obejmuje tematy transformacji cyfrowej w przemyśle, w tym systemy cyberfizyczne (CPS), Internet rzeczy (IoT) oraz ich zastosowania w nowoczesnych modelach produkcji. Szczególną uwagę zwrócono na środowisko Node-RED, opisane jako elastyczna platforma do tworzenia aplikacji przemysłowych i rozwiązań Iot. Zaprezentowane zagadnienia i przykłady mają na celu rozwijanie kompetencji studentów w zakresie integracji systemów, automatyzacji procesów oraz wykorzystania narzędzi cyfrowych w środowisku przemysłowym.

W drugiej części przedstawiono praktyczny przykład aplikacji w Node-RED, obrazujący pełny cykl przetwarzania danych – od pozyskiwania informacji z czujników po ich wizualizację i analizę.

Ostatni rozdział obejmuje omówienie węzłów funkcjonalnych oraz języków przetwarzania danych wykorzystywanych w Node-RED: JavaScript i JSONata, wraz z ich porównaniem pod kątem zastosowań w automatyzacji i strumieniowym przetwarzaniu danych.

Materiał łączy treści teoretyczne z praktycznymi przykładami, umożliwiając ich bezpośrednie zastosowanie podczas zajęć laboratoryjnych i projektów studenckich.

1. Transformacja cyfrowa w przemyśle

1.1. Przemysł 4.0

Rewolucja IT (ang. Information Technology) przyniosła ze sobą upowszechnienie się datafikacji (ang. *datafication*) procesów we wszystkich obszarach funkcjonowania systemów społeczno-technicznych. Datafikacja to nie tylko proces digitalizacji; to również proces tworzenia ilościowej reprezentacji danego zjawiska w formacie dogodnym dla celów analizy. Datafikacja umożliwia pojawienie się w świecie organizacji procesu transformacji cyfrowej (ang. *Digital Transformation* - DT) definiowanego jako wykorzystanie technologii IT do radykalnej poprawy wydajności lub zasięgu oddziaływania organizacji Proces DT prowadzi do rewitalizacji istniejących zasobów strategicznych organizacji, wprowadzając je do nowych modeli biznesowych (tzw. firmy modyfikowane cyfrowo) oraz tworzy nowe zasoby określane jako możliwości cyfrowe (ang. *Digital Capabilities*).

Innowacje w technologiach informatycznych i automatyce przemysłowej ukształtowały nowy paradygmat systemów produkcyjnych znany jako Przemysł 4.0 (*Industrie 4.0*). Koncepcja Przemysł 4.0 (I4.0) obejmuje cztery kluczowe elementy:

1. Inteligentne fabryki (ang. Smart Factories).
2. Systemy cyberfizyczne (Cyber-Physical Systems, CPS).
3. Internet rzeczy (ang. Internet of Things IoT).
4. Internet usług (Internet of Services, IoS).

Model czterech rewolucji przemysłowych określa zmiany technologiczne i społeczno-techniczne w produkcji jako cztery kolejne epoki scharakteryzowane w ()

Tabela 1. Model czterech rewolucji przemysłowych

Rewolucja przemysłowa	**Charakterystyka**
Przemysł 1.0	Mechanizacja (koniec XVIII-XIX w.): wprowadzenie produkcji mechanicznej napędzanej wodą/parą, zmechanizowanych maszyn i systemu fabrycznego.
Przemysł 2.0	Masowa elektryfikacja i masowa produkcja (koniec XIX - początek XX wieku): powszechna elektryfikacja, organizacja linii montażowych i produkcja na skalę ekonomiczną umożliwiły znormalizowaną produkcję masową (kontekst i przejście do systemów przemysłowych na większą skalę).
Przemysł 3.0	Automatyzacja, elektronika i informatyka (koniec XX wieku). Wdrożenie elektroniki, programowalnych sterowników logicznych, komputerów i wczesnej automatyzacji umożliwiło elastyczną kontrolę procesów produkcyjnych.

Przemysł 4.0	Integracja cyber-fizyczna, cyfryzacja i inteligentna produkcja (XXI wiek). Ścisła integracja świata cybernetycznego i fizycznego poprzez CPS/IIoT, Digital Twins, big data/AI, cloud/edge computing i platformy interoperacyjne w celu umożliwienia wykrywania, modelowania, optymalizacji i autonomicznego lub półautonomicznego sterowania w czasie rzeczywistym w inteligentnych fabrykach.

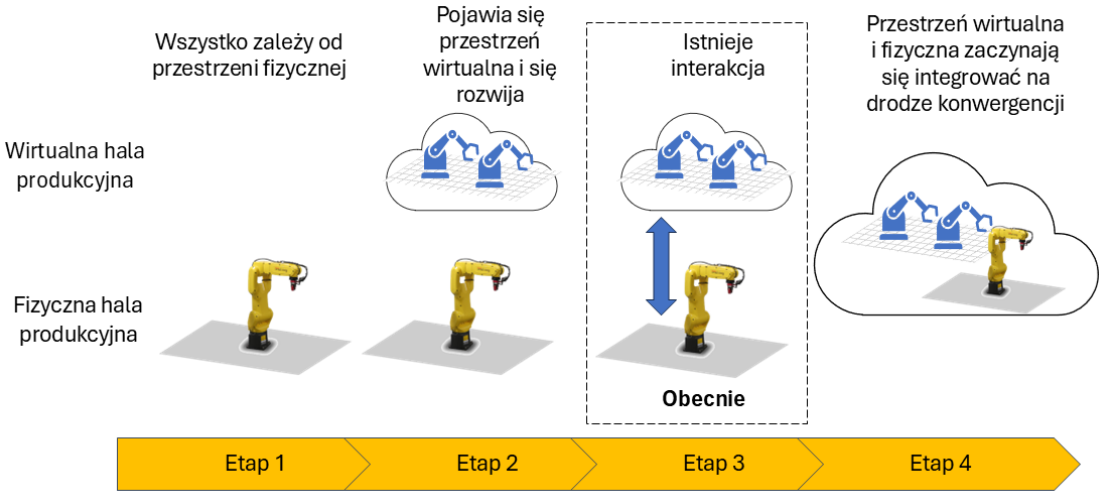
Cyfrowa transformacja procesów produkcyjnych i zarządzania jest napędzana rozwojem elastycznych systemów, które wykorzystują zaawansowane technologie informacyjno-komunikacyjne (ang. *Information and Communication Technology*, ICT) w automatyzacji zarówno procesów produkcyjnych, jak i biznesowych. Cyfryzacja umożliwia efektywne wdrożenie organizacji skoncentrowanej na kliencie, opierającej się na integracji przepływu danych i informacji zarówno wewnątrz, jak i na zewnątrz przedsiębiorstwa. Struktura sieciowa wspiera tworzenie łańcucha wartości ukierunkowanego na potrzeby klienta. W efekcie powstaje tzw. organizacja klientocentryczna.

1.2. Systemy cyberfizyczne

Systemy cyberfizyczne to połączenie naturalnych i stworzonych przez człowieka elementów (przestrzeń fizyczna) z systemami obliczeniowymi, komunikacyjnymi i kontrolnymi (cyberprzestrzeń). Są one kontrolowane lub monitorowane przez algorytmy komputerowe i ściśle zintegrowane z Internetem oraz jego użytkownikami. Elementy fizyczne i programowe w systemach cyberfizycznych są ze sobą powiązane, działając na różnych skalach przestrzennych i czasowych, wykazując różne zachowania i wchodząc w liczne interakcje, które zależą od kontekstu. Obecnie powszechnie przyjmuje się, że wdrożenie systemów CPS w przemyśle napędza koncepcję Przemysłu 4.0. Dzięki integracji cyberprzestrzeni i świata fizycznego, CPS mogą zapewnić maszynom przemysłowym autonomiczną kontrolę, samoświadomość i zdolności samozarządzania. . Początkowo procesy na hali produkcyjnej funkcjonowały bez jakiegokolwiek wsparcia cyfrowego. W drugim etapie nastąpił rozwój systemów planowania i sterowania produkcją. Przełom i przejście do etapu trzeciego rozpoczął się w pierwszej dekadzie XXI wieku wraz z wprowadzeniem zintegrowanych systemów wytwarzania (CIM), co było możliwe dzięki cyfryzacji obrabiarek (CNC) oraz ewolucji narzędzi projektowych (CAD/CAE/CAM). Obecnie branża znajduje się w trzeciej fazie, w której technologie takie jak Internet Rzeczy (IoT) i analiza Big Data zacieśniają interakcję między sferą fizyczną a wirtualną. Kolejny, czwarty etap rewolucji jest napędzany przez rozwój cyfrowych bliźniaków (Digital Twins, DT). Wsparte rozwojem IT technologie cyber-fizyczne zapewniają intensywną, dwukierunkową komunikację między obiektami rzeczywistymi a jej cyfrową repliką, przyspieszając ich konwergencję.

Na Rys. 1. zobrazowano ewolucję systemów informatycznych w przemyśle produkcyjnym w opisanych powyżej czterech etapach

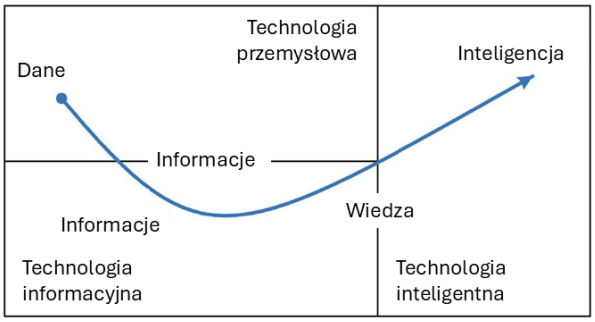
[Tekst alternatywny: Schemat czteroetapowego procesu pokazuje ewolucję od produkcji wyłącznie fizycznej przez pojawienie się przestrzeni wirtualnej i interakcję aż do ich konwergentnej integracji, z wyróżnieniem, że „obecnie” znajdujemy się na etapie 3.]



Rys. 1. Proces ewolucji zastosowania systemów informatycznych w produkcji.
Źródło: opracowanie własne

Rola technologii informacyjnej (IT) we wdrażaniu systemów cyber-fizycznych polega na integracji technologii przemysłowych, informatycznych i tzw. inteligentnych (Rys. 2).

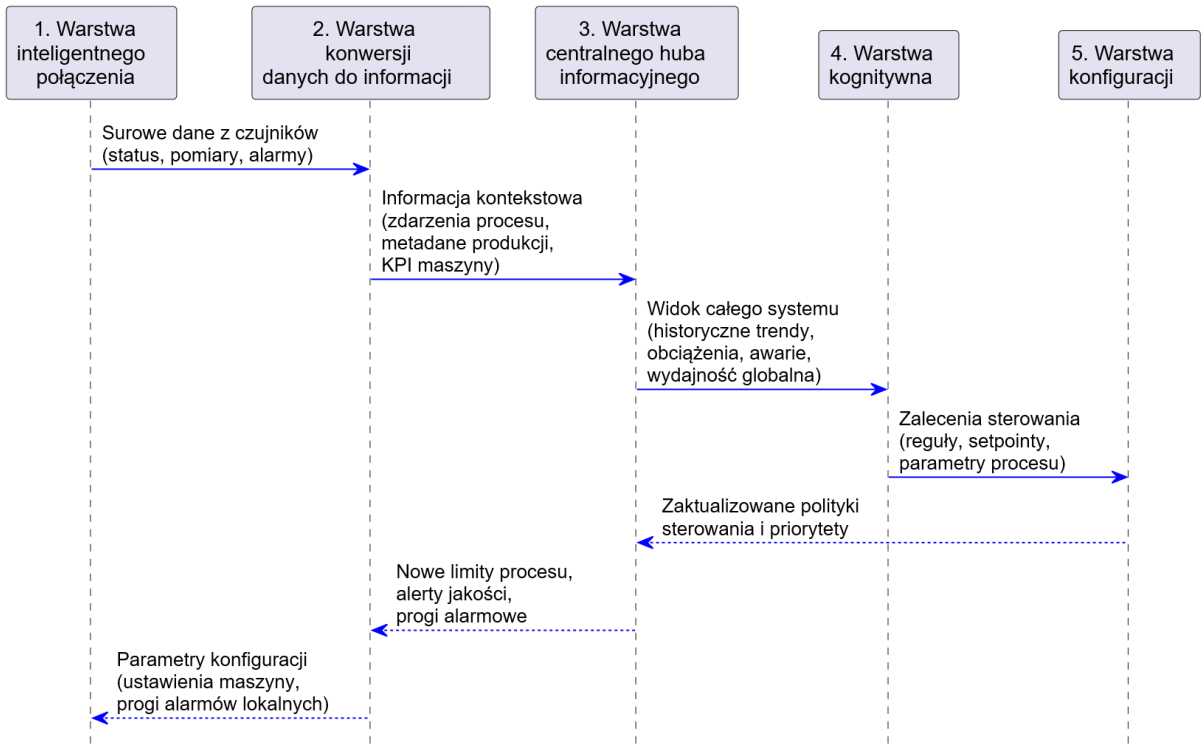
[Tekst alternatywny: Wykres koncepcyjny na siatce 2x2, ilustrujący ewolucję od danych do inteligencji w kontekście rozwoju technologii IT. Przez diagram przebiega zakrzywiona, niebieska linia w kształcie litery „U”, która obrazuje czterofazowy proces rozwoju: od fazy „Dane” przez „Informacje”, „Wiedza” do etapu końcowego „Inteligencja”]



Rys. 2. Integracja z nowymi technologiami informatycznymi
Źródło: opracowanie własne na podstawie (Tao et al., 2019)

Model 5C architektury systemów CPS składa się z 5-ciu warstw (Rys. 3).

