Część III. Łączenie się z innymi rzeczami

Część IV. Łączenie się z chmurą

Przedmowa

Internet rzeczy (Internet of Things, IoT) ma szansę stać się jedną z najbardziej przełomowych zmian technologicznych naszych czasów. Jeśli zgodnie z prognozami w ciągu kilku następnych lat do sieci dołączonych zostanie ponad 50 miliardów urządzeń, IoT będzie miał ogromny wpływ na wszystkie branże. Opracowane zostaną nowe ekscytujące urządzenia podłączane i systemy pomocnicze, które będą rozwiązywać problemy z zakresu medycyny, transportu, rolnictwa, budownictwa, energetyki, produkcji i innych. Wiele z tych projektów już rozpoczęto i większość firm chce rozszerzyć swoje produkty i ofertę z wykorzystaniem IoT.

Internet rzeczy leży na przecięciu wielu technologii rozwijanych w ciągu ostatnich kilkudziesięciu lat i bazuje na innowacjach i redukcji kosztów w obszarze pomiarów, metrologii, mikroelektroniki, komunikacji bezprzewodowej, usług w chmurze oraz ekspansji sieci WWW. Choć tak szeroki zakres pociąga za sobą szereg wyzwań, stwarza również możliwość zaprojektowania i zbudowania oprogramowania z wyłamaniem się poza tradycyjne ścieżki kariery. Praca w IoT zmusza do wyjścia poza strefę komfortu i poznania pełnego spektrum inżynierii oprogramowania. W większości sytuacji rozwiązanie zahacza o wszystkie poziomy stosu technologicznego: począwszy od kodu typu bare-metal na urządzenia podłączonych, poprzez agregację przez węzły brzegowe o większych możliwościach przed skierowaniem do chmury. To oznacza, że programista musi wystarczająco dobrze znać każdy z poziomów, aby zrozumieć dostępne protokoły, języki, wzorce i platformy w celu zbudowania stabilnego i skalowalnego sytemu.

Programowanie Internetu rzeczy to rzetelnie przygotowany podręcznik budowania kompleksowego systemu IoT. Uważnie studiując tekst i wykorzystując materiały pomocnicze, czytelnicy zaimplementują uniwersalną platformę testową zbudowaną w oparciu o wzorce i protokoły wykorzystywane w prawdziwych komercyjnych systemach IoT. Zachęcam do zainwestowania w opcjonalne komponenty sprzętowe. Jednym z najbardziej satysfakcjonujących aspektów tej pracy jest integracja oprogramowania i świata fizycznego. Angażując się w prostą czynność kontrolowania świateł LED z chmury albo monitorowania warunków klimatycznych w pomieszczeniach, łatwo złapać bakcyla IoT.

Struktura tej książki i jej materiałów pomocniczych świadczy o tym, że Andy ma duże doświadczenie praktyczne nie tylko w zakresie IoT, ale także ogólnej inżynierii i architektury oprogramowania. Poszczególne etapy zostały celowo zaprojektowane tak, aby naśladować proces realizacji rzeczywistych projektów programistycznych, w oparciu o system zarządzania metodą Kanban wykorzystywaną przez większość zespołów programistycznych. Czytelnicy, którzy zrealizują każde ćwiczenie wraz z materiałami pomocniczymi, lepiej zrozumieją IoT i przy okazji zapoznają się z procesami i najlepszymi metodami powszechnie stosowanymi w procesach rozwoju oprogramowania.

Choć Andy omawia Internet rzeczy, schodząc do poziomu implementacji, moim zdaniem ta książka dostarcza również cennych informacji osobom pełniącym rolę liderów technologicznych. Studiując architekturę oprogramowania, która jest podstawą większości systemów IoT, można lepiej zrozumieć złożoność procesu jej implementacji. Każde z ćwiczeń jest cennym źródłem informacji o wyzwaniach, zagrożeniach i kompromisach związanych z danym segmentem systemu.

Zdecydowanie zalecam zwrócenie uwagi na liczne uwagi i odwołania dotyczące bezpieczeństwa w Internecie rzeczy. Jak w każdej innej technologii, presja, by dostarczyć rozwiązanie, może często przyćmić konieczność zapewnienia solidnych zabezpieczeń. Luki w zabezpieczeniach doprowadziły już do upadku wiele systemów IoT. Technologia ta rozpowszechniania się i w związku z tym będzie stawała się częścią wielu kluczowych systemów. Bezpieczeństwo musi być jednym z pierwszych aspektów rozważanych w każdym projekcie IoT i powinno być regularnie poddawane rewizji w cyklu życia rozwiązania.

Książka *Programowanie Internetu rzeczy* pokazuje programistom na dowolnym poziomie zaawansowania, jak dostać się do jednej z najszybciej rozwijających się dziedzin informatyki. Zachęcam do wnikliwego przestudiowania jej treści i rozważenia, jak wykorzystać podobne systemy do rozwiązania wyzwań napotykanych we własnym życiu. Mam nadzieję, że wkrótce zobaczę projekty utworzone przez inżynierów, którzy zainspirowani tą książką zajęli się budowaniem IoT.

Tim Strunck
Dyrektor działu Software Engineering
Atom Power Charlotte, Karolina Północna
Kwiecień 2021

Wstęp

Internet rzeczy (Internet of Things, IoT) to złożony, połączony „system systemów" składający się z różnych urządzeń sprzętowych, aplikacji, standardów danych, paradygmatów komunikacji oraz usług w chmurze. Dość trudno jest zdecydować, od czego zacząć własny projekt IoT. Książka *Programowanie Internetu rzeczy* została zaprojektowana tak, aby pomóc czytelnikom w rozpoczęciu przygody z budowaniem systemów IoT i pokazać programistom, jak zapewnić sprawne działanie tych rozwiązań.

Osoby, które przestudiują niniejszą książkę od początku do końca, dowiedzą się, jak pisać, testować i wdrażać oprogramowanie potrzebne do zbudowania własnego prostego, ale kompletnego rozwiązania IoT.

Do kogo kierowana jest ta książka?

Programowanie Internetu rzeczy jest przede wszystkim książką o budowaniu rozwiązań IoT – od urządzenia po chmurę.

Ta książka została pierwotnie napisana jako pomoc naukowa do kursu Connected Devices prowadzonego przez Andiego Kinga na Northeastern University oraz z myślą o studentach, którzy chcą nauczyć się programowania rozwiązań IoT. Chociaż jej podstawowym założeniem było wsparcie studentów i praktyków, może również pomóc osobom, które chcą lepiej poznać koncepcje i zasady IoT.

W tej książce znaleźć można instrukcje, jak krok po kroku zbudować własne całościowe rozwiązanie IoT, a każdy z rozdziałów zawiera stopniowo rozbudowywane ćwiczenia, które pomagają w utrwaleniu wiedzy o IoT. Jednak osoby zainteresowane przede wszystkim ideami również mogą skorzystać z tej książki. Mogą dowiedzieć się z niej *co* i *dlaczego*, szybko przechodząc przez *jak* i całkiem pomijając ćwiczenia.

Autor książki jest edukatorem i konsultantem, dlatego zorganizował ją tak, by mogła ona posłużyć jako program nauczania dla kursu Wprowadzenie do programowania IoT, który ma na celu omówienie kluczowych idei i stopniowe rozwijanie podstawowej wiedzy w tym ważnym obszarze. Książka ma za zadanie pomóc w budowaniu kursu i wiedzy

zarówno nauczycielom akademickim, jak i studentom, którzy chcą rozwinąć swoje umiejętności w zakresie IoT.

Książka pomija większość szczegółowych instrukcji i opisów, polegając na istniejących specyfikacjach i interfejsach API typu open source. Choć pewne fragmenty książki mogą posłużyć jako ogólne wprowadzenie, głównym celem jest pomoc w wykorzystaniu istniejących informacji do budowania własnego rozwiązania. Na szczęście mamy do dyspozycji dobrze napisane specyfikacje protokołów i aktywną społeczność projektów open source. Jesteśmy wdzięczni tym, którzy uczestniczą w tych przedsięwzięciach.

Do programistów

Przyjęto założenie, że praktycy, którzy zaczynają naukę IoT, chcą przede wszystkim zdobyć nowe umiejętności. Być może zauważyli rosnące zapotrzebowanie na specjalistów IoT i chcą uczestniczyć w ewolucji tej ważnej technologii. Wieloletnie doświadczenie nauczyło nas, że najłatwiej jest zrozumieć zawiłości IoT, krok po kroku budując od podstaw zintegrowane, oparte na otwartych standardach rozwiązania IoT, dlatego ten model został zastosowany w niniejszej książce.

Przedstawione przykłady programistyczne autor książki opracował na własnej drodze ku poznawaniu IoT i wiele z nich zostało zapoczątkowanych jako zadania z kursu Connected Devices, który wchodzi w skład programu studiów magisterskich Cyber Physical Systems na Northeastern University.

Każdy rozdział i zestaw ćwiczeń bazuje na poprzednim, dlatego czytelnikom rozpoczynającym pracę z IoT i niezaznajomionym z tym, w jaki sposób poszczególne komponenty oprogramowania budują całościowy system IoT, zaleca się przejście przez ćwiczenia w wyznaczonej kolejności i wykonanie każdego z nich zgodnie z opisem. Ewentualne ulepszenia najlepiej rozważyć dopiero po opanowaniu każdego rozdziału.

Doświadczeni programiści, którzy używają tej książki jak przewodnika, mogą pominąć niektóre podstawowe procedury konfigurowania środowiska deweloperskiego i tylko przejrzeć wybrane ćwiczenia programistyczne, jednak zaleca się przestudiowanie wymagań określonych w każdym z nich, nawet osobom, które zdecydowały się na użycie własnego projektu.

Do instruktorów

Materiały, na których opiera się ta książka, od kilku lat były z powodzeniem wykorzystywane przez jej autora na magisterskim kursie Connected Devices. Niniejsza książka porządkuje te fragmenty wykładów, prezentacji oraz ćwiczeń i została zorganizowana w taki sam sposób, jak kurs.

Kurs miał być początkowo wprowadzeniem do IoT, ale po uwzględnieniu opinii studentów i sugestii asystentów szybko przekształcił się w kurs programistyczny skupiający się na realizacji projektu. Są to aktualnie jedne z ostatnich wymaganych zajęć, którymi studenci kończą swoje studia magisterskie o specjalizacji Cyber Physical Systems. Celem tego kursu jest dostarczenie studentom solidnej podstawowej wiedzy o IoT, która pomoże

im w przejściu ze świata akademickiego do przemysłu i wniesieniu podstawowej wiedzy oraz umiejętności IoT do docelowych organizacji.

Niniejsza książka została zorganizowana w taki sposób, by mogła służyć jako materiał referencyjny lub nawet główny komponent pełnego programu i może pomóc w opracowaniu własnego kursu IoT. Koncentruje się na budowaniu całościowego, zintegrowanego i opartego na otwartych standardach rozwiązania IoT, od urządzenia po chmurę, z wykorzystaniem metody nauczania, którą można nazwać „nauką poprzez budowanie". Ponieważ każdy rozdział bazuje na poprzednim, ta książka może pomóc studentom w rozwijaniu od podstaw własnej platformy.

Ćwiczenia można śledzić na bieżąco w internetowej tablicy Kanban zatytułowanej Programming the IoT (*https://oreil.ly/5b4Gx*)[1]. Dodatkowe materiały pomocne w nauczaniu i wyjaśnianiu niektórych idei przedstawionych w książce znaleźć można na stronie internetowej *https://programmingtheiot.com/programming-the-iot-book*.

Do menedżerów i dyrektorów ds. technologii

Ta książka powinna pomóc w lepszym zrozumieniu wyzwań integracyjnych wiążących się z każdym projektem IoT i pokazać, jakie umiejętności musi posiadać zespół IT, aby móc pomyślnie realizować inicjatywy IoT.

Przyjęliśmy założenie, że głównym celem członków tej grupy jest zrozumienie całego obszaru technologicznego IoT: wyzwań związanych z integracją, wymagań zespołów IT dotyczących środowiska programistycznego, obszarów kompetencji zespołu, obaw wobec IoT wyrażanych przez użytkowników biznesowych i innych zainteresowanych oraz problemów z zarządzaniem zmianami, jakie mogą się pojawić, gdy organizacja rozpoczyna przygodę z IoT.

Dyrektorzy i menedżerowie ds. technologii mogą zrezygnować z implementacji ćwiczeń, ale warto by przeczytali całą książkę, by zrozumieć wyzwania, jakie prawdopodobnie napotka ich zespół.

Użytkownikom biznesowym i innym zainteresowanym, którzy chcą przede wszystkim zrozumieć, z czym wiąże się IoT, zaleca się przeczytanie przynajmniej zestawienia znajdującego się na początku każdego rozdziału, a następnie skoncentrowanie się na końcowym rozdziale, w którym opisane zostały pewne praktyczne przypadki, scenariusze i sugestie dotyczące implementacji.

Co trzeba wiedzieć?

Chociaż przedstawione w tej książce ćwiczenia bazują na założeniu, że czytelnik ma pewne doświadczenie w programowaniu, większość z nich nie wymaga zaawansowanych umiejętności ani wykształcenia informatycznego. Jednak aby zrealizować ćwiczenia zawarte na końcu każdego rozdziału, trzeba posiadać podstawowe umiejętności w zakresie:

1 Wszystkie prezentowane w tej książce ćwiczenia bazują na tablicy Kanban *Programming the IoT* utworzonej przez autora i są wykorzystywane za jego zgodą.

programowania w językach Python oraz Java w stopniu umożliwiającym budowanie prostych aplikacji, korzystania ze zintegrowanych środowisk programistycznych (IDE), czytania, pisania i wykonywania testów jednostkowych oraz konfigurowania systemów opartych na Linuksie za pośrednictwem wiersza poleceń[2].

Wszystkie ćwiczenia są poprzedzone projektem stanu docelowego dla danego zadania, który ilustruje, w jaki sposób nowo budowane komponenty logiczne powinny współdziałać z istniejącymi komponentami, które zostały utworzone wcześniej. Większość z nich to proste diagramy blokowe, które pokazują podstawowe relacje między komponentami aplikacji.

Większość diagramów nie została zaprojektowana zgodnie z żadną konkretną metodą dokumentowania i przedstawia jedynie ogólny widok komponentów i ich podstawowych interakcji. Niektóre diagramy wymagają doprecyzowania, dlatego, aby wyjaśnić zamierzenia projektu, dołączono do nich jeden lub więcej diagramów klas bazujących na języku UML (Unified Modeling Language)[3].

Jak została zorganizowana ta książka?

Ta książka przedstawia proces budowania krok po kroku całościowego, zintegrowanego rozwiązania IoT obejmującego wszystkie warstwy stosu technologicznego i wykorzystującego różne biblioteki open source oraz komponenty programistyczne. Każdy komponent będzie częścią większego systemu i realizując ćwiczenia przedstawione w każdym rozdziale będzie można zobaczyć, w jaki sposób komponenty łączą się i mapują do docelowej architektury.

W każdym rozdziale zawarte zostanie krótkie wprowadzenie do tematu danego rozdziału wraz z pomocnymi materiałami uzupełniającymi, które zawierać będą pewne powiązane definicje. Przedstawione zostanie również wyjaśnienie, dlaczego wybrany temat jest ważny i czego można się będzie nauczyć. Później zaprezentowane zostaną ćwiczenia programistyczne związane z tematem. Wiele rozdziałów zawiera na końcu dodatkowe ćwiczenia, które warto zaimplementować, aby lepiej utrwalić zdobytą wiedzę.

Powiązane rozdziały zostały ze sobą zgrupowane i w ten sposób powstały cztery części książki: Część I: *Rozpoczęcie pracy*, Część II: *Łączenie się ze światem fizycznym*, Część III: *Łączenie się z innymi rzeczami* oraz Część IV: *Łączenie się z chmurą*. Każda część i rozdział zostaną teraz pokrótce omówione, a następnie przedstawione zostaną pewne informacje o samym Internecie rzeczy.

2 Jako że kod źródłowy, a także tablice Kanban i inne zasoby dołączone do książki pozostają w oryginale, konieczna jest również znajomość języka angielskiego w zakresie koniecznym do czytania dokumentacji, identyfikowania poszczególnych plików i kroków procedur (przyp. red. wydania polskiego).

3 Dodatek zawiera podstawową reprezentację każdego ćwiczenia w postaci diagramu UML. Najnowszą specyfikację języka UML można znaleźć na stronie internetowej Object Management Group (OMG) (*https://www.omg.org/spec/UML*).

Każda część i rozdział zaczynają się od miniatury poetyckiej, która ma na celu oddanie istoty wiedzy, jaka jest przekazywana i wyzwań, jakich należy się spodziewać. Skąd taki pomysł? Na początku kariery programisty autor książki pracował w zespole, w którym obowiązywała następująca reguła: osoba odpowiedzialna za kod, który zakłócił przeprowadzany w nocy proces kompilacji, musiała napisać i wysłać do wszystkich członków zespołu haiku nawiązujące do problemu. Choć wykonywanie przedstawionych w niniejszej książce ćwiczeń najprawdopodobniej nie będzie pociągać za sobą tak poważnych konsekwencji, może warto napisać własne haiku, popełniając błędy i ucząc się na nich.

Część I: *Rozpoczęcie pracy*

Ta część stanowi wprowadzenie do budowania rozwiązań IoT. Na początku utworzymy środowisko programistyczne i testowe, a następnie napiszemy dwie proste aplikacje do sprawdzania, czy utworzone środowisko działa prawidłowo.

- Rozdział 1: *Rozpoczęcie pracy* to najdłuższy rozdział w książce. Buduje fundamenty dla całościowego rozwiązania i pomaga w zdobyciu podstawowej wiedzy o IoT. Przeprowadza czytelników po procesie konfigurowania stacji roboczej i środowiska programistycznego, aby jak najszybciej mogli oni zacząć produktywną pracę. W tym rozdziale omówimy pewne podstawowe pojęcia IoT, napiszemy prosty opis problemu, zdefiniujemy najważniejsze idee architektoniczne i ustalimy początkowy projekt, który będzie punktem odniesienia dla każdego kolejnego ćwiczenia.

