



**WYŻSZA SZKOŁA  
INFORMATYKI i ZARZĄDZANIA**  
z siedzibą w Rzeszowie

## **KOLEGIUM INFORMATYKI STOSOWANEJ**

**Kierunek: INFORMATYKA**  
**Specjalność: Programowanie**

Yurii-Volodymyr Shcheliuk  
Nr albumu studenta: 58913

### ***System do obsługi klientów restauracji***

Promotor: Dr Marek Jaszuk

## **PRACA DYPLOMOWA INŻYNIERSKA**

**Rzeszów 2022**



## Spis treści

Wstęp .....	4
1. Wprowadzenie do problemu .....	5
2. Technologie informatyczne.....	7
2.1. Porównywanie narzędzi i technologii mobilnych.....	7
2.2. Platforma Xamarin.....	8
2.3. API i jego rodzaje .....	11
2.4. JWT.....	14
2.5. Postman.....	14
2.6. MSSQL Server.....	14
2.7. C#.....	15
2.8. .NET Core.....	15
2.9. Entity Framework Core .....	16
2.10. Angular.....	16
2.11. Wzorce architektoniczne.....	16
2.12. Wzorce projektowe .....	17
2.13. Schemat komunikacji.....	20
3. Część praktyczna.....	21
3.1. Analiza wymagań .....	21
3.2. Specyfikacja wymagań .....	21
3.3. Diagram przypadków użycia .....	22
3.4. Prototypy interfejsu .....	24
3.5. Implementacja.....	30
3.6. Opis działania aplikacji.....	35
3.7. Testy (ewaluacja).....	36
Podsumowanie .....	38
Literatura .....	39
Streszczenie.....	40
Załączniki.....	41

## Wstęp

Mobilne technologie rozwijają się krócej niż komputerowe, ale w różnych sferach życia są stosowane coraz częściej. Od dawna są używane nie tylko do rozmów ale oprócz tego do transmisji danych, opłaty rachunków, dokonania zakupów, operacji obliczeniowych itp. Zgodnie ze statystyką ponad połowa wszystkich zakupów online jest dokonywana z telefonu [WWW-1, 2022].

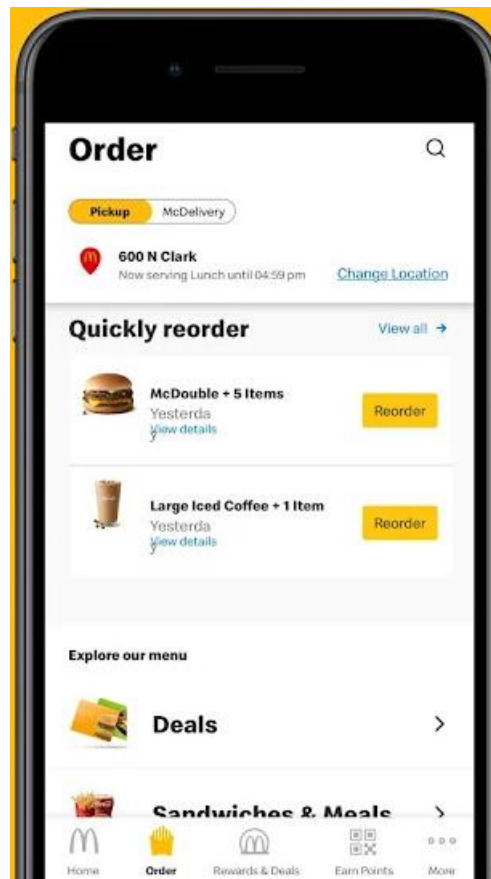
W pracy zostanie przedstawiony projekt, celem którego jest zmniejszenie ilości pracy pracowników obsługujących klientów restauracji, poprzez przeniesienie z rzeczywistości takich funkcjonalności jak składanie zamówienia, rezerwacja miejsca w restauracji oraz czat po opłacie do aplikacji mobilnej i webowej. Ma na celu założenie bazy do rozwoju systemu w kierunku pełnej automatyzacji oraz robotyzacji w wyniku czego zysk restauracji za pomocą aplikacji będzie maksymalnie wysoki a zasoby ludzkie będą minimalizowane. Zostaną zaimplementowane też administratorskie narzędzia, takie jak zarządzanie menu oraz przegląd danych zamówień wybranego dnia z możliwością wyeksportowania ich do pliku, a dzięki wykorzystanym wzorcom architektonicznym oraz projektowym, aplikacja będzie łatwo wspierana oraz rozbudowana w przyszłości dla klientów, którzy będą chcieli kupić licencję lub jakieś osobne moduły, na przykład moduł finansów, zarządzania produktami lub dostawy zamówień itp.

Projekt jest oparty o Xamarin.Forms, .NET Core API oraz Angular. Są to narzędzia, za pomocą których zostanie stworzony system z kilku aplikacji przeznaczonych na różne platformy jednocześnie, czyli Android, iOS oraz przeglądarki webowe. W ten sposób dostać się do usług dostępnych w aplikacji można też z różnych systemów operacyjnych. Biblioteka Xamarin.Forms umożliwia tworzenie krosplatformowych aplikacji mobilnych. W frameworku Angular zostanie stworzona strona internetowa dla przeglądarek. Backend występujący jako API dla klientów jest oparty o .NET Core 3.1 z wykorzystaniem mniejszych bibliotek, takich jak Swagger, AutoMapper i innych. Dzięki podobnemu do mikroserwisowego rozwiązaniu, rozwój i wspieranie będzie pochłaniało dużo mniej kosztów i czasu.

Praca zostanie podzielona i przedstawiona w trzech rozdziałach. W pierwszej części pracy będą przeanalizowane aplikacje już istniejące na rynku, a w następnej części, czyli teoretycznej, zostaną omówione wykorzystywane narzędzia oraz ich zastosowanie, będą przeanalizowane ich wady i zalety w porównaniu do innych instrumentów. W ostatniej części, czyli praktycznej zostanie przedstawiony prototypy interfejsu, będzie opisana interakcja użytkownika z programem oraz implementacja aplikacji.

## 1. Wprowadzenie do problemu

Aplikacje do obsługi restauracji są powszechnie wykorzystywane w większych korporacjach, takich jak McDonald's, KFC, Subway i wiele innych. Przyczyną tego jest fakt, że telefony są zawsze pod ręką. Z ich pomocą łatwo można złożyć zamówienie lub dokonać innych funkcji, które kiedyś były możliwe tylko na komputerze. Podejście realizacji aplikacji mobilnej przynosi korzyści instytucji, która używa danej aplikacji. Nie traci się czasu na komunikację i złożenie zamówienia, nie tworzy się kolejki do kasy, a menu jest zawsze pod ręką. Nie mniej ważny powód to marketing i ładny wygląd aplikacji, który przyciąga klientów. Z punktu widzenia marketingu, różne punkty, które można zdobyć przy zakupie też motywują aby wrócić jeszcze raz do tej aplikacji by dostać rabat lub jakąś nagrodę jak jest to w aplikacji McDonald's. Jest to wygodniejsze niż na przykład zbierać naklejki, które się gubią.



Rys. 1 Aplikacja mobilna McDonald's

Źródło: [https://play.google.com/store/apps/details?id=com.mcdonalds.app&hl=en\\_IN&gl=US](https://play.google.com/store/apps/details?id=com.mcdonalds.app&hl=en_IN&gl=US), z dnia 05.12.2021

Na Rys.2 aplikacja Subway pokazuje to, jak można zaimplementować bardzo ładny widok produktu z pełną kontrolą tego, co zamawiamy. Oczywiście że dostęp do żądanych funkcji nie powinien być zbyt skomplikowany.



Rys. 2 Aplikacja mobilna Subway

Źródło: <https://www.appstoreapps.com/app/subway/>, z dnia 05.12.2021

Aplikacje różnego przeznaczenia są publikowane każdego dnia w tym aplikacje służące do komunikacji online, rezerwacji rzeczy lub świadczenia usług dla różnych potrzeb [WWW-2, 2021]. Problemy które rozwiązuje realizowana aplikacją to automatyzacja wymienionych procesów biznesowych celem czego jest skrócenie czasu oczekiwania oraz zamówienia. Oprócz tego, robiąc wnioski z powyższych aplikacji, korzystanie z programu powinno być łatwe do przyzwyczajania się ze strony użytkownika, czyli interfejs musi być intuicyjny, ma sprawnie działać na różnych wersjach i systemach operacyjnych oraz łatwa w wspieraniu i rozwoju ze strony deweloperów.

## 2. Technologie informatyczne

Przed wyborem technologii do implementacji aplikacji warto zwrócić uwagę na docelowe systemy operacyjne. Najpopularniejszym mobilnym systemem operacyjnym na świecie, który pojawił się jako projekt open-source na licencji Apache 2.0 jest Android. W tej chwili ilość zarejestrowanych użytkowników korzystających z tego systemu jest większa w porównaniu z systemem Windows o 7%, a od iOS jest popularniejszy 2 razy. Android, który jest najpopularniejszy stanowi 39.75%, 32.44% rynku zajmuje Windows oraz 16.7% iOS i inne [WWW-3, 2021]. System został stworzony przez Android Inc., który następnie został kupiony przez Google w 2005. We wrześniu 2008 była przedstawiona pierwsza wersja systemu. W 2009 roku została wydana aktualizacja pod nazwą Cupcake i wszystkie następne wersje zaczynały się kolejną literą łacińskiego alfabetu nazwą jakiegoś deseru, czyli: 1.6 Donut, 2.0/2.1 Eclair i tak do wersji 9.0 Pie, a dalej tylko numeracją. W czasie ewolucji pokonał konkurentów jak Windows Phone, BlackBerry oraz Symbian firmy Nokia. Wśród mobilnych systemów operacyjnych obecnie jedynym konkurentem tego systemu Linuxowego jest produkt firmy Apple, czyli iOS.

Co do historii iPhone OS można powiedzieć, że w swoich czasach to była naprawdę rewolucja w świecie technologii. Na pierwszy rzut oka te smartfony podbijają niezrównanym designem zarówno samego telefonu, jak i systemu operacyjnego. Oprócz wyglądu i interfejsu jest możliwość dokonania transakcji bezdotykowych, FaceID, Siri i inne nowoczesne technologie. Owszem, że takie same możliwości są na telefonach z systemem operacyjnym Android, na przykład Samsung, Xiaomi, Google Pixel lub inne. Jednak Apple udało się dopracować te technologie do perfekcji, mimo tego, że jak i w każdym innym telefonie są wady jak i zalety.

Obecność systemu operacyjnego jest główną cechą wyróżniającą smartfon od zwykłego telefonu komórkowego. Przy wyborze konkretnego systemu operacyjnego jest często czynnikiem decydującym o modelu telefonu lub innego urządzenia. Najpopularniejsze systemy operacyjne dla smartfonów to:

1. Android - system operacyjny też dla tabletów, e-booków, odtwarzaczy cyfrowych, zegarków i notebooków. Oparty na jądrze Linux. Po nabyciu praw, Google utworzyło organizację Open Handset Alliance (OHA), która obecnie wspiera i rozwija platformę. Android co pozwala na tworzenie aplikacji opartych na Javie, które sterują urządzeniem za pomocą bibliotek Android NDK (*ang. Android Native Development Kit*) opracowanych przez Google, napisanych w językach C i C++;
2. iOS (iPhone OS do 24 czerwca 2010) to mobilny system operacyjny, opracowany i produkowany przez amerykańską firmę Apple. Został wydany w 2007 roku; początkowo dla iPhone 'a i iPod'a Touch a później dla urządzeń takich jak iPad i Apple TV. W przeciwieństwie do Google Android, jest on dostępny tylko dla urządzeń firmy Apple;

W następnym podrozdziale zostaną opisane wybrane technologie w porównaniu do innych możliwych rozwiązań oraz podejścia architektoniczne i projektowe realizowanego systemu. Będzie opisana analiza rynku pod kątem celów wykorzystania aplikacji dla danej branży.

### 2.1. Porównywanie narzędzi i technologii mobilnych

Języki programowania mobilne nieco się różnią od języków programowania systemów desktopowych lub webowych. Wynika to z tego, że logika która jest napisana jako reakcja na działania użytkownika najczęściej jest powiązana z językiem XAML, który służy do tworzenia UI. Przy wyborze technologii do implementacji porównywałem następujące instrumenty:

1. Kotlin – język programowania działający na JVM systemu Android z funkcją pod nazwą „Multiplatform”, która pozwala na wykorzystanie kodu też dla systemu iOS. Jest wydajniejszy w porównaniu do aplikacji cross-platformowych, ponieważ natywne aplikacje pozwalają na uzyskanie bezpośredniego dostępu do podzespołów urządzenia, na przykład aparatu, GPS, czy mikrofonu w bardziej wydajny sposób. Jest to istotne ze względu na szybkość działania i ma to istotny wpływ na ogólną jakość aplikacji.
2. Swift – służy zasadniczo do tworzenia aplikacji na iOS i inne produkty Apple. Od wersji 2.2 pozwala na napisanie aplikacji na Linux, a od wersji 5.3 na Windows. Aktualną wersją tego języka jest wersja 5.5, od września 2020 roku.

Używa kodowania UTF-8, aby zoptymalizować wydajność dla szerokiej gamy przypadków użycia. Pamięć jest zarządzana automatycznie przy użyciu ścisłego, deterministycznego zliczania referencji, utrzymując zużycie pamięci na minimalnym poziomie bez kosztów ogólnych zbierania śmieci.

3. React Native - Łączy elementy natywnego programowania z React i biblioteką JavaScript do budowania interfejsów użytkownika. Obecnie jest wersja 0.66, mimo upływu 7 lat od premiery. Oznacza to, że framework nie jest stabilny i wciąż nie wiadomo, kiedy będzie nowa odsłona wersji. Mimo faktu, że jest ciągle rozwijany, deweloperzy na całym świecie (Facebook pierwszy w kolejce) używają ją w dużej skali
4. Xamarin – ta technologia pozwala na tworzenie aplikacji cross-platformowych. Zaletą technologii możliwość zaoszczędzenia np. 30-40% budżetu tylko na napisaniu kodu, ponieważ pierwsze wersje projektu będą wydane na Androida i iOS w dość krótkim okresie, jeśli budżet jest ograniczony a dodatkowo czas nie pozwala na wymyślnie czasochłonnych technologii. W 2016 roku Microsoft kupiła Xamarin, czyli projekt Mono za \$400 mln i zrobiła go darmowym i open-source.
5. Flutter - według badań przeprowadzonych w 2021 roku przez niemiecką firmę „Statista”, okazał się najpopularniejszym hybrydowym frameworkiem mobilnym preferowanym przez deweloperów na całym świecie, z oszałamiającym udziałem w rynku wynoszącym 42%. Pozwala tworzyć na różne platformy przy użyciu tej samej bazy kodu i zachować natywną wydajność. Mnóstwo wbudowanych widżetów i hot reload.
6. MAUI - open-source'owy, cross-platformowy framework do tworzenia natywnych aplikacji mobilnych i desktopowych za pomocą XAML i C# docelowo na wersji minimalnej .NET 5 lub 6. Pierwszy release odbył się w listopadzie 2021, twierdząc, że jest to ewolucja ich wcześniej znanego hybrydowego frameworku o nazwie Xamarin. .NET MAUI dodał do tego zestawu wsparcie dla rozwoju aplikacji desktopowych. Zestaw narzędzi .NET MAUI zastąpił tradycyjne zestawy narzędzi Xamarin [WWW-4, 2021].