[Tekst alternatywny: Diagram sekwencji przedstawia pięciowarstwową architekturę 5C systemu cyberfizycznego. Dane przepływają od warstwy inteligentnego połączenia, przez konwersję danych do informacji, centralny hub informacyjny i warstwę kognitywną, aż do warstwy konfiguracji. Następnie następuje sprzężenie zwrotne w dół — od decyzji konfiguracyjnych do poziomu maszyn.]



Rys. 3. Model 5C architektury systemów CPS.
Źródło: opracowanie własne na podstawie (Lee i in. 2015).

1. **Warstwa inteligentnego połączenia** (*Smart Connection Level*) – ta warstwa odpowiada za pozyskiwanie dokładnych i wiarygodnych danych o maszynach, urządzeniach i procesach. Dane mogą pochodzić bezpośrednio z czujników lub być uzyskane z programowalnych sterowników logicznych PLC (*Programmable Logic Controller*), stosowanych w automatyce, systemów realizacji produkcji MES (*Manufacturing Execution System*).
2. **Warstwa konwersji danych na informacje** (*Data-to-Information Conversion Level*) nadaje znaczenie danym, szczególnie rozwijając algorytmy do prognozowania stanu maszyn i urządzeń (utrzymanie ruchu).

3. **Warstwa centralnego huba informacyjnego** (*Cyber Level)* obejmuje sieciowe czujniki w maszynach, które dostarczają informacje pozwalające na porównanie ich stanu i predykcję na podstawie danych innych maszyn lub danych historycznych.
4. **Warstwa kognitywna** (*Cognition Level*) polega na wydobywaniu wiedzy, prezentując porównania i status maszyn, często w graficznej formie, co pomaga użytkownikom systemu zrozumieć obecną sytuację i podejmować decyzje.
5. **Warstwa konfiguracji** (*Configuration level*) zapewnia sprzężenie zwrotne między przestrzenią wirtualną a fizyczną, pełniąc funkcję nadzorczą i umożliwiając wprowadzanie korekt lub decyzji prewencyjnych.

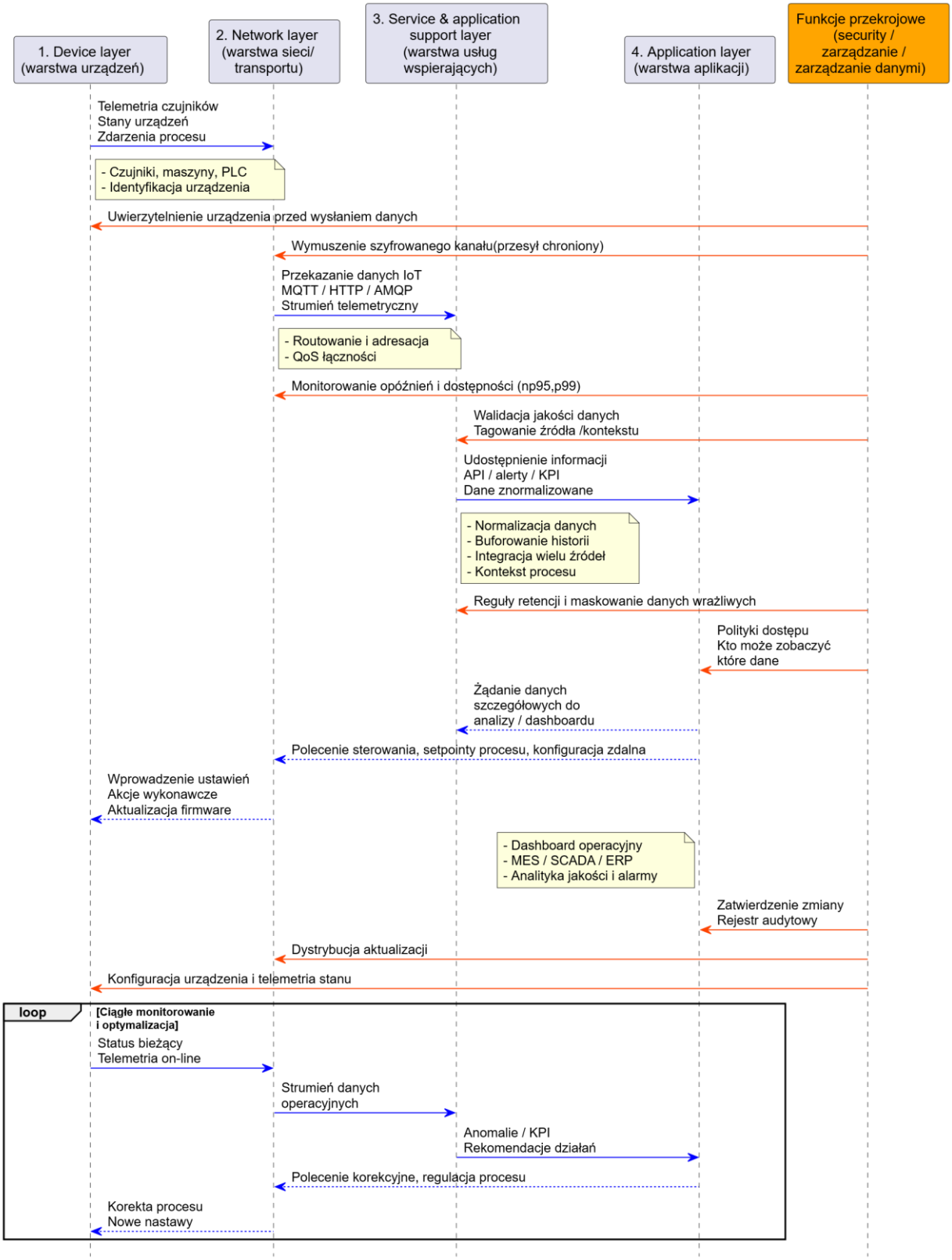
1.3. Internet rzeczy

Internet rzeczy IoT (*Internet of Things*) można zdefiniować jako scharakteryzować jako rozległą, wzajemnie połączoną sieć inteligentnych urządzeń, zdolnych do autonomicznego organizowania, udostępniania i zarządzania informacjami oraz zasobami danych.

Przemysłowy Internet rzeczy IIoT(*Industrial Internet of Things*) jest rozwinięciem koncepcji IoT w kierunku zastosowań przemysłowych, w których nadrzędnym celem jest bezpośrednia integracja urządzeń pomiarowych, systemów sterowania oraz aplikacji przetwarzających dane w czasie rzeczywistym z infrastrukturą informatyczną przedsiębiorstwa. Jest to sieć połączonych ze sobą urządzeń fizycznych, czujników, elementów wykonawczych (aktuatorów) i systemów wbudowanych, które gromadzą, wymieniają i analizują dane za pośrednictwem komunikacji przewodowej lub bezprzewodowej. Tego rodzaju łączność umożliwia monitorowanie procesów, ich automatyzację oraz świadczenie usług o wartości dodanej.

Model warstwowy IoT, zdefiniowany w rekomendacji ITU-T Y.2060 (06/2012) (*International Telecomunication Union*), opisuje strukturę systemu IIoT w postaci czterech logicznych warstw: urządzeń, sieci, usług wspierających oraz aplikacji. Każda z nich realizuje odrębne funkcje - od pozyskiwania danych z otoczenia po ich przetwarzanie i wykorzystanie w aplikacjach użytkowych. Model ten uzupełniają funkcje przekrojowe (*cross-cutting functions*), obejmujące wszystkie warstwy i zapewniające spójne zarządzanie bezpieczeństwem, infrastrukturą oraz cyklem życia danych (Rys. 4).

[Tekst alternatywny: Diagram sekwencji przedstawia przepływ danych w modelu IIoT według ITU-T Y.2060 — od warstwy urządzeń, przez sieć i usługi wspierające, do warstwy aplikacji. Pokazuje również funkcje przekrojowe, takie jak bezpieczeństwo, zarządzanie i zarządzanie danymi, które wspierają każdą warstwę systemu.]



Rys. 4. Przepływ danych i usług w modelu IIoT wg ITU-T Y.2060. Źródło: opracowanie własne.

W Tabela 2 zestawiono główne role poszczególnych warstw modelu referencyjnego oraz ich powiązania z funkcjami przekrojowymi.

Tabela 2. Model referencyjny IoT wg ITU-T Y.2060

Nr	Warstwa (ang./pol.)	Główna rola i funkcje	Funkcje przekrojowe (cross-cutting)
1	**Device layer** *(warstwa urządzeń)*	• Czujniki, aktuatory, sterowniki PLC. • Zbieranie danych z otoczenia, identyfikacja urządzeń, generowanie sygnałów i zdarzeń procesu.	• **Bezpieczeństwo:** uwierzytelnianie urządzeń, zabezpieczenie komunikacji. • **Zarządzanie:** konfiguracja, aktualizacje firmware, telemetria stanu • **Zarządzanie danymi:** wstępna filtracja, integralność danych źródłowych.
2	**Network layer** *(warstwa sieci / transportu)*	• Transmisja danych między urządzeniami a systemami centralnymi. • Routowanie, adresacja, QoS, łączność (Ethernet, Wi-Fi, 5G, LPWAN).	• **Bezpieczeństwo:** szyfrowanie transmisji, kontrola dostępu. • **Zarządzanie:** monitorowanie dostępności, opóźnień, niezawodności (p95/p99). • **Zarządzanie danymi:** walidacja poprawności pakietów, eliminacja duplikatów.
3	**Service & Application Support layer** *(warstwa usług wspierających)*	• Middleware, brokery komunikacyjne, buforowanie, przetwarzanie strumieniowe. • Integracja wielu źródeł danych, udostępnianie API.	• **Bezpieczeństwo:** kontrola dostępu do usług i brokerów. • **Zarządzanie:** orkiestracja usług, rejestracja komponentów, skalowanie. • **Zarządzanie danymi:** reguły retencji, jakość danych, standaryzacja formatów.
4	**Application layer** *(warstwa aplikacji)*	• Aplikacje biznesowe i domenowe (MES, SCADA, ERP, dashboardy, analityka). Prezentacja danych i wspomaganie decyzji.	• **Bezpieczeństwo:** autoryzacja użytkowników, ochrona danych wrażliwych. • **Zarządzanie:** raportowanie, SLA, zarządzanie konfiguracją i zmianą. • **Zarządzanie danymi:** prywatność, zgodność z regulacjami, własność danych.

Do kluczowych cech IoT należą:

- Rozbudowane możliwości w zakresie sensoryki i łączności — takie jak RFID, bezprzewodowe sieci czujników (WSN) czy inteligentne sensory.
- Gromadzenie danych na tzw. brzegu sieci (*edge data collection*).

- Zastosowanie standaryzowanych protokołów i platform komunikacyjnych (np. MQTT, CoAP, HTTP, OPC UA).
- Warstwowe przetwarzanie danych — od obliczeń na brzegu sieci (*edge*) i we mgle (*fog computing*) aż po chmurę (*cloud*).
- integracja z systemami analitycznymi i sztuczną inteligencją w celu podejmowania decyzji w czasie rzeczywistym, sterowania oraz automatyzacji.

Przykładowe technologie wykorzystywane w praktycznych wdrożeniach IIoT przedstawiono w Tabela 3.

Tabela 3. Technologie stosowane w IIoT

Nr	Warstwa (ang./pol.)	Przykłady technologii / narzędzi
1	**Device layer** *(warstwa urządzeń)*	Arduino, ESP32, Raspberry Pi, Siemens S7, Beckhoff, Modbus, Edge Gateway
2	**Network layer** *(warstwa sieci / transportu)*	MQTT, AMQP, HTTP, CoAP, 5G, Wi-Fi, Ethernet, LoRaWAN, VPN, Firewall, TLS/SSL
3	**Service & Application Support layer** *(warstwa usług wspierających)*	Node-RED, Eclipse Mosquitto, InfluxDB, Kafka, Redis, OPC UA Pub/Sub, Docker, Kubernetes
4	**Application layer** *(warstwa aplikacji)*	Grafana, Power BI, Node-RED Dashboard, Ignition SCADA, Azure IoT Hub, AWS IoT, Siemens MindSphere, ThingWorx

2. Node-RED jako platforma integracyjna IoT

2.1. Środowisko uruchomieniowe Node.js dla aplikacji sieciowych

Rozwój technologii informatycznych doprowadził do powstania chmury obliczeniowej (*Cloud Computing*), która zrewolucjonizowała sposób budowy systemów informatycznych. W tym modelu zarówno aplikacje, jak i dane są udostępniane użytkownikom za pośrednictwem Internetu.

