POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Bazy Danych i Inżynieria Oprogramowania

Wojciech Wróblewski
Nr albumu 1323

UNIWERSALNY SYSTEM INFORMATYCZNY W GOSPODARSTWIE DOMOWYM WSPOMAGAJĄCY ZARZĄDZANIE ZASOBAMI ŻYWNOŚCIOWYMI

Praca magisterska napisana
pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, marzec 2018

Pracę dedykuję Rodzicom,
bez których nie byłbym w miejscu,
do którego doszedłem.

STRESZCZENIE

W niniejszej pracy omówiono zagadnienia informatyzacji wybranej sfery życia związanej z zasobami żywnościowymi. Przedstawione w niej dotychczasowe rozwiązania nie doczekały się wytworzenia wbudowanego, kompleksowego produktu dostępnego powszechnie na rynku. W ostatnich latach prototypy tego rodzaju rozwiązań były wielokrotnie prezentowane przez światowe koncerny. Jednocześnie propozycje dostępnych rozwiązań aplikacyjnych nie są na tyle kompleksowe, aby znacząco zmieniły współczesny styl życia.

Autor poddaje analizie powody niepowodzenia proponowanych rozwiązań oraz przedstawia ich krótką charakterystykę. Brak pomyślnych rezultatów w upowszechnieniu się ich użycia w codziennym życiu skłonił Autora do zaprezentowania własnej koncepcji rozwiązania. W założeniach miałaby na celu poprawę tego stanu rzeczy. Na jej podstawie stworzono i omówiono prototyp systemu podejmujący tematykę pracy. Rozwiązania związane z rozszerzalnością, które zaimplementowano w prototypie, pozwoliły na rozszerzenie jego wbudowanej funkcjonalności, bez konieczności ingerencji w jego strukturę. Proponowane rozwiązanie opisano w zakresie umożliwiającym zapoznanie się z jego szczegółową koncepcją. Omówiono również proces towarzyszący jego implementacji wraz z ciekawszymi przykładami. Na zakończenie przedstawiono rezultaty stworzonego prototypu oraz poddano dyskusji ewentualny kierunek jego rozwoju.

SPIS TREŚCI

WPROWADZENIE

Żyjemy w świecie, który rozwija się nieustannie z coraz to większą prędkością. Pojawianie się nowych wynalazków technologicznych inspiruje do kreowania nowych rozwiązań. Zauważalne jest to w kolejnych dziedzinach współczesnego życia. W ekspansji rozwiązań technologicznych niewątpliwie kluczową rolę odgrywa ich informatyzacja. Nieustannie poszukiwane są nowe obszary jej zastosowania. Ma to na celu usprawnienie, ale również poprawę funkcjonowania określonych sfer współczesnego życia. Z każdym rokiem wprowadzane są na rynek coraz to nowsze i bardziej zaawansowane rozwiązania. Wypełniają sukcesywnie dotychczas istniejące luki lub niedostatecznie rozwiniętą informatyzację tych obszarów. Starają się tym samym wyjść naprzeciw stale rosnącym potrzebom oraz coraz to wyżej postawionym wymaganiom.

Jedną z jeszcze nie do końca zagospodarowanych przez informatyzację gałęzi przemysłu jest segment AGD. Działanie należących do niego urządzeń oparte jest na elektronicznych układach sterujących. Przy pomocy odpowiednich algorytmów funkcjonowania realizują pracę tych urządzeń. Jednak do dnia dzisiejszego nie zdobyły miana produktu pracującego pod kontrolą systemu informatycznego. Z tego powodu nigdy nie miały połączenia z informatycznym światem zewnętrznym. Wpływa to na sposób ich obsługi ograniczony do wbudowanych, hermetycznych rozwiązań. Na chwilę obecną mimo dużego zaawansowania technologicznego rozwiązania te zaczynają być stosunkowo mało elastyczne. Postęp informatyczny obserwowany na przykładzie innych urządzeń wymusza taki obraz sytuacji. Główną przyczyną postrzegania ich w ten sposób jest wrodzona ludzka łatwość przyzwyczajania się do nieskomplikowanych, intuicyjnych rozwiązań. Niemałe znaczenie ma także

możliwość posiadania dostępu do szeroko pojmowanej komunikacji. Do tego rodzaju rozwiązań przyzwyczajają inne oferowane na rynku urządzenia z różnych sektorów przemysłu. W naturalny sposób szukamy rozwiązań podobnych w miejscach, w których jeszcze nie funkcjonują. Niewątpliwie w obszarach tych widzielibyśmy ich olbrzymią przydatność. Decyzja o konieczności ich posiadania i zainteresowania się nimi spowodowana jest coraz to bardziej zauważalnym brakiem możliwości do nich dostępu. Przekłada się to w znaczący sposób na zainteresowanie koncernów do ich szybszego stworzenia.

Pomimo dość szybkiego rozwoju technologii informatycznych, chociażby w sektorze RTV, sektor AGD nie wypracował jeszcze podobnych standardów. Pozostaje on nadal w fazie wczesnego rozwoju opracowywania koncepcji. W wyniku czego koncerny pracujące nad tymi rozwiązaniami skupione są w większym stopniu na samej strategii marketingowej. Ma ona na celu budowanie marki w świadomości odbiorcy jako firmy posiadającej innowacyjne rozwiązania. Nie przekłada się to jednak znacząco na wytworzenie finalnego produktu. Między innymi dlatego od kilku lat prezentują swoje bardziej lub mniej użyteczne rozwiązania na licznych targach wystawienniczych. Wystawiają się na pokazach poświęconych nowym technologiom, ale także i na ściśle związanych z ich branżą. Pomimo dużej aktywności w prezentowaniu swoich wizji produktów nie podejmują się ich masowej produkcji. Stoi za tym określona ilość powodów, które omawiane są w niniejszej pracy.

Przemysł spożywczy, ze względu na swoją specyficzną funkcję, jest jednym z obszarów sektora AGD, który w naturalny sposób podlegałby informatyzacji. Typowymi urządzeniami z tego sektora są urządzenia chłodnicze zwane potocznie lodówkami. Mamy z nimi do czynienia na co dzień w gospodarstwach domowych. W powszechnym użytkowaniu są dostępne od kilku dziesięcioleci. Podstawową i właściwie jedyną funkcją lodówek jest ochrona żywności. Dokładniej to ujmując, ich zadanie polega na wydłużaniu okresu przydatności do spożycia zgromadzonych w nich produktów spożywczych. Rozwój technologii informatycznych w innych branżach spowodował, że zaczęto zastanawiać się, w jaki sposób można byłoby wykorzystać ją w tego typu urządzeniach.

Niniejsza praca nie skupia się jedynie na zarządzaniu zasobami zgromadzonymi w urządzeniach typu chłodziarka. Podchodzi do tematyki w sposób bardziej kompleksowy. Z tego powodu prezentowane rozwiązanie nie jest powiązane z jakimkolwiek urządzeniem z sektora AGD. Przedstawiana propozycja jest zaoferowaniem rozwiązania bardziej uniwersalnego, takiego które miałoby wpływ na zmianę współczesnego stylu życia.

Rozdział I.

WSTĘP W TEMATYKĘ PRACY

W rozdziale tym krótko przedstawiono wstępne informacje dotyczące zawartości niniejszej pracy. Rozwinięcie rozpoczętej w nim tematyki następuje w kolejnych rozdziałach.

1.1. Cel pracy

Podstawowym celem pracy jest opracowanie koncepcji systemu wspomagającego końcowego odbiorcę w procesie zarządzania artykułami spożywczymi. Praca ma także na celu dostarczenie mu możliwości na automatyzację tego procesu. System powinien zapewniać stosunkowo prosty sposób wdrożenia w każdym gospodarstwie domowym. Nie powinien wymuszać ponoszenia dodatkowych kosztów związanych m.in. z wymianą urządzeń chłodniczych. Istotną cechą systemu jest oparcie jego działania na powszechnie dostępnych urządzeniach typu smartfon oraz tablet.

Na podstawie proponowanej koncepcji zrealizowano prototyp w postaci działającego systemu informatycznego. System w zrealizowanym kształcie ma głównie na celu zobrazowanie sposobu jego funkcjonowania. Odgrywa także rolę zapoznawczą, dając możliwość wyrobienia opinii na temat użyteczności tego rodzaju rozwiązań.

1.2. Zarys koncepcji

Funkcjonowanie proponowanego systemu podzielone jest na dwa obszary. Obsługa modułu klienckiego przewidziana jest na urządzenia mobilne. Serwer systemu umieszczony

jest w chmurze publicznej znajdującej się w Internecie. W jego gestii jest zarówno przechowywanie danych użytkowników, jak i współpraca z różnymi serwisami zewnętrznymi. Przykładowo mogą to być sklepy internetowe, serwisy identyfikujące produkty lub chociażby serwisy poświęcone tematyce przepisów kulinarnych. Użytkownik końcowy posiada możliwość użytkowania wspomnianych serwisów przy pomocy proponowanego systemu. Dostęp do funkcjonalności serwisów zapewniony jest dzięki wykorzystaniu technologii rozszerzalności. System uzyskuje ją poprzez użycie wbudowanych mechanizmów wtyczek.

1.3. Przyjęte rozwiązania

Niniejsza praca opiera się na wykorzystaniu technologii systemów rozproszonych. Zorientowana jest na wykorzystanie usług internetowych dostępnych globalnie. Na potrzeby pracy użyto platformę chmurową Microsoft Azure. Do obsługi systemu wykorzystano powszechnie dostępne urządzenia mobilne w postaci smartfonów i tabletów. Przyjęto współpracę systemu z urządzeniami pracującymi pod kontrolą systemu operacyjnego Android. Wybrana platforma Azure łączy wewnętrzną komunikację pomiędzy tymi urządzeniami, dostarczając rozwiązanie efektywnej wymiany informacji.

Do identyfikowania produktów spożywczych system wykorzystuje wbudowane możliwości urządzeń mobilnych. Posłużyły w tym celu istniejące w nich cyfrowe aparaty fotograficzne. Identyfikacja produktów oparta jest na kodach kreskowych znajdujących się na większości dostępnych artykułów spożywczych.

Istotnym elementem systemu jest także wykorzystanie rozwiązań generycznych oraz takich, które zapewniają obsługę wtyczek. Umożliwia to poszerzanie jego funkcjonalności, bez konieczności przeprowadzania zmian w jego strukturze.

1.4. Osiągnięte rezultaty

Docelowym rezultatem pracy jest opracowanie alternatywnej koncepcji budowy rozszerzonego systemu wspomagającego automatyzację procesów związanych z zarządzaniem produktami żywnościowymi. Osiągnięty rezultat miał w założeniu oferować bardziej uniwersalne podejście do zagadnienia.

Na podstawie wypracowanej koncepcji powstał prototyp systemu, który pracuje pod kontrolą środowiska powszechnie dostępnego w urządzeniach mobilnych. Rozwiązanie to

spełnia ustalone założenia dotyczące wypracowania rozszerzonego, w pełni niezależnego środowiska jego pracy.

Jest to odmienne podejście, niż w typowych urządzeniach służących do przechowywania żywności, które bazują tylko na wbudowanych rozwiązaniach. Jest to także inne podejście, niż w aplikacjach skupiających się na idei samego ewidencjonowania żywności. Wypracowane rozwiązanie przedstawia tematykę zagadnienia w szerszym kontekście.

1.5. Organizacja pracy

W rozdziale drugim, przedstawiono przegląd koncepcji wbudowanych rozwiązań prezentowanych przez producentów sprzętu AGD. Poruszono także temat dostępności istniejących rozwiązań aplikacyjnych. W podsumowaniu rozdziału dokonano ich krótkiego omówienia, analizując zalety i wady.

W rozdziale trzecim, omówiono założenia oraz koncepcję dla proponowanego rozwiązania. Przedstawiono także czynniki motywujące do poruszenia zagadnień rozpatrywanych w niniejszej pracy.

W rozdziale czwartym, wymieniono wraz z opisem technologie, które posłużyły do realizacji prototypu proponowanego rozwiązania.

W rozdziale piątym, opisano sposób implementacji poszczególnych obszarów zrealizowanego prototypu.

W rozdziale szóstym, kończącym pracę, przedyskutowano wady i zalety proponowanego rozwiązania. Rozpatrzono także kierunki ewentualnego rozwoju, kończąc rozdział przedstawieniem wniosków końcowych.

Rozdział II.

ANALIZA ISTNIEJĄCYCH ROZWIĄZAŃ

Po analizie większej ilości dostępnych publikacji związanych z tematyką pracy i przedstawieniu rezultatów przeprowadzonego badania rynku, może nasuwać się jeden wniosek. W chwili obecnej wbudowane rozwiązania nie pojawiły się jeszcze w powszechnym użytkowaniu. Istnieją za to jednostkowe wykonania. Informacje na ich temat można jednak znaleźć prędzej na stronach producenta niż w ofercie sklepów z artykułami AGD. Jednocześnie rozwiązania aplikacyjne nie posiadają zbytnio rozbudowanej funkcjonalności. Nie pozwala to na nadanie tym rozwiązaniom miana — znacząco zmieniających sposób postępowania w zarządzaniu żywnością.

W poniższych podrozdziałach wykonano przegląd aktualnego stanu dla badanej problematyki. Omówiono także wątpliwości co do proponowanych rozwiązań oraz nakreślono sposób poprawy tego stanu rzeczy.

2.1. Motywacje do rozwoju

Coraz to nowsze oferowane przez producentów technologie i rozwiązania, wkraczają w do tej pory niezagospodarowane obszary współczesnego życia. Mając z nimi styczność każdego dnia, przyzwyczajamy się do nich. Z czasem nie wyobrażamy sobie życia bez ich istnienia. Uzależnienie to jest naturalną właściwością człowieka.

Miniaturyzacja kolejnych rozwiązań powoduje większe możliwości ich wykorzystania w szerszym zakresie niż dotychczas. Informatyzacja rozwiązań jest naturalnym kierunkiem

rozwoju. To właśnie systemy informatyczne sprawiają, że obsługa ich staje się prostsza. Niemały wpływ na to mają coraz to bardziej wyszukane algorytmy pracy. Tym samym rozwiązania te wkraczają w fazę rozwoju, kiedy nie tylko służą pomocą, ale i zastępują w wykonywaniu powtarzalnych czynności.

Z jednej strony obawiamy się wkraczania w życie nowych rozwiązań, nie chcąc w ten sposób naruszać własnoręcznie zbudowanego, dobrze funkcjonującego świata. Z drugiej jednak strony jak tylko przełamiemy barierę i zaczniemy się do nich przekonywać, to w niedługim okresie czasu uzależniamy się od nich. Jak już wcześniej wspomniano, moment braku możliwości ich użytkowania jest szybko odczuwalny. Skutkuje to tym, że te same czynności, które były wielokrotnie wykonywane do tej pory bez większego trudu, stają się dość skomplikowane w realizacji.

Niewątpliwie szybko przyzwyczajamy się do nowych rozwiązań. Tym bardziej do takich, które zdejmują ciężar wykonywania żmudnych, niekiedy powtarzalnych czynności. Powstają dla wygody i ułatwienia codziennego funkcjonowania. Innowacyjność rozwiązań to jedynie kwestia kreatywności zespołów pracujących nad ich rozwojem. To są jedne z głównych powodów ewolucji rozwiązań, również tych omawianych w niniejszej pracy.

2.2. Koncepcje i trendy w urządzeniach

W kreowaniu rozwiązań wbudowanych w urządzenia przemysłowe główną rolę odgrywają światowe koncerny. Istnieją na rynku od bardzo wielu lat. Posiadają w swojej ofercie szeroki asortyment produktów. To w naturalny sposób stwarza im możliwość do rozwoju informatycznej, bardziej kompleksowej funkcjonalności. Wpływa to na zmianę sposobu obsługi urządzeń, z prostego sterowania na bazujące na systemach informatycznych. Tego rodzaju rozwiązania istnieją właściwie tylko jako element promocyjny. Prezentuje się je na specjalistycznych targach poświęconych takiej tematyce. Prezentowanie ich wyłącznie na tego rodzaju pokazach spowodowane jest nie tyle że wczesną fazą rozwoju takich rozwiązań, ale czasem potrzebnym na ich wdrożenie. Do dnia dzisiejszego urządzenia te praktycznie nie są powszechne w użyciu, a jeśli nawet są dostępne, to cena ich jest dość wysoka.

W przygotowanym poniżej zestawieniu przedstawiono przegląd trendów w rozwiązaniach na przestrzeni ostatnich kilku lat. Większość z prezentowanych produktów nadal nie jest dostępna w sprzedaży. Pozostałe nie do końca realizują w pełni „inteligentną" funkcjonalność. Uzmysławia to, że realizacja finalnego produktu godnego miana „inteligentnej lodówki" oraz upowszechnienie się takiego rozwiązania, nie jest sprawą prostą.

2.2.1. Wbudowana „inteligencja”

Można zgodzić się z producentami, że inteligentne lodówki to przyszłość. Podkreśla to artykuł [1], w którym rozważany jest ich rozwój. „Inteligentną” funkcjonalność lodówki producenci pojmują jednak w dwojaki sposób.

Pierwszy z nich dotyczy metod jej pracy. Jedną z takich funkcji jest samo diagnozowanie się. W przypadku wykrycia problemów lodówka automatycznie powiadomi serwis i prześle opis objawów. „Inteligentna” lodówka potrafi także tak cyrkulować powietrze, że rozkład temperatur w każdym jej miejscu jest jednakowy. Funkcja ta jest użyteczna, gdyż nie wymaga od użytkownika odpowiedniego rozmieszczania produktów na półkach. Kolejną formą „inteligencji” jest odpowiednie sterowanie jej pracą, tak aby nie dopuścić do powstawania szronu w chłodziarce czy zamrażarce. Nie trzeba się przez to martwić, że cyklicznie musimy rozmrażać produkty. Oprócz tego w takich lodówkach możemy sprawdzić, co znajduje się w środku, bez konieczności jej otwierania. Taką funkcjonalność oferują przeźroczyste drzwi, które mogą spełniać dodatkową rolę sterowania dotykowego.

Drugi rodzaj „inteligencji” lodówek dotyczy bezpośrednio ich informatyzacji. Lodówka jest w stanie sama sprawdzić, co ma w środku przy pomocy wbudowanych kamer. Podpowie, jakie potrawy można przygotować z posiadanych produktów. Osobom, dla których ważne są kalorie, dodatkowo przeliczy ich ilość. Potrafi także powiadomić o kończących się zapasach lub o zbliżającej się dacie zdatności do spożycia produktów. Na końcu „inteligentna” lodówka zrobi zakupy. Niektóre, a właściwie większość z tych funkcjonalności są w fazie koncepcji. Pozostałe miały już swoją premierę w pierwszych komercyjnych egzemplarzach. Na tym rodzaju „inteligencji” skupia się sporządzona analiza rozwiązań zawarta w niniejszym rozdziale.

2.2.2. Liebherr SmartDeviceBox

W obszarze sprzętu AGD postanowiła zaistnieć także firma Microsoft. Wspólnie ze szwajcarskim producentem sprzętu AGD, firmą Liebherr, wdraża autorski projekt „inteligentnej” lodówki [2]. To kolejny raz, kiedy Microsoft chciałby wypracować sobie pozycję lidera w świecie Internetu Rzeczy[1] [3]. Całość proponowanego rozwiązania pracuje w module o nazwie SmartDeviceBox (pol. *Inteligentne urządzenie*), stworzonym przez firmę

[1] Internet Rzeczy (ang. *Internet of Things*, IoT) jest to zbiór urządzeń codziennego użytku, komunikujących się ze sobą poprzez globalną sieć Internet. Urządzenia te posiadają wbudowane czujniki i odpowiednie algorytmy pracy, zapewniające im „inteligencję” oraz możliwość zdalnego zarządzania i sterowania.

Liebherr. Daje to gwarancję, że przez kolejne lata lodówka będzie nadal tak innowacyjna, jak w trakcie zakupu. Odpowiada za to zrealizowany „sprytny" moduł uaktualnień.

W zamierzeniach producenta zadaniem tego modułu było pomaganie w zakupach, wspieranie w planowaniu posiłków oraz inteligentne zarządzanie żywnością. Przy jego pomocy można nie tylko monitorować przechowywane produkty, ale i rozpoznawać je z użyciem wbudowanych kamer. Pozwala to na automatyczną ewidencję zawartości lodówki. System rozpoznawania treści wizualnych jest autorską technologią Microsoftu [4]. Współpracuje z interfejsem API Microsoft Cognitive Services Computer Vision [5]. W przypadku nierozpoznania określonego produktu system prosi użytkownika o pomoc w identyfikacji.

Rysunek II-1. Automatyczne rozpoznawanie produktów w lodówce

Źródło: [2]

Informacje o zidentyfikowanych produktach przechowywane są w chmurze publicznej. Wizja mechanizmu rozpoznawania żywności stworzona przez naukowców Microsoftu nie sprowadza się tylko do ich identyfikacji z bazy danych. Posiada możliwość uzupełniania jej na bieżąco wykonywanymi bezpośrednio przez użytkownika zdjęciami produktów. W ten sposób wszystkie lodówki wyposażone w tak zaawansowane mechanizmy uczą się od siebie rozpoznawania produktów.

Realizacja funkcjonalności zawartych w module SmartDeviceBox wykonywana jest przy wsparciu inteligentnej asystentki głosowej Cortana. Daje to możliwość bezpośredniej

interakcji z urządzeniem i szybszego przekazywania poleceń. Moduł przeznaczony jest na większość dostępnych platform mobilnych takich jak Android, iOS oraz UWP. Opisywane rozwiązanie zaprezentowano na rys. II-1.

2.2.3. Samsung model T9000

Bardzo dobrym przykładem na powolny proces wdrożeń nowych rozwiązań jest powszechnie dostępny model lodówki T9000 firmy Samsung. Wersję z wbudowanym komputerem pokładowym zaprezentowano po raz pierwszy w 2013 r. na międzynarodowych targach CES[2]. Prezentowany egzemplarz wyposażony był w jednostkę sterującą w postaci minikomputera z ekranem dotykowym. Wielkość przekątnej wynosiła 10,1 cala [6]. Systemem operacyjnym obsługującym urządzenie był Android w wersji „Jelly Bean" 4.1. Zaprezentowane nowatorskie rozwiązanie przedstawiono na rys. II-2.

Rysunek II-2. Nowatorskie rozwiązanie „inteligentnej" lodówki

Źródło: [7]

Z ówczesnych informacji przekazanych przez producenta [7], można było dowiedzieć się, że model ten trafi do powszechnej sprzedaży w przeciągu kilku miesięcy. Od tamtego momentu minęło kilka lata i do dnia dzisiejszego model z prezentowanym inteligentnym

[2] CES (ang. *Consumer Electronics Show*) są to międzynarodowe targi poświęcone elektronice oraz nowym technologiom, odbywające się raz do roku w styczniu w Las Vegas, w Stanach Zjednoczonych.

wykonaniem nie wszedł do cyklicznej produkcji. Można było się o tym przekonać, odwiedzając witrynę internetową producenta, poświęconą ofercie na tenże model lodówki [8].

Rysunek II-3. Rozwiązanie sterowania bez użycia panelu dotykowego

Źródło: [8]

Omawiany model oferowany na początku 2015 r., możliwy był do nabycia tylko w wykonaniu ze zwykłym panelem sterującym. Model przedstawiono na rys. II-3.

2.2.4. LG Smart InstaView Door-in-Door Fridge

Na targach technologicznych IFA[3] w 2016 r. koncern LG zaprezentował rozwiązanie z tabletem o imponującej przekątnej 21,5 cala. Donoszą o tym serwisy poświęcone nowoczesnym technologiom [9], [10] oraz [11]. Urządzenie pracuje pod kontrolą systemu operacyjnego Windows 10. Zamysłem producenta było wykorzystanie tabletu do różnych domowych celów. Surfowania po Internecie, pozostawiania wirtualnych notatek, oznaczania swoich produktów w środku, jak również sporządzania listy zakupów. Obecność systemu operacyjnego firmy Microsoft rozbudowuje rozwiązanie o możliwość użycia interaktywnej asystentki głosowej Cortana. Mechanizmy interakcji z użytkownikiem nie zostały jednak jeszcze zaimplementowane w pełnym zakresie.

Trudno jednak przypisywać wymienione funkcjonalności do terminu „inteligentna" lodówka. Przypomina to trochę sytuację z początku bieżącego wieku, kiedy pojawiły się

[3] IFA (ang. *International Fair of Broadcasting Services*) są to międzynarodowe targi poświęcone elektronice, które odbywają się corocznie we wrześniu w Berlinie, w Niemczech.

telefony komórkowe określane mianem telefonów dotykowych. Użytkowanie ich poprzez dotyk polegało na tym, że dotykało się przycisków płaskich, zamiast wciskania przycisków wypukłych. Nie miało to jednak żadnego związku z obsługą dotykową ekranu, jaką dobrze znamy z dzisiejszych czasów.

Rysunek II-4. Lodówka z przeźroczystym 21,5 calowym ekranem dotykowym

Źródło: [9]

Z nieoficjalnych informacji przekazanych przez producenta dowiadujemy się, że lodówka będzie docelowo pracować pod autorskim systemem operacyjnym webOS. Na tę zapowiedź powołuje się artykuł [11]. Podjęcie przez producenta takiej decyzji wpływa jednak negatywnie na standaryzację tego rodzaju rozwiązań. Jest to także jedynie półśrodek do realizacji głównego celu, jakim jest zaprojektowanie inteligentnej lodówki. Opisywane rozwiązanie zaprezentowano na rys. II-4.

Omawiany produkt mógłby być przykładem na to, iż realizacja naprawdę inteligentnej lodówki to bardzo powolny proces. Proces ten rozpoczął się na początku trwającej dekady. W przypadku tego urządzenia producent pojmuje określenie „inteligentna" zupełnie inaczej, niż zdążyliśmy już sobie stworzyć na ten temat wyobrażenie. Pomimo upływu czasu wydaje się, że „inteligentne" rozwiązania pozostają ciągle w tej samej fazie koncepcji. Zmienia się tylko forma ich wizualnej prezentacji. Można dojść do takiego wniosku patrząc na kolejne prototypy tego urządzenia prezentowane przez ostatnie lata.

2.2.5. Smarter FridgeCam

Oprócz rozwiązań wbudowanych bezpośrednio w lodówki, powstają także urządzenia od nich niezależne. Dzięki nim jesteśmy w stanie spowodować, że posiadany aktualnie sprzęt chłodniczy, stanie się bardziej „inteligentny". Sformułowanie to w ich przypadku nie jest jednak do końca trafne.

Rysunek II-5. Monitorowanie lodówki przy pomocy „inteligentnej" kamery

Źródło: [12]

FridgeCam (pol. *Lodówkowa kamera*) to nic innego jak kamera montowana na wewnętrznych drzwiach lodówki [13]. Jej zadaniem jest wykonywanie zdjęć zawartości lodówki po zamknięciu drzwi. Podgląd aktualnego stanu można przejrzeć na ekranie smartfona. Takie podejście do tematu wymusza na użytkowniku odpowiednie układanie produktów, tak aby nie zasłaniały się nawzajem. Urządzenie to nie rozpoznaje produktów. Podczas robienia zakupów może posłużyć do przypomnienia, co znajduje się w lodówce. Wygląd kamery zaprezentowano na rys. II-5.

2.3. Dostępne rozwiązania aplikacyjne

Oprócz rozwiązań wbudowanych w urządzenia chłodnicze, powstała także większa liczba aplikacji wspomagających zarządzanie zapasami żywności. Żadne z proponowanych rozwiązań nie udostępnia jednak bardziej rozbudowanej funkcjonalności. Skupiają się głównie na ewidencjonowaniu żywności, tworzeniu list zakupowych lub ewentualnie sugerowaniu przepisów kulinarnych. Nie wykorzystują także globalnych baz danych do przechowywania informacji o zawartości lodówki. W przygotowanym poniżej zestawieniu opisano kilka najbardziej popularnych aplikacji tego typu. Każda z przedstawianych aplikacji posiada inny zestaw funkcjonalności.

2.3.1. Fresh Box

Rozwiązanie to jest najprostszym ze wszystkich opisywanych. Fresh Box (pol. *Świeże pudełko*) działa w myśl sformułowania „to, co widzisz, jest tym, co posiadasz" [14]. Określenie to bierze się ze sposobu, w jaki dostarcza się aplikacji informacji o przechowywanych w lodówce produktach. Dzieje się to poprzez wykonywanie zdjęć produktów wstawianych do lodówki. W momencie dodawania produktów można określić datę ich przydatności do spożycia. Informacja ta jest potem wykorzystywana do informowania użytkownika o ilości dni pozostałych do ich przeterminowania się.

Rysunek II-6. Przykładowy zrzut z ekranu aplikacji Fresh Box

Źródło: [14]

Aplikacja posiada ładnie zaprojektowany interfejs. Nie jest za bardzo rozbudowana, skupia się właściwie na samej ewidencji żywności. W trakcie realizacji niniejszej pracy aplikacja przestała być dostępna do pobrania ze sklepu iTunes Store. Przykładowy ekran z aplikacji zaprezentowano na powyższym rys. II-6.

2.3.2. Fridge Pal

Trochę bardziej rozbudowaną koncepcję przedstawia aplikacja Fridge Pal (pol. *Lodówkowy kumpel*) [14]. Aktualizację informacji o zawartości lodówki wykonuje się poprzez skanowanie kodów kreskowych produktów. Ułatwia to proces ich ewidencjonowania. Terminy przydatności do spożycia wyświetlane są na podstawie danych wprowadzanych przez

konsumenta. W każdej chwili można uzyskać listę brakujących produktów co jest pomocne w momencie robienia zakupów. Interfejs użytkownika zaprojektowano dość starannie. Przykładowy ekran z aplikacji zaprezentowano na rys. II-7.

Rysunek II-7. Przykładowy zrzut z ekranu aplikacji Fridge Pal

Źródło: [15]

Pewnym mankamentem jest ograniczenie jej dostępności wyłącznie do systemów iPhone oraz iPad. Nie jest przewidziana dla systemu Android. Aplikacja jest płatna, co nie pomaga jej w zdobyciu większej popularności.

2.3.3. Best Before

Best Before (pol. *Najlepiej spożyć przed*) jest przykładem aplikacji stosunkowo popularnej, lecz nierozwijanej już od dłuższego czasu. W odróżnieniu od wcześniej opisywanych aplikacji, informacje na temat zeskanowanego produktu pobierane są automatycznie z internetowych baz produktów [14]. Aplikacja posiada już wcześniej wspomnianą funkcjonalność — informowania o zbliżającym się terminie ważności produktu. Wydaje się, że jest to już standard w tego rodzaju systemach.

Rysunek II-8. Przykładowy zrzut z ekranu aplikacji Best Before

Źródło: [16]

Wizualny interfejs użytkownika jest zachęcający. Zaprojektowany system nawigacji nie przysparza większych problemów w poruszaniu się po niej, jest wręcz intuicyjny. Aplikacja dostępna jest w wersji darmowej dla systemów operacyjnych Android oraz iOS. Przykładowy ekran z aplikacji zaprezentowano na rys. II-8.

2.3.4. Keezeen

Keezeen jest jedną z bardziej rozbudowanych propozycji omawianego rodzaju aplikacji. Oprócz skanowania kodów kreskowych oraz informowania o dacie przydatności do spożycia, znaleźć można tutaj kolejne rozwiązania. Jednym z nich jest automatyczne generowanie listy zakupów sporządzanej na podstawie wybranych przez siebie przepisów kulinarnych [14]. Keezeen posiada także możliwość podpowiadania, co można przygotować do jedzenia jedynie z posiadanych produktów.

Rysunek II-9. Przykładowy zrzut z ekranu aplikacji Keezeen

Źródło: [17]

Aplikacja dostępna jest tylko dla systemu operacyjnego Windows Phone. Do jej budowy użyto interfejsu projektowania Metro UI. Oprawa graficzna jest zachęcająca, a sama nawigacja po niej nie powoduje problemów w obsłudze. Koszt jej zakupu jest niewielki. Podobnie jak w przypadku wcześniej omawianej aplikacji, konieczność uiszczenia opłaty może mieć negatywny wpływ na jej popularność. Przykładowy ekran z aplikacji zaprezentowano na rys. II-9.

2.3.5. Whaz in the Fridge

Aplikacja Whaz in the Fridge (pol. *Co jest w lodówce*) nie za bardzo oddaje swoją nazwą to, do czego ją przeznaczono. Zamiarem jej twórców nie było stworzenie systemu zarządzania zawartością lodówki. W założeniach system miał umożliwiać składowanie informacji tylko i wyłącznie o dacie przydatności do spożycia znajdujących się w niej produktów [14]. To wyróżnia ją na tle wcześniej opisywanych rozwiązań. W ramach swojego zadania aplikacja informuje o zbliżającym się niedługo terminie ważności każdego produktu. Jest to jedyna funkcja tej aplikacji.

Rysunek II-10. Przykładowy zrzut z ekranu aplikacji Whaz in the Fridge

Źródło: [18]

Aplikację zaprojektowano starannie. Posiada ładnie wkomponowane efekty graficzne. Obsługa jej jest bardzo prosta tak samo jak i nawigacja pomiędzy poszczególnymi panelami. Whaz in the Fridge dostępna jest tylko dla systemu Windows w wersji mobilnej. Przykładowy ekran z aplikacji zaprezentowano na rys. II-10.

2.4. Powody powolnego rozwoju

Jedną z bardziej istotnych rzeczy, która spowalnia powszechne wprowadzenie na rynek tego typu rozwiązań i tym samym szersze ich wykorzystywanie, jest zaporowa cena. Jest ona poza zasięgiem dla przeciętnego, a nawet i zamożniejszego gospodarstwa domowego. Dotyczy to głównie rozwiązań wbudowanych w urządzenia chłodzące. Wersje aplikacyjne nie są aż tak kosztowne. Nie mają jednak wsparcia ze strony samego urządzenia, jak i światowych producentów. Drugą mającą niemałe znaczenie kwestią jest brak odczuwania potrzeby użytkowania takich rozwiązań przez potencjalnych nabywców. Traktowane są bardziej jako gadżet niż produkt niezbędny do codziennego użytkowania.

Niewiele mniejsze znaczenie ma niedoskonałość, hermetyczność, a co za tym idzie niekompatybilność oferowanych rozwiązań. Ich bardziej powszechne wykorzystanie mogłoby mieć wpływ na współczesne życie. Tak jak potrafiliśmy żyć kiedyś bez telefonów

komórkowych, a dzisiaj nie wyobrażamy sobie bez nich normalnego funkcjonowania. Tak można przewidzieć, że za niedługi okres czasu rozwiązania te staną się codziennością.

Prawidłowość tę zauważa Sherry Turkle[4], która od wielu lat jest cenioną badaczką Internetu i nowych technologii. W poniżej cytowanej wypowiedzi z artykułu poświęconego uzależnieniu się od nowych technologii [19], wypowiada się w następujących słowach:

> *"30 lat temu zastanawialiśmy się, do czego przyda nam się komputer, a dzisiaj pytamy: do czego się nie przyda? Kiedyś człowiek wykształcił technologie, budował je i rozwijał, teraz to one kształtują nas."*

Możemy założyć, że budowany wokół świat i rozwijane w nim technologie, mają bardzo znaczący wpływ na sposób w jaki funkcjonujemy. Sposób wykorzystania tych udogodnień będzie jednak zależeć od wybranych kierunków rozwoju. Niewątpliwie poprzez reakcję na sposób ich działania mamy wpływ na ten kierunek.

2.5. Tendencje w rozwoju

Projektanci systemów poświęcili wiele lat na realizację tematyki poruszanej w niniejszej pracy. Pomimo dużych starań nie udało się tego w pełni osiągnąć. Widoczne są jednak tendencje, które zmierzają w kierunku realizacji naprawdę inteligentnej lodówki. Określenie to w tym przypadku oznacza taki sposób wykonania urządzenia, aby jego obsługa nie była uciążliwa dla użytkownika, a z samego użytkowania płynęłyby wymierne korzyści. Inaczej to ujmując, im więcej będzie samo robiło i myślało, tym bardziej będzie „inteligentne".

Wszystko zależy głównie od samych konsumentów i podejmowanych przez nich decyzji. W pewnym momencie będą bardziej zainteresowani konkretnymi rozwiązaniami. Moment ten to sytuacja, kiedy rozwiązania te nie będą pełniły roli jedynie zbędnego gadżetu, ale będą przynosiły użytkownikowi konkretne korzyści.

2.6. Podsumowanie

Powszechniejsze zaistnienie na rynku rozwiązań podobnych do proponowanego w niniejszej pracy nie jest sprawą prostą. Świadczą o tym wspomniane w niniejszym rozdziale powody. W porównaniu z informatyzacją innych obszarów życia, informatyzacja sektora

[4] Sherry Turkle jest profesorem programu akademickiego poświęconemu nauce, technologii i społeczeństwie w Instytucie Technologicznym w Massachusetts (ang. *Massachusetts Institute of Technology*) [65].

AGD przebiega stosunkowo wolno. Jak już wcześniej wspomniano, brak konieczności użytkowania oraz brak wymiernych korzyści ze stosowania tego typu rozwiązań, mogą mieć tutaj niewątpliwie decydujące znaczenie.

Zakładane w pracy podejście powinno prowadzić do powszechniejszego użytkowania tego rodzaju rozwiązań i w konsekwencji do zwiększenia dynamiki rozwoju takich technologii. Jednocześnie powinno pozwalać użytkownikowi na łatwe przyswojenie nowej technologii przy niewielkim koszcie wprowadzenia do użytku w określonej sferze życia.

Rozdział III.

ZAŁOŻENIA PROPONOWANEGO ROZWIĄZANIA

W niniejszym rozdziale Autor przybliża powody zajęcia się tematyką pracy. Nakreśla także szkic koncepcji oraz przedstawia główne założenia funkcjonowania systemu.

3.1. Motywacje przyjętego rozwiązania

Stworzenie koncepcji proponowanego rozwiązania motywowane było m.in. długoletnim brakiem dostępności podobnych rozwiązań. Systemy informatyczne odgrywają w życiu ogromną rolę. W każdym miejscu, gdzie się pojawią, uprawniają jego funkcjonowanie. Jednym z takich miejsc, w którym jeszcze nie do końca funkcjonują, jest sposób zarządzania żywnością w gospodarstwie domowym wraz z powiązanymi z nim obszarami użytkowymi.

3.1.1. Nieracjonalne podejście

Wykonywanie zakupów to czynność, którą wykonujemy każdego dnia. W większości przypadków kupujemy stale te same produkty. Jednak nie zawsze pamiętamy, co właściwie zazwyczaj kupujemy. Wielokrotnie zdarza się tak, że o najważniejszych produktach, i to właśnie tych, które skończyły się, przypominamy sobie dopiero po powrocie do domu. Zdarza się także, że kupujemy te same produkty po raz kolejny, gdyż nie pamiętamy, że już je posiadamy. W wielu przypadkach okazuje się, że w nagromadzonych w ten sposób produktach, dawno już minął termin przydatności do spożycia. Trudno jest nad tym zapanować, bez poświęcenia czasu na przygotowanie listy tego, co faktycznie jest niezbędne.