Internet rzeczy składa się z wielu różnych urządzeń oraz systemów i dostępnych jest wiele narzędzi wspierających programowanie oraz automatyzację systemu. W tej książce wykorzystywane będą narzędzia typu open source. Stanowią one jedynie niewielki podzbiór wszystkich dostępnych rozwiązań i nie należy ich traktować jako uniwersalnych rekomendacji – być może czytelnicy mają inne preferencje. Chcieliśmy po prostu uzyskać dobrze zdefiniowaną treść i opisać ogólne podejście do rozwoju i automatyzacji, które pomoże w implementacji ćwiczeń.

- Rozdział 2: *Początkowe aplikacje warstwy brzegowej* opisuje konfigurowanie środowiska programistycznego i metody zbierania wymagań, a następnie przechodzi do programowania. W tym rozdziale utworzymy pierwsze aplikacje IoT – jedną w języku Python, a drugą w języku Java. Będą one dość proste, ale stanowią przygotowanie dla kolejnych rozdziałów. Wykonanie tych ćwiczeń w opisany sposób zaleca się nawet doświadczonym programistom, abyśmy w dalszej części książki mogli bazować na tych samych podstawach.

Część II: *Łączenie się ze światem fizycznym*

W tej części omówiona zostanie główna właściwość IoT – integracja ze światem fizycznym. Choć ćwiczenia koncentrują się na symulacji i emulacji sprzętu, przedstawione zasady (sposób odczytywania danych czujników i wirtualnego wyzwalania siłownika, czyli elementu wykonawczego) okażą się pomocne, jeśli w przyszłości czytelnicy zdecydują się na użycie prawdziwego sprzętu.

- Rozdział 3: *Symulacja danych* przedstawia różne metody zbierania danych ze świata fizycznego (pomiarów dokonywanych za pomocą czujników) i wyzwalania akcji w oparciu o te dane (wykonywania) przy użyciu symulacji. Na początku zbudowany zostanie zestaw symulatorów, który będzie wykorzystywany we wszystkich kolejnych ćwiczeniach. Te symulatory, choć mają bardzo prosty projekt, ułatwią zrozumienie podstawowych zasad zbierania danych i wykorzystywania tych danych do wyzwalania zdarzeń wykonania.
- Rozdział 4: *Emulacja danych* rozszerza funkcje symulacji opracowane w rozdziale 3, aby emulować działanie czujnika i siłownika. Ten rozdział również koncentruje się na świecie wirtualnym, wykorzystując emulator sprzętu typu open source, który można uruchomić na własnym komputerze.
- Rozdział 5: *Zarządzanie danymi* omawia telemetrię i formatowanie danych, z uwzględnieniem metod strukturalizowania danych w taki sposób, aby mogły być łatwo przechowywane, przesyłane i interpretowane zarówno przez ludzi, jak i maszyny. To zbuduje fundament dla współpracy z innymi „rzeczami".

Część III: *Łączenie się z innymi rzeczami*

To tu następuje konfrontacja z rzeczywistością. Część III koncentruje się na integracji urządzeń. Prawdziwa integracja wymaga opracowania metody przesyłania danych telemetrycznych i innych informacji z jednego miejsca w drugie. W tym rozdziale będzie można poznać i wykorzystać protokoły warstwy aplikacji zaprojektowane z myślą o ekosystemach IoT. Przyjęte zostanie założenie, że warstwa sieciowa jest już dostępna i działa, choć przy okazji zostanie omówionych również kilka bezprzewodowych protokołów.

- Rozdział 6: *Integracja MQTT – Wprowadzenie i klient Python* opisuje protokoły publikowania/subskrypcji, a konkretnie protokół Message Queuing Telemetry Transport (MQTT), który jest powszechnie wykorzystywany w aplikacjach IoT. Rozdział zawiera omówienie wybranych szczegółów specyfikacji i wyjaśnia, jak można zacząć budowanie prostej warstwy abstrakcji, która ułatwia korzystanie z wspólnych bibliotek open source, zaczynając od języka Python.
- Rozdział 7: *Integracja MQTT – klient Java* kontynuuje rozszerzanie wiedzy o MQTT, opisując bibliotekę open source, która umożliwia aplikacjom Java łączenie się z serwerem MQTT. W końcowej części rozdziału wykorzystamy protokół MQTT w ćwiczeniach i testach służących do zintegrowania się z kodem Python opracowanym w rozdziale 6.

- Rozdział 8: *Implementacja serwera CoAP* koncentruje się na protokołach żądanie/odpowiedź, w szczególności protokole Constrained Application Protocol (CoAP), który także jest powszechnie wykorzystywany w aplikacjach IoT. Jednak tym razem na początku użyjemy języka Java i zbudujemy serwer CoAP wykorzystujący inną bibliotekę open source. Rozdział zawiera również opcjonalne ćwiczenia, w których serwer CoAP jest budowany w języku Python.
- Rozdział 9: *Integracja klienta CoAP* kontynuuje pracę z protokołem CoAP, ale koncentruje się na pisaniu kodu klienta, który posłuży do łączenia się z nowo zbudowanym serwerem Java. Ten kod klienta, napisany zarówno w języku Python, jak i Java, pozwoli na wspieranie komunikacji między urządzeniami z wykorzystaniem protokołu CoAP.
- Rozdział 10: *Integracja warstwy brzegowej* koncentruje się na integracji, umożliwiając nam połączenie dwóch utworzonych wcześniej aplikacji ze sobą za pomocą protokołu MQTT lub CoAP. Dodane zostały ćwiczenia dla każdego z tych protokołów, aby ułatwić wybór tego lepiej dostosowanego do wymogów sytuacji. Podobnie jak rozdział 9, ten rozdział będzie wymagał zaimplementowania rozwiązań zarówno w języku Python, jak i Java.

Część IV: *Łączenie się z chmurą*

Na zakończenie, na samej „górze” stosu integracji będzie można się dowiedzieć, jak połączyć całą infrastrukturę urządzeń IoT z chmurą, wykorzystując aplikację bramy jako pośrednika między funkcjami chmury a wszystkimi urządzeniami.

W tej części przedstawione zostaną podstawowe zasady łączności z chmurą i omówione różne usługi w chmurze, które pozwalają przechowywać i analizować dane oraz zarządzać środowiskiem IoT. Zbudowana zostanie ta sama prosta aplikacja w chmurze dla każdej z platform.

- Rozdział 11: *Integracja z różnymi usługami w chmurze* wyjaśnia kluczowe koncepcje związane z łączeniem rozwiązania IoT z chmurą i opisuje różne ćwiczenia integracji z chmurą, jakie można zaimplementować przy użyciu protokołu MQTT. Ponieważ platformom chmury poświęcono już wiele książek i samouczków, przedstawiony zostanie jedynie przegląd ich możliwości, bez szczegółowego omawiania jakichkolwiek rozwiązań. Ćwiczenia pomogą czytelnikom w zadecydowaniu, której platformy chmury chcą użyć we własnej implementacji.
- Rozdział 12: *Oswajanie IoT* opisuje kluczowe czynniki wspierające rozwiązania IoT i mapuje je do kilku prostych przypadków użycia IoT, które były szczególnie pomocne w przygotowywaniu kursu Connected Devices. Dla każdego z nich przedstawiony zostanie ogólny opis problemu, oczekiwany wynik i wysokopoziomowy projekt koncepcyjny.

Mamy nadzieję, że ta książka pomoże czytelnikom zrozumieć i zbudować zintegrowany, całościowy system IoT.

Wprowadzenie do IoT

Oto krótkie podsumowanie historii IoT.

Ogromnym postępem w dziedzinie informatyki było wynalezienie tranzystora w latach 1950., a następnie opublikowanie w latach 1960. artykułu Gordona Moore'a opisującego podwajanie się liczby tranzystorów mieszczących się na tej samej powierzchni fizycznej (uaktualnionego w latach 1970.)[4].

Wraz z nowoczesną informatyką nadeszły nowoczesne sieci i zalążki Internetu zapoczątkowane przez wynalezienie ARPAnet w 1969[5]. To doprowadziło do opracowania w latach 1970. nowych metod *porcjowania* lub pakietowania danych przy użyciu protokołów NCP (Network Control Protocol) oraz TCP (Transmission Control Protocol) przez protokół IP (Internet Protocol) i wykorzystanie istniejącej infrastruktury sieci przewodowych. Znalazły one zastosowanie w przemyśle, umożliwiając rozwój automatyki przemysłowej w kierunku scentralizowanego zarządzania rozproszonymi, połączonymi systemami. Systemy SCADA (Supervisory Control and Data Acquisition) – przodkowie technologii M2M (machine-to-machine), które doprowadziły do powstania IoT – wyłoniły się z różnych niestandardowych rozwiązań, a programowalne sterowniki logiczne (PLC) – wynalezione tuż przed ARPAnet – ewoluowały tak, aby wykorzystywać TCP/IP i powiązane standardy sprzętowe[6].

W latach 1980. pojawił się protokół UDP (User Datagram Protocol[7]) i narodziło się to, co przez wielu jest postrzegane jako początek nowoczesnego Internetu (World Wide Web) wynalezionego w końcówce lat 1980. przez Tima Bernersa-Lee[8].

Ten okres był również ważny dla rozwoju tego, co obecnie nazywane jest Przemysłowym Internetem rzeczy (Industrial Internet of Things, IIoT), czyli podzbiorem IoT i ważnym składnikiem jego ewolucji.

Czytelnicy na pewno zauważyli pewien powtarzający się schemat: problem prowadzi do powstania rozwiązującej go innowacji technologicznej (często niestandardowej), która następnie zostaje ustandaryzowana lub zostaje wyparta przez jeden lub więcej standardów, co prowadzi do rozpowszechniania się i dalszych innowacji.

---

4 *Encyclopaedia Britannica Online*, pod hasłem „Moore's law" (*https://oreil.ly/6sqQr*), napisanym przez edytorów *Encyclopaedia Britannica*, ostatnia aktualizacja 26 grudnia 2019.

5 *Encyclopaedia Britannica Online*, pod hasłem „ARPANET" (*https://oreil.ly/tHVZq*), napisanym przez Kevina Featherly'a, ostatnia aktualizacja 23 marca 2021.

6 Simon Duque Antón i inni, artykuł: „Two Decades of SCADA Exploitation: A Brief History" (*https://doi.org/10.1109/AINS.2017.8270432*), opublikowany w *2017 IEEE Conference on Application, Information and Network Security (AINS)* (Nowy Jork: IEEE, 2017), 98–104.

7 John Postel, artykuł: „User Datagram Protocol" (*https://tools.ietf.org/html/rfc768*), Internet Standard RFC 768, 28 sierpnia 1980.

8 Dodatkowe informacje o propozycji sieci WWW Tima Bernersa-Lee znaleźć można na stronie *https://www.w3.org/History/1989/proposal.html*.

I tak oto nadeszła era IoT. W latach 1980. i na początku lat 1990. pojawiły się pierwsze podłączone urządzenia, w tym pierwszy połączony z Internetem toster zaprezentowany przez Johna Romkey i Simona Hacketta na konferencji Interop 1990[9]. W roku 1991 użytkownicy zajęć laboratoryjnych w pobliżu pokoju Trojan Room na uniwersytecie University of Cambridge podłączyli kamerkę webową tak, by móc monitorować ekspres do kawy – bo komu chce się pokonywać cały ten dystans tylko po to, by odkryć, że dzbanek jest pusty[10]?

To oczywiście utorowało drogę dla innych urządzeń – wiele z nich zostało pewnie zbudowanych i podłączonych w ramach eksperymentów w uczelnianych laboratoriach, akademikach, domach, mieszkaniach i firmach. W tym samym czasie komputery i sieci stawały się coraz tańsze, mocniejsze i oczywiście mniejsze. Powszechnie uważa się, że termin *Internet of Things* został użyty po raz pierwszy przez Kevina Ashtona w 1999 roku, gdy prowadził on poświęconą tej tematyce prezentację w firmie Proctor & Gamble[11].

Przechodząc prosto do roku 2005, Interaction Design Institute Ivrea we Włoszech wydał niedrogi, zaprojektowany z myślą o laikach komputer jednopłytkowy (*single-board computer*, SBC) Arduino [12], który pozwolił większej liczbie ludzi tworzyć własne systemy detekcji i automatyzacji. Wystarczy łatwo dostępna pamięć i funkcja analizowania danych za pomocą usług dostępnych z dowolnego miejsca w Internecie i powstają podstawy ekosystemu IoT: czyli wielu osobnych, unikalnych rzeczy, które można połączyć ze sobą w celu osiągania lepszych rezultatów.

Jednak postrzeganie IoT jako zbioru rzeczy, które łączą świat fizyczny z Internetem, nie oddaje całej prawdy. Kwintesencją IoT i głównym motorem napędzającym jego złożoność jest *heterogeniczność*: odmienność i ogromne zróżnicowanie typów, funkcji i możliwości urządzeń, ich zastosowań, sposobów implementacji, wspieranych protokołów, zabezpieczeń i technik zarządzania.

Ponowne definiowanie koncepcji

A zatem czym dokładnie jest Internet rzeczy? To złożony zbiór ekosystemów technologii, które łączą świat fizyczny z Internetem z wykorzystaniem różnych brzegowych urządzeń komputerowych i usług obliczeniowych w chmurze.

W tej książce zastosowane zostaną nieco uproszczone definicje. Do *brzegowych urządzeń komputerowych* możemy zaliczyć wbudowane urządzenia elektroniczne, systemy komputerowe i aplikacje, które albo podejmują bezpośrednią interakcję ze światem fizycznym za pomocą czujników i siłowników, albo zapewniają *bramę* bądź mostek

9 Wyjaśnienie znaleźć można w artykule: „The Internet Toaster" (*https://oreil.ly/VxRcP*).

10 *Encyclopaedia Britannica Online*, artykuł: „Know Your Joe: 5 Things You Didn't Know About Coffee" (*https://oreil.ly/j7pVk*) (2. The Watched Pot), napisany przez Alison Eldridge, dostępny 18 stycznia 2021.

11 Kevin Ashton, artykuł „That 'Internet of Things' Thing" (*https://oreil.ly/BqzmG*), *RFID Journal*, 22 czerwca 2009.

12 Krótka historia narodzin Arduino została przedstawiona w artykule „Timeline of Computer History" na stronie Computer History Museum (*https://oreil.ly/Qv6AW*).

umożliwiający tym urządzeniom i aplikacjom łączenie się z Internetem. *Usługi obliczeniowe w chmurze* to systemy komputerowe, aplikacje, magazyny danych i inne usługi obliczeniowe, które funkcjonują w jednym lub wielu centrach danych i są zawsze dostępne za pośrednictwem Internetu.

W dalszej części książki będziemy odwoływać się do tych dwóch obszarów za pomocą odpowiadających im *warstw architektonicznych*: *warstwy chmury* dla usług obliczeniowych w chmurze i *warstwy brzegowej* dla brzegowych urządzeń komputerowych. Warstwy architektoniczne rozdzielają kluczowe funkcje IoT z perspektywy fizycznej i logicznej, co oznacza na przykład, że pomiary i wykonania zachodzą w warstwie brzegowej, natomiast wszystkie operacje przechowywania długoterminowego i złożone analizy zachodzą w warstwie chmury.

Czytelnicy, którzy chcą dowiedzieć się więcej o architekturze i powiązanych standardach, mogą zainteresować się organizacjami biorącymi aktywny udział w pracach i publikującymi informacje na ten temat. Oto kilka z nich:

- European Telecommunications Standards Institute (ETSI) (*https://www.etsi.org*), Europejska organizacja standaryzacyjna
- The International Telecommunication Union (ITU) (*https://www.itu.int/en/Pages/default.aspx*), agencja Organizacji Narodów Zjednoczonych ds. technologii informacyjnych i komunikacyjnych
- The Internet Engineering Task Force (IETF) (*https://www.ietf.org*), międzynarodowa organizacja zajmująca się standardami w Internecie
- The Industrial Internet Consortium (IIC) (*https://www.iiconsortium.org*), zrzesza firmy i organizacje zainteresowane IIoT

IIC opublikowała różne przydatne dokumenty. Szczególnie interesujący jest dokument *Industrial Internet Reference Architecture*, który omawia framework i wspólną terminologię dla systemów IIoT i ma ogromny wpływ na to, w jaki sposób autor książki podchodzi do tematu architektury IoT[13].

Wnoszenie wartości

Prawdziwą wartością każdego systemu IoT jest zdolność oferowania lepszego lub wzbogaconego rezultatu poprzez integrację świata fizycznego i logicznego oraz zbieranie i analizowanie danych szeregów czasowych.

Za prosty przykład może posłużyć zwykła kuchnia wyposażona w wiele urządzeń. Jeśli można na przykład mierzyć co minutę temperaturę wewnątrz lodówki, można ustalić, jak długo rzeczy przechowywane w lodówce były wystawione na określoną temperaturę. Gdyby temperatura wewnątrz lodówki była sprawdzana tylko raz dziennie, nie istniałyby wystarczająco szczegółowe dane, by można to było ustalić.

13 *The Industrial Internet of Things, Volume G1: Reference Architecture* (*https://www.iiconsortium.org/IIRA.htm*), Version 1.9 (Needham, MA: Industrial Internet Consortium, 2019).

Oczywiście jeśli inne systemy nie rozumieją zbieranych danych, są one mało przydatne. Choć każda poszczególna część może być unikatowa i niezależna od innych części, budowanie integrowalnych rozwiązań IoT wymaga od programisty uważnego przemyślenia, w jaki sposób projekt i dane poszczególnych części systemu IoT mogą wpływać na inne systemy (i innych programistów).