Node.js stanowi wieloplatformowe środowisko uruchomieniowe, umożliwiające wykonywanie kodu JavaScript po stronie serwera. Opiera się na wydajnym silniku V8, znanym z przeglądarki Google Chrome, i jest dostępne na licencji open-source dla systemów Windows, macOS oraz Linux. Kluczową cechą Node.js jest jego architektura oparta na zdarzeniach i asynchronicznych operacjach wejścia/wyjścia (I/O). Dzięki temu jest on idealny do tworzenia skalowalnych i wydajnych aplikacji sieciowych, takich jak serwery WWW, które muszą obsługiwać wiele jednoczesnych połączeń. Potwierdzeniem znaczenia i popularności Node.js jest jego dostępność w największych chmurach obliczeniowych, takich jak Microsoft Azure i Google Cloud Platform.

Platforma Node.js jest rozwijana i udostępniana na licencji Open-Source; strona projektu: https://nodejs.org. Po pobraniu pliku instalacyjnego i zainstalowaniu Node zaleca się utworzenie katalogu, w którym będą umieszczane aplikacje użytkownika; przykładowo może to być katalog \Project w katalogu użytkownika.

JavaScript jest językiem skryptowym opartym na paradygmacie obiektowym, charakteryzującym się dynamiczną obsługą typów, służącym do tworzenia złożonych programów osadzonych w kodzie HTML. Język ten jest w pełni zintegrowany z technologiami HTML i CSS. Operacje wejścia i wyjścia realizowane są za pomocą obiektowego modelu dokumentu W3C DOM (ang. *Document Object Model*). Dane zapisane na stronie internetowej są traktowane jako obiekty o strukturze hierarchicznej.

2.2. Środowisko Node-RED do budowy aplikacji sieciowych

Projekt Node-RED powstał pod koniec 2013 roku w firmie IBM jako narzędzie open-source do prototypowania rozwiązań IoT. Szybko zyskał popularność i ewoluował w uniwersalną platformę do integracji urządzeń, interfejsów API i usług online. Obecnie projekt jest rozwijany w ramach fundacji JS Foundation i dostępny na licencji Apache 2.0.

Node-RED jest narzędziem wizualnym służącym do tworzenia programów, opartym na środowisku Node.js, umożliwiającym szybkie opracowywanie aplikacji, szczególnie dla Internetu Rzeczy (IoT). Zamiast tradycyjnego kodowania, programy są tworzone w przeglądarce poprzez łączenie gotowych bloków funkcyjnych (węzłów) w logiczne przepływy danych. Proces ten można porównać do programowania komponentowego: łączenia predefiniowanych elementów w całość. Wykorzystanie i łączenie węzłów danych wejściowych, przetwarzania oraz wyjściowych pozwala na implementację algorytmu jako przepływu (ang. *flow*).

W Tabela 4 przedstawiono podstawowe elementy Node-RED.

Tabela 4. Podstawowe elementy środowiska Node-RED.

Element	Opis	Rola
Node (Węzeł)	Podstawowy element przepływu (*flow*). Reaguje na wiadomości lub zdarzenia zewnętrzne (np. HTTP, timer, GPIO). Może mieć jedno wejście i wiele wyjść.	Realizuje pojedynczą funkcję, np. przetwarzanie danych, wyzwalanie akcji lub komunikację z urządzeniem.
Configuration Node (Węzeł konfiguracyjny)	Specjalny typ węzła przechowujący wspólną konfigurację, współdzieloną przez inne węzły.	Umożliwia ponowne użycie parametrów połączenia (np. MQTT Broker, serwer HTTP).
Flow (Przepływ)	Zestaw połączonych węzłów przedstawiony jako zakładka w edytorze. Może zawierać wiele logicznych przepływów.	Organizuje logikę aplikacji i grupuje węzły realizujące wspólne zadanie.
Context (Kontekst)	Mechanizm przechowywania danych dostępnych dla węzłów bez przekazywania ich przez msg. Istnieją trzy poziomy: Node, Flow, Global.	Umożliwia zapamiętywanie stanów i współdzielenie danych pomiędzy węzłami.
Message (Wiadomość)	Obiekt JavaScript przekazywany pomiędzy węzłami. Zazwyczaj zawiera pole msg.payload z właściwymi danymi.	Nośnik danych w przepływie – przekazuje wyniki między węzłami.
Subflow (Podprzepływ)	Zbiór węzłów połączonych w jeden komponent, który można ponownie wykorzystać w różnych miejscach.	Upraszcza strukturę przepływu i umożliwia tworzenie modułowych komponentów.
Wire (Połączenie)	Łącze między węzłami reprezentujące przepływ wiadomości (msg).	Określa kierunek i logikę przepływu danych w aplikacji.

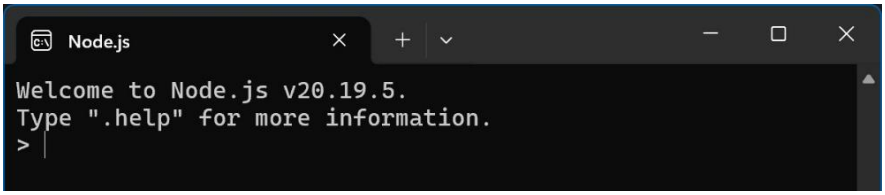
Instalacja Node-RED, dostępnego globalnie w środowisku Node.js, wymaga uruchomienia konsoli operatora (cmd lub PowerShell) w trybie administratora i wykonania instrukcji:

```
npm install -g --unsafe-perm node-red
```

Instalacja interfejsu Web dla aplikacji Node-RED (*Node-RED-Dashboard*) wymaga wykonania instrukcji:

```
npm i node-red-dashboard
```

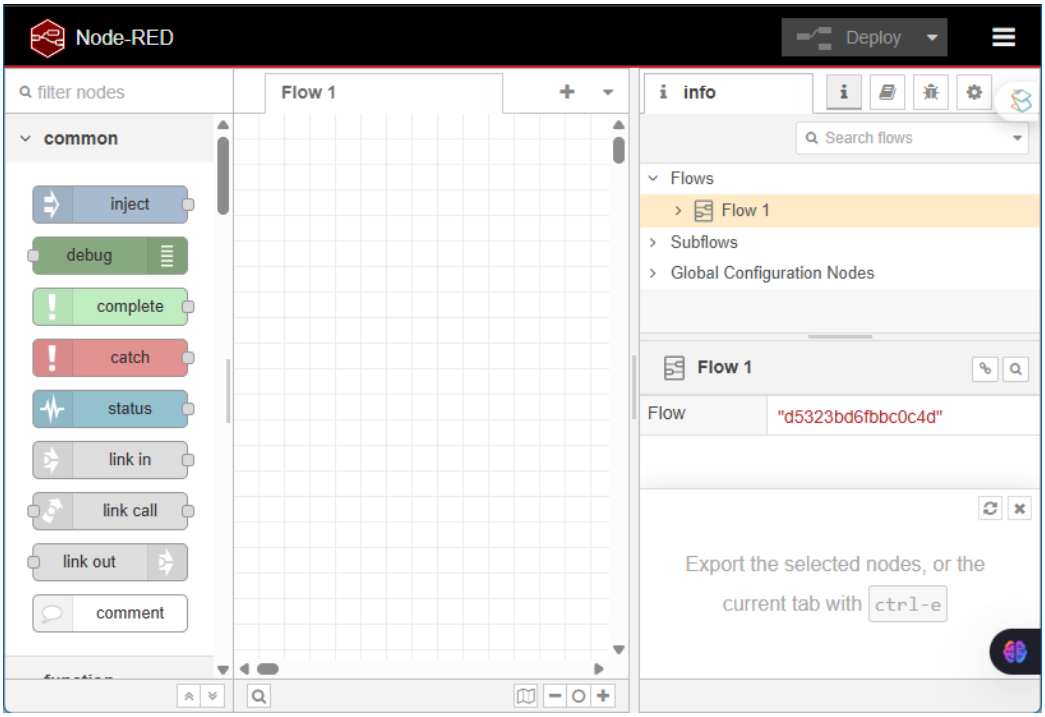
Po uruchomieniu środowiska Node.js na ekranie otwiera się okno terminala Node.js (Rys. 5).
[Tekst alternatywny: Okno terminala Node.js w systemie Windows pokazujące uruchomione środowisko Node.js. Na ekranie widoczny jest komunikat powitalny oraz znak zachęty >]



Rys. 5. Okno terminala Node.js w systemie Windows.
Źródło: zrzut ekranu.

Uruchomienie Node-RED odbywa się poprzez wywołanie w konsoli operatora polecenia: node-red. Edytor graficzny (Rys. 6) jest dostępny jako aplikacja przeglądarkowa pod adresem: `http://127.0.0.1:1880.`

[Tekst alternatywny: zrzut ekranu okna edytora Node-RED]



.

Rys. 6. Edytor Node-RED.
Źródło: zrzut ekranu.

Elementy interfejsu:

- **Paleta węzłów** (po lewej stronie) stanowi pionową listę dostępnych bloków funkcjonalnych (węzłów), pogrupowanych w kategorie (na zrzucie widoczna jest kategoria „common"). Użytkownik przeciąga węzły z tej palety na centralny obszar roboczy. Widoczne są tu podstawowe węzły, takie jak „inject" (do ręcznego inicjowania przepływu), „debug" (do wyświetlania komunikatów) oraz „comment" (do dodawania komentarzy).
- **Obszar roboczy** (w centrum) stanowi główną przestrzeń, w której buduje się program. Użytkownik umieszcza tu węzły z palety i łączy je ze sobą w celu tworzenia logicznych przepływów (*flows*).
- **Panel boczny** (po prawej stronie) zapewnia dostęp do dodatkowych informacji i narzędzi. Na zrzucie ekranu aktywna jest zakładka „info", prezentująca strukturę projektu (listę przepływów, podprzepływów) oraz dane dotyczące zaznaczonych elementów. Dostępne są tu również inne zakładki, np. do debugowania (wyświetlania komunikatów z węzła „debug") oraz zarządzania konfiguracją.
- Na górze po prawej stronie znajduje się kluczowy przycisk **„Deploy"**, który służy do wdrażania (uruchamiania) zmian wprowadzonych w przepływach.

Główne zalety Node-RED :

- **Szybkość** i prostota pozwalają na szybkie tworzenie nawet najbardziej skomplikowanych aplikacji bez konieczności pisania rozbudowanego kodu. •
- **Programowanie wizualne** – logika programu jest intuicyjna od pierwszego kontaktu, co ułatwia debugowanie i modyfikacje.
- **Biblioteka węzłów** – społeczność użytkowników stworzyła tysiące gotowych węzłów, które integrują się z niemal każdą popularną usługą (np. Twitter, bazy danych, systemy pogodowe) oraz urządzeniami (np. Raspberry Pi, Arduino, inteligentne oświetlenie, elementy automatyki przemysłowej, platformy AI). AI).