## **2.2. Platforma Xamarin**

### **2.2.1. Składnia Xamarin**

Kluczowe elementy, które pozwalają na tworzenie aplikacji dla kilku systemów:

- Język C# - umożliwia znajomą składnię oraz zaawansowane funkcje, takie jak uniwersalne szablony, LINQ oraz bibliotekę zadań asynchronicznych.
- Mono Framework - zapewnia multiplatformową implementację zaawansowanych funkcji Microsoft .NET Framework dla Xamarin
- Kompilator - w zależności od platformy, tworzy aplikację podobną do natywnej na przykład na iOS lub zintegrowaną aplikację .NET i runtime dla Androida. Kompilator wykonuje również wiele optymalizacji dla wdrożeń mobilnych, takich jak usuwanie nieużywanego kodu.
- Narzędzia IDE - Visual Studio na Mac i Windows pozwala na tworzenie, budowanie i wdrażanie projektów Xamarin.

Chociaż Xamarin pozwala na tworzenie aplikacji w C# i współdzielenie tego samego kodu na wielu platformach ale konfiguracja różnych ustawień, takich jak tytuł wyświetlanej nazwa aplikacji, ikona, numery wersji i wszelkie kody, które są potrzebne dla iOS lub Android i inne specyficzne rzeczy, które nie są dostępne z cross-platformowego wdrożenia, są różne na każdym systemie.

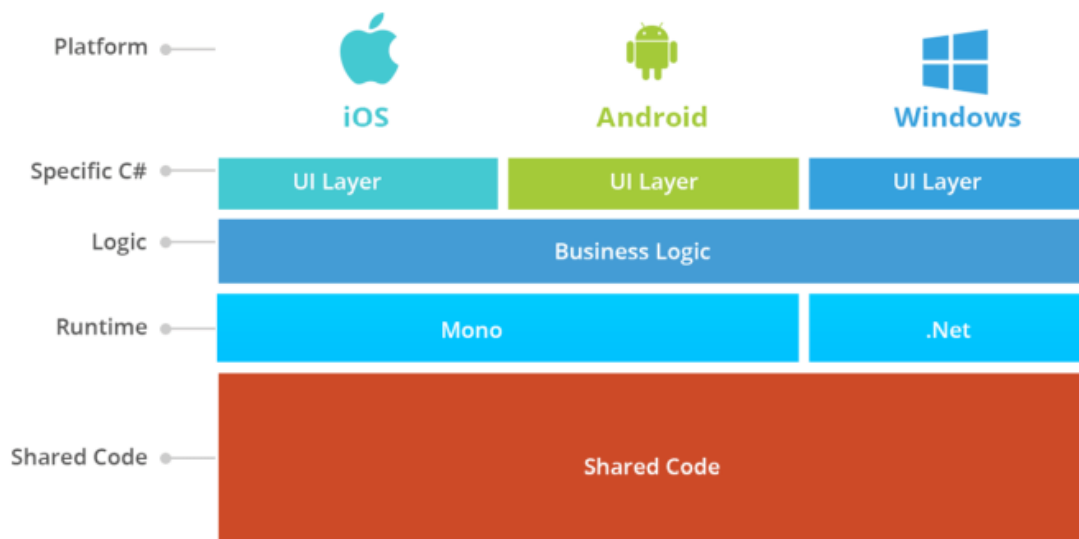
### **2.2.2. Wady i zalety**

Istnieje kilka powodów, korzystania z Xamarin:

1. Jeden stos technologiczny dla rozwoju na wszystkich platformach. Xamarin wykorzystuje język C# oraz .NET Framework do tworzenia aplikacji na dowolną platformę mobilną. Można ponownie wykorzystać około 30% swojego kodu źródłowego, przyspieszając tym samym proces tworzenia oprogramowania. Xamarin nie wymaga również przełączania się pomiędzy środowiskami



programistycznymi: wszystkie aplikacje Xamarin mogą być tworzone przy użyciu narzędzi Visual Studio. Narzędzia do programowania wieloplatformowego są dostarczane jako wbudowana część IDE bez dodatkowych kosztów.



Rys. 3 Działanie aplikacji krosplatformowej na bazie .NET

Źródło: <https://www.nexgendesign.com/xamarin-troubles>, z dnia 15.12.2021

- Wydajność jest zbliżona do natywnej. Aplikacje wieloplatformowe tworzone za pomocą Xamarin można zakwalifikować jako natywne, w przeciwieństwie do tradycyjnych hybrydowych rozwiązań webowych. Metryki wydajności są porównywalne z Javą dla Androida i Objective-C oraz Swift dla rozwoju aplikacji iOS. Co więcej metryki wydajności Xamarin były stale poprawiane i udoskonalane, aby w pełni zgodnym standardom natywnego rozwoju. Platforma Xamarin oferuje kompleksowe rozwiązanie do testowania i śledzenia wydajności aplikacji: „Xamarin Test Cloud” w połączeniu z narzędziem Xamarin Test Recorder pozwala na przeprowadzanie zautomatyzowanych testów UI i wyszukiwanie problemów z wydajnością jeszcze przed wydaniem aplikacji. Usługa ta jest dostępna za dodatkową opłatą.
- Native UI. Używając elementów UI zależnych od platformy Xamarin pozwala na stworzenie interfejsu. Zalecane jest używanie Xamarin.iOS i Xamarin.Android osobno dla lepszego rozwoju UI aplikacji. Zapewnia to lepsze wyniki.
- Kompatybilność sprzętu. Xamarin eliminuje problemy związane z kompatybilnością sprzętową poprzez wykorzystanie pluginów i różnych API do obsługi wspólnych funkcji urządzeń na wszystkich platformach. Wraz z dostępem do API specyficznych dla danej platformy Xamarin może być połączony z bibliotekami specyficznymi dla tej platformy. Pozwala to na lepszą konfigurację i wsparcie dla specyficznych dla danej platformy funkcji warstwy natywnej.
- Łatwe wsparcie. Dzięki swojej międzyplatformowej naturze, Xamarin ułatwia właśnie wsparcie i rozwój oprogramowania. Można wprowadzić zmiany w jednym pliku źródłowym i będą one stosowane zarówno w aplikacjach na iOS, jak i na Androida. Dotyczy to jednak tylko aplikacji korzystających ze wspólnej logiki biznesowej i także wspólnego kodu dla aplikacji Xamarin.iOS i Xamarin.Android. Pomoże to zaoszczędzić czas i pieniądze podczas utrzymywania aplikacji w ruchu.
- Pełny zestaw narzędzi programistycznych. Xamarin w jednym pakiecie zawiera pełen zestaw narzędzi deweloperskich: natywne IDE (Visual Studio), Xamarin SDK, testowanie (Xamarin Test Cloud), dystrybucję i analitykę (Hockeyapp i Xamarin.Insights). Nie ma więc potrzeby inwestowania w dodatkowe narzędzia lub integrowania innych aplikacji do tworzenia, testowania i wdrażania aplikacji Xamarin.

Mimo swoich zalet, utrudniają tworzenie aplikacji na Xamarinie niektóre wady platformy. Oto główne z nich:

1. Ograniczony dostęp do bibliotek typu open-source. Natywny rozwój szeroko wykorzystuje technologie open source. Z Xamarinem jest konieczne korzystanie z komponentów dostarczanych przez platformę i niektóre zasoby .NET, z których korzystają programiści i konsumenci. Wybór z pewnością nie jest tak duży, jak w przypadku tworzenia natywnych aplikacji mobilnych na iOS i Android, ale komponenty Xamarin zapewniają tysiące różnych elementów interfejsu: diagramy i grafiki, motywy i inne przydatne funkcje, które można dodać do każdej aplikacji za pomocą kilku kliknięć. Platforma ta obejmuje wbudowane funkcje przetwarzania płatności pod nazwą Stripe, obsługę sygnałów nawigacyjnych, usługi powiadomień Push, rozwiązania do przechowywania w chmurze, możliwości multimedialne, streaming i wiele innych.
2. Opóźnienia w aktualizacjach platformy. To zależy wyłącznie od zespołu developerów Xamarina. Mimo to, że Xamarin twierdzi, że zapewnia wsparcie tego samego dnia, nadal może wystąpić opóźnienie. Też nie jest możliwe, aby narzędzia "firm trzecich" zapewniały natychmiastowe wsparcie dla wydania najnowszych aktualizacji iOS i Android: wprowadzenie nowych zmian, nowych wtyczek itp. zajmuje trochę czasu.
3. Ograniczenia ekosystemu. Choć platforma Xamarin jest wspierana przez Microsoft ale społeczność związana z Xamarin jest znacznie mniejsza niż innych grup programistów, więc znalezienie doświadczonego programisty może być trudne. Na podstawie informacji z różnych źródeł społeczność Xamarin stanowi 10% całej społeczności programistów mobilnych. Choć liczba inżynierów Xamarin nie jest porównywalna z liczbą specjalistów iOS czy Android, twórcy platformy zapewniają wsparcie dla swoich specjalistów. Na przykład istnieje dedykowana instytucja edukacyjna Xamarin University, która zapewnia wiele zasobów i możliwości praktycznych szkoleń dla profesjonalistów z branży. Dzięki takiemu wsparciu, krzywa uczenia się jest minimalna dla doświadczonych programistów C#.
4. Xamarin nie nadaje się do aplikacji z wysokowydajną grafiką. Główną zaletą Xamarin jest możliwość wykorzystania kodu na różnych platformach. Chodzi tu o logikę, kod UI będzie w większości unikalny dla danej platformy. Pozwala to na tworzenie gier w Xamarin, ale bogate UI lub złożone animacje z niewielką ilością kodu generycznego sprawiają, że Xamarin nie nadaje się do tego.
5. Większe zasoby. W zależności od rodzaju i złożoności, aplikacje Xamarin są zazwyczaj większe od aplikacji natywnych, czasami nawet dwukrotnie większe. Na Androidzie, proste „Hello, World” może zająć do 16 MB, z czego większość jest wykorzystywana przez powiązane biblioteki, środowisko Mono i bibliotekę klas bazowych (*ang. Base Class Library*). W związku z tym aplikacja zazwyczaj wymaga dodatkowej optymalizacji, aby utrzymać jej rozmiar pliku na niskim poziomie.

### 2.2.3. Kompilacja

Kod źródłowy C# trafia do natywnej aplikacji w różny sposób na każdej platformie:

- Android - aplikacja działa równolegle z Java/ART (*ang. Android Runtime*) i współdziela z natywnymi typami poprzez JNI (*ang. Java Native Interface*). C# jest kompilowany do Intermediate Language i budowany z MonoVM plus JIT. Popiera się przez interfejsy oraz biblioteki Android SDK od Google. Nieużywane klasy w platformie są usuwane podczas przetwarzania;
- iOS - C# używa wstępnej kompilację (*ang. Ahead of Time Compilation*) do języka assembler. Framework włącza się i nieużywane klasy są usuwane podczas przetwarzania w celu zmniejszenia rozmiaru aplikacji, czyli tak samo, jak na Androidzie. Mono uwidacznia struktury zestawu SDK CocoaTouch od Apple, do których można się odwoływać w Xamarin, ale niektóre funkcje Apple nie pozwala do wykorzystania w tym refleksji lub dynamiczne generowanie kodu w czasie rzeczywistym w iOS [WWW-5, 2021].

## 2.3. API i jego rodzaje

API (*ang. Application programming interface*) – opis zachowania się aplikacji i w jaki sposób ona może komunikować się z innym programem i operować danymi lub innymi serwisami. W miarę standaryzacji komunikacji sieciowej informacje były wymieniane cyfrowo przez linie telefoniczne i przewody sieciowe przy użyciu protokołów ogólnego przeznaczenia, takich jak Telnet, SMTP, FTP i HTTP. Na początku był stworzony i używany XML, który jest nadal szeroko stosowany. Jednak teraz w większości przypadków jest wykorzystywany JSON, oparty na tekście. Mniej popularne formaty binarne, na przykład Protocol Buffers i Thrift.

### 2.3.1. SOAP

Simple Object Access Protocol (SOAP) - jest protokołem wymiany informacji zakodowanej w XML (*ang. Extensible Markup Language*) pomiędzy klientem i procedurą lub usługą, która znajduje się w Internecie. Został wydany w świat w 1999 roku i jest publikowany przez W3C jako otwarty standard.

SOAP może być używany przez różne protokoły transportowe oprócz HTTP, na przykład FTP i SMTP. (Klasycznym wzorcem jest użycie HTTP do synchronicznej wymiany danych i SMTP lub FTP do interakcji asynchronicznych).

W celu zapewnienia spójności podczas strukturyzacji danych SOAP wykorzystuje standardowy schemat XML (XSL) do kodowania XML. Dodatkowo programiści mogą tworzyć własne schematy XML, aby dodać niestandardowe elementy XML do wiadomości SOAP.

SOAP jest zwykle używany z językiem opisu usług sieciowych WSDL (*angl. Web Services Description Language*). Ważność użycia WSDL polega na tym, że programiści i maszyny mogą sprawdzić usługę sieciową, która obsługuje SOAP, aby odkryć specyfikę wymiany informacji przez sieć. Ponadto WSDL opisuje, w jaki sposób strukturyzować komunikaty żądania i odpowiedzi SOAP, które dana usługa obsługuje. Odkrywanie poprzez WSDL upraszcza programowanie usług sieciowych wykorzystujących SOAP.

Zalety:

- SOAP pozwala na generowanie kodu opierając się na XML. XML jest dobrze znany i wszechobecny z dużą elastycznością (na przykład, przestrzenie nazw).
- Ponieważ jest oparty na tekście, debugowanie jest wygodne i nie ma ograniczeń dotyczących transportu. Podejścia do napisania własnych plików są opisane w WSDL, plik działa jako wiążący kontrakt dla przesyłania wiadomości i typów.
- Dodatkowa warstwa WS-Security zapewniana przez SOAP działa na poziomie wiadomości, aby upewnić się, że nie tylko treść wiadomości może być odczytana przez właściwy serwer, ale także właściwy proces na serwerze.

Wady:

- Działa tylko z XML, a XML jest bardzo dosłowny i pomiar wiadomości ma tendencję do wzrostu eksponencjalnego.
- Pliki są ciężkie ze względu na duży rozmiar XML tekstu.
- Wsparcie dla technologii kodu w obecnych językach jest ograniczone do niedużej ilości języków, na przykład to Java, Python i C#, podczas gdy SOAP przez Go/Rust/PHP/Elixir nie jest obsługiwany,

### 2.3.2. gRPC

gRPC, czyli Remote Procedure Call od Google jest akronimem rekurencyjnym dla metody zdefiniowanej wokół wszystkich zalet HTTP/2, ponieważ używa binarnego formatu do transferu, który jest niezwykle wydajny. Celem gRPC jest umożliwienie wywoływania procedur, a nie interakcji danych, co czyni go nieco podobnym do SOAP w koncepcji.

Zalety:

- Bardzo wydajny. Wykorzystuje wszystko co HTTP/2 ma do zaoferowania w tym, że można wysyłać rzeczy synchronicznie (czekając na odpowiedź), strumieniować, multipleksować, wszystko przez to samo połączenie. Dodatkowo wiadomości binarne są bardzo małe, które nie obciążają wydajności.

