

Uniwersytet Śląski w Katowicach  
Wydział Nauk Ścisłych i Technicznych  
Instytut Informatyki

**Adam Krzysztof Wielogórski**  
344071

**Webowa aplikacja do poznawania ludzi realizująca płatności  
kryptowalutowe**

PRACA DYPLOMOWA  
PRACA INŻYNIERSKA

**Dr. Łukasz Strąk**

Sosnowiec 2024

Słowa kluczowe: Kryptowaluty, Angular, .NET

### **Oświadczenie autora pracy**

Ja, niżej podpisany/-a:

Adam Krzysztof Wielogórski

Autor pracy dyplomowej pt. Webowa aplikacja do poznawania ludzi realizująca płatności kryptowalutowe

Numer albumu: 344071

Student/-ka Wydziału Nauk Ścisłych i Technicznych Instytut Informatyki Uniwersytetu Śląskiego w Katowicach

Kierunku studiów: informatyka

Specjalności: Inżynieria Oprogramowania

Oświadczam, że ww. praca dyplomowa:

- została przygotowana przeze mnie samodzielnie<sup>1</sup>,
- nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst jednolity Dz. U. z 2006 r. Nr 90, poz. 631, z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- nie była podstawą nadania dyplomu uczelni wyższej lub tytułu zawodowego ani mnie, ani innej osobie.

Oświadczam również, że treść pracy dyplomowej zamieszczonej przeze mnie w Archiwum Prac Dyplomowych jest identyczna z treścią zawartą w wydrukowanej wersji pracy.

**Jestem świadomy/-a odpowiedzialności karnej za złożenie fałszywego oświadczenia.**

Data

Podpis autora

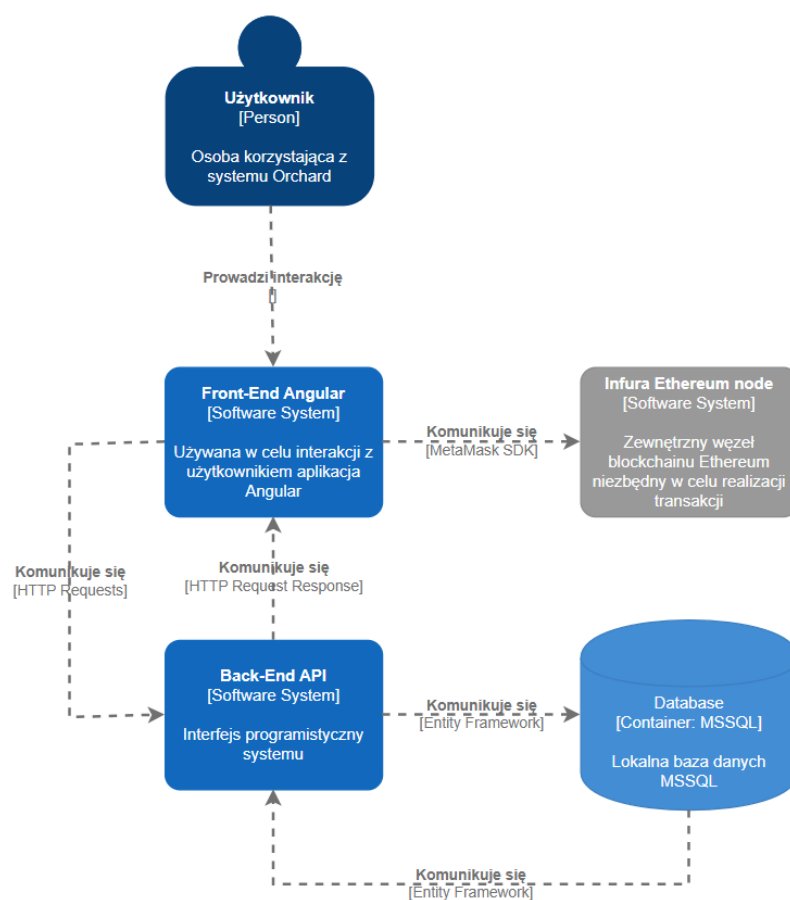
<sup>1</sup> uwzględniając merytoryczny wkład promotora (w ramach prowadzonego seminarium dyplomowego)