Innym przykładem na świadome, irracjonalne podejście, jest długie przetrzymywanie produktów w lodówce. Z takimi produktami nie do końca ma się pomysł co zrobić. W ten sposób zalegają w niej aż do momentu, kiedy przystąpimy do jej czyszczenia. Wtedy okazuje się, że dawno temu zdążyły się już zepsuć. Przeoczyliśmy to z kilku możliwych powodów. Produkt był tak ukryty, że nie zwracał na siebie uwagi. Brakowało pomysłów, co można dzięki niemu przygotować. Otwierając lodówkę, sięgaliśmy po najbliżej leżące produkty albo te, które konsumujemy najchętniej. W każdym z tych przypadków nie wykazaliśmy się większym zaangażowaniem w celu wykorzystania tego produktu. Punktuje to Bank Żywności na przygotowanym prospekcie informacyjnym, który przedstawiono na rys. III-1.

Rysunek III-1. Główne przyczyny marnowania żywności

Źródło: [20]

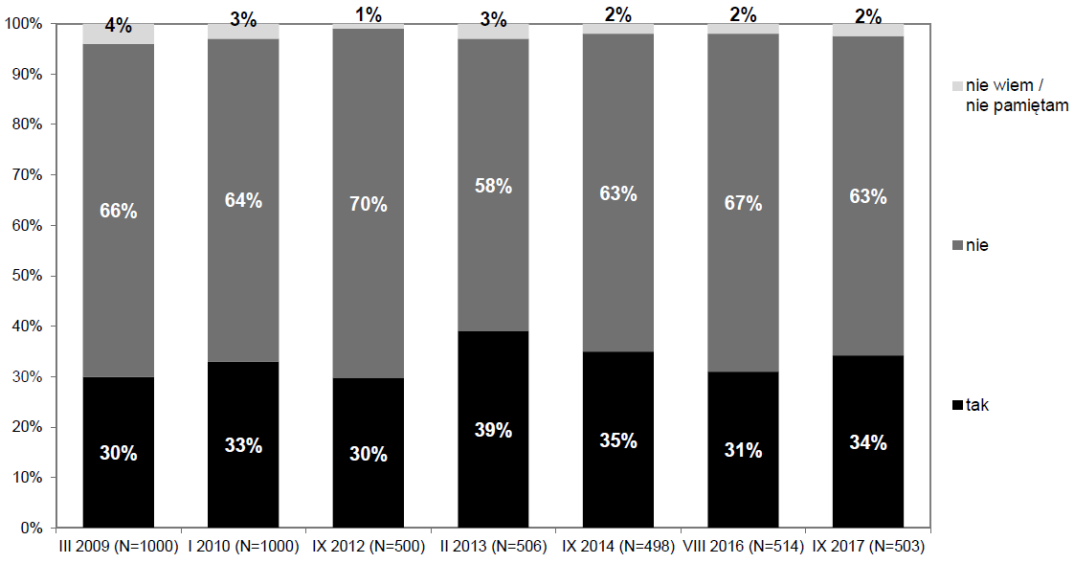
W takich przypadkach jak w opisanych powyżej, pojawia się rola dla takiego rozwiązania jak proponowane w niniejszej pracy. Wydaje się, że bez inteligentnych podpowiadaczy nie jesteśmy w stanie wymusić na sobie bardziej racjonalnego podejścia.

To są jedne z kilku powodów, które skłoniły Autora do zajęcia się tą tematyką. W następstwie tego Autor zaproponował koncepcję rozwiązania, która pozwalałaby na większe zautomatyzowanie tych procesów. Pomogłoby to wyręczyć człowieka z wykonywania powtarzalnych czynności. Analiza przedstawionego w następnym punkcie raportu, dotyczącego marnowania żywności, rozszerza listę motywacji.

3.1.2. Marnowanie żywności

Na podstawie raportu „Nie marnuj jedzenia 2017", który przygotowano na zlecenie Federacji Polskich Banków Żywności, można uświadomić sobie, jaka jest to skala problemu. Raport powstał na podstawie przeprowadzonej ankiety, która została wykonana przez instytut Kantar Millward Brown[5]. Ankieterzy zapytali Polaków czy zdarza im się wyrzucać żywność, a jeśli tak, to jakie wskazywaliby tego powody. Wyniki raportu opublikowano na stronach Banku Żywności [21].

Na sposób przeprowadzenia badania wybrano metodę sondażową CAPIBUS[6] o zasięgu ogólnopolskim. Wielkość całkowita próby wynosiła: n=503. Próba była losowa, imienna. Próba reprezentacyjna mieściła się w przedziale wiekowym od 15 do 75 lat. Populacja próby wynosiła ok. 30 221 tys. osób. Badanie przeprowadzono w terminie od 29 września do 4 października 2017 r.



Rysunek III-2. Odsetek osób, którym zdarza się wyrzucać żywność

Źródło: [22]

Wynika z niego, że liczba osób, które przyznają, że zdarza im się czasem wyrzucić żywność, wynosi aż 34%. Odsetek ten nie jest zadowalający. W porównaniu z poprzednimi

[5] Kantar Millward Brown jest to międzynarodowy koncern, który zajmuje się badaniem rynku i opinii publicznej.
[6] CAPIBUS jest to sondaż przeprowadzany przy pomocy bezpośredniego wywiadu ankieterskiego w domach respondentów. Wykonywany jest w oparciu o standardowy kwestionariusz z zastosowaniem metody CAPI (ang. *Computer Assisted Personal Interview*) [22].

latami utrzymuje się na tym samym poziomie. Tendencję tę w poniżej cytowanej wypowiedzi zauważa Marek Borowski, prezes Federacji Polskich Banków Żywności [21]. Zestawienie odsetku osób, którym zdarza się wyrzucać żywność, przedstawiono na rys. III-2.

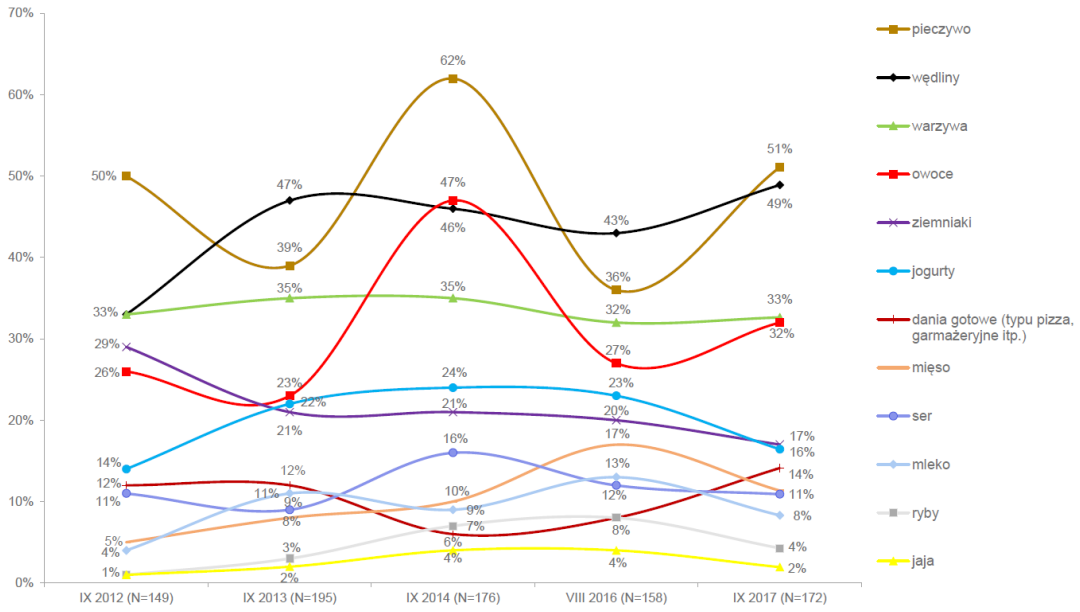
> *"Liczba osób, które przyznają się do marnowania jedzenia, od lat utrzymuje się na pewnym, stałym poziomie. Cieszy fakt, iż od kilku lat Polacy nie deklarują, że marnują więcej. Jednak dane te mobilizują nas do jeszcze bardziej wytężonej pracy na rzecz edukacji społeczeństwa, jak nie marnować jedzenia."*

Świadomość społeczeństwa dla tego dość poważnego problemu nadal nie jest zbyt wysoka. Można zaobserwować, że w ogóle się nie zwiększa. Pewnym z lekka zaskakującym zjawiskiem jest to, że im gospodarstwo domowe jest zamożniejsze, tym mniej zwraca uwagę na te kwestie. Wydawać mogłoby się, że edukacja, a co za tym idzie i świadomość, powinna być w ich przypadku większa. Zestawienie najczęściej wyrzucanych produktów, które wymieniono podczas przeprowadzania badania, przedstawiono w tabeli III-1 oraz na rys. III-3.

Tabela III-1. Zestawienie najczęściej wyrzucanych produktów

Produkty najczęściej wyrzucane	**Odsetek osób wyrzucających dany produkt**
Pieczywo	51%
Wędliny	49%
Warzywa	33%
Owoce	32%
Ziemniaki	17%
Jogurty	16%
Dania gotowe (typu pizza, garmażeryjne itp.)	14%
Mięso	11%
Ser	11%
Mleko	8%
Ryby	4%
Jaja	2%

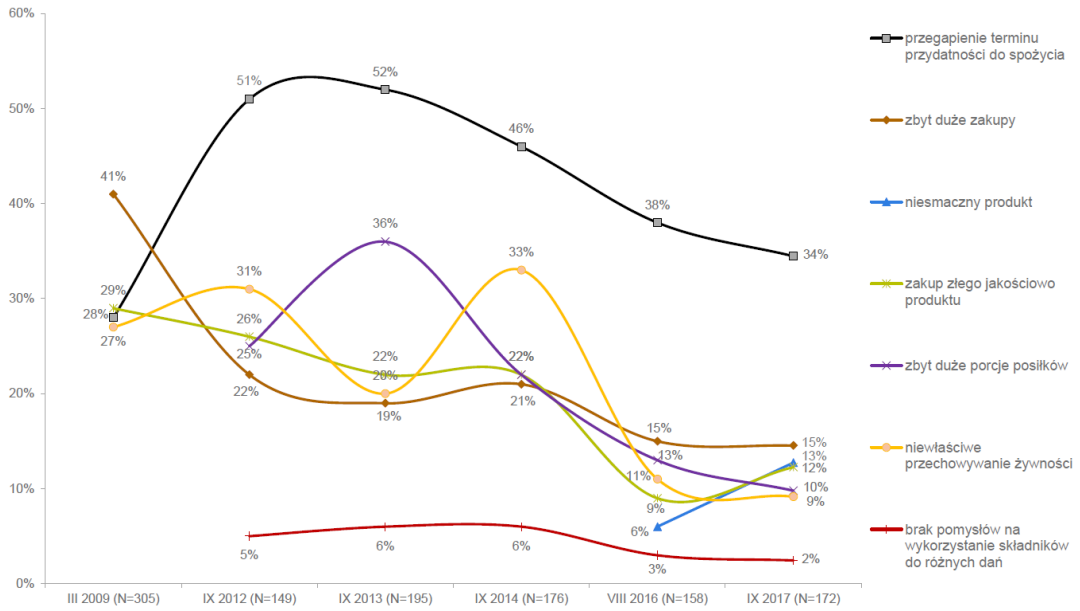
Źródło: Opracowanie własne na podstawie [22]



Rysunek III-3. Najczęściej wyrzucane produkty

Źródło: [22]

W samej tylko Polsce co roku marnowanych jest ok. 9 mln ton żywności. Szacunkowe dane na ten temat przedstawia Greenpeace Polska, podkreślając, że stawia to Polskę na dość niechlubnym 5. miejscu w całej Unii Europejskiej [23].



Rysunek III-4. Powody wyrzucania żywności

Źródło: [22]

Jest to problem nie tylko społeczny, ale także i ekonomiczny. Do produkcji wyrzucanej żywności zużyto dość konkretną ilość wody, energii, jak i czasu. Transport tych produktów

spowodował większą emisję dwutlenku węgla do atmosfery. Przykładów można by było wyliczyć wiele. Wymienione w czasie badania najczęstsze powody wyrzucania żywności przedstawiono w tabeli III-2 oraz na rys. III-4.

Tabela III-2. Zestawienie najczęstszych powodów wyrzucania żywności

Powody wyrzucania jedzenia	**Odsetek osób wskazujących na dany powód**
Przegapienie terminu przydatności do spożycia	34%
Zbyt duże zakupy	15%
Niesmaczny produkt	13%
Zakup złego jakościowo produktu	12%
Zbyt duże porcje posiłków	10%
Niewłaściwe przechowywanie żywności	9%
Brak pomysłów na wykorzystanie składników do różnych dań	2%
Brak listy zakupów	1%
Inne	1%
Nie zastanawiam się nad tym	2%

Źródło: Opracowanie własne na podstawie [22]

Można by zadać pytanie: Dlaczego zamiast zjeść, marnujemy jedzenie?

Jedną z głównych przyczyn wskazywanych przez respondentów jest przegapianie terminu ważności produktów. Znaczyłoby to, że nie rzucały im się w oczy w momencie otwierania lodówki. Zwyczajnie nikt im o nich nie przypomniał, a sami o nich nie pamiętali. Analizując kolejne powody, to wymieniamy, robienie zbyt dużych zakupów lub zakup produktów o niskiej jakości. Trudno się z tym nie zgodzić. Potrafimy skusić się na wszelkie promocje albo po prostu nie mamy świadomości, że większość z kupowanych produktów mamy już w lodówce. Uwidacznia się też tutaj naturalne ludzkie lenistwo. Zakupy wolimy robić rzadziej, przez to kupujemy więcej, niż częściej, a bardziej rozsądnie. Powoduje to to, że większość produktów kupowana jest na zapas. Większość z nich szybciej zmarnuje się, niż zostanie zjedzona. Pod koniec listy z wymienionymi powodami znalazły się: brak pomysłów na wykorzystanie produktów, jak i brak przygotowanej listy zakupów. Listę zakupów two-

rzymy zazwyczaj stojąc przy półce sklepowej. Mało kto tworzy ją wcześniej. Odczucie straconego czasu poświęconego na jej sporządzenie, jak i wrażenie podwójnego wtedy kompletowania tej listy, powoduje, że przestajemy podchodzić do tej sprawy racjonalnie.

Tak się przyzwyczailiśmy postępować i dopóki nie znajdzie się alternatywa potrafiąca to zmienić, tak zapewne będzie nadal. Taki stan rzeczy, jak opisany w niniejszym podrozdziale, to kolejny powód, dzięki któremu tematyka pracy dotyczy zagadnień związanych z usprawnieniem zarządzania żywnością.

Źródło: Opracowano na podstawie [21], [22] oraz [23].

3.2. Wymagania na system

W celu wykonania analizy wymagań na system posłużono się modelem klasyfikacji FURPS[7]. Na bazie tej funkcjonalności powstały konkretne funkcje użytkowe, wybrano środowiska pracy oraz stworzono odpowiednią architekturę systemu. Niektóre wymagania niefunkcjonalne spełniono poprzez użycie platformy chmurowej Microsoft Azure.

3.2.1. Funkcjonalność

- system powinien zapewniać:
    - możliwość zmiany stanu zawartości lodówki,
    - obsługę systemów zewnętrznych z interfejsu systemu,
    - możliwość tworzenia listy zakupów przy użyciu różnych kryteriów, tworzoną zarówno automatycznie, jak i ręcznie,
    - możliwość realizacji zakupów przy użyciu różnych kryteriów, wykonywaną zarówno automatycznie, jak i ręcznie,
    - dostęp do informacji o produktach z baz produktów,
    - dostęp do przepisów kulinarnych.
- system powinien wspomagać użytkownika przy podejmowaniu decyzji o wyborze dań do przyrządzenia,

[7] FURPS jest to model klasyfikacji wymagań funkcjonalnych i niefunkcjonalnych. Określa obszary wymagań: Funkcjonalność (ang ***F**unctionality*), Użyteczność (ang. ***U**sability*), Niezawodność (ang. ***R**eliability*), Wydajność (ang. ***P**erformace*) oraz Przystosowalność (ang. ***S**upportability*) [66].

- system powinien zawczasu informować użytkownika o zbliżających się zdarzeniach związanych z produktami,
- system powinien pracować w najbardziej popularnym środowisku mobilnym,
- konstrukcja systemu powinna zapewniać łatwą możliwość rozbudowy na inne platformy mobilnych systemów operacyjnych,
- system powinien zapewniać mechanizmy kompatybilności w celu łatwej migracji danych użytkownika z bieżącej wersji systemu do wersji następnych,
- dane przechowywane w systemie powinny być odpowiednio zabezpieczone przed nieautoryzowanym dostępem,
- wszelkie połączenia do bazy danych systemu nie mogą być wykonywane bezpośrednio z urządzeń mobilnych.

3.2.2. Użyteczność

- system powinien zapewniać łatwość użytkowania poprzez intuicyjną obsługę,
- system powinien posiadać bogaty graficznie interfejs użytkownika stworzony przy użyciu nowoczesnych technologii,
- nawigacja po systemie powinna być spójna, tzn. podobne funkcjonalnie obszary systemu powinny być obsługiwane w ten sam sposób,
- system powinien być dostępny całodobowo.

3.2.3. Niezawodność

- czas awarii nie może być dłuższy niż 30 minut,
- czas reakcji na awarię nie może być dłuższy niż 3 minuty,
- częstotliwość występowania awarii nie może być większa niż raz w roku,
- system podczas swojej pracy nie powinien nigdy doprowadzić do zawieszenia się urządzeń mobilnych.

3.2.4. Wydajność

- czas logowania do systemu nie powinien być dłuższy niż 5 sekund,

- czas uzyskania odpowiedzi z serwera systemu na wysłane żądanie nie powinien być dłuższy niż 3 sekundy,
- czas uzyskania odpowiedzi z serwisów zewnętrznych na wysłane żądanie nie powinien być dłuższy niż 5 sekund,
- system powinien być w stanie obsłużyć 1 tys. symultanicznych połączeń,
- system powinien uwzględniać wzrost liczby aktywnych użytkowników systemu o 100 tys. w przeciągu 1 roku,
- system nie powinien powodować większego zużycia baterii urządzeń mobilnych niż zaleca to dostawca mobilnego systemu operacyjnego.

3.2.5. Przystosowalność

- architektura systemu powinna opierać się na wykorzystaniu powszechnie używanych wzorców projektowych,
- konstrukcja systemu powinna być modułowa,
- konstrukcja systemu powinna umożliwiać dość łatwą rozszerzalność systemu z uwzględnieniem wykorzystania wtyczek,
- konstrukcja systemu powinna opierać się na mechanizmach pozwalających na łatwe i miarodajne testowanie systemu,
- system powinien zapewniać jego łatwą konfigurację poprzez samo opisujące się menu konfiguracyjne,
- system powinien być dostępny z oficjalnych sklepów należących do dostawców mobilnych systemów operacyjnych,
- system powinien być dostępny przede wszystkim w języku polskim z możliwością jego lokalizacji na inne języki.

3.3. Propozycja nowego rozwiązania

Proponowane rozwiązanie ma na celu głównie wspomaganie, a nie zastępowanie użytkownika, w czynnościach związanych z zarządzaniem żywnością. Dotyczy to zarówno gospodarstw domowych, jak i innych podmiotów mających z nią styczność, będących zainteresowanymi usprawnieniem jej obrotu.

3.3.1. Założenia rozwiązania

Podstawowym założeniem proponowanego rozwiązania jest jego praca na urządzeniach mobilnych współdziałających z serwerem systemu działającym w chmurze publicznej.

Serwer systemu:

- bazuje na chmurze Microsoft Azure,
- pełni rolę zarządzającą całym systemem,
- pełni także rolę pośrednika w przekazywaniu informacji pomiędzy aplikacją użytkową systemu a serwisami zewnętrznymi,
- obsługę bazodanową zapewnia relacyjna baza danych Azure SQL,
- dostęp do bazy danych możliwy jest jedynie z poziomu serwera systemu,
- obsługa systemu wykonywana jest poprzez udostępnione API, które zrealizowane jest w architekturze REST.

Aplikacja użytkowa:

- obsługa systemu odbywa się przy pomocy standardowych urządzeń mobilnych takich jak smartfony i tablety,
- na system operacyjny urządzeń mobilnych wybrano Android,
- komunikacja aplikacji użytkowej z serwerem systemu realizowana jest przy pomocy działającej i udostępnionej na nim usługi RESTful API,
- dostęp do serwisów zewnętrznych realizowany jest z aplikacji użytkowej przy pomocy wtyczek zaimplementowanych w serwerze systemu. Wtyczki zapewniają komunikację z serwisami zewnętrznymi, mapując w odpowiedni sposób komunikaty z i do aplikacji,
- użyto wbudowane możliwości urządzeń mobilnych do odczytu kodów kresowych, wykorzystując to do identyfikacji produktów,
- użyto wbudowane możliwości urządzeń mobilnych do wykonywania zdjęć produktów, wykorzystując to do ich ewidencji.

3.3.2. Koncepcja rozwiązania

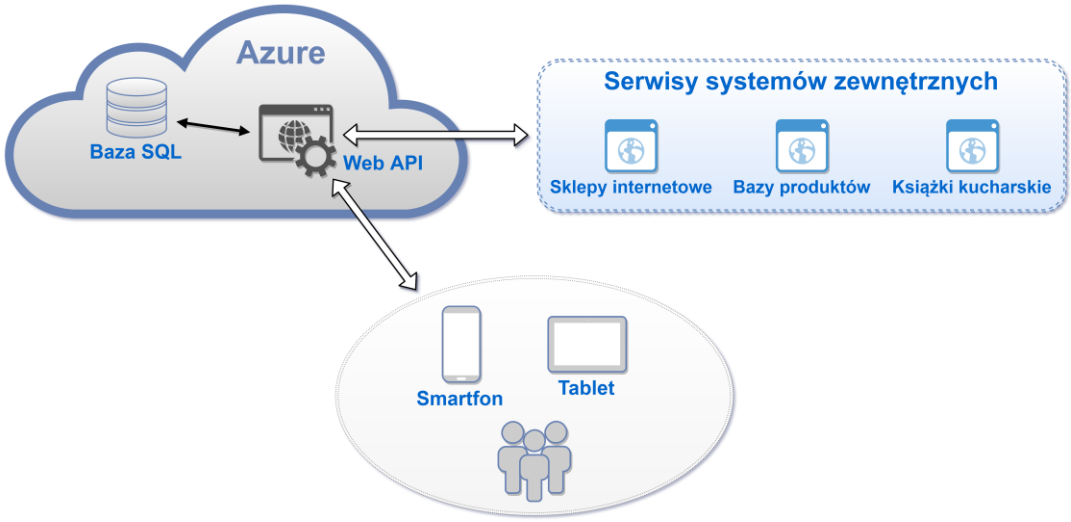
Koncepcja proponowanego rozwiązania bazuje na trzech obszarach użytkowych: aplikacji użytkownika, serwerze systemu oraz integracji z serwisami systemów zewnętrznych.

Centralna jednostka sterująca całym systemem znajduje się na serwerze umiejscowionym w chmurze publicznej Azure. Obsługa systemu realizowana jest poprzez urządzenia mobilne takie jak smartfon lub tablet. System posiada przewidziany zestaw dziedzin wtyczek umożliwiających jego rozszerzanie. Przy ich pomocy system integruje się z serwisami zewnętrznymi. Ustalony zakres funkcjonalności dla tych dziedzin pozwala na wykonywanie podstawowych czynności w tych serwisach. Użytkownik ma możliwość użytkowania serwisów zewnętrznych z aplikacji użytkowej w ramach udostępnionej funkcjonalności.

Komunikacja pomiędzy serwisami zewnętrznymi a systemem realizowana jest poprzez serwer systemu. Serwer systemu posiada do tego celu odpowiednio zaprojektowany moduł Web API. Tym samym kanałem aplikacja mobilna wymienia z serwerem systemu informacje wewnątrz systemowe.

Dostęp do bazy danych systemu zapewniony jest tylko z serwera systemu. Nie ma możliwości na bezpośrednią komunikację do niej z aplikacji mobilnych. Na przechowywanie danych systemu wybrano relacyjną bazę Azure SQL.

Wszystkie wymienione elementy systemu ściśle ze sobą współdziałają. Poglądowy schemat koncepcji przedstawiono na rys. III-5.



Rysunek III-5. Schemat koncepcji proponowanego systemu

Źródło: Opracowanie własne

Użytkownik ma możliwość zarządzania swoją lodówką w systemie, tzn. może zmieniać stan zawartości lodówki poprzez dodawanie, usuwanie z niej produktów. Realizuje to przy pomocy m.in. skanowania kodów kreskowych. Identyfikację kodów kreskowych system zapewnia poprzez użycie wbudowanych w urządzenia mobilne kamer cyfrowych. Dane

dotyczące produktów mogą być pobrane automatycznie z baz produktów albo wpisane ręcznie. W każdej chwili użytkownik ma możliwość sprawdzenia aktualnej zawartości lodówki.

System posiada szereg funkcjonalności ułatwiających użytkownikowi zarządzanie lodówką. Podzielone są na dwa rodzaje: działających w tle oraz ujawniających się podczas działań wykonywanych przez użytkownika. Jedną z takich funkcji działających w tle jest informowanie o niskim stanie produktów lub przypominanie o zbliżającym się terminie zdatności do spożycia produktów. System analizuje także zawartość lodówki i przygotowuje propozycję listy zakupów. Jest w stanie także sam tworzyć zamówienia na brakujące produkty, pod warunkiem, że kryterium zamawiania tych produktów zostało wcześniej ustawione przez użytkownika na „stale zamawiane".

System jest w stanie także podpowiadać użytkownikowi w wielu kwestiach. Użytkownik podczas przygotowywanej samodzielnie listy zakupów otrzymuje informacje, co jeszcze mógłby zamówić. Podpowiadanie działa na podstawie kryteriów pomocy, np. wyświetlana jest informacja o częstotliwości kupowania produktu. Istnieje także możliwość zamówienia produktów na okoliczność jakiegoś wydarzenia, np. planowanego przyjęcia lub innego kryterium określonego przez użytkownika. Użytkownik ma także możliwość pobrania przepisów kulinarnych z serwisów kucharskich. Wykonuje to bezpośrednio z interfejsu systemu. System automatycznie podpowie w tym momencie, których składników brakuje w lodówce, aby przyrządzić wybrane danie. Wskaże także te przepisy, dla których użytkownik posiada wszystkie produkty i dzięki temu może przygotować posiłek.

3.3.3. Opis funkcjonalności

Analiza funkcjonalno–użytkowa dla proponowanego rozwiązania pozwoliła wyodrębnić poniżej spisaną listę funkcji użytkowych:

1) Dodawanie produktów do zasobów lodówki.

   Wykonywane jest to poprzez sczytanie kodu kreskowego produktu. W przypadku, gdy jest to nowy produkt informacje na jego temat dostarcza się na dwa poniżej przedstawione sposoby:

   - użytkownik opisuje produkt własnoręcznie lub uzupełnia dane o produkcie pobrane z bazy produktów,
   - dane identyfikujące produkt pobierane są automatyczne z serwisów zawierających bazy produktów.

W momencie dodawania produktu można podać jego ilość, jaka jest wstawiana do lodówki. Dostarczanymi podstawowymi informacjami o produkcie są: nazwa, kod kreskowy, kategoria, zdjęcie oraz termin ważności produktu.

2) Usuwanie produktów z zasobów lodówki.

   Wykonywane jest to poprzez sczytanie kodu kreskowego produktu. W momencie usuwania można podać ilość, jaka jest wyjmowana z lodówki.

3) Pobranie aktualnego stanu produktów.

   Informowanie o ilości dostępnych w lodówce sztuk danego produktu. Informację otrzymuje się na dwa sposoby:

   - poprzez ręczne sczytanie kodu kreskowego produktu,
   - automatyczne powiadamianie o braku produktu lub zbliżającym się momencie jego nastąpienia.

   W przypadku automatycznego powiadomienia wyświetlany jest adekwatny komunikat. W obu przypadkach wyświetlana jest aktualna ilość produktu. W momencie wyświetlenia informacji o stanie istnieje możliwość bezpośredniego zamówienia produktu.

4) Oznaczanie braku produktu.

   Produkty można oznaczyć na dwa sposoby:

   - poprzez ręczne sczytanie kodu kreskowego produktu,
   - automatyczne oznaczanie produktu przez algorytmy systemu analizujące zawartość lodówki.

   Istnieje możliwość przypisania brakującemu produktowi statusu pilnego jego uzupełnienia. W momencie oznaczenia produktu istnieje możliwość jego bezpośredniego zamówienia.

5) Tworzenie zapotrzebowania na produkty.

   Oznaczanie produktów do zakupu tworzy listę zakupów. Można ją sporządzić na cztery sposoby:

   - poprzez ręczne sczytanie kodu kreskowego produktu,
   - poprzez oznaczenie produktu filtrem, np. polecane przez znajomych,

- automatycznie poprzez wybranie filtru, np. na okoliczność przyjęcia lub sugerowane zakupy. Lista uzupełniana jest wtedy wcześniej przypisanymi do filtru produktami,
- wybór produktu ze sklepów internetowych.

Istnieje możliwość przypisania zamawianemu produktowi statusu pilnego zakupu. W momencie oznaczenia produktu istnieje dodatkowo bezpośrednia możliwość realizacji zamówienia.

6) Pobieranie informacji na temat produktu.

Otrzymanie informacji szczegółowej dotyczącej produktu można wykonać na dwa różne sposoby:

- poprzez ręczne sczytanie kodu kreskowego produktu,
- wybór produktu z serwisów baz produktów.

Wyświetlane są informacje o produkcie oraz historia jego zakupu.

7) Automatyczne zakupy.

Automatyczne zamawianie produktów w sklepie internetowym wykonywane przez system. Istnieją trzy możliwości realizacji automatycznych zakupów:

- realizacja zakupów oznaczonych jako stale zamawiane,
- lista sugerowanych zakupów tworzona jest automatycznie, jednak wymaga akceptacji w celu realizacji,
- realizacja zakupów na podstawie wybranego kryterium, np. uzyskanie wartości zamówienia pozwalającą na darmową dostawę.

8) Pomoc w przygotowywaniu posiłków.

Możliwość uzyskania informacji dotyczącej przygotowywanych dań:

- co można ugotować z posiadanych produktów,
- czy posiadamy wszystkie produkty do ugotowania wybranego dania.

9) Podpowiadanie zakupów.

Usługa działająca w tle, służąca do automatycznego powiadamiania o zbliżającym się niedługo momencie, w którym zabraknie produktów.

10) Powiadomienie o terminie ważności produktów.

    Usługa działająca w tle, służąca do automatycznego powiadamiania o zbliżającym się terminie przydatności do spożycia produktów.

11) Pobieranie przepisów kulinarnych.

    Pozyskiwanie przepisów kulinarnych z serwisów kucharskich. Realizowane jest to poprzez integrację systemu z zewnętrznymi serwisami kucharskimi.

12) Konfiguracja parametrów.

    Istnieje możliwość ustawienia głównych parametrów pracy systemu. Parametry te wykorzystywane są do informowania użytkownika o m.in. minimalnym stanie produktów lub zbliżającym się terminie zdatności do ich spożycia. Informacje wyświetlane są w postaci powiadomień systemowych.

13) Obsługa kryteriów zakupów.

    Istnieje możliwość ustawiania kryteriów zakupów oraz edycji ich nazw. Kryteria ustala użytkownik. Wbudowanymi kryteriami są: stale zamawiane produkty, na okoliczność przyjęcia, po uzyskaniu wartości zamówienia, polecane przez znajomych oraz sugerowane zakupy.

3.3.4. Modułowość rozwiązania

Proponowane rozwiązanie posiada budowę modułową. Dzięki temu jest w stanie zapewnić rozszerzalność systemu przy pomocy wtyczek. Obsługa wtyczek realizowana jest przy pomocy mechanizmu MEF. Przewidziano dwa obszary wykorzystania mechanizmów wtyczek. Pierwszy w aplikacji webowej pozwala na podłączenie do systemu serwisów zewnętrznych. Drugi w aplikacji mobilnej służy do tworzenia alternatywnych rozwiązań dla określonej grupy funkcjonalności.

System do realizacji wtyczek udostępnia zestaw interfejsów. Każda wtyczka posiada oprogramowany interfejs przewidziany dla danej dziedziny zagadnienia. Daje to możliwość przygotowania nowych wtyczek przez osoby trzecie nieznające struktury systemu.

Aplikacja webowa posiada następujące rodzaje wtyczek:

1) Sklepy internetowe.

    Wtyczki pozwalają m.in. na realizację zakupów w sklepach internetowych.

2) Bazy produktów.

   Wtyczki pozwalają m.in. na pobieranie pełnych informacji o produktach z serwisów udostępniających bazy produktów.

3) Książki kucharskie.

   Wtyczki pozwalają m.in. na pobieranie przepisów kulinarnych z serwisów kucharskich poświęconych gotowaniu.

4) Realizacje sposobu komunikacji z usługami internetowymi.

   Wtyczki te upraszczają sposób wymiany danych z usługami internetowymi poprzez zaimplementowaną obsługę komunikacji. Ukierunkowana jest ona na konkretny sposób komunikacji, np. REST API z użyciem JSON, REST API z użyciem XML lub komunikacja poprzez protokół SOAP. Wtyczce wystarczy tylko przekazać dane związane z komunikatem. Sam proces tworzenia, wysyłania, i odbierania komunikatu, realizowany jest przez wtyczkę.

Aplikacja mobilna posiada następujące rodzaje wtyczek:

1) Identyfikacja produktu.

   Różne sposoby na dostarczenie informacji o produkcie, np. przy pomocy kodów kreskowych, kodów DataMatrix, RFID lub ręczne uzupełnienie danych.

2) Algorytmy analizujące pod różnym kątem sprawy związane z produktem.

   Różne sposoby realizacji algorytmów dla analizy spraw związanych z produktami oraz ewentualne stworzenie na nie zapotrzebowania.

3) Kryteria zakupowe.

   Różne sposoby na stworzenie listy produktów do zamówienia, np.: na okoliczność przyjęcia, sugerowane zakupy.

3.3.5. Integracja z systemami zewnętrznymi

Proponowane rozwiązanie, w swoim założeniu, powinno posiadać możliwość współpracy z niezależnymi systemami zewnętrznymi. Obsługa tych serwisów odbywa się bezpośrednio z poziomu systemu. Komunikacja z nimi następuje za pośrednictwem serwera systemu zlokalizowanego w chmurze. Serwer systemu pełni rolę pośrednika w przekazywaniu

informacji. Integracja z systemami zewnętrznymi możliwa jest poprzez przygotowanie odpowiednich interfejsów dla poszczególnych rodzajów serwisów.

Przyjęto współpracę z trzema rodzajami serwisów zewnętrznych:

1) Sklepy internetowe.

   Umożliwia użytkownikowi m.in. realizację zakupów.

2) Serwisy posiadające skatalogowaną bazę produktów.

   Umożliwia użytkownikowi m.in. pobieranie danych o produktach na podstawie kodów kreskowych.

3) Serwisy poświęcone przepisom kulinarnym.

   Umożliwia systemowi m.in. pobieranie przepisów kulinarnych.

Rozdział IV.

ZASTOSOWANE NARZĘDZIA I TECHNOLOGIE

W niniejszym rozdziale przedstawiono krótką charakterystykę użytych technologii. Posłużyły Autorowi w implementacji prototypu dla proponowanego rozwiązania. Technologie warte uwagi opisano w większym zakresie.

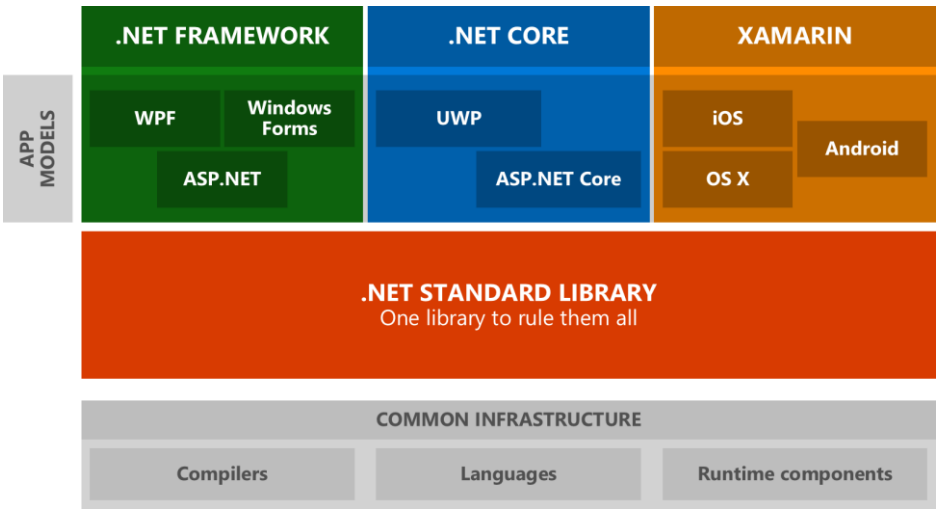
4.1. Platformy i języki programistyczne

Na realizację prototypu wybrano najnowsze wersje platform programistycznych. Ich finalne postaci ukazały się w połowie 2017 r. Do stworzenia kodu wybrano jeden z najpopularniejszych języków programowania. Wybór tych rozwiązań spowodował pewną ilość ograniczeń i utrudnień podczas realizacji prototypu. Konieczne było rozwiązywanie pewnych sfer zagadnień w inny sposób, niż było to realizowane na standardowej platformie .NET. Oprócz wspomnianych ograniczeń, wybór platformy w wersji Core przyniósł także wymierne korzyści. Dało to możliwość m.in. do generowania prostszego kodu lub do stosowania najnowszych rozwiązań. Poruszony temat użytych rozwiązań programistycznych opisano dokładniej w niniejszym podrozdziale.

4.1.1. .NET Standard 2.0

.NET Standard w wersji 2.0 jest stosunkowo nową platformą programistyczną. Jej premiera nastąpiła w sierpniu 2017 r.

Głównym jej założeniem jest możliwość współdzielenia kodu programistycznego na wielu systemach operacyjnych. Traktowana jest jako biblioteka bazowa dla wszystkich platform opartych na architekturze .NET. Zależności pomiędzy istniejącymi środowiskami .NET przedstawiono na rys. IV-1.



Rysunek IV-1. Zależności pomiędzy środowiskami .NET

Źródło: [24]

.NET Standard charakteryzuje się poniższymi właściwościami:

- współdzielenie kodu programistycznego – .NET Standard jest zbiorem API, do którego dostęp musi być zapewniony przez wszystkie istniejące implementacje środowisk .NET,
- olbrzymi zbiór API – do tej pory zrealizowano ponad 30 tys. interfejsów API,
- tryb kompatybilności – w okresie przejściowym wprowadzono możliwość wymuszenia kompatybilności ze wcześniejszymi komponentami bazującymi na podstawowym środowisku .NET,
- szerokie wsparcie dla poniższych platform:
  - .NET 4.6.1,
  - .NET Core 2.0,
  - Mono 5.4,
  - Xamarin.iOS 10.14 / Xamarin.Mac 3.8 / Xamarin.Android 7.5,
  - nadchodzące wersje UWP.

Źródło: Opracowano na podstawie [24], [25] oraz [26]

4.1.2. .NET Core 2.0

.NET Core w wersji 2.0 to stosunkowo nowe, prężnie rozwijające się, wieloplatformowe środowisko programistyczne. Udostępniane jest na zasadach otwartego oprogramowania (ang. *open source*) przez firmę Microsoft. Pojawiło się w połowie 2017 r. Ukierunkowane jest na tworzenie rozwiązań opartych na chmurze publicznej. Składa się z narzędzi programistycznych, bibliotek oraz środowiska uruchomieniowego.