Innym kluczowym elementem w łańcuchu wartości jest *skalowalność*. Co innego jest zbudować system, który wspiera garstkę sygnałów wejściowych, a co innego obsługiwać tysiące, miliony, a nawet miliardy takich sygnałów. To właśnie w skalowalności, czyli zdolności systemu do obsługiwania zarówno dużej, jak i małej ilości danych w zależności od potrzeb, leży prawdziwa moc Internetu rzeczy. Na przykład, skalowalny system w chmurze wspierający IoT to taki, który może z powodzeniem obsługiwać jedno urządzenie bramy nadsyłające do niego dane, jak również tysiące (lub miliony bądź miliardy) takich źródeł danych.

Jest już pewnie jasne, że budowanie zintegrowanego systemu IoT wymaga uwzględnienia na każdym etapie istotnych niuansów. Nie można oczekiwać, że systemy przysyłające dane będą mogły być automatycznie podłączane lub nawet że będą zachowywały się w przewidywalny sposób. Co więcej, nawet jeśli kod zostanie napisany w tak ogólny sposób, aby mógł działać tak samo na różnych urządzeniach sprzętowych, nie zawsze będzie się sprawdzał na każdej platformie.

W jednej książce nie można omówić wszystkich specjalistycznych platform. Nie jest również łatwo napisać spójny, semi-niskopoziomowy kod na poziomie urządzenia, którego nie trzeba optymalizować dla każdego urządzenia. Choć każde urządzenie ma pewne cechy szczególne, które trzeba uwzględnić tworząc (i testując) rozwiązania, przykłady kodu przedstawione w tej książce są przenośne (z pewnymi małymi wyjątkami) i mogą być wykorzystywane w większości systemów, w których można uruchomić maszynę wirtualną Java lub interpretera Python 3.

Na brzegu

Ze względu na ogromną moc i elastyczność warstwy chmury, rozwiązanie IoT zazwyczaj wykazuje największą złożoność na brzegu, gdzie skupia się większość (a często cała) heterogeniczność systemu. Ta książka koncentruje się na dwóch kategoriach urządzeń: urządzeniach ograniczonych i urządzeniach bramy.

W ogromnym uproszczeniu: *urządzenia ograniczone* mają jakieś ograniczenia związane z mocą, komunikacją lub przetwarzaniem, natomiast *urządzenia bramy* w zasadzie ich nie mają.

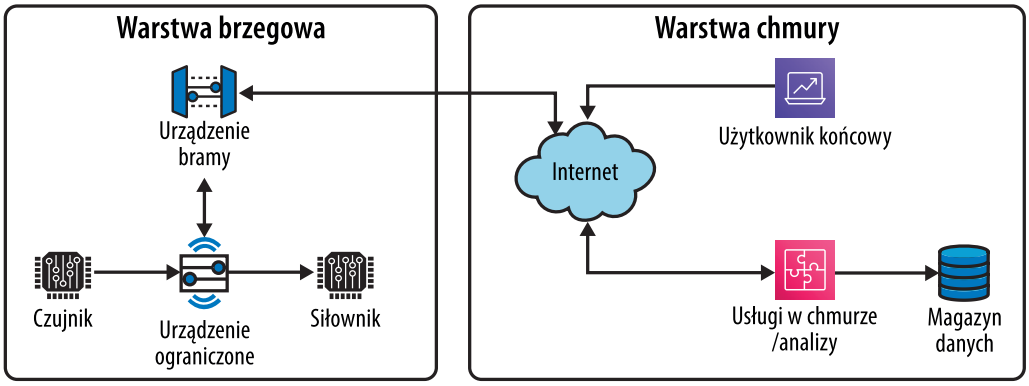
Celowo nie używamy nazwy „urządzenia inteligentne (smart)”, ponieważ coraz trudniej jest zdefiniować „urządzenia inteligentne” tak, aby odróżnić je od tych „mniej inteligentnych”.

Urządzenie ograniczone to może być jednopłytkowy komputer o ograniczonej mocy (czasem zasilany baterią), który albo odczytuje dane ze środowiska (takie jak temperatura, ciśnienie czy wilgotność), albo inicjuje jakąś mechaniczną akcję (takie jak otwarcie lub zamknięcie zaworu).

IETF oferuje szczegółowe definicje i terminologię dla różnych urządzeń ograniczonych i węzłów" w RFC 7228[14]. Intencją tej książki *nie* jest zmienianie tych definicji w jakikolwiek sposób – terminy *urządzenie ograniczone* i *aplikacja urządzenia ograniczonego* służą po prostu do oddzielenia przewidywanej funkcjonalności urządzenia ograniczonego od urządzenia bramy, wskazując jednocześnie naturę tych dwóch typów urządzeń (w skrócie, to pierwsze ma więcej ograniczeń technicznych niż to drugie).

Urządzenie bramy można również zaimplementować jako komputer jednopłytkowy, ale ma ono dużo większy potencjał: może komunikować się z wieloma różnymi urządzeniami ograniczonymi i ma dość mocy obliczeniowej, aby agregować dane z różnych urządzeń, wykonywać pewne funkcje analityczne oraz określać, kiedy (i jak) przesłać odpowiednie dane do chmury w celu ich dalszego przechowywania i przetwarzania.

Rysunek P-1 przedstawia koncepcyjny projekt architektury systemu IoT, który reprezentuje relacje między tymi typami urządzeń w warstwie brzegowej a usługami i innymi funkcjami znajdującymi się w warstwie chmury.



Rysunek P-1 *Koncepcyjny projekt architektury systemu IoT*

Na potrzeby niniejszej książki te urządzenia zostaną skategoryzowane w następujący sposób:

- Urządzenia bramy koncentrują się na przetwarzaniu, interpretowaniu i integrowaniu danych brzegowych oraz interakcji z chmurą. Przeprowadzają analizy „na brzegu", konwertują i transformują stosy protokołów, ustalają sposób przekierowywania wiadomości (i czy to konieczne) oraz oczywiście bezpośrednio łączą się z Internetem oraz różnymi usługami w chmurze, dzięki którym system IoT przynosi korzyści biznesowe.

14 Carsten Bormann, Mehmet Ersue i Ari Keränen, „Terminology for Constrained-Node Networks" (*https://tools.ietf.org/html/rfc7228*), IETF Informational RFC 7228, Maj 2014, 8–10.

- Urządzenia ograniczone odpowiadają jedynie za pomiary i/lub wykonanie oraz kontrolowanie, samodzielnie przetwarzając wiadomości lub przekazując je, jeśli zaimplementowany został odpowiedni protokół i wysyłając wiadomości do urządzenia bramy. W skrócie, mają one ograniczone możliwości i czasem nie mogą bezpośrednio łączyć się z Internetem, bazując na łączności z chmurą zapewnianej przez urządzenie bramy.

Czy urządzenie ograniczone może bezpośrednio łączyć się z Internetem? Oczywiście, jeśli wspiera stos protokołów TCP/IP, ma rutowalny adres IP dostępny dla i z publicznego Internetu i zawiera odpowiedni sprzęt do łączenia się z Internetem.

Jednak w tej książce zawęziliśmy kategorię urządzeń ograniczonych, narzucając następujące dwa ograniczenia:

- Nie wspierają one przesyłania pakietów bezpośrednio do i z publicznego Internetu (przy jednoczesnym założeniu, że wspierają TCP/IP i UDP/IP) i muszą podejmować interakcję z urządzeniem bramy, aby być częścią jakiegokolwiek ekosystemu IoT.
- Nie zawierają odpowiednich zasobów obliczeniowych, aby inteligentnie określać tok postępowania w oparciu o zebrane dane.

Ta książka koncentruje się na paradygmacie „urządzenie ograniczone do urządzenia bramy do połączenia z chmurą", chociaż w różnych przypadkach użycia warto rozważyć inne potencjalnie lepsze modele warstwy brzegowej.

Wnioski

To wszystko oznacza, że rozwijanie urządzeń komputerowych, które są mniejsze, szybsze i tańsze, wykorzystywanie ich do interakcji ze światem fizycznym i łączenie ich (lub ich danych) z Internetem w celu przetwarzania przy użyciu usług w chmurze umożliwia uzyskiwanie informacji, które pomagają w osiąganiu lepszych wyników biznesowych.

Dziękujemy za lekturę!

Konwencje użyte w tej książce

Oto konwencje typograficzne przyjęte w tej książce:

Kursywa
: Wyróżnia nowe terminy, adresy URL i adresy email, nazwy i rozszerzenia plików.

`Stała szerokość`
: Służy do prezentowania fragmentów kodu programu, a także wewnątrz akapitów do odwoływania się do elementów programu, takich jak nazwy zmiennych i funkcji, bazy danych, typy danych, zmienne środowiskowe, instrukcje czy słowa kluczowe.

`Stała szerokość z pogrubieniem`
: Przedstawia polecenia i inne rodzaje tekstu, który ma zostać wpisany przez użytkownika w identyczny sposób.

`Stała szerokość z kursywą`
: Przedstawia tekst, który ma zostać zastąpiony wartościami podanymi przez użytkownika lub wynikającymi z kontekstu, a także komentarze w przykładach kodu.

Ten element oznacza wskazówkę lub sugestię.

Ten element oznacza ogólną uwagę.

Ten element oznacza ostrzeżenie lub przestrogę.

Wykorzystywanie przykładów kodu

Materiały pomocnicze (przykładowy kod i szablony dokumentacji) są dostępne na stronie *https://github.com/programming-the-iot*. Przedstawione w książce przykłady kodu są objęte licencją MIT (*https://www.mit.edu/~amini/LICENSE.md*).

Pytania techniczne bądź problemy z wykorzystaniem przykładów kodu można zgłaszać drogą mailową: *bookquestions@oreilly.com*.

Ta książka ma pomóc programistom w osiąganiu własnych celów. Zasadniczo jeśli książka zawiera przykładowy kod, można użyć go we własnych programach oraz dokumentacji. Nie ma potrzeby kontaktowania się w celu uzyskania zezwolenia, o ile nie

planuje się reprodukcji znaczącej części kodu. Na przykład, napisanie programu wykorzystującego kilka fragmentów kodu z książki nie wymaga zezwolenia. Natomiast sprzedaż lub dystrybucja przykładów z książek wydawnictwa O'Reilly wymaga zezwolenia. Udzielenie odpowiedzi poprzez zacytowanie tej książki i przykładowego kodu nie wymaga zezwolenia. Umieszczenie znaczącej części przykładów kodu z tej książki w dokumentacji własnego produktu wymaga zezwolenia.

Będziemy wdzięczni za wskazanie źródła, choć nie jest to wymagane. Odwołanie do źródła zwykle zawiera nazwisko autora, tytuł, wydawnictwo oraz numer ISBN. Na przykład: Andy King, „*Programowanie Internetu rzeczy* (wydawnictwo O'Reilly). Prawa autorskie Andrew D. King 2021, 978-1-492-08141-8."

W przypadku wątpliwości czy planowane zastosowanie przykładowego kodu wykracza poza przedstawione powyżej zezwolenia, prosimy o kontakt za pośrednictwem adresu e-mail: *permissions@oreilly.com*.

Jak się z nami skontaktować

Komentarze i pytania dotyczące tej książki prosimy kierować do wydawnictwa:

O'Reilly Media, Inc.
1005 Gravenstein Highway North Sebastopol, CA 95472
800-998-9938 (w USA lub Kanadzie) 707-829-0515 (połączenia międzynarodowe lub lokalne)
707-829-0104 (fax)

Przygotowaliśmy stronę internetową dla tej książki, na której umieszczamy erratę, przykłady i dodatkowe informacje. Jest ona dostępna pod adresem: *https://oreil.ly/programming-the-IoT*.

Aby przekazać swój komentarz lub zadać pytanie techniczne dotyczące tej książki, można wysłać e-mail na adres: *bookquestions@oreilly.com*.

Aktualne informacje dotyczące naszych książek i kursów umieszczamy na stronie: *http://oreilly.com*.

Można nas znaleźć na Facebooku: *http://facebook.com/oreilly*
śledzić na Twitterze: *http://twitter.com/oreillymedia*
i oglądać na kanale YouTube: *http://youtube.com/oreillymedia*

Podziękowania

Myślę, że wszyscy stale szukamy samych siebie, czy to poprzez karierę, relacje z innymi czy pracę misjonarską. Różne zawirowania, przez które Bóg poprowadził mnie na drodze życia oraz mentorzy i towarzysze, których umieścił na tej drodze, reprezentują okno, przez które doświadczam Jego miłości do mnie. Oczywiste jest, że niniejsza książka nie powstałaby bez cierpliwości, krytyki, zachęty i mądrości mojej rodziny, przyjaciół i kolegów, którzy byli gotowi przebyć tę drogę ze mną. Spróbuję podziękować wszystkim, którzy mieli wpływ na mnie i na tę książkę, ale obawiam się, że mogę niechcący pominąć kilka osób. Proszę wybaczcie mi przeoczenia.

Zacząłem programowanie w wieku około 12 lat, gdy mój tata przyniósł do domu Atari 400, które pożyczył mu kolega. Gdy zainteresowałem się językiem BASIC (tracąc efekty pracy po każdym wyłączeniu zasilania), moi rodzice kupili mi własny komputer Atari 800XL, w którym nośnikiem danych były kasety (kto to jeszcze pamięta?), które później zastąpiły dyskietki 5¼" , a końcową inwestycją był piekielnie szybki modem 1200 Kbps. Dziękuję Wam Mamo i Tato. Tato… bardzo chciałbym byś nadal był z nami.

I tak oto 25 lat później (w końcu) zostałem programistą i informatykiem. Ale jak to się zaczęło? Mój przyjaciel Jimmy Song, autor książki *Programming Bitcoin* (O'Reilly), skierował mnie do zespołu O'Reilly Media, gdy pewnego dnia powiedziałem, że „Myślę o napisaniu książki…" Jestem naprawdę wdzięczny Jimmiemu za szczerą poradę, opinie i zachęcanie mnie do tej podróży.

Oczywiście źródłem praktycznej zawartości tej książki był kurs Connected Devices, jaki prowadziłem na Northeastern University. I mam ogromne szczęście, że miałem okazję współpracować ze wspaniałymi akademikami, takimi jak dr Peter O'Reilly, dyrektor programu Cyber Physical Systems na wydziale College of Engineering uniwersytetu Northeastern, który dał mi okazję wykładać Connected Devices. dr Rolando Herrero, również z uniwersytetu Northeastern, był jednym z pierwszych recenzentów zawartości tej książki. Jego ekspercka znajomość systemów i protokołów IoT niezmiernie pomogła w określaniu formatu i funkcji prezentowanego materiału. Kooperacja z Peterem i Rolando, zarówno w roli przyjaciół, jak i współpracowników w rozwijaniu dziedziny IoT, jest jednym z najciekawszych aspektów mojej kariery i jestem im niezmiernie wdzięczy za mentoring.

Ogromne podziękowania dla wszystkich studentów Connected Devices, których miałem zaszczyt uczyć i z którymi miałem okazję współpracować. Treść tej książki i prezentowane w niej ćwiczenia zostały zainspirowane przez Wasze opinie, sugestie, prośby, komentarze i wyrozumiałość dla moich odgrzewanych, niemądrych żartów. To głównie dzięki Wam powstała ta książka i mogę realizować swoją pasję, ucząc (zamiast robić karierę jako kabareciarz).

Wielu asystentów prowadzonego przeze mnie kursu recenzowało i komentowało tę książkę i pomogło mi zbudować pomost między tekstem a ćwiczeniami online. Serdeczne podziękowania dla tych wspaniałych kolegów: Chenga Shu, Ganeshrama Kanakasabai, Harsha Vardhanrama Kalyanaramana, Jaydeepa Shaha, Yogesha Suresha

oraz Yuxianga Cao. Cieszę się, że miałem okazję pracować z każdym z nich i jestem ogromnie wdzięczny za ich wkład w sukces kursu.

Niektórzy przyjaciele i koledzy z branży dostarczyli szczegółowych opinii i komentarzy dotyczących wersji roboczej książki. Tim Strunck, ekspert w budowaniu komercyjnych rozwiązań IoT, przekazał wiele informacji zwrotnych dotyczących ogólnej organizacji książki i jej zawartości technicznej – jestem wdzięczny za jego przyjaźń i cenne uwagi oraz za wspaniałą Przedmowę do tej książki. Steve Resnick, dyrektor zarządzający w dużej firmie konsultingowej i autor dwóch książek, podzielił się ze mną swoją menedżerską perspektywą i udzielił rad dotyczących organizacji, kształtu i zawartości, co pomogło w uwypukleniu pewnych kluczowych założeń. Ben Pu popatrzył na książkę z nowej perspektywy, co pomogło mi lepiej zrozumieć, w jaki sposób mogą postrzegać ją przyszli czytelnicy. Doceniam to, że mam kontakt z takimi profesjonalistami, jak Tim, Steve oraz Ben i jestem bardzo wdzięczny za szybkie przekazywanie komentarzy i opinii.

Dziękuję wszystkim kolegom i przyjaciołom w O'Reilly Media – co za zespół! Mike Loukides, VP i Intake Editor, który poświęcił godziny na rozmowy video i telefoniczne oraz korespondencję e-mail, pomógł mi sformułować propozycję wydawniczą i przeprowadził ją przez proces przyjęcia. Melissa Duffield, VP of Content Strategy, która pracowała ze mną nad potencjałem komercyjnym książki i w ekspresowym tempie przepchnęła ją przez proces akceptacji. Cassandra Furtado, która cierpliwie odpowiedziała na wszystkie moje początkowe pytania. Chris Faucher, który przekształcił mój manuskrypt w książkę. Virginia Wilson, która pomogła w zarządzaniu harmonogramem i procesem recenzji technicznej. Arthur Johnson, który jako redaktor wyłapał i poprawił wiele błędów w tekście i problemów ze spójnością formatowania. I oczywiście Sarah Grey, która jako Development Editor pomogła mi przemyśleć organizację materiału i urzeczywistnić go. Bardzo ich szanuję i dziękuję za profesjonalizm, wsparcie, szybką reakcję i koleżeństwo w czasie realizacji tego projektu.

Do przygotowywania tej książki, jej przykładów i ćwiczeń wykorzystano wiele oprogramowania open source i ogólnie dostępnych specyfikacji. Chciałbym podziękować programistom i kontrybutorom każdego z tych projektów.