WSTĘP .....	4
1. GRAFICZNY INTERFEJS UŻYTKOWNIKA .....	8
1.1 Architektura komponentów .....	9
1.2 Współpraca z API .....	12
1.3 Nawigacja i usługi .....	13
1.4 Komponent główny .....	18
2. INTERFEJS PROGRAMISTYCZNY APLIKACJI .....	22
2.1 Mechanizmy zabezpieczające system .....	22
2.2 Wykorzystanie Entity Framework .....	25
2.3 Architektura kontrollerów API .....	29
3. BAZA DANYCH SYSTEMU .....	33
4. TRANSAKCJE KRYPTOWALUTOWE .....	36
4.1 Przygotowanie portfela MetaMask .....	37
4.2 Obsługa transakcji .....	41
5. ANALIZA PODOBNYCH ROZWIĄZAŃ .....	47
ZAKOŃCZENIE .....	51
BIBLIOGRAFIA .....	52
PRZYPISY .....	54

## WSTĘP

Aplikacje webowe są nieodłączną częścią dzisiejszego świata. Pozwalają na obsługę wielu systemów z poziomu przeglądarki internetowej. Umożliwiają też poznawanie innych osób oraz komunikację z nimi. Większość z nich będzie oferowała, za pewną opłatą, możliwość użytkowania dodatkowych funkcjonalności i jest to całkowicie zrozumiałe, gdyż godziwym jest wypłacić należność pracownikowi za jego pracę. Wszystkie omówione wyżej aspekty autor pracy połączył tworząc Orchard. Jest on systemem, na który składają się 4 części i umożliwia on poznawanie innych osób oraz oferuje możliwość wykupienia dodatkowych funkcjonalności za pomocą kryptowaluty ether za pośrednictwem portfela kryptowalutowego MetaMask.

Orchard pozwala swoim użytkownikom na wysyłanie wiadomości do siebie z kilku miejsc w systemie, przeglądanie wszystkich otrzymanych wiadomości z możliwością filtrowania, szukanie oraz przeglądanie profili innych osób, edycję własnego profilu a także wprowadzonych danych oraz kontakt z administratorem. Dodatkowe funkcjonalności obejmują powiadomienie o osobie, która odwiedziła profil użytkownika, zniesienie limitu wysyłania maksymalnie 5 wiadomości każdego dnia oraz specjalna ikona przy wieku, gdy profil użytkownika zostanie wyszukany.



Ilustracja 1. Diagram kontekstowy C4 komunikacji między częściami systemu.

Na powyższej ilustracji pokazano diagram kontekstowy C4 komunikacji między częściami systemu. Pokazano na nim, że użytkownik systemu prowadzi interakcję z warstwą Front-End systemu. Jest to jedyna część dla niego widoczna i dostępna. Front-End komunikuje się z warstwą Back-End używając żądań HTTP, a Back-End odpowiada na nie przez odpowiedzi na żądania HTTP. Front-End komunikuje się także z zewnętrznym węzłem blockchainu Ethereum Infura celem wykonywania transakcji używając MetaMask SDK. Jest to biblioteka umożliwiająca podłączenie aplikacji webowej do wtyczki przeglądarkowej MetaMask. Jedynie warstwa Back-End może komunikować się z bazą danych. W tym celu użyty został Entity Framework. Baza danych zwraca określone dane do warstwy Back-End.

Autor wybrał ten temat pracy ze względu na fascynację technologiami blockchain oraz kryptowalut. Aplikacja webowa zapewni, że system będzie działał poprawnie w przeglądarkach co pozwoli na łatwe korzystanie z systemu nie tylko na komputerach stacjonarnych, lecz także na urządzeniach mobilnych lub nowoczesnych konsolach do gier. Orchard ułatwi ludziom nawiązywanie znajomości. Płatności kryptowalutowe są obecnie najnowocześniejszym sposobem dokonywania płatności oraz oferują one prywatność na poziomie porównywalnym jedynie do handlu wymiennego.

## Cel i zakres pracy

Celem pracy jest stworzenie Webowej aplikacji do poznawania ludzi realizującej płatności kryptowalutowe. Ma ona za zadanie umożliwić ludziom komunikację z innymi jak również poznawanie ich przy jednoczesnej możliwości dokupienia dodatkowych funkcjonalności serwisu poprzez zastosowanie technologii kryptowalut.

Przedmiotem analizy oraz implementacji systemu informatycznego jest Orchard będący aplikacją webową służącą do poznawania ludzi realizującą płatności kryptowalutowe.