- Generowanie kodu ułatwia konfigurację i HTTP/2 wymaga TLS i pozwala na żądanie wielu plików w tym samym czasie.

Wady:

- Trudny do debugowania, ponieważ komunikaty są binarne i nie są czytelne dla człowieka. Chociaż jest obsługiwany we wszystkich językach, niektóre mają ograniczenia (np. nie można mieć serwerów PHP gRPC, tylko klientów), i ze względu na naturę HTTP/2 nie ma wsparcia (obecnie) bezpośrednio w aplikacjach frontendowych.
- Ponieważ musi być TLS wszędzie, konieczne jest to, że trzeba uwzględnić też w wymaganiach konfiguracyjnych obecność TLS protokołu lub nabywając certyfikaty do użytku wewnętrznego od znanego CA (*ang. Certificate Authority*), lub tworzyć własny wewnętrzny CA do wydawania certyfikatów, ale musimy dodać go jako zaufany w całej infrastrukturze/kontenerach/itd.

### 2.3.3. GraphQL

GraphQL - produkt Facebooka, plasuje się gdzieś pomiędzy REST i gRPC. Jego celem jest ułatwienie żądania danych poprzez własny język zapytań, który daje kontrolę klientowi. Komunikuje się za pomocą protokołu HTTP i wykorzystuje format danych JSON.

Zalety:

- Jego oparcie na JSON czyni go nieco podobnym do REST, ale z dodatkową korzyścią, że adaptacyjne zapytania oznaczają, że można poprosić o dane, które chcesz, jak chcesz nie ma dużo wielokrotnych zapytań, aby uzyskać wszystko, czego potrzebujesz.
- Przy debugowaniu dostajemy detaliczny opis problemu, czyli w tym dodatkowo referencję do połączenia i całą informację z handlera.
- Posiada walidację schematu i typowanie, więc w tym sensie jest nieco zbliżony do tego, jak gRPC definiuje rzeczy.

Wady:

- Cała ta elastyczność odbija się na zdolności do buforowania i jest słaby wydajnościowo. Wynika to z tego jak detaliczną i ogromną informację dostaje się przy zapytaniach.
- Pomimo nazwy, nie jest to do końca interfejs grafowy. Nie można na przykład pobrać wszystkich przodków jednostki nadrzędnej.
- Nie ma dużo informacji o tworzeniu serwisów na podstawie tego API.

### 2.3.4. REST

REST jest akronimem od Representational State Transfer, jest stylem architektonicznym opracowanym przez Roya Fieldinga w jego pracy doktorskiej z 2000 roku. W REST wszystko odbywa się przy użyciu metod HTTP, takich jak:

- GET - odczyt;
- POST – tworzenie i odczyt;
- PUT - modyfikacja;
- DELETE - usuwanie
- PATCH - częściowe modyfikacje zasobu
- OPTIONS – przekazywanie metadanych
- HEAD – odczyt, ale bez odpowiedzi

Używanie tekstowych formatów danych stało się konwencją. JSON jest najbardziej popularnym formatem danych, chociaż można używać innych, takich jak XML, CSV, i nawet RSS. W taki sposób jak przedstawiono niżej wyglądają zapytania i odpowiedzi z i do serwera, którą otrzymuje klient aplikacji.

```
curl -X POST "http://localhost:49887/api/Accounts/Login" -H
"accept: application/json" -H
"Content-Type: application/json-patch+json"
-d
{
    "Email": "client@email.com",
    "Password": "Aacc&556"
}
```

```
HTTP/1.1 200 OK
ETag: "b8a7ef8b4b282a70d1b64ea5e79072df"
X-Runtime: 13
Cache-Control: private, max-age=0, must-revalidate
Content-Length: 449
Status: 200
Keep-Alive: timeout=2, max=100
Content-type: application/problem+json; charset=utf-8
{
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJjbGllbnRAZW1haW-wuY29tIiwianRpIjoiaZTdhZTg3NDQtOGI5OC00OWNiLTllYzgtNDUwMGQyZ-GEyNTE1IiwiaWwiOiJjbGllbnRAZW1haW-wuY29tIiwiaHR0cDovL3NjaGVtYXMuZG1sc29hcC5vcmcvd3MvMjAwNS8wNS9pZGV-udGl0eS9jbGFpbXMvbmFtZSI6ImNsaWVudEB1bWFpbC5jb20iLCJodHRwOi8vc2NoZW1hcy5taW-Nyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJvc2VyliwibmJmI-joxNjQzMjY0NTcxLCJleHAiOiJlbnR5cCI6ImxvY2FsaG9zdC5jb20iLCJhdWQiOiJsb2Nhbnhvc3QuY29tIn0.U1UdWYMMna8yaWvnSYXNv1Qyu7PB_JcHcp7NrBFcy9Y",
    "token_type": "bearer",
    "user_Id": 3,
    "user_name": "client",
    "expires_in": 86400,
    "creation_Time": 1643064571,
    "expiration_Time": 1643150971
}
```

#### Zalety:

- Bazowanie na istniejących standardach HTTP wiąże się zarówno z elastycznością jak i ograniczeniami. Większość problemów związanych z transportem jest obsługiwana przez istniejące standardy i ewoluuje wraz z nimi. Czasowniki są powiązane z operacjami (co ułatwia mapowanie operacji CRUD na przykład), encje nadają nazwy i znaczenie URI. Proste konwencje sprawiają, że interakcja z interfejsami API REST jest dość prosta.
- JSON sprawia, że REST jest szczególnie prosty w użyciu dla usług frontendowych, gdzie JS jest wszechobecny, tak że tłumaczenie danych na obiekty jest natychmiastowe.
- Ponadto JSON jest dość czytelny dla człowieka i łatwy do debugowania.

#### Wady:

- W praktyce, usługi REST rzadko są tak przyjazne jak zamierzone, ponieważ nie ma wspólnej koncepcji tworzenia. Elastyczność często przekłada się na chaos implementacyjny i na dodatek nie ma wiążącego kontraktu na strukturę używaną w wiadomościach.
- HTTP - aplikacje RESTful są ograniczone do protokołu HTTP

- Gdy ludzie zwracają odpowiedź HTTP 200 w każdym przypadku i zamiast tego kodują status błędu w ciele odpowiedzi to tu leży pis pogrzebany. To jest to, co ludzie SOAP zwykli robić.

Właśnie w tym projekcie będzie wykorzystywany REST API napisany tworzony w .NET Core do komunikacji z aplikacją na Xamarinie i Angularze za pomocą HTTP zapytań z powodu wyżej opisanych zalet.

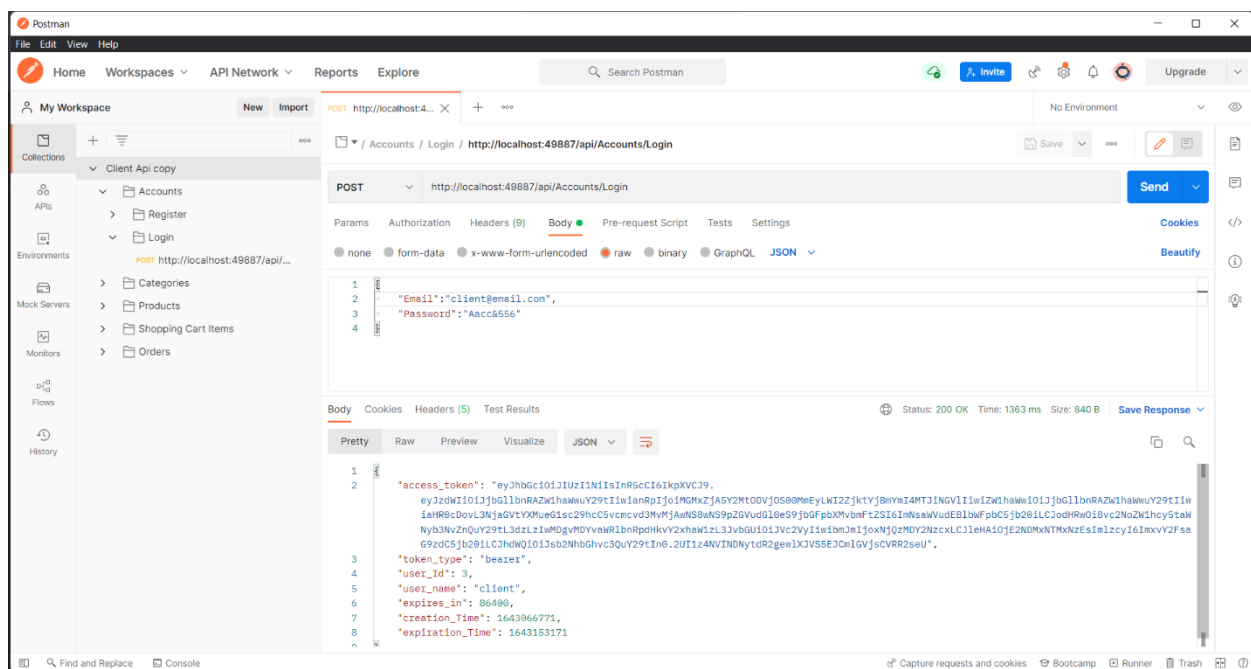
## 2.4. JWT

Tak jak REST nie mają dobrze zdefiniowanego protokołu bezpieczeństwa, JSON Web Tokens (JWTs) są najbardziej powszechną metodą uwierzytelniania i autoryzacji żądań.

JWT tokeny (*ang JSON Web Token*) jest dobrym sposobem na bezpieczne przesyłanie informacji pomiędzy stronami, ponieważ mogą być podpisane co oznacza, że można mieć pewność, że nadawcy są tymi, za których się podają. Dodatkowo, struktura JWT pozwala zweryfikować, czy treść nie została zmodyfikowana.

## 2.5. Postman

Jest klient HTTP używany do tworzenia, testowania, udostępniania i dokumentowania interfejsów API wykorzystując graficzny interfejs użytkownika. Służy do testowania backendu, gdzie wpisujemy adres URL punktu końcowego, wysyła żądanie do serwera i odbiera odpowiedź z serwera. W tej pracy jest wykorzystywany do testowania punktów końcowych napisanych w .NET Core, które się łączą z aplikacją.



Rys. 4 Testowanie poprawności działania API

Źródło: Opracowanie własne

## 2.6. MSSQL Server

W danym przypadku, baza danych będzie się znajdować na lokalnym urządzeniu, gdzie będzie przechowywana informacja z aplikacji, taka jak: dane z tabel, zdjęcia itp. Wśród najpopularniejszych programów jako serwer może wystąpić Oracle, PostgreSQL, MySQL, MSSQL itp.

Rank			DBMS	Database Model	Score		
Jan 2022	Dec 2021	Jan 2021			Jan 2022	Dec 2021	Jan 2021
1.	1.	1.	Oracle	Relational, Multi-model	1266.89	-14.85	-56.05
2.	2.	2.	MySQL	Relational, Multi-model	1206.05	+0.01	-46.01
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	944.81	-9.21	-86.42
4.	4.	4.	PostgreSQL	Relational, Multi-model	606.56	-1.66	+54.33
5.	5.	5.	MongoDB	Document, Multi-model	488.57	+3.89	+31.34
6.	6.	7.	Redis	Key-value, Multi-model	177.98	+4.44	+22.97
7.	7.	6.	IBM Db2	Relational, Multi-model	164.20	-2.98	+7.03
8.	8.	8.	Elasticsearch	Search engine, Multi-model	160.75	+3.03	+9.50
9.	10.	11.	Microsoft Access	Relational	128.95	+2.96	+13.61
10.	9.	9.	SQLite	Relational	127.43	-1.25	+5.54

Rys. 5 Ranking najpopularniejszych baz danych

Źródło: W <https://db-engines.com/en/ranking>, z dnia 25.01.2022

Zgodnie ze statystyką sierpnia 2022-ego roku najpopularniejszą bazą jest Oracle 1980 roku powstania od firmy „Oracle Corporation”. Porównując bazy danych, na przykład MySQL i MSSQL (Microsoft SQL Server), otrzymujemy dane o różnicach jak i cechach wspólnych. Właścicielem MySQL, jak i bazy danych Oracle jest firma „Oracle Corporation”. Obydwa programy mogą być zainstalowane na platformie Windows i Linux, ale tylko MySQL może być zainstalowany też na OS X, Solaris i FreeBSD. MySQL wspiera wszystkie języki programowania co i MSSQL i jeszcze kilku dodatkowo. Ale skrypty na MySQL są tworzone wyłącznie na SQL, a MSSQL wspiera też Transact-SQL, .NET języki, R oraz Java.

## 2.7. C#

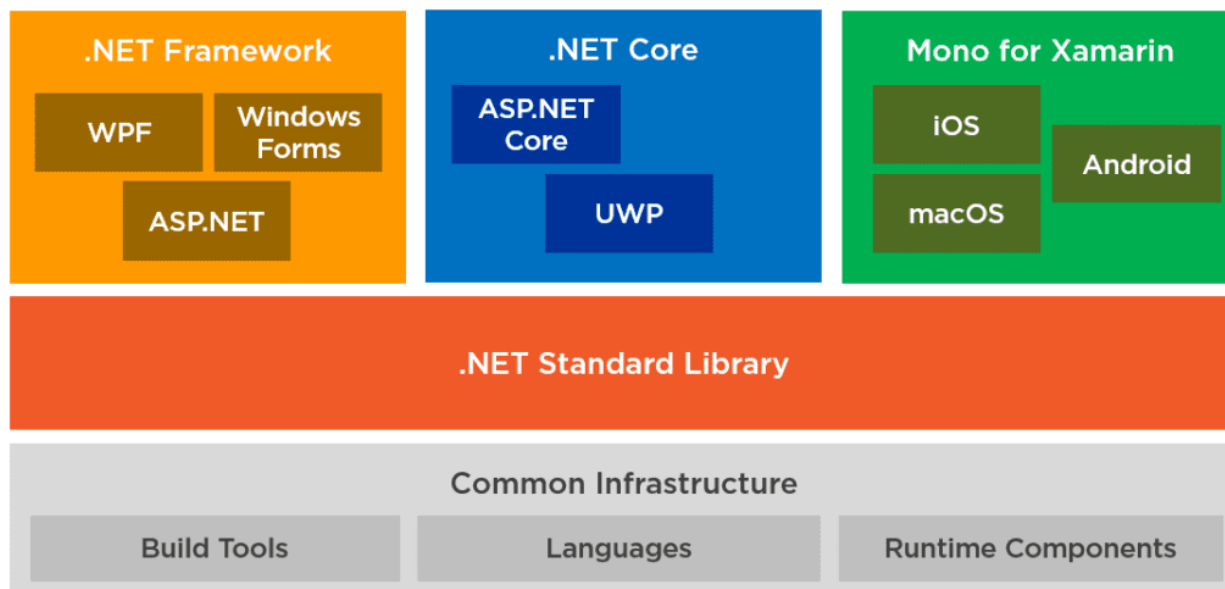
C# jest obiektowym językiem programowania opracowanym przez firmę Microsoft w 2000 roku w celu zaadaptowania najlepszych cech języków programowania Java i C++. Jest używany z wielu powodów, ale jego popularność polega na wykorzystaniu go do tworzenia serwisów backendowych, aplikacji desktopowych, tworzenia aplikacji webowych, game dewelopmentu i w mniejszym stopniu do machine learningu oraz tworzenia aplikacji mobilnych.