A przede wszystkim dziękuję mojej wspaniałej żonie Yoon-Hi i niesamowitemu synowi Eliotowi: jesteście moją ostoją i nie mam pojęcia, jak mógłbym w ogóle zacząć ten projekt bez Waszej zachęty i wsparcia. Dziękuję za wszystkie opinie, dyskusje nad moim tekstem (i edytowanie go), wyzwania, wsparcie i wiarę we mnie! Wyciągacie mnie z dolin i dajecie siłę, by zdobywać szczyty. Jesteście cudowni i kocham Was bardziej, niż potrafię wyrazić słowami.

CZĘŚĆ I

Rozpoczęcie pracy

Wprowadzenie

Trzeba ruszyć do przodu.
Ale od czego zacząć?
Zacznijmy od początku, czyli od przygotowania.

W przedmowie wprowadzone zostały pewne podstawowe zasady i idee IoT, ale do końca nie wyjaśniono, czym jest IoT, jak działa i do czego się nadaje. W tej części również nie będzie można dowiedzieć się wszystkiego o Internecie rzeczy jako ekosystemie mocno zróżnicowanych urządzeń i funkcji. Skoncentrujemy się na podstawowym zbiorze kluczowych idei, aby pomóc w uzyskaniu ogólnego obrazu, w jaki sposób IoT może pomóc w rozwiązywaniu różnych problemów.

Proces przeprowadzony w tej części i wykorzystywany w pozostałych częściach książki jest taki sam, jaki autor książki stosuje, by pomóc swoim studentom lepiej zrozumieć IoT i (co być może ważniejsze) zbudować proste, całościowe rozwiązanie IoT, od urządzenia po chmurę.

Czego można się dowiedzieć z tej części

Rozdziały 1 i 2 koncentrują się na trzech głównych zagadnieniach: (1) definiowanie problemu i kategoryzowanie funkcji (2) konfiguracja środowiska programistycznego oraz (3) budowanie początkowych aplikacji warstwy brzegowej. Omówimy je przy okazji definiowania prostego problemu IoT, zarysowywania podstawowej architektury służącej do rozwiązania tego problemu i przedstawiania filozofii projektowej, która będzie dyktować sugerowaną ścieżkę implementacji. Ta ścieżka zaczyna się od skonstruowania dwóch prostych aplikacji: aplikacji urządzenia bramy oraz aplikacji urządzenia ograniczonego. Pora zabrać się do pracy.

Rozdział 1

Rozpoczęcie pracy

Podstawy IoT i przygotowanie środowiska programistycznego

Droga przed nami,
Krzaki i kolce, ale później prosto.
Już blisko. Cierpliwości.

Podstawowe założenia: Identyfikowanie problemu i definiowanie architektury jako fundamentu dla rozwiązania IoT. Przygotowanie środowiska programistycznego zoptymalizowanego pod kątem IoT i wspierającego różne możliwości wdrożenia.

Jak można się już było zorientować, Internet rzeczy może być ogromny, skomplikowany i trudny do opanowania. Planując dalszą drogę, warto zacząć od zidentyfikowania problemu do rozwiązania, a następnie utworzyć architekturę, na podstawie której projektowane i budowane będzie rozwiązanie.

Na początek kilka kluczowych pytań w celu ustalenia punktu odniesienia. Jaki problem ma zostać rozwiązany? Gdzie zaczyna się on i kończy? Dlaczego potrzebujemy ekosystemu IoT? W jaki sposób wszystkie komponenty będą współpracować ze sobą w celu rozwiązania problemu? Jakiego efektu możemy się spodziewać, gdy wszystko będzie działać zgodnie z planem? Wnikliwie przeanalizujemy wszystkie te pytania, budując po drodze całościowe, zintegrowane rozwiązanie IoT, które spełnia nasze wymagania.

Czego będzie się można nauczyć w tym rozdziale

Aby pomóc w zrozumieniu, w jaki sposób można i należy budować system IoT, przedstawimy pewne podstawowe koncepcje architektoniczne bazujące na wcześniejszych pytaniach i wykorzystamy je jako podstawę implementacji. Od tego momentu zaczniemy

budować rozwiązanie, które warstwa po warstwie rozwiązuje opisany problem, wzbogacając je o dodatkowe funkcje w kolejnych rozdziałach.

Nie ulega wątpliwości, że właściwe narzędzia programistyczne pozwalają oszczędzić czas i nerwy, nie wspominając o ułatwionym testowaniu, sprawdzaniu poprawności i wdrażaniu. Dostępnych jest wiele doskonałych narzędzi programistycznych, zarówno open source, jak i komercyjnych, które pomagają w pracy.

Doświadczeni programiści mają pewnie własne preferencje w zakresie środowiska programistycznego, które najlepiej pasuje do ich stylu i metod programowania. Autor książki też ma zestaw ulubionych narzędzi, na którym opierać się będą przedstawiane przykłady. Jednak ten rozdział nie służy do narzucania wyboru narzędzi, a jedynie wsparcia programisty w wyposażeniu środowiska programistycznego IoT w sposób, który umożliwia szybkie zamknięcie tego etapu, aby w przyszłych projektach mógł on korzystać z własnych narzędzi.

Najważniejsze będą prezentowane koncepcje – języki programowania, narzędzia (i ich wersje) oraz metody można zmienić. Te koncepcje reprezentują pewne podstawowe zasady spójnego programowania: projekt systemu, kodowanie i testowanie.

Definiowanie systemu

Opracowanie opisu problemu to prawdopodobnie najważniejsza część tej łamigłówki. Zacznijmy od zarysowania czegoś prostego, a jednocześnie wystarczającego do zilustrowania szeregu interesujących wyzwań IoT:

> Chcemy zrozumieć środowisko w swoim domu, w jaki sposób zmienia się ono na przestrzeni czasu i dokonywać korekty w celu zwiększenia komfortu, jednocześnie ograniczając koszty.

Opis problemu wydaje się dość prosty, ale jest to bardzo ogólny cel. Możemy go zawęzić, definiując kluczowe działania i obiekty w opisie problemu. Celem jest wyłuskanie *co*, *dlaczego* i *jak*. Na początku przeanalizujemy *co* i *dlaczego*, a później zidentyfikujemy działanie (lub działania), które projekt powinien uwzględnić w tym procesie.

Analiza problemu

Prezentowane w tej książce ćwiczenia będą koncentrować się na budowaniu rozwiązania IoT, które pomoże w lepszym zrozumieniu środowiska domowego i odpowiednim reagowaniu. Przyjęliśmy założenie, że użytkownik chce wiedzieć, co dzieje się w jego domu (w granicach rozsądku) i podejmować pewne działania, gdy jest to uzasadnione (na przykład włączając klimatyzację, jeśli temperatura powietrza jest za wysoka).

Ta część metody projektowania analizuje trzy kluczowe działania:

Mierzenie: Zbieranie danych
: Zdefiniujmy je jako to, co można ustalić za pomocą czujników np. temperatury, wilgotności itp. Koncentruje się na przechwytywaniu i przesyłaniu *danych telemetrycznych*

(danych pomiarów). Akcja, a dokładniej kategoria akcji, zostanie nazwana *zbieraniem danych* i będzie zawierać następujące elementy danych (w przyszłości można dodać ich więcej):

- temperatura,
- względna wilgotność,
- ciśnienie atmosferyczne,
- wydajność systemu (metryki użycia procesora, pamięci i dysku).

Modelowanie: Określenie odpowiednich zmian względem punktu odniesienia
: Aby decydować, które dane są istotne oraz czy zmiana wartości ma duże znaczenie, trzeba nie tylko zbierać dane, ale również przechowywać i śledzić dane szeregów czasowych dla czynników badanych za pomocą czujników (takich jak temperatura, wilgotność itd., jak wskazano we wcześniejszej definicji). Zwykle określamy to mianem *konwersji dane → informacja*. W niniejszej książce ta kategoria będzie nazywana *zarządzaniem danymi*.

Zarządzanie: Podejmowanie działania
: Ustalimy pewne podstawowe reguły określające, czy przekroczone zostały jakieś istotne progi, co oznacza po prostu, że trzeba będzie wysłać gdzieś sygnał, gdy przekroczony zostanie próg, który wymaga podjęcia pewnej akcji (np. przekręcenia termostatu w górę lub w dół). Zwykle określamy to mianem *konwersji informacja → wiedza*. W niniejszej książce ta kategoria będzie nazywana *wyzwalaczami systemu*.

W prowadzonym kursie uniwersyteckim poświęconym Internetowi rzeczy autor książki często odwołuje się do cyklu Mierzenie, Modelowanie i Zarządzanie. Reprezentuje on ważne aspekty każdego projektu IoT, które prowadzą do osiągnięcia przez system określonych celów (lub wyników) biznesowych.

Definiowanie odpowiednich wyników

Skoro wiadomo już, jakie działania trzeba podjąć, pora na przeanalizowanie składowej *dlaczego* opisu problemu. Można podsumować ją, używając następujących dwóch argumentów:

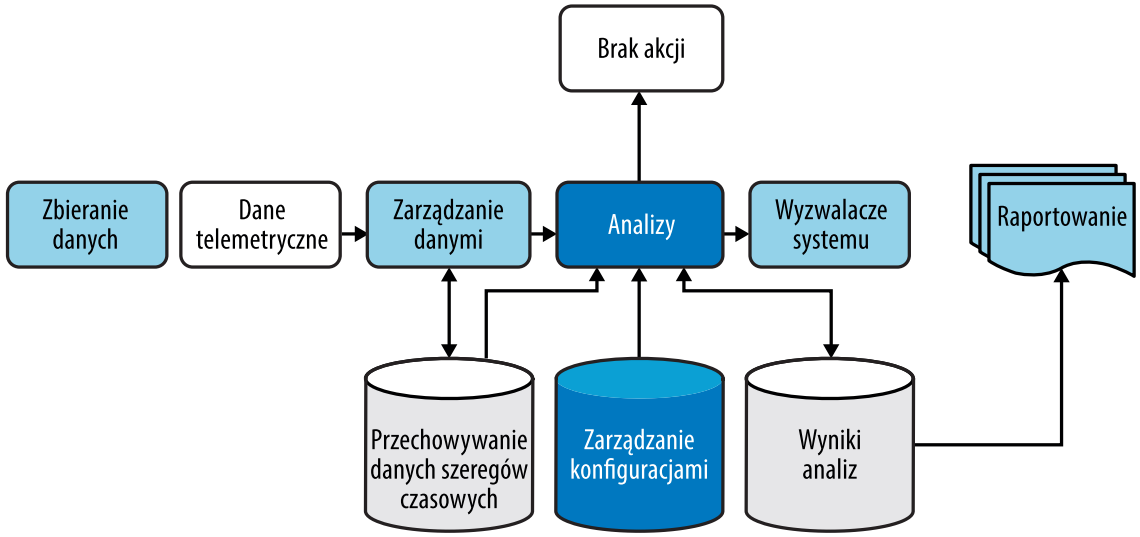
- *Zwiększenie wygody:* Najlepiej byłoby utrzymywać stałą temperaturę i wilgotność domu. Jednak sprawa nieco się komplikuje, gdy uwzględni się liczbę pokoi, sposób ich wykorzystywania itd. Ta kategoria akcji nazywana będzie *zarządzaniem konfiguracją* i jest ona powiązana zarówno z zarządzaniem danymi, jak i wyzwalaczami systemu.
- *Oszczędność:* Nie jest to prosta sprawa. Najbardziej oczywistym sposobem wprowadzenia oszczędności jest po prostu niewydawanie żadnych pieniędzy! Jednak ponieważ prawdopodobnie trzeba przeznaczyć pewne środki finansowe na ogrzanie, schłodzenie lub nawilżenie określonych pomieszczeń, warto to zoptymalizować – nie za dużo (by nie marnować pieniędzy) i nie za mało (by w zimie nie zamarzły rury). Ponieważ

problem może być dość złożony i wymagać uwzględnienia np. kosztów energii, zmian pór roku, a także aspektów związanych z zarządzaniem konfiguracją, prawdopodobnie potrzebne będą pewne zaawansowane analizy. Ta kategoria akcji będzie nazywana *analizami*.

Jak można zauważyć, poszczególne działania w częściach *co* i *dlaczego* noszą nazwy kategorii akcji, które pomogą w projektowaniu rozwiązania, gdy nadejdzie czas zajęcia się częścią *jak*. Dla przypomnienia, są to kategorie: zbieranie danych, zarządzanie danymi, wyzwalacze systemu, zarządzanie konfiguracjami i analizy. W fazie implementacji każda z tych części zostanie dogłębnie zbadana.

Chociaż opis problemu wydaje się z pozoru dość banalny, okazuje się, że elementy wchodzące w skład rozwiązania tego problemu występują w wielu innych systemach IoT. Istnieje potrzeba zbierania danych u źródła, przechowywania i analizowania tych danych oraz podejmowania akcji, jeśli pewien wskaźnik sugeruje, że przyniesienie to korzyści. Gdy zdefiniujemy architekturę IoT (która odnosi się do konkretnego problemu) i zaczniemy implementować jej komponenty, zobaczymy, że można by wykorzystać ją do rozwiązania wielu innych problemów.

Oto uproszczony przepływ danych reprezentujący ten proces decyzyjny: na diagramie przepływu danych zaprezentowanym na rysunku 1-1 wyróżniona została każda kategoria akcji.



Rysunek 1-1 *Prosty przepływ danych IoT*

Większość systemów IoT wymaga przynajmniej kilku z pięciu wspomnianych kategorii akcji. To oznacza, że możemy zdefiniować architekturę, która mapuje je do diagramu systemów, a następnie zacząć tworzyć komponenty oprogramowania, które implementują część systemu.

Z punktu widzenia inżynierów dopiero tu zaczyna się prawdziwa zabawa, dlatego zabierzmy się do definiowania architektury, która wspiera nasz opis problemu (i której tak naprawdę będzie można użyć także w innych sytuacjach).

Tworzenie architektury rozwiązania

Organizacja, struktura i czytelność są wyznacznikami dobrej architektury, jednak zbyt duży nacisk na nie może prowadzić do powstania sztywnego systemu, który trudno będzie skalować w przyszłości. Natomiast dążenie do uzyskania architektury, która spełnia wszystkie możliwe wymagania, może uniemożliwić ukończenie pracy (a nawet jej rozpoczęcie). Trzeba znaleźć złoty środek, definiując architekturę z pewną dozą elastyczności na przyszłość, ale jednocześnie dość precyzyjnie ją określając. To pozwoli nam skoncentrować się na szybkim uzyskaniu rozwiązania i jednocześnie umożliwi dostosowywanie go w przyszłości. Jednak najpierw trzeba zdefiniować klika podstawowych terminów, by pomóc w zdefiniowaniu architektury bazowej, na której będzie się opierać rozwiązanie.

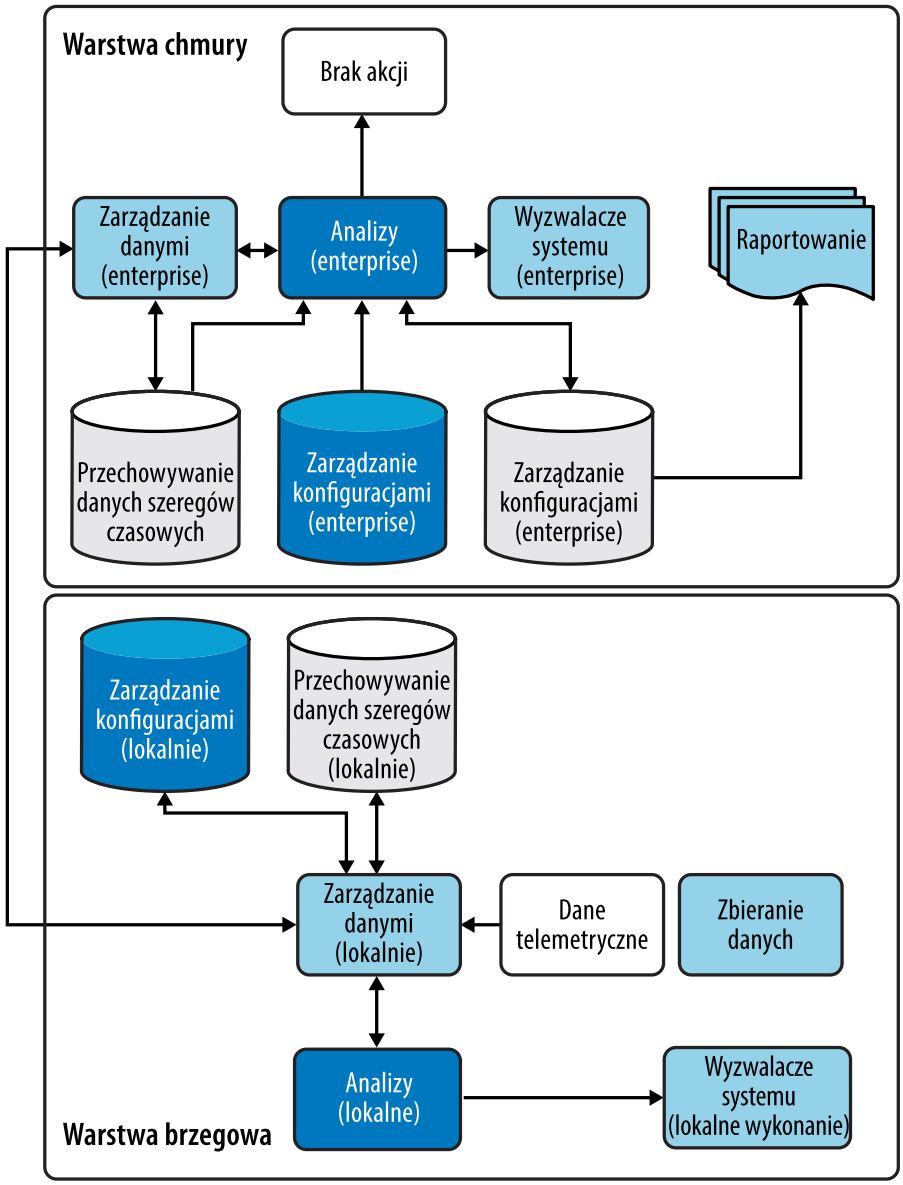
Jak pamiętamy z rysunku P-1 z przedmowy, systemy IoT są zwykle projektowane z myślą o przynajmniej dwóch (a czasem trzech i więcej) warstwach architektonicznych. Dzięki temu można rozdzielić funkcje zarówno fizycznie, jak i logicznie, co umożliwia elastyczne schematy wdrożeń. To oznacza, że usługi działające w warstwie chmury mogą z technicznego punktu widzenia być uruchomione w dowolnym miejscu na świecie, natomiast urządzenia działające w warstwie brzegowej muszą znajdować się w tym samym miejscu, co systemy fizyczne, które będą obiektem pomiarów. Jak wynika z rysunku P-1, przykładem takiego podziału na warstwy może być urządzenie ograniczone z czujnikami lub siłownikami komunikującymi się z urządzeniem bramy, które z kolei komunikują się z usługą w chmurze i vice versa.