Autor zastosował dwie teoretyczne metody badawcze oraz jedną empiryczną metodę badawczą. Pierwszą z zastosowanych, teoretycznych metod badawczych jest Analiza. Autor chcąc rozwiązać problem inżynierski dokonał jego analizy. Na jej podstawie wyodrębnił 4 człony, które poddane drugiej, teoretycznej metodzie badawczej – syntezie, utworzą aplikację inżynierską. Te człony to:

1. Front-End – Jest to część aplikacji, która odpowiada za interakcje z użytkownikiem oraz mechanizm obsługi tych interakcji do pewnego etapu.
2. Back-End – Jest to część aplikacji, która odpowiada na żądania Front-End'u, dba o bezpieczeństwo danych w bazie danych oraz jest swoistym pomostem pomiędzy nimi. Na jej podstawie również, z użyciem Entity Framework, zdefiniowana jest też struktura bazy danych.
3. Baza danych – Jest to część całego systemu, która odpowiada za przechowywanie danych o użytkownikach, wiadomościach oraz powiadomieniach. Na żądanie, przetworzone przez API, wyszukuje odpowiednie dane przez zapytania SQL, na które są odpowiednio tłumaczone fragmenty kodu przez ORM Entity Framework, i odsyła je.
4. Transakcje kryptowalutowe – Jest to część całego systemu, która odpowiada za transakcje kryptowalutowe użytkowników. Autor wykorzystał w tym celu API Infura (jest to węzeł blockchainu ethereum, który udostępnia API do wykonywania operacji na blockchainie przez niego, gdyż, żeby dokonywać operacji na blockchainie trzeba zrobić to przez węzeł, a wiąże się to z koniecznością dużych wymagań sprzętowych, więc autor skorzystał z zewnętrznego węzła) oraz MetaMask SDK.

Autor zastosował syntezę w celu połączenia elementów zrodzonych z analizy w jeden system. Składając Front-End oraz Back-End dokonano wielu badań z zakresu komunikacji między tymi warstwami. Synteza tych dwóch warstw była najbardziej czasochłonna ze względu na ilość opcji komunikacyjnych, z których każda musiała być obsługana odpowiednio. Złączenie Back-Endu oraz bazy danych przebiegło pomyślnie. ORM jakim jest Entity Framework znacząco skrócił ten proces oraz go zautomatyzował. Badając syntezę reszty systemu z Transakcjami kryptowalutowymi sprawdzano działanie MetaMask SDK w zakresie realizacji tej części pracy. Dokonano odkrycia, że wykorzystując to narzędzie oraz jego współpracę z węzłem Infura można osiągnąć założony cel. Dokonano syntezy wszystkich 4 członów pracy i stworzono z nich jeden system.

Zastosowano także empiryczną metodę badawczą - metodę obserwacyjną. Cały okres prac i badań obfitował w obserwacje. Autor obserwował zachowania poszczególnych elementów systemu w celu zrozumienia ich działania oraz możliwości zrealizowania za ich pomocą celu pracy. Na ich podstawie dokonano wielu odkryć odnośnie funkcjonowania frameworków, technologii oraz komunikacji między komponentami systemu. Każda obserwacja była podstawą do krótkich rozważań przyczynowo skutkowych i wyciągnięcia wniosków. Były to obserwacje bezpośrednie z interwencjami, gdyż autor mógł w te obserwacje jak i w ich przebieg ingerować. Były to badania absolutnie niezbędne do zrealizowania pracy.

Praca zawiera 4 główne rozdziały:

1. Front-End - rozdział poświęcony warstwie wizualnej aplikacji
2. Back-End - rozdział poświęcony interfejsowi programistycznemu aplikacji
3. Baza danych - rozdział poświęcony bazie danych systemu
4. Transakcje kryptowalutowe - rozdział poświęcony transakcjom kryptowalutowym.