Środowisko uruchomieniowe CLR zawiera kompilator JIT z pośredniego języka IL do kodu platformy, na której zainstalowane jest środowisko CLR; Zawiera Garbage Collector i użyteczne funkcje cukru syntaktycznego, na przykład getery i setery, anonimowa inicjalizacja obiektów, lockowanie wątków asynchronicznych, i w wersji .NET SDK 6, C# 10 to globalne usingi oraz inne.

Kompilator C# tłumaczy kod źródłowy na kod IL; moduł kompatybilny z CLS może być używany w każdym języku programowania na platformie .NET, a w drugim kroku IL jest transformowany do kodu maszynowego. Jedno co można dodać, że JIT nie przetwarza od razu cały program, a po kolei wywoływane metody.

## 2.8. .NET Core

.NET Core na podstawie CLR platformy jest produktem open-source używanym do tworzenia oprogramowania na macOS, Linux oraz Windows. Zawiera w sobie JIT, Base Class Library, Entity Framework, WPF, VB.NET, F# i wiele innych narzędzi. Nowsze wersje są lżejsze i szybsze w porównaniu np. do .NET Framework lub .NET Core 2.1, który już nie jest wspierany. Ostatnia wersja to .NET 6 LTS, release której odbył się w 2021 i będzie rozwijany do 2024 natomiast w tej aplikacji jest używany .NET Core 3.1 LTS z datą supportu do końca 2022 roku. Nowsze wersje ciągle się rozbudowują, a wsparcie mikroserwisowej architektury i kontenerów robią ją wydajniejszą i perspektywiczną [Jeffrey Richter, 2012] oraz [Gaurav, Jeffrey, 2019].



Rys. 6 Ekosystem .NET

Źródło <https://stackify.com/net-ecosystem-demystified/>

## 2.9. Entity Framework Core

Nowsza, wydajniejsza krosplatformowa wersja ORM (*ang. Object–relational mapping*), która zapewnia dostęp do bazy danych przez modele, czyli klasy i obiekty kontekstu reprezentujące bazę danych. Kontekst służy do zapisywania rekordów i pisania zapytań do bazy danych za pomocą LINQ.

Alternatywą EF jest ADO.NET lub NHibernate. W pierwszym przypadku są wykorzystywane zwykłe zapytania SQL, dzięki czemu mamy większą kontrolę nad wymaganymi zadaniami. Ale minusem jest ogromna ilość kodu oraz potencjalne ataki w postaci SQL iniekcji choć można się zabezpieczyć wykorzystując parametryzowane procedury. Gdyż NHibernate zapewnia dostęp przez obiekt sesji i używa mapowania do tabel za pomocą XML, dzięki czemu do zarządzania jest potrzebne znacznie mniej kodu ale jest dość trudny w debugowaniu.

## 2.10. Angular

Jeden z najpopularniejszych frameworków do tworzenia aplikacji webowych. Polega na tworzeniu własnych komponentów z wykorzystaniem TypeScript oraz HTML. Do stylowania strony wykorzystałem bibliotekę bootstrap oraz SCSS. W porównaniu do nie mniej popularnej biblioteki React, jest trochę wolniejszy, ale za pomocą komponentów tworzenie, wspieranie oraz rozwijanie jest łatwiejsze. Została wykorzystana wersja CLI 12 złączeniem bibliotek RxJS, ngx-bootstrap oraz boostwatch.

## 2.11. Wzorce architektoniczne

Realizacja tego projektu jest oparta architektonicznie na wzorze MVC (*ang. Model View Controller*), który polega na oddzieleniu widoku, który jest reprezentowany przez modele tabel bazy danych, w których znajdują się dane walidowane i opracowane przez logikę w kontrolerach [Eric Evans, 2003]. Działanie wygląda w taki sposób, że kontrolery to endpointy z aplikacji backendowej, a widok to klient aplikacji Xamarin oraz Angular. Do komunikacji między warstwami umieściłem na localhost IIS host aplikację backendową z wykorzystaniem portu pod numerem 57792.



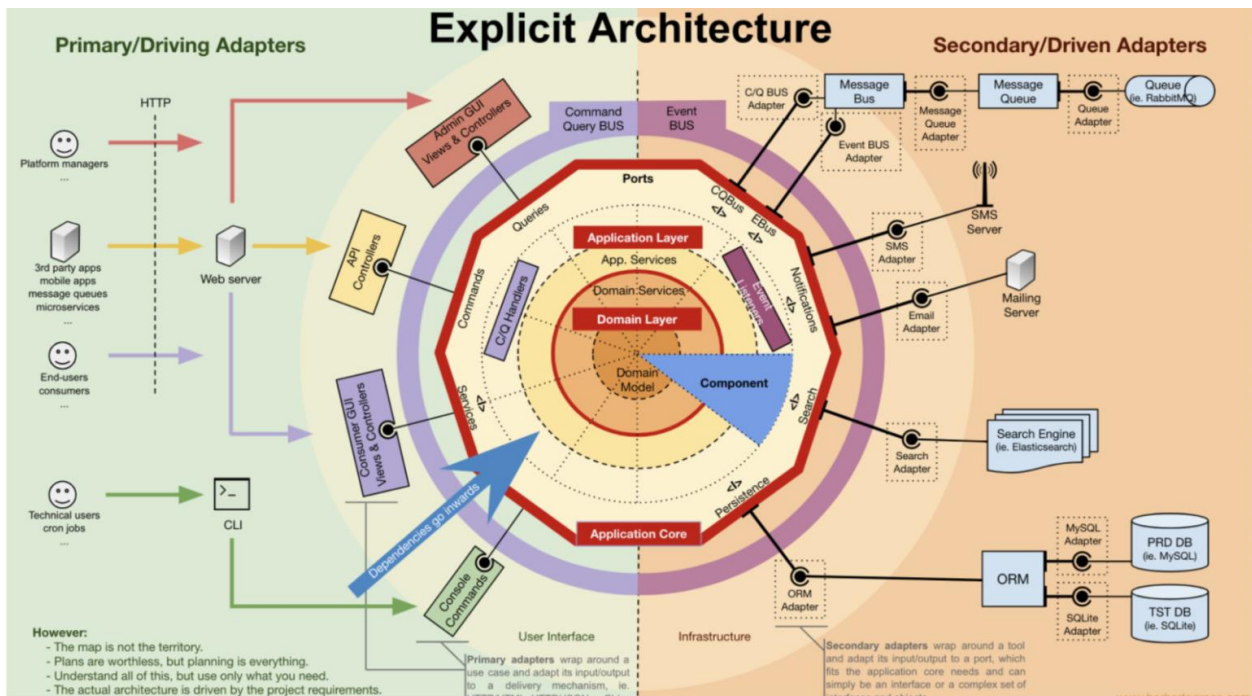
<b>(General)</b>	
Application Pool	XamarinApp
Bindings	http*:57792:
ID	3
Name	XamarinApp
Physical Path	marin_Food_App_API\FoodApi\bin\Release\netcoreapp3.1\publish...
Physical Path Credentials	
Physical Path Credentials Logon Type	ClearText
Preload Enabled	False

Rys. 7 Konfiguracja aplikacji na IIS  
Źródło: Opracowanie własne

Realizacja aplikacji backendowej opiera się na architekturze DDD [Eric Evans, 2003]. Domain-driven design - zbiór zasad i schematów mających na celu tworzenie optymalnych systemów obiektów. Proces rozwoju sprowadza się do tworzenia abstrakcji oprogramowania zwanych modelami domenowymi. Modele te obejmują logikę biznesową, która łączy rzeczywiste warunki aplikacji produktu z kodem.

Co nam to daje w końcu:

- prawie wszyscy członkowie zespołu mogą odczytać kod projektu;
  - zestawienie zadań staje się bardziej jednoznaczne;
  - błędy logiki biznesowej stają się łatwiejsze do znalezienia;
- Minusy:
- wymagane są wysokie kwalifikacje programistów, zwłaszcza na początku projektu;
  - nie wszyscy klienci są gotowi na takie koszty, DDD musi być poznany przez wszystkich uczestników procesu rozwoju.



Rys. 8 Schemat komunikacji między warstwami z podejściem architektonicznym DDD

Źródło: <https://medium.com/the-software-architecture-chronicles/ddc-hexagonal-onion-clean-cqrs-how-i-put-it-all-together-f2590c0aa7f6>, z dnia 10.01.2022

## 2.12. Wzorce projektowe

Wzorzec repozytorium – został zaimplementowany w projekcie do komunikacji między warstwami:

- Domain
- Infrastructure
- Application.

W Domain znajdują się wszystkie modele klas oraz interfejsy, które za pomocą Dependency Injection realizują dany wzorzec [WWW-6, 2022].

Infrastruktura przechowuje migracje bazodanowe przy wykorzystaniu podejścia „Code First”, realizacja interfejsów oraz połączenie z bazą danych za pomocą kontekstu. Połączenie do bazy danych jest definiowane w appsettings.json, gdzie też znajduje się logowanie danych oraz konfiguracja JWT tokenu w celu autoryzacji.



```
1 {
2   "Tokens": {
3     "Key": "ASPNETCORESECRETKEYFORAUTHENTICATIONANDAUTHORIZATION",
4     "Issuer": "localhost.com",
5     "AccessExpireSeconds": "86400"
6   },
7   "ConnectionStrings": {
8     "DefaultConnection": "Server=.;Database=FoodAppDb;Trusted_Connection=True"
9   },
10  "Logging": {
11    "LogLevel": {
12      "Default": "Information",
13      "Microsoft": "Warning",
14      "Microsoft.Hosting.Lifetime": "Information"
15    }
16  },
17  "AllowedHosts": "*"
18 }
19
```

Rys. 9 Konfiguracja tokenu autoryzacji oraz połączenia do bazy danych

Źródło: Opracowanie własne

Właśnie kontrolery, które są punktami końcowymi aplikacji, znajdują się w module Application. Działa to w następujący sposób:

1. Interfejsy definiują zachowywanie aplikacji, które są wstrzykiwane w Startup.cs w kontenerze IOC (ang. *Inversion of Control*), by można było ich wykorzystać w kontrolerach.



```
68     });
69     services.AddDbContext<FoodDbContext>(option => option.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
70     services.AddSwaggerGen();
71
72     services.AddScoped<IUnitOfWork, UnitOfWork>();
73     services.AddScoped<ICategoriesRepository, CategoriesRepository>();
74     services.AddScoped<IAccountsRepository, AccountsRepository>();
75     services.AddScoped<IOrdersRepository, OrdersRepository>();
76     services.AddScoped<IProductsRepository, ProductsRepository>();
77     services.AddScoped<IShoppingCartItemsRepository, ShoppingCartItemsRepository>();
78 }
```

Rys. 10 Wstrzykiwanie dostępu do serwisów za pomocą interfejsów

Źródło: Opracowanie własne

2. Klasy realizują metody interfejsów, które później będą wykorzystane w kontrolerze. W aplikacji też zdefiniowana klasa generyczna, która prowadzi dostęp do podstawowych metod bez konieczności ich implementowania od zera.

Rys. 11 Klasa generyczna która zapewnia dostęp do podstawowych metod CRUD  
Źródło: Opracowanie własne

Takie podejście pozwala skrócić napisanie kodu, ponieważ:

- Nie tworzymy dla każdej klasy realizacje podstawowych metod;
  - Zachowanie aplikacji oraz różnych klas będzie podobne między sobą, w taki sposób kod jest bardziej czytelny;
  - Symbioza takich poleceń pozwala na łatwiejszą kontrolę kodu i wyłapanie bugów.
3. Kontrolery wykorzystują z serwisów przeznaczonych dla tych kontrolerów w związku z czym, logika jest ukryta, a zależności ładują się w middlewareach, co robi kod bardziej zabezpieczonym [Gaurav, Jeffrey, 2019].



### 3. Część praktyczna

W tym rozdziale zostaną zdefiniowane wymagania biznesowe oraz wymagania funkcjonalne i нефункционалне. Będą przeanalizowane najważniejsze przypadki użycia oraz procesy występujące w systemie. Zostanie też opisany projekt danego systemu.

#### 3.1. Analiza wymagań

Analiza wymagań to proces, który prowadzi do decyzji, co powinien oferować produkt końcowy. Od tego, w jaki sposób ona została przeprowadzona zależy cały projekt. Wymagania funkcjonalne oraz нефункционалне zostały przygotowane na podstawie analizy aplikacji ze świata rzeczywistego, w oparciu o dobre praktyki programowania oraz wskazania z tematu bezpieczeństwa.

Korzyści, które osiąga dana aplikacja skupiają się jak na użytkowniku dla którego będzie skrócony czas złożenia zamówienia tak i na pracowniku, który będzie wykonywał mniej pracy, a w ideale restauracja będzie pracować całkiem automatycznie bez konieczności zaangażowania ludzi do pracy. Najważniejsze funkcje aplikacji to funkcje, które bezpośrednio dotyczą interakcji z instytucją, czyli wszystkie możliwe manipulacje z produktami i koszykiem zakupów, mniej priorytetowe to czat z restauracją oraz najmniej ważne to działania z punktami zdobytymi przy zakupach lub innych promocjach.

Rozszerzone możliwości użytkowników z uprawnieniami administratorów służą do dodawania i usuwania elementów menu oraz przeglądu dziennych zamówień z możliwością wyeksportowania ich do JSON. Takie narzędzie może być pomocne przy tworzeniu codziennych raportów i może zautomatyzować procesy biznesowe instytucji, na przykład z integracją systemu ERP.

#### 3.2. Specyfikacja wymagań

Wymagania funkcjonalne opisują dostępne usługi, które oferuje aplikacja oraz jakie konsekwencje powodują różne działania. Jest to bardzo ważna część projektu informatycznego, ponieważ realizacja opiera się o wymienione wymagania funkcjonalne w tym rozdziale. W wyniku analizy została wydedukowana następująca lista wymagań.

Tab. 1. Wymagania funkcjonalne

ID	Opis wymagania	Priorytet
1	Rejestracja użytkowników	Wysoki
2	Rezerwacja miejsca	Wysoki
3	Złożenie zamówienia	Wysoki
4	Czat z restauracją	Średni
5	Eksportowanie wszystkich dziennych zamówień w postaci pliku .JSON	Wysoki
6	Zmiana uprawnień użytkowników	Średni
7	Ocena zamówienia	Mały
8	Przegląd historii zamówień użytkownika	Mały
9	Sortowanie oraz filtrowanie w tym po kaloriach	Mały

Źródło: Opracowanie własne

Wymagania нефункционалне opisują wymagania wydajnościowe, związane bezpieczeństwem aplikacji oraz częścią wizualną.