Wśród licznych cech .NET Core możemy wyliczyć:

- kompatybilność z pozostałymi implementacjami .NET takimi jak platforma .NET, Xamarin, Mono,
- współpraca z usługami ISS, Nginx, Apache, Docker,
- wbudowany mechanizm wstrzykiwania zależności,
- obsługa menadżera pakietów NuGet,
- multi-platformowość środowiska zapewnia uruchamianie aplikacji na systemach operacyjnych Windows, Linux, Raspberry Pi oraz macOS,
- możliwość wyboru spośród wielu języków programowania takich jak C#, Visual Basic, F#, ale także i AngularJS.

Źródło: Opracowano na podstawie [27]

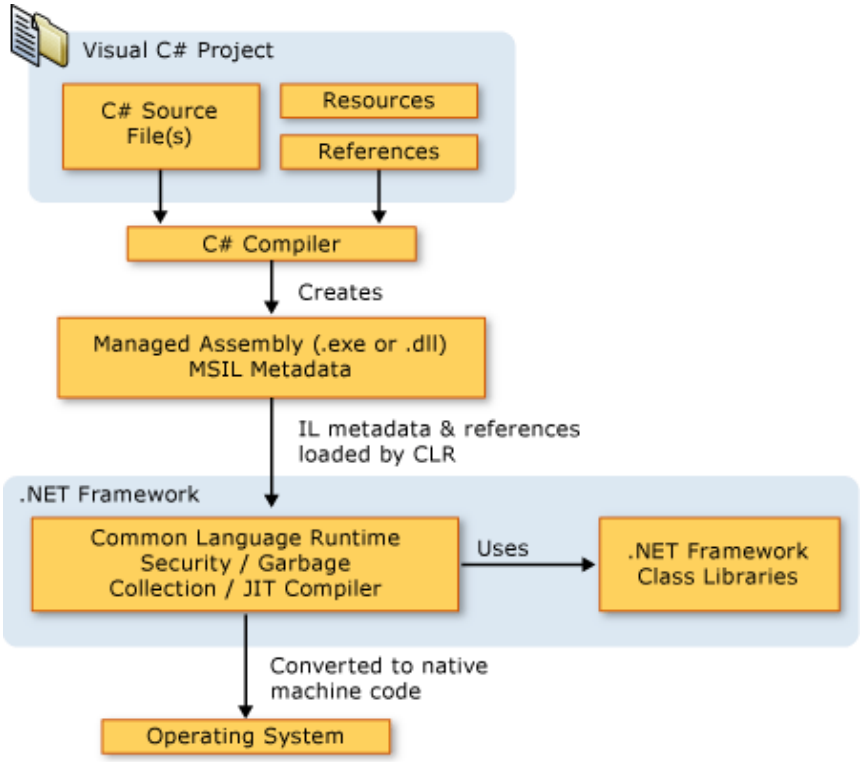
4.1.3. C#

C# (ang. *C sharp*) jest językiem programowania obiektowego. Zaprojektowany został dla firmy Microsoft na przełomie XX i XXI w. przez zespół pod kierownictwem Andersa Hejlsberga. Łączy w jedną całość wszystko to, co najlepsze z takich języków programowania jak C, C++ oraz Java. Jego nazwa sugeruje kolejną ewolucję w gronie języków C. Tak jak to miało miejsce w przypadku języka C i rozwinięcia do C++, tak tutaj postawiono na analogie muzyczne. Wykorzystano symbol krzyżyka ♯ (ang. *sharp*) [28], który w muzyce oznacza dźwięk wyższy o pół tonu niż dźwięk podstawowy. W tym przypadku byłby to dźwięk C. Ze względu na brak możliwości uzyskania tego znaku na klawiaturze komputera, przyjęto w nazewnictwie symbol kratki # (ang. *hash*).

Poniższe cele przyświecały zespołowi podczas tworzenia języka C#:

- język programowania musi być obiektowy,
- powinien być prosty i nowoczesny,

- powinien uwzględniać przenośność,
- powinien zapewniać wsparcie dla istniejących zasad Inżynierii Oprogramowania takich jak:
    - silne typy,
    - wykrywanie próby użycia niezainicjowanych zmiennych,
    - automatyczne usuwanie śmieci z pamięci (ang. *garbage collector*).



Rysunek IV-2. Proces uruchamiania kodu napisanego w języku C#

Źródło: [29]

Programy napisane w C# pracują pod kontrolą środowiska programistycznego .NET. Kod źródłowy napisany w C# kompilowany jest do kodu pośredniego IL (ang. ***I**ntermediate **L**anguage*). Jest on zgodny ze specyfikacją CIL (ang. ***C**ommon **I**ntermediate **L**anguage*). W zależności od rodzaju projektu, skompilowany kod zestawiany jest do bibliotek lub plików wykonywalnych. W momencie uruchamiania ładowany jest do środowiska uruchomieniowego CLR (ang. ***C**ommon **L**anguage **R**untime*). Jeśli nic nie stanie na przeszkodzie w wykonywaniu kodu, CLR uruchamia w locie kompilację JIT (ang. ***J**ust **I**n **T**ime*). Zadaniem JIT jest konwersja kodu języka pośredniego IL do kodu maszynowego. Program uruchomiony

w CLR działa w przestrzeni kontrolowanej zwanej kodem zarządzalnym (ang. *managed code*). Przebieg całego opisanego procesu zobrazowano na rys. IV-2.

Główne cechy języka C#:

- jako język obiektowy wspiera enkapsulację, dziedziczenie i polimorfizm,
- nie posiada mechanizmów pozwalających na wielodziedziczenie,
- może implementować nieograniczoną ilość interfejsów,
- posiada dość pokaźną ilość funkcjonalności, które są użyteczne podczas tworzenia najróżniejszego oprogramowania.

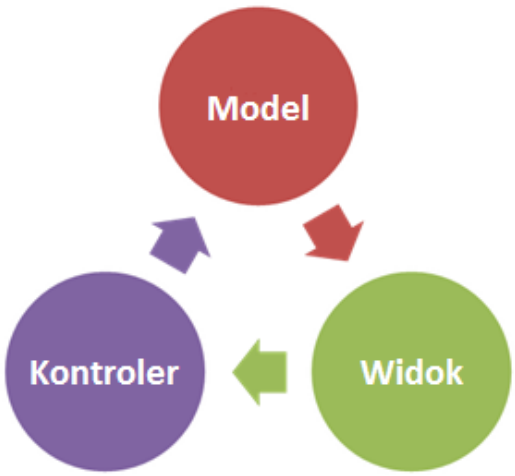
Źródło: Opracowano na podstawie [29] i [28]

4.2. Wzorce projektowe

Wzorce projektowe pomagają w organizacji kodu aplikacji. W niniejszym podrozdziale opisano dwa wzorce, które wykorzystano zarówno podczas realizacji aplikacji webowej, jak i aplikacji mobilnej.

4.2.1. MVC

Wzorzec projektowy MVC (ang. ***M**odel-**V**iew-**C**ontroller*) użyto w realizacji prototypu w części dotyczącej aplikacji webowej. Jest to jeden z najpopularniejszych wzorców projektowych [30]. Wzorzec ten pomaga programistom w konstruowaniu aplikacji poprzez separację różnych obszarów jej funkcjonowania.



Rysunek IV-3. Schemat wzorca projektowego MVC

Źródło: Opracowanie własne na podstawie [31]

Zaliczamy do nich logikę biznesową, prezentacji i sterowania. Każda z nich znajduje się w osobnym obszarze kodu aplikacji. Sprawia to, że kod programu jest bardziej czytelny. Rozdzielenie kodu ze względu na logikę daje szereg innych korzyści wymienionych w dalszej części opisu. Zależności pomiędzy tymi obszarami przedstawiono na rys. IV-3.

Główne zadania każdego z obszarów:

- Model (ang. *Model*) – w modelach zaimplementowana jest cała logika biznesowa. Z reguły modele obiektów tworzone są na podstawie schematów danych wejściowych, np. z baz danych,
- Widok (ang. *View*) – widoki obsługują warstwę prezentacji. Odpowiadają za wyświetlenie użytkownikowi treści generowanej z modelu,
- Kontroler (ang. *Controller*) – kontrolery odpowiadają za obsługę żądań użytkownika. Współpracują z modelem i widokiem w celu wyświetlenia użytkownikowi treści w odpowiedzi na jego żądanie.

Zalety stosowania wzorca MVC:

- łatwiejsze zarządzanie projektem poprzez rozdzielenie logiki funkcjonowania na odrębne obszary kodu współpracujące ze sobą,
- szybszy proces programowania,
- możliwość dostarczenia wielu widoków dla tego samego kontrolera,
- wsparcie dla technik asynchronicznych,
- modyfikacje części kodu nie wpływają na inne obszary kodu aplikacji.

Źródło: Opracowano na podstawie [32]

4.2.2. MVVM

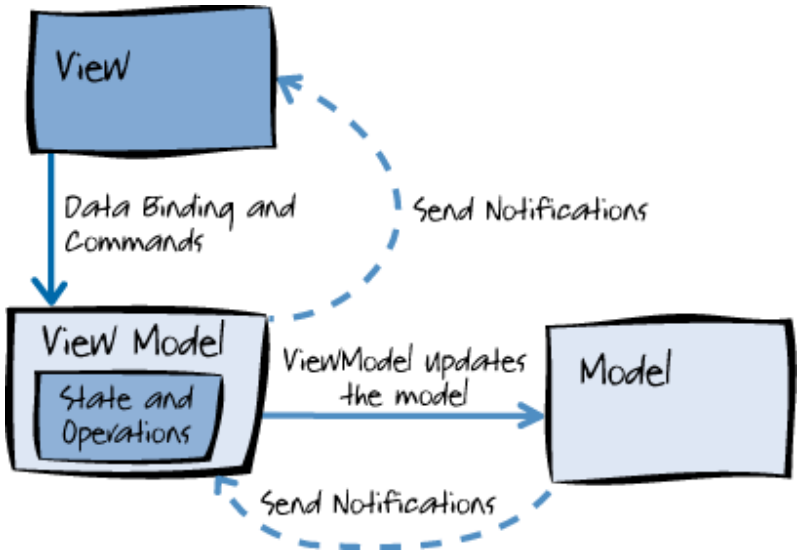
Wzorzec projektowy MVVM (ang. ***M**odel-**V**iew-**V**iew**M**odel*) użyto w realizacji prototypu w części dotyczącej aplikacji mobilnej. Obok wzorca MVC jest to jeden z najbardziej popularnych wzorców projektowych [33]. Podobnie jak on, wzorzec MVVM pomaga programistom w konstruowaniu aplikacji poprzez separację różnych obszarów kodu.

Główne zadania każdego z obszarów:

- Model (ang. *Model*) – w modelach zaimplementowana jest logika biznesowa aplikacji. Z reguły modele obiektów tworzone są na podstawie schematów danych wejściowych, np. z baz danych,

- Model widoku (ang. *ViewModel*) – działa jako warstwa pośrednia pomiędzy modelem a widokiem. Ujmując to w uproszczeniu, to pobiera dane z modelu, a następnie przekazuje je do widoku w celu ich prezentacji. Oprócz wywoływania metod w modelu, zawiera także implementację poleceń, które inicjowane są w widoku,
- Widok (ang. *View*) – widoki obsługują warstwę prezentacji. Odpowiadają za wyświetlenie użytkownikowi treści generowanej z modelu widoku. Widoki same z siebie nie generują danych do wyświetlenia.

W przypadku wzorca MVVM występuje mechanizm wiązania pomiędzy widokiem a modelem widoku. Zależności pomiędzy wszystkimi wymienionymi obszarami przedstawiono na rys. IV-4.



Rysunek IV-4. Zasada działania wzorca projektowego MVVM

Źródło: [34]

Zalety stosowania wzorca MVVM:

- kod programistyczny jest niezależny od technologii, w której wykonywana jest warstwa prezentacji,
- logika aplikacji jest niezależna od sposobu prezentacji danych,
- prosta zamiana widoków,
- możliwość wykonywania testów zautomatyzowanych.

Źródło: Opracowano na podstawie [34] i [35]

4.3. Komunikacja systemu

Komunikacja systemu pracuje pomiędzy wszystkimi jego modułami i realizowana jest poprzez opisane w niniejszym podrozdziale style i technologie.

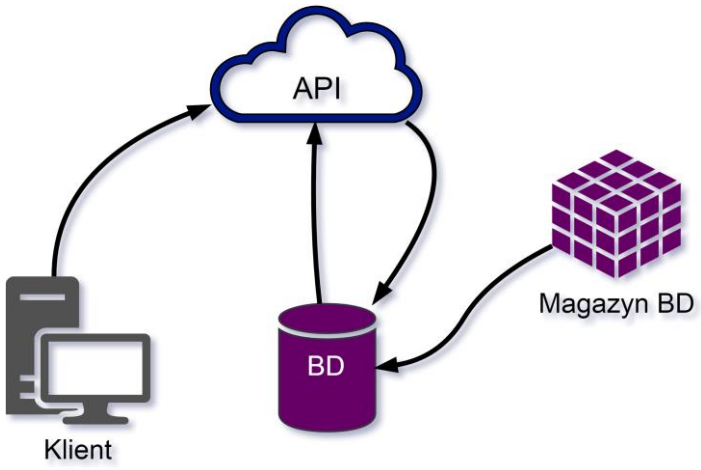
4.3.1. REST API

REST (ang. ***RE****presentational* ***S****tate* ***T****ransfer*) jest stylem architektury dla zapewnienia standardów wymiany danych pomiędzy klientem a serwerem, w rozproszonych systemach informatycznych.

Wymusza przestrzeganie poniżej wypisanych wytycznych:

- klient-serwer – ograniczenie to odpowiada koncepcji, że klient i serwer powinny być od siebie oddzielone. Oznacza to, że ich rozwój odbywa się niezależnie i nie ma na siebie wpływu,
- bezstanowość – interfejsy API usług w stylu REST są bezstanowe. Oznacza to, że połączenia mogą być nawiązywane niezależnie od siebie. Czyli że, każde wywołanie zawiera wszystkie dane, które są niezbędne do jego całościowego, pomyślnego przetworzenia,
- buforowanie – ze względu na to, że interfejs API może zwiększać obciążenie związane z żądaniami, usługa REST API powinna być zaprojektowana w taki sposób, aby umożliwiała buforowanie danych,
- jednolity interfejs – ważnym aspektem rozdzielenia klienta od serwera jest posiadanie jednolitego interfejsu. Umożliwia to wtedy klientowi komunikację z serwerem w tym samym języku niezależnie od architektonicznego zaplecza. Interfejs ten powinien zapewniać niezmienne, standardowe środki komunikacji pomiędzy klientem a serwerem takie jak korzystanie z HTTP z zasobami URI, obsługę CRUD oraz wymianę danych przy pomocy np. JSON,
- system warstwowy – różne warstwy stylu architektury REST współpracują ze sobą tworząc hierarchię, która pomaga w tworzeniu bardziej skalowalnej oraz modułowej aplikacji,
- kod na żądanie – ograniczenie to jest opcjonalne. Pozwala na przesłanie kodu lub apletów za pośrednictwem API.

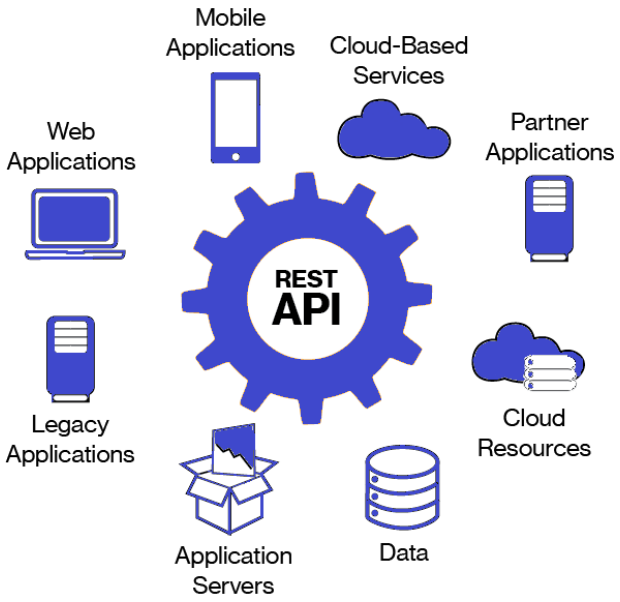
RESTful API jest to usługa internetowa (ang. *Web Service*) przestrzegająca reguł REST z zaimplementowanym wykorzystaniem protokołu HTTP. Działanie usług internetowych zaprezentowano na rys. IV-5.



Rysunek IV-5. Zasada działania usług internetowych

Źródło: Opracowanie własne na podstawie [36]

RESTful API funkcjonuje na zasadzie identyfikacji zasobów. Wykorzystuje podstawowe metody protokołu HTTP takie jak GET, PUT, POST, DELETE oraz PATCH. Każda z nich określa inny rodzaj polecenia, które zostanie wykonane po stronie serwera. Stosowanym formatem wymiany danych najczęściej jest JSON. Może to być jednak jakikolwiek inny poprawny schemat danych, jak chociażby XML czy YAML.



Rysunek IV-6. Możliwości zastosowań REST API

Źródło: [36]

REST API przewidziany jest do tworzenia zarówno prostych, jak i złożonych systemów rozproszonych. Wykorzystuje się go w celu zwiększenia wydajności, prostoty obsługi, skalowalności, możliwości modyfikacji, przenośności i niezawodności aplikacji. Aktualnie jest najczęściej stosowanym interfejsem przeznaczonym do wymiany danych pomiędzy systemami. Możliwości zastosowań REST API przedstawiono na rys. IV-6.

Źródło: Opracowano na podstawie [36]

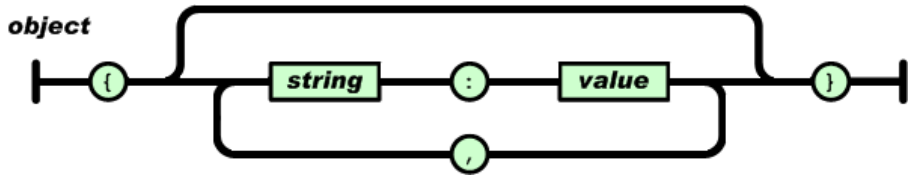
4.3.2. JSON

JSON (ang. ***J**ava**S**cript **O**bject **N**otation*) jest formatem wymiany danych. W przeciwieństwie do innych tego typu rozwiązań, jak chociażby XML, nie ma narzutu zbędnych informacji w postaci znaczników opakowujących treść przesyłanych danych. Jest niezależny od jakiegokolwiek języka programowania. Konwencja jego zapisu bazuje na konstrukcjach przyjętych w językach z rodziny C. JSON do zapisu danych używa formatu tekstowego. Przykładowy sposób zapisu danych typu obiekt przedstawiono na rys. IV-7.

JSON posiada trzy główne właściwości:

- dane zapisywane są jako zbiór par nazwa/wartość,
- lista wartości jest uporządkowana,
- zapisane dane mają postać jawną.

Zapis danych wykonywany jest przy użyciu sześciu specjalnych znaków strukturalnych: {, }, [,], : (dwukropek) oraz , (przecinek). Przy ich pomocy format JSON jest w stanie opisać każdą strukturę danych.



Rysunek IV-7. Schemat zapisu danych typu obiekt w formacie JSON

Źródło: [37]

Struktury danych zapisywane w JSON używane są przez prawie wszystkie nowoczesne języki programowania. Tym samym uzyskano bardzo prosty sposób ich mapowania. W wielu językach dane mapowane z formatu JSON przekształcane są do obiektów, tablic, struktur, słowników, tabel asocjacyjnych lub list z kluczem.

Źródło: Opracowano na podstawie [37]

4.4. Aplikacja Webowa

W aplikacji webowej stworzono główny mechanizm działania wtyczek systemu. Zbudowano je na bazie technologii opisywanych w niniejszym podrozdziale. Technologie te posłużyły także do wykonania części aplikacji obsługującej system po stronie serwera oraz prezentacji systemu w witrynie internetowej.

4.4.1. ASP.NET Core MVC/Web API/Razor Pages

ASP.NET Core jest wieloplatformowym środowiskiem programistycznym przeznaczonym do tworzenia aplikacji webowych. Posiada możliwość spersonalizowania projektu pod konkretne zastosowanie. W zależności, jaki jest to rodzaj projektu, przygotowuje odpowiednie szablony programistyczne. Wszystkie wymienione w tym punkcie sposoby na realizację usług internetowych można ze sobą łączyć.

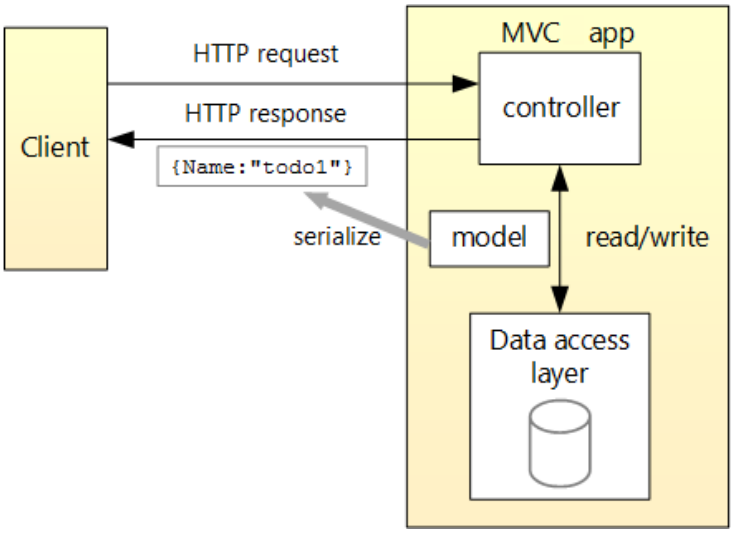
W realizacji prototypu użyto trzy takie sposoby:

1) ASP.NET Core MVC.

   Schemat przeznaczony jest głównie do realizacji złożonych serwisów internetowych. Umożliwia pełne wykorzystanie wzorca MVC. W odróżnieniu od opisywanych poniżej stron Razor, posiada wbudowaną możliwość routingu do identyfikowania zasobów.

2) ASP.NET Core Web API.

   Jest to schemat projektowania aplikacji zorientowany na realizację usługi API. Nie posiada widoków. W schemacie wykorzystywane są jedynie modele oraz kontrolery. Podobnie jak w poprzednim szablonie kontrolery służą do obsługi żądań. Identyfikację zasobów uzyskuje się poprzez wbudowane mechanizmy routingu. Oparto je na trzech elementach: obszarach (ang. *area*), kontrolerze (ang. *controller*) oraz akcji (ang. *action*). Zaimplementowano w nim większość metod HTTP do obsługi typowych rodzajów żądań. Schemat Web API przystosowany jest do pracy w architekturze REST. Ogólny schemat działania Web API przedstawiono na rys. IV-8.



Rysunek IV-8. Schemat działania aplikacji Web API

Źródło: [38]

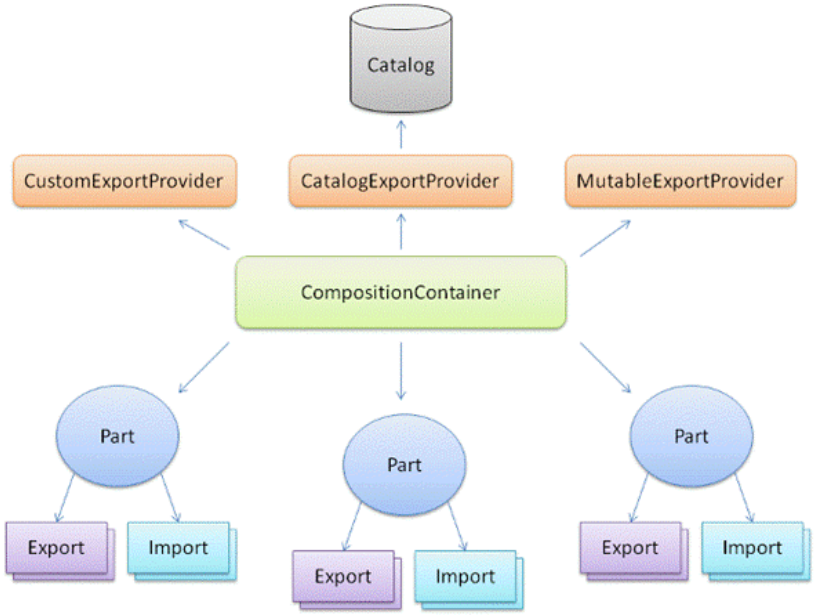
3) ASP.NET Core Razor Pages.

Strony Razor oferują prosty sposób tworzenia witryn internetowych, bez konieczności używania kontrolera. Tworzy się je jako tzw. Pages (pol. *strony*) na podstawie modelu. Strony Razor nie posiadają wbudowanego routingu. Dostęp do nich jest bezpośredni, wyznaczony poprzez fizyczną lokalizację w drzewie katalogów witryny.

Źródło: Opracowano na podstawie [38], [39] oraz [40]

4.4.2. Managed Extensibility Framework

MEF (ang. ***M**anaged **E**xtensibility **F**ramework*) to warstwa kompozycji dla platformy .NET. Poprawia elastyczność, łatwość konserwacji i testowalność złożonych aplikacji. Jej główną zaletą jest możliwość tworzenia rozszerzalnych aplikacji poprzez wbudowane mechanizmy tworzenia wtyczek. Daje to możliwość twórcom oprogramowania do ponownego wykorzystania rozszerzeń. Pozwala to również na ich tworzenie przez osoby nieznające środowiska docelowego, w którym zostaną ostatecznie uruchomione. Uproszczoną architekturę MEF przedstawiono na rys. IV-9.



Rysunek IV-9. Uproszczona architektura MEF

Źródło: [41]

Mechanizm działania MEF oparty jest na wspólnym interfejsie, który łączy aplikację z wtyczkami. W odróżnieniu od normalnej deklaracji typów, MEF wykorzystuje metadane w postaci atrybutów importu i eksportu. Oznaczone w odpowiedni sposób typy danych będą podlegać eksportowi w czasie działania aplikacji.

Źródło: Opracowano na podstawie [41] i [42]

4.4.3. Wstrzykiwanie zależności

Wstrzykiwanie zależności (ang. ***D**ependency **I**njection*, DI) jest szczególną formą szerszej techniki zwanej odwróconym sterowaniem (ang. ***I**nversion **o**f **C**ontrol*, IoC). Mechanizmy wstrzykiwania zależności służą do przekazywania instancji obiektu poprzez jego wstrzyknięcie w miejscu żądania.

Występuje kilka rodzajów wstrzykiwania zależności:

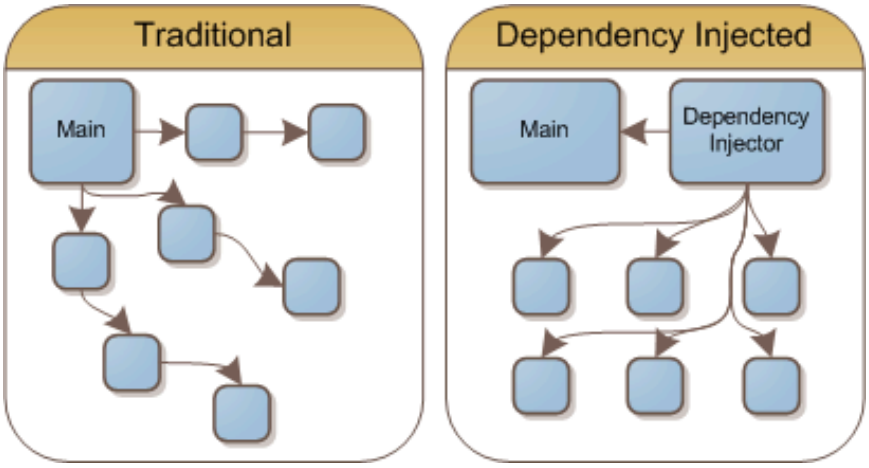
- wstrzykiwanie przez konstruktor (ang. *constructor injection*),
- wstrzykiwanie przez seter (ang. *setter injection*),
- wstrzykiwanie przez właściwość (ang. *property injection*),
- wstrzykiwanie przez wywołanie metody (ang. *method call injection*).

Wszystkie zależności przechowywane są w kontenerze typów zależności. Zanim jednak będą mogły być wstrzyknięte w obiekt, muszą najpierw zostać zarejestrowane w kontenerze. Rejestrację typów zwykle realizuje się poprzez przekazanie kontenerowi interfejsu oraz konkretnego typu implementującego ten interfejs.

Istnieją dwa sposoby na rejestrację w kontenerze:

- rejestracja jako typ – podczas żądania kontener tworzy instancję tego typu, którą przekazuje do wstrzykiwanego obiektu. W zależności od cyklu życia ustalonego w czasie rejestracji, tworzona jest nowa bądź przekazywana wcześniej stworzona instancja obiektu,
- rejestracja obiektów z pojedynczym (ang. *singleton*) cyklem życia – podczas żądania kontener zwraca referencję do wcześniej zarejestrowanej instancji obiektu bądź tworzy ją przy pierwszym żądaniu. Instancja obiektu tworzona jest tylko jeden raz na całe życie aplikacji.

Po zarejestrowaniu typu w kontenerze można pobrać go na żądanie lub wstrzyknąć jako zależność w wybrany obiekt. Jeśli w momencie żądania zachodzi potrzeba stworzenia przez kontener nowej instancji, tworzy ją i wstrzykuje w nią wszelkie zależności. Uproszczone porównanie koncepcji tworzenia zależności w sposób tradycyjny oraz przy pomocy wstrzykiwania zależności, przedstawiono na rys. IV-10.



Rysunek IV-10. Porównanie sposobów tworzenia zależności

Źródło: [43]

Instancje obiektów określonego typu tworzone przez kontener posiadają odpowiednio określony żywot. Jest to zależne od rodzaju cyklu życia wybranego podczas rejestracji typu.

Informacja o rodzaju cyklu życia instruuje kontener, jak ma postąpić w przypadku kolejnych żądań przekazania obiektu tego typu.

Istnieją trzy rodzaje cykli życia wstrzykiwanych obiektów:

- pojedynczy (ang. *singleton*) – obiekt instancji tworzy się tylko jeden raz na całe życie aplikacji,
- przejściowy (ang. *transistent*) – za każdym razem, kiedy następuje żądanie, tworzy się nowa instancja obiektu,
- zakresowy (ang. *scoped*) – oznacza tworzenie się nowej instancji obiektu w obrębie danej przestrzeni.

Źródło: Opracowano na podstawie [43]

4.5. Aplikacja Mobilna

Aplikację mobilną stworzono dla systemu operacyjnego Android. Nie wykorzystano jednak jego natywnego środowiska programistycznego, jakim jest Android Studio. W niniejszym podrozdziale opisano właśnie te technologie, które wykorzystano podczas realizacji prototypu w części mobilnej. Są jednocześnie alternatywą dla natywnych środowisk programistycznych ukierunkowanych na konkretną platformę mobilną.

4.5.1. Android

System operacyjny Android stworzono na potrzeby urządzeń mobilnych przez firmę Google. Obecnie jest najpopularniejszym systemem, pod którego kontrolą pracuje większość smartfonów i tabletów. Z tego powodu wybór środowiska pracy aplikacji mobilnej był dość oczywisty. Realizację prototypu zorientowano właśnie na ten system.

Źródło: Opracowano na podstawie [44]

4.5.2. Xamarin.Forms .NET Standard

Xamarin.Forms jest środowiskiem programistycznym wykorzystywanym do tworzenia wieloplatformowych aplikacji mobilnych. Posiada możliwość tworzenia i kompilowania wspólnego kodu pod wszystkie dostępne obecnie systemy operacyjne urządzeń mobilnych: Android, iOS, Windows Phone oraz UWP (ang. *Universal Windows Platform*). Xamarin.Forms oferuje dzięki temu bardziej ekonomiczne podejście do tworzenia kodu aplikacji niż Xamarin Native (np.: Xamarin.Android, Xamarin.iOS).

W Xamarin.Forms projekt programistyczny realizuje się we wspólnym kodzie. Specyficzne rozwiązania dla konkretnego systemu operacyjnego realizowane są w natywnych projektach. Łączy się je ze wspólnym kodem głównie przy pomocy interfejsów.

Na platformę programistyczną projektu wybrano .NET Standard. Zastąpiła wcześniej wykorzystywaną, lecz nierozwijaną już bibliotekę PCL (ang. ***P**ortable **C**lass **L**ibrary*). Podczas przygotowywania się do realizacji prototypu, okazało się, że .NET Standard nie jest zgodna ze wszystkimi komponentami, które miały być użyte w prototypie. Udało się jednak doprowadzić do ich użycia poprzez zastosowanie odpowiednich technik kompatybilności.

Źródło: Opracowano na podstawie [45] i [46]

4.5.3. XAML

XAML (ang. *e**X**tensible **A**pplication **M**arkup **L**anguage*) jest prostym językiem opisu wizualnych interfejsów opartym na języku znaczników XML. Stworzono go z myślą o technologii WPF, ale może być z powodzeniem stosowany do projektowania jakichkolwiek innych rozwiązań. XAML w Xamarin.Forms używa się do projektowania graficznych interfejsów oraz do zarządzania interakcją z użytkownikiem.

XAML jest alternatywnym sposobem do budowania graficznych interfejsów użytkownika. Te same obiekty mogą być stworzone i zainicjalizowane zarówno w kodzie aplikacji, jak i w XAML.

Deklarowanie interfejsu użytkownika w XAML ma jednak poniższe zalety:

- kod generowany przy pomocy XAML jest zwięzły i łatwy do czytania,
- odseparowanie logiki interfejsu od języka XAML pozwala jasno oddzielić prace programisty od projektanta interfejsu graficznego,
- odseparowanie kodu projektanta interfejsu od logiki biznesowej.

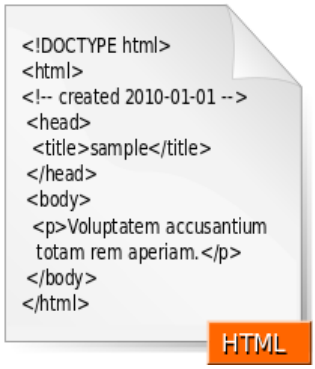
Źródło: Opracowano na podstawie [47]

4.6. Środowisko serwisów zewnętrznych

Realizacja środowiska serwisów zewnętrznych oparta jest na podstawowych technologiach tworzenia witryn internetowych. W niniejszym podrozdziale przedstawiono ich krótką charakterystykę.

4.6.1. HTML 5

HTML (ang. ***H**yper**T**ext **M**arkup **L**anguage*) jest to hipertekstowy język znaczników. Przy ich pomocy tworzy się szkielet strony internetowej. Tak opisaną stronę przetwarzają przeglądarki internetowe, wyświetlając odpowiednio sformatowaną treść. HTML jest obecnie standardem w tworzeniu treści stron internetowych.



Rysunek IV-11. Przykład najprostszej strony internetowej w HTML

Źródło: [48]

HTML można łączyć z innymi technologiami tworzenia stron. Przykład najprostszej strony internetowej zrealizowanej poprawnie w języku HTML przedstawiono na rys. IV-11.

Źródło: Opracowano na podstawie [48]

4.6.2. CSS

Kaskadowe arkusze stylów (ang. ***C**ascading **S**tyle **S**heets*, CSS) jest to zbiór reguł prezentacji treści. Kontrolują sposób, w jaki przeglądarka internetowa wyświetli zawartość wybranego elementu języka (X)HTML. Kontrola prezentacji elementów odbywa się na kilka sposobów. Przy pomocy styli elementów, klas styli lub identyfikację elementów poprzez jego rodzaj. CSS posiada większą kontrolę nad procesem pozycjonowania elementów składających się na stronę internetową niż HTML.

Źródło: Opracowano na podstawie [49]

4.6.3. PHP

PHP (ang. ***P**ersonal **H**ome **P**age*) jest interpretowanym, skryptowym językiem programowania. Przetwarzany jest po stronie serwera przez interpreter tego języka. Służy głównie do tworzenia stron internetowych i rozbudowanych aplikacji webowych.

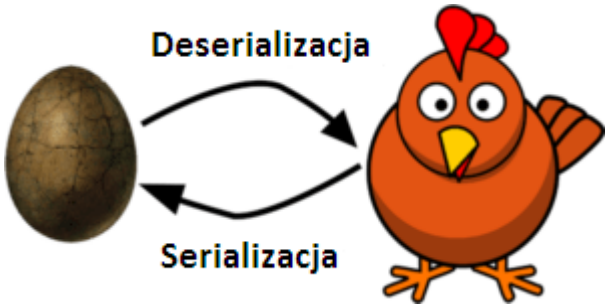
Źródło: Opracowano na podstawie [50]

4.7. Komponenty

Wszystkie komponenty opisane w niniejszym podrozdziale użyto w projektach webowym oraz mobilnym. Dostęp do nich zapewniony jest z menadżera pakietów NuGet wchodzącego w skład środowiska programistycznego .NET. W niniejszym podrozdziale przedstawiono krótką charakterystykę istotnych komponentów.

4.7.1. Json.NET

Jest to prosta biblioteka przewidziana dla platformy .NET. Pozwala na parsowanie i generowanie danych w formacie JSON. Realizowane jest to przy pomocy procesów serializacji i deserializacji obiektów. Zaprezentowano to na rys. IV-12.



Rysunek IV-12. Zobrazowanie procesu serializacji i deserializacji obiektu

Źródło: Opracowanie własne na podstawie [51]

Obiekty i struktury w środowisku .NET zapisywane są w formacie JSON i na odwrót. Proces mapowania danych wykonywany jest automatycznie przez tę bibliotekę.

Źródło: Opracowano na podstawie [51] i [52]

4.7.2. Entity Framework Core

Entity Framework (EF) w wersji Core to lekki, rozszerzalny oraz wieloplatformowy komponent dla platformy .NET. Służy do mapowania obiektowo-relacyjnego oraz do dostępu bazodanowego.

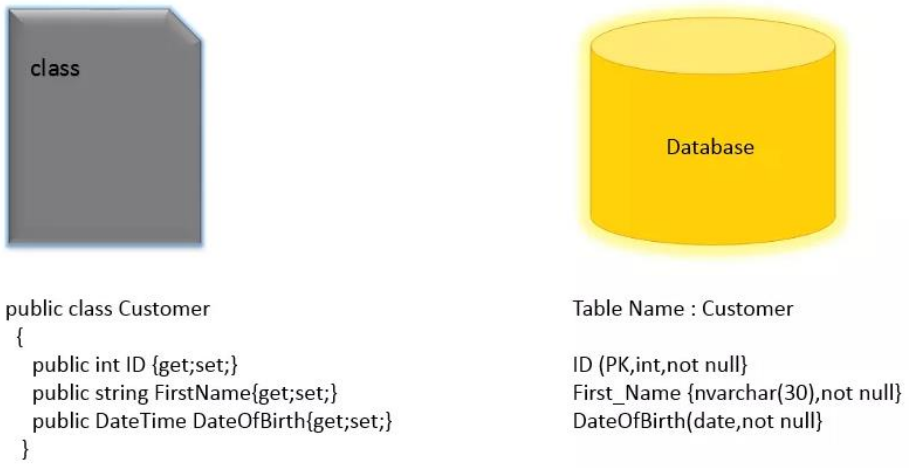
Dostęp do bazy danych realizowany jest przy pomocy modelu. Model ten posiada zmapowane klasy obiektów, zgodne z ich relacyjnym odbiciem w bazie danych.