Prowadząc badania nad analizą istniejących rozwiązań stwierdza się co następuje - Na czas pisania tej pracy istnieje wiele aplikacji pozwalających na poznawanie ludzi, lecz nie znaleziono takiej, która łączy tę funkcjonalność z możliwością realizacji płatności kryptowalutowych. "Facebook", "Instagram", "Tinder" czy "Badoo" pozwalają na poznawanie nowych osób, lecz nie realizują płatności kryptowalutowych. Autor podał kilka podobnych istniejących rozwiązań w nieprzypadkowej kolejności. Jest ona kierowana stopniem zaawansowania danego systemu. Uwidacznia to, jak unikalne rozwiązanie autor zastosował w skali rozwiązań już istniejących, gdyż czy to system tak złożony jak Facebook, czy znacznie mniej zaawansowane Badoo nie realizują one płatności kryptowalutowych. Na czas pisania tej pracy rozwiązanie zastosowane przez autora jest niespotykane oraz niespopularyzowane. Odniesie tego faktu do obecnej rzeczywistości pozwala dostrzec przyczyny takiego stanu. Blockchain'y to stosunkowo nowe technologie, a kryptowaluty, obrót nimi oraz wpływ na ich cenę, która jest bardzo niestabilna skutecznie opóźnia proces popularyzacji takiego sposobu płatności. Podobne systemy oferują szybkie płatności dzięki usłudze BLIK, jednak osoby, które maksymalnie cenią sobie prywatność do tego stopnia, że nie życzą sobie, aby nawet organy państwowe dysponowały wiedzą o ich wydatkach nie znajdują upragnionej satysfakcji oraz prywatności korzystając z takich rozwiązań. Orchard będzie dla nich idealnym narzędziem, gdyż przeprowadzając transakcję kryptowalutami (autor posłuży się przykładem sieci Ethereum) niemożliwe jest ustalenie tożsamości właściciela portfela przez jakąkolwiek organizację prywatną lub państwową.

# 1. GRAFICZNY INTERFEJS UŻYTKOWNIKA

Do zrealizowania części Front-End systemu autor wykorzystał framework Angular w wersji 17. Został on stworzony przez firmę Google<sup>[1]</sup> i jest oparty na języku programowania TypeScript. Pozwala on na szybkie tworzenie interfejsów graficznych użytkownika. Framework używa architektury opartej na komponentach (każdy komponent posiada osobny kod HTML, CSS oraz TypeScript przez co łatwiej jest testować poszczególne elementy aplikacji). Angular ma także wbudowany Routing, który pozwala na definiowanie ścieżek w aplikacji oraz używa dyrektyw będących dodatkowymi funkcjonalnościami, które mogą zmieniać zachowanie lub wygląd elementów DOM<sup>[2]</sup>. Autor posłużył się także frameworkiem Bulma, który jest frameworkiem CSS, dostarczającym dostosowywalne, gotowe komponenty do użycia w aplikacji, będące dodatkowo responsywne<sup>[3]</sup>.

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>OrchFrontAnguPraca</title>
6    <base href="/">
7    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.0/css/bulma.min.css">
8    <meta name="viewport" content="width=device-width, initial-scale=1">
9    <link rel="icon" type="image/x-icon" href="favicon.ico">
10 </head>
11 <body>
12   <app-root></app-root>
13 </body>
14 </html>
```

Ilustracja 2. Kod indexu głównego z podkreślonym załączeniem arkusza stylów Bulma

Na powyższej ilustracji pokazano kod pliku index.html, który zostanie załadowany tuż po uruchomieniu aplikacji. Podkreślony fragment to załączenie arkusza stylów Bulma.

Autor, stylizując interfejs graficzny użytkownika, działał z przyświecającą mu ideą o nowoczesności i unikalności jego wyglądu. Prace nad nim oraz badania prowadzone przy użyciu metody obserwacyjnej pozwoliły na utworzenie interfejsu stylizowanego na kolor ciemny. Nadane tło przestrzeni kosmicznej strony logowania uwydatnia efekt nowoczesności i ambitnym założeniem autora było, aby dzięki temu, po zalogowaniu, użytkownik mógł odczuć chociażby drobną immersję “zanurzając się” w Orchard – jak gdyby wchodził do innego świata. Kolorystycznie w aplikacji dominują barwy ciemne, a kontrastujące z nimi barwy jasne uwydatniają na ekranie elementy, na które użytkownik miałby zwrócić uwagę w pierwszej kolejności.

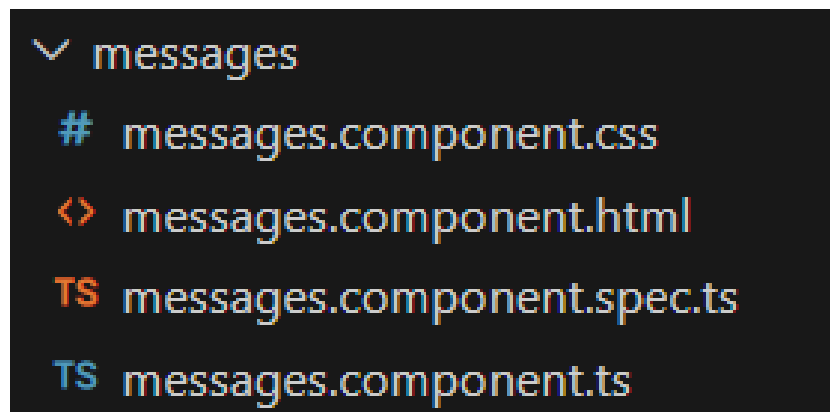


## 1.1 Architektura komponentów

Autor, tworząc warstwę Front-End'u systemu, komponował ją z mniejszych jednostek zwanych komponentami. Na każdy komponent we frameworku Angular składają się podstawowo 4 pliki:

- arkusz styli CSS szablonu HTML komponentu<sup>[4]</sup>
- szablon HTML określający co zostanie pokazane na stronie<sup>[5]</sup>
- klasa TypeScript definiująca zachowanie<sup>[6]</sup>
- plik zawierający testy komponentu z rozszerzeniem .spec.ts<sup>[7]</sup>

Komponent messages został opracowany w celu umożliwienia użytkownikowi systemu przeglądania wiadomości wysłanych do niego przez innych użytkowników, natychmiastowej odpowiedzi na nie oraz oględzin profilu nadawcy.



Ilustracja 3. Zawartość komponentu messages.

Na powyższej ilustracji pokazano zawartość komponentu messages. Ma on budowę podstawową dla komponentu we frameworku Angular.