Ponieważ musimy zadecydować, gdzie zaimplementować pięć wspomnianych wcześniej kategorii funkcji, musimy określić ich lokalizację w architekturze. Niektóre komponenty mogą być uruchomione blisko centrum działania, a inne w chmurze, gdzie możemy łatwo uzyskiwać do nich dostęp (a nawet dostosowywać je). Wracając do warstwy brzegowej i warstwy chmury omówionych w przedmowie, oto w jaki sposób można zmapować każdą z kategorii akcji z części *co* i *dlaczego* do konkretnych warstw:

- Warstwa brzegowa (urządzenia ograniczone i urządzenia bramy): zbieranie danych, zarządzanie danymi, wyzwalacze urządzeń, zarządzanie konfiguracjami i analizy.
- Warstwa chmury (usługi w chmurze): zarządzanie danymi, zarządzanie konfiguracjami i analizy.

Dlaczego umieściliśmy w warstwie brzegowej i warstwie chmury podobne funkcje? Częściowo dlatego, że zaistniała taka konieczność, ale również dlatego, że możemy. Techniczne granice i rozdział obowiązków między warstwą chmury a brzegową stają się coraz bardziej rozmyte w miarę jak rośnie moc obliczeniowa, a potrzeby biznesowe wymuszają umieszczanie funkcji obliczeniowych i analitycznych „jak najbliżej brzegu". Na przykład niektóre autonomiczne decyzje mogą nie wymagać przesyłania wiadomości do chmury i z powrotem, ponieważ warstwa brzegowa może bezpośrednio nimi zarządzać. Dlatego ważne jest uwzględnienie tej możliwości, gdy i gdzie ma ona sens.

Rysunek 1-2 pokazuje, jak prosty przepływ danych z rysunku 1-1 wpisuje się w architekturę z warstwami.



Rysunek 1-2 *Koncepcyjny diagram przepływu danych IoT między warstwą brzegową a warstwą chmury*

Jak można zauważyć, niektóre funkcje znowu są współdzielone, a kategorie akcji zaimplementowane w obu warstwach. Z reguły duplikacja nie jest wskazana, ale w tym przypadku może przynosić korzyści. Analizy mogą posłużyć do określania, czy wyzwalacz powinien zostać wysłany do urządzenia, w zależności od pewnych podstawowych ustawień np. jeśli temperatura w domu przekracza 30°C, być może warto od razu zainicjować system klimatyzacji i zacząć chłodzenie do np. 22°C. W tym przypadku nie trzeba polegać na zdalnej usłudze w chmurze, choć warto byłoby poinformować warstwę chmury o tym zdarzeniu i być może przechowywać pewne dane historyczne do późniejszych analiz.

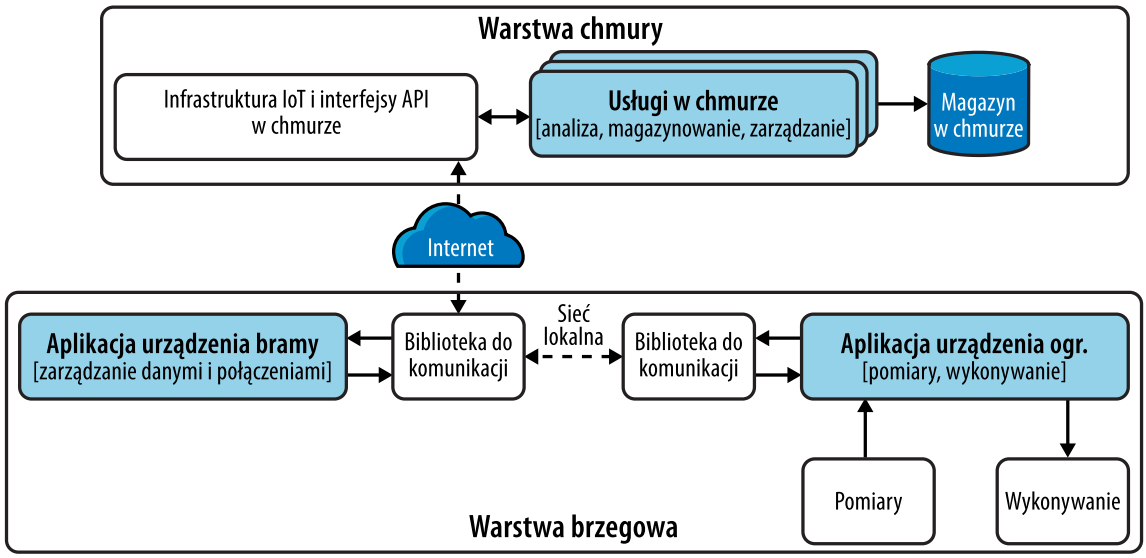
Architektura rozwiązania zaczyna nabierać kształtu. Teraz trzeba jedynie zmapować ją do diagramu systemu, aby móc podejmować interakcję ze światem fizycznym (przy użyciu czujników i siłowników). Ponadto dobrze byłoby określić taką strukturę elementów

w warstwie brzegowej, by unikać niepotrzebnego narażania komponentów na dostęp z Internetu. Tę funkcję można zaimplementować jako aplikację, która jest uruchamiana bezpośrednio na urządzeniu lub na laptopie lub w innym ogólnym systemie komputerowym, który oferuje logikę symulującą działanie czujnika i siłownika. Będzie to fundament dla jednej z dwóch aplikacji, których implementację zaczniemy w tym rozdziale.

Ponieważ ostatecznie dostęp do Internetu będzie potrzebny, projekt powinien zawierać bramę do realizowania tego i innych zadań. Tę funkcjonalność można zaimplementować jako część drugiej aplikacji, której implementację zaczniemy w tym rozdziale. Aplikacja zostanie zaprojektowana tak, by działać na urządzeniu bramy (lub jak wcześniejsza na laptopie, lub w innym ogólnym systemie komputerowym). Aplikacja urządzenia bramy (Gateway Device Application, GDA) oraz aplikacja urządzenia ograniczonego (Constrained Device Application, CDA) będą stanowiły „brzeg" projektu IoT, który w dalszej części książki będziemy nazywać warstwą brzegową architektury.

Będziemy potrzebować również usług analitycznych oraz funkcji przechowywania danych i zarządzania zdarzeniami w sposób, który jest bezpieczny i jednocześnie dostępny dla urządzenia bramy oraz dla ludzi. Można osiągnąć ten cel na wiele sposobów, ale niniejsza książka koncentruje się na realizowaniu tych funkcji przy użyciu jednej lub więcej usług w chmurze.

Rysunek 1-3 przedstawia nowy widok, który lepiej wyjaśnia, co będzie budowane i jak możemy zacząć dołączać akcje z pięciu wspomnianych kategorii. W szarych ramkach przedstawione zostały usługi w warstwie chmury oraz dwie aplikacje w warstwie brzegowej, które będą zawierały odpowiednio funkcje urządzenia ograniczonego oraz funkcje urządzenia bramy.



Rysunek 1-3 *Koncepcyjna uproszczona architektura logiczna IoT z warstwą brzegową i chmury*

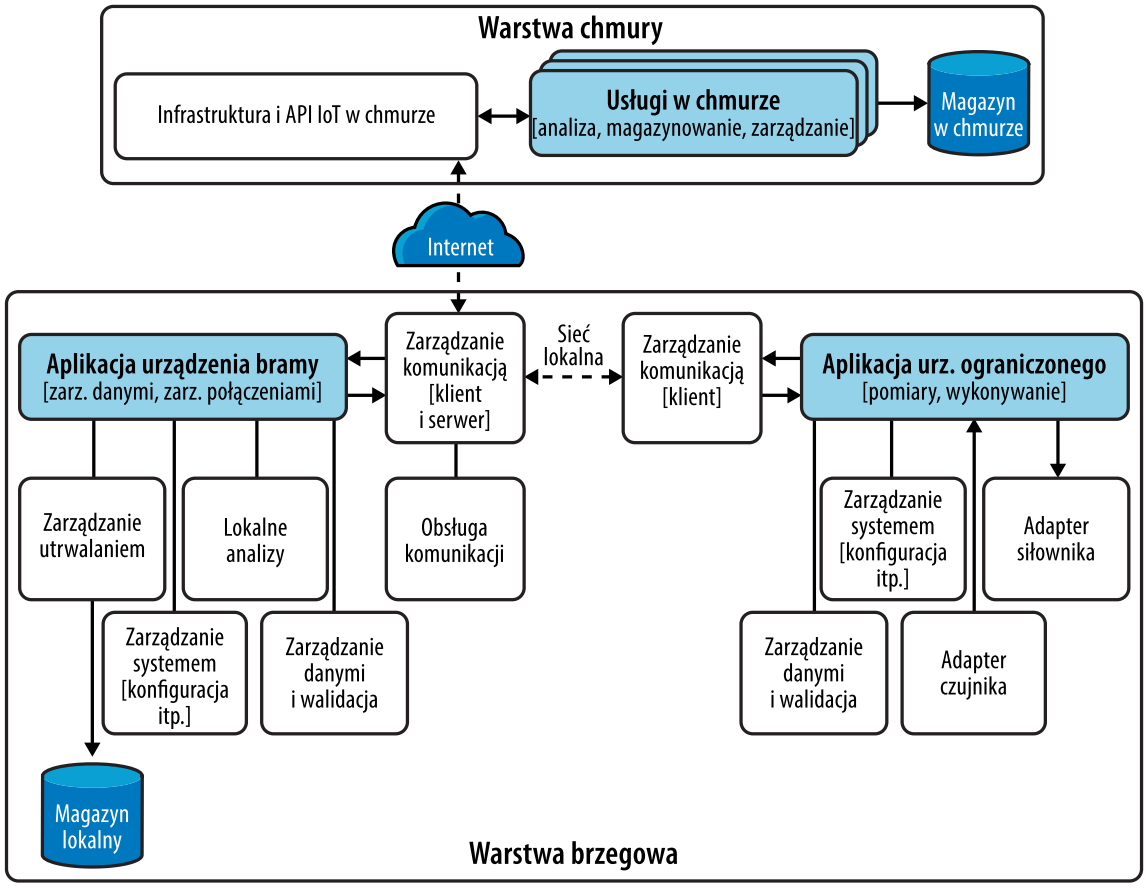
A oto dodatkowe informacje o każdej z nich:

Aplikacja urządzenia ograniczonego (CDA)
: Ta aplikacja zostanie zbudowana tak, by działała na urządzeniu ograniczonym (emulowanym lub prawdziwym) i będzie dostarczała funkcje zbierania danych oraz wyzwalaczy systemu. Będzie obsługiwała interfejs między czujnikami (które odczytują dane ze środowiska) a siłownikami (które wykonują akcje, takie jak włączenie lub wyłączenie termostatu). Będzie również ogrywała rolę w realizowaniu akcji, gdy konieczne jest wykonanie. Ostatecznie zostanie połączona z biblioteką do komunikacji, aby przesyłać wiadomości i odbierać wiadomości z aplikacji urządzenia bramy.

Aplikacja urządzenia bramy (GDA)
: Ta aplikacja zostanie zbudowana tak, by działała na urządzeniu bramy (emulowanym lub prawdziwym) i będzie dostarczała funkcje zarządzania danymi, analizy i zarządzania konfiguracją. Główną jej rolą jest zarządzanie danymi i połączeniami między aplikacją CDA a usługami w chmurze, które znajdują się w warstwie chmury. W zależności od sytuacji będzie zarządzała danymi lokalnie lub *czasami* podejmowała akcję, przesyłając polecenie do urządzenia ograniczonego, które wyzwala wykonanie. Będzie również zarządzała pewnymi ustawieniami konfiguracyjnymi, które reprezentują zakres nominalny dla danego środowiska i będzie przeprowadzała pewne wstępne analizy po otrzymaniu nowych danych telemetrycznych.

Usługi w chmurze
: Wszystkie aplikacje i funkcje usług w chmurze zajmują się zaawansowanym przetwarzaniem danych i ich przechowywaniem, ponieważ teoretycznie można skalować je w nieskończoność. To oznacza po prostu, że jeśli zostały odpowiednio zaprojektowane, można dodawać dowolną liczbę urządzeń, przechowywać dowolną ilość danych i przeprowadzać dogłębną analizę tych danych (trendów, maksimów, minimów, wartości konfiguracji itd.), a jednocześnie pokazywać odpowiednie informacje użytkownikowi końcowemu, a być może nawet generować akcje warstwy brzegowej, gdy przekroczony zostanie jakiś zdefiniowany próg. Z technicznego punktu widzenia usługi w chmurze w środowisku IoT mogą obsługiwać wszystkie wspomniane wcześniej kategorie akcji, z wyjątkiem zbierania danych (ponieważ nie mogą one bezpośrednio podejmować akcji pomiaru ani wykonania). Zbudujemy pewne usługi w chmurze realizujące te zadania, ale przede wszystkim użyjemy uniwersalnych usług dostępnych na niektórych platformach chmury.

Integrując wszystkie te komponenty w jedną szczegółową architekturę logiczną, rysunek 1-4 przedstawia, w jaki sposób główne komponenty logiczne w dwóch warstwach architektonicznych współpracują z innymi komponentami.



Rysunek 1-4 *Koncepcyjna szczegółowa architektura logiczna IoT z warstwami brzegową i chmury*

Rysunki 1-3 oraz 1-4 posłużą jako architektura bazowa we wszystkich prezentowanych w tej książce ćwiczeniach.

Skoro wiemy już, co nas czeka, przygotujmy środowisko programistyczne, by móc zacząć pisać kod.

Przygotowywanie środowiska programistycznego i testowego

Budowanie i instalowanie kodu w różnych systemach operacyjnych, konfiguracjach sprzętu i systemach konfiguracji nie jest prostym zadaniem. Typowy projekt IoT wymaga obsłużenia nie tylko różnych komponentów sprzętowych, ale i niezliczonych sposobów programowania i wdrażania rozwiązań na tych platformach, nie mówiąc o różnicach w mechanizmach ciągłej integracji/ciągłego wdrażania (continuous integration/continuous deployment, CI/CD) w zależności od dostawcy usług w chmurze.

Od czego w ogóle zacząć w obliczu tak wielu wyzwań? Najlepiej zacząć od zastanowienia się, jaki problem próbujemy rozwiązać. Jako programiści, chcemy zaimplementować swój projekt IoT w kodzie, przetestować go i spakować w coś, co można łatwo rozdystrybuować i bezpiecznie wdrożyć w jednym lub wielu systemach. Wyzwania tego

projektu można podzielić na dwie fazy budowania, testowania i wdrażania, które można zmapować do dwóch zastosowanych warstw architektonicznych: środowiska warstwy brzegowej i środowiska warstwy chmury. Funkcje dostępne w warstwie chmury zostaną szczegółowo omówione w rozdziale 10. Wcześniejsza część książki będzie koncentrować się na warstwie brzegowej.

Warstwa brzegowa może zawierać specjalistyczny sprzęt, możemy także zasymulować pewne zachowania systemu i emulować pewne komponenty sprzętowe w lokalnym środowisku programistycznym. To znacznie ułatwi wdrażanie i w zupełności wystarczy do realizacji wszystkich ćwiczeń prezentowanych w tej książce.

Rozwijając IoT, można przyjąć wiele różnych strategii, ale my koncentrujemy się na trzech konkretnych ścieżkach. Jedna z nich jest całkowicie symulowana i jak wspomnieliśmy wcześniej, wystarczy do zrealizowania wszystkich podstawowych ćwiczeń w tej książce. Pozostałe dwie wymagają sprzętu właściwego dla IoT i zostaną omówione bardziej szczegółowo w rozdziale 4.

Zintegrowane symulowane wdrożenie

Ta strategia nie wymaga żadnego specjalistycznego sprzętu i pozwala na użycie deweloperskiej stacji roboczej (laptopa) zarówno jako urządzenia bramy, jak i urządzenia ograniczonego. To oznacza, że uruchomimy aplikacje GDA i CDA w lokalnym środowisku komputerowym. Będziemy emulować sprzętowe czujniki i siłowniki, budując proste symulatory programowe w celu dołączenia tych funkcji do aplikacji CDA. Wszystkie ćwiczenia, za wyjątkiem dostępnych online opcjonalnych ćwiczeń z rozdziału 4 (*https://oreil.ly/Zm4Po*), będą działały przy zastosowaniu tego sposobu wdrożenia.

Rozdzielone fizyczne wdrożenie

Ta strategia wymaga użycia urządzenia sprzętowego, takiego jak Raspberry Pi, które umożliwia nawiązywanie połączenia i interakcję z prawdziwymi czujnikami oraz siłownikami. Chociaż wiele dostępnych na rynku urządzeń jednopłytkowych (SBC) może pełnić rolę prawdziwych stacji roboczych, w tej strategii będą one nazywane urządzeniami ograniczonymi i aplikacja CDA będzie uruchamiana bezpośrednio na urządzeniu. Jak w przypadku zintegrowanego symulowanego wdrożenia, aplikacja GDA będzie uruchamiana w lokalnym środowisku komputerowym.