Tab. 2. Wymagania нефункционалне

ID	Opis wymagania	Priorytet
1	Konfigurowany token autoryzacji	Wysoki
2	Haszowanie wrażliwych danych	Wysoki
3	Ochrona przed XSS oraz SQL iniekcjami	Wysoki
4	System powinien być skalowany	Średni

5	Aplikacja powinna być dostępna dla użytkowników przeglądarek Android, iOS Google Chrome oraz Mozilla Firefox.	Średni
---	---	--------

Źródło: Opracowanie własne

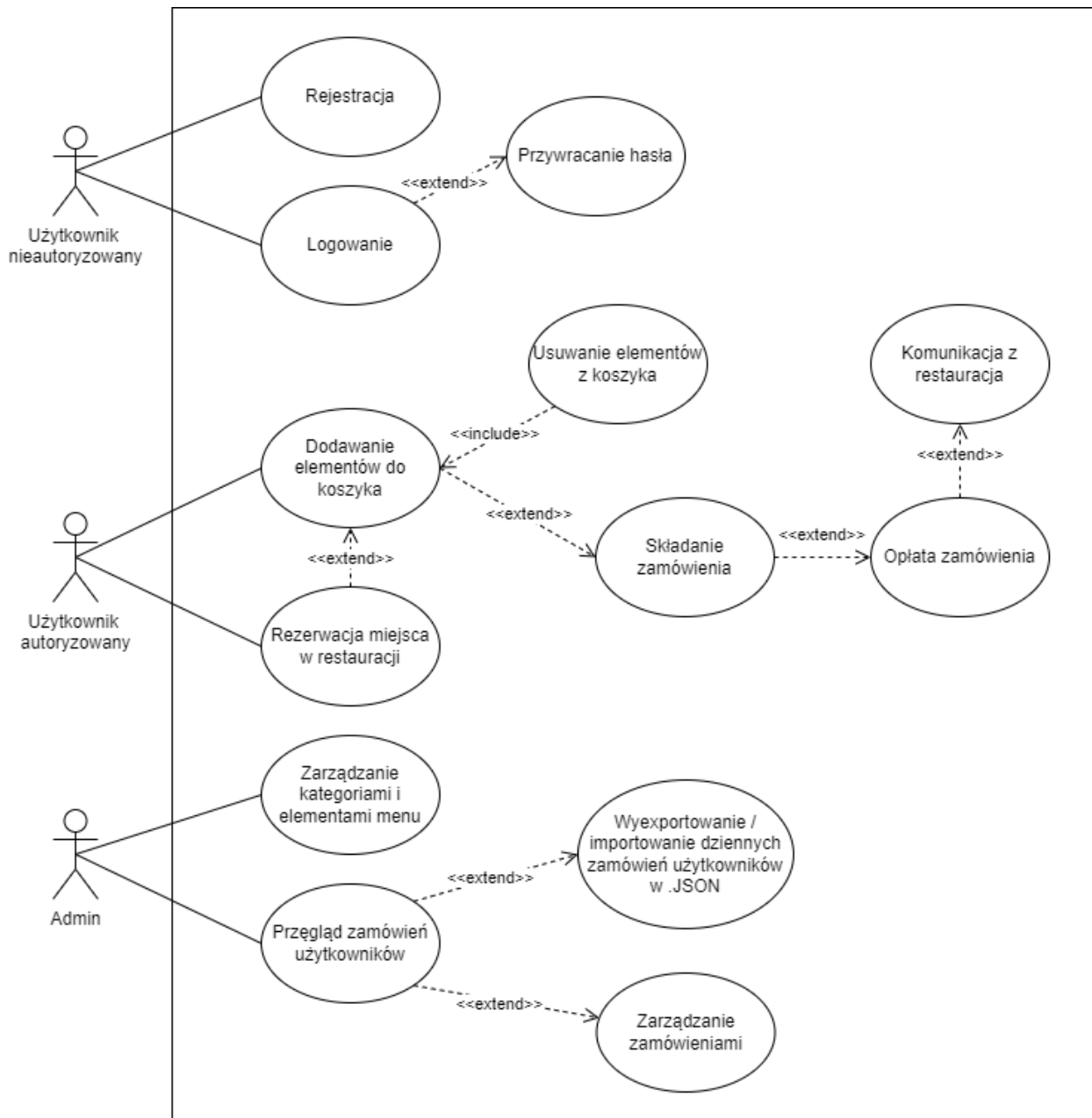
### 3.3. Diagram przypadków użycia

W następnej kolejności projektowania aplikacji został opracowany diagram przypadków użycia, który jest bardzo ważnym elementem procesu projektowania aplikacji, ponieważ wizualizuje wszystkie przypadki, które muszą być uwzględnione podczas kolejnych etapów projektowania oraz podczas procesu implementacji rozwiązania.

Diagram przedstawia możliwe interakcje końcowego użytkownika z aplikacją. W tym przypadku zostały zdefiniowane trzy aktorzy którzy reprezentują poruszania użytkownika zalogowanego, administratora oraz gościa.

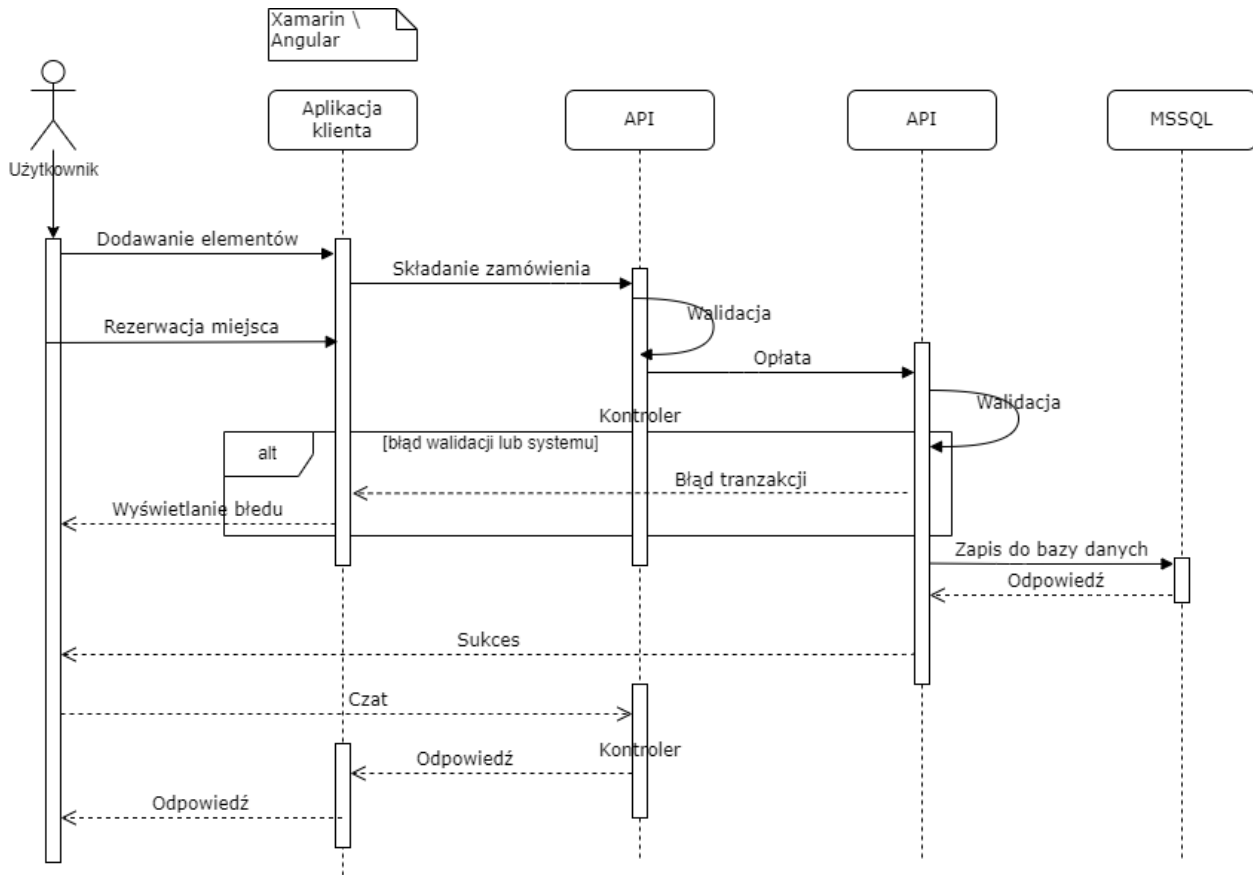
Użytkownicy mają dostęp do funkcji aplikacji jak z wersji mobilnej tak i webowej ale administratorskie narzędzia dostępne tylko z przeglądarki. Przed lub po założeniu konta użytkownik może działać z menu restauracji ale złożyć zamówienie lub zarezerwować miejsce można dopiero po zalogowaniu się. Po dokonaniu transakcji jest możliwość zostawiania opinii lub komunikacji z tym miejscem publicznym.

Administrator zarządza jednostkami bazodanowymi oraz zamówieniami. Może wyliczyć sumę, ilość lub usunąć niezrealizowane zamówienia ale najważniejszym jest możliwość wyeksportowania do pliku .JSON by inny program mógł odczytać te dane i wygenerować raport lub inne działania z systemem ERP (*ang. Enterprise resource planning*). Poniżej przedstawiony diagram przypadków użycia, który pokazuje interakcję użytkownika z systemem.



Rys. 14 Diagram przypadków użycia  
Źródło: opracowanie własne

Poniżej jest pokazany diagram sekwencji na podstawie analizy wymagań projektowych oraz diagramu przypadków użycia, który reprezentuje komunikację między systemami. Pokazano proces interakcji użytkownika z aplikacją oraz reakcję systemu na jego działania.



Rys. 15 Diagram sekwencji systemu  
Źródło: opracowanie własne

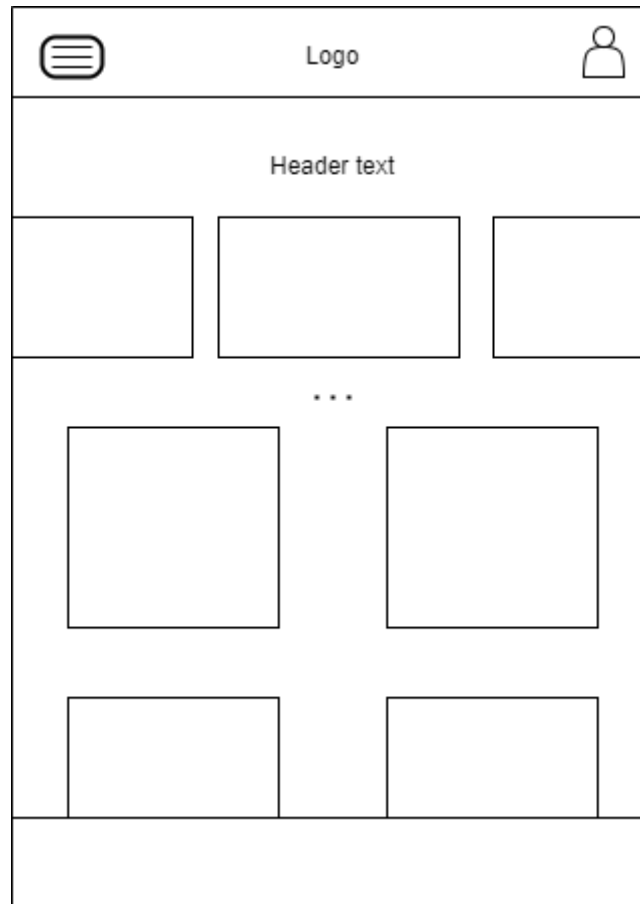
### 3.4. Prototypy interfejsu

Interfejs w większym stopniu jest związany z designem. Właśnie te dwa pojęcia są definiowane za pomocą terminologii UI / UX. Ładny interfejs użytkownika nie oznacza intuitywności. Tym się zajmuje UX designer. A intuitywność w aplikacjach oznacza jak szybko użytkownik zrozumie jego działanie.

W projektowaniu interfejsu dość ważne jest zachowywanie logiczne, warto też korzystać z odpowiednich ikon i kolorów. Tak w tym projekcie czarny kolor to obramowanie i treść, żółty to ostrzeżenia oraz pola do wypełnienia, niebieski oraz zielony zwykle neutralne kolory, czerwony służy do komunikowania o krytycznych sytuacjach, a pomarańczowy to główny kolor projektu.

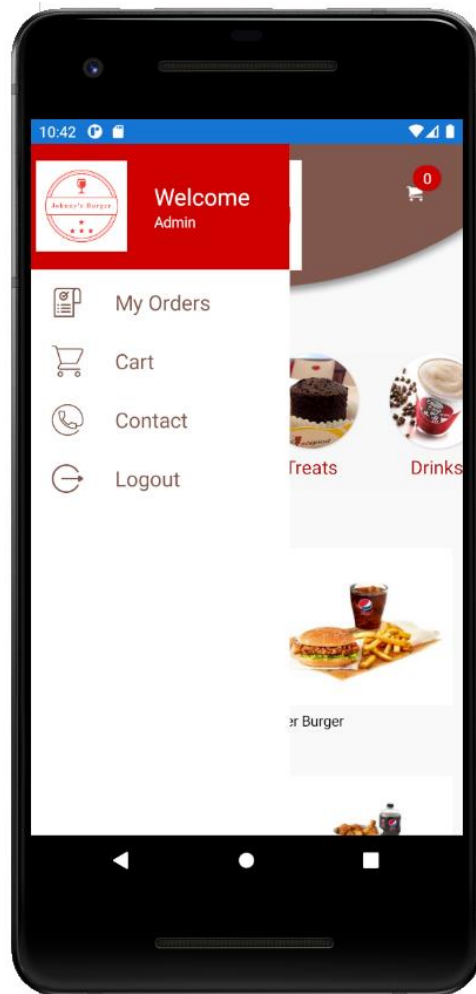
Interfejs w aplikacji mobilnej był projektowany na podstawie zwykłych XML. Takie podejście jest dość podobne do projektowania strony z wykorzystaniem HTML z jednym wyjątkiem że interfejs jest oparty o Grid layout, ponieważ responsywność aplikacji mobilnych bardziej statyczna.





Rys. 16 Makieta interfejsu aplikacji mobilnej  
Źródło: opracowanie własne

Opierając się na powyższym prototypie była stworzona aplikacja mobilna z lebelkami do rysunków, czyli w karuzeli znajdują się kategorie menu, a niżej lista popularnych pojedynczych produktów.



Rys. 17 Prototyp interfejsu użytkownika  
Źródło: Opracowanie własne

Został dodany też banner by zrobić aplikację przyjemniejszą wizualnie. Pod kątem tego, że aplikacja mobilna jest podobna do aplikacji webowej, prototyp interfejsu wygląda dość podobnie, dzięki czemu zmiana sposobu korzystania nie będzie zaskoczeniem. Na Rys. 18 jest przedstawiony prototyp interfejsu aplikacji webowej.