```

1  @if (canBeShown == 2){
2    <div class="hero is-fullheight-with-navbar is-dark">
3      <div class="columns">
4        <div class="column is-one-third is-offset-one-third">
5          <article class="panel">
6
7            <div class="panel-block">
8              <p class="control has-icons-left">
9                <input class="input is-hovered is-rounded is-link" type="text" placeholder="Filter by username" [value]="filter()" (keyup)="filterMessages($event)"/>
10               <span class="icon is-left">
11                 <i class="fa fas fa-search" aria-hidden="true"></i>
12               </span>
13             </p>
14           </div>
15
16           @for (user of objectsToDisplay; track user) {
17             <div class="panel-block is-block">
18               <div class="columns is-vcentered">
19                 <div class="column has-text-centered is-one-fifth">
20                   <figure class="image is-inline-block">
21                     <img class="{{user.Username}}" alt="Image"/>
22                   </figure>
23                 </div>
24                 <div class="column has-text-left">
25                   <strong>{{user.Username}}</strong>
26                   {{user.MessageText}}
27                 </div>
28                 <div class="column has-icons-right has-text-right is-one-fifth">
29                   <div class="columns is-vcentered">
30                     <div class="column"></div>
31                     <div class="column has-icons-left has-text-left is-two-fifth">
32                       <a>
33                         <span class="icon is-small" style="font-size: 25px;">
34                           <i class="fa fas fa-envelope" aria-hidden="true" (click)="SendMessage(user.AuthorId)"></i>
35                         </span>
36                       </a>
37                     </div>
38                     <div class="column has-icons-left has-text-left">
39                       <a>
40                         <span class="icon is-small" style="font-size: 25px;">
41                           <i class="fa fas fa-arrow-right" aria-hidden="true" (click)="SeeUserProfile(user.AuthorId)"></i>
42                         </span>

```

Ilustracja 4. Część zawartości pliku messages.component.html.

Na powyższej ilustracji pokazano część zawartości pliku messages.component.html. Zawiera on elementy takie jak struktura kolumnowa, panel, warstwę prezentacji funkcjonalności filtrowania wiadomości, pętlę, data binding (będący zjawiskiem, w którym występuje połączenie między elementem klasy komponentu definiującej zachowanie oraz jego wartością, użytą w pliku .html<sup>[8]</sup>, a algorytm Angular Change Detection odpowiada za sprawdzanie czy stan aplikacji się zmienił, toteż czy którykolwiek element DOM potrzebuje aktualizacji<sup>[9]</sup>) oraz obsługa zdarzeń.

```

1  .panel-block{
2    background-color: whitesmoke;
3    border-style: 10px solid;
4    border-color: black;
5    align-items: left;
6  }
7
8  textarea{
9    resize: none;
10 }
11
12 figure > img{
13   height: 64px;
14   width: 64px;
15 }

```

Ilustracja 5. Zawartość pliku messages.component.css.

Na powyższej ilustracji pokazano zawartość pliku messages.component.css. Zawiera on definicję niektórych elementów stylu dla klasy .panel-block, elementu textarea oraz dla wszystkich elementów img, których rodzicem jest element figure.