2.3. Zarządzanie przepływami

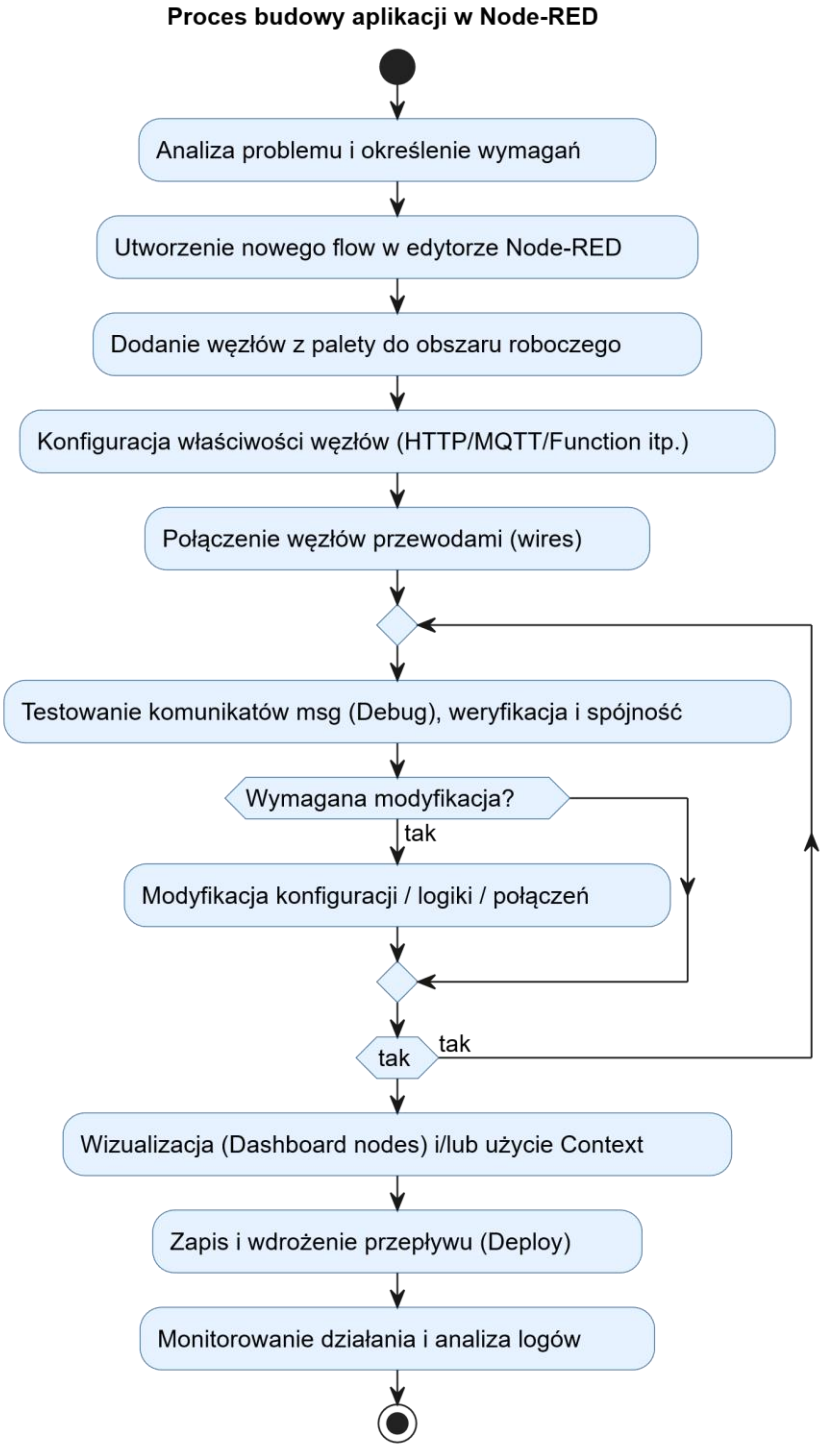
Tabela 5. przedstawia listę funkcji edytora Node-RED związanych z zarządzaniem przepływami. Zawiera najważniejsze operacje, takie jak dodawanie, edycja, ukrywanie, blokowanie i usuwanie przepływów, wraz z odpowiadającymi opcjami menu i akcjami systemowymi.

Tabela 5. Zarządzanie przepływami w edytorze Node-RED

Funkcja / Czynność	**Opis**	**Opcja menu**
Definicja przepływu (*Flow*)	Przepływ jest reprezentowany jako zakładka w obszarze roboczym edytora i stanowi główny sposób organizowania węzłów. Termin *flow* może również odnosić się do jednego	–

	zestawu połączonych węzłów. Jedna zakładka może więc zawierać wiele logicznych przepływów.	
Nazwa i opis przepływu	Każdy przepływ może mieć nazwę i opis, wyświetlane w panelu informacyjnym (*Information Sidebar*). Opis można formatować w składni *Markdown*.	Flows → Rename
Kontekst przepływu (Flow *Context*)	Wszystkie węzły w danym przepływie mają dostęp do wspólnego kontekstu o zasięgu *flow*, który umożliwia współdzielenie danych bez użycia wiadomości *msg*.	–
Dodawanie przepływu	Aby dodać nowy przepływ, kliknij przycisk „+” na pasku zakładek lub kliknij dwukrotnie pustą przestrzeń w pasku zakładek.	Flows → Add
Zmienianie kolejności przepływów	Przepływy można przeciągać w pasku zakładek, aby zmienić ich kolejność.	–
Edycja właściwości przepływu	Dwukrotne kliknięcie zakładki przepływu otwiera okno *Flow Properties*, w którym można zmienić nazwę, opis oraz ustawić zmienne środowiskowe.	Flows → Rename
Zmienna środowiskowa (*Environment Variable*)	Właściwości przepływu mogą być udostępniane jako zmienne środowiskowe, dostępne dla węzłów w ramach danego *flow*.	–
Włączanie / wyłączanie przepływu	W oknie *Flow Properties* znajduje się przełącznik do aktywacji lub dezaktywacji przepływu. Wyłączony przepływ nie jest uruchamiany podczas wdrażania (*deploy*).	–
Ukrywanie przepływu	Można ukryć przepływ, klikając go prawym przyciskiem myszy i wybierając *Hide flow* lub korzystając z menu zakładek. Ukryte przepływy są oznaczone ikoną oka w panelu informacyjnym.	[Tab Menu] → Hide flow
Pokazywanie ukrytego przepływu	Przywrócenie ostatniego ukrytego przepływu.	[Tab Menu] → Show last hidden flow
Ukryj inne / wszystkie przepływy	Ukrywa pozostałe lub wszystkie zakładki przepływów w obszarze roboczym.	[Tab Menu] → Hide other / Hide all flows
Pokaż wszystkie przepływy	Przywraca widoczność wszystkich przepływów.	[Tab Menu] → Show all flows
Blokowanie przepływu	Chroni przepływ przed edycją. Można go zablokować: • klikając prawym przyciskiem zakładkę i wybierając *Lock flow*, • w oknie *Flow Properties*, • klikając ikonę kłódki w panelu informacyjnym.	[Tab Menu] → Lock / Unlock flow
Usuwanie przepływu	Przepływ można usunąć przyciskiem *Delete* w oknie *Flow Properties*.	Flows → Delete
Lista przepływów (*Search/List Flows*)	Kliknij przycisk menu w górnym pasku i wybierz *List Flows*, aby zobaczyć listę wszystkich przepływów.	–
Nawigacja między przepływami	Przełączanie się między zakładkami	–

Rys. 7. przedstawia kolejność czynności przy tworzeniu aplikacji Node-RED.

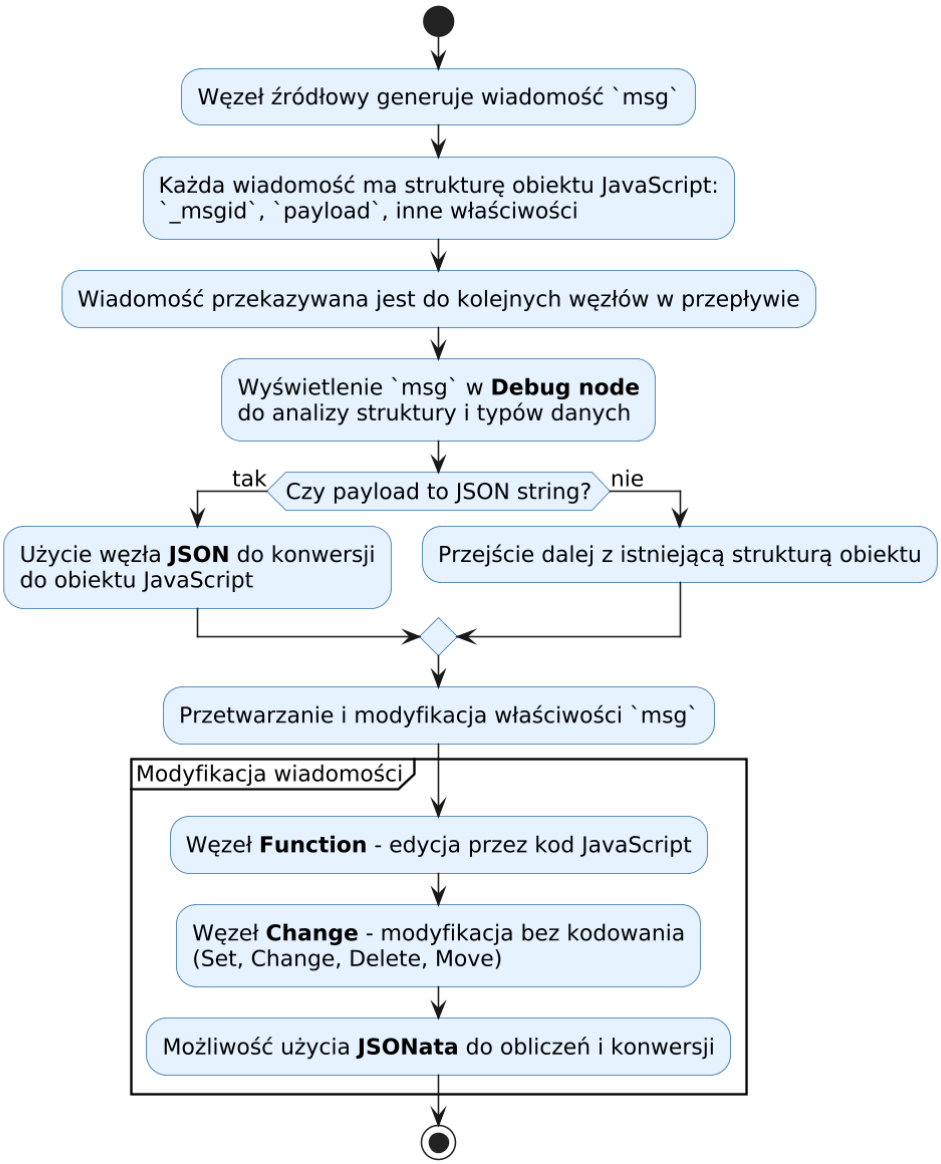


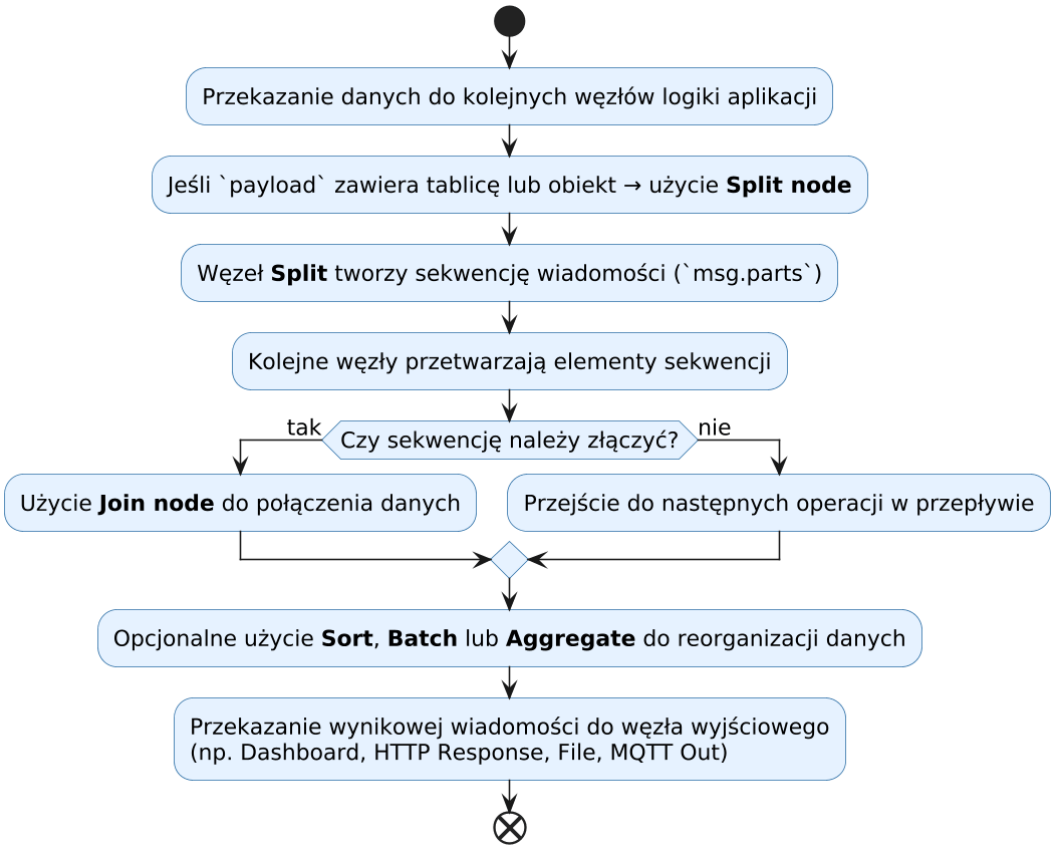
Rys. 7. Cykl tworzenia aplikacji w Node-RED.
Źródło: opracowanie własne.

2.4. Cykl przetwarzania wiadomości

Diagram aktywności (Rys. 8) ilustruje typowy cykl przetwarzania wiadomości w Node-RED – od ich utworzenia, przez analizę struktury i zmianę właściwości (*Function, Change, JSON*), aż po przetwarzanie sekwencji za pomocą węzłów *Split i Join*. Pokazuje typowy cykl życia obiektu *msg*, obejmujący konwersję, przetwarzanie i przesyłanie danych do węzłów wyjściowych aplikacji.