Sort

Alphabetical

Categories

All

Category 1

Category 2

Category 3 :on hover

Category 4

Category 5

Category 6

Showing 1 - 6 of 18 Results

Search

Search

Reset

Product 1

PRODUCT 1

\$180.00

Product 2

PRODUCT 2

\$150.00

Product 3

PRODUCT 3

\$200.00

Product 4

PRODUCT 4

\$18.00

Product 5

PRODUCT 5

\$10.00

Product 6

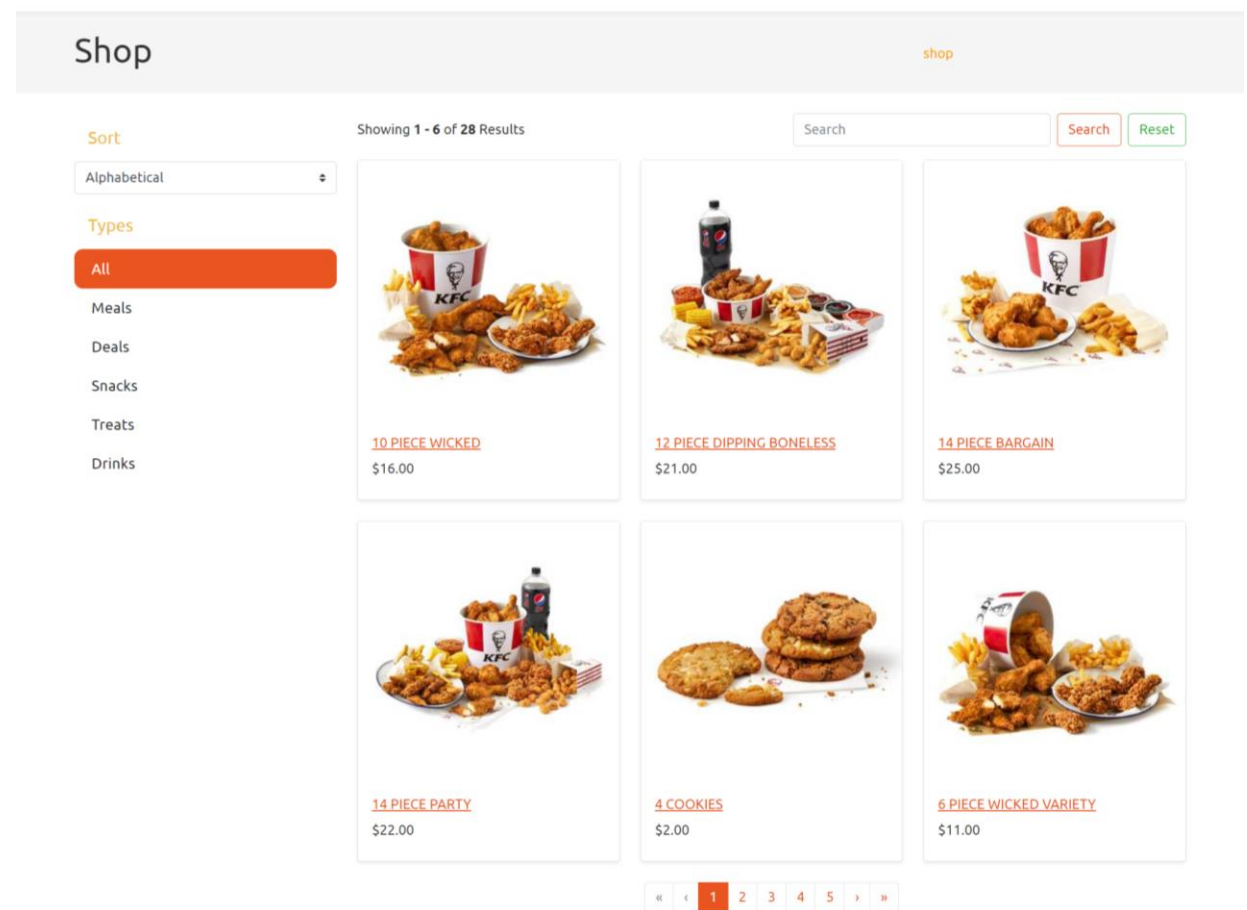
PRODUCT 6

\$180.00

« ‹ 1 2 3 › »

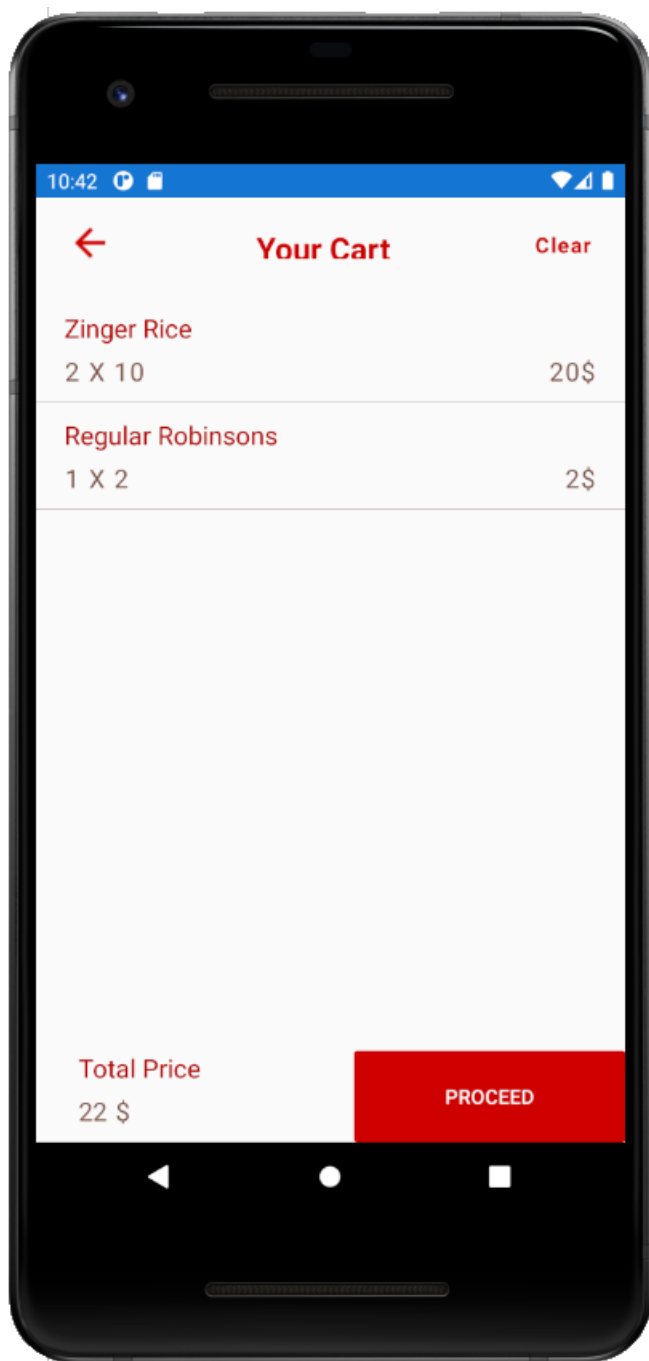
Rys. 18 Prototyp strony internetowej restauracji  
Źródło: Opracowanie własne

W oparciu o prototyp interfejsu była stworzona aplikacja webowa z główną stroną jak na Rys.19.



Rys. 19 Główna strona aplikacji  
Źródło: opracowanie własne

Przegląd zamówienia nieco się różni między aplikacjami, dlatego w aplikacji mobilnej nie ma sensu umieszczać zdjęcia produktów, ponieważ nie będzie ich dobrze widać by rozwiązać ten problem jest dodane przekierowanie strony po kliknięciu na element żeby upewnić się w wyborze.



Rys. 20 Przegląd zamówienia w aplikacji mobilnej  
Źródło: Opracowanie własne

Checkout

checkout

ADDRESS		DELIVERY	REVIEW	PAYMENT	ORDER SUMMARY
<b>PRODUCT</b>			<b>PRICE</b>	<b>QUANTITY</b>	<b>TOTAL</b>
<u>Hot Chocolate</u> Type: Drinks			\$3.00	1	\$3.00
<u>Regular 7UP</u> Type: Drinks			\$1.00	3	\$3.00
<u>10 Piece Wicked</u> Type: Snacks			\$16.00	1	\$16.00
<u>12 Piece Dipping Boneless</u> Type: Deals			\$21.00	1	\$21.00
<a href="#">&lt; Back to Delivery</a>			<a href="#">Go to Payment &gt;</a>		
<i>Shipping costs will be added depending on choices made during checkout</i>					
Order subtotal					\$43.00
Shipping and handling					\$5.00
Total					\$48.00

Rys. 21 Potwierdzenie zamówienia w aplikacji webowej  
Źródło: Opracowanie własne

Po opłacie zamówienia możemy przejść do komunikacji z restauracją by sprawdzić stan zamówienia lub żeby dołożyć coś jeszcze itp.

HOME SHOP RESERVATIONS

Welcome Bob

Chat

features / chat

Chat in case of order no 4

Andrew@test.com  
Dzień dobry  
May 3, 2022, 3:41:26 PM

Dzień dobry, w czym mogę pomóc?  
May 3, 2022, 3:41:40 PM

Andrew@test.com  
Piszę w sprawie zamówienia #4, chciałbym jeszcze dodać frytki do zamówienia  
May 3, 2022, 3:42:30 PM

Dobrze, wszystko jasne, zaraz dodam do zamówienia  
May 3, 2022, 3:43:11 PM

Andrew@test.com  
❤️  
May 3, 2022, 3:43:35 PM

😊  
May 3, 2022, 3:43:44 PM

Type a message

HOME SHOP RESERVATIONS

Welcome Andrew

Chat

features / chat

Chat in case of order no 4

Dzień dobry  
May 3, 2022, 3:41:26 PM

bob@test.com  
Dzień dobry, w czym mogę pomóc?  
May 3, 2022, 3:41:40 PM

Piszę w sprawie zamówienia #4, chciałbym jeszcze dodać frytki do zamówienia  
May 3, 2022, 3:42:30 PM

Dobrze, wszystko jasne, zaraz dodam do zamówienia  
May 3, 2022, 3:43:11 PM

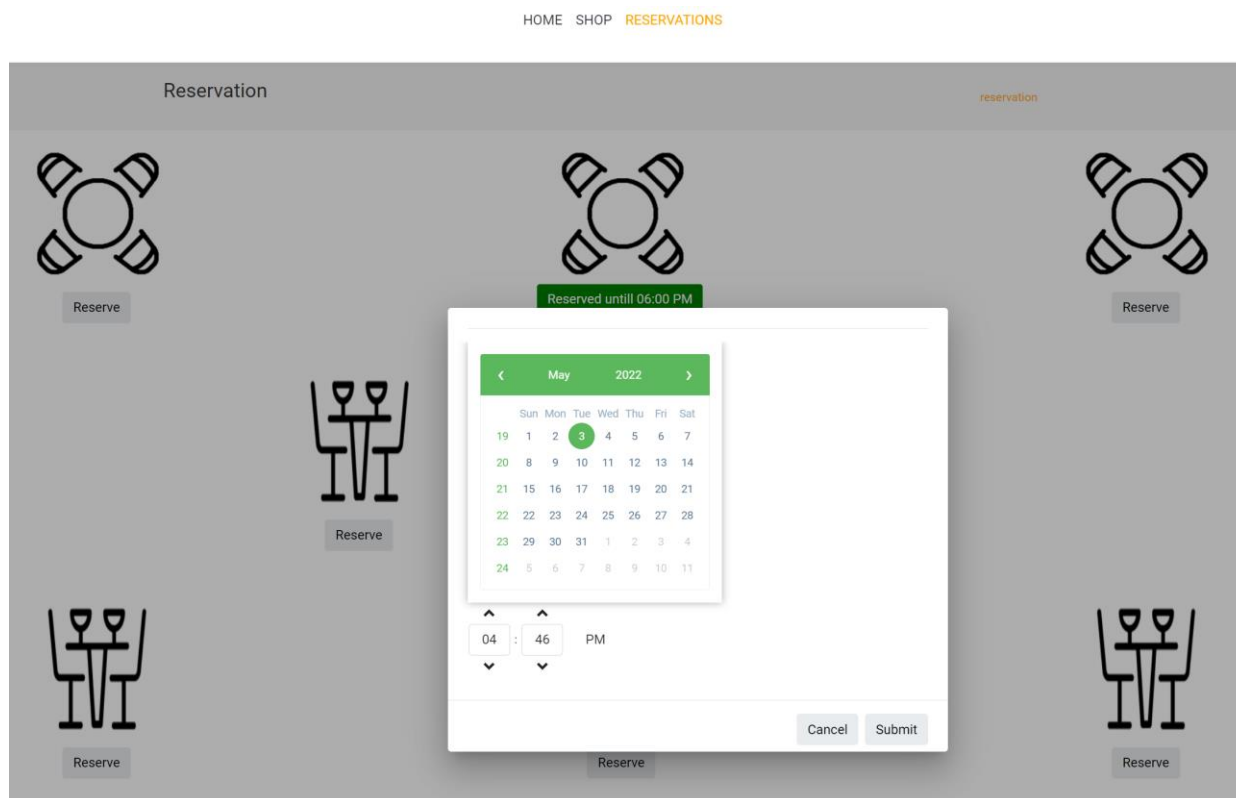
bob@test.com  
❤️  
May 3, 2022, 3:43:35 PM

😊  
May 3, 2022, 3:43:44 PM

Type a message

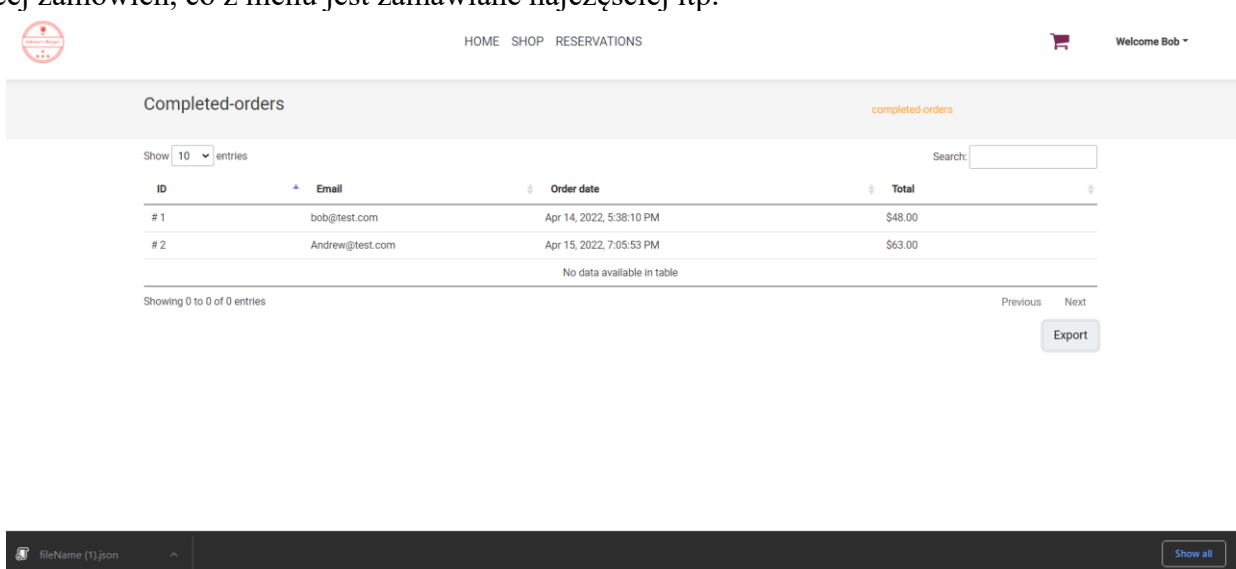
Rys. 22 Czat z restauracją  
Źródło: Opracowanie własne

Inną dostępną opcją jest rezerwacji miejsca. W aplikacji wygląda następująco: wybieramy stolik, po kliknięciu na który wyświetla się okienko z datą oraz godziną do rezerwacji. Czas rezerwacji ustawia się automatycznie, a po kliknięciu inny użytkownik będzie widział, że niektóre miejsca już zostały zarezerwowane.