```
1  import { ComponentFixture, TestBed } from '@angular/core/testing';
2
3  import { MessagesComponent } from './messages.component';
4
5  describe('MessagesComponent', () => {
6    let component: MessagesComponent;
7    let fixture: ComponentFixture<MessagesComponent>;
8
9    beforeEach(async () => {
10     await TestBed.configureTestingModule({
11       imports: [MessagesComponent]
12     })
13     .compileComponents();
14
15     fixture = TestBed.createComponent(MessagesComponent);
16     component = fixture.componentInstance;
17     fixture.detectChanges();
18   });
19
20   it('should create', () => {
21     expect(component).toBeTruthy();
22   });
23 });
```

Ilustracja 6. Zawartość pliku messages.component.spec.ts.

Na powyższej ilustracji pokazano zawartość pliku messages.component.spec.ts. Jest to plik z testami dla komponentu. W tym konkretnym pliku autor nie zmieniał nic - zawartość jest taka, jaką wygenerowało polecenie generujące cały komponent.

```
18  export class MessagesComponent implements OnInit {
19
20  constructor(private fb : FormBuilder, private apiConn : APIService, private LoggedUserData : LoggedUserDataService, private router1 : Router){
21    this.MessageToSendForm = this.fb.group({
22      Message:"type message here"
23    })
24
25    this.formDataMessage = new FormData();
26  }
27
28  ngOnInit() {
29    const id = this.LoggedUserData.GetLoggedInUserId();
30    this.apiConn.GetAllUserMessages(id)
31    .pipe(
32      catchError(error => {
33        if (error.status === 404) {
34          this.canBeShown = 1;
35        }
36        return throwError(() => new Error("Error occurred"));
37      }),
38      map((response) => {
39        const data = response.body;
40        return(data);
41      })
42    )
43    .subscribe({
44      next: (result) => {
45        this.canBeShown = 2;
46        this.messages = result.data.messages;
47        this.users = result.data.users;
48        this.messages.forEach((message: { authorId: any; content: any; sendDate: Date; }) => {
49          this.users.forEach((user: { id: any; username:string; age:number; }) => {
50            if(message.authorId == user.id){
51              let prof = new Blob();
52              let um : UserMessage = {
53                ProfilePhoto: prof,
54                Username: user.username,
55                Age: user.age,
56                MessageText: " " + message.content,
57                SendDate: message.sendDate,
58                AuthorId: user.id
59              }
60              if(!this.objectsToDisplay.includes(um)){
61                this.objectsToDisplay.push(um);
62              }
63            }
64          })
65        })
66      }
67    });
68  }
69 }
```

Ilustracja 7. Część zawartości pliku messages.component.ts.

Na powyższej ilustracji pokazano część zawartości pliku `messages.component.ts`. Jest to klasa określająca zachowanie komponentu. Na powyższym rysunku możemy zobaczyć konstruktor owej klasy oraz implementację interfejsu `OnInit` oraz workflow typu `pipe`.

## 1.2 Współpraca z API

Dwa elementy z całego systemu będącego rozwiązaniem zadanego problemu inżynierskiego, to Front-End oraz Back-End. W celu umożliwienia komunikacji tych dwóch modułów ze sobą autor zastosował w części Front-End usługę `HttpClient`. Pozwala ona wysyłać żądania `http`<sup>[10]</sup> do API systemu i odbierać odpowiedzi na owe żądania. Za ich pomocą Front-End współpracuje z API i zapewnia działanie konkretnych funkcjonalności.

```
export const appConfig: ApplicationConfig = {  
  providers: [provideRouter(routes), provideClientHydration(), provideHttpClient(withFetch()), LoggedUserDataServiceService]  
};
```

Ilustracja 8. Część zawartości pliku konfiguracyjnego aplikacji.

Zaznaczony fragment kodu pozwala na konfigurację usługi `HttpClient`, aby była dostępna do wstrzyknięcia<sup>[11]</sup>. Jest to niezbędny krok do dalszej poprawnej konfiguracji usługi oraz całej komunikacji.