[Tekst alternatywny: diagram aktywności przedstawia pełny cykl przetwarzania wiadomości w Node-RED: od utworzenia obiektu `msg`, poprzez analizę struktury, modyfikację właściwości, aż po operacje na sekwencjach.]





Rys. 8. Cykl przetwarzania wiadomości (*msg*) w Node-RED.
Źródło: opracowanie własne.

W środowisku Node-RED przepływy działają poprzez przekazywanie wiadomości (`msg`) pomiędzy węzłami. Każda wiadomość to obiekt JavaScript, który może mieć dowolny zestaw właściwości. Najczęściej wykorzystywaną z nich jest `msg.payload`, zawierająca dane przetwarzane przez większość węzłów. Każda wiadomość posiada również identyfikator `_msgid`, umożliwiający śledzenie jej w przepływie (Listing 1).

Listing 1. Przykład struktury wiadomości

```
{
  "_msgid": "12345",
  "payload": "Temperatura: 23.4°C",
  "topic": "sensor/temperature",
  "timestamp": 1730531600000
}
```

Węzeł *Debug* pozwala analizować zawartość wiadomości i strukturę danych. Domyślnie pokazuje `msg.payload`, ale można ustawić, by wyświetlał inne właściwości lub cały obiekt. Po kliknięciu przycisku w panelu bocznym można skopiować ścieżkę dostępu do właściwości (np. `payload.data[0].value`) i wykorzystać ją w węźle *Function* lub *Change*.

Jeśli dane przychodzą w formacie JSON jako łańcuch tekstowy, należy przekonwertować je na obiekt JavaScript (Listing 2).

Listing 2. Przykład konwersji w węźle *Function*.

```
/* Fragment kodu pobiera reprezentację ciągu znaków obiektu w formacie JSON i
konwertuje ją na rzeczywisty obiekt JavaScript. */
// msg.payload = '{"temperature": 22.5, "humidity": 45}'
msg.payload = JSON.parse(msg.payload);
return msg;
```

Operacja odwrotna (Listing 3).

Listing 3. Przykład konwersji w węźle *Function*.

```
/* Fragment kodu konwertuje obiekt `msg.payload` na ciąg znaków JSON przy użyciu
metody `JSON.stringify()`.Następnie przypisuje ciąg znaków JSON z powrotem do
`msg.payload` i zwraca zmodyfikowany obiekt `msg`. */
msg.payload = JSON.stringify(msg.payload);
return msg;
```

Wiadomości można modyfikować węzłem *Change* (bez kodowania) lub *Function* z wykorzystaniem JavaScriptu (Listing 4).

Listing 4. Przykład modyfikacji wiadomości w węźle *Function*.

```
/* Fragment kodu konwertuje wartość temperatury z Fahrenheita na Celsjusza.
Najpierw pobiera wartość temperatury z obiektu `msg.payload`,
a następnie oblicza równoważną temperaturę w stopniach Celsjusza,
korzystając z wzoru `(Fahrenheit - 32) * 5 / 9`. Wynik jest następnie przypisywany
do nowej właściwości `temperature_c` w obiekcie `msg.payload`. */
let temp = msg.payload.temperature;
msg.payload.temperature_c = (temp - 32) * 5 / 9;
return msg;
```

Wynik przekształceń wiadomości `msg` przedstawiono w Listing 5.

Listing 5. Wynik modyfikacji wiadomości.

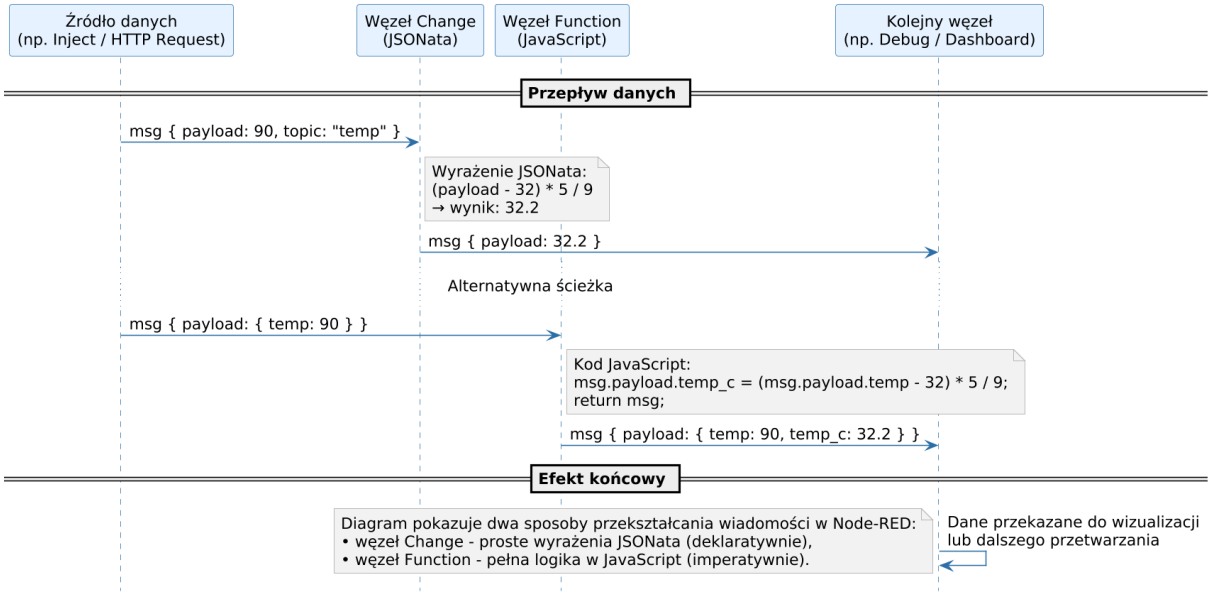
```
{
  "payload": {
    "temperature": 90,
    "temperature_c": 32.2 }
}
```

Węzeł Change wspiera również JSONata, np.:

```
msg.payload.temperature_c = (msg.payload.temperature - 32) * 5 / 9
```

JSONata to język zapytań dla JSON, który pozwala na tworzenie transformacji danych w Node-RED w sposób prosty, czytelny i bezpieczny. Jest szczególnie przydatny w węźle *Change*, gdy potrzebujesz zmienić lub obliczyć wartość pola w wiadomości (`msg`) bez pisania kodu w węźle *Function* (Rys. 8).

[Tekst alternatywny: Diagram pokazuje przepływ danych w Node-RED od źródła (np. Inject lub HTTP Request) przez węzły Change i Function aż do Debug lub Dashboard. Przedstawia dwa sposoby przekształcania danych: deklaratywny (JSONata w węźle Change) i imperatywny (JavaScript w węźle Function).



Rys. 9. Przepływ wiadomości w Node-RED: JSONata vs JavaScript. Źródło: opracowanie własne.

JSONata umożliwia szybkie i proste przekształcenia danych bez pisania kodu, ma zastosowanie w szczególności do wyrażeń w węźle *Change* lub *Switch:*

- szybkie filtrowanie i przekształcanie danych z API, czujników lub baz,
- tworzenie uproszczonych struktur JSON,
- agregowanie lub obliczanie wartości w przepływach,
- zamiana nazw pól bez pisania funkcji JavaScript.

JavaScript jest stosowany w węźle *Function* i zapewnia pełną kontrolę nad przepływem programu; używany do logiki biznesowej, warunków, pętli i przetwarzania wielu źródeł danych.

Przykład: filtrowanie obiektów w JavaScript i JSONata (Listing 6).

Listing 6. Filtrowanie obiektów: JavaScript vs. JSONata.

```
/* Fragment kodu JavaScript filtruje obiekty na podstawie właściwości `type`
w tablicy czujników w obiekcie `msg.payload`. W szczególności filtruje tylko
 obiekty czujników, które mają właściwość `type` równą „humidity”. */
msg.payload = msg.payload.sensors.filter(s => s.type === "humidity");
return msg;

/* Wyrażenie JSONata do wyodrębnienia wartości czujnika wilgotności*/
payload.sensors[type="humidity"]
```

Przykład: tworzenie tekstu wynikowego (Listing 7).

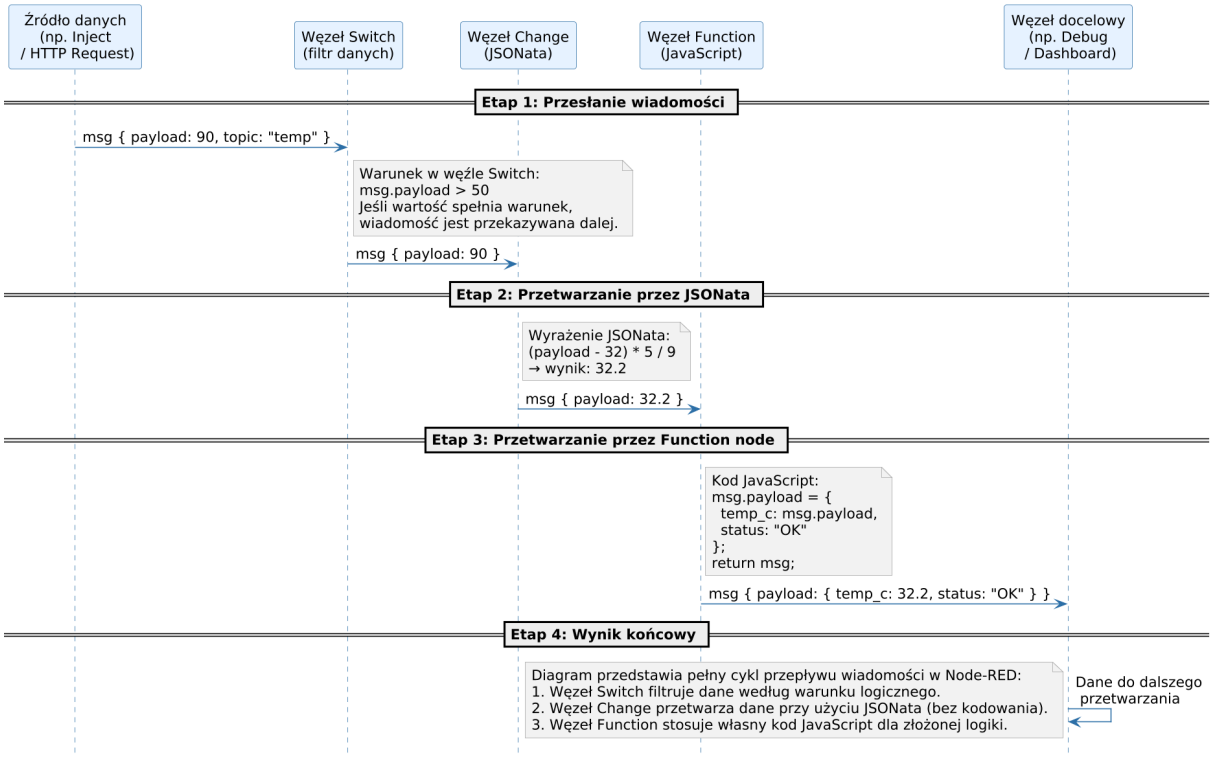
Listing 7. Tworzenie tekstu wynikowego: JavaScript vs. JSONata.

```
/* Fragment kodu JavaScript formatuje treść wiadomości, aby zawierała nazwę miasta
i temperaturę  w stopniach Celsjusza. Wykorzystuje literały szablonowe do
połączenia nazwy miasta, wartości temperatury i jednostki „°C” w jeden ciąg
znaków. Na koniec przypisuje ten sformatowany ciąg znaków z powrotem do
właściwości `msg.payload` i zwraca zmodyfikowany obiekt `msg`.*/
msg.payload = `${msg.payload.city}: ${msg.payload.temp}°C`;
return msg;

/* Wyrażenie JSONata łączy wartości „payload.city”, „payload.temp” oraz ciągi
znaków „:” i „°C”. W rezultacie powstaje ciąg znaków reprezentujący nazwę
miasta, temperaturę i jednostkę w formacie „miasto: temperatura°C”.*/
payload.city & ": " & payload.temp & "°C"
```

Schemat przepływu danych w środowisku Node-RED (Rys. 10) przedstawia pełny proces przetwarzania wiadomości `msg` – od jej utworzenia w węźle źródłowym, poprzez etap filtrowania w węźle *Switch*, aż po transformację w węźle *Change* z użyciem wyrażenia JSONata i dalsze przetwarzanie logiczne w węźle *Function* z kodem JavaScript. Diagram ilustruje typowy tok pracy w Node-RED, w którym dane przechodzą przez kolejne węzły odpowiadające za selekcję, obliczenia i przygotowanie wyników do wizualizacji lub publikacji w systemach zewnętrznych.