Istnieją trzy sposoby na realizację mapowania:

- można stworzyć model z istniejącej bazy danych,

- można oprogramować go własnoręcznie, tak aby poprawnie odzwierciedlał struktury bazodanowe,
- do stworzenia bazy danych z istniejącego modelu można także użyć eksportu zaimplementowanego w EF.

Przykładowy sposób mapowania typów danych na struktury bazodanowe i na odwrót przedstawiono na rys. IV-13.



Rysunek IV-13. Mapowanie typów danych na struktury bazodanowe

Źródło: [53]

Dużą rolę w przetwarzaniu danych w EF stanowi LINQ (ang. ***L**anguage **IN**tegrated **Q**uery*). Technologia ta umożliwia zadawanie pytań na obiektach i otrzymywanie rezultatu w postaci np. kolekcji danych spełniających kryteria.

Do upraszczania kodu zapytań możliwe jest zastosowanie wyrażeń Lambda. Wyrażenia te to zwięzłe, anonimowe funkcje. Mają postać wyrażenia z przekazywanymi argumentami po lewej jego stronie i zwrotem po prawej.

Źródło: Opracowano na podstawie [53] i [54]

4.7.3. Autofac

Autofac jest rozbudowanym kontenerem odwróconego sterowania. Dostępny jest dla wszystkich platformy z rodziny .NET. Służy do rejestracji typów i pozyskiwania instancji obiektów na ich podstawie. Obsługuje wstrzykiwanie zależności przez konstruktor, właściwość oraz wywołanie metody. Komponent wykorzystano do rejestracji serwisów w postaci instancji obiektów w aplikacji mobilnej.

Źródło: Opracowano na podstawie [55]

4.7.4. Xam.Plugin.Media

Xam.Plugin.Media jest prostą wtyczką umożliwiającą wykonywanie zdjęć oraz nagrywanie filmów wideo z urządzeń mobilnych. Posiada także możliwość wybierania mediów z galerii tych urządzeń. Obsługuje większość aktualnie dostępnych systemów mobilnych takich jak Android, iOS oraz UWP.

Źródło: Opracowano na podstawie [56]

4.7.5. ZXing.Net.Mobile

ZXing.Net.Mobile jest biblioteką zrealizowaną na platformę .NET. Przeznaczona jest dla urządzeń mobilnych. Służy do skanowania i rozpoznawania kodów kreskowych (ang. *barcode*), ale również kodów 2D. Rozpoznaje większość kodów dwuwymiarowych m.in. DataMatrix, QR-Code oraz Aztec. W prototypie wykorzystano ją do identyfikowania produktów poprzez skanowanie kodów kreskowych. Wykonano ją na większość aktualnie dostępnych systemów mobilnych takich jak Android, iOS oraz Windows Phone.

Wtyczka nie jest kompatybilna z wybraną platformą programistyczną .NET Standard. Sprawiło to na początku niewielkie trudności z jej uruchomieniem. Udało się rozwiązać ten problem poprzez zastosowanie technik kompatybilności.

Źródło: Opracowano na podstawie [57]

4.8. Bazy danych

W niniejszym podrozdziale opisano silniki bazodanowe, które posłużyły do realizacji zarówno prototypu, jak i serwisów zewnętrznych. Obie bazy znajdują się na niezależnych serwerach umiejscowionych w przestrzeni publicznej.

4.8.1. Azure SQL Database

Azure SQL jest relacyjną bazą danych pracującą jako usługa w chmurze publicznej. Wykorzystuje silnik bazodanowy Microsoft SQL Server. Jest wydajna, niezawodna i bezpieczna. Można jej użyć do tworzenia aplikacji webowych, bez konieczności zajmowania się całą infrastrukturą zarządzania bazami danych. Zarządzanie nią odbywa się zdalnie przy pomocy narzędzi bazodanowych, np. Microsoft SQL Server Management Studio. Użyto jej do realizacji aplikacji webowej.

Źródło: Opracowano na podstawie [58]

4.8.2. MySQL

MySQL jest jedną z najpopularniejszych baz danych udostępnianą na licencji otwartego oprogramowania (ang. *open source*). Zapewnia łatwość obsługi, skalowalność i wysoką wydajność. Jest zgodna z właściwościami ACID. Daje to pewność, że transakcje są bezpieczne. Bazę danych wykorzystano do realizacji środowiska serwisów zewnętrznych.

Źródło: Opracowano na podstawie [59]

4.9. Narzędzia i środowiska pracy

Zastosowane w pracy narzędzia do realizacji prototypu związane są zarówno z usługami pracującymi w chmurze, jak i tymi użytkowanymi lokalnie. Z tego powodu oprócz narzędzi tradycyjnych, użyto także te, które są udostępnione zdalnie przez dostawcę usług. W niniejszym podrozdziale opisano narzędzia, ale i środowiska pracy, w których i przy pomocy których realizowano całość prototypu.

4.9.1. Azure

Platforma Azure jest kompleksowym zestawem usług znajdujących się w chmurze publicznej. Przy jej użyciu można wdrażać najróżniejsze aplikacje oraz nimi zarządzać. Na Azure będą pracowały jako usługa w sieci globalnej. Podczas wdrażania nowych rozwiązań wsparcie zapewniają zintegrowane narzędzia, metodyka DevOps oraz witryna Marketplace. Programiści mogą tworzyć i wdrażać od najprostszych aplikacji mobilnych po złożone rozwiązania internetowe. Jest to usługa płatna, hostowana przez firmę Microsoft.

Źródło: Opracowano na podstawie [60]

4.9.2. Azure Web Apps

Usługa Azure Web Apps daje możliwość tworzenia i rozwijania aplikacji webowych. Do jej użytkowania nie jest potrzebna znajomość środowiska jej pracy. Jej wieloplatformowa konstrukcja pozwala na obsługę systemu zarówno Windows, jak i Linux. Posiada także możliwość integracji z takimi usługami jak GitHub, Visual Studio Team Services czy też z dowolnym repozytorium Git. Oferuje automatyczne skalowanie oraz wysoką wydajność, jak i dostępność.

Źródło: Opracowano na podstawie [61]

4.9.3. Microsoft Visual Studio 2017

Jest to środowisko programistyczne firmy Microsoft. Pozwala na tworzenie wydajnych, kompleksowych rozwiązań na wiele platform użytkowych. Głównym językiem programowania, w którym się je realizuje, jest C#. Udostępnia także możliwość do używania takich rozwiązań jak AngularJS, Bootstrap, Node.js, Python lub R. Środowisko wykorzystano do realizacji aplikacji webowej oraz mobilnej.

Źródło: Opracowano na podstawie [62]

4.9.4. Microsoft SQL Server Management Studio 17

Microsoft SQL Server Management Studio (SSMS) jest zintegrowanym środowiskiem do zarządzania najróżniejszymi relacyjnymi bazami danych SQL. Posiada narzędzia do konfigurowania, zarządzania oraz monitorowania bazami danych. Nie ma znaczenia miejsce, w którym znajduje się baza danych. Mogą to być bazy zlokalizowane zarówno lokalnie, jak i w chmurze. SSMS użyto w pracy do zarządzania bazą danych Azure SQL.

Źródło: Opracowano na podstawie [63]

Rozdział V.

PRZEDSTAWIENIE IMPLEMENTACJI PROTOTYPU

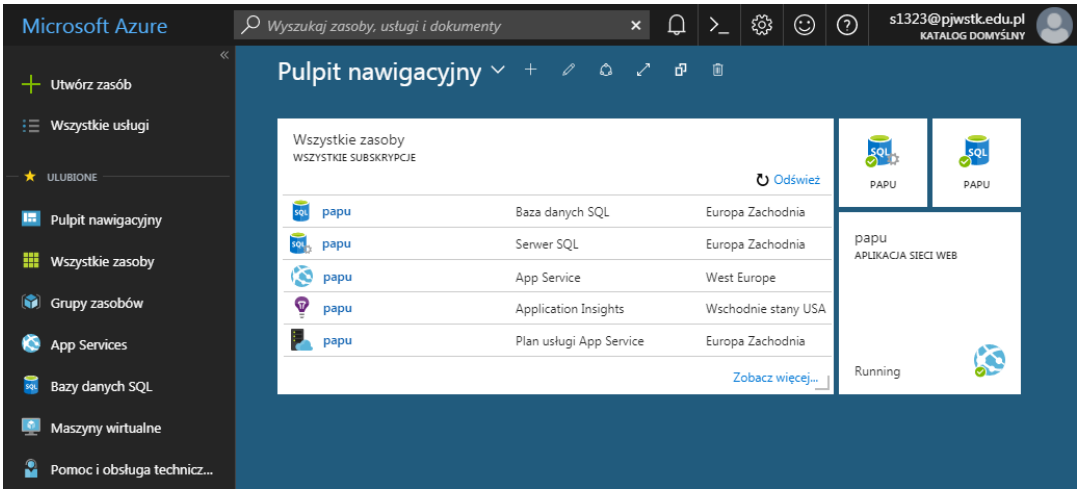
Rozdział ten w całości poświęcono implementacji prototypu dla proponowanego rozwiązania. Omówiono sposoby jego realizacji zarówno w kontekście jego architektury, wykonania ciekawszych jego części, jak i użytych rozwiązań technologicznych. Oprócz tego, opisano sposób realizacji interfejsu aplikacji webowej, mobilnej oraz sposoby integracji ze środowiskami systemów zewnętrznych.

5.1. Aplikacja webowa

Implementację aplikacji webowej stworzono w języku programistycznym C#. Do jej stworzenia wykorzystano środowisko Microsoft Visual Studio 2017. Konstrukcję projektu oparto na wzorcu projektowym MVC. Projekt aplikacji składa się z dwóch części: modułu Web API oraz części obsługiwanej przez przeglądarkę internetową.

5.1.1. Środowisko uruchomieniowe

Aplikacja webowa pracuje jako usługa internetowa w chmurze publicznej Microsoft Azure. Na całość działającej usługi składają się trzy elementy: serwer SQL, baza danych SQL oraz aplikacja sieci web. Zarządzanie nimi obywa się poprzez pulpit nawigacyjny, który przedstawiono na rys. V-1.



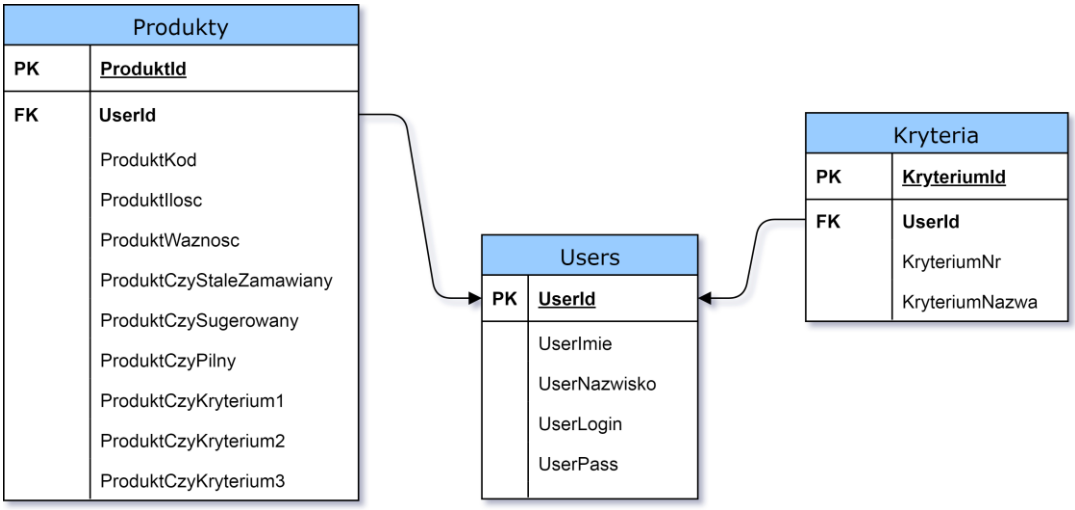
Rysunek V-1. Pulpit nawigacyjny platformy Azure

Źródło: Opracowanie własne

Środowisko programistyczne Visual Studio zorientowane jest na bezpośrednią współpracę z modułem aplikacji sieci web platformy Azure. Po skonfigurowaniu połączenia wdrożenie aplikacji wykonuje się poprzez wybranie odpowiedniego polecenia z menu kontekstowego. Znacznie ułatwia to czynności programistyczno-wdrożeniowe.

5.1.2. Baza danych

Aplikację webową oparto o relacyjny model bazy danych Azure SQL. Dostęp do niej możliwy jest tylko za pośrednictwem aplikacji webowej. Aplikacja mobilna nie posiada do niej bezpośredniego dostępu. Schemat bazy danych przedstawiono na poniższym rys. V-2.



Rysunek V-2. Schemat bazy danych systemu

Źródło: Opracowanie własne

Baza danych przechowuje tylko te dane, które bezpośrednio są związane z obsługą systemu. Wszelkie inne dane otrzymywane z zewnętrznych źródeł, takich jak serwisy zewnętrzne, nie są w niej składowane. Zarządzanie bazą danych było możliwe przy pomocy narzędzia Microsoft SQL Server Management Studio 17.

5.1.3. Usługa RESTful API

Realizację API systemu wykonano w architekturze REST. Udostępniony interfejs komunikacji podzielony jest na dwie części: wewnętrzny oraz przeznaczony do komunikacji z serwisami zewnętrznymi. Obie części działają na podobnej zasadzie przetwarzania danych związanych z komunikacją.

5.1.3.1. Sposób działania

Odpowiedź na wewnętrzne żądania generowana jest na podstawie informacji przechowywanych w bazie danych. Baza danych odpytywana jest przy pomocy modeli z użyciem komponentu Entity Framework Core. Komunikacja z serwisami zewnętrznymi odbywa się z wykorzystaniem wtyczek. Przekazywana jest im treść żądania w ustalonym formacie JSON. Zadaniem wtyczki jest przetworzenie tej treści, przekierowanie jej do serwisu zewnętrznego i zwrócenie odpowiedzi w akceptowalnym formacie JSON.

Przykład V-1. Wstrzykiwanie dostawcy wtyczek przez konstruktor

```
1 public ProductsBanksController(IPluginsProvider _plugingProvider)
```

Źródło: Opracowanie własne

Wtyczki dostarczane są poprzez wstrzykiwanie zależności konstruktorem kontrolera. Tą drogą przekazywany jest dostawca wtyczek, który zapewnia do nich dostęp. Wstrzykiwanie to przedstawiono w przykładzie V-1.

Przykład V-2. Pobranie określonej wtyczki i zwrot otrzymanego rezultatu

```
1 var apiService = PluginsProvider.GetWebApiProductsBank(_serviceGuid);
2 if (apiService == null)
3         return NotFound();
4 JObject request = new JObject
5 {
6         ["id"] = _id,
7 };
8 return await apiService.GetProduct(JsonConvert.SerializeObject(request));
```

Źródło: Opracowanie własne

W tym momencie pobierana jest odpowiednia wtyczka na podstawie przekazanego w żądaniu identyfikatora wtyczki *GUID*. Następnie wywoływana jest pożądana metoda wtyczki. Po odebraniu rezultatu dla otrzymanego żądania jest on zwracany do aplikacji mobilnej. Opisaną procedurę przedstawiono w przykładzie V-2.

5.1.3.2. Konstrukcja żądania i zwrotu

Do obsługi żądań wykorzystano pięć standardowych metod HTTP: *GET*, *POST*, *PUT*, *PATCH* oraz *DELETE*. Każda z nich oznacza wykonanie specyficznego polecenia po stronie systemu, jak i systemów zewnętrznych. Przykładowe żądania dla każdej z metod przedstawiono w tabeli V-1. Prefiksem dla ścieżki zasobu adresu URL w tych przykładach jest */api/v1/services/shops/{GUID}*. Pozostałą część adresu wpisano w kolumnę *Sufiks URL*.

Tabela V-1. Zestawienie przykładowych metod RESTful API

Metoda	**Sufiks URL**	**Opis**
GET	/ProductsCategories	Pobierz listę kategorii produktów.
POST	/ProductsInShoppingCard/?userId={USERID}	Wstaw produkt do koszyka.
PUT	/ShoppingCard/?userId={USERID}	Zrealizuj koszyk.
PATCH	/ProductsInShoppingCard/{ID}?userId={USERID}	Modyfikuj ilość produktu w koszyku.
DELETE	/ProductsInShoppingCard/{ID}?userId={USERID}	Usuń produkt z koszyka.

Źródło: Opracowanie własne

W ścieżce zasobu adresu URL żądania zawarta jest identyfikacja zasobu. Oprócz niej używane są także parametry z części zapytania. Metody *POST* i *PATCH* zawierają szczegółowe dane w treści żądania.

Zwrot odpowiedzi składa się z trzech części: *info*, *data* oraz *original*. W części *info* zawarta jest informacja o rezultacie żądania. Część *data* zawiera główną treść odpowiedzi. Odpowiedź oryginalna zwrócona przez serwis zewnętrzny wpisana jest w części *original*. Pozycja ta jest opcjonalna i nie jest dalej przetwarzana przez system. Przykładowe żądania oraz zwroty odpowiedzi opisane są w kolejnych podpunktach.

5.1.3.3. Mapowanie parametrów żądania

Wbudowane mechanizmy środowiska programistycznego dbają o automatyczne mapowanie parametrów żądania. Mapowanie wykonywane jest dzięki zastosowaniu identycznego nazewnictwa parametrów żądania oraz parametrów użytych w wywoływanej metodzie.

Przykład V-3. Żądanie z parametrami w części zapytania

```
/shops/{GUID}/Products/?categories={CATEGORIES}&
       keywords={KEYWORDS}&sort={SORT}&sortorder={SORTORDER}
```

Źródło: Opracowanie własne

Mapowanie powyżej przedstawionych w przykładzie V-3 parametrów żądania przedstawiono w przykładzie V-4. Przypisanie wartości dla parametru *_serviceGuid* zrealizowano w atrybucie kontrolera. Konstrukcję atrybutu przedstawiono w przykładzie V-22 na str. 76.

Przykład V-4. Mapowanie parametrów żądania na parametry metody

```
[HttpGet("Products")]
public async Task<IActionResult> GetProducts(
       Guid _serviceGuid,
       string categories,
       string keywords,
       string sort,
       string sortorder,
       [FromQuery]Dictionary<string, string> _query)
```

Źródło: Opracowanie własne

Część adresu URL określana jako zapytanie mapowana jest w całości poprzez atrybut *FromQuery*. Parametr dla mapowania zapytania przechowuje zbiór par nazwa/wartość. Jego użycie przedstawiono m.in. w przykładzie V-4.

Przykład V-5. Mapowanie parametrów przy pomocy atrybutów

```
public async Task<IActionResult> UpdateProductInShoppingCard(
       Guid _serviceGuid,
       int _id,
       int userId,
       [FromBody]JObject _body,
       [FromQuery]Dictionary<string, string> _query)
```

Źródło: Opracowanie własne

Treść znajdująca się w ciele żądania mapowana jest automatycznie do parametru metody poprzez przypisanie atrybutu *FromBody*. Mapowanie przy pomocy obu tych atrybutów przedstawiono powyżej w przykładzie V-5.

5.1.3.4. Przykładowe żądanie

Poniżej zaprezentowano przykład pełnego żądania, które w swoim ciele zawiera treść wraz z odpowiedzią na to żądanie. Przetwarzanie danych dla tego przykładu realizowane jest przez wtyczkę z dziedziny sklepy internetowe.

W ciele żądania umieszczona jest treść, którą zaprezentowano w przykładzie V-6. Zawiera ona dane, które posłużą do wykonania konkretnego polecenia po stronie sklepu internetowego. W omawianym przykładzie będzie to modyfikacja ilości zamawianego produktu znajdującego się w koszyku sklepowym.

Przykład V-6. Przykładowa treść żądania

```
{
  "data": {
    "koszyk": {
      "produkt": {
        "ZamowienieDetailIlosc": 24
      }
    }
  }
}
```

Źródło: Opracowanie własne

Po przesłaniu żądania jest ono wykonywane po stronie sklepu, a następnie generuje on odpowiedź z informacją o rezultacie wykonania polecenia. W odpowiedzi na żądanie zostaje zwrócony komunikat wyglądający tak, jak przedstawiono to w przykładzie V-7.

Przykład V-7. Przykładowa treść zwrotu

```
{
  "info": {
    "status": "SUCCESS",
    "code": 203,
    "description": "Data updated."
  },
  "data": {
    "koszyk": {
      "produkt": {
        "ProduktId": 19,
        "ZamowienieDetailIlosc": 24
      }
    }
  }
}
```

Źródło: Opracowanie własne

Tabela V-2. Informacje dotyczące żądania

Opis	Modyfikuj ilość produktu w koszyku.
Metoda	PATCH
Prefiks URL	/api/v1/services/shops/{GUID}
Sufiks URL	/ProductsInShoppingCard/{ID}?userId={USERID}

Źródło: Opracowanie własne

Wtyczka stworzona dla tego sklepu nie posiada opcjonalnej części *original*. Informacje na temat tego żądania, jak i sposób jego wywołania, znajdują się w powyższej tabeli V-2

5.1.3.5. Przykładowe żądanie do serwisu UpcItemDb

Przykładową część adresu URL żądania do komercyjnego serwisu UpcItemDb realizowanego z systemu przedstawiono w przykładzie V-8. Serwis posiada prosty interfejs oparty właściwie tylko na metodzie *GET*. Posłużyła ona do realizacji tego żądania.

Przykład V-8. Przykładowa część adresu żądania do serwisu UpcItemDb

```
/productsbanks/75185c7f-14a6-465e-ab74-1550c1c44cf1/Products/5907814664624
```

Źródło: Opracowanie własne

Przy jej pomocy można uzyskać szczegółowe informacje o produkcie na podstawie przesłanego numeru kodu paskowego.

Przykład V-9. Przykładowa treść zwrotu dla serwisu UpcItemDb

```
{
  "info": {
    "status": "SUCCESS",
    "code": 200,
    "description": null
  },
  "data": {
    "produkt": [
      {
        "ProduktId": 351610263822,
        "ProduktKatId": null,
        "ProduktNazwa": "Helix Halogen Outdoor Post Light 1.5w",
        "ProduktKod": "5906340131570",
        "ProduktZdjecieBinary": null
      }
    ]
```

```
  },
  "original": {
    "code": "OK",
    "message": null,
    "total": 1,
    "offset": 0,
    "items": [
      {
        "ean": "5906340131570",
        "title": "Helix Halogen Outdoor Post Light 1.5w",
        "description": "",
        "elid": "351610263822",
        "brand": "Techmar Garden Lighting",
        "model": "",
        "color": "",
        "size": "",
        "dimension": "",
        "weight": "",
        "lowest_recorded_price": 42.53,
        "highest_recorded_price": 42.53,
        "images": [],
        "offers": []
      }
    ]
  }
}
```

Źródło: Opracowanie własne

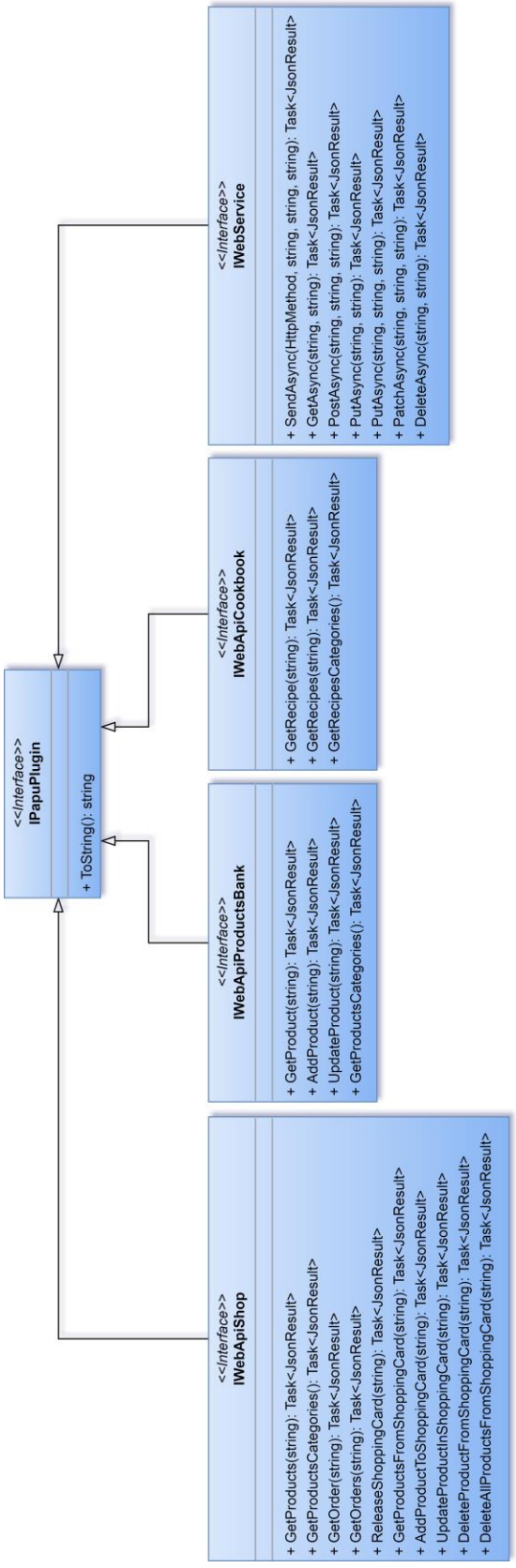
Odpowiedź na to żądanie przedstawiono w przykładzie V-9. Przygotowany zwrot odpowiedzi zawiera także część *original* z niezmodyfikowaną odpowiedzią uzyskaną z tego serwisu. Dane zawarte w tej części nie uczestniczą w procesie przetwarzania danych.

5.1.4. Realizacja wtyczek

Wtyczki znajdują się w osobnych plikach bibliotek umieszczonych w ustalonym katalogu. W kolejnych podpunktach opisano ich szczegółową konstrukcję.

5.1.4.1. Kontrakty

Wtyczki zbudowane są na podstawie kontraktów. Każda z grup zastosowań wtyczek posiada własny opis interfejsu. Wszystkie grupy implementują interfejs główny, którym jest *IPapuPlugin*. Każda grupa posiada listę metod przewidzianą dla wybranej dziedziny zastosowania. Opis wszystkich kontraktów przedstawiono na rys. V-3.



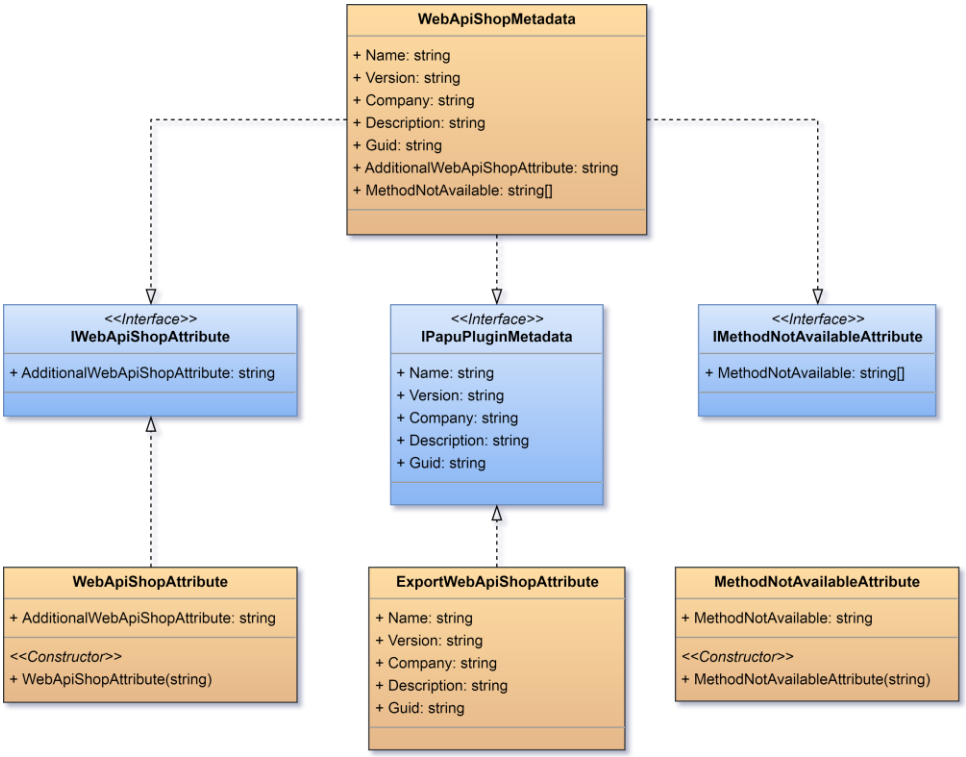
Rysunek V-3. Opis wszystkich kontraktów wtyczek

Źródło: Opracowanie własne

Plik kontraktów znajduje się w osobnym pliku w postaci biblioteki. Przygotowanie kontraktów w ten sposób pozwala na użycie ich przez osoby trzecie do budowania kolejnych wtyczek obsługujących integracje z nowymi serwisami.

5.1.4.2. Metadane wtyczek

Kontrakty posiadają własne opisy w postaci metadanych. Metadane w postaci atrybutów przypisane są do każdej klasy wtyczki. Przewidziano trzy rodzaje klas takich atrybutów: *Export*[*dziedzina*]*Attribute*, [*dziedzina*]*Attribute* oraz *MethodNotAvailableAttribute*. Użyte określenie *dziedzina* oznacza konkretną nazwę dziedziny zastosowań wtyczki. Stworzono cztery dziedziny: *WebApiShop*, *WebApiProductsBank*, *WebApiCookbook* oraz *WebService*.



Rysunek V-4. Opis klas metadanych dla przykładowego kontraktu

Źródło: Opracowanie własne

W przekazywaniu informacji o wtyczce każda z wymienionych klas atrybutów pełni odrębną funkcję:

- *Export*[*dziedzina*]*Attribute* – zestaw metadanych głównych, identyczny dla wszystkich wtyczek,
- [*dziedzina*]*Attribute* – metadane przewidziane są tylko i wyłącznie dla określonej dziedziny wtyczek,

- *MethodNotAvailableAttribute* – atrybut dostarcza informację o niedostępnych metodach interfejsu wtyczki.

Klasą, która służy podczas eksportu wtyczki do scalenia wszystkich atrybutów, jest [*dziedzina*]*Metadata*. Klasa ta implementuje wszystkie interfejsy atrybutów. Zależności pomiędzy powyżej opisywanymi klasami przestawiono na rys. V-4. Ze względu na to, że atrybut *MethodNotAvailableAttribute* może być przypisany do klasy wtyczki kilka razy, nie implementuje interfejsu *IMethodNotAvailableAttribute*. Właściwość *MethodNotAvailable* tego interfejsu użyta jest podczas eksportu wtyczki do przypisania tablicy w postaci *string[]*. Tablica zawiera dane ze wszystkich oznaczeń atrybutem *MethodNotAvailableAttribute*. Atrybut ten jest typu *string*.

Przykład V-10. Atrybut z możliwością kilkukrotnego przypisania

```
1 [MetadataAttribute]
2 [AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
3 public class MethodNotAvailableAttribute : Attribute
```

Źródło: Opracowanie własne

Oznaczenie klasy atrybutu jako atrybutu z możliwością kilkukrotnego przypisania do tej samej klasy wtyczki, realizowane jest poprzez przypisanie wartości *true* dla właściwości *AllowMultiple*. Przypisanie to przedstawiono w przykładzie V-10.

5.1.4.3. Eksport wtyczek

W celu eksportu klasy wtyczki należy przypisać jej odpowiednie atrybuty opisane w poprzednim podpunkcie.

Przypisanie możliwości eksportu klasy wtyczki zrealizowano na dwa sposoby:

1) Poprzez sam interfejs, co zostało przedstawione w poniżej prezentowanym przykładzie V-11.

2) Poprzez interfejs oraz nazwę kontraktu, co zostało przedstawione w kolejnym przykładzie V-12.

Przykład V-11. Oznaczenie eksportu przez interfejs

```
1 public ExportWebApiShopAttribute(string _name, string _version,
2        string _company, string _description, string _guid)
3        : base(typeof(IWebApiShop))
```

Źródło: Opracowanie własne

Przykład V-12. Oznaczenie eksportu przez interfejs oraz nazwę kontraktu

```
public ExportRestApiAttribute(string _name, string _version,
       string _company, string _description, string _guid)
       : base("RestApi", typeof(IWebService))
```

Źródło: Opracowanie własne

Ze względu na użycie nazw kontraktów dla dziedziny *WebService*, nazwy atrybutów metadanych posiadają dwa rozwinięcia dla tej dziedziny: *RestApi* oraz *Soap*. Identyfikacja wtyczek dla tej dziedziny na podstawie nazw kontraktów spowodowana była posiadaniem przez nie tego samego interfejsu. Uniemożliwiało to identyfikację jedynie przy pomocy samego interfejsu. Użycie wszystkich stworzonych atrybutów, które składają się na metadane przekazywane o wtyczce, przedstawiono w przykładach V-13 oraz V-14.

Przykład V-13. Przypisanie metadanych dla serwisu UpcItemDb

```
[ExportWebApiProductsBank("UpcItemDb", "6.2", "UPCITEMdb.com",
       "Globalny bank kodów kreskowych produktów.",
       "2c8e7ac4-4416-4f35-bc90-0b091d4b1237")]
[WebApiProductsBank(null)]
[MethodNotAvailable("AddProduct")]
[MethodNotAvailable("GetProductCategories")]
[MethodNotAvailable("UpdateProduct")]
public class UpcItemDb : IWebApiProductsBank
```

Źródło: Opracowanie własne

Przykład V-14. Przypisanie metadanych z użyciem atrybutu dziedziny

```
[ExportRestApi("Rest json", "1.2", "PapuService",
       "Komunikacja przy pomocy REST Api Json.",
       "f3cb500a-b9db-4a71-ba13-06b2eaae528b")]
[WebService(EnumContentType.Json)]
public class RestJson : IWebService
```

Źródło: Opracowanie własne

Export[*dziedzina*] jest głównym atrybutem wtyczki posiadającym w sobie przypisane dane służące do jej identyfikowania podczas importu.

5.1.4.4. Wstrzykiwanie zależności podczas importu wtyczek

We wtyczkach wykorzystano wstrzykiwanie zależności, podczas którego to wstrzykiwania, wstrzykiwany jest obiekt innej wtyczki. Zrealizowane jest to przy pomocy wstrzykiwania poprzez konstruktor. Mechanizm ten użyto do dostarczenia wtyczkom z danej dziedziny, wtyczki związanej z obsługą komunikacji.

Przykład V-15. Wstrzykiwanie wtyczki poprzez konstruktor innej wtyczki

```
[ImportingConstructor]
public CoJaJem([ImportMany("RestApi")]IEnumerable<Lazy<IWebService,
       WebServiceMetadata>> _webServices)
```

Źródło: Opracowanie własne

Wtyczki importowane są jako klasy z opóźnioną inicjalizacją. Jest to tzw. inicjalizacja leniwa (ang. *Lazy*). Oznacza to, że wtyczka zostanie zainicjalizowana dopiero w momencie pierwszego jej wykorzystania. Leniwa inicjalizacja wymuszona jest pobieraniem metadanych o wtyczkach. Wszystkie wtyczki importowane są właśnie w ten sposób. Opisane w tym podpunkcie sposoby importu przedstawiono w przykładzie V-15.

5.1.5. Zarządzanie wtyczkami

Zarządzanie wtyczkami rozłożono na trzy klasy funkcyjne działające jako serwisy. Pobranie wszystkich wtyczek następuje dopiero w momencie pierwszego żądania o którąkolwiek z nich. Klasą inicjującą import wtyczek jest klasa dostawcy *PluginsProvider*. Za import wtyczek odpowiedzialna jest bezpośrednio klasa *PluginsLoader*.

5.1.5.1. Import wtyczek

Import wtyczek wykonywany jest przy pomocy serwisu *PluginsLoader*. Wtyczki ładowane są z katalogu *Plugins* poprzez pobranie listy plików bibliotek. Miejsce przechowywania wtyczek na dysku można określić w pliku konfiguracyjnym aplikacji *appsettings.json*.

Przykład V-16. Pobranie listy typów do importu poprzez użycie refleksji

```
var pluginTypesIsUsed =
       (
              from properties in
                     (
                            typeof(PluginsProvider).GetProperties()
                                   .ToList()
                     )
              from att in properties.GetCustomAttributes<PluginAttribute>()
              where att.IsUsed
              select properties.PropertyType.GenericTypeArguments[0]
       )
       .ToList();
```

Źródło: Opracowanie własne

Następnie przy użyciu biblioteki MEF importowane są tylko te typy, które oznaczono do eksportu. To czy wtyczka zostanie zaimportowana podlega dodatkowo dwóm innym restrykcjom. Pierwsza z nich filtruje typy na podstawie wpisów w pliku konfiguracyjnym *appsettings.json*. Druga importuje tylko te typy, które istnieją w postaci właściwości w klasie dostawcy *PluginsProvider* oraz posiadają wpisaną wartość *true* w atrybucie *IsUsed*.

Lista typów wtyczek pobierana jest z klasy *PluginsProvider* poprzez użycie refleksji. Ze względu na ich deklarację jako typ *Lazy* pobierany jest pierwszy typ generyczny z tego typu. Określa on właściwy typ wtyczki. Zaimportowane typy wtyczek przekazywane są do serwisu *PluginsProvider*. Opisane powyżej pobranie listy typów z użyciem refleksji przedstawiono w przykładzie V-16.

5.1.5.2. Dostawca wtyczek

Za dostarczanie wtyczek odpowiada klasa *PluginsProvider*. Realizowane jest to przy pomocy wstrzykiwania zależności głównie w konstruktor klas kontrolerów API. Serwis dostawcy *PluginsProvider* posiada metody do bezpośredniego pobierania wtyczek z serwisów kontenerów wtyczek *PluginsContainer* oraz *PluginsContainerGeneric*.

Przykład V-17. Pobranie wtyczki z użyciem identyfikatora GUID

```
1 public IWebApiProductsBank GetWebApiProductsBank(Guid _guid)
2 {
3         return WebApiProductsBank.Where(p =>
4                 p.Metadata.Guid == _guid.ToString()).FirstOrDefault()?.Value;
5 }
```

Źródło: Opracowanie własne

Serwis dostawcy wtyczek posiada możliwość pobrania wtyczek na podstawie identyfikatora *GUID* wtyczki lub jej nazwy. Dodatkowo stworzono metody pozwalające na pobranie listy wtyczek lub listy ich metadanych. Lista generowana jest dla konkretnej dziedziny wtyczek. Przykładowe pobranie wtyczki z użyciem identyfikatora *GUID* przedstawiono w powyższym przykładzie V-17.

5.1.5.3. Generyczne przechowywanie wtyczek

W celu zaprezentowania możliwości wykonania alternatywnych rozwiązań stworzono dwie różne klasy kontenerów przechowujących wtyczki. Obie klasy działają niezależnie i są identyczne pod względem udostępnionych metod. *PluginsProvider* jest w stanie użytkować obie klasy, bez konieczności modyfikowania ich kodu.