```
9  export class APIConnectionService {  
26  MessagesSendUserMessage = 'https://localhost:7023/api/Messages/recvMessageSendByUserToUser';  
27  UsersCheckIfUsernameExists = 'https://localhost:7023/api/Users/checkifusernameexists';  
28  UsersCheckIfEmailExists = 'https://localhost:7023/api/Users/checkifemailexists';  
29  MessagesGetLast5UserMessages = 'https://localhost:7023/api/Messages/GetLast5UserMessages';  
30  PaymentsCheck = 'https://localhost:7023/api/Payments/paymentcheck';  
31  PaymentsStoretxhash = 'https://localhost:7023/api/Payments/storetxhash';  
32  PaymentsGettxhash = 'https://localhost:7023/api/Payments/gettxhash';  
33  
34  PermitMessageSend(Id : any){  
35    const token = this.TC.getToken();  
36    const headers = new HttpHeaders({  
37      'Authorization': `Bearer ${token}`  
38    });  
39    return this.http.post<any>(this.MessagesPermitMessageSend, {Id}, {observe: 'response', headers: headers })  
40  }  
41  
42  GetUserById(Id : any){  
43    const token = this.TC.getToken();  
44    const headers = new HttpHeaders({  
45      'Authorization': `Bearer ${token}`  
46    });  
47    return this.http.post<any>(this.UsersGetUserByIdUrl, {Id}, {observe: 'response', headers: headers })  
48  }  
49  
50  PaymentGettxhash(Id : any){  
51    const token = this.TC.getToken();  
52    const headers = new HttpHeaders({  
53      'Authorization': `Bearer ${token}`  
54    });  
55    return this.http.post<any>(this.PaymentsGettxhash, {Id}, {observe: 'response', headers: headers })  
56  }  
57  
58  PaymentStoretxhash(Id : any, City : string){  
59    const token = this.TC.getToken();  
60    const headers = new HttpHeaders({  
61      'Authorization': `Bearer ${token}`  
62    });  
63    return this.http.post<any>(this.PaymentsStoretxhash, {Id, City}, {observe: 'response', headers: headers })  
64  }  
65  
66  PaymentsChecking(Id : any){  
67    const token = this.TC.getToken();  
68    const headers = new HttpHeaders({  
69      'Authorization': `Bearer ${token}`
```

Ilustracja 9. Część zawartości klasy definiującej usługę komunikacji z API.

Na powyższej ilustracji pokazano część kodu klasy definiującej usługę komunikacji z API. Klasa zawiera właściwości przechowujące konkretne adresy endpoint'ów API, konstruktor oraz metody, za pomocą których można odpytać konkretny endpoint. Większość metod zawartych w tej klasie zawiera nagłówek z tokenem JWT w celu autoryzacji. Jest to mechanizm zaimplementowany przez autora w API, który uniemożliwia dostanie się do większości endpoint'ów bez ważnego tokenu JWT. Jest on przyznawany po poprawnym zalogowaniu się. Właściwość TC jest instancją klasy `TokenContainerService` zadeklarowaną i zdefiniowaną w konstruktorze przez mechanizm `dependency injection`.

```
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class TokenContainerService {
7
8    constructor() { }
9
10   getToken(){
11     return sessionStorage.getItem("Token");
12   }
13 }
14
```

Ilustracja 10. Usługa `TokenContainer`.

Na powyższej ilustracji pokazano klasę definiującą usługę `TokenContainer`. Usługa ta odpowiada za odzyskiwanie z magazynu sesji tokenu JWT przekazanego przez API podczas logowania. Będzie on niezbędny do uzyskania dostępu do większości endpoint'ów API. Na ilustracji 9 autor pokazał wywołanie metody `getToken()` z klasy `TokenContainerService`, której definicję pokazano na ilustracji 10.

### 1.3 Nawigacja i usługi

Komponując nawigację podstron i komponentów autor wykorzystał usługę `Router` z modułu `@angular/router`. Pozwala ona na nawigowanie między widokami podczas użytkowania aplikacji<sup>[12]</sup>. Plikiem definiującym konkretne ścieżki w systemie jest plik `app.routes.ts`. To tam autor zawarł konkretne wskazania do jakiego komponentu powinna skierować się aplikacja w przypadku wejścia pod konkretny adres URL oraz specjalne warunki, które muszą zostać spełnione, aby użytkownik mógł się dostać do komponentu o danym adresie.