[Tekst alternatywny: diagram przedstawia przepływ wiadomości w aplikacji Node-RED, w której dane z węzła źródłowego są filtrowane przez Switch, przekształcane w węźle Change z użyciem wyrażenia JSONata, a następnie przetwarzane w węźle Function przy użyciu kodu JavaScript przed przekazaniem do węzła wyjściowego.



Rys. 10. Proces przetwarzania wiadomości w Node-RED: JSONata + JavaScript. Źródło: opracowanie własne.

Niektóre węzły (np. *Split* i *Join*) przetwarzają zestawy wiadomości powiązanych ze sobą w sekwencję. Każdy element sekwencji zawiera właściwość msg.parts, opisującą jego pozycję (Listing 8).

Listing 8. Przetwarzanie zestawów wiadomości.

```
/* Obiekt JSON reprezentuje zawartość wiadomości */
{  "payload": 42,
  "parts": {
    "id": "abcd1234",
    "index": 2,
    "count": 5
  }
}
// Węzeł Split dzieli dane, np. tablicę:
{ "payload": [10, 20, 30] }
// na serię wiadomości:
{ "payload": 10, "parts": { "index": 0 } }
{ "payload": 20, "parts": { "index": 1 } }
{ "payload": 30, "parts": { "index": 2 } }
// Węzeł Join scala je z powrotem w pojedynczą wiadomość:
{ "payload": [10, 20, 30] }
```

3. Przykład aplikacji w Node-RED

Główną funkcją aplikacji jest automatyczne pobieranie aktualnej temperatury dla Kielc z serwisu internetowego co 15 minut i jej wyświetlanie w panelu diagnostycznym Node-RED. W projekcie użyto darmowego API do pobierania aktualnych danych pogodowych dla Kielc z platformy Open-Meteo: https://api.open-meteo.com/v1/forecast?latitude=50.87&longitude=20.63&current_weather=true

Po wejściu na ten adres (lub wysłaniu do niego zapytania z aplikacji) można otrzymać odpowiedź w formacie JSON, zawierającą aktualne dane pogodowe dla Kielc (Listing 9).

Listing 9. Dane serwisu pogodowego w formacie JSON

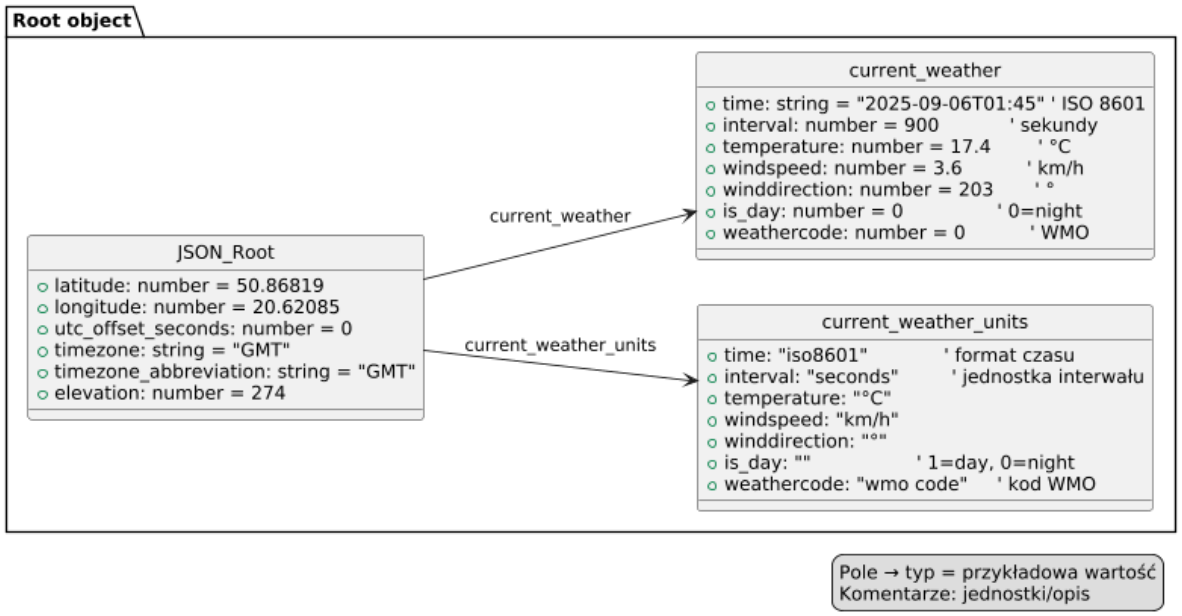

```
{{
latitude: 50.86819,
longitude: 20.62085,
utc_offset_seconds: 0,
timezone: "GMT",
timezone_abbreviation: "GMT",
elevation: 274,
current_weather_units: {
time: "iso8601",
interval: "seconds",
temperature: "°C",
windspeed: "km/h",
winddirection: "°",
is_day: "",
weathercode: "wmo code"
},
current_weather: {
time: "2025-09-06T01:45",
interval: 900,
temperature: 17.4,
windspeed: 3.6,
winddirection: 203,
is_day: 0,
weathercode: 0
}}
```


Diagram (

Rys. 11) przedstawia strukturę odpowiedzi JSON (tzw JSON contract) z Open-Meteo. W węźle JSON_Root znajdują się pola lokalizacyjne i metadane

[Tekst alternatywny: Diagram przedstawia strukturę obiektu JSON z danymi pogodowymi, obejmującą główny obiekt JSON_Root z pozycją geograficzną oraz

dwa powiązane obiekty: current_weather (bieżące dane pogodowe) i current_weather_units (jednostki miar)].



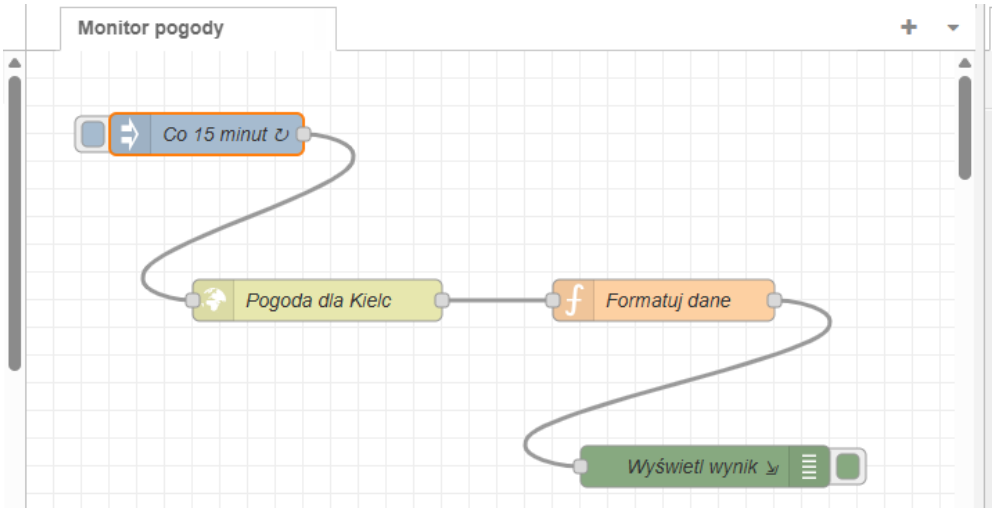
Rys. 11. Diagram struktury danych JSON pobieranych z serwera Open-Meteo. Źródło: opracowanie własne.

Dwie relacje prowadzą do obiektów zagnieżdżonych:

1. current_weather – właściwe dane pomiarowe z przykładowymi wartościami (czas ISO-8601, interwał odświeżania w sekundach, temperatura w °C, prędkość wiatru w km/h, kierunek w stopniach, flaga dnia/nocy, kod WMO).
2. current_weather_units – słownik jednostek i formatów dla odpowiadających pól (np. "°C", "km/h", "seconds", "iso8601").

Zrzut ekranu (Rys. 12) przedstawia przepływ (flow) aplikacji zbudowanej w środowisku Node-RED, służącej do cyklicznego monitorowania temperatury powietrza w Kielcach.

[Tekst alternatywny: lustracja przedstawia gotowy przepływ aplikacji w edytorze Node-RED. Aplikacja „Monitor pogody” automatycznie pobiera i wyświetla dane pogodowe. Przepływ składa się z czterech połączonych ze sobą węzłów (*nodes*), które wykonują zadania w określonej sekwencji].



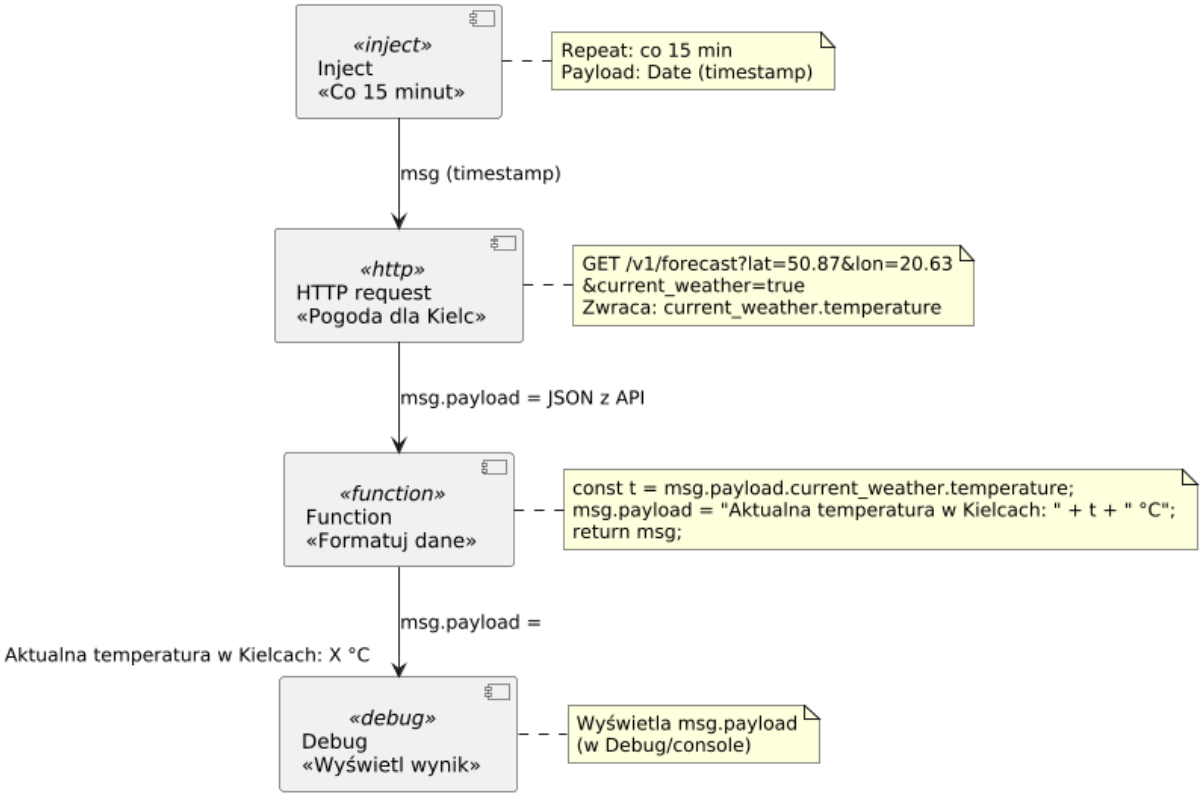
Rys. 12. Przykładowa aplikacja w Node-RED.
Źródło: zrzut ekranu.

Kod programu (Listing 10) w formacie JSON (serializowanym) można skopiować i importować doNode-RED w celu uruchomienia aplikacji.

Listing 10. Kod aplikacji Monitor pogody

```
[{"id":"d5323bd6fbbc0c4d","type":"tab","label":"Monitor
pogody","disabled":false,"info":"","env":[]},{"id":"a1b2c3d4.e5f6g7","type
":"inject","z":"d5323bd6fbbc0c4d","name":"Co 15
minut","props":[{"p":"payload"}],"repeat":"900","crontab":"","once":true,"
onceDelay":0.1,"topic":"","payload":"","payloadType":"date","x":130,"y":60
,"wires":[["b2c3d4e5.f6g7h8"]]},{"id":"b2c3d4e5.f6g7h8","type":"http
request","z":"d5323bd6fbbc0c4d","name":"Pogoda dla
Kielc","method":"GET","ret":"obj","paytoqs":"ignore","url":"https://api.op
en-
meteo.com/v1/forecast?latitude=50.87&longitude=20.63¤t_weather=true","tls"
:"","persist":false,"proxy":"","insecureHTTPParser":false,"authType":"","s
enderr":false,"headers":[],"x":330,"y":60,"wires":[["c3d4e5f6.g7h8i9"]]},{
"id":"c3d4e5f6.g7h8i9","type":"function","z":"d5323bd6fbbc0c4d","name":"Fo
rmatuj dane","func":"// Odczytaj temperaturę z obiektu JSON\nconst temp =
msg.payload.current_weather.temperature;\n\n// Przygotuj nową, prostą
wiadomość\nmsg.payload = \"Aktualna temperatura w Kielcach: \" + temp + \"
°C\";\n\n// Zwróć wiadomość do następnego węzła\nreturn
msg;","outputs":1,"timeout":"","noerr":0,"initialize":"","finalize":"","li
bs":[],"x":320,"y":180,"wires":[["d4e5f6g7.h8i9j0"]]},{"id":"d4e5f6g7.h8i9
j0","type":"debug","z":"d5323bd6fbbc0c4d","name":"Wyświetl
wynik","active":true,"tosidebar":true,"console":false,"tostatus":true,"com
plete":"payload","targetType":"msg","statusVal":"","statusType":"counter",
"x":520,"y":180,"wires":[]}]
```

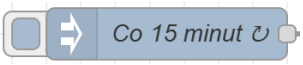
Schemat (Rys. 13) ilustruje sekwencję przetwarzania: od cyklicznego wywołania zapytania co 15 minut (węzeł *Inject*), przez pobranie danych pogodowych z API (węzeł HTTP request), po ich obróbkę w skrypcie JavaScript (węzeł *Function*) i wyświetlenie wyniku w konsoli debugowania (węzeł *Debug*). Diagram pokazuje przepływ komunikatu *msg* między kolejnymi węzłami, uwzględniając zarówno format danych, jak i sposób ich przetwarzania na poszczególnych etapach działania aplikacji.