Jak wspomniano w przedmowie, dokument IETF RFC 7228 definiuje różne klasy urządzeń ograniczonych (nazywanych również *węzłami ograniczonymi*). Są to klasy: 0 (bardzo ograniczone), 1 (ograniczone) oraz 2 (nieco ograniczone)[1]. W tej książce przyjęto założenie, że aplikacja CDA może być uruchamiana na klasie 2 urządzeń lub nawet na urządzeniach o większej mocy, które zwykle wspierają pełny stos protokołów sieciowych bazujących na IP, co oznacza, że stosowane w tej książce protokoły będą zazwyczaj działały na tego typu urządzeniach. Chociaż z technicznego punktu

1 Carsten Bormann, Mehmet Ersue i Ari Keränen, „Terminology for Constrained-Node Networks" (*https://tools.ietf.org/html/rfc7228*), IETF Informational RFC 7228, Maj 2014, str. 8–10.

widzenia można połączyć urządzenia klasy 2 bezpośrednio z Internetem, wszystkie przykłady i ćwiczenia będą polegały na niebezpośredniej interakcji z Internetem poprzez aplikację GDA.

Zmieszane fizyczne wdrożenie
: Ta strategia jest niemal identyczna jak rozdzielone wdrożenie, ale obie aplikacje, CDA i GDA, będą działały na urządzeniu SBC. To w praktyce oznacza, że można zdecydować się na wdrożenie każdej aplikacji na osobnym urządzeniu SBC, choć nie jest to wymagane w żadnym z przedstawionych ćwiczeń.

Osoby, które chcą zastosować jedną z dwóch ostatnich strategii, mają do dyspozycji szeroką gamę niedrogich urządzeń SBC. Jedyne ćwiczenia w tej książce, które wymagają użycia sprzętu, to opcjonalne ćwiczenia w rozdziale 4 i choć można zaimplementować je na innych platformach sprzętowych, zostały one zaprojektowane z myślą o następującym sprzęcie: Raspberry Pi Model 3 lub 4 oraz płytka Sense HAT (która łączy się z interfejsem GPIO i wykorzystuje magistralę Inter-Integrated Circuit [I2C] do komunikacji urządzeń). Osoby, które chcą użyć w tych ćwiczeniach innego urządzenia, mogą poszukać takiego, które oferuje następujące funkcje: GPIO, I2C, operacje sieciowe TCP/IP i UDP/IP za pośrednictwem Wi-Fi lub Ethernetu, wsparcie dla systemu operacyjnego bazującego na systemie Linux (takiego jak Debian lub jego pochodna) oraz wsparcie dla Python 3 oraz Java 11 (lub wyższych wersji).

Przedstawione w tej książce ćwiczenia będą koncentrować się na strategii zintegrowanego symulowanego wdrożenia. W części II wprowadzona zostanie idea integracji ze światem fizycznym i w rozdziale 4 omówione zostaną pewne koncepcje związane z integracją ze sprzętem, jednak książka nadal skupiać się będzie na symulowanych danych i emulowanym sprzęcie.

Niezależnie od wybranej ścieżki wdrożenia, wszystkie ćwiczenia i przykłady bazują na założeniu, że programowanie i wdrażanie będzie odbywać się na jednej stacji roboczej. To wiąże się z trzyetapowym procesem, który obejmuje przygotowanie środowiska programistycznego, definiowanie strategii testowania i określanie metody automatyzacji budowania oraz wdrażania. W tym rozdziale opiszemy podstawy tego procesu konieczne do rozpoczęcia pracy, ale omówienie każdej z faz będzie stopniowo rozszerzane w późniejszych ćwiczeniach, które wprowadzają dodatkowe zależności i wymagania dotyczące testowania oraz automatyzacji.

Niektórzy czytelnicy mają już doświadczenie w rozwijaniu aplikacji w językach Java i Python z wykorzystaniem własnego środowiska programistycznego. W takiej sytuacji zalecamy zapoznanie się z pierwszym etapem – przygotowaniem środowiska programistycznego – w celu ustalenia, czy dotychczasowe środowisko programistyczne (w tym system operacyjny, środowisko uruchomieniowe Java oraz interpreter Python) spełnia wymagania ćwiczeń.

Krok I: Przygotowanie środowiska programistycznego

Jak wspomnieliśmy wcześniej, aplikacja CDA zostanie napisana w języku Python, natomiast aplikacja GDA w języku Java. Choć zasadniczo oznacza to, że większość ćwiczeń można zrealizować w dowolnym systemie operacyjnym wspierającym wersję Python 3.7 lub wyższą i wersję Java 11 lub wyższą, przed skonfigurowaniem środowiska programistycznego warto mieć świadomość, że użyjemy kilku rozwiązań właściwych dla systemu Linux:

- Rozdział 4: Emulator sprzętu, który zostanie omówiony w tym rozdziale, wymaga środowiska bazującego na Unix wraz z serwerem X11 wspierającym jego graficzny interfejs użytkownika (GUI). Linux i macOS powinny wystarczyć, natomiast Windows będzie wymagał podsystemu Windows Subsystem for Linux (WSL[2]) i dodatkowo serwera X11.
- Rozdział 8: Serwer CoAP[3] bazujący na języku Python to opcjonalne ćwiczenie, które zostało przetestowane z klientem Java z rozdziału 9 w systemach Windows 10, macOS oraz Linux. Jednak zaawansowane funkcje protokołu i/lub biblioteki mogą być zależne od przypisań właściwych dla systemu Linux, które mogą być niekompatybilne z systemami Windows oraz macOS.
- Rozdział 9: Niektóre ćwiczenia, w których budujemy klienta CoAP bazującego na języku Python, aktualnie są zależne od przypisań właściwych dla systemu Linux, które mogą być niekompatybilne z systemami Windows oraz macOS.

Chociaż omówione zostanie przygotowanie środowisk Linux, macOS oraz Windows, zaleca się użycie systemu Linux jako środowiska programistycznego, aby uniknąć wspomnianych problemów z integracją. Dodatkowe informacje o przygotowywaniu środowiska operacyjnego i kwestiach związanych ze zgodnością bibliotek znaleźć można w karcie PIOT-CFG-01-001 (*https://oreil.ly/PrtRU*). Większość bibliotek open source, na których opierają się ćwiczenia, jest aktywnie utrzymywana, jednak mogą pojawić się wyjątki. Warto sprawdzić, jaka jest najnowsza przetestowana wersja dla każdej z bibliotek i jej ograniczenia w zakresie zgodności z różnymi środowiskami operacyjnymi.

Autor książki zwykle programuje w systemie Windows 10, ale aplikacje wykonuje i testuje przy użyciu WSL z jądrem Ubuntu 20.04 LTS. Czytelnicy, którzy zdecydowanie wolą użyć środowiska operacyjnego innego niż Linux, mogą zbudować całościowe rozwiązanie, pomijając ćwiczenia z rozdziałów 4, 8 oraz 9, zamiast tego polegając na symulatorach danych opisanych

2 Dodatkowe informacje o podsystemie WSL i instalowaniu go na wybranych platformach można znaleźć na stronie *https://oreil.ly/YK9Rb*.

3 CoAP (Constrained Application Protocol) to protokół do obsługi wiadomości, który zostanie omówiony w części III (a dokładniej w rozdziałach 8 i 9).

w rozdziale 3 i używając protokołu MQTT[4] do łączności. Jednak należy uważnie przestudiować wszystkie rozdziały, ponieważ każdy z nich dostarcza cenne wskazówki pomocne w rozwijaniu aplikacji w przyszłości.

Aby sprawdzić, czy na stacji roboczej zainstalowane jest oprogramowanie, które wspiera te języki i powiązane zależności, należy postępować zgodnie z poniższą instrukcją:

1. Zainstaluj na swojej stacji roboczej wersję Python 3.7 lub wyższą (w czasie pisania książki najnowsza wersja to 3.9.4, choć w naszym środowisku WSL wykorzystujemy wersję 3.8.5). Aby sprawdzić, czy jest ona już zainstalowana i zainstalować ją, jeśli jeszcze nie jest:
   a. Otwórz okno terminala lub konsoli i wpisz następujące polecenie (zwróć uwagę na dwa myślniki przed parametrem):

      ```
      $ python3 --version
      ```

   b. Powinien zostać wyświetlony wynik podobny do następującego:

      ```
      Python 3.8.5
      ```

   c. Jeśli zwrócona zostanie wersja niższa niż 3.7 lub w przypadku pojawienia się komunikatu o błędzie (np. „not found”), trzeba zainstalować wersję Python 3.7 lub wyższą. Postępuj zgodnie z instrukcjami dla swojego systemu operacyjnego (Windows, macOS, Linux) ze strony *https://www.python.org/downloads*.

Czasami trzeba pobrać kod źródłowy interpretera języka Python, a następnie zbudować i zainstalować pliki wykonalne. Jeśli okaże się to konieczne, należy postępować zgodnie z instrukcjami ze strony *https://devguide.python.org/setup/*. Warto mieć świadomość, że może być to czasochłonny proces.

2. Zainstaluj narzędzie pip, pobierając skrypt ładowania i instalacji (*https://bootstrap.pypa.io/get-pip.py*). Jak w przypadku instrukcji Python z etapu 1, najpierw sprawdź, czy narzędzie pip jest już zainstalowane. Jeśli używasz WSL lub Ubuntu, może pojawić się konieczność zainstalowania narzędzia pip przy użyciu menedżera pakietów apt.
   a. Otwórz okno terminala lub konsoli i wpisz następujące polecenie (ponownie zwróć uwagę na dwa myślniki przed parametrem):

      ```
      $ pip --version
      ```

   b. Powinien zostać wyświetlony wynik podobny do następującego:

      ```
      pip 21.0.1
      ```

4 MQTT (Message Queuing Telemetry Transport) to protokół do obsługi wiadomości, który zostanie omówiony w części III (a dokładniej w rozdziałach 6 i 7).

c. Jeśli narzędzie pip nie jest zainstalowane lub jego wersja jest nieaktualna, użyj polecenia Python do wykonania skryptu instalacyjnego pip. Otwórz okno terminala lub konsoli i wpisz:

```
$ python3 get-pip.py
```

3. Upewnij się, że na stacji roboczej zainstalowana jest wersja Java 11 lub wyższa (w czasie pisania książki najnowszą wersją OpenJDK była Java 14). Możesz sprawdzić, czy jest ona już zainstalowana i zainstalować ją, gdy nie jest, podejmując poniższe kroki:

a. Otwórz okno terminala lub konsoli i wpisz następujące polecenie (parametr jest poprzedzony dwoma myślnikami, choć polecenie prawdopodobnie zadziała też z jednym):

```
$ java --version
```

b. Powinien zostać wyświetlony wynik podobny do następującego (upewnij się, że jest to przynajmniej Java 11):

```
openjdk 14.0.2 2020-07-14
OpenJDK Runtime Environment (build 14.0.2+12-Ubuntu-120.04)
OpenJDK 64-Bit Server VM (build 14.0.2+12-Ubuntu-120.04, mixed mode, sharing)
```

c. Jeśli otrzymasz komunikat o błędzie (np. „not found"), musisz zainstalować wersję Java 11 lub wyższą. Postępuj zgodnie z instrukcjami dla swojej platformy (Windows, macOS lub Linux) na stronie internetowej OpenJDK (*https://openjdk.java.net/install*).

4. Zainstaluj program Git. Wejdź na stronę „Installing Git" (*https://oreil.ly/cmRX3*) i przejrzyj instrukcje dotyczące odpowiedniego systemu operacyjnego.

Aby móc wykonać prezentowane w tej książce ćwiczenia i przygotować swoje środowisko programistyczne, trzeba choć trochę znać Git – narzędzie do zarządzania kodem źródłowym i wersjami. Wiele środowisk IDE automatycznie wspiera zarządzanie kodem źródłowym dzięki wbudowanemu klientowi Git. W poprzednim kroku program Git został zainstalowany za pomocą wiersza polecenia, dzięki czemu można uruchamiać polecenia Git niezależnie od środowiska IDE. Więcej informacji o wykorzystywaniu narzędzia Git z wiersza polecenia znaleźć można w dokumentacji samouczka Git (*https://git-scm.com/docs/gittutorial*).

Można wykorzystywać program Git jako niezależne narzędzie do zarządzania kodem źródłowym na lokalnej deweloperskiej stacji roboczej i zarządzać kodem źródłowym w chmurze przy użyciu jednej z wielu darmowych i komercyjnych usług. My używamy usługi GitHub[5] i będziemy wykorzystywać pewne oferowane przez tę platformę funkcje, które wspierają potoki CI/CD (lub kroki przepływów pracy) i tym samym umożliwiają automatyzowanie procesu budowania, testowania, tworzenia pakietów oraz wdrażania.

5 Dodatkowe informacje znaleźć można na stronie platformy GitHub (*https://github.com*).

5. Utwórz katalog roboczy projektu i pobierz kod źródłowy oraz testy jednostkowe z tej książki:
   a. Otwórz okno terminala lub konsoli, utwórz nowy katalog roboczy projektu i przejdź do niego. A później wpisz następujące polecenie:
      i. Linux/macOS:

```
mkdir $HOME/programmingtheiot cd $HOME/programmingtheiot
```

      ii. Windows:

```
mkdir C:\programmingtheiot cd C:\programmingtheiot
```

   b. Sklonuj następujące dwa repozytoria kodu źródłowego z tej książki, wpisując:

```
$ git clone https://github.com/programmingtheiot/python-components.git
$ git clone https://github.com/programmingtheiot/java-components.git
```

6. Przygotuj swoje środowisko wirtualne Python.

   Istnieją różne sposoby tworzenia wirtualnego środowiska pracy w Python i naszym celem nie jest opisanie wszystkich opcji. Wersja Python 3.3 i wyższe zawierają wbudowany moduł środowiska wirtualnego, dzięki czemu jeśli nie jesteśmy zwolennikami narzędzia virtualenv, nie musimy go instalować. Dodatkowe informacje o stosowaniu modułu venv można uzyskać na stronie *https://docs.python.org/3/library/venv.html*.

   a. Utwórz wirtualne środowisko Python. Otwórz okno terminala lub konsoli, zmień katalog na wybraną ścieżkę instalacji środowiska wirtualnego (np. *$HOME/programmingtheiot/.venv*) i utwórz środowisko wirtualne (venv) w następujący sposób:
      i. Linux/macOS:

```
$ python3 -m venv $HOME/programmingtheiot/.venv
```

      ii. Windows:

```
C:\programmingtheiot > python -m venv C:\programmingtheiot \.venv
```

   b. Zainstaluj wymagane moduły Python. Zmień katalog na ten, w którym znajduje się kod `python-components` sklonowany wcześniej z platformy GitHub, a następnie wpisz:
      i. Linux/macOS:

```
$ cd $HOME/programmingtheiot
$ . .venv/bin/activate
(venv) $ pip install -r basic_imports.txt
```

      ii. Windows:

```
cd C:\programmingtheiot
C:\programmingtheiot > .venv\Scripts\activate.bat
(.venv) C:\programmingtheiot > pip install -r basic_imports.txt
```

c. Upewnij się, że narzędzie virtualenv może być aktywowane. Można łatwo to sprawdzić, aktywując je przy użyciu skryptu `activate`, a następnie dezaktywując przy użyciu polecenia `deactivate` z wiersza poleceń:

   i. Linux/macOS:

   ```
   $ . .venv/bin/activate (venv) $ deactivate
   ```

   ii. Windows:

   ```
   C:\programmingtheiot > .venv\Scripts\activate.bat (.venv) C:\
   programmingtheiot > deactivate
   ```

Konfiguracja stacji roboczej dobiega już końca. Kolejnym krokiem jest skonfigurowanie środowiska programistycznego i sklonowanie przykładowego kodu źródłowego książki.

Konfigurowanie zintegrowanego środowiska programistycznego

Istnieje wiele doskonałych narzędzi i zintegrowanych środowisk programistycznych (Integrated Development Environment, IDE), które pomagają programistom w pisaniu, testowaniu i wdrażaniu aplikacji napisanych w językach Java oraz Python. Mamy pewne narzędzia, które bardzo dobrze znamy i które pasują do naszego stylu programowania. Czytelnicy też pewnie mają swoje ulubione narzędzia. Tak naprawdę wybór narzędzi nie jest ważny, o ile spełniają one pewne podstawowe wymagania, takie jak kolorowanie i uzupełnianie kodu, formatowanie i refaktoryzacja kodu, debugowanie, kompilowanie i tworzenie pakietów, testowanie jednostkowe i nie tylko oraz kontrola kodu źródłowego.

Na rynku dostępnych jest wiele fantastycznych środowisk IDE, zarówno komercyjnych, jak i open source. Przykłady z tej książki zostały napisane przy użyciu środowiska Eclipse (*https://www.eclipse.org/ide*) z zainstalowanym dodatkiem PyDev (*https://oreil.ly/YjLFD*), ponieważ spełnia ono nasze wymagania i oferuje wiele innych wygodnych funkcji, które regularnie wykorzystujemy w naszych projektach programistycznych. Czytelnicy mogą być zaznajomieni z innymi środowiskami IDE, takimi jak Visual Studio Code (*https://code.visualstudio.com*) oraz IntelliJ IDEA (*https://www.jetbrains.com/idea*), które wspierają zarówno programowanie w języku Java, jak i Python. Wybór środowiska IDE wykorzystywanego w ćwiczeniach z tej książki należy do czytelnika.

Dla czytelników, którzy umieją pisać kod, testować aplikacje i zarządzać nimi przy użyciu innego środowiska IDE, większość informacji przedstawianych w tej części rozdziału nie będzie żadną nowością. Jednak zalecamy jej przeczytanie, ponieważ przygotowuje ona do pracy nad aplikacjami GDA oraz CDA.