Rys. 23 Prototyp interfejsu rezerwacji miejsca  
Źródło: Opracowanie własne

Z administratorskich narzędzi mamy możliwość wybrania dnia do eksportowania danych zamówienia. Może być to przydatne w różnych przypadkach analizy danych, na przykład: w jakim dniu zrobiono najwięcej zamówień, co z menu jest zamawiane najczęściej itp.



Rys. 24 Przegląd danych zamówień  
Źródło: Opracowanie własne

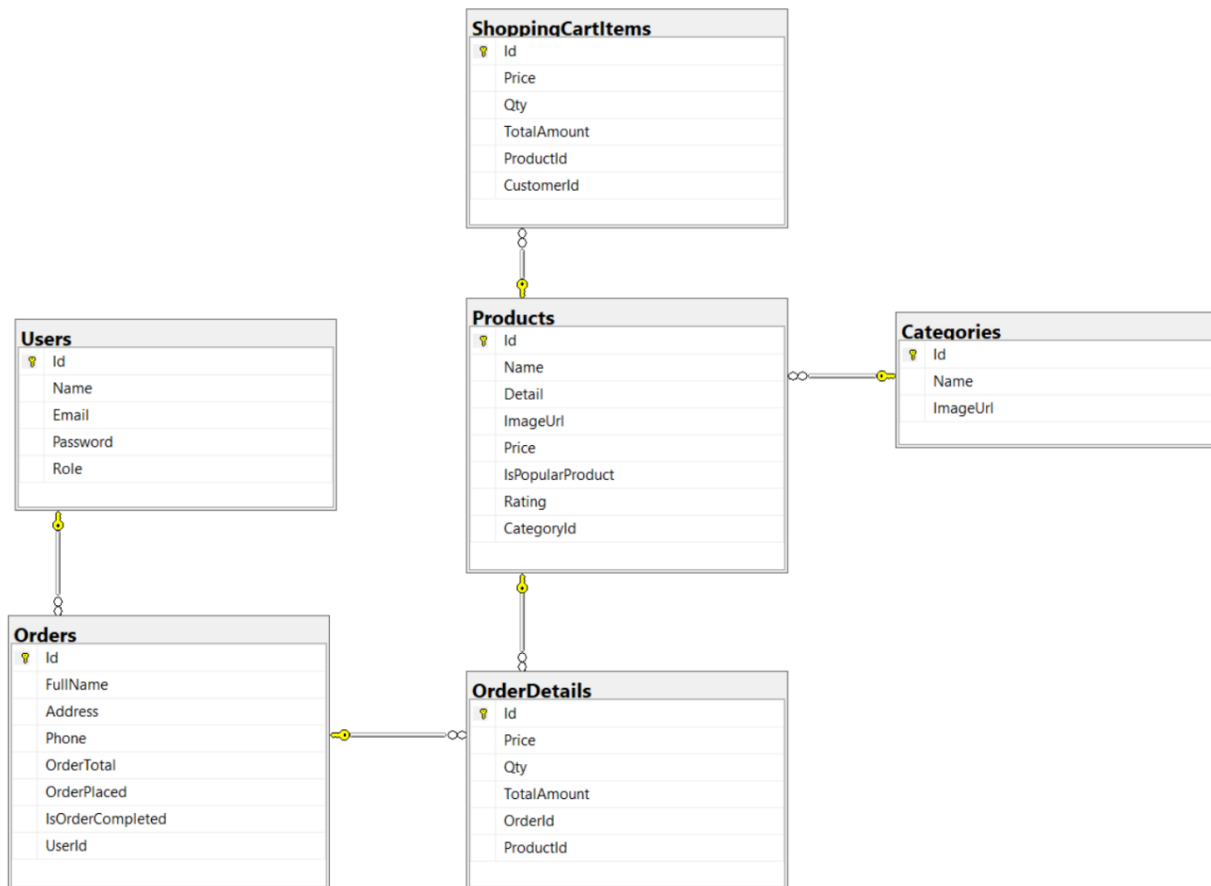
### 3.5. Implementacja

W tym rozdziale zostanie opisany proces implementacji aplikacji. Implementacja każdego programu jest osobnym procesem, ponieważ one są powiązane między sobą w małym stopniu, dlatego że każde rozwiązanie to osobna aplikacja. Dzięki takiemu podejściu mamy bardziej wydajny oraz łatwo rozszerzany system.

Przy przeciążeniu jednej aplikacji pozostałe będą działać poprawnie i użytkownik będzie mógł dokonać zaplanowanych zadań.

### 3.5.1. Projekt bazy danych

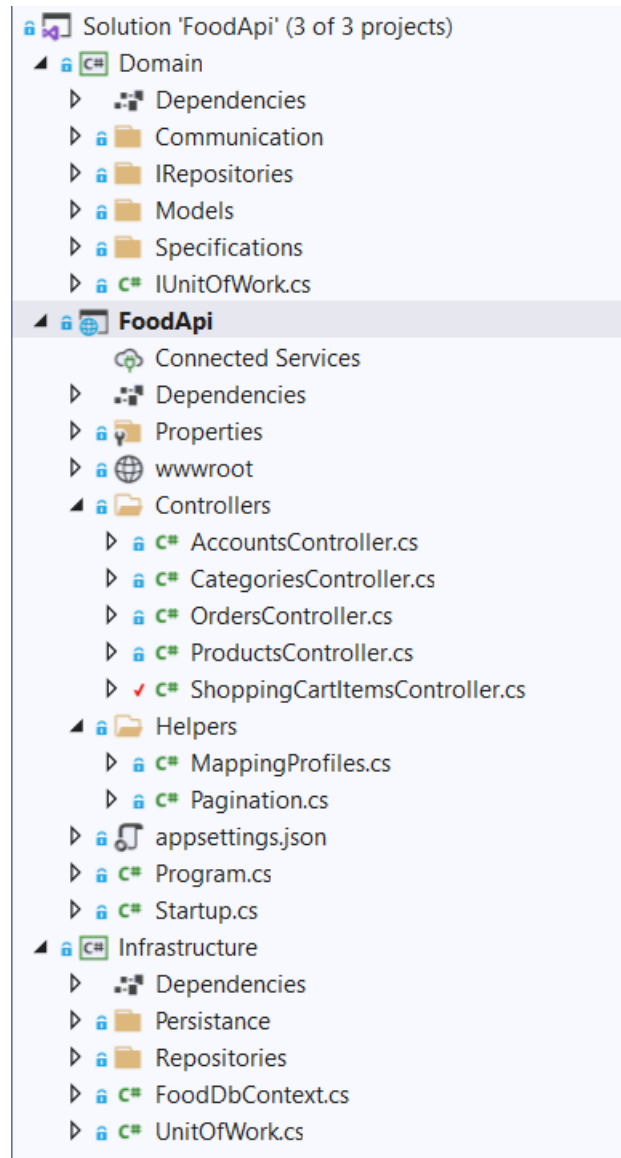
Pierwszym etapem projektowania systemu jest projektowanie struktury bazy danych. Rozwiązanie polega na wykorzystaniu podejścia „DB-First”. W takim razie są generowane modele, czyli klasy na serwerze na podstawie tabel w bazie, w tym przypadku MSSQL. Podczas projektowania bazy danych zostały uwzględnione wymagania zarówno funkcjonalne jak i нефункционалне. Zaprojektowano 6 tablic bazy z odpowiednimi relacjami pomiędzy nimi o strukturze przedstawionej na Rys. 15.



Rys. 25 Diagram relacji tabel  
Źródło: opracowanie własne

### 3.5.2. Implementacja aplikacji

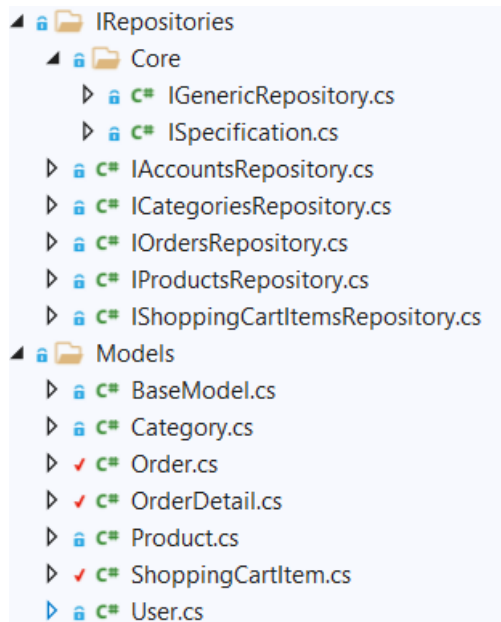
Pierwsza część implementacji, to implementacja API części serwerowej w technologii ASP.NET Core 3.1 i C#, REST API, tokenów JWT, Postmana, MSSQL i EntityFrameworka. Architektura projektu jest oparta o wzorzec architektoniczny DDD z wykorzystaniem wzorca projektowego pod nazwą „Repozytorium”.



Rys. 26 Struktura plików rozwiązania REST API  
Źródło: opracowanie własne

Relacja między projektami jest następująca: Domain to część niezależna, gdzie są przechowywane modele i interfejsy, które są używane w kontrolerach za pomocą DI (*ang. Dependency Injection*).





Rys. 27 Serwisy i modele aplikacji w Domain  
Źródło: opracowanie własne

Każdy interfejs jest zaimplementowany w warstwie Infrastructure oraz wstrzyknięty w klasie oprogramowania pośredniczego, gdzie też jest definiowana baza danych oraz token JWT.



Rys. 28 Konfiguracja aplikacji backendowej  
Źródło: opracowanie własne

Warstwa Infrastructure jest zależna od Domain, ponieważ wykorzystuje jego modele do realizacji interfejsów, migracji bazodanowych oraz kontekstu bazy danych.

I na samym końcu, gdzie cała ta informacja jest opracowana oraz przetwarzana - to warstwa „Application”, która jest zależna od Domain oraz Infrastructure, ponieważ wykorzystuje DTO (ang. *Data Transfer*

*Object*) do wysyłania i przyjmowania danych z Domain i repozytoria do przetwarzania informacji z Infrastruktury. Właśnie jest to zaimplementowane w kontrolerach, które są endpointami do klienckich aplikacji.

Implementacja aplikacji mobilnej polega na komunikacji między warstwami Android z Application oraz iOS z Application, gdzie Application to spójna biblioteka, przeznaczeniem której jest komunikacja z API oraz definiowanie spójnych widoków. Właśnie w poszczególnych przesłonach implementują się rozwiązania, które na innych platformach są ograniczone ale XMLe widoków są takie same na wszystkich platformach.

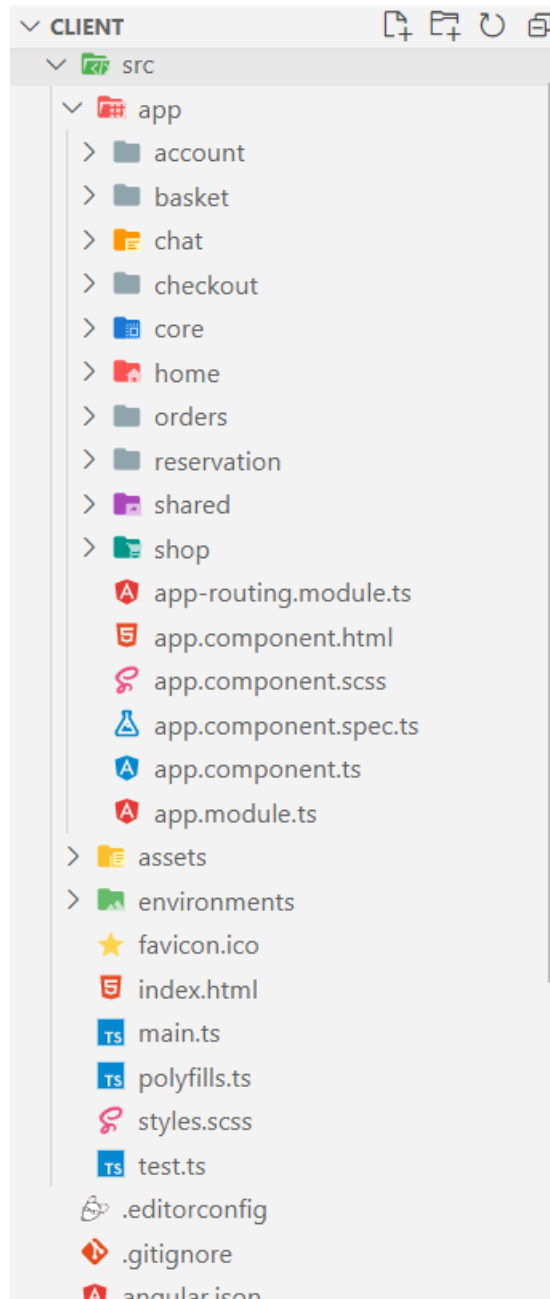
```
        BackgroundColor="#F7F5F4"
        NavigationPage.HasNavigationBar="False">
<ContentPage.Content>
    <StackLayout Spacing="20"
        Margin="25">
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="1*" /><ColumnDefinition Width="8*" /><ColumnDefinition Width="1*" />
            </Grid.ColumnDefinitions>
            <Image Source="backarrow.png" HeightRequest="25" HorizontalOptions="Center" VerticalOptions="Center">
                <Image.GestureRecognizers>
                    <TapGestureRecognizer x:Name="TapBackArrow" Tapped="TapBackArrow_Tapped" />
                </Image.GestureRecognizers>
            </Image>
            <Label Grid.Column="1" TextColor="#CE0B06" Text="LOGIN" FontSize="Large" FontAttributes="Bold" VerticalOptions="Center"
                HorizontalTextAlignment="Center" HorizontalOptions="Center" />
        </Grid>
        <Frame HasShadow="True" Padding="25" Margin="0,80,0,50">
            <StackLayout Margin="0,60,0,60">
                <Label Text="Sign In" FontAttributes="Bold" FontSize="Title" HorizontalTextAlignment="Center" TextColor="#CE0B06" />
                <Entry Placeholder="Email" PlaceholderColor="#80574D" TextColor="#80574D" x:Name="EntEmail" Keyboard="Email" />
                <Entry Placeholder="Password" PlaceholderColor="#80574D" TextColor="#80574D" IsPassword="True" x:Name="EntPassword" />
            </StackLayout>
        </Frame>
        <Button Text="Login"
            BackgroundColor="#CE0B06"
            TextColor="White"
            x:Name="BtnLogin"
            Clicked="BtnLogin_Clicked"/>
    </StackLayout>
</ContentPage.Content>
```

Rys. 29 Projektowanie widoku na przykładzie logowania

Źródło: opracowanie własne

Widoki są reprezentowane w postaci XML, a bindowanie danych polega na definiowaniu atrybutów z odpowiedniej klasy. Właśnie w taki sposób dane są przekazywane z programu do widoku oraz z widoku do programu.