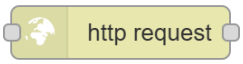
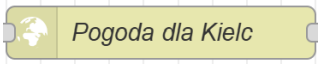
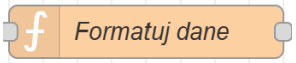
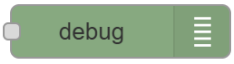


Rys. 13. Diagram przepływu danych w aplikacji Monitor pogody.
Źródło: opracowanie własne.

Przepływ aplikacji składa się z czterech połączonych ze sobą węzłów (*nodes*), które wykonują zadania w określonej sekwencji (Tabela 6).

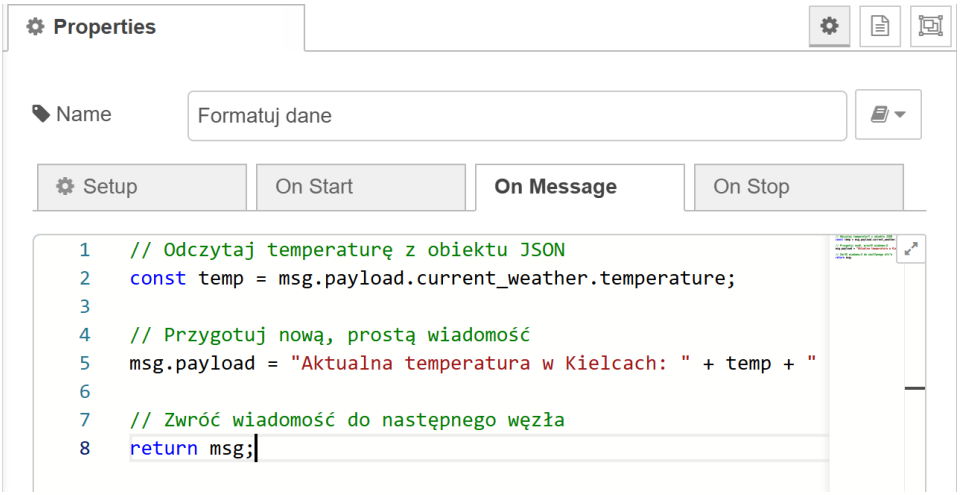
Tabela 6. Węzły użyte w przykładowej aplikacji Monitor pogody

Użyty węzeł	Opis węzła	Implementacja w aplikacji
	Wprowadza komunikat do przepływu ręcznie lub w regularnych odstępach czasu. Treść komunikatu może mieć różne typy, w tym ciągi znaków, obiekty JavaScript lub aktualny czas.	 Automatycznie uruchamia cały przepływ co kwadrans.

	Wysyła żądania HTTP i zwraca odpowiedź.	 Wysyła zapytanie do zewnętrznego serwisu API w celu pobrania aktualnych danych pogodowych dla Kielc.
	Funkcja JavaScript uruchamiana dla komunikatów odbieranych przez węzeł. Komunikaty są przekazywane jako obiekt JavaScript o nazwie *msg*. Zgodnie z konwencją będzie on posiadał właściwość *msg.payload* zawierającą treść komunikatu.	 Przetwarza surową odpowiedź z API (w formacie JSON), aby wyodrębnić tylko interesujące informacje (temperaturę) i uformować je w czytelny tekst.
	Wyświetla wybrane właściwości komunikatu w zakładce *Debug* paska bocznego edytora i opcjonalnie w dzienniku czasu wykonywania. Domyślnie wyświetla *msg.payload*, ale można skonfigurować wyświetlanie dowolnej właściwości, pełnego komunikatu lub wyniku wyrażenia JSON.	 Odbiera sformatowane dane i wyświetla je w panelu bocznym edytora, umożliwiając użytkownikowi odczytanie wyniku.

Zrzut ekranu (Rys. 14) przedstawia konfigurację węzła „Formatuj dane” w aplikacji Node-RED, odpowiedzialnego za przetwarzanie danych pogodowych

[Tekst alternatywny: zrzut ekranu przedstawia edytor węzła Function w środowisku Node-RED o nazwie „Formatuj dane”, który zawiera prosty skrypt JavaScript. Skrypt odczytuje temperaturę z obiektu JSON pobranego z API pogodowego, formatuje komunikat tekstowy „Aktualna temperatura w Kielcach: X °C” i przekazuje go dalej.



Rys. 14. Konfiguracja węzła „Formatuj dane” w aplikacji Monitor pogody. Źródło: zrzut ekranu.

Kroki, które należy wykonać w celu zbudowania przepływu danych w Node-RED:

1. **Stworzenie wyzwalacza czasowego (***węzeł będzie automatycznie "wstrzykiwał" wiadomość do przepływu co 15 minut, uruchamiając całą sekwencję*):
   - Przeciągnij węzeł **inject** z lewej palety na środek obszaru roboczego.
   - Kliknij go dwukrotnie, aby otworzyć konfigurację.
   - W polu „Payload" pozostaw domyślną wartość (timestamp).
   - W sekcji „Repeat" wybierz opcję **„interval"** i ustaw interwał na **15 minut**.
   - Zatwierdź przyciskiem „Done".
2. **Zapytanie o dane pogodowe** *(ten węzeł wyśle zapytanie pod wskazany URL i odbierze dane pogodowe w formacie JSON)***:**
   - Przeciągnij węzeł **http request** i umieść go na prawo od węzła inject.
   - Połącz wyjście węzła inject z wejściem węzła http request.
   - Kliknij dwukrotnie węzeł http request, aby go skonfigurować.
   - W polu **URL** wklej następujący adres darmowego API pogodowego (dla Kielc).
   - Upewnij się, że w polu "Return" jest wybrana opcja **„a parsed JSON object"**.
   - Zatwierdź przyciskiem „Done".
3. **Przetworzenie Danych** *(węzeł wyciągnie z całej odpowiedzi JSON tylko interesującą nas informację (temperaturę) i sformatuje ją w czytelny teksy..)*
   - Przeciągnij węzeł **function** i umieść go na prawo od węzła http request.
   - Połącz oba węzły.
   - Kliknij dwukrotnie węzeł function i w edytorze kodu wklej poniższy fragment JavaScript:

```
// Odczytaj temperaturę z obiektu JSON otrzymanego z API
const temp = msg.payload.current_weather.temperature;

// Przygotuj nową, prostą wiadomość tekstową
msg.payload = "Aktualna temperatura w Kielcach: " + temp + " °C";

// Zwróć wiadomość, aby przekazać ją dalej
return msg;
```

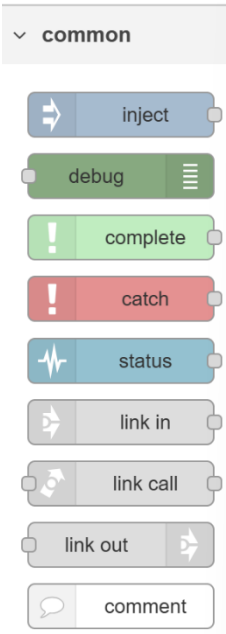
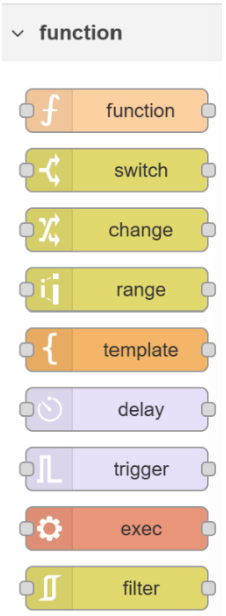
   - Zatwierdź przyciskiem „Done".
4. **Wyświetlenie Wyniku**
   - Przeciągnij węzeł debug i umieść go na końcu przepływu.
   - Połącz wyjście węzła function z wejściem węzła debug.
   - Upewnij się, że węzeł debug jest aktywny (przycisk po prawej stronie węzła jest zielony).

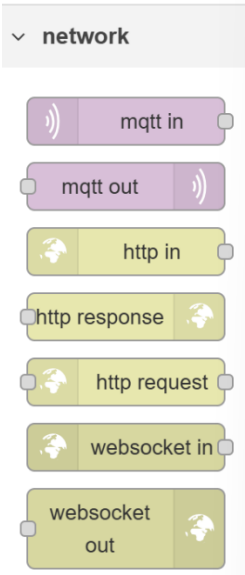
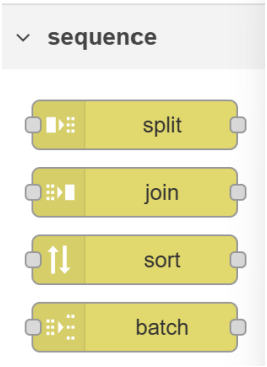
Aby uruchomić aplikację, należy kliknąć czerwony przycisk „Deploy" w prawym górnym rogu ekranu.

4. Węzły i języki przetwarzania danych w Node-RED

4.1. Podstawowe węzły Node-RED

Tabela 7. Podstawowe węzły Node-RED.

Paleta	Nazwa węzła	Opis działania
	Węzły z kategorii **common** służą do **podstawowej obsługi przepływu danych**, testowania, łączenia fragmentów flow i monitorowania działania aplikacji.	
	inject	Pozwala ręcznie, cyklicznie lub przy starcie Node-RED wstrzykiwać wiadomości (msg) do przepływu w celu testowania lub inicjowania procesu.
	debug	Wyświetla dane wiadomości w panelu Debug. Pomaga analizować zawartość msg.payload lub innych właściwości.
	complete	Reaguje, gdy inne węzły zakończą swoje działanie – np. przy zakończeniu wykonania zapytania HTTP lub funkcji.
	catch	Przechwytuje błędy generowane przez inne węzły w tym samym przepływie. Pozwala obsługiwać wyjątki.
	status	Nasłuchuje i raportuje zmiany statusu węzłów (np. połączenie MQTT, HTTP). Może wyświetlać je lub przekazywać dalej.
	link in	Umożliwia połączenie przepływów między różnymi kartami (*Tab*). Odbiera wiadomości od węzłów link out lub link call.
	link call	Pozwala wywoływać przepływ z innego miejsca, jak funkcję. Wysyła wiadomość do powiązanego węzła link in i czeka na odpowiedź.
	link out	Wysyła wiadomości do powiązanego węzła link in lub zwraca wynik do link call. Łączy przepływy bez fizycznych przewodów.
	comment	Dodaje komentarz tekstowy w edytorze. Pomaga dokumentować i opisywać poszczególne fragmenty przepływu.
	Węzły z tej grupy należą do kategorii **function** i służą głównie do **przetwarzania oraz kontroli przepływu danych**.	
	function	Pozwala pisać własny kod JavaScript do przetwarzania danych w wiadomości msg. Umożliwia dowolne modyfikacje logiki przepływu.
	switch	Kieruje wiadomość na różne wyjścia w zależności od warunku logicznego (np. wartość msg.payload). Działa jak instrukcja if.
	change	Umożliwia zmianę właściwości wiadomości (np. msg.payload), ustawianie wartości, kopiowanie, kasowanie lub zamianę tekstu.
	range	Przeskalowuje wartość liczbową z jednego zakresu na inny (np. z 0–1023 na 0–100). Idealny przy pracy z czujnikami.
	template	Generuje tekst lub HTML z wykorzystaniem szablonów (np. do tworzenia wiadomości e-mail, raportów, stron WWW).
	delay	Wprowadza opóźnienie czasowe w przekazywaniu wiadomości lub ogranicza częstotliwość wysyłania (rate limit).
	trigger	Wysyła wiadomość po określonym czasie, cyklicznie lub jako reakcję na sygnał wejściowy (np. do tworzenia impulsów).
	exec	Uruchamia zewnętrzne polecenia systemowe lub skrypty (np. python, bash) i przekazuje wynik do przepływu.
	filter	Przepuszcza tylko te wiadomości, które spełniają określony warunek (np. zgodność z wzorcem lub zakresem wartości).