Przykład V-18. Istotne części klasy PluginsContainer

```
public class PluginsContainer : IPluginsContainer
{
        private List<object> Plugins { get; set; }
…
        public void Register<T>(IEnumerable<T> _plugin)
        {
                Plugins.AddRange((IEnumerable<object>)_plugin);
        }

        public IEnumerable<Lazy<T, TMetadata>> GetPlugins<T, TMetadata>()
        {
                return GetPluginsData<Lazy<T, TMetadata>>(() =>
                                typeof(Lazy<T, TMetadata>), d => d);
        }

        public IEnumerable<T> GetPlugins<T>()
        {
                return GetPluginsData<T>(() =>
                PluginsFunctions.GetPluginLazyType<T>(), d => d.Value);
        }

        public IEnumerable<IPapuPluginMetadata> GetPluginsInfo<T>()
        {
                return GetPluginsData<IPapuPluginMetadata>(() =>
                PluginsFunctions.GetPluginLazyType<T>(), d => d.Metadata);
        }

        private IEnumerable<T> GetPluginsData<T>(
                Func<Type> _lazyPluginType,
                Func<dynamic, dynamic> _getDynamicData)
        {
                List<T> returnTypes = new List<T>();
                var filteredTypes = Plugins.Where(p =>
                        p.GetType() == _lazyPluginType());
                filteredTypes.ToList()
                        .ForEach(p =>
                        {
                                dynamic dyn = Convert.ChangeType(
                                                        p, _lazyPluginType());
                                returnTypes.Add(_getDynamicData(dyn));
                        });
                return returnTypes;
        }
}
```

Źródło: Opracowanie własne

Pierwszą z nich jest *PluginsContainer*. Posiada metody generyczne do rejestracji oraz pobierania wtyczek. Wtyczki przechowywane są w postaci listy obiektów. Każda wtyczka

to osobny obiekt. Stworzona w tej klasie metoda *GetPluginsData* służy do pobierania wtyczek bezpośrednio z listy. Najważniejsze metody klasy przedstawiono w przykładzie V-18.

Przykład V-19. Metoda GetPluginsData z klasy PluginsContainerGeneric

```
private IEnumerable<T> GetPluginsData<T>(
        Func<Type> _lazyPluginType,
        Func<dynamic, dynamic> _getDynamicData)
{
        List<T> returnTypes = new List<T>();
        var filteredTypes = Plugins.Where(p =>
                p.GetType() == GetPluginsContainerListGenericType(
                _lazyPluginType())).FirstOrDefault();
        var filteredTypes2 = (IEnumerable)typeof(PluginsFunctions)
                .GetMethod("GetPluginsContainerListGenericList")
                .MakeGenericMethod(_lazyPluginType())
                .Invoke(this, new object[] { filteredTypes });

        foreach (var value in filteredTypes2)
        {
                dynamic dyn = Convert.ChangeType(value, _lazyPluginType());
                returnTypes.Add(_getDynamicData(dyn));
        }
        return returnTypes;
}
```

Źródło: Opracowanie własne

Alternatywnym rozwiązaniem do przechowywania wszystkich wtyczek jest stworzona klasa *PluginsContainerGeneric*. Posiada podobne metody generyczne jak wcześniej opisana klasa. Metody publiczne z prefiksem *GetPlugins*, służące do pobierania wtyczek, są dokładnie identyczne. Inaczej skonstruowano jednak metodę *GetPluginsData*. Wynika to z innego sposobu przechowywania wtyczek. Wygląd metody przedstawiono w przykładzie V-19.

Przykład V-20. Klasa generyczna PluginsContainerListGeneric<T>

```
[Serializable]
public class PluginsContainerListGeneric<T> : PluginsContainerListBase,
        IPluginsContainerListGeneric<T>, IList<T>, ICollection<T>,
        IEnumerable<T>, IReadOnlyList<T>, IReadOnlyCollection<T>
{
        private List<T> Plugins { get; set; }
…
}
```

Źródło: Opracowanie własne

Zasadnicza różnica polega jednak na sposobie, w jakim wtyczki są przechowywane. W tym celu stworzono klasę w pełni generyczną *PluginsContainerListGeneric<T>*. Klasa

ta jest serializowalną listą generyczną implementującą m.in. interfejs *IList<T>*. Służy jako kontener do składowania wtyczek tego samego typu. Deklarację klasy oraz listy generycznej przedstawiono w przykładzie V-20.

W celu przechowywania klas *PluginsContainerListGeneric<T>* w klasie kontenera, stworzono w nim właściwość *Plugins*. Właściwość ta to lista z zadeklarowanym abstrakcyjnym typem klasy *PluginsContainerListBase*. Klasa *PluginsContainerListGeneric<T>* dziedziczy po tej klasie. Deklarację klasy kontenera przedstawiono w przykładzie V-21.

Przykład V-21. Klasa PluginsContainerGeneric

```
1 public class PluginsContainerGeneric : IPluginsContainerGeneric
2 {
3         private List<PluginsContainerListBase> Plugins { get; set; }
4 …
5 }
```

Źródło: Opracowanie własne

Oba alternatywne rozwiązania klas kontenera wtyczek służą do ich przechowywania i udostępniania na życzenie dostawcy wtyczek.

5.1.6. Routing

W aplikacji występują trzy rodzaje mapowania tras. Wybór sposobu realizacji trasowania był uwarunkowany od rodzaju oprogramowywanego zagadnienia.

5.1.6.1. Mapowanie dla REST API

Routing żądań modułu API oparto na mapowaniu poprzez kontroler oraz akcję. Oznaczenie routingu na kontrolerze wykonano przy pomocy własnego atrybutu. Kod klasy atrybutu przedstawiono w przykładzie V-22, a przypisanie do kontrolera w przykładzie V-23.

Przykład V-22. Własny atrybut do routingu żądań

```
1 public class ApiServicesV1RouteAttribute : Attribute, IRouteTemplateProvider
2 {
3         public string Template =>
4                         "api/v1/services/[controller]/{_serviceGuid:guid}";
5         public int? Order { get; set; }
6         public string Name { get; set; }
7 }
```

Źródło: Opracowanie własne

Atrybut ten przypisano do każdego kontrolera dziedziny wtyczki. Istnieją inne sposoby wykonania routing, ale ten opisany poniżej wydał się najwłaściwszy. W szablonie ścieżki do zasobów *Template* użyto dwa parametry. Pierwszym z nich jest *controller*, który przekierowuje żądanie bezpośrednio do kontrolera o identycznej nazwie jak wartość tego parametru. Drugim parametrem jest *_serviceGuid* typu *GUID*, który przekazywany jest bezpośrednio do metody wywoływanej z kontrolera. Tym samym nazwa kontrolera dziedziny ma bezpośredni wpływ na wyznaczanie trasy routingu.

Przykład V-23. Przypisanie atrybutu routingu do kontrolera dziedziny

```
1 [ApiServicesV1Route]
2 public class ProductsBanksController : Controller
```

Źródło: Opracowanie własne

Ostatecznie żądanie przekierowywane jest automatycznie do odpowiedniej metody. To czy dana metoda zostanie wywołana dla żądania uwarunkowane jest od szablonu adresu wpisanego w atrybut akcji oraz użytej metody HTTP. Przedstawiono to w przykładzie V-24.

Przykład V-24. Routing poprzez atrybut metody

```
1 [HttpGet("Products/{_id:regex(^[[0-9]]{{1,30}}$)}")]
2 public async Task<IActionResult> GetProduct(…)
```

Źródło: Opracowanie własne

Szablon adresu, który dla omawianego w tym podpunkcie mapowania wywoła metodę *GetProduct*, powinien wyglądać tak, jak przedstawiono w przykładzie V-25.

Przykład V-25. Przykładowy adres dla omawianego przykładu

```
1 api/v1/services/ProductsBanks/75185c7f-14a6-465e-ab74-1550c1c44cf1
2       /Products/5907814664624
```

Źródło: Opracowanie własne

Istotną sprawą podczas realizacji routingu jest użycie odpowiedniej metody HTTP. Pozwala to na jednoznaczne wyznaczenie trasy do metody kontrolera, która przewidziana jest dla otrzymanego żądania.

5.1.6.2. Mapowanie dla stron systemu

Strony systemu wykorzystują wbudowany w środowisko mechanizm mapowania trasy żądania poprzez użycie kontrolera oraz akcji. Mechanizm ten jest podobny do tego opisywanego dla API systemu.

Przykład V-26. Wpis określający szablon adresu

```
app.UseMvc(routes =>
{
        routes.MapRoute(
                name: "default",
                template: "{controller=Home}/{action=Index}/{id?}");
…
}
```

Źródło: Opracowanie własne

Nie wykorzystano jednak atrybutu routingu ani na kontrolerze, ani na metodach kontrolera. W aplikacji trasa wyznaczana jest na podstawie samych nazw kontrolerów oraz ich metod. Powoduje to jednak pewne ograniczenie i uzależnienie adresu strony od nazewnictwa przyjętego dla klas i metod. Wpis konfiguracyjny określający szablon adresu przedstawiono w powyższym przykładzie V-26.

5.1.6.3. Mapowanie dla strony API systemu

Stronę z zestawieniem API systemu stworzono w technologii stron Razor. Technologia ta nie wykorzystuje wbudowanego routing, ponieważ nie opiera się na kontrolerze. Dostęp do stron wykonanych tą metodą jest bezpośredni, wyznaczony fizyczną lokalizacją pliku strony w drzewie katalogów witryny.

Przykład V-27. Przypisanie domyślnego katalogu stron Razor

```
services
        .AddRazorPagesOptions(options =>
        {
                options.RootDirectory = "/Pages";
        });
```

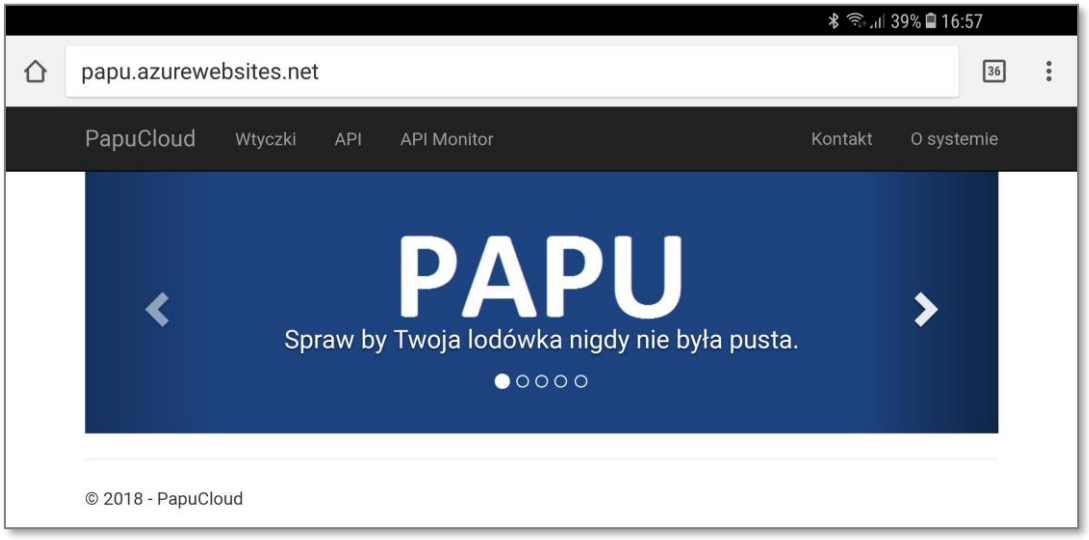
Źródło: Opracowanie własne

Wpis przedstawiony w przykładzie V-27 wymagany jest w przypadku zmiany domyślnej nazwy katalogu dla stron Razor. Tym samym dostęp do zrealizowanej strony poświęconej API systemu został uwarunkowany od umiejscowienia pliku strony w tymże katalogu.

5.1.7. Wizualizacja systemu

Wizualizację systemu stworzono na dwa sposoby. Większą część witryny oparto na wzorcu MVC. Stronę dotyczącą API prototypu stworzono w postaci strony Razor. Witrynę prototypu wykonano w celu zaprezentowania funkcjonalności aplikacji webowej. Przy jej

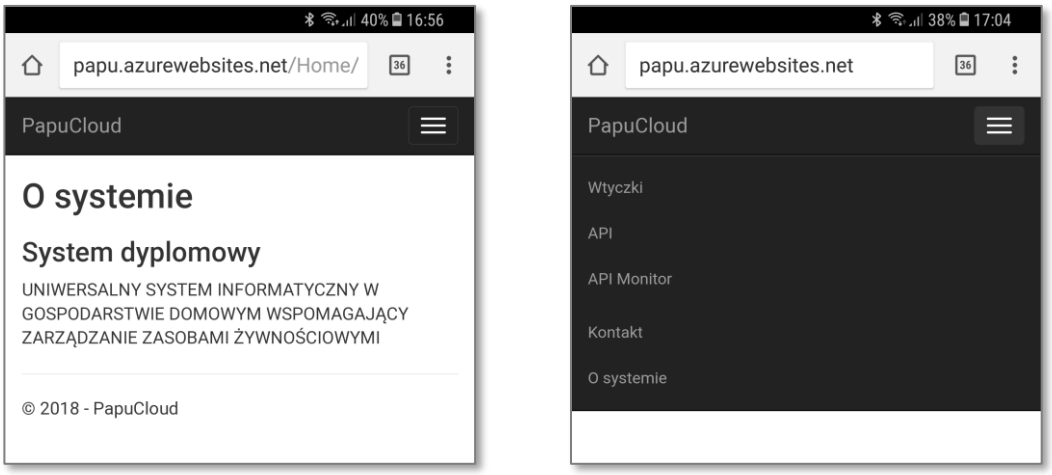
projektowaniu wykorzystano responsywną technikę tworzenia stron. Technika ta pozwoliła na stworzenie stron, które poprawnie wyświetlają się zarówno na urządzeniach mobilnych, jak i w przeglądarkach komputerowych.



Rysunek V-5. Strona tytułowa systemu

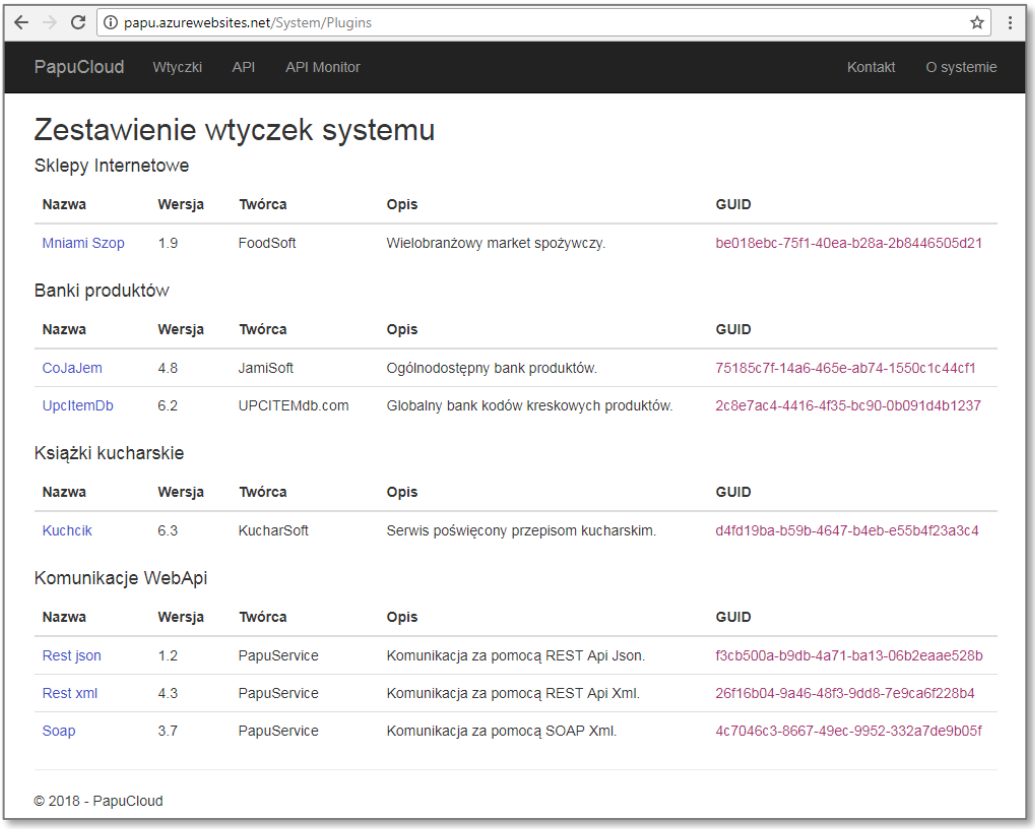
Źródło: Opracowanie własne

Oprócz treści informacyjnej o systemie, w witrynie można także odczytać informacje o aktualnie używanych wtyczkach, przejrzeć zestawienie udostępnionego interfejsu API oraz monitorować komunikację do systemu wykonywaną z urządzeń mobilnych. Na kolejnych stronach na rys. V-5, V-6, V-7, V-8 oraz V-9, zaprezentowano wygląd wszystkich stron witryny systemu zarówno z przeglądarek mobilnych, jak i komputerowych.



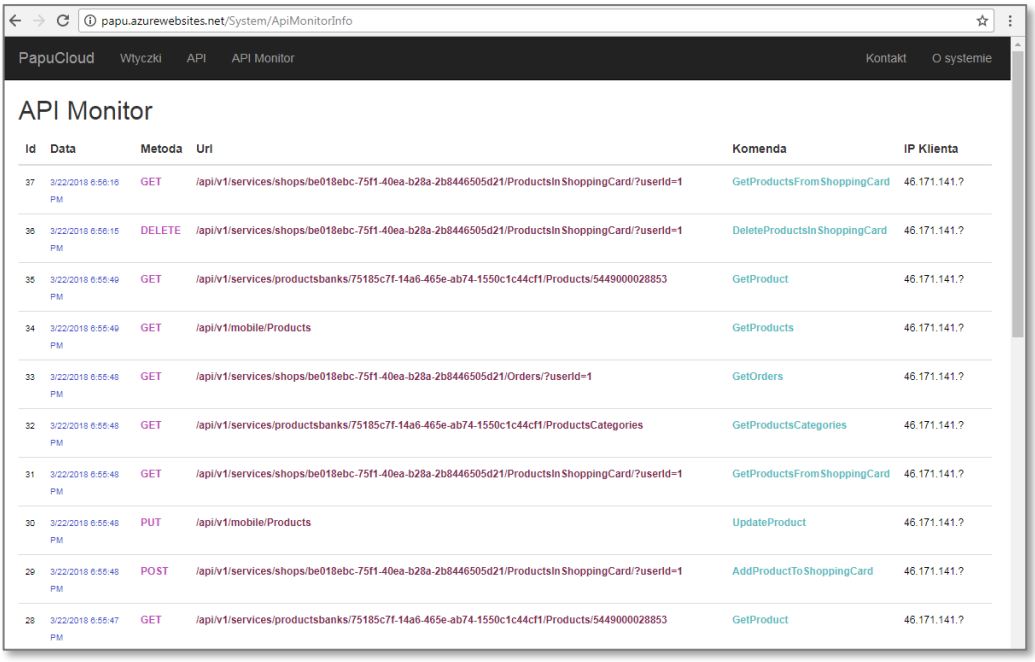
Rysunek V-6. Wygląd witryny w wersji mobilnej

Źródło: Opracowanie własne



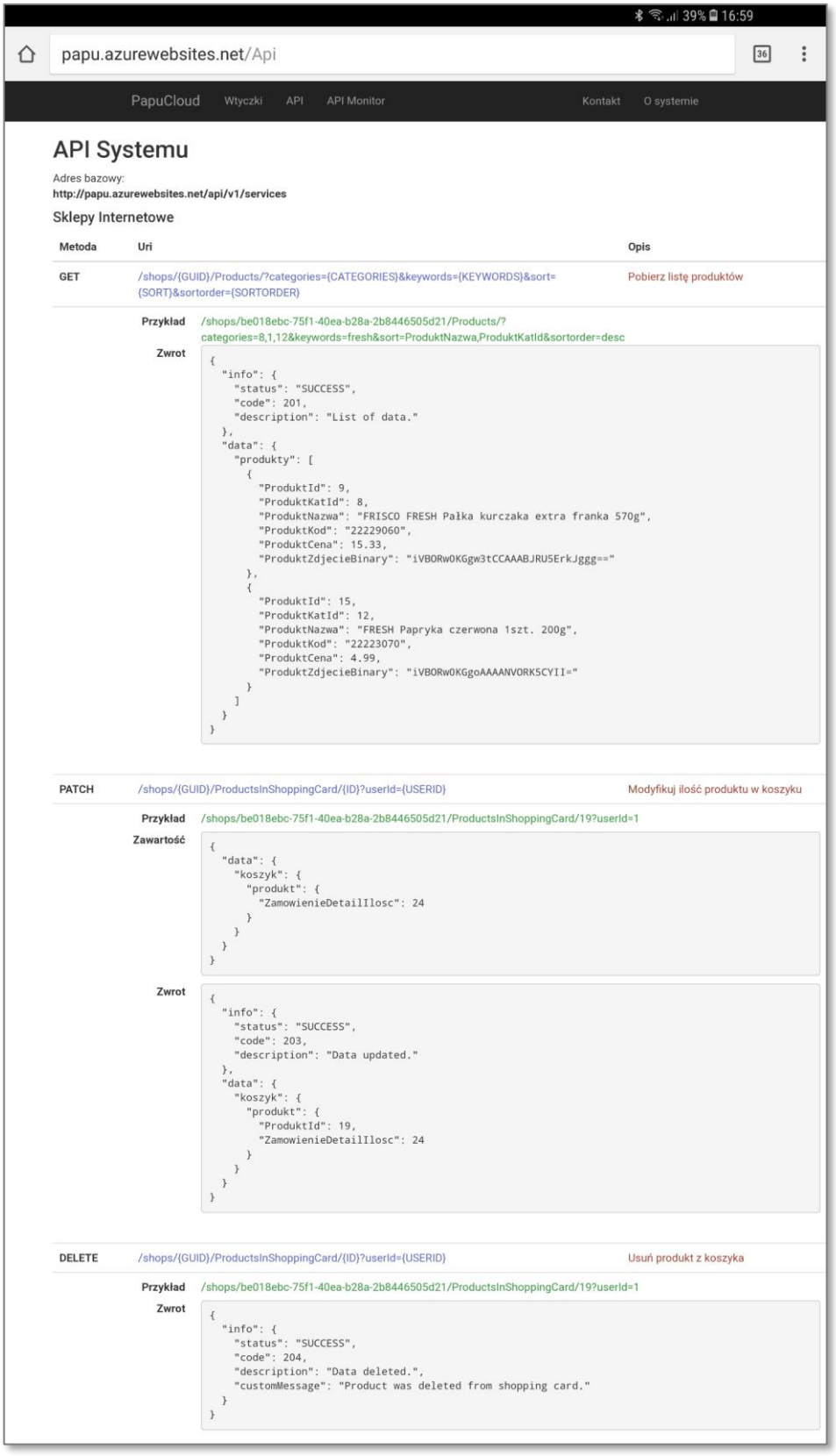
Rysunek V-7. Zestawienie wtyczek systemu

Źródło: Opracowanie własne



Rysunek V-8. Monitoring żądań kierowanych do API systemu

Źródło: Opracowanie własne



Rysunek V-9. Fragment zestawienia API systemu

Źródło: Opracowanie własne

5.2. Aplikacja mobilna

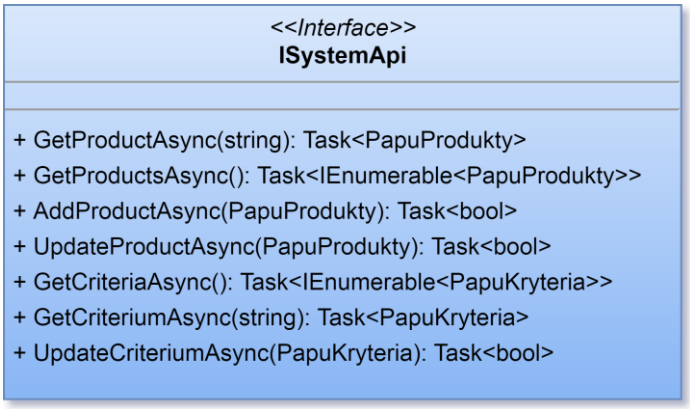
Podobnie jak implementację aplikacji webowej, aplikację mobilną stworzono w języku programistycznym C#. Do jej stworzenia również wykorzystano środowisko Microsoft Visual Studio 2017. Zamiast natywnego środowiska programistycznego dla systemu operacyjnego Android, jakim jest Android Studio, wykorzystano środowisko programistyczne Xamarin.Forms. Konstrukcję projektu oparto na wzorcu projektowym MVVM. Aplikacja składa się z dwóch połączonych ze sobą projektów: projektu wspólnego kodu oraz projektu z kodem dedykowanym dla systemu operacyjnego Android.

5.2.1. Komunikacja

Aplikacja mobilna komunikuje się tylko i wyłącznie z opisywanym w poprzednim podrozdziale API aplikacji webowej. Kierowane są do niego żądania związane zarówno z wewnętrzną komunikacją, jak i te dotyczące serwisów zewnętrznych. Interfejsy do obydwóch zasobów są niezależne. Przedstawiono je na rys. V-10 oraz V-11. Żądania wewnętrzne kierowane są na adres zasobu z prefiksem */api/v1/mobile*, natomiast żądania obsługujące serwisy zewnętrzne posiadają prefiks */api/v1/services*. Pozostała część adresu do zasobu bezpośrednio identyfikuje żądanie. Metody HTTP określają rodzaj żądania.

Rysunek V-10. Interfejs komunikacji zewnętrznej IServicesApi

Źródło: Opracowanie własne



Rysunek V-11. Interfejs komunikacji wewnętrznej ISystemApi

Źródło: Opracowanie własne

W celu realizacji żądań skonstruowano wtyczkę *RestJson*, której zadaniem jest wysyłanie i odbieranie danych komunikacji. Usprawniła przetwarzanie danych, sprowadzając metody obsługujące te żądania do kilku linijkowych zapisów. Ustawiane są w nich dane wejściowe, wywoływana jest odpowiednia metoda wtyczki w zależności od adekwatnej do żądania metody HTTP oraz odebrane są dane zwrotu. Wszystkie metody żądań do API systemu posiadają podobny schemat wywołania. Metodę przedstawiono w przykładzie V-28.

Przykład V-28. Wysłanie i odbiór żądania dla konkretnego zasobu

```
1  public async Task<bool> ShopsUpdateProductInShoppingCard(
2         string _serviceGuid, int _id, int _userId,
3         ApiServicesProductShopContent _content)
4  {
5         try
6         {
7                var contentTo = _content.ToJson();
8                var query = $"?userId={_userId}";
9                var url = Constants.ServicesApiUrl + $"/shops/{_serviceGuid}"
10                       + $"/ProductsInShoppingCard/{_id}{query}";
11               var restJson = App.Container.Resolve<RestJson>();
12               var response = await restJson.PatchAsync(
13                       Constants.RestUrlBase, url, contentTo);
14               var content = await response.Content.ReadAsStringAsync();
15               var result = JsonConvert.
16                       DeserializeObject<ApiServicesResponse>(content);
17
18               if (!response.IsSuccessStatusCode)
19                      return response.IsSuccessStatusCode;
20               return await Task.FromResult(response.IsSuccessStatusCode);
21        }
22        catch { return false; }
23 }
```

Źródło: Opracowanie własne

5.2.2. Realizacja funkcjonalności

W aplikacji mobilnej użyto cztery rodzaje komponentów, które pozwoliły na realizację przewidzianych zadań. W kolejnych podpunktach opisano te rozwiązania.

5.2.2.1. Skaner

Głównym sposobem dostarczania informacji o produktach jest wykorzystanie skanera. Jego obsługę zrealizowano w oparciu o bibliotekę ZXing.Net.Mobile.Forms. Po wywołaniu wbudowanego w bibliotekę ekranu skanowania kodów kreskowych pojawia się czerwona linia. Po nakierowaniu aparatu urządzenia mobilnego na kod kreskowy jest on automatycznie odczytywany i w rezultacie przesyłany poprzez zdarzenie. Rezultat zwiera odczytany kod kreskowy. Skanowanie produktu przedstawiono na rys. V-16 na str. 89.

5.2.2.2. Wykonywanie zdjęć

Dostarczanie zdjęć produktów do aplikacji wykonano poprzez użycie biblioteki Xam.Plugin.Media. Biblioteka daje możliwość zarówno wykonania zdjęcia poprzez domyślną aplikację aparatu lub wybrania zdjęcia z galerii mediów. W aplikacji mobilnej wykorzystano obie te funkcjonalności, dając użytkownikowi możliwość wyboru, z którego zasobnika będzie chciał dodać zdjęcie dla wprowadzanego produktu. Po automatycznym uruchomieniu domyślnej aplikacji aparatu i wykonaniu zdjęcia, biblioteka daje możliwość zatwierdzenia zdjęcia lub ponowne jego wykonanie. Po wykonaniu zdjęcia zwracany jest obraz w postaci typu *Stream*. Wykonanie zdjęcia przedstawiono na rys. V-18 na str. 91.

5.2.2.3. Powiadomienia

W celu realizacji powiadomień użyto prostą bibliotekę Xam.Plugins.Notifier. Posiada dwie metody *Show* oraz *Cancel*. Pierwsza z nich służy do wyświetlenia, a druga do anulowania wcześniej wyświetlonego powiadomienia na ekranie urządzeń mobilnych. Sposób ich użycia przedstawiono w poniższych przykładach V-29 oraz V-30. Przykładowy wygląd powiadomień przedstawiono na rys. V-35 na str. 108.

Przykład V-29. Wyświetlenie powiadomienia

```
1 Plugin.LocalNotifications.CrossLocalNotifications.Current.Show(
2       "Termin ważności", "Minął termin ważności produktów !!!", msg);
```

Źródło: Opracowanie własne

Przykład V-30. Anulowanie powiadomienia

```
1 Plugin.LocalNotifications.CrossLocalNotifications.Current.Cancel(msgOld);
```

Źródło: Opracowanie własne

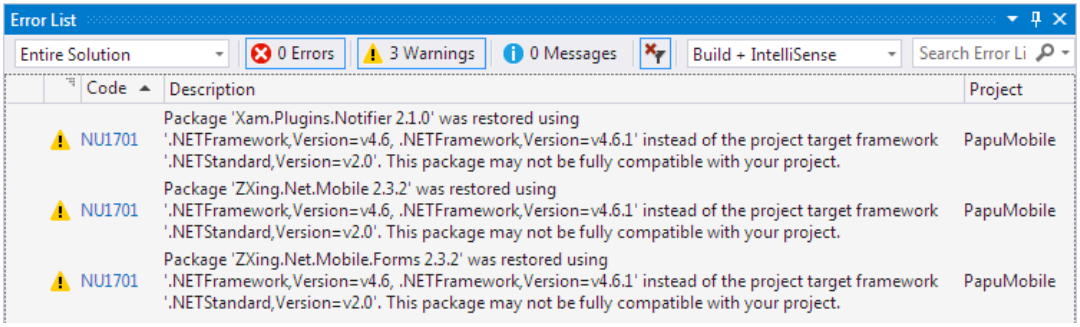
Każde powiadomienie posiada unikalny numer. Wykorzystywany jest on w celu anulowania poprzednio wyświetlonego powiadomienia. Dzieje się to w przypadku, kiedy zachodzi potrzeba wyświetlenia kolejnego powiadomienia tego samego rodzaju, ale zawierającego inną treść.

5.2.2.4. Przechowywanie danych

Aplikacja mobilna nie przechowuje żadnych danych związanych z przetwarzanymi przez siebie informacjami dotyczącymi produktów. Wszelkie wyświetlane przez nią informacje dostarczane są na bieżąco z zewnętrznych źródeł. Niemniej jednak zaistniała potrzeba do przechowywania podstawowych informacji dotyczących pracy systemu, jak np. numer ostatnio wyświetlonego powiadomienia. Do przechowywania tego rodzaju informacji wykorzystano bibliotekę Xam.Plugins.Settings. Zapisuje ona informacje związane z aplikacją w jej przestrzeni zasobów. Po odinstalowaniu aplikacji dane te są automatycznie usuwane.

5.2.3. Niekompatybilność komponentów

Wykorzystane biblioteki ZXing.Net.Mobile/.Forms oraz Xam.Plugins.Notifier sprawiły na początku trudności z ich uruchomieniem. Powodem był brak ich kompatybilności ze środowiskiem .NET Standard. Bardziej szczegółowe informacje na temat napotkanych trudności podczas realizacji całego prototypu przedstawiono całościowo w rozdziale 6.3 na str. 117. Komunikaty ostrzeżeń ze środowiska Visual Studio przedstawiono na rys. V-12.



Rysunek V-12. Ostrzeżenia dotyczące niekompatybilności komponentów

Źródło: Opracowanie własne

5.2.4. Obsługa systemu

Obsługa systemu podzielona jest na trzy obszary tematyczne. Dotyczą zarządzaniem zawartością lodówki, realizacją zamówień na produkty oraz pozyskiwaniem przepisów wraz z informacją o posiadaniu produktów potrzebnych do ich realizacji. Dodatkowo aplikacja posiada możliwość konfiguracji systemu oraz informację o programie. W kolejnych podpunktach opisano sposoby realizacji typowych czynności, które dostępne są z aplikacji.

5.2.4.1. Nawigacja systemu

Nawigację systemu wykonano z wykorzystaniem szablonu *MasterDetailPage*. Jest to standardowy sposób realizacji poruszania się po aplikacji w systemach Android. Wygląd ekranu startowego oraz wysuwane menu zaprezentowano na poniższym rys. V-13.

Rysunek V-13. Ekran startowy oraz wysuwane menu systemu

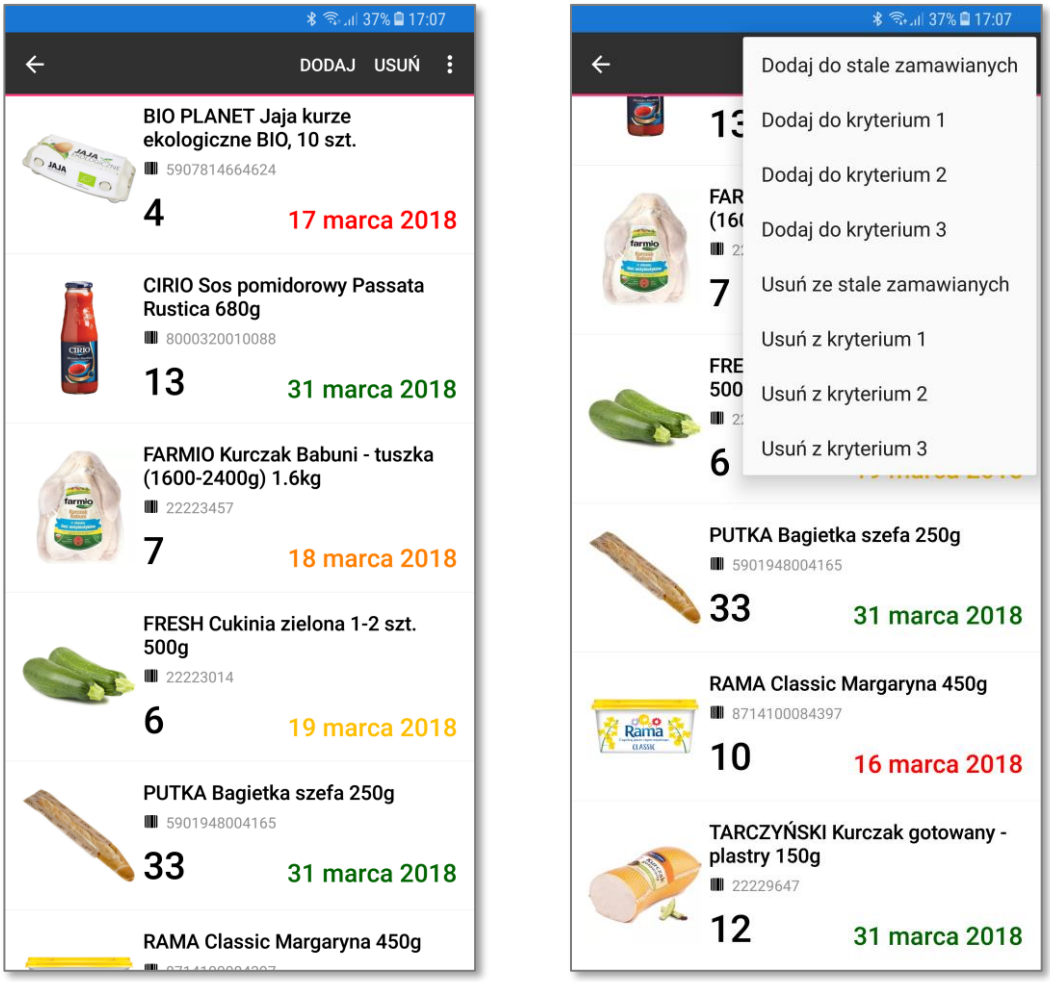
Źródło: Opracowanie własne

5.2.4.2. Obszar Lodówka

W obszarze tym użytkownik ma możliwość wykonywania typowych czynności związanych z zarządzaniem lodówką. Główna jego część przedstawia listę produktów, które znajdują się w lodówce. Oprócz tego z panelu można wykonywać dodatkowe czynności. Można m.in. dodawać, usuwać oraz sprawdzać stan produktów. Dodatkowo istnieje możliwość oznaczenia braku produktu. Czynności te inicjuje się poprzez dotknięcie z menu narzędziowego odpowiedniej ikony. Oznaczają one w kolejności: dodawanie, usuwanie oraz sprawdzanie produktów. Istnieje także menu drugiego rzędu, zobrazowane jako trzy pionowe kropki, pozwalające na oznaczanie braku produktów. Większość czynności w tym obszarze realizuje się poprzez skanowanie kodu kreskowego produktów. Wygląd ekranu lodówki przedstawiono na rys. V-14. Na kolejnych stronach niniejszej pracy przedstawiono sposoby poruszania się, jak i wykonywania czynności w obrębie tego obszaru.