Aplikacja webowa ma architekturę opartą o komponenty. W folderze „src” znajduje się cała aplikacja webowa wraz ze zdjęciami oraz modułami do instalacji. W środku mamy komponent logowania się pod nazwą „account”, dodawanie do koszyka który się nazywa „basket”, opłata oraz eksportowanie danych administratorskich to „checkout”, a w „core” mamy header oraz footer, ponieważ są wykorzystane na każdej stronie. W „home” umieściłem główną stronę, a w „shared” się znajdują inne elementy aplikacji.



Rys. 30 Architektura aplikacji frontendowej  
Źródło: opracowanie własne

Aplikacja oparta o Angular CLI 12, do manipulacji danymi, które przychodzą z API wykorzystuje zapytania http wraz z biblioteką RxJS. Opracowane dane z serwisów są przekazywane do komponentów konkretnego modułu.

Komponenty korzystają z serwisów i mają swoje przeznaczenie dla osobnych części wizualizacji danych. By uruchomić aplikację należy zainstalować biblioteki za pomocą node.js i menedżera pakietów „npm” lub „yarn” zatem wykonać polecenie w terminalu „ng serve” i aplikacja się uruchomi na „http://localhost:4200”, który się łączy do API na „https://localhost:5001”. Dzięki temu że jest zainstalowany certyfikat SSL, połączenie jest bezpieczniejsze, ponieważ odbywa się na podstawie protokołu HTTPS.

### 3.6. Opis działania aplikacji

Aplikacja mobilna oraz webowa wykorzystują endpointy z serwera IIS do pobrania, edytowania lub dodawania nowych danych. Tak przez aplikację mobilną możemy przeprowadzić wszystkie działania z menu restauracji, czyli przeglądnąć listę, dodać lub usunąć do zamówienie oraz menu kontaktowe z restauracją.

W aplikacji webowej oprócz wyżej wymienionych funkcjonalności, dla administratorów też jest możliwość zarządzania menu oraz wyeksportowania danych zamówień z wybranego dnia do pliku JSON.

W wyniku implementacji danego systemu zostały wytworzone trzy główne moduły:

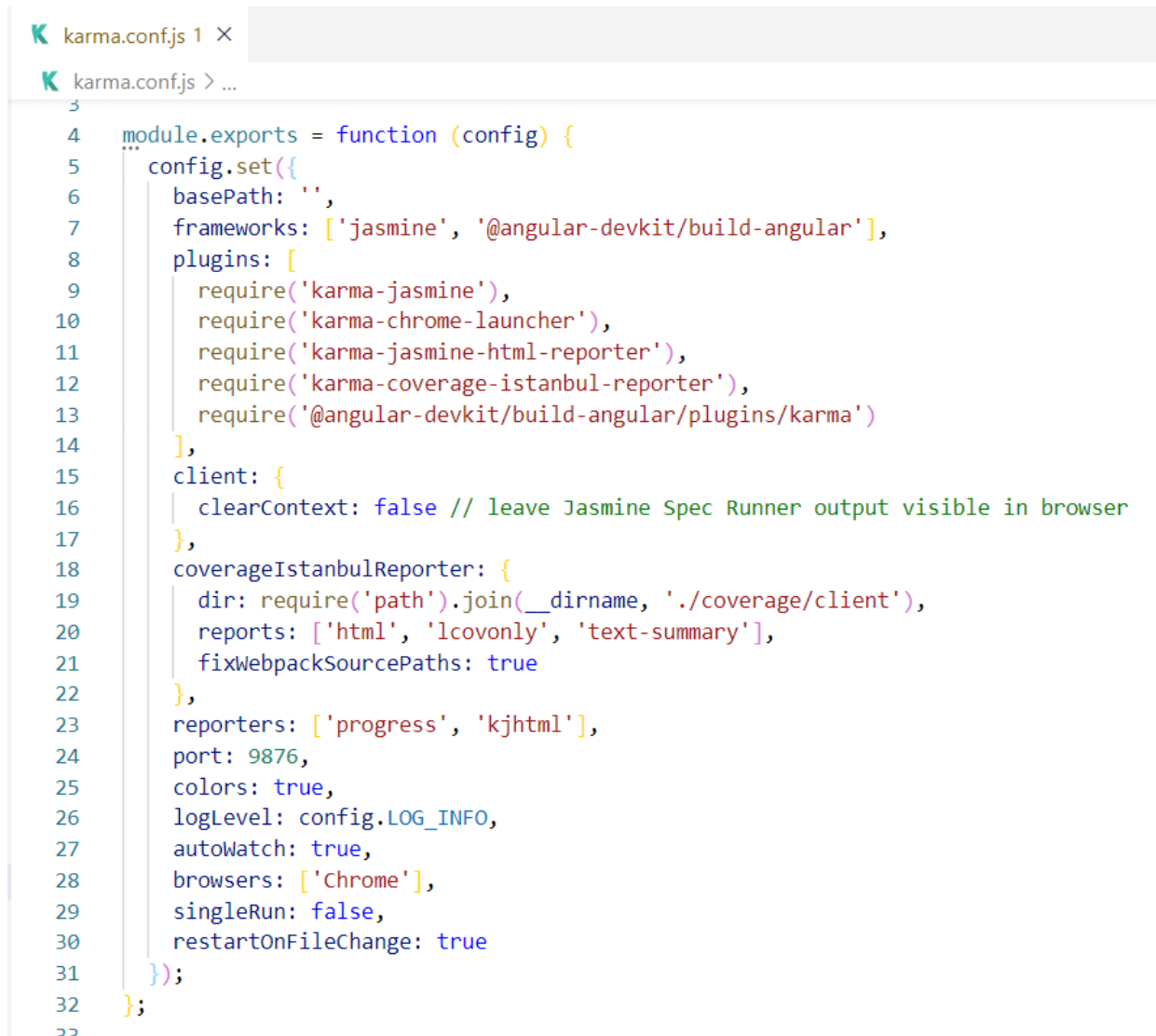
1. Aplikacja mobilna na podstawie Xamairn, który umożliwia korzystanie z podstawowych funkcjonalności dla użytkownika.
2. Aplikacja webowa w postaci SPA (*ang. Single page application*), który oferuje korzystanie z wszystkie podstawowych funkcjonalności oraz narzędzia do zarządzania menu.
3. Backend w postaci REST API, który został umieszczony na IIS hostcie.

Z aplikacji webowej po opłacie za pomocą Stripe jest dostępny przegląd oczekujących zamówień oraz czat z restauracją.

Po zakończeniu procesu implementacji projekt był gotowy do przetestowania.

### 3.7. Testy (ewaluacja)

Po zakończeniu implementacji były przeprowadzone testy manualne oraz automatyczne. Do testowania serwisów aplikacji były napisane testy jednostkowe oraz integracyjne. Napisane testy zostały stworzone dla osobnych jednostek kontrolerów oraz głównego modułu aplikacji. Do testowania komponentów aplikacji Frontendowej został wykorzystany framework „Jasmine” oraz „Karma”.



```
3
4 module.exports = function (config) {
5   config.set({
6     basePath: '',
7     frameworks: ['jasmine', '@angular-devkit/build-angular'],
8     plugins: [
9       require('karma-jasmine'),
10      require('karma-chrome-launcher'),
11      require('karma-jasmine-html-reporter'),
12      require('karma-coverage-istanbul-reporter'),
13      require('@angular-devkit/build-angular/plugins/karma')
14    ],
15    client: {
16      clearContext: false // leave Jasmine Spec Runner output visible in browser
17    },
18    coverageIstanbulReporter: {
19      dir: require('path').join(__dirname, './coverage/client'),
20      reports: ['html', 'lcovonly', 'text-summary'],
21      fixWebpackSourcePaths: true
22    },
23    reporters: ['progress', 'kjhtml'],
24    port: 9876,
25    colors: true,
26    logLevel: config.LOG_INFO,
27    autoWatch: true,
28    browsers: ['Chrome'],
29    singleRun: false,
30    restartOnFileChange: true
31  });
32  };
33
```

Rys. 31 Konfiguracja modułu testowania

Źródło: opracowanie własne

Testowanie osobnych komponentów polega na wykorzystaniu funkcji danego frameworku, takich jak „describe”, „it” oraz „expect”. Działanie jest dość podobne do „AAA” (ang. *Arrange Act Assert*), czyli przygotowujemy dane wejściowe dalej wywołujemy działania testowanej funkcji i zatem porównujemy wynik faktyczny z wynikiem oczekiwanym.



```
input-text.component.spec.ts U X
src > app > shared > components > input-text > input-text.component.spec.ts > describe("In
1  import { ComponentFixture, TestBed } from '@angular/core/testing';
2  import { InputTextComponent } from './input-text.component';
3
   Run | Debug
4  describe('InputTextComponent', () => {
5    let component: InputTextComponent;
6    let fixture: ComponentFixture<InputTextComponent>;
7
8    beforeEach(async () => {
9      await TestBed.configureTestingModule({
10        declarations: [ InputTextComponent ]
11      })
12      .compileComponents();
13    });
14
15    beforeEach(() => {
16      fixture = TestBed.createComponent(InputTextComponent);
17      component = fixture.componentInstance;
18      fixture.detectChanges();
19    });
20
21    it('should create', () => {
22      expect(component).toBeTruthy();
23    });
24  });
25
```

Rys. 32 Testowanie głównego komponentu aplikacji  
Źródło: opracowanie własne

W momencie testowania udało się naprawić większość błędów logicznych, które nie zostały wykryte przy testowaniu manualnym. Zostały naprawione komponenty, które uwalniały aplikację oraz został przeprowadzony refaktoring kodu, a większe metody zostały podzielone na kilka mniejszych w celu łatwiejszego napisania testów oraz w celu łatwiejszego rozwoju i wsparcia, i lepszą do zrozumienia przez innego dewelopera.

## Podsumowanie

W ramach pracy dyplomowej został zaimplementowany i zaprojektowany program do obsługi klientów restauracji, który spełnił wymagania zarówno funkcjonalne i нефunkcjonalne, a w wyniku zostały osiągnięte ustawione cele. System rozwiązuje problemy opisane w pierwszym rozdziale pracy, czyli implementuje zautomatyzowanie procesów złożenia zamówienia, rezerwacji miejsca oraz czat z restauracją, w wyniku czego czas dokonania zakupu jest skrócony, a za pomocą instrumentów administratorskich można kontrolować dane restauracji oraz wyeksportować zamówienia użytkowników co jest pomocne przy obliczeniach księgowych w systemach ERP.

W trakcie projektowania były przeanalizowane dostępne na rynku rozwiązania podobnych systemów, czyli MCDonalds, Subway i inne. Narzędzie do tworzenia aplikacji mobilnej, czyli Xamarin.Forms, pozwala na korzystanie z jednego rozwiązania dla systemów Android oraz iOS, co pozwala skrócić czas na wytwarzanie aplikacji oraz na jej wspieranie. Technologie do wytwarzania aplikacji internetowych w postaci SPA, które były wykorzystane w projekcie, to Angular CLI 12 w połączeniu z biblioteką RxJS, ngx-bootstrap oraz boostwatch. Serwisy backendowe oparte o REST API służą do komunikacji z bazą danych MSSQL oraz do przetwarzania danych wchodzących i wychodzących. Została też uzupełniona wiedza nie tylko w dziedzinie wytwarzania oprogramowania, a też w obszarze marketingu oraz designu UI/UX. Oprócz nawyków technologicznych, które były wzbogacone podczas wykonywania pracy dyplomowej, ważnym zadaniem było nauczenie się projektowania użytecznych i nowoczesnych interfejsów użytkownika oraz projektowania systemów krosplatformowych.

Takie rozwiązanie w dniu dzisiejszym ma duży potencjał na rynku, a dzięki wzorcom projektowym w przyszłości może być łatwo wspierany i rozwijany lub rozszerzany w stronę pełnej automatyzacji i robotyzacji w wyniku czego zysk restauracji za pomocą aplikacji będzie maksymalnie wysoki, a zasoby ludzkie będą minimalizowane. Oprócz wymienionych możliwości rozwoju danego systemu, można byłoby też zaprojektować moduł dostawy zamówienia za pomocą robotów kurierskich z śledzeniem lokalacji oraz stacji doładowujących.

## **Literatura**

[Eric Evans, 2003]

Eric Evans, Domain Driven Design

DDD Series 2003, 560 s.

[Rakitow, 1998]

Rakitow A.I. Informatyka, technika w globalnych wymiarach historycznych. M

INION RAN, 1998, 104 s.

[Gaurav, Jeffrey, 2019]

Gaurav Aroraa, Jeffrey Chilberto, Hands-On Design Patterns with C# and .NET Core 2019, 410 s.

Packt Publishing, 2019

[Jeffrey Richter, 2012]

CLR via C#, Fourth Edition

Microsoft Corporation Press, 2012, 813 s

## **Źródła internetowe (WWW)**

[WWW-1, 2022]

<https://www.outerboxdesign.com/web-design-articles/mobile-ecommerce-statistics>, z dnia 05.01.2022

[WWW-2, 2021]

<https://www.statista.com/statistics/268251/number-of-apps-in-the-itunes-app-store-since-2008/>, z dnia 20.11.2021

[WWW-3, 2021]

<https://gs.statcounter.com/os-market-share>, z dnia 14.11.2021

[WWW-4, 2021]

<https://medium.com/xorum-io/cross-platform-mobile-apps-development-in-2021-xamarin-vs-react-native-vs-flutter-vs-kotlin-ca8ealf5a3e0>, z dnia 29.11.2021

[WWW-5, 2021]

<https://docs.microsoft.com/pl-pl/xamarin/cross-platform/app-fundamentals/building-cross-platform-applications/understanding-the-xamarin-mobile-platform>, z dnia 29.11.2021

[WWW-6, 2022]

<https://refactoring.guru/design-patterns>, z dnia 20.12.2021

## **Streszczenie**

**Wyższa Szkoła Informatyki i Zarządzania z siedzibą w Rzeszowie**  
**Kolegium Informatyki Stosowanej**

### **Streszczenie pracy dyplomowej** **System do obsługi klientów restauracji**

**Autor:** Yurii-Volodymyr Shcheliuk

**Promotor:** dr Marek Jaszuk

**Słowa kluczowe:** Xamarin.Forms, Angular, restauracja, .NET Core API

Realizowany system rozwiązuje czasochłonny problem obsługi dużej ilości klientów poprzez automatyzację procesów biznesowych, takich jak zamówienie rzeczy dostępnych w menu, rezerwacja miejsca w restauracji, a po dokonaniu opłaty za pomocą Stripe, można dołączyć do czatu z restauracją. Natomiast uprawnienia administratora pozwalają na edytowanie danych restauracji oraz eksportowanie do pliku dokonanych zamówień z wybranego dnia. Na tym etapie aplikacja daje podstawę do rozwoju systemu w kierunku pełnej automatyzacji oraz robotyzacji instytucji, w wyniku czego za pomocą aplikacji zysk będzie maksymalnie wysoki, a zasoby ludzkie będą minimalizowane. Praca przedstawia połączenie kilku aplikacji tworzących system krosplatformowy. Dla systemu operacyjnego Android oraz iOS był używany framework Xamarin.Forms, a wersja dla przeglądarek webowych jest oparta o Angular 12. Własny serwis backendowy oparty o .NET Core i REST API, pozwala tworzyć spójną logikę dla obu programów.



## **Załączniki**

Załączniki zostały zamieszczone na płycie CD, na której dodane:

1. Dokumentacja.
2. Folder z kodem źródłowym do aplikacji.