	Węzły z tej grupy służą do **komunikacji sieciowej** i integracji Node-RED z urządzeniami IoT, usługami webowymi lub przeglądarkami (HTTP, MQTT, WebSocket).	
	mqtt in	Odbiera wiadomości z brokera MQTT (np. Mosquitto). Nasłuchuje wybranego tematu (topic) i przekazuje dane dalej w przepływie.
	mqtt out	Publikuje wiadomości do brokera MQTT na określony temat. Umożliwia wysyłanie danych do urządzeń IoT lub serwerów.
	http in	Tworzy punkt końcowy (endpoint) HTTP do odbierania żądań metodą GET, POST, PUT itp. Jest początkiem przepływu webowego.
	http response	Wysyła odpowiedź HTTP do klienta, który wysłał żądanie do węzła http in. Umożliwia zwracanie tekstu, JSON-a, HTML itp.
	http request	Wysyła zapytania HTTP do zewnętrznych serwerów (np. API). Obsługuje metody GET, POST, PUT, DELETE i zwraca wynik.
	websocket in	Odbiera wiadomości przez połączenie WebSocket, umożliwiając komunikację w czasie rzeczywistym między przeglądarką a serwerem Node-RED.
	websocket out	Wysyła wiadomości przez połączenie WebSocket do klientów podłączonych w czasie rzeczywistym. Stosowany np. w dashboardach i aplikacjach live.
	Węzły **sequence** służą do **zarządzania strumieniami wiadomości** — ich dzieleniem, scalaniem, porządkowaniem i grupowaniem, co jest przydatne przy przetwarzaniu danych w partiach lub kolejkach.	
	split	Dzieli jedną wiadomość na wiele mniejszych – np. rozbija tablicę lub tekst na pojedyncze elementy wysyłane kolejno w przepływie.
	join	Łączy wiele wiadomości w jedną – np. scala elementy tablicy lub obiekty JSON w całość.
	sort	Sortuje dane w sekwencji wiadomości według wybranej właściwości (np. msg.payload).
	batch	Grupuje wiadomości w partie (batch), np. po określonej liczbie elementów lub czasie, aby je przetwarzać razem.
	Węzły z kategorii **parser** służą do **konwersji i analizy formatów danych** (CSV, HTML, JSON, XML, YAML), co umożliwia Node-RED współpracę z różnorodnymi źródłami informacji i API.	
	csv	Konwertuje dane między formatem CSV a obiektami JSON. Umożliwia wczytywanie i eksport danych tabelarycznych.
	html	Parsuje kod HTML i umożliwia ekstrakcję danych za pomocą selektorów CSS (np. , #id, .class).
	json	Konwertuje dane między formatem tekstowym JSON a obiektem JavaScript (msg.payload). Bardzo często używany.
	xml	Zamienia dane między formatem XML a obiektem JavaScript. Pozwala analizować dane z systemów starszego typu.
	yaml	Przekształca dane między formatem YAML a JSON/JavaScript. Użyteczny przy integracji z systemami DevOps lub IoT.

4.2. Elementy składni Języka JavaScript

Tabela 8. Podstawowe elementy składni JavScript.

Temat	Składnia	Przykład
Deklaracje zmiennych	let, const, var	let x=1; const PI=3.14; var stara="tak";
Typy podstawowe	number, string, boolean, null, undefined, symbol, bigint	typeof 42 // "number"
Template literals	`tekst ${zmienna}`	const n=3; `Mam ${n} jabłka`;
Operatory arytm./porówn.	+ - * / % **, === !== > <	2**3===8 // true
Logiczne i nullish	&&, `	
Instrukcja if	if (...) { ... } else { ... }	if(x>0){ok()} else{nope()}
Switch	switch (x) {case 1: ...}	switch(d){case 6: w="Sob"; break;}
Pętle	for, while, for...of	for (const v of arr) log(v)
Pętla po kluczach	for...in	for (const k in obj) log(k,obj[k])
Funkcja deklaracja	function f(a,b){...}	function sum(a,b){return a+b}
Parametry domyślne	function f(a=1){...}	f(); // a=1
Rest/Spread	...args, ...array	function f(...xs){} / [...arr1,...arr2]
Destrukturyzacja obiektów	{a,b}=obj	const {x,y:yy}=pt
Destrukturyzacja tablic	[a,b]=arr	const [first,,third]=xs
Obiekt literał	{ klucz: wartość }	const u={id:1,name:"A"}
Metody tablic	map/filter/reduce	[1,2,3].map(x=>x*2)
Klasa	class Nazwa { constructor(){...} met(){...} }	class P{constructor(n){this.n=n}}
Dziedziczenie	extends	class B extends A { ... }
Obsługa błędów	try { ... } catch(e) { ... } finally { ... }	try{JSON.parse(x)}catch(e){...}
JSON	JSON.parse, JSON.stringify	JSON.stringify({a:1})
Data/Czas	new Date()	new Date().toISOString()
Timer	setTimeout, setInterval	setTimeout(()=>doIt(),1000)

4.3. Elementy składni JSONata

Tabela 9. Podstawowe elementy składni JSONata.

Temat	Składnia	Przykład
Dostęp do pola	fieldName	name → "Jan"
Dostęp zagnieżdżony	parent.child	address.city → "Warszawa"
Filtracja	array[condition]	people[age > 30]
Iteracja po tablicy	array.field	people.name → ["Jan", "Anna"]
Operacje matematyczne	+ - * /	price * quantity
Łączenie tekstu	&	"Witaj " & name → "Witaj Jan"
Funkcje wbudowane	function(args)	length("test") → 4
Tworzenie obiektów	{ "key": value }	{ "fullName": name & " " & surname }
Warunki (if-else)	condition ? then : else	age >= 18 ? "dorosły" : "dziecko"
Sprawdzanie istnienia	field ? default	name ? name : "brak"
Zmienne lokalne	($var := expr)	($x := 5) + $x → 10
Indeks tablicy	array[index]	people[0].name → "Jan"
Filtracja z indeksem	array[$index = value]	items[$index = 2]
Operatory logiczne	and, or, not	age > 18 and country = "PL"
Przekształcenie na liczbę	$number(string)	$number("123") → 123
Przekształcenie na tekst	$string(value)	$string(123) → "123"
Długość tablicy lub tekstu	$count(array), length(string)	$count(people) → 5
Wyszukiwanie w tablicy	$contains(array, value)	$contains(tags, "node-red") → true
Usuwanie duplikatów	$distinct(array)	$distinct(["a", "b", "a"]) → ["a", "b"]
Data i czas (ISO)	$now(), $fromMillis(ms), $formatDateTime()	$formatDateTime($now(), "[Y0001]-[M01]-[D01]")
Łączenie tablic	$append(array1, array2)	$append([1,2], [3]) → [1,2,3]
Sortowanie tablicy	$sort(array)	$sort([3, 1, 2]) → [1, 2, 3]
Podciągi tekstu	substring(text, start, length)	substring("abcdef", 2, 3) → "cde"
Parsowanie JSON	$eval('{"a":1}')	$eval('{"x":5}').x → 5
JSON jako tekst	$stringify(object)	$stringify({"a":1}) → '{"a":1}'

4.4. Porównanie JSONata i Javascript

Tabela 10. Porównanie JSONata i Javascript.

Aspekt	JSONata	JavaScript
Miejsce użycia	Węzeł *Change*, *Switch*, *Join, Split*, niektóre właściwości Dashboard	Węzeł *Function*
Sposób działania	Deklaratywny (opisujesz *co* chcesz uzyskać)	Imperatywny (piszesz *jak* to zrobić)
Typowe zastosowanie	Proste obliczenia, filtrowanie, łączenie pól, zmiana struktury danych	Złożone logiki sterowania, pętle, warunki, operacje asynchroniczne
Dostęp do danych	Krótkie ścieżki typu payload.temp	Jawne użycie obiektu msg, np. msg.payload.temp
Obsługa kontekstu	Może odczytywać wartości z kontekstu flow i global	Może odczytywać i zapisywać w flow, global przez API (flow.get(), flow.set())
Tworzenie nowych struktur	Bardzo proste, np.: {"miasto": payload.city, "temp": payload.temp }	Wymaga konstrukcji obiektów, np.: msg.payload = { city: msg.payload.city, temp: msg.payload.temp }
Filtrowanie tablic	Wbudowana składnia, np.: payload.sensors[type="temp"]	Wymaga pętli lub Array.filter()
Konwersje jednostek i proste obliczenia	Jednolinijkowe wyrażenie, np.: (payload.temp - 32) * 5 / 9	Wymaga przypisania w kodzie, np.: msg.payload = (msg.payload.temp - 32) * 5 / 9
Obsługa błędów	Brak mechanizmu try/catch – błędy powodują przerwanie obliczeń	Pełna obsługa wyjątków (try/catch, if/else, walidacja danych)
Wydajność	Szybsza przy prostych transformacjach	Lepsza przy złożonych przetwarzaniach
Złożoność składni	Bardzo prosta, przypomina ścieżki JSON	Pełny JavaScript – wymaga większej znajomości składni
Kiedy używać	Gdy wystarczy prosty wzór lub formuła	Gdy potrzebna jest logika warunkowa, pętle, interakcje z systemem lub API
Przykładowe węzły korzystające	Change, Switch, Join, Split	Function, Template (dla HTML), HTTP Request (callback)

Literatura

1. Buchwald, P. i in. (2022) *Internet rzeczy i jego przemysłowe zastosowania*. W: R. Knosala (red.). Warszawa: Polskie Wydawnictwo Ekonomiczne.
2. Ćwikła, G. i in. (2021) *Wspomaganie informacyjne menadżerów produkcji*. W: R. Knosala (red.). Warszawa: Polskie Wydawnictwo Ekonomiczne.
3. Freeman, E. i Robson, E. (2025) *Programowanie w JavaScript. Rusz głową! Wydanie II.* Gliwice: Helion.
4. Hagino, T. and O'Leary, N. (2021) *Practical Node-RED Programming: Learn powerful visual programming techniques and best practices for the web and IoT.* Packt Publishing.
5. Hermann, M., Pentek, T. and Otto, B. (2016) 'Design principles for industrie 4.0 scenarios', *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2016-March, pp. 3928–3937. doi:10.1109/HICSS.2016.488.
6. JavaScript Tutorial (2025) *W3Schools Online Web Tutorials.* Available at: https://www.w3schools.com/js/default.asp (Data dostępu: 29 września, 2025).
7. Kazała, R., Luściński, S., Strączyński, P. and Taneva, A. (2021) 'An enabling open-source technology for development and prototyping of production systems by applying digital twinning', *Processes*, 10(1), p. 21. doi:10.3390/PR10010021.
8. Lee, J., Bagheri, B. and Kao, H. (2015) 'A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems', *Manufacturing Letters*, 3, pp. 18–23.
9. Madsen, O., Berger, U., Møller, C., Lassen, H., Astrid, V., Waehrens, B. and Schou, C. (eds.) (2023) *The Future of Smart Production for SMEs: A Methodological and Practical Approach Towards Digitalization in SMEs.* Cham: Springer International Publishing.
10. Node-RED Documentation (no date) *Node-RED Docs.* Available at: https://nodered.org/docs/ (Data dostępu: 29 września, 2025).
11. Node.js – Run JavaScript Everywhere (no date) *Node.js.* Available at: https://nodejs.org/en (Data dostępu: 29 września, 2025).
12. Pivoto, D. G. S., de Almeida, L. F. F., da Rosa Righi, R., Rodrigues, J. J. P. C., Lugli, A. B. and Alberti, A. M. (2021) 'Cyber-physical systems architectures for industrial internet of things applications in Industry 4.0: A literature review', *Journal of Manufacturing Systems*, 58, pp. 176–192. doi:10.1016/j.jmsy.2020.11.017.

13. Tao, F., Qi, Q., Wang, L. and Nee, A. Y. C. (2019) 'Digital Twins and Cyber–Physical Systems toward Smart Manufacturing and Industry 4.0: Correlation and Comparison', *Engineering*, 5(4), pp. 653–661. doi:10.1016/j.eng.2019.01.014.
14. Tsbmail (2012) *Y.2060: Overview of the Internet of Things.* International Telecommunication Union. Available at: https://www.itu.int/rec/T-REC-Y.2060-201206-I (Data dostępu: 29 września, 2025).
15. Vogel-Heuser, B. and Wimmer, M. (eds.) (2023) *Digital Transformation: Core Technologies and Emerging Topics from a Computer Science Perspective.* Berlin, Heidelberg: Springer Vieweg Berlin Heidelberg.

Spis rysunków

Spis tabel

Spis listingów