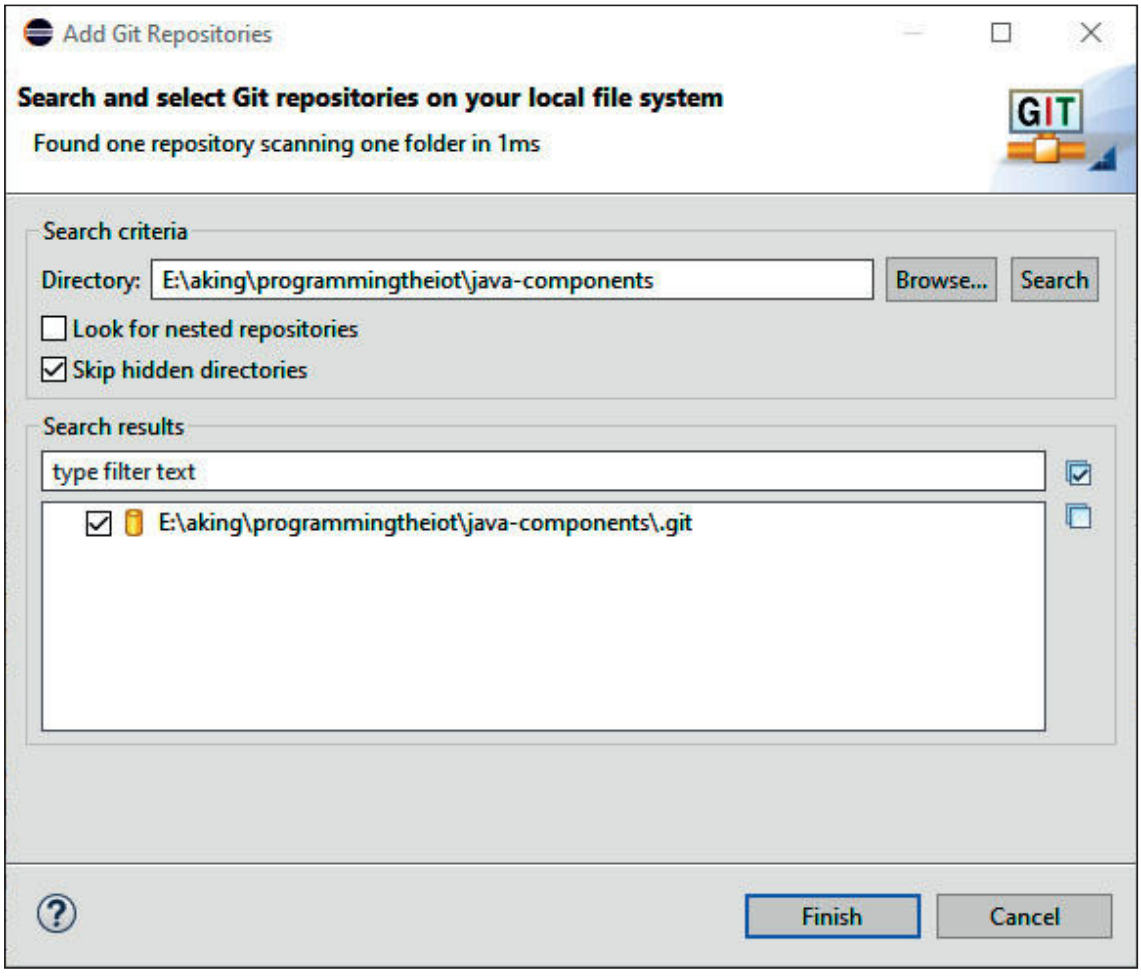
Przygotowywanie projektu Gateway Device Application

Pierwszym krokiem w tym procesie jest zainstalowanie najnowszej wersji Eclipse IDE do programowania w języku Java. Łącza do pobrania najnowszej wersji Eclipse znaleźć można na stronie *https://www.eclipse.org/downloads*. Jak będzie można zauważyć, dostępnych jest wiele odmian środowiska, ale na potrzeby tej książki wystarczy wybrać „Eclipse

IDE for Java Developers", a następnie postępować zgodnie z instrukcjami, aby zainstalować środowisko w swoim lokalnym systemie. Po zainstalowaniu uruchomiamy program Eclipse, wybieramy File (Plik) → Import (Importuj), znajdujemy Git → „Projects from Git" (Projekty Git) i klikamy przycisk Next (Dalej).

Zaznaczamy „Existing local repository" (Istniejące repozytorium lokalne), a następnie klikamy przycisk Next. Jeśli w katalogu domowym znajdują się już jakieś repozytoria Git, Eclipse najprawdopodobniej je rozpozna i zaprezentuje jako opcje do zaimportowania w następnym oknie dialogowym (niepokazanym). Aby ściągnąć nowo sklonowane repozytorium, klikamy przycisk Add (Dodaj), co spowoduje wyświetlenie następnego okna, pokazanego na rysunku 1-5. To okno pozwala na dodanie nowego repozytorium Git.

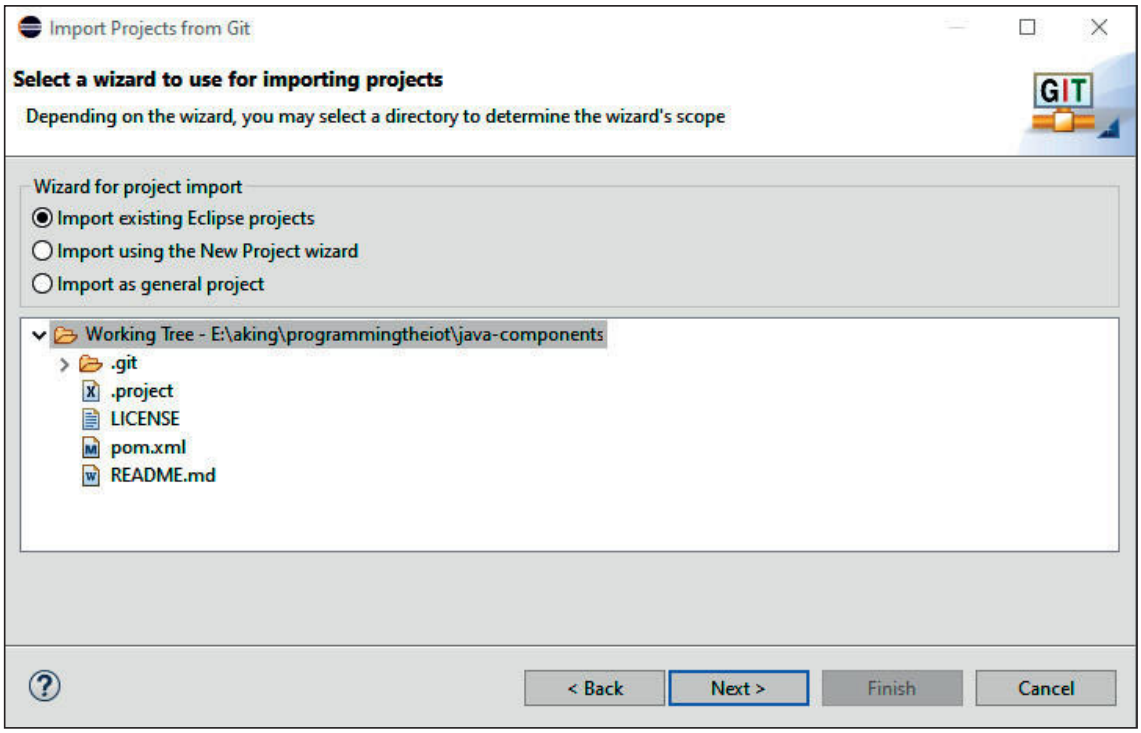
Na naszej stacji roboczej repozytorium do zaimportowania znajduje się w folderze *E:\aking\programmingtheiot\java-components*. Na komputerze czytelnika nazwa będzie prawdopodobnie inna, dlatego trzeba zwrócić uwagę na poprawne jej wpisanie!



Rysunek 1-5 *Importowanie java-components z lokalnego repozytorium Git*

Klikamy przycisk Finish (Zakończ), co spowoduje, że nowe repozytorium zostanie dodane do listy repozytoriów, które można zaimportować. Zaznaczamy to nowe repozytorium

i klikamy przycisk Next. Eclipse wyświetli kolejne okno dialogowe i zaproponuje zaimportowanie projektu przy użyciu jednej z kilku opcji, jak pokazano na rysunku 1-6.



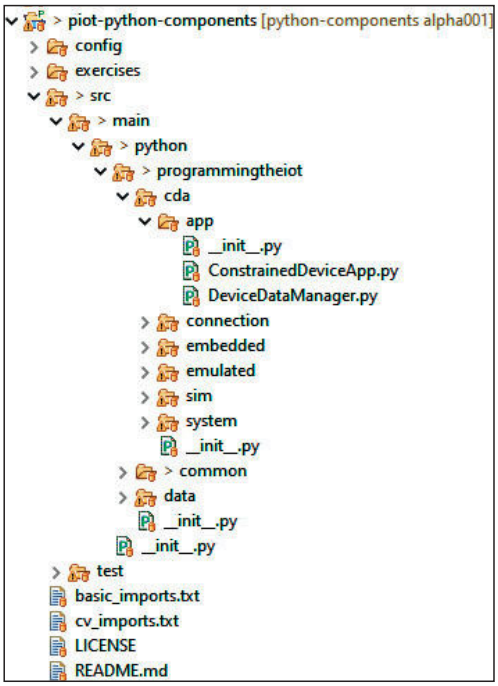
Rysunek 1-6 *Importowanie java-components jako istniejącego projektu Eclipse*

Można wybrać jedną z dwóch możliwości: zaimportować `java-components` jako istniejący projekt Eclipse z wykorzystaniem nowego kreatora projektu lub jako ogólny projekt. Czytelnikom, którzy nie planują całkowicie zmieniać projektu, zalecamy wybranie pierwszej opcji, czyli zaimportowanie istniejącego projektu. Ten proces wiąże się z wyszukaniem pliku `.project` w katalogu roboczym (dołączonym do każdego ze sklonowanych już repozytoriów), w wyniku czego powstanie nowy projekt Java o nazwie `piot-java-components`. Czytelnicy, którzy wolą utworzyć własny projekt, mogą usunąć go i zaimportować jako nowy projekt przy użyciu odpowiedniego kreatora.

Klikamy przycisk Finish, w wyniku czego nowy projekt zostanie dodany do listy projektów w oknie Eclipse Package Explorer, które domyślnie powinno znajdować się po lewej stronie okna IDE.

Projekt GDA został już dodany do środowiska Eclipse i można obejrzeć znajdujące się w nim pliki. Wystarczy przejść do tego projektu w oknie Eclipse i kliknąć symbol strzałki (>), aby go rozwinąć, jak pokazano na rysunku 1-7.

A co, jeśli komuś nie odpowiada nazwa projektu? Nie ma problemu, wystarczy prawym przyciskiem myszy kliknąć nazwę `piot-java-components`, wybrać opcję Rename (Zmień nazwę), wpisać nową nazwę i kliknąć OK. Jednak należy mieć na uwadze, że w pozostałej części książki stosowana będzie oryginalna nazwa projektu.



Rysunek 1-7 *Projekt GDA dodany i gotowy do użycia*

Jak widać, projekt zawiera już dwa pliki: jeden z nich to `GatewayDeviceApp` w pakiecie `programmingtheiot.gda.app`, a drugi (na najwyższym poziomie) nosi nazwę *pom.xml*. `GatewayDeviceApp` to początkowa nazwa robocza, więc można ją dostosować. Jednak zalecamy zachowanie niezmienionej konwencji nazw, ponieważ *pom.xml* wymaga jej do kompilowania kodu, testowania i tworzenia pakietów. Czytelnicy, którzy dobrze znają narzędzie Maven (*https://maven.apache.org*), mogą wprowadzać dowolne zmiany.

Oto kilka informacji dla czytelników mniej zaznajomionych z narzędziem Maven: *pom.xml* to podstawowy plik konfiguracyjny tego narzędzia, który zawiera instrukcje dotyczące ładowania zależności, ich odpowiednich wersji, konwencje nazewnictwa dla aplikacji, instrukcje kompilowania i oczywiście instrukcje budowania. Większość zależności jest już dołączona, choć można dodać inne, jeśli pojawi się taka potrzeba. Warto również zauważyć, że Maven ma własną domyślną strukturę katalogów, która została utrzymana w repozytorium Java. Aby dowiedzieć się więcej o tych i innych funkcjach narzędzia Maven, można obejrzeć pięciominutowy samouczek (*https://oreil.ly/oLetb*).

A teraz, aby upewnić się, że wszystko zostało przygotowane i można zbudować, spakować i uruchomić aplikację GDA, należy podjąć następujące kroki:

1. Sprawdź, czy stacja robocza jest podłączona do Internetu.
2. Zbuduj projekt i utwórz pakiet wykonywalny.
   a. Prawym przyciskiem myszy kliknij projekt `piot-java-components` w Project Workspace, przewiń do „Run As” (Uruchom jako) i kliknij „Maven install”

(ponieważ Maven musi zainstalować wszystkie brakujące zależności określone w pliku *pom.xml*, jego uruchomienie może potrwać nieco dłużej, w zależności od szybkości łącza internetowego i innych czynników).

b. Sprawdź wynik w konsoli na dole ekranu Eclipse IDE. Nie powinny wystąpić żadne błędy, a kilka ostatnich wierszy powinno wyglądać następująco:

```
[INFO] --- maven-install-plugin:2.4:install (default-install) @ gateway-device-
app ---
[INFO] Installing E:\aking\workspace\gda\java-components\target\ gateway-device-
app-0.0.1.jar to C:\Users\aking\.m2\repository\ programmingtheiot\gda\gateway-
device-app\0.0.1\
gateway-device-app-0.0.1.jar
[INFO] Installing E:\aking\workspace\gda\java-components\pom.xml to C:\Users\
aking\.m2\repository\programmingtheiot\gda\gateway-device- app\0.0.1\gateway-
device-app-0.0.1.pom
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 4.525 s
[INFO] Finished at: 2020 07-04T14:31:45-04:00
[INFO] ------------------------------------------------------------
```

3. Uruchom swoją aplikację GDA w środowisku Eclipse.

a. Prawym przyciskiem myszy ponownie kliknij projekt `java-components` i przewiń w dół do opcji „Run As” i tym razem kliknij „Java application”.

b. Sprawdź wynik w konsoli na dole ekranu Eclipse IDE. Jak w przypadku kompilacji Maven, nie powinny wystąpić żadne błędy, a wynik powinien wyglądać podobnie do następującego:

```
Jul 04, 2020 3:10:49 PM programmingtheiot.gda.app.GatewayDeviceApp initConfig
INFO: Attempting to load configuration.
Jul 04, 2020 3:10:49 PM programmingtheiot.gda.app.GatewayDeviceApp startApp
INFO: Starting GDA...
Jul 04, 2020 3:10:49 PM programmingtheiot.gda.app.GatewayDeviceApp startApp
INFO: GDA ran successfully.
```

Teraz jesteśmy już gotowi do pisania własnego kodu w aplikacji GDA. Pora zająć się przygotowaniem stacji roboczej dla aplikacji CDA.

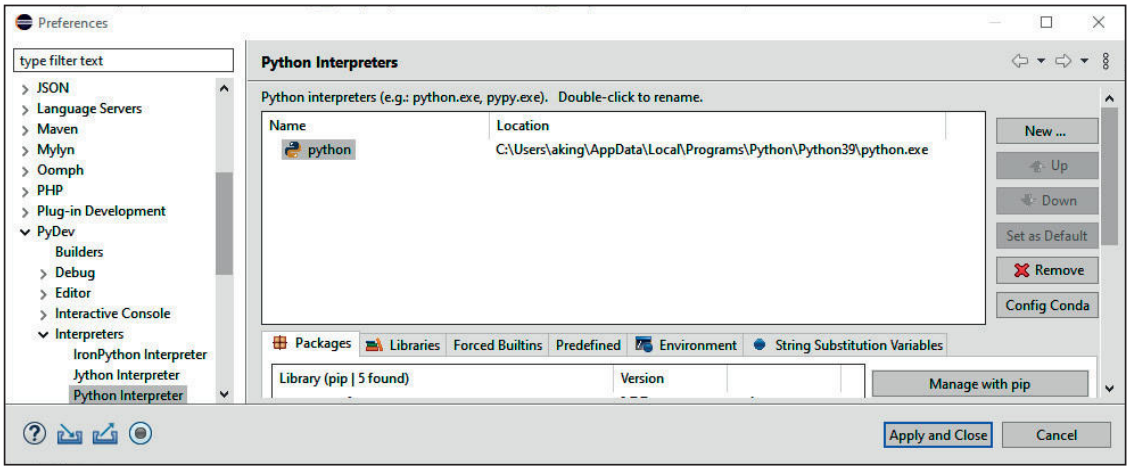
Przygotowanie projektu Constrained Device Application

Ten proces będzie przebiegał podobnie do procesu przygotowywania projektu GDA, ale wymaga dodania wtyczki PyDev do środowiska Eclipse. Oto podsumowanie czynności, które musimy wykonać.

Włączamy środowisko Eclipse, jeśli nie jest ono jeszcze włączone. W osobnym oknie otwieramy przeglądarkę sieci Web i wchodzimy na stronę pobierania PyDev – Python

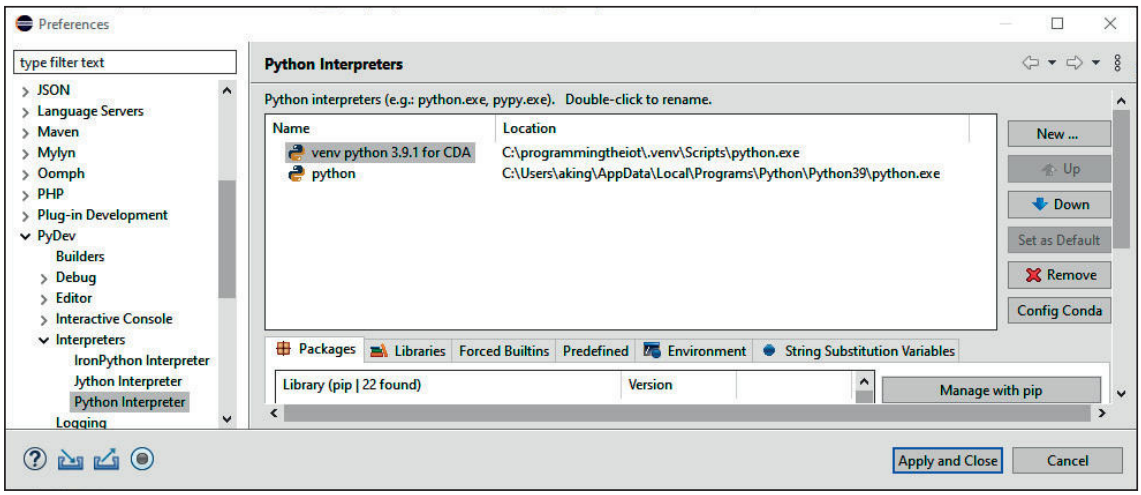
IDE for Eclipse (*https://oreil.ly/4Je1X*), przeciągamy ikonę PyDev „Install” ze strony internetowej i upuszczamy ją w górnej części okna Eclipse IDE (pojawi się zielona ikona „plusa”, która wskazuje, że można upuścić ją na tym obszarze). Następnie program Eclipse automatycznie zainstaluje wtyczkę PyDev i jej zależności.