Rysunek V-14. Wyświetlenie zawartości lodówki

Źródło: Opracowanie własne

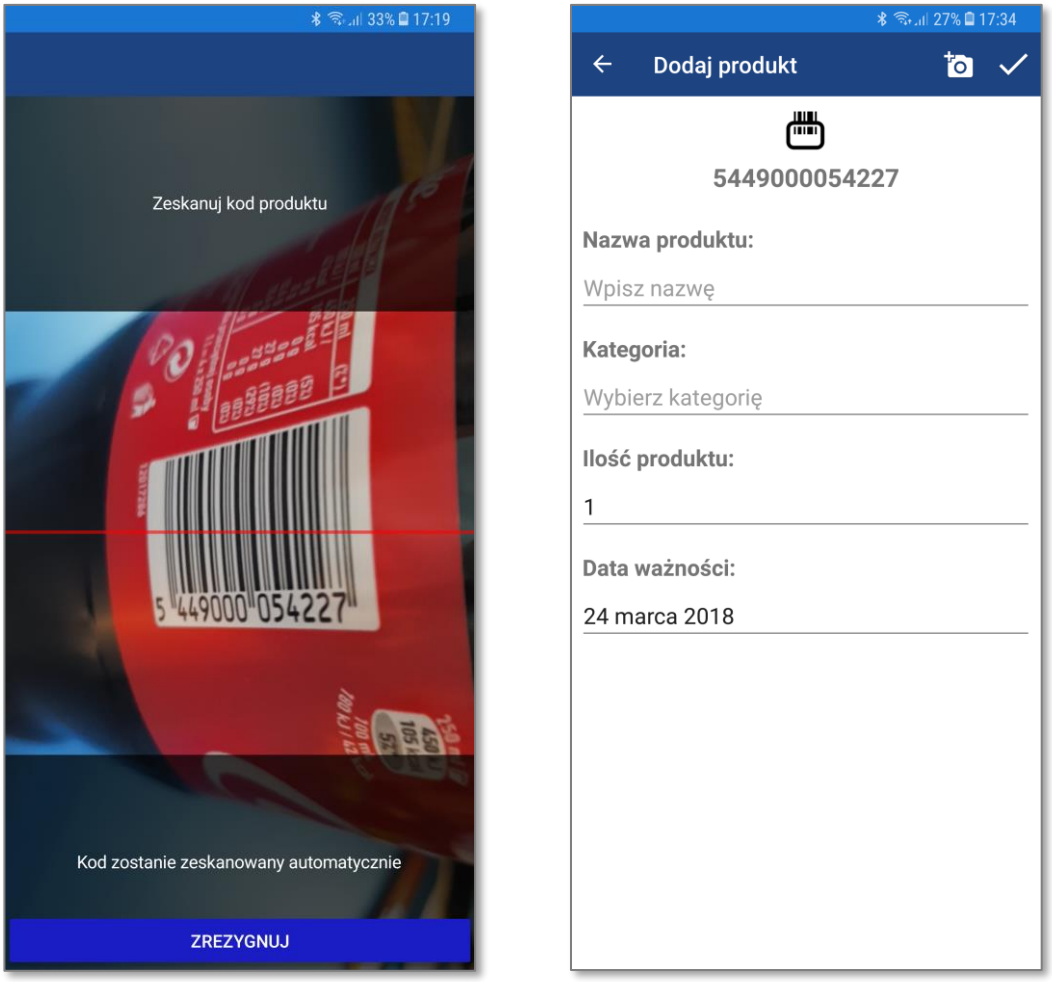


Rysunek V-15. Wybór opcji z menu kontekstowego dla produktu

Źródło: Opracowanie własne

Podstawowymi wyświetlanymi informacjami o produkcie jest jego nazwa, kod kreskowy, ilość w lodówce oraz termin ważności. W przypadku ilości produktu oraz terminu ważności zastosowano odpowiednią kolorystykę tych informacji. I tak kolorem zielonym oznaczono stan poprawny. Odcienie żółtego oznaczają różne stopnie ostrzeżeń o zbliżającym się niedługo terminie przydatności do spożycia lub niskim stanie produktu. Kolorem czerwonym oznaczono bardzo niski stan produktu lub produkt przeterminowany.

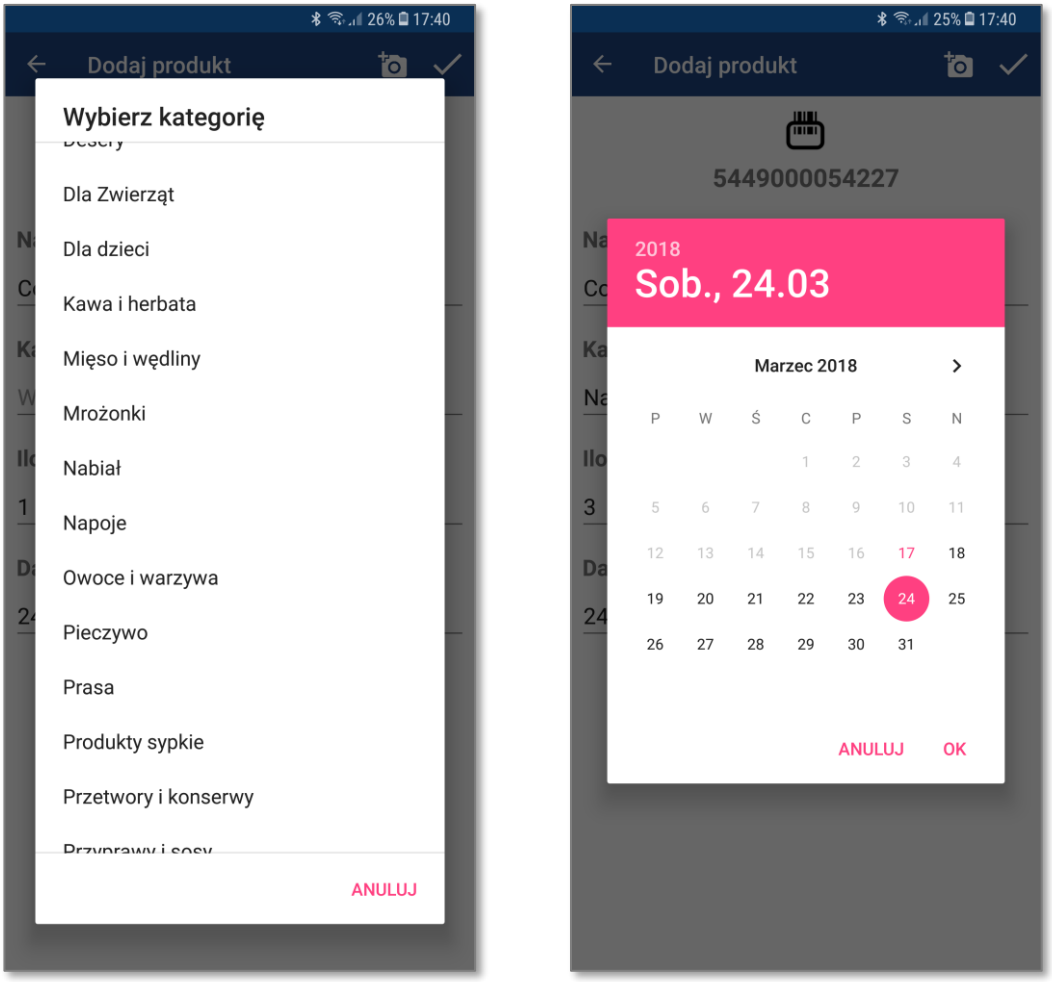
Wybierając produkt i przytrzymując jego zaznaczenie można wyświetlić menu kontekstowe. Dostępne są w nim podstawowe opcje takie jak dodawanie oraz usuwanie produktu. W menu drugiego rzędu udostępnione są opcje pozwalające na dodawanie lub usuwanie z kryteriów. Kryteria te służą w obszarze *Zamówień* do podjęcia decyzji o konieczności zakupu danego produktu. Menu kontekstowe przestawiono na powyższym rys. V-15.



Rysunek V-16. Skanowanie produktu w celu dodania do lodówki

Źródło: Opracowanie własne

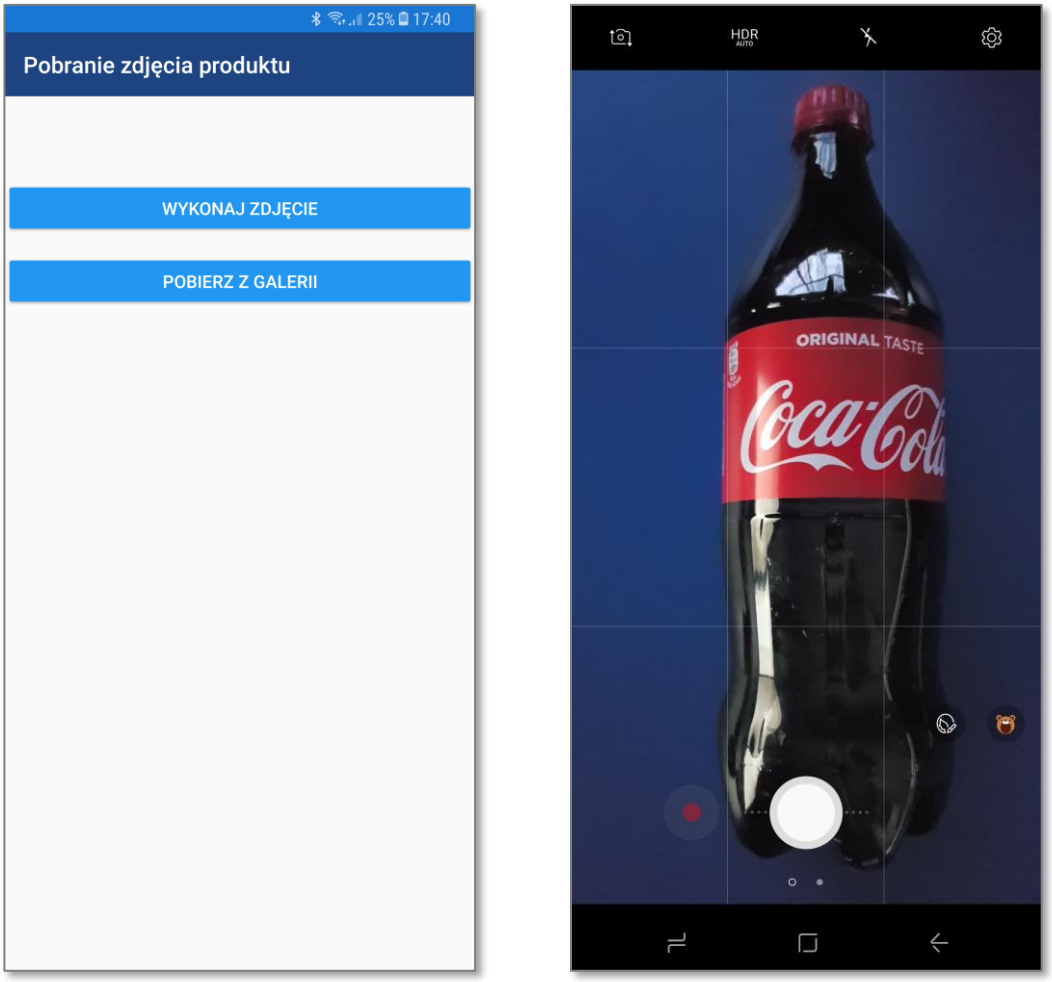
W momencie wybrania opcji dodania produktu wyświetla się ekran służący do odczytu kodu kreskowego. Można zrezygnować z wykonywania tej czynności lub naprowadzić aparat urządzenia mobilnego na kod kreskowy w celu jego odczytu. Rozpoznawanie kodu realizowane jest automatycznie. Po jego odczycie pojawia się ekran z polami do wypełnienia. Wszystkie wyświetlone pola są wymagane. Pola te to w kolejności: nazwa produktu, kategoria, ilość produktu oraz data ważności. Jeśli produkt znajduje się w serwisach udostępniających bazy produktów, to pola, nazwa produktu oraz kategoria, uzupełniane są automatycznie. Istnieje także możliwość dołączenia zdjęcia produktu w przypadku jego braku. Dodanie produktu akceptuje się poprzez dotknięcie ikony zapisu w menu narzędziowym. Opisywaną funkcjonalność zobrazowano na powyższym rys. V-16.



Rysunek V-17. Wybór kategorii oraz daty

Źródło: Opracowanie własne

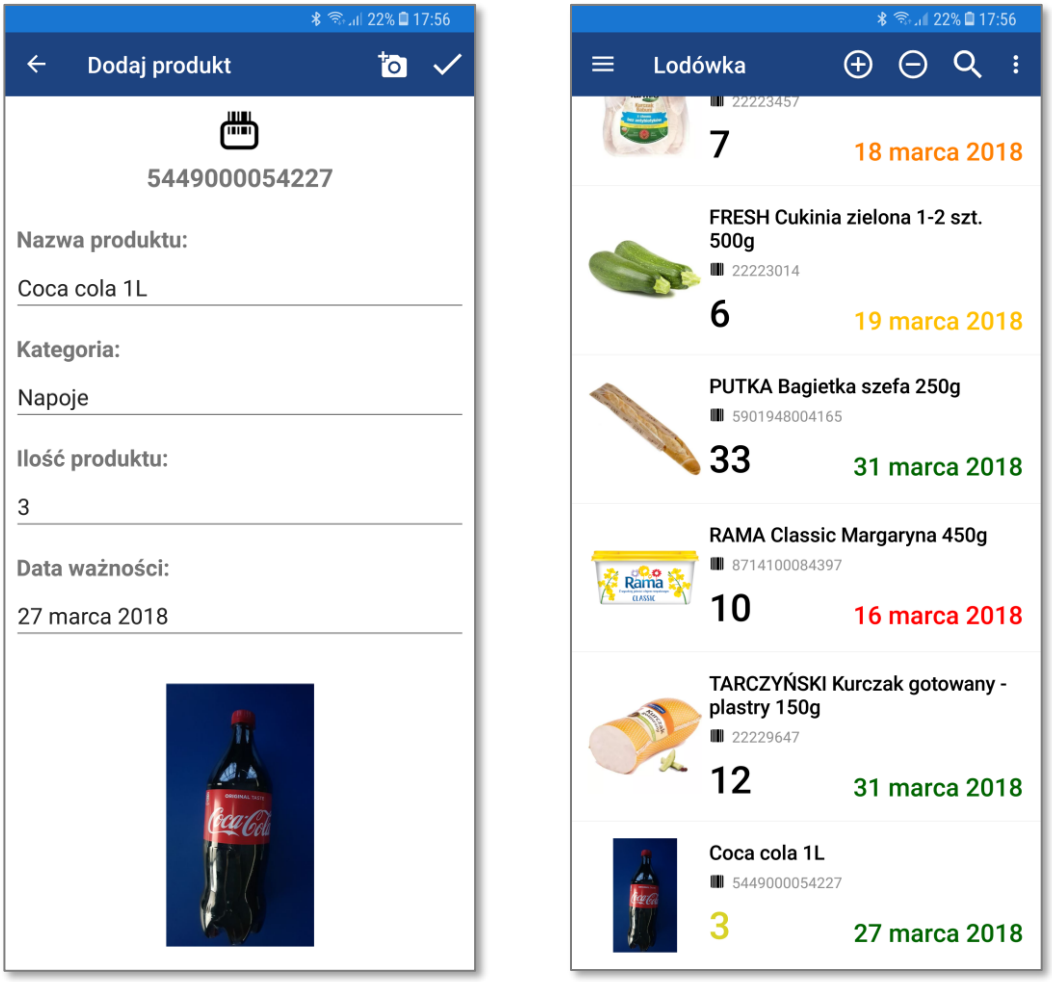
Jednym z wymaganych pól jest uzupełnienie kategorii dla wprowadzanego produktu. Kategorie te pozyskiwane są z serwisów posiadających bazy produktów. Istnieje także możliwość określenia daty terminu ważności produktu z kalendarza. Obie opisane funkcjonalności przedstawiono na powyższym rys. V-17.



Rysunek V-18. Wybór sposobu dostarczenia zdjęcia oraz jego wykonanie

Źródło: Opracowanie własne

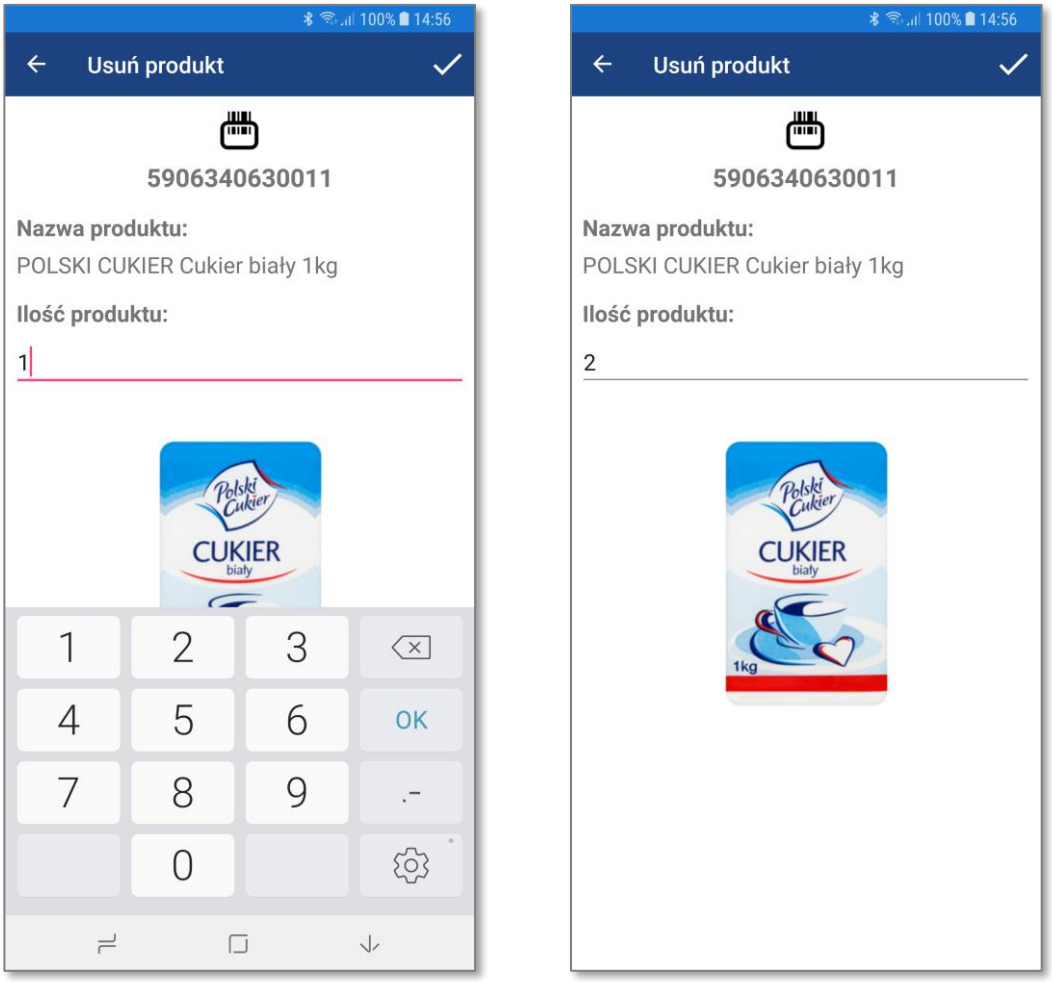
Uruchamiając opcję dodawania zdjęcia produktu, użytkownik otrzymuje możliwość na jego wykonanie przy pomocy aparatu lub dodanie go z galerii mediów. Wybór ten dokonuje się poprzez wybranie odpowiedniej opcji na wyświetlonym ekranie. W przypadku wybrania opcji wykonania zdjęcia pojawia się ekran domyślnej aplikacji aparatu. Po zrobieniu zdjęcia pojawia się prośba o jego akceptację lub ponowne wykonanie. Realizację zdjęć przedstawiono na powyższym rys. V-18.



Rysunek V-19. Finalne dodanie produktu do lodówki

Źródło: Opracowanie własne

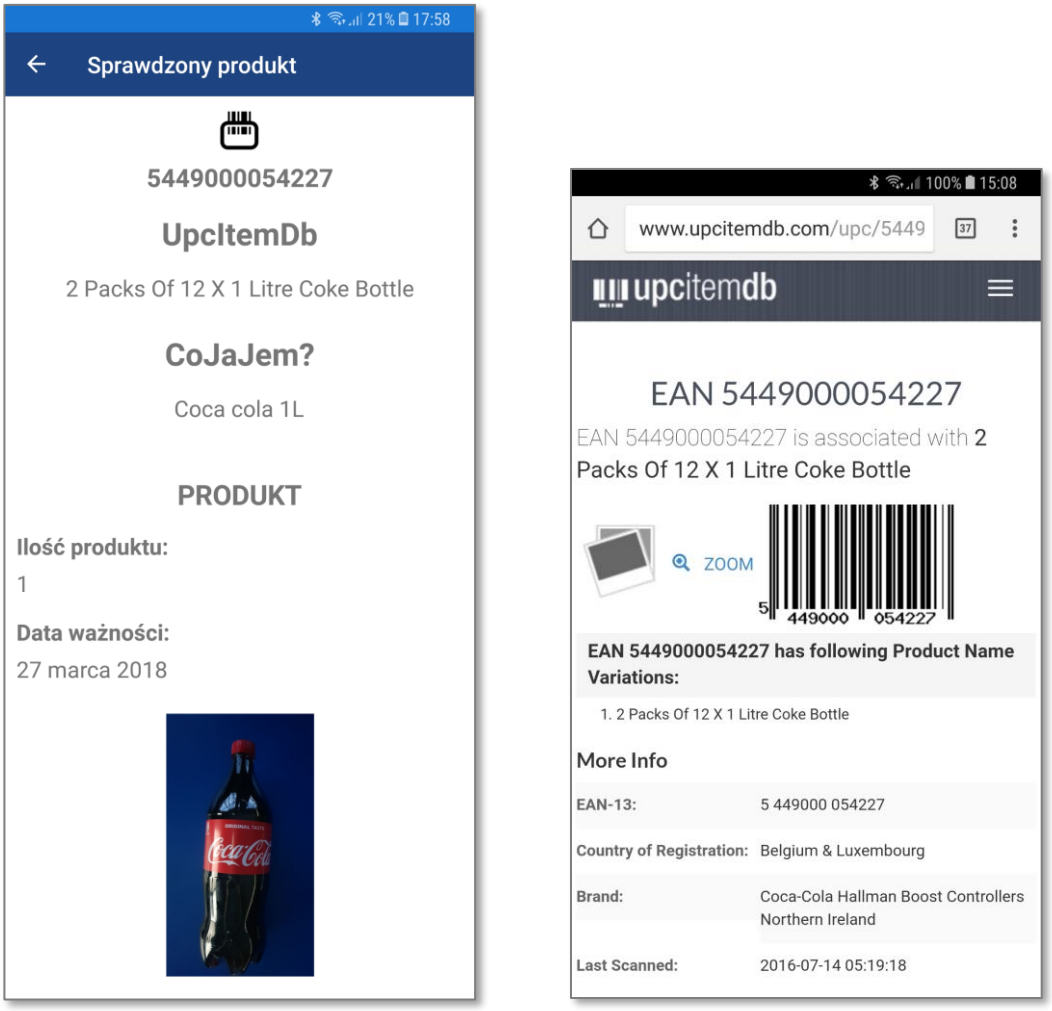
W przypadku akceptacji zdjęcie dołączane jest do opisu wprowadzanego produktu. Po zapisaniu produktu informacje na jego temat wysyłane są do serwisów zawierających bazy produktów. Aktualizowany jest także stan zawartości lodówki. Przebieg dodawania produktu zobrazowano na powyższym rys. V-19.



Rysunek V-20. Usuwanie produktu z lodówki

Źródło: Opracowanie własne

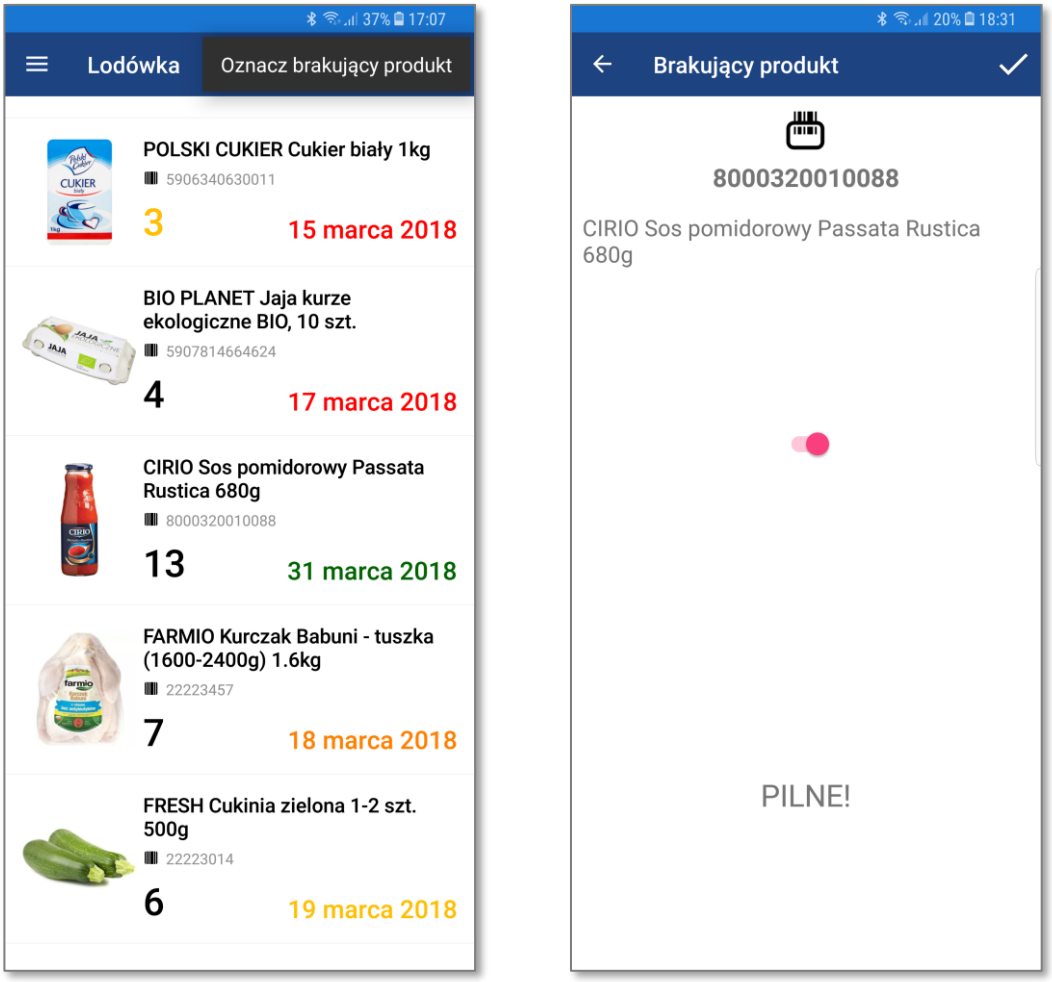
Podobnie jak dodawanie produktów, usuwanie można wykonać poprzez zeskanowanie kodu kreskowego lub wybór z menu kontekstowego na głównym ekranie obszaru *Lodówki*. W momencie usuwania pojawia się ekran, na którym wyświetlone są podstawowe informacje o produkcie. Użytkownik ma możliwość podania ilości, która wyjmowana jest z lodówki. Sposób usuwania produktu zaprezentowano na powyższym rys. V-20.



Rysunek V-21. Sprawdzenie produktu w serwisach baz produktów

Źródło: Opracowanie własne

Interesującą opcją jest możliwość sprawdzenia informacji dotyczących produktu. Dostarczane są nie tylko informacje na temat aktualnego jego stanu w lodówce, ale także jego nazwa pozyskiwana z serwisów zawierających bazy produktów. Na powyższym rys. V-21 zaprezentowano sprawdzenie dodanego wcześniej produktu. Na ekranie aplikacji mobilnej wyświetlone zostały nazwy dla tego produktu z dwóch serwisów: UpcItemDb oraz zrealizowanego na potrzeby prezentacji systemu, serwisu CoJaJem. W przypadku serwisu CoJaJem została wyświetlona nazwa, którą wprowadzono wcześniej podczas dodawania produktu. Z kolei serwis UpcItemDb dla tego produkt podaje nazwę *2 Packs Of 12 X 1 Litre Coke Bottle*. Nazwa ta jest zgodna z nazwą prezentowaną dla tego kodu kreskowego na oficjalnej stronie tego serwisu. Potwierdza to zrzut z ekranu przeglądarki na powyższym rys. V-21.



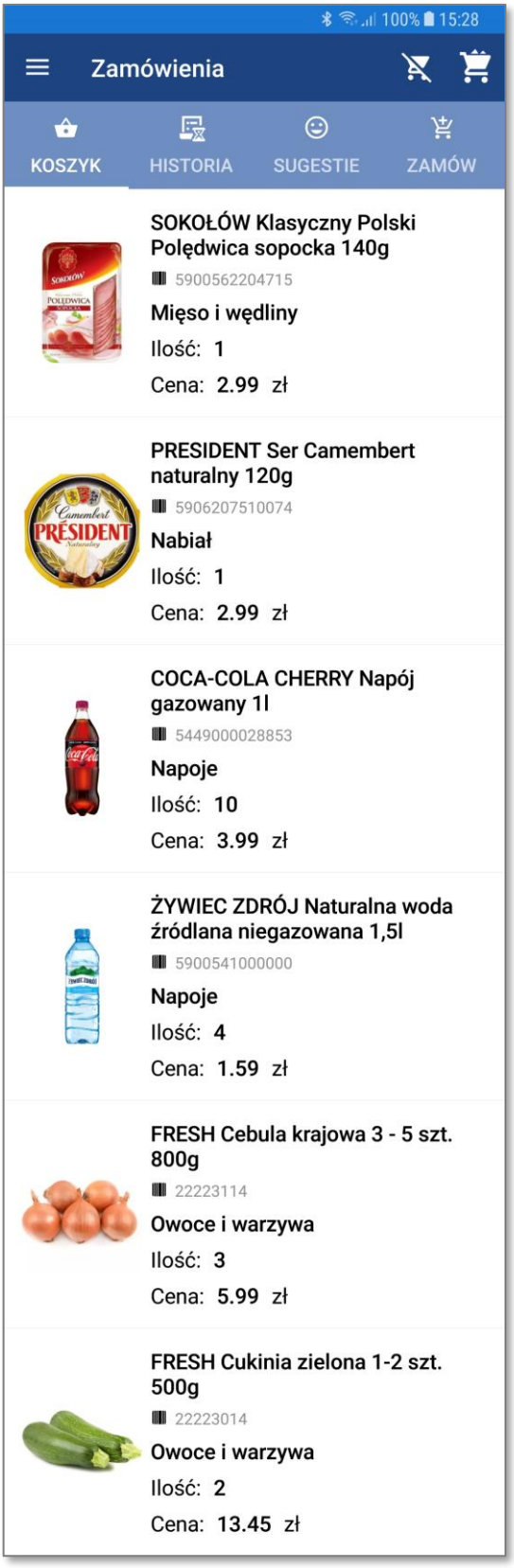
Rysunek V-22. Oznaczenie brakującego produktu

Źródło: Opracowanie własne

Ostatnią czynnością, którą można wykonać w obszarze *Lodówki*, jest oznaczenie brakującego produktu. Podobnie jak w poprzednich przypadkach realizuje się to przy pomocy skanowania kodu kreskowego produktu. W momencie oznaczania można przypisać mu etykietę pilnego zamówienia. Zobrazowano to na powyższym rys. V-22.

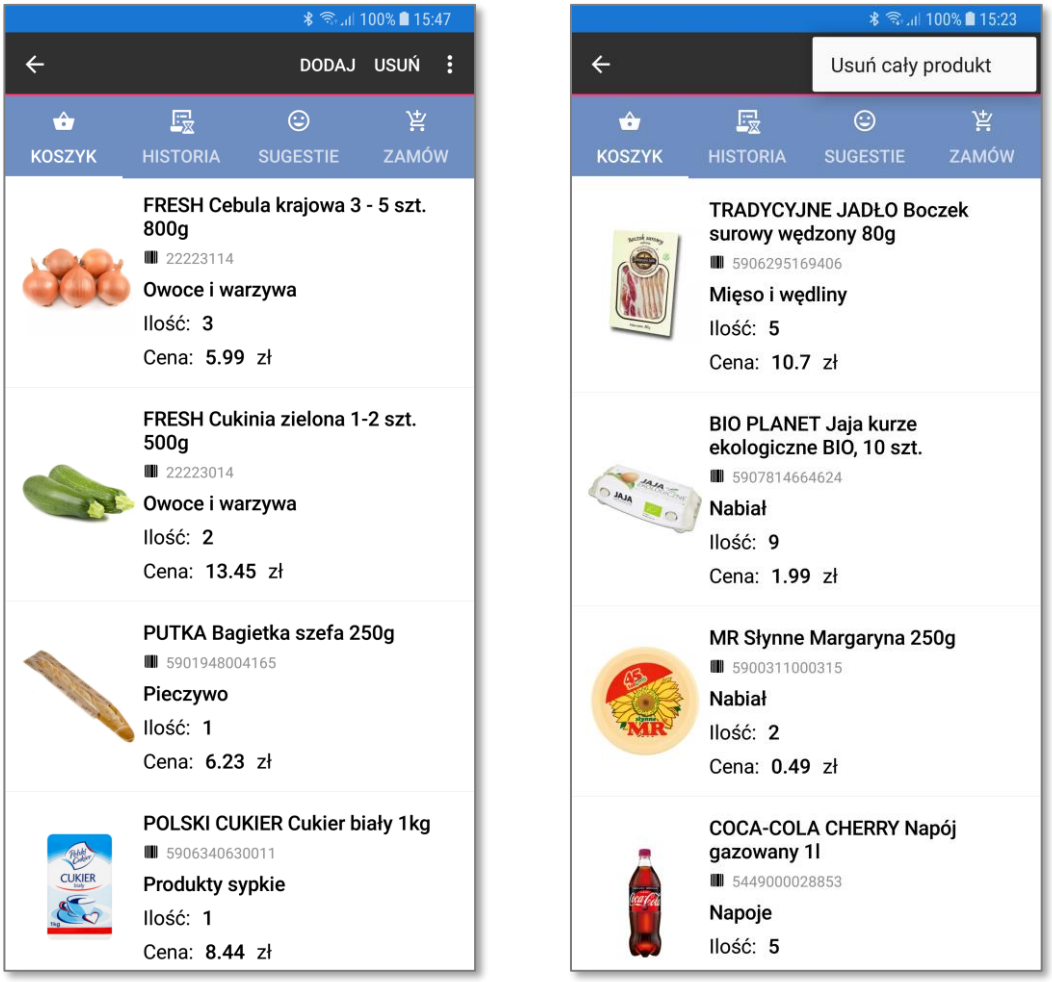
5.2.4.3. Obszar Zamówienia

Obszar ten pozwala na podgląd aktualnego stanu koszyka zamawianych produktów, dorzucanie do niego nowych produktów oraz przeglądanie historii zamówień. Przedstawia także użytkownikowi informacje o sugerowanych produktach do zakupu. Dostęp do tych funkcjonalności uzyskuje się poprzez udostępnione zakładki. Przy pomocy ikon znajdujących się w pasku menu można zrealizować zamówienie albo usunąć całą zawartość koszyka. Zakładka *Koszyka* wyświetla zawartość całego koszyka. Przedstawiono to na rys. V-23.



Rysunek V-23. Zawartość koszyka

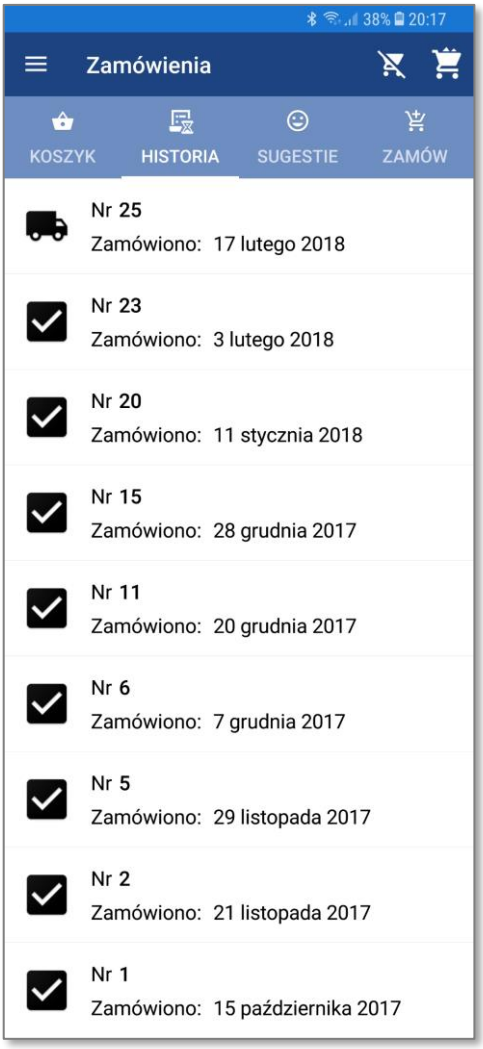
Źródło: Opracowanie własne



Rysunek V-24. Wybór opcji z menu kontekstowego dla produktu

Źródło: Opracowanie własne

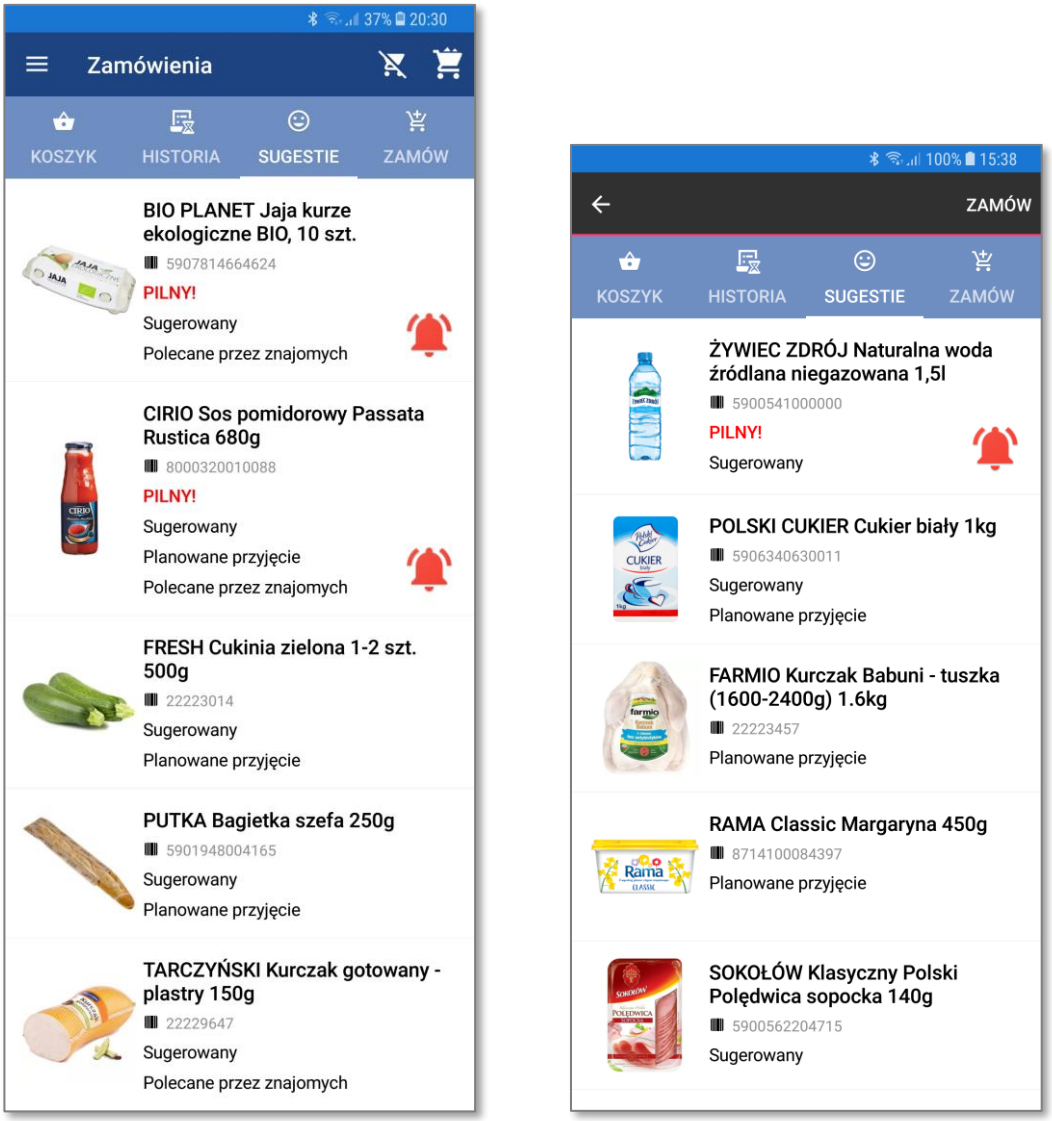
Podstawowymi informacjami wyświetlanymi dla produktów są nazwa, kod kreskowy, kategoria oraz ilość, która jest zamawiana wraz z ceną jednostkową. W zakładce *Koszyka* istnieje dodatkowo możliwość zwiększenia lub zmniejszenia ilości zamawianego produktu. Wykonywane jest to przy pomocy menu kontekstowego wyświetlanego po przytrzymaniu zaznaczenia wybranego produktu. W tym momencie istnieje także możliwość na całościowe usunięcie produktu. Realizację tych funkcjonalności przedstawiono na rys. V-24.



Rysunek V-25. Podgląd historii zamówień

Źródło: Opracowanie własne

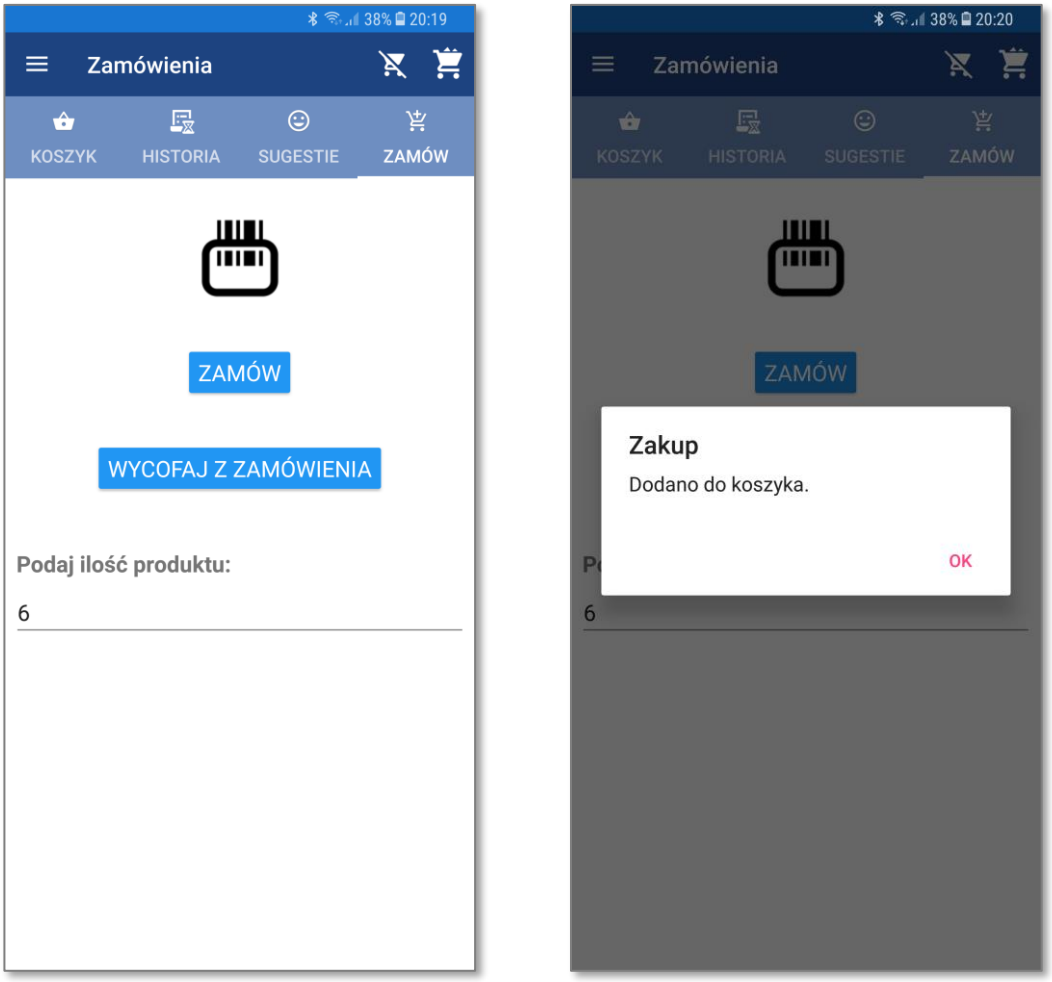
Kolejną zakładką w obszarze *Zamówień* jest zakładka *Historia*. Wyświetlane są tutaj wszystkie dotychczasowe zamówienia zrealizowane przez użytkownika. Dostarczanymi informacjami o zamówieniach są numer zamówienia, data oraz stan jego realizacji. Historię zamówień przedstawiono na rys. V-25.



Rysunek V-26. Lista sugestii i zamówienie produktu z menu kontekstowego

Źródło: Opracowanie własne

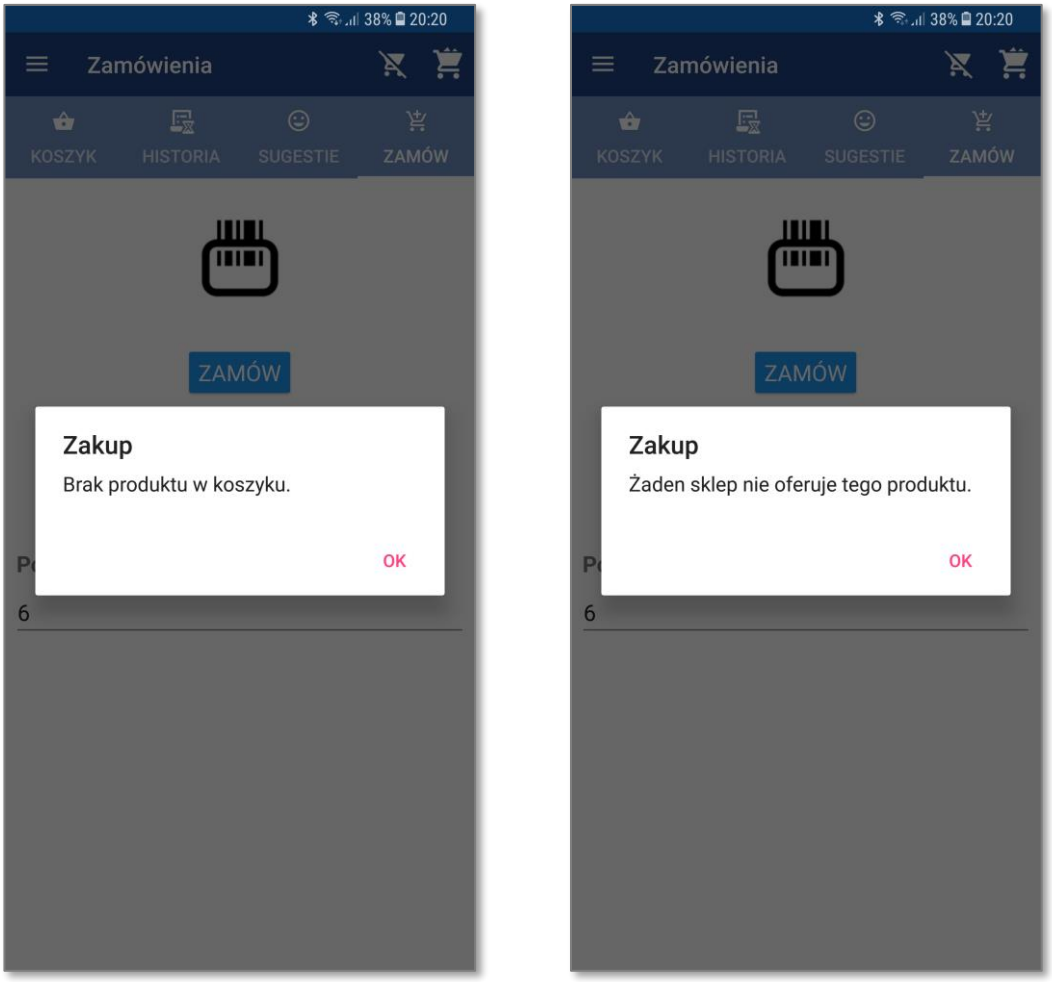
Trzecia zakładka *Sugestie* daje użytkownikowi możliwość na podjęcie decyzji o konieczności zakupu wybranego produktu. Widnieją w niej te produkty, które zostały oznaczone odpowiednim kryterium w obszarze *Lodówki*. Wyświetlane są tutaj także produkty sugerowane przez system. Produkty pilne wyświetlane są z dodatkowym piktogramem czerwonego dzwonka. Podobnie jak w innych przypadkach także tutaj istnieje możliwość zamówienia wybranego produktu bezpośrednio z menu kontekstowego wyzwalanego poprzez przytrzymanie produktu na liście. Zakładkę *Sugestie* przedstawiono na rys. V-26.



Rysunek V-27. Zmiana zawartości koszyka poprzez skanowanie produktów

Źródło: Opracowanie własne

Ostatnią zakładką w tym obszarze jest zakładka *Zamów*. Daje ona użytkownikowi możliwość do zamówienia produktu lub jego wycofania z koszyka poprzez zeskanowanie jego kodu kreskowego. Przed zeskanowaniem produktu należy podać jego ilość dla wykonywanej czynności. Po dodaniu lub wycofaniu z koszyka wyświetla się odpowiedni komunikat. Przedstawiono to na rys. V-27.



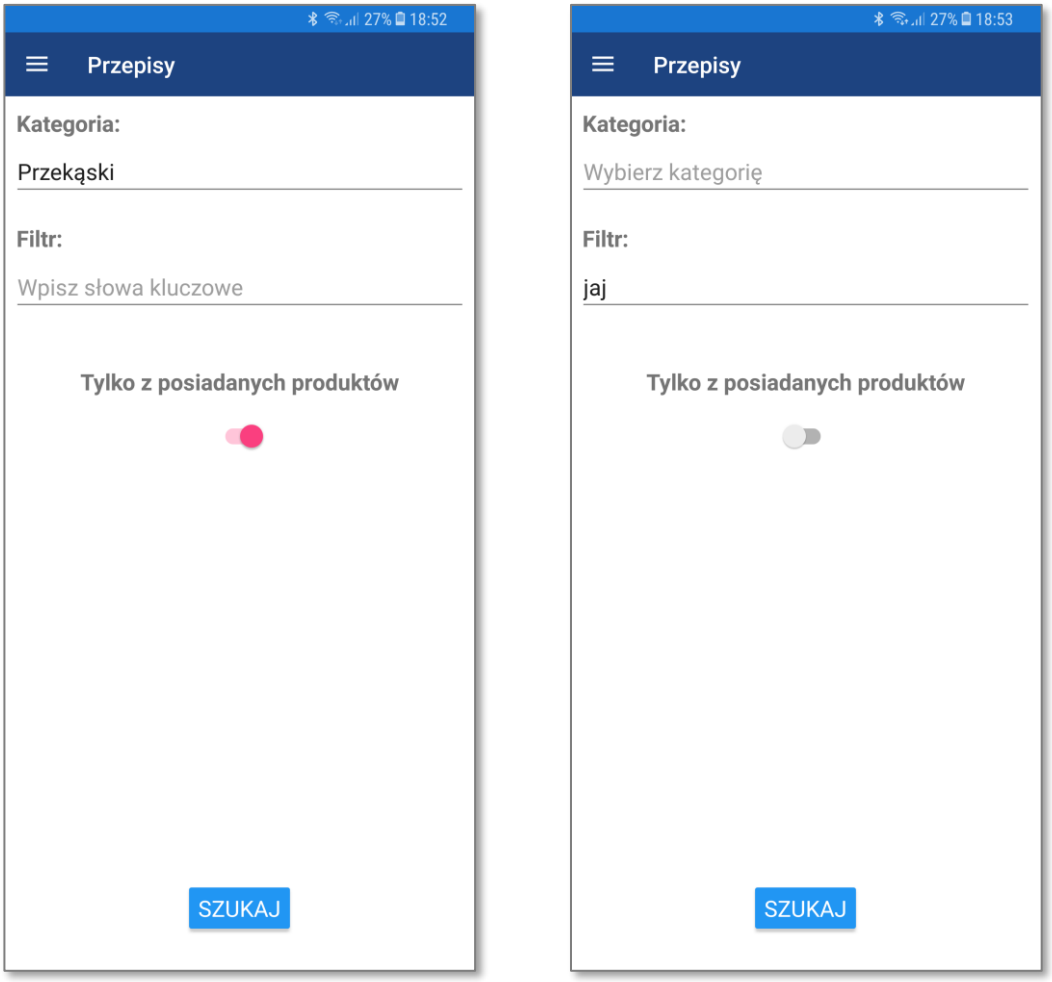
Rysunek V-28. Informacje związane z niepowodzeniem przy zakupie

Źródło: Opracowanie własne

W przypadku braku produktu zarówno w koszyku, jak i w sklepie internetowym, wyświetlany jest odpowiedni komunikat. Przykładowe komunikaty wyświetlane podczas realizacji funkcjonalności przewidzianych w tej zakładce przedstawiono na rys. V-28.

5.2.4.4. Obszar Przepisy

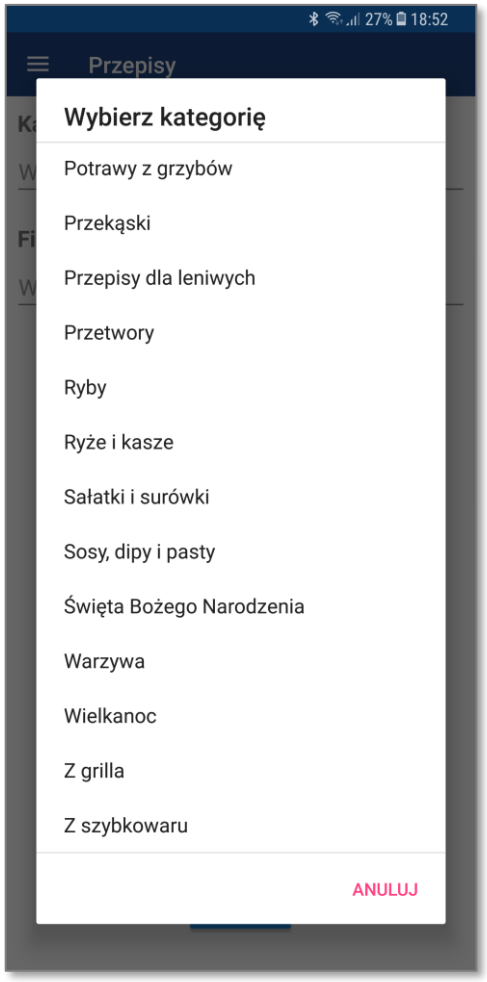
Ostatnim głównym obszarem aplikacji jest pozyskiwanie informacji o przepisach. Oprócz samych przepisów, uzyskać można także informację, czy w lodówce dostępne są wszystkie produkty potrzebne do ich zrealizowania. Można także przy pomocy filtru wyświetlić tylko te przepisy, dla których wszystkie składniki znajdują się w lodówce.



Rysunek V-29. Wypełnienie kryteriów dla wyszukiwania przepisu

Źródło: Opracowanie własne

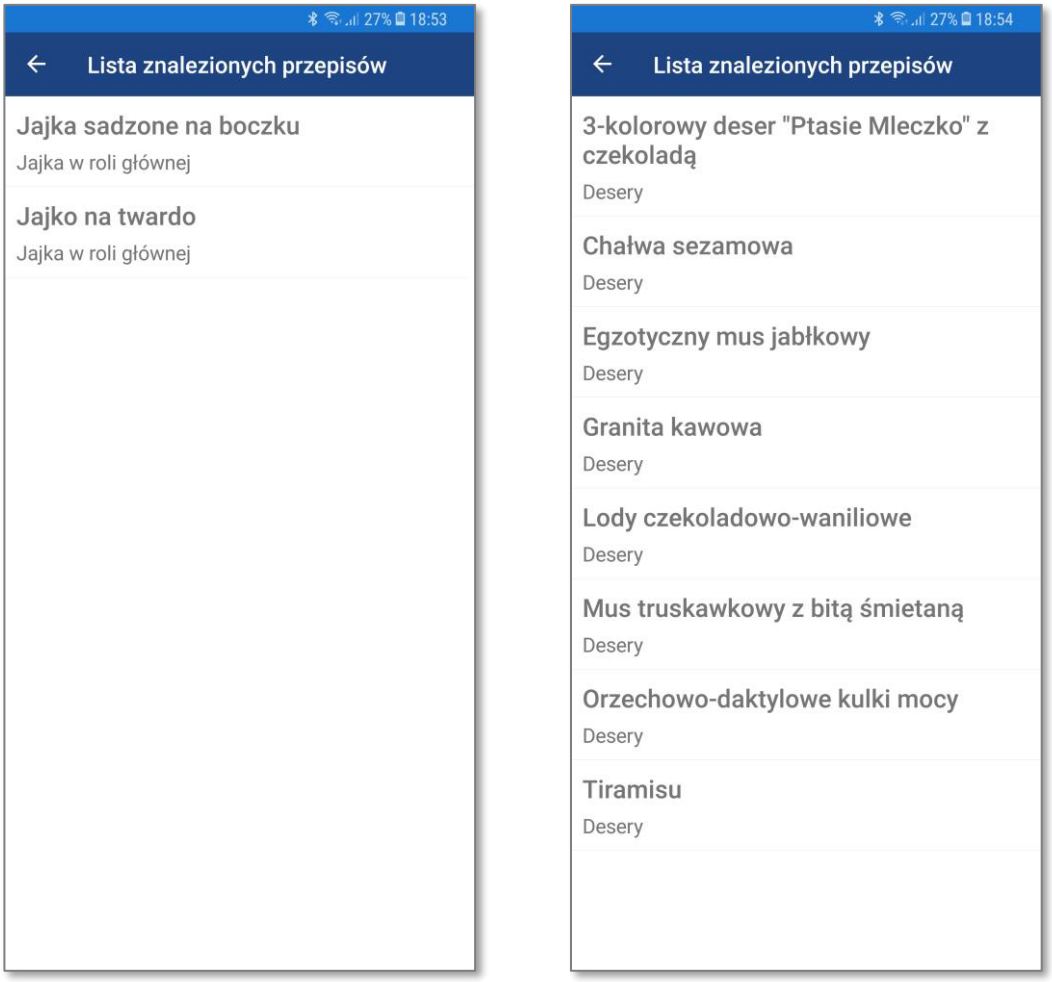
Aplikacja daje możliwość wyszukania przepisów na podstawie trzech kryteriów: kategorii, słów kluczowych oraz wyświetlenia tylko tych przepisów, dla których w lodówce dostępne są wszystkie składniki. Jest to główne zadanie tego obszaru. Możliwości ustawiania kryteriów wyszukiwania przedstawiono na rys. V-29. W celu wyszukania produktów należy użyć przycisku *Szukaj*.



Rysunek V-30. Wybór kategorii przepisów

Źródło: Opracowanie własne

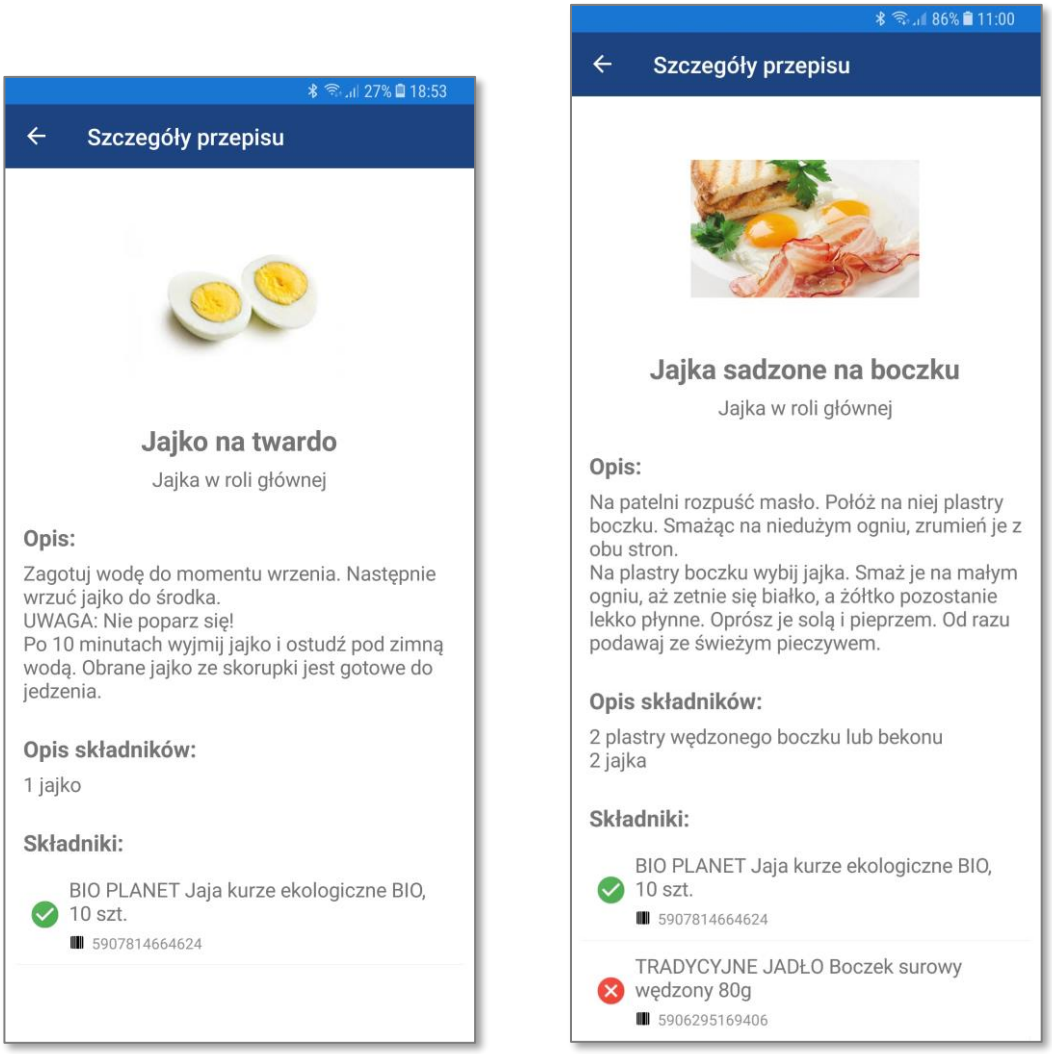
Lista kategorii do wyszukiwania przepisów pozyskiwana jest z serwisów kucharskich. W przypadku, gdy dostępna jest większa ilość serwisów, wyświetlane są wszystkie dostępne z nich kategorie. Możliwość wyboru z listy kategorii przedstawiono na rys. V-30.



Rysunek V-31. Wynik poszukiwań przepisów dla wprowadzonych kryteriów

Źródło: Opracowanie własne

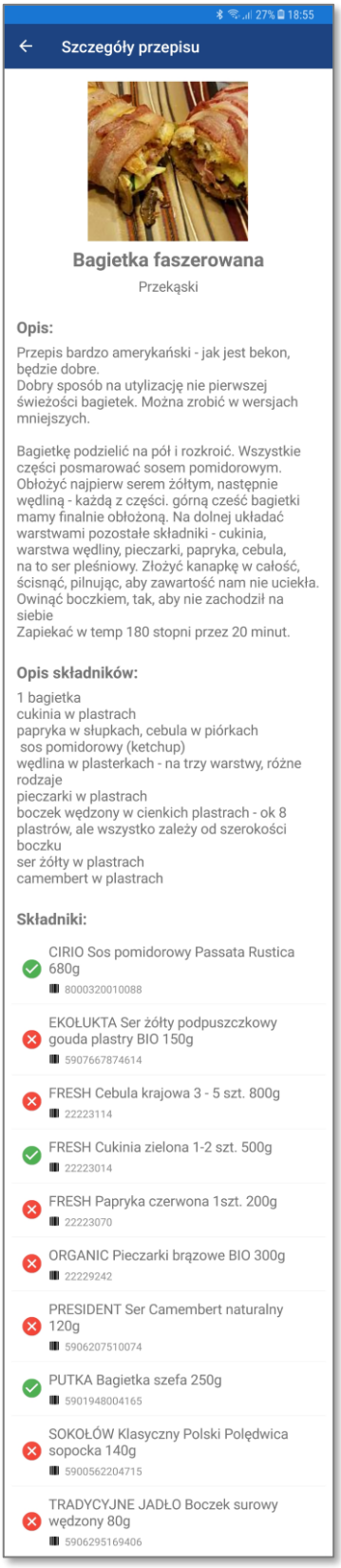
Po ustawieniu kryteriów i zainicjowaniu wyszukiwania, wyświetlana jest lista przepisów. Podstawowymi informacjami wyświetlanymi w wyniku znalezionych przepisów są nazwa oraz kategoria. Listę wyszukanych przepisów przedstawiono na rys. V-31.



Rysunek V-32. Szczegóły przepisów

Źródło: Opracowanie własne

Po wybraniu któregokolwiek z przepisów wyświetlane są jego szczegółowe informacje. Oprócz podstawowych informacji o przepisie, wyświetlany jest także jego opis, opis składników oraz lista wszystkich składników wraz z informacją o ich dostępności w lodówce. Dostępność ich oznaczona jest na liście odpowiednimi kolorowymi piktogramami. Przykładowe przepisy, dla których dostępne są w lodówce wszystkie składniki lub tylko niektóre z nich przedstawiono na rys. V-32 oraz V-33.

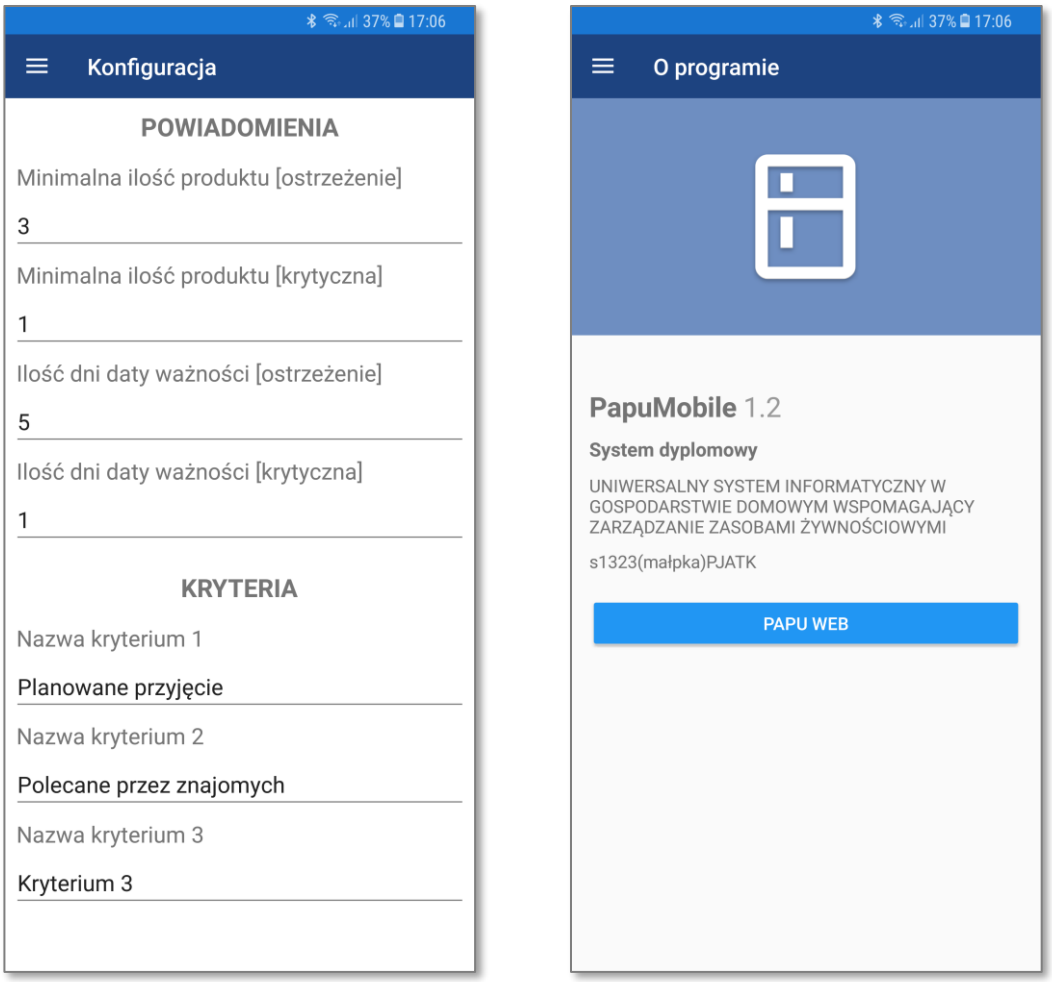


Rysunek V-33. Szczegóły przepisów

Źródło: Opracowanie własne

5.2.4.5. Pozostałe obszary

Obszarami dodatkowymi aplikacji są konfiguracja systemu oraz obszar poświęcony przedstawieniu informacji dotyczących aplikacji. W obszarze konfiguracji możliwe jest ustawienie podstawowych parametrów, które wykorzystywane są przez aplikację podczas jej działania. Można ustawić minimalną ilość produktu, przy której system będzie informował użytkownika o zaistniałym stanie, oraz ilość dni pozostałych do końca terminu ważności produktu. Oba parametry można ustawić dla dwóch sytuacji jakimi są ostrzeżenie oraz wystąpienie sytuacji krytycznej. W obszarze konfiguracji istnieje także możliwość ustawienia nazw dla kryteriów, które użytkownik przypisuje poszczególnym produktom. W obszarze informacji o programie oprócz podstawowych informacji o aplikacji, istnieje bezpośrednia możliwość otwarcia witryny systemu. Opisane obszary przedstawiono na rys. V-34.

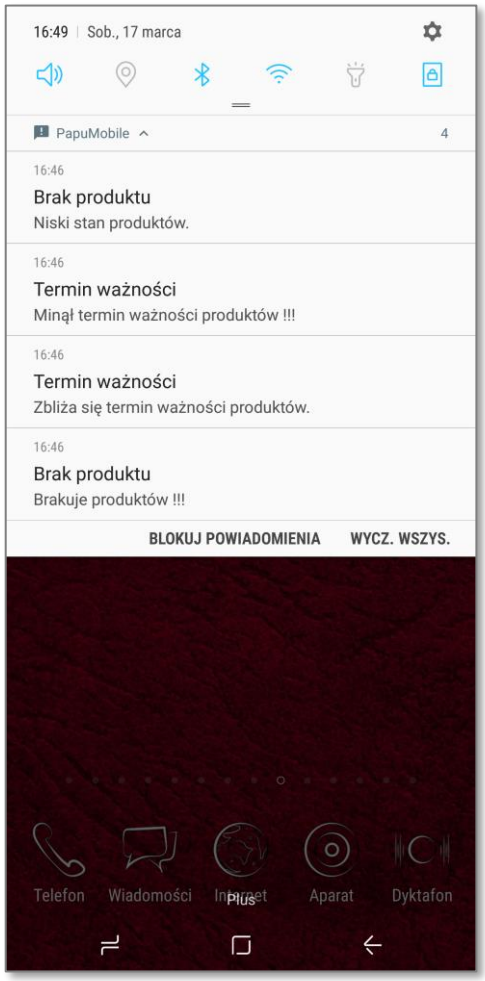


Rysunek V-34. Konfiguracja systemu oraz informacja o programie

Źródło: Opracowanie własne

5.2.4.6. Powiadomienia

Aplikacja mobilna posiada możliwość informowania użytkownika o występowaniu różnych zdarzeń związanych z przechowywanymi w lodówce produktami. Typowymi zdarzeniami związanymi z ilością produktów są niski stan lub brak produktów. Zbliżający się termin ważności produktów lub przekroczenie tego terminu to kolejne powiadomienia, które realizowane są z aplikacji. Wyświetlenie kilku przykładowych powiadomień przedstawiono na poniższym rys. V-35.



Rysunek V-35. Powiadomienia z systemu

Źródło: Opracowanie własne

5.3. Serwisy zewnętrzne

Nieodłącznym elementem działania prototypu są serwisy zewnętrzne. Bez nich nie byłoby możliwe rozbudowanie go o szereg użytecznych funkcjonalności. W celu prezentacji

jego pracy zrealizowano integrację z jednym serwisem komercyjnym oraz z trzema serwisami wykonanymi w postaci prototypów. Każdy z tych prototypów wykonano dla innego typu działalności serwisu. Integracje zrealizowano przy pomocy wtyczek.

5.3.1. Serwis UpcItemDb

UpcItemDb jest to jeden z większych komercyjnych serwisów udostępniających bazę produktów. Na daną chwilę w swoim zbiorze posiada ponad 142 mln unikalnych kodów UPC/EAN. Na ich podstawie identyfikowane są produkty. W ramach realizacji prototypu stworzono wtyczkę obsługującą udostępnione API. Przy jej pomocy można pobrać z tego serwisu informacje na temat produktów. Sposób implementacji wtyczki szczegółowo opisano w pkt. 5.1.4 na str. 67.

5.3.2. Prototypowa implementacja serwisów

Ze względu na problemy z dostępem do innych komercyjnych rozwiązań, zdecydowano się na ich prototypową implementację.

Implementację serwisów wykonano przy pomocy języka skryptowego PHP. Na silnik bazodanowy wybrano MySQL. Każdy z serwisów posiada własny moduł REST API. Routing dla API oparto na odpowiednich wpisach w plikach *.htaccess*. Zdjęcia produktów przechowywane są w bazie danych binarnie jako typ danych *BLOB*. W celu zobrazowania czynności wykonywanych przez interfejs API stworzono także wizualną część serwisów w postaci stron internetowych.

W ramach prototypowej implementacji serwisów stworzono sklep internetowy, serwis z bazą produktów oraz serwis kucharski z przepisami kulinarnymi. Serwisy mają budowę modułową dzięki czemu odseparowano od siebie kluczowe części kodu.

Każdy z serwisów posiada identyczny schemat budowy:

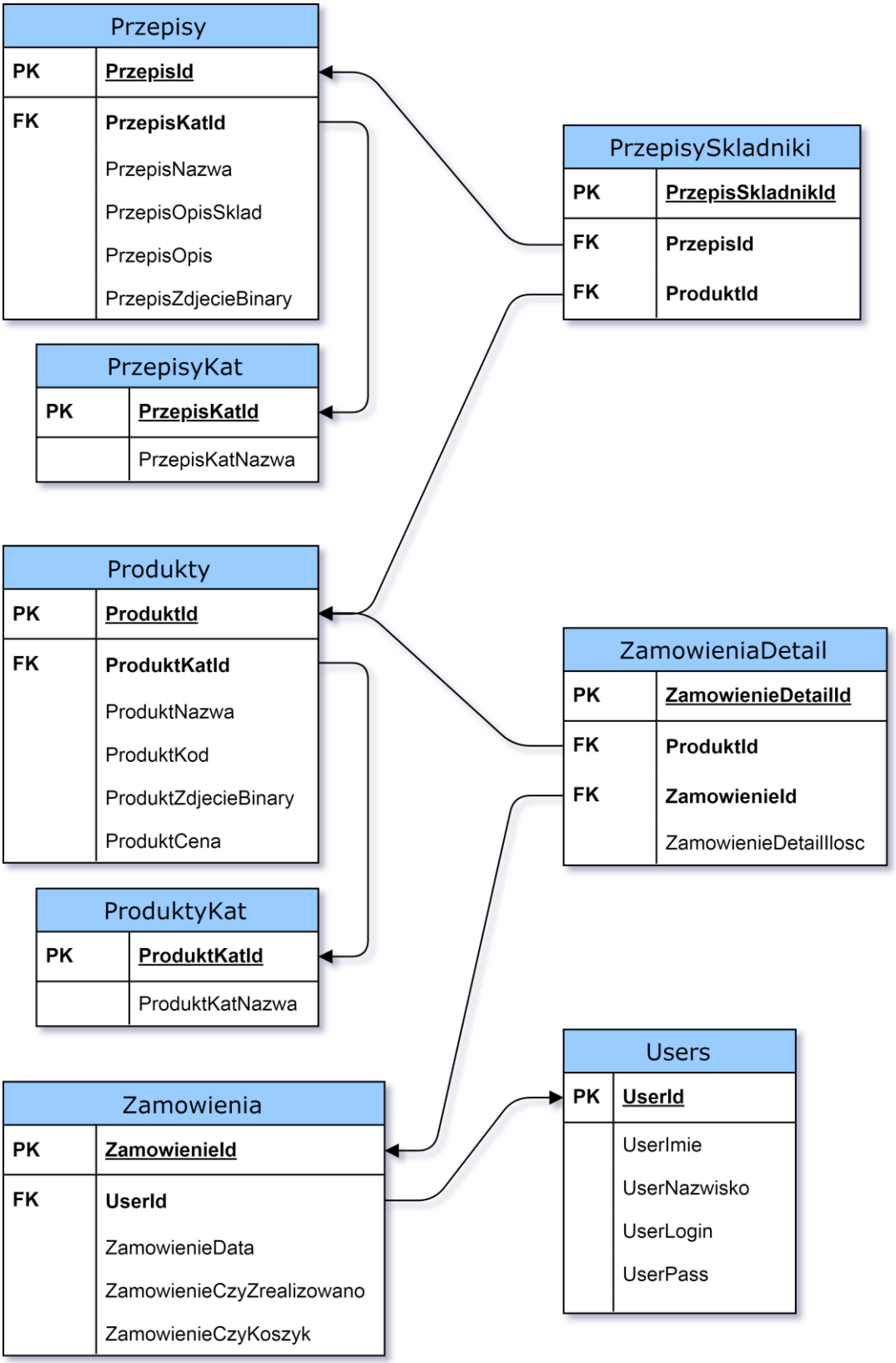
- strona API – przechwycenie i wstępne przetworzenie żądania,
- klasa *RestController* – przetworzenie żądania w metodzie odpowiadającej rodzajowi żądania i zwrot rezultatu,
- klasa *ResourceController* – stworzenie zapytania bazodanowego i przetworzenie zwróconej odpowiedzi.

Wszystkie serwisy w klasie *ResourceController* używają klasy *DbControl* do wykonywania poleceń bazodanowych. API serwisów nie jest identyczne z tym zastosowanym w aplikacji webowej. Główna różnica polega na innym konstruowaniu adresów URL żądania, choć w niektórych przypadkach sposób konstrukcji pokrywa się.

Ze względu na to, iż realizacja środowiska serwisów zewnętrznych nie jest integralną częścią prototypu, bardziej szczegółowe informacje na temat jego budowy dostępne są w dokumentacji elektronicznej.

5.3.3. Baza danych

W bazie danych umieszczone są przykładowe dane pozwalające na pracę systemu. Każdy z serwisów posiada swoje własne tabele, w których realizuje swoją pracę. Jedyną wspólną tabelą dla sklepu internetowego oraz przepisów kucharskich jest tabela *Produkty*. Połączenie to zostało wykonane w celu optymalizacji pracy poświęconej na implementację środowiska serwisów, które to serwisy nie są częścią głównego prototypu. Schemat bazy danych dla wykonanego środowiska serwisów zewnętrznych przedstawiono na rys. V-36.



Rysunek V-36. Schemat bazy danych zewnętrznych serwisów

Źródło: Opracowanie własne

Rozdział VI.

PODSUMOWANIE

W niniejszym rozdziale nastąpiło podsumowanie realizacji proponowanego rozwiązania, uniwersalnego systemu informatycznego w gospodarstwie domowym, wspomagającego zarządzanie zasobami żywnościowymi. Omówiono dobre i złe strony jego funkcjonowania, jak również ewentualne kierunki rozwoju. Wspomniano także o trudnościach, jakie wystąpiły podczas realizacji. Na zakończenie rozdziału przedstawiono wnioski końcowe.