Po zainstalowaniu wtyczki PyDev możemy przejść do interpretatora języka Python, aby użyć środowiska venv (lub virtualenv), jeśli zostało ono utworzone we wcześniejszej części rozdziału. Po wybraniu opcji Preferences → PyDev → Interpreters → „Python Interpreter” Eclipse wyświetli okno dialogowe podobne do tego zaprezentowanego na rysunku 1-8.



Rysunek 1-8 *Dodawanie nowego interpretera języka Python*

Następnie dodajemy nowy interpreter przez wybranie opcji „Browse for python/pypy.exe” (Przeglądaj w poszukiwaniu python/pypy.exe) i wpisujemy odpowiednie informacje w kolejnym okienku, jakie się pojawi. Później zaznaczamy interpreter venv (lub virtualenv) i klikamy przycisk Up (W górę), aż pojawi się on na pierwszej pozycji listy. W rezultacie venv (lub virtualenv) stanie się domyślnym interpreterem języka Python, jak widać na rysunku 1-9. Klikamy „Apply and Close” (Zastosuj i zamknij).



Rysunek 1-9 *Interpreter virtualenv został ustawiony jako domyślny*

Po ukończeniu tego procesu wybieramy opcje File → Import i importujemy repozytorium Git `python-components`, które zostało wcześniej sklonowane z platformy GitHub. Proces jest prawie taki sam, jak ten przedstawiony na rysunkach 1-5, 1-6 oraz 1-7, z tą różnicą, że zaimportowane zostanie repozytorium Git `python-components` sklonowane z usługi GitHub.

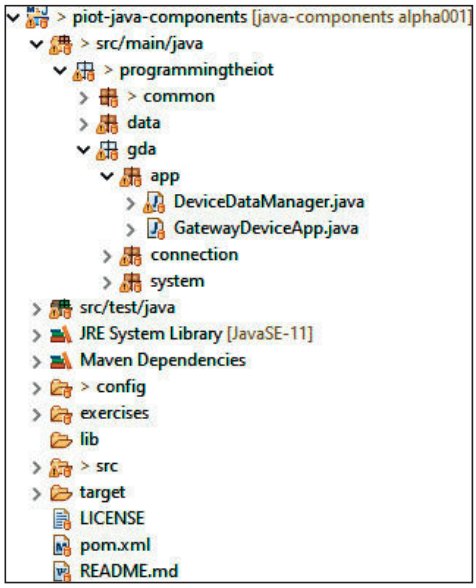
Na naszej stacji roboczej repozytorium do zaimportowania znajduje się pod adresem:

```
C:\programmingtheiot\python-components
```

Jak w przypadku projektu GDA, repozytorium może mieć inną nazwę, dlatego należy pamiętać o użyciu odpowiedniej ścieżki. Do repozytorium dołączony został także plik Eclipse `.project`, a więc można zaimportować je jako projekt Eclipse. Domyślnie ustawiony zostanie język Python, dlatego zastosowany zostanie szablon projektu PyDev. Jak poprzednio, można przeprowadzić operację importowania w dowolny sposób, ale zaleca się powtórzenie procesu zastosowanego dla aplikacji GDA.

Po ukończeniu procesu importu w oknie Package Explorer pojawi się nowy projekt o nazwie `piot-python-components`. Teraz Eclipse IDE zawiera przygotowane do użycia komponenty CDA.

Aby wyświetlić dostępne pliki, przechodzimy do elementu `piot-python-components` i klikamy znak strzałki (>), aby dodatkowo go rozwinąć, jak pokazano na rysunku 1-10.

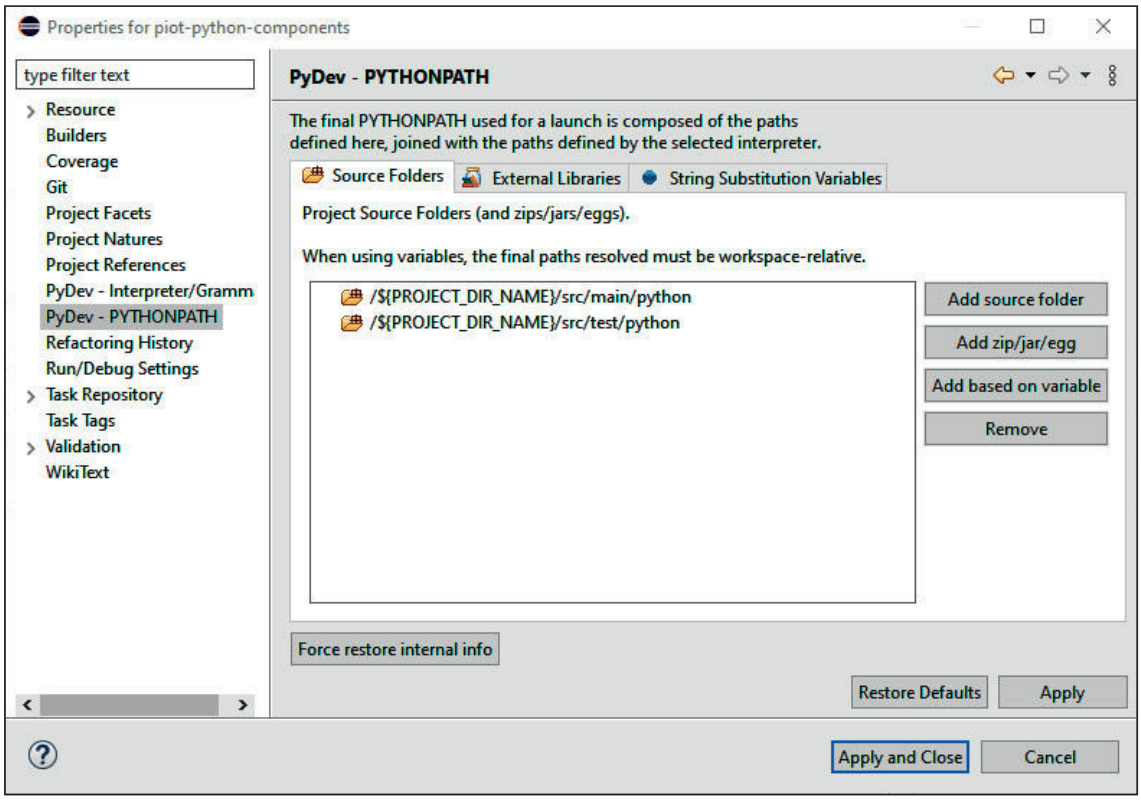


Rysunek 1-10 *Projekt CDA dodany i gotowy do użycia*

Jak widać, projekt zawiera już wiele plików Python. Jednym z nich jest plik `ConstrainedDeviceApp.py` w pakiecie `programmingtheiot.cda.app`, który jest wrapperem aplikacji CDA. Ponadto wszystkie pakiety zawierają pliki *init.py* – są to puste pliki, które interpreter języka Python wykorzystuje do ustalania, w których katalogach szukać plików Python (na razie można je zignorować). Podobnie jak pokazana wcześniej (i napisana w języku Java) przykładowa aplikacja GDA, `ConstrainedDeviceApp` to po prostu szkielet, który pomaga w rozpoczęciu pracy.

Projekt zawiera również dwa pliki .txt: *basic_imports.txt* oraz *cv_imports.txt*. Pierwszy z nich zostanie wykorzystany do zainstalowania zależności bibliotek wymaganych w kolejnych ćwiczeniach programistycznych związanych z CDA. Drugi nie będzie wykorzystywany w żadnych ćwiczeniach w tej książce – został dołączony, aby ułatwić czytelnikom rozpoczęcie pracy z podstawowym systemem rozpoznawania obrazów. Czytelnikom zainteresowanym aplikacjami rozpoznawania obrazów zalecamy samodzielne przestudiowanie dokumentacji biblioteki.

Osoby, które mają doświadczenie w programowaniu w języku Python, prawdopodobnie znają zmienną środowiskową `PYTHONPATH`. Ponieważ chcieliśmy uzyskać podobny schemat pakietów GDA oraz CDA, być może trzeba będzie poinformować PyDev (i środowisko virtualenv), jak przechodzić przez tę strukturę katalogów w celu uruchomienia aplikacji. Aby upewnić się, że folder podrzędny *python* jest wybrany zarówno dla *main*, jak i *test* w `PYTHONPATH`, prawym przyciskiem myszy klikamy „piot-python-components", wybieramy „PyDev – PYTHONPATH", a następnie klikamy „Add source folder" (Dodaj folder źródłowy), jak pokazano na rysunku 1-11. Wybieramy folder *python* w *main* i klikamy Apply. Następnie powtarzamy tę operację dla folderu *python* w *test*. Na zakończenie klikamy przycisk „Apply and Close".



Rysunek 1-11 *Uaktualnianie zmiennej środowiskowej* `PYTHONPATH` *w PyDev i Eclipse*

Pora uruchomić aplikację CDA w środowisku Eclipse.

1. Prawym przyciskiem myszy ponownie kliknij „piot-python-components”, przewiń w dół do „Run As” i tym razem kliknij „Python Run”.
2. Sprawdź komunikaty wyświetlone w konsoli w dolnej części ekranu Eclipse IDE. Jak w przypadku testowego uruchomienia aplikacji GDA, nie powinny wystąpić żadne błędy i wynik powinien być podobny do następującego:

```
2020-07-06 17:15:39,654:INFO:Attempting to load configuration... 2020-07-06
17:15:39,655:INFO:Starting CDA...
2020-07-06 17:15:39,655:INFO:CDA ran successfully.
```

Konfiguracja aplikacji

Po uruchomieniu aplikacji CDA oraz GDA można zauważyć komunikaty związane z konfiguracją. Problem zarządzania konfiguracją został również omówiony we wcześniejszej części rozdziału. Ponieważ każda aplikacja wiąże się z obsługą szeregu różnych parametrów konfiguracyjnych, do każdego repozytorium kodu dodano podstawową klasę narzędziową `ConfigUtil`, która pomaga w realizacji tego zadania.

W projekcie Python `ConfigUtil` deleguje do wbudowanego modułu `configparser` (*https://oreil.ly/4MuGf*), natomiast w projekcie Java `ConfigUtil` deleguje do biblioteki Apache commons-configuration (*https://oreil.ly/ev5un*). Obie umożliwiają załadowanie domyślnego pliku konfiguracyjnego (*./config/PiotConfig.props*) lub dostosowanej wersji. W języku Python można to osiągnąć, przekazując nazwę pliku do konstruktora, a w języku Java ustawiając właściwość systemu. Na razie zalecamy użycie domyślnego pliku konfiguracyjnego, ponieważ będzie on wspierał wszystkie parametry konfiguracji stosowane w poszczególnych ćwiczeniach w tej książce. Można użyć „stałych” zdefiniowanych w klasie `ConfigConst` każdego repozytorium (lub dodać własne).

Format pliku konfiguracyjnego jest taki sam dla projektów CDA oraz GDA. Oto krótki fragment z *PiotConfig.props* aplikacji CDA:

```
[ConstrainedDevice]
deviceLocationID        = constraineddevice001
enableEmulator          = False
enableSenseHAT          = False
enableMqttClient        = True
enableCoapClient        = False
enableLogging           = True
pollCycleSecs           = 60
testGdaDataPath         = /tmp/gda-data
testCdaDataPath         = /tmp/cda-data
```

A oto fragment z *PiotConfig.props* aplikacji GDA:

```
[GatewayDevice]
deviceLocationID        = gatewaydevice001
```

```
enableLogging            = True
pollCycleSecs            = 60
enableMqttClient         = True
enableCoapServer         = False
enableCloudClient        = False
enableSmtpClient         = False
enablePersistenceClient  = False
testGdaDataPath          = /tmp/gda-data
testCdaDataPath          = /tmp/cda-data
```

Warto zauważyć, że sekcje zostały oznaczone słowami kluczowymi zawartymi w nawiasie, natomiast właściwości mają format klucz = wartość. To ułatwia dodawanie nowych sekcji i par klucz/wartość.

Jednym szczególnym przypadkiem uwzględnionym w każdej implementacji `ConfigUtil` jest możliwość definiowania (i ładowania) osobnego pliku konfiguracyjnego, który zawiera dane uwierzytelniające i inne poufne dane, które nie powinny być częścią konfiguracji repozytorium. W każdej sekcji można określić wartość klucza `credFile` mapowanego do lokalnego pliku, który może i powinien znajdować się poza repozytorium.

Analizując plik *PiotConfig.props* dla CDA oraz GDA, można zauważyć, że niektóre sekcje zawierają wpis `credFile`. Służą one do przeniesienia haseł, tokenów uwierzytelniania użytkowników, kluczy API itp. poza główny plik konfiguracyjny. Bardzo ważne jest, by przechowywać tego typu informacje poufne poza repozytorium. Nie należy NIGDY dodawać do repozytorium Git nazw użytkowników, haseł, prywatnych kluczy i innych poufnych danych. Jeśli pojawi się konieczność przechowywania tego typu informacji, warto uważnie przeczytać artykuł „Encrypted Secrets” (*https://oreil.ly/PZye4*), aby dowiedzieć się więcej o tym procesie w usłudze GitHub. Bezpieczne przechowywanie danych uwierzytelniających to ważny temat, jednak wykracza on poza zakres niniejszej książki.

Mechanizm konfiguracji, jaki przedstawiliśmy, jest dość prosty i został zaprojektowany tylko na potrzeby testowania i prototypowania. W częściach III i IV zaprezentujemy pewne dodatkowe techniki dynamicznego ustawiania danych konfiguracji w warstwie brzegowej i w warstwie chmury. Większość doświadczonych programistów opracowała własne strategie i rozwiązania do konfigurowania aplikacji. Nic nie stoi na przeszkodzie, by dostosowywać funkcje prezentowane w tej książce do własnych potrzeb lub zastosować własne.

Na tym etapie aplikacje GDA i CDA są już gotowe i działają w środowisku IDE. Wiemy również, na czym polega mechanizm konfiguracji. Możemy zacząć pisanie kodu w obu aplikacjach!

Jednak przed przejściem do ćwiczeń z rozdziału 2 musimy omówić dwa dodatkowe zagadnienia: testowanie i automatyzację.

Krok II: Definiowanie strategii testowania

Skoro środowisko programistyczne zostało już przygotowane do uruchamiania aplikacji GDA oraz CDA, pora wyjaśnić, w jaki sposób będziemy testować pisany kod. Testowanie jest oczywiście bardzo istotnym aspektem każdego przedsięwzięcia inżynieryjnego i programowanie nie jest tu wyjątkiem. Każda budowana aplikacja powinna zostać dokładnie przetestowana niezależnie od tego, czy działa zupełnie niezależnie od innych aplikacji, czy też jest ściśle zintegrowana z innymi systemami. Co więcej, każda jednostka pisanego kodu powinna być testowana w celu sprawdzenia, czy działa zgodnie z oczekiwaniami. Czym dokładnie jest jednostka? W tej książce jednostka będzie zawsze reprezentowana jako funkcja lub metoda, która ma być przetestowana.

Czym różni się funkcja od metody? W ogromnym uproszczeniu funkcja to nazwany fragment kodu, który realizuje określone zadanie (takie jak dodanie do siebie dwóch liczb) i zwraca wynik. Jeśli funkcja akceptuje dane wejściowe, zostają one przekazane w postaci jednego lub wielu parametrów. Metoda różni się od funkcji tym, że jest dołączona do obiektu. W zorientowanej obiektowo nomenklaturze obiekt jest po prostu klasą, dla której utworzono wystąpienie, a klasa to formalna definicja komponentu – jej metod, parametrów, konstruktora i dekonstruktora. Wszystkie prezentowane w tej książce przykłady w języku Java będą miały postać klas z metodami zdefiniowanymi w ramach tych klas. Kod Python można pisać w formie skryptu lub klas, ale my wolimy pisać klasy Python z metodami i w taki sposób zaimplementujemy wszystkie przykłady kodu Python (z kilkoma wyjątkami).

Testy jednostkowe, integracyjne i wydajnościowe

Istnieje wiele sposobów testowania aplikacji oraz systemów informatycznych i tej tematyce poświęconych zostało wiele doskonałych książek, artykułów oraz blogów. Opracowanie działającego rozwiązania IoT wymaga zwrócenia szczególnej uwagi na testowanie samej aplikacji oraz interakcji z innymi aplikacjami i systemami. W rozwiązaniu budowanym w tej książce skoncentrujemy się na zaledwie trzech rodzajach testów: jednostkowych, integracyjnych oraz wydajnościowych.

Testy jednostkowe to moduły kodu napisane z myślą o testowaniu najmniejszej możliwej *jednostki kodu*, do której test ma dostęp, takiej jak funkcja czy metoda. Te testy służą do weryfikowania, czy zbiór danych wejściowych wybranej funkcji bądź metody prowadzi do uzyskania oczekiwanego wyniku. Często testowane są również warunki brzegowe w celu sprawdzenia, czy funkcja bądź metoda odpowiednio radzi sobie z takimi warunkami.