6.1. Zalety i wady przyjętych rozwiązań

Szukając zalet prototypu należałoby wymienić możliwość użytkowania niezależnych serwisów bezpośrednio z jego interfejsu. Eliminuje to konieczność użytkowania do tych samych celów serwisów macierzystych. Jest to duże ułatwienie. Ich funkcjonalność, do której użytkownik potrzebuje mieć dostęp, dostępna jest z prototypu. Dodatkowym atutem rozwiązania jest umiejscowienie bazy danych w usłudze internetowej. Pozwoliło to na podgląd aktualnego stanu lodówki w każdym momencie, niezależnie od tego, gdzie się znajdujemy.

Automatyczne zestawianie list zakupów jest tą funkcjonalnością, dzięki której system jest bardziej samoobsługowy. Dzięki jego możliwości podpowiadania przepisów kulinarnych jesteśmy w stanie zaplanować swoje zakupy w sposób bardziej ekonomiczny.

Jedną z większych wad prototypu w jego obecnym kształcie jest brak możliwości uwierzytelniania się użytkowników. Jest to jednak funkcjonalność, która nie miała związku z wypracowywaną koncepcją — szerszego rozwiązania problemu zarządzania żywnością. Z tego powodu nie była brana pod uwagę podczas realizacji implementacji.

Pewnym dyskomfortem w obsłudze prototypu jest także konieczność manualnego ewidencjonowania produktów. Wydaje się, że uzyskanie niezawodnego, automatycznego rozwiązania będzie możliwe dopiero z chwilą pojawienia się nowych technologii. Temat ten rozwinięto w następnym podrozdziale, który poświęcony jest rozwojowi.

Kolejnym słabszym punktem prototypu jest dostępność wyłącznie dla systemu operacyjnego Android. Takie było jednak założenie jego realizacji, dlatego od samego początku była świadomość o tej niedoskonałości. Niedoskonałość tę można jednak w łatwy sposób naprawić ze względu na świadomy wybór środowiska programistycznego Xamarin.Forms. Daje ono możliwość do wygenerowania wersji prototypu na inne systemy przy niewielkich nakładach pracy. Prace te związane byłyby głównie z dostosowaniem aktualnej implementacji pod specyficzne rozwiązania danego systemu operacyjnego.

6.2. Kierunki rozwoju prototypu

Każdą pojawiającą się na rynku propozycję rozwiązania jakiegoś zagadnienia można rozwijać o kolejne możliwości. Żadne z nich nigdy nie będzie posiadało skończonej, nierozwijalnej już funkcjonalności. Wpisuje się to w naturalną ewolucję każdego rozwiązania. Tak jak we wszystkich przypadkach także i to proponowane w niniejszej pracy można dowolnie rozszerzać. Zarówno pod względem ergonomii obsługi, ilości dostępnej funkcjonalności, jak i budowania bardziej zaawansowanych algorytmów pracy.

6.2.1. Automatyzacja rozwiązań

Tego rodzaju rozwiązanie jak proponowane w niniejszej pracy będzie dawało wymierne korzyści tylko wtedy, kiedy nie będziemy obarczeni zbytnio jego obsługą. Idealną sytuacją byłoby, gdyby funkcjonowało bez jakiejkolwiek ingerencji, a my bylibyśmy beneficjentami jego pracy.

Tutaj uwidacznia się istota funkcjonowania w pełni zautomatyzowanych systemów informatycznych. Tylko poprzez jednoznaczną, zautomatyzowaną identyfikację produktów, jest możliwe automatyczne, poprawne przetwarzanie danych. System oparty w jakimś stopniu na czynniku ludzkim, w trakcie procesu obrabiania danych, nie może pełnić roli w pełni zautomatyzowanego systemu informatycznego.

Rozważmy chociażby wspomnianą propozycję rozwiązania z pkt. 2.2.2 na str. 8. Identyfikacja wizualna produktów może sprawdzać się w prostych przypadkach. Biorąc pod

uwagę wszelkie ewentualności należy pomyśleć także o nietypowych sytuacjach. Takie opakowanie może być chociażby pomazane. Produkt może być również przepakowany. Pojawia się wtedy problem z jego identyfikacją. Po nauczeniu systemu zdjęcie tego produktu trafi do globalnej bazy danych. Spowoduje to jej zaśmiecanie. To w konsekwencji przełoży się na spadek jej wydajności lub generowanie błędnych rezultatów. Na to przykładem może być również błędne identyfikowanie przez konsumenta wprowadzanych przez niego zdjęć produktów. W społeczeństwie istnieje szeroki przekrój osobowości. Wystarczy przytoczyć przypadki wstawiania „uśmieszków" na kartach do głosowania podczas tak poważnych okoliczności, jak wybory władz państwowych.

Tak skonstruowany system byłby zależny od poczynań człowieka. Tym samym w dość prosty sposób narażałby się na generowanie błędów. Pewnym rozwiązaniem mogłoby być weryfikowanie wprowadzanych danych. Opierałoby się jednak na manualnej kontroli realizowanej przez człowieka. W tym przypadku główną rolę mogłyby odegrać nowe technologie takie jak sztuczna inteligencja. Można jednak przypuszczać, że w dłuższym okresie czasu nadal będzie w niej więcej sztuczności niż inteligencji. Z tego powodu rdzeń prototypu oparty na czynniku ludzkim nie jest proponowanym kierunkiem rozwoju.

6.2.2. Nowe technologie

Na rozwój stworzonego prototypu mogłyby mieć wpływ także wspomniane przed chwilą nowe rozwiązania technologiczne. Są to rozwiązania, które jeszcze nie pojawiły się na rynku lub koszt ich wdrożenia na obecną chwilę byłby zbyt wysoki. Moglibyśmy na przykład wyobrazić sobie, że każdy produkt posiada zaszytą w sobie informację. Zawarte byłyby w niej nie tylko dane identyfikacyjne, ale także data przydatności do spożycia, skład zawartości lub inne właściwości produktu. Rozwijając tę koncepcję moglibyśmy dać możliwość dopisywania kolejnych danych w trakcie całego łańcucha dystrybucji. Dawałoby to nieograniczone możliwości wykorzystania takiego rozwiązania. Mogłaby na przykład istnieć możliwość dopisania jego ostatecznej ceny, po której go zakupiono. Dołączanie informacji, w którym sklepie nastąpił zakup lub od którego dostawcy żywności pochodzi to tylko kolejne z nich. Dane te mogłyby być użyte w celach statystycznych, księgowych lub innych. Możliwości ich wykorzystania byłyby ograniczone jedynie wyobraźnią projektanta systemu. Usprawniłoby to jednak znacznie nad nimi kontrolę.

Takie możliwości wydaje się, że mogłaby przynieść technologia RFID[8]. Patrząc jednak na jej tempo wdrażania w przemyśle, nie jest to rozwiązanie proponowane na daną chwilę. Spowodowane jest to głównie olbrzymimi kosztami jej zastosowania. Rozwiązania tego typu, co RFID, są jednak kluczowym elementem. Pozwoliłyby rozbudowanemu prototypowi zaistnieć w świecie na szeroką skalę. Przy ich pomocy byłaby odnotowywana każda zmiana zawartości lodówki. Byłoby to realizowane poprzez automatyczne sczytywanie nazw produktów wraz z przypisanymi do nich właściwościami. Wzbogacenie prototypu o nową funkcjonalność wykorzystującą technologie bezprzewodowe dawałoby użytkownikowi końcowemu ogromne korzyści. Zapewne byłby to przełom w użytkowaniu tego rodzaju systemów jak proponowany w niniejszej pracy.

Istnieje także kolejny aspekt mogący mieć wpływ na rozwój prototypu. Jest to standaryzacja rozwiązań opartych na identyfikacji produktów. Nowe sposoby identyfikacji musiałyby zostać uzgodnione globalnie przez wszystkich producentów. Wymuszałoby to zmianę sposobu przetwarzania danych o produktach w procesie ich dystrybucji do konsumenta. Dotyczyłoby to m.in. systemów informatycznych, jak również czytników kodów. Do przeprowadzenia zmian byłby zobligowany każdy sklep oraz przedsiębiorstwo. Możemy wyobrazić sobie, jaka byłaby to skala przedsięwzięcia. Nie jest to proste do wykonania i musiałoby być wprowadzone etapami. Wdrożenie tego typu rozwiązań mogłoby znacznie rozciągnąć się w czasie. Poruszaną problematykę potwierdzałaby historia kodów 2D. W porównaniu do kodów kreskowych posiadają większą ilość miejsca do zapisu danych. Istnieją na rynku od ponad półtorej dekady. Nie spowodowało to jednak, że były w stanie zastąpić kody kreskowe, które są stosowane po dzień dzisiejszy.

6.2.3. Rozwój przyszłościowy

Wybiegając bardzo daleko w przyszłość, moglibyśmy w rozważaniach pójść o krok dalej. Można szacować, że kolejny lub dopiero XXIII w. przyniesie zupełnie nowe spojrzenie na poruszaną tematykę. Możliwe, że w dalekiej przyszłości nie będzie potrzeby własnoręcznego zarządzania lodówką. Sama się będzie uzupełniać na podstawie wybieranych dań z przygotowanego menu. Zamawiane przez nią produkty będą dostarczane ze sklepów drogą powietrzną. Następnie zostaną automatycznie odebrane i umieszczone bezpośrednio w jej wnętrzu także w sposób automatyczny. Wszystko to bez udziału kogokolwiek. W połączeniu

[8] RFID (ang. *Radio Frequency Identification*) jest to bezprzewodowa technologia automatycznej identyfikacji obiektów przy użyciu fal radiowych.

z „inteligentnym" piekarnikiem zostaną przez niego ugotowane i podane do stołu przez „inteligentnego" robota. „Tradycyjna" kuchnia będzie pełnić w ten sposób rolę bezobsługowego, dobrze prosperującego centrum logistyczno-restauracyjnego.

Mogłoby jednak okazać się, że globalna sytuacja żywności będzie wyglądała zupełnie inaczej. Nieodpowiednia gospodarka zasobami żywnościowymi mogłaby spowodować brak dostępności naturalnych surowców. Pozostaną wtedy substytuty w postaci kostek żywieniowych. Przechowywane w ujemnych temperaturach i podgrzewane tuż przed posiłkiem będą przyjmować postać naturalnych produktów. W innym całkiem realnym scenariuszu może okazać się, że w ogóle nie będziemy potrzebowali żywności. Energię, którą z niej czerpiemy, będziemy pobierali z innych źródeł. Nie będą już wtedy potrzebne lodówki ani stworzone dla nich rozwiązania. Przedstawione hipotetyczne scenariusze uzmysławiają, że wizja przyszłości rozwiązań tego rodzaju w dużym stopniu zależy także od poczynań człowieka.

Biorąc pod uwagę powyżej przedstawiane argumenty, wydaje się, że szerszy rozwój prototypu byłby w jakimś stopniu uwarunkowany czynnikami zewnętrznymi. Niektóre z nich są nawet na skalę globalną.

6.2.4. Rozwój teraźniejszy

Oprócz przyszłościowych wizji, prototyp można byłoby rozszerzyć także o funkcjonalności niezależne od innych czynników. Modyfikacjom podlegałyby zarówno algorytmy pracy, jak i estetyka wykonania.

Mogłyby to być m.in.:

- udostępnienie aplikacji na pozostałe mobilne systemy operacyjne,
- wbudowanie uwierzytelniania użytkowników,
- rozbudowa algorytmów związanych z udogodnieniami systemu,
- poszerzenie obszarów integracji z systemami zewnętrznymi,
- realizacja wymiany danych pomiędzy użytkownikami systemu,
- implementacja obsługi głosowej,
- możliwość rozpoznawania tekstu, jak i treści wizualnej,
- bardziej przyjazny wizualnie interfejs użytkownika.

Wymienione powyżej funkcjonalności usprawniałyby obsługę prototypu i mogłyby poszerzyć grono jego odbiorców. Ważnym udogodnieniem w obsłudze systemu byłoby

zwiększenie ilości niezależnych serwisów, z którymi mógłby współpracować. Integracja z takimi serwisami, jak np. system księgowy, dawałaby możliwość ewidencji kosztów, statystyk wydatków itp. Jak w każdym przypadku wymiany informacji obie strony musiałyby obsługiwać taką samą funkcjonalność.

Bez wątpienia można jednak stwierdzić, że rozwój prototypu ograniczony jest w głównym stopniu tylko wizją projektanta oraz dostępnością użytecznych technologii potrafiących wzbogacić system o nowe obszary obsługi.

6.3. Napotkane trudności

Podczas realizacji prototypu napotkano na poniżej opisane problemy implementacyjne. Wystąpiły zarówno podczas tworzenia aplikacji mobilnej, jak i aplikacji webowej.

Jednym z nich był brak kompatybilności komponentów mobilnych z wybraną platformą .NET Standard. Dotyczyło to biblioteki ZXing.Net.Mobile odpowiedzialnej za obsługę skanowania kodów kreskowych. Problem ten rozwiązano poprzez zastosowanie technik kompatybilności. Polegały na wprowadzeniu odpowiedniego zapisu w pliku konfiguracyjnym projektu. Zapis ten pozwolił na bezproblemowe użytkowanie biblioteki pomimo wyświetlanych w projekcie ostrzeżeń o jej niekompatybilnym środowisku kompilacji. Projekt kompiluje się, a aplikację użytkuje się bez żadnych problemów. Wymagany zapis przedstawiono w przykładzie VI-1.

Przykład VI-1. Ustawienie trybu kompatybilności

```
<PropertyGroup>
  <AssetTargetFallback>
    $(AssetTargetFallback);net46;
  </AssetTargetFallback>
</PropertyGroup>
```

Źródło: Opracowanie własne

Inne problemy zaistniały podczas realizacji części webowej. Były związane z realizacją mechanizmu obsługi wtyczek. Podobnie jak w poprzednim przypadku dotyczyło to środowiska pracy. Tym razem kłopot sprawiała zbyt świeża wersja .NET Core. Okazało się, że komponenty zarówno MEF, jak i Autofac, nie miały wystarczająco rozbudowanej funkcjonalności dla tej platformy. Możliwość użycia ich wbudowanych rozwiązań, dostępnych tylko dla pełnego środowiska .NET, zapewne przyspieszyłoby prace nad prototypem. Ostatecznie problem rozwiązano ładując biblioteki ręcznie. Technologię MEF użyto jedynie do

samego eksportu i importu typów. Natomiast zamiast rejestracji w Autofac użyto kontenera zależności wbudowanego w środowisko programistyczne .NET Core.

Podczas implementacji nie napotkano na więcej istotnych trudności, oprócz tych opisanych powyżej. Można nawet stwierdzić, że decyzja o realizacji prototypu w najnowszych środowiskach, dała dość dużą ilość korzyści.

6.4. Wnioski końcowe

Wypracowanie koncepcji rozwiązującej tematykę niniejszej pracy wraz z realizacją prototypu pomogło uwidocznić skalę jej rozmiaru. Dało to możliwość do dokładniejszego zastanowienia się nad nią. Proponowane rozwiązania w podrozdziale 6.2 na str. 113 są tylko kolejnymi etapami w całym procesie ewolucji systemu. Prace w tym kierunku przyniosą jeszcze jedną wcześniej wspominaną korzyść. Co roku marnuje się miliony ton żywności. Rozwój systemów takich jak ten pomógłby temu zaradzić.

ZAKOŃCZENIE

Niewątpliwie proponowane rozwiązanie ma swoją przyszłość. Trudno w chwili obecnej stwierdzić, czy jest w stanie zrewolucjonizować styl współczesnego życia. Na pewno jest w stanie zmienić sposób, w jaki podchodzimy na co dzień do omawianego w niniejszej pracy zagadnienia. W połączeniu z wprowadzanymi nowymi technologiami, wspomnianymi w niniejszej pracy, perspektywa realizacji bardziej zautomatyzowanych rozwiązań tego typu jest obiecująca. Przełomem mogą okazać się technologie, nad którymi nie rozpoczęły się jeszcze nawet prace badawcze.

Na przestrzeni wielu lat udowodniliśmy sobie jako społeczeństwo, że jesteśmy w stanie docenić korzyści, jakie niosą ze sobą nowe rozwiązania oraz istniejące technologie. Jest jedynie kwestią czasu oraz poświęcenia odpowiedniej ilości środków na ten cel, aby rozwiązania stały się bardziej kompleksowe, zautomatyzowane. Miałyby wtedy niewątpliwą szansę większego wpłynięcia na współczesne życie. Stało się tak w wielu obszarach przemysłu. Wspomnieć należy o takich jak transport i komunikacja, teleinformatyka, więc dlaczego nie miałoby to nastąpić także i w branży spożywczej. Przecież nie do pomyślenia było kiedyś, że telefon będzie można nosić w kieszeni, a ludzie będą przemieszczać się nie tylko o własnych siłach. Tak jak w tamtych przypadkach było wiele głosów krytycznych, tak dzieje się także w przypadku omawianej tematyki. Nie powinno to jednak zrażać wizjonerów do kontynuacji opracowywania swoich koncepcji. Wszak to dzięki nim żyjemy w świecie pełnym wszelakich rozwiązań technologicznych ułatwiających funkcjonowanie.

Jedno jest pewne — wizja przyszłości tego rodzaju rozwiązań nie jest jeszcze do końca nakreślona. Tym samym niniejsza praca miała na celu pomoc w jej kreowaniu poprzez uwidocznienie problematyki zagadnienia.

BIBLIOGRAFIA

[1] S. Bobbe, „Inteligentna lodówka: co potrafi ten nowoczesny sprzęt AGD," Polska Press sp. z o.o., 28 Czerwiec 2016. [Online]. Available: http://regiodom.pl/portal/wnetrze/rtv-agd/inteligentna-lodowka-co-potrafi-ten-nowoczesny-sprzet-agd. [Data uzyskania dostępu: 5 Luty 2018].

[2] T. J. Hazen, „Microsoft and Liebherr Collaborating on New Generation of Smart Refrigerators," Microsoft, 2 Wrzesień 2016. [Online]. Available: https://blogs.technet.microsoft.com/machinelearning/2016/09/02/microsoft-and-liebherr-collaborating-on-new-generation-of-smart-refrigerators/. [Data uzyskania dostępu: 7 Luty 2018].

[3] A. Golański, „Inteligentne lodówki z modułami Microsoftu wiedzą, co mają w środku," DobrePrzepisy, 5 Wrzesień 2016. [Online]. Available: https://www.dobreprogramy.pl/Inteligentne-lodowki-z-modulami-Microsoftu-wiedza-co-maja-w-srodku,News,76123.html. [Data uzyskania dostępu: 19 Styczeń 2018].

[4] A. Linn, „Microsoft researchers win ImageNet computer vision challenge," Microsoft, 10 Grudzień 2015. [Online]. Available: https://blogs.microsoft.com/ai/microsoft-researchers-win-imagenet-computer-vision-challenge/. [Data uzyskania dostępu: 7 Luty 2018].

[5] Microsoft, „Przetwarzanie obrazów — przetwarzanie i analizowanie obrazów | Microsoft Azure," Microsoft, 2017. [Online]. Available: https://azure.microsoft.com/pl-pl/services/cognitive-services/computer-vision/. [Data uzyskania dostępu: 7 Luty 2018].

[6] K. Kawczyński, „Inteligentna lodówka od Samsunga z Androidem (wideo)," 29 Styczeń 2013. [Online]. Available: http://www.telix.pl/artykul/inteligentna-lodowka-od-samsunga-z-androidem--wideo--3,53022.html. [Data uzyskania dostępu: 22 Styczeń 2016].

[7] P. Gontarczyk, „Samsung T9000 z Androidem. Smartfon? Tablet? Nie... to lodówka,” 22 Styczeń 2013. [Online]. Available: http://pclab.pl/news52227.html. [Data uzyskania dostępu: 21 Styczeń 2016].

[8] Samsung, „Samsung T9000 Four-Door 766 Litre Refrigerator RF858VALASL - Samsung UK,” Samsung, 28 Grudzień 2014. [Online]. Available: http://www.samsung.com/uk/consumer/home-appliances/refrigeration/multi-door/RF858VALASL/EU. [Data uzyskania dostępu: 25 Styczeń 2016].

[9] C. Ry, „LG's newest fridges turn translucent when you knock,” CNET, 23 Sierpnia 2016. [Online]. Available: https://www.cnet.com/products/lg-lmxs30796d-instaview-door-in-door-refrigerator/preview/. [Data uzyskania dostępu: 17 Styczeń 2017].

[10] S. Zimkowska, „IFA 2016: Inteligentna lodówka LG z Cortaną na pokładzie,” Chip, 3 Wrzesień 2016. [Online]. Available: https://www.chip.pl/2016/09/ifa-2016-inteligentna-lodowka-lg-z-cortana-na-pokladzie/. [Data uzyskania dostępu: 14 Listopad 2017].

[11] S. Kupski, „IFA 2016: Inteligentna lodówka z wbudowanym wielkim tabletem z Windows 10,” Grupa WP, 5 Wrzesień 2016. [Online]. Available: https://tech.wp.pl/ifa-2016-inteligentna-lodowka-z-wbudowanym-wielkim-tabletem-z-windows-10-6034869423215233a. [Data uzyskania dostępu: 18 Styczeń 2017].

[12] Smarter, „Smarter FridgeCam,” Smarter, 2018. [Online]. Available: https://store.smarter.am/collections/frontpage/products/fridgecam. [Data uzyskania dostępu: 12 Luty 2018].

[13] R. Smithers, „The war on food waste has a new weapon: a £99 fridge camera,” Guardian News and Media, 13 Sierpień 2017. [Online]. Available: https://www.theguardian.com/environment/2017/aug/12/food-waste-smart-homes-fridge-cameras. [Data uzyskania dostępu: 8 Luty 2018].

[14] J. Rodway, „What's in Your Fridge? 6 Apps to Help You Manage the Refrigerator,” Apartment Therapy, 22 Maj 2013. [Online]. Available: https://www.apartmenttherapy.com/whats-for-dinner-6-fridge-management-apps-weekly-smartphone-app-roundup-189441. [Data uzyskania dostępu: 18 Styczeń 2017].

[15] Apple Inc., „App Store Preview,” 28 Luty 2017. [Online]. Available: https://itunes.apple.com/us/app/fridge-pal-shopping-lists/id496451091?mt=8. [Data uzyskania dostępu: 19 Styczeń 2017].

[16] Google, „Best Before - Aplikacje na Androida w Google Play,” 1 Sierpnia 2013. [Online]. Available: https://play.google.com/store/apps/details?id=th.co.crie.bestbefore. [Data uzyskania dostępu: 14 Marzec 2016].

[17] Microsoft, „keezeen for Windows 10 free download on Windows App Store,” 2018. [Online]. Available: http://keezeen.10appstore.net/win10apps.html. [Data uzyskania dostępu: 28 Styczeń 2018].

[18] Microsoft, „Uzyskaj produkt Whaz in the fridge — Sklep Microsoft pl-PL," CyberPhone, 2017. [Online]. Available: https://www.microsoft.com/pl-pl/store/p/whaz-in-the-fridge/9nblggh0jc85?rtc=1#. [Data uzyskania dostępu: 18 Styczeń 2017].

[19] K. Orzechowska, „Jesteś uzależniony od technologii? Grozi Ci samotność," BLOMEDIA.PL SP.Z O.O., 30 Kwiecień 2012. [Online]. Available: http://gadzetomania.pl/6385,jestes-uzalezniony-od-technologii-grozi-ci-samotnosc. [Data uzyskania dostępu: 4 Luty 2016].

[20] N. Ch., „Co czwarty Polak wyrzuca jedzenie," ddwloclawek.pl, 25 Sierpień 2016. [Online]. Available: https://ddwloclawek.pl/pl/546_ciekawostka/22175_co_czwarty_polak_wyrzuca_jedzenie.html. [Data uzyskania dostępu: 18 Luty 2018].

[21] Banki Żywności, „Banki Żywności apelują – nie marnuj jedzenia! Wyniki badania," Banki Żywności, 12 Październik 2017. [Online]. Available: https://bankizywnosci.pl/banki-zywnosci-apeluja-nie-marnuj-jedzenia-wyniki-badania-2/. [Data uzyskania dostępu: 18 Luty 2018].

[22] O. Wagner i K. Antoszewski, „BANKI ŻYWNOŚCI — Raport „Nie marnuj jedzenia 2017"," Październik 2017. [Online]. Available: http://www.siedlce.pl/components/download/send.php?pos_id=3286. [Data uzyskania dostępu: 18 Luty 2018].

[23] Greenpeace Polska, „Czas skończyć z marnowaniem żywności," Greenpeace Polska, 12 Kwiecień 2017. [Online]. Available: http://www.greenpeace.org/poland/pl/wydarzenia/polska/Czas-skonczyc-z-marnowaniem-zywnosci/. [Data uzyskania dostępu: 18 Luty 2018].

[24] S. D. Simone, „.NET Standard 2.0: Setting Expectations Straight," 28 Listopad 2016. [Online]. Available: https://www.infoq.com/news/2016/11/dotnet-standard-20-goals. [Data uzyskania dostępu: 9 Luty 2018].

[25] Microsoft, „.NET Standard | Microsoft Docs," Microsoft, 13 Sierpień 2017. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/net-standard. [Data uzyskania dostępu: 8 Luty 2018].

[26] I. Landwerth, „Announcing .NET Standard 2.0," Microsoft, 14 Sierpień 2017. [Online]. Available: https://blogs.msdn.microsoft.com/dotnet/2017/08/14/announcing-net-standard-2-0/. [Data uzyskania dostępu: 8 Luty 2018].

[27] R. Lander, „Announcing .NET Core 2.0 | .NET Blog," Microsoft, 14 Sierpnia 2017. [Online]. Available: https://blogs.msdn.microsoft.com/dotnet/2017/08/14/announcing-net-core-2-0/. [Data uzyskania dostępu: 8 Luty 2018].

[28] A. Boduch, Wstęp do programowania w języku C#, Październik: Helion, 2006.

[29] Microsoft, „Introduction to the C# Language and the .NET Framework," Microsoft, 20 Lipiec 2015. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework. [Data uzyskania dostępu: 8 Luty 2018].

[30] D. K., „Brutalne prawdy o MVC,” 30 Maj 2016. [Online]. Available: http://commitandrun.pl/2016/05/30/Brutalne_prawdy_o_MVC/. [Data uzyskania dostępu: 8 Luty 2018].

[31] VavaTech, „PHP - Poznaj ciekawy język programowania,” VavaTech, 7 Marzec 2018. [Online]. Available: http://vavatech.pl/technologie/architektura/mvc. [Data uzyskania dostępu: 9 Marzec 2018].

[32] Microsoft, „ASP.NET MVC Overview,” Microsoft, 2018. [Online]. Available: https://msdn.microsoft.com/pl-pl/library/dd381412(v=vs.108).aspx. [Data uzyskania dostępu: 8 Luty 2018].

[33] C. Petzold, „WinRT i MVVM,” w *Windows 8 — Programowanie aplikacji z wykorzystaniem C# i XAML*, Gliwice, Helion, 2013, p. 215.

[34] Microsoft, „The MVVM Pattern,” Microsoft, 10 Luty 2012. [Online]. Available: https://msdn.microsoft.com/en-us/library/hh848246.aspx. [Data uzyskania dostępu: 8 Luty 2018].

[35] P. Zieliński, „Wprowadzenie do wzorca projektowego Model-View-ViewModel na przykładzie aplikacji WPF,” Microsoft, 20 Styczeń 2012. [Online]. Available: https://msdn.microsoft.com/pl-pl/library/wprowadzenie-do-wzorca-projektowego-model-view-viewmodel-na-przykladzie-aplikacji-wpf.aspx. [Data uzyskania dostępu: 8 Luty 2018].

[36] S. Rizwan, „An Introduction to RESTful APIs,” Integration Zone, 30 Marzec 2017. [Online]. Available: https://dzone.com/articles/an-introduction-to-restful-apis. [Data uzyskania dostępu: 8 Luty 2018].

[37] Ecma International, „The JSON Data Interchange Syntax,” Ecma International, Grudzień 2017. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. [Data uzyskania dostępu: 8 Luty 2018].

[38] Microsoft, „Create a Web API with ASP.NET Core and VS Code | Microsoft Docs,” Microsoft, 22 Wrzesień 2017. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-vsc. [Data uzyskania dostępu: 9 Luty 2018].

[39] Microsoft, „Getting started with ASP.NET Core MVC and Visual Studio | Microsoft Docs,” Microsoft, 7 Listopad 2017. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?tabs=aspnetcore2x. [Data uzyskania dostępu: 9 Luty 2018].

[40] Microsoft, „Introduction to Razor Pages in ASP.NET Core | Microsoft Docs,” Microsoft, 12 Wrzesień 2017. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/mvc/razor-pages/?tabs=visual-studio. [Data uzyskania dostępu: 9 Luty 2018].

[41] D. Dobric, „Managed Extensibility Framework,” Microsoft, 2018. [Online]. Available: https://msdn.microsoft.com/de-de/library/ee332203.aspx. [Data uzyskania dostępu: 8 Luty 2018].

[42] Microsoft, „Managed Extensibility Framework (MEF) | Microsoft Docs," Microsoft, 30 Marzec 2017. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/mef/index. [Data uzyskania dostępu: 9 Luty 2018].

[43] J. L. Aasenden, „Inversion of control, dependency Injection, service oriented programming?," 13 Styczeń 2015. [Online]. Available: https://jonlennartaasenden.wordpress.com/2015/01/13/inversion-of-control-dependency-injection-service-oriented-programming/. [Data uzyskania dostępu: 8 Luty 2018].

[44] Wikipedia, „Android (system operacyjny) — Wikipedia, wolna encyklopedia," Wikipedia, 2018. [Online]. Available: https://pl.wikipedia.org/wiki/Android_(system_operacyjny). [Data uzyskania dostępu: 9 Luty 2018].

[45] Xamarin, „Getting Started with Xamarin.Forms — Xamarin," Xamarin, 2018. [Online]. Available: https://developer.xamarin.com/guides/xamarin-forms/getting-started/. [Data uzyskania dostępu: 9 Luty 2018].

[46] Xamarin, „.NET Standard 2.0 Support in Xamarin.Forms — Xamarin," Xamarin, 2018. [Online]. Available: https://developer.xamarin.com/guides/xamarin-forms/under-the-hood/net-standard/. [Data uzyskania dostępu: 9 Luty 2018].

[47] C. Mosers, „Introduction to XAML," 13 Grudzień 2013. [Online]. Available: https://wpftutorial.net/XAML.html. [Data uzyskania dostępu: 9 Luty 2018].

[48] Wikipedia, „HTML — Wikipedia," 2018. [Online]. Available: https://pl.wikipedia.org/wiki/HTML. [Data uzyskania dostępu: 9 Luty 2018].

[49] Wikipedia, „Kaskadowe arkusze stylów — Wikipedia," Wikipedia, 2018. [Online]. Available: https://pl.wikipedia.org/wiki/Kaskadowe_arkusze_styl%C3%B3w. [Data uzyskania dostępu: 9 Luty 2018].

[50] Wikipedia, „PHP — Wikipedia," Wikipedia, 2018. [Online]. Available: https://pl.wikipedia.org/wiki/PHP. [Data uzyskania dostępu: 9 Luty 2018].

[51] M. Chęciński, „JSON.NET: serializacja i przeszukiwanie danych," 21 Marzec 2017. [Online]. Available: https://michalchecinski.pl/json-net/. [Data uzyskania dostępu: 9 Luty 2018].

[52] Newtonsoft, „NuGet Gallery | Newtonsoft.Json," Newtonsoft, 9 Styczeń 2018. [Online]. Available: https://www.nuget.org/packages/newtonsoft.json/. [Data uzyskania dostępu: 9 Luty 2018].

[53] Code Hotfix, „Getting Started with Entity Framework Core," 23 Kwiecień 2017. [Online]. Available: http://codehotfix.com/getting-started-entity-framework-core/. [Data uzyskania dostępu: 9 Luty 2018].

[54] Microsoft, „Entity Framework Core Quick Overview," Microsoft, 27 Październik 2016. [Online]. Available: https://docs.microsoft.com/en-us/ef/core/. [Data uzyskania dostępu: 9 Luty 2018].

[55] Autofac Contributors, „Autofac: Home,” Autofac Contributors, 2018. [Online]. Available: https://autofac.org/. [Data uzyskania dostępu: 9 Luty 2018].

[56] J. Montemagno, „jamesmontemagno/MediaPlugin: Take & Pick Photos and Video Plugin for Xamarin and Windows,” 2018. [Online]. Available: https://github.com/jamesmontemagno/MediaPlugin. [Data uzyskania dostępu: 9 Luty 2018].

[57] Redth, „Redth/ZXing.Net.Mobile: Zxing Barcode Scanning Library for MonoTouch, Mono for Android, and Windows Phone,” 2018. [Online]. Available: https://github.com/Redth/ZXing.Net.Mobile. [Data uzyskania dostępu: 9 Luty 2018].

[58] Microsoft, „Azure SQL Database Documentation | Microsoft Docs,” Microsoft, 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/sql-database/. [Data uzyskania dostępu: 9 Luty 2018].

[59] Oracle, „MySQL,” Oracle, 2018. [Online]. Available: http://www.oracle.com/technetwork/database/mysql/index.html. [Data uzyskania dostępu: 9 Luty 2018].

[60] Microsoft, „Co to jest platforma Azure? — usługa firmy Microsoft w chmurze | Microsoft Azure,” Microsoft, 2018. [Online]. Available: https://azure.microsoft.com/pl-pl/overview/what-is-azure/. [Data uzyskania dostępu: 9 Luty 2018].

[61] Microsoft, „Azure Web Apps Documentation | Microsoft Docs,” Microsoft, 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/app-service/. [Data uzyskania dostępu: 9 Luty 2018].

[62] Wikipedia, „Microsoft Visual Studio — Wikipedia, wolna encyklopedia,” Wikipedia, 2018. [Online]. Available: https://pl.wikipedia.org/wiki/Microsoft_Visual_Studio. [Data uzyskania dostępu: 9 Luty 2018].

[63] Microsoft, „Download SQL Server Management Studio (SSMS) | Microsoft Docs,” Microsoft, 12 Grudzień 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms. [Data uzyskania dostępu: 9 Luty 2018].

[64] Microsoft, „What is Cortana?,” Microsoft, 30 Listopad 2017. [Online]. Available: https://support.microsoft.com/pl-pl/help/17214/windows-10-what-is. [Data uzyskania dostępu: 7 Luty 2018].

[65] S. Turkle, „Sherry Turkle,” MIT, 4 Luty 2015. [Online]. Available: http://www.mit.edu/~sturkle/. [Data uzyskania dostępu: 4 Luty 2016].

[66] S. Wrycza i B. Marcinkowski, Język inżynierii systemów SysML Architektura i zastosowania, Gliwice: Helion, 2010.

SPIS ILUSTRACJI

SPIS PRZYKŁADÓW

SPIS TABEL

Dodatek A.

WYKAZ UŻYTEJ TERMINOLOGII I SKRÓTÓW

AGD	Artykuły gospodarstwa domowego typu: lodówki, pralki, kuchenki, suszarki do włosów, zmywaki.
Android	System operacyjny firmy Google przeznaczony dla urządzeń mobilnych takich jak smartfony czy tablety.
Bazy produktów	Serwisy internetowe posiadające skatalogowane informacje dotyczące produktów.
Chmura publiczna	Zintegrowane usługi obliczeniowe oferowane przez dostawców dostępne w Internecie.
Cortana	Osobista asystentka systemu Windows posiadająca wbudowaną inteligencję, stworzona przez firmę Microsoft. Posiada możliwość rozpoznawania i odczytywania głosu użytkownika, jak i reagowania na przekazywane przez niego polecenia. [64]
Google Play®	Internetowy sklep firmy Google, z którego możliwe jest pobranie zarówno darmowych, jak i płatnych treści takich jak aplikacje, muzyka, książki czy filmy.
Interoperacyjność	Zdolność dwóch niezależnych systemów do wymiany oraz przetwarzania informacji.

iOS	System operacyjny dla urządzeń mobilnych firmy Apple.
iPad	Tablet produkowany przez firmę Apple.
iPhone	Smartfon produkowany przez firmę Apple.
iTunes Store®	Internetowy sklep firmy Apple sprzedający muzykę w postaci plików muzycznych.
Kod kreskowy	Kod jednowymiarowy w postaci graficznego zapisu kombinacji ciemnych i jasnych kresek.
Kod maszynowy	Szereg wykonywalnych rozkazów interpretowanych bezpośrednio przez procesor komputera stworzonych na podstawie skompilowanego kodu aplikacji.
Kod 2D	Kod dwuwymiarowy w postaci graficznego zapisu złożonego z czarno-białych kwadratów. Do najpopularniejszych kodów 2D możemy zaliczyć kody w formacie DataMatrix oraz QR.
Linux	Darmowy system operacyjny bazujący na systemie UNIX.
macOS	System operacyjny stworzony przez firmę Apple, przeznaczony dla komputerów Macintosh.
Metodyka DevOps	Metoda wytwarzania oprogramowania, która zwraca główną uwagę na komunikację, współpracę i integrację pomiędzy deweloperami a specjalistami z zakresu utrzymania IT.
Metro UI	Język służący do projektowania interfejsu, oparty na typografii, stworzony przez firmę Microsoft.
Microsoft® Azure	Platforma chmurowa firmy Microsoft, dostępna w Internecie, oparta o model PaaP (ang. ***P**latform **a**s **a** **S**ervice*).
NuGet	Darmowy menadżer komponentów stworzony dla firmy Microsoft, używany w środowisku programistycznym Visual Studio.
Open source	Określenie na wytwarzanie oraz dostarczanie darmowego oprogramowania wraz z jego kodem źródłowym.

Raspberry Pi	Platforma komputerowa składająca się z serii małych, pojedynczych płytek drukowanych.
RTV	Skrót określający przede wszystkim urządzenia typu odbiorniki radiowe oraz telewizyjne.
Strona internetowa	Dokument HTML utworzony trwale lub wygenerowany na podstawie przetwarzania danych, umiejscowiony na serwerze w sieci Internet.
Smartfon	Urządzenie przenośne, które łączy w sobie główne funkcje telefonu komórkowego oraz minikomputera z możliwością użytkowania go przy pomocy ekranu dotykowego.
Tablet	Płaski, przenośny komputer z obsługą dotykową ekranu o większych rozmiarach przekątnej.
Usługa internetowa	System dostępny w sieci Internet składający się z modułów dostarczających użytkownikom odpowiednich interfejsów do świadczenia im usług.
UWP	Jednorodna platforma architektury aplikacji stworzona przez firmę Microsoft.
Witryna internetowa	Zestaw powiązanych ze sobą tematycznie stron internetowych umieszczonych na jednym serwerze.
Witryna Marketplace	Rynek aplikacji oraz usług dostępnych przez Internet stworzony przez firmę Microsoft.
Wtyczka	Niezależne komponenty stworzone do współdziałania z głównym systemem, rozszerzające jego funkcjonalność.

Dodatek B.

OPIS ZAWARTOŚCI NOŚNIKA CD

Załączony do niniejszej pracy nośnik zawiera:

1) Dokument niniejszej pracy.
2) Kod źródłowy prototypu:
    a. Aplikacji webowej,
    b. Aplikacji mobilnej.
3) Kod źródłowy prototypowego środowiska systemów zewnętrznych.