```

1 import { Routes } from '@angular/router';
2 import { LoginPageComponent } from './login-page/login-page.component';
3 import { DashboardComponent } from './dashboard/dashboard.component';
4 import { BrowseUserGuard, TokenGuard } from '../AuthGuardService/auth-guard.service';
5 import { RegisterComponent } from './register/register.component';
6 import { LoggedUserProfileComponent } from './logged-user-profile/logged-user-profile.component';
7 import { NotificationsComponent } from './notifications/notifications.component';
8 import { PeopleSearchComponent } from './people-search/people-search.component';
9 import { MessagesComponent } from './messages/messages.component';
10 import { PaymentsComponent } from './payments/payments.component';
11 import { RouterModule } from '@angular/router';
12 import { ContactComponent } from './contact/contact.component';
13 import { OtherUserProfileComponent } from './other-user-profile/other-user-profile.component';
14
15 export const routes: Routes = [
16   {path: '', component: LoginPageComponent},
17   {path: 'Dashboard', component: DashboardComponent, canActivate: [TokenGuard]},
18   {path: 'Register', component: RegisterComponent},
19   {path: 'Profile', component: LoggedUserProfileComponent, canActivate: [TokenGuard]},
20   {path: 'Notifications', component: NotificationsComponent, canActivate: [TokenGuard]},
21   {path: 'Search', component: PeopleSearchComponent, canActivate: [TokenGuard]},
22   {path: 'Messages', component: MessagesComponent, canActivate: [TokenGuard]},
23   {path: 'Payments', component: PaymentsComponent, canActivate: [TokenGuard]},
24   {path: 'Contact', component: ContactComponent, canActivate: [TokenGuard]},
25   {path: 'BrowseUsers', component: OtherUserProfileComponent, canActivate: [TokenGuard, BrowseUserGuard]}
26 ];

```

Ilustracja 11. Zawartość pliku ze ścieżkami nawigacyjnymi.

Na powyższej ilustracji pokazano zawartość pliku ze ścieżkami nawigacyjnymi aplikacji. Pokazano zaimportowanie licznych komponentów, których import jest możliwy tylko ze względu na słowo kluczowe ‘export’ przed słowem ‘class’ w definicji klasy<sup>[13]</sup>, a jest to zabieg niezbędny w celu określenia ścieżki do danego komponentu w pokazanej na ilustracji 10 klasie. Autor pokazał także export stałej routes typu Routes będącej tablicą ścieżek w aplikacji. Parametr path określa ścieżkę do danego komponentu (localhost:4200/Dashboard), component określa komponent docelowy po przejściu w daną ścieżkę, a canActivate pozwala na określenie specjalnych warunków, które muszą zostać spełnione celem umożliwienia użytkownikowi przejścia do danego adresu.

Każda ścieżka może zostać zabezpieczona. W obecnej w aplikacji wersji framework Angular (Angular 17) są to tzw. Functional Router Guard. W starszych wersjach frameworku Angular istniało podejście klasowe do zabezpieczania ścieżek, ale począwszy od Angular 15 jest ono już przestarzałe i niezalecane. Obecnie zaleca się użycie podejścia funkcjonalnego<sup>[14]</sup>.

```

1  import { inject } from "@angular/core";
2  import { DashboardComponent } from "../src/app/dashboard/dashboard.component";
3  import { TokenContainerService } from "../token-container.service";
4  import { Router } from "@angular/router";
5  import { LoggedUserDataServiceService } from "../LoggedUserData/logged-user-data-service.service";
6
7  export const TokenGuard = () => {
8
9      const TC = inject(TokenContainerService);
10     const router = inject(Router);
11     if(TC.getToken() != null){
12         return true;
13     }
14     else{
15         router.navigate([""]);
16         return false;
17     }
18 }
19
20 export const BrowseUserGuard = () => {
21     const router = inject(Router);
22     const loggedUserService = inject(LoggedUserDataServiceService);
23     if(loggedUserService.LoggedUser != null){
24         return true;
25     }
26     else{
27         router.navigate([""]);
28         return false;
29     }
30 }

```

Ilustracja 12. Definicje strażników ścieżek.

Na powyższej ilustracji pokazano definicje strażników ścieżek funkcji. Ich działanie polega na zwróceniu wartości boolean true lub false w zależności od pewnej logiki. Jeżeli zwrócone zostanie true to strażnik pozwoli użytkownikowi na dostęp do konkretnego komponentu przez zadaną ścieżkę. W przeciwnym wypadku nie pozwoli. TokenGuard jest strażnikiem sprawdzającym czy obecny jest token JWT, a BrowseUserGuard sprawdza czy obecnie zalogowany jest jakikolwiek użytkownik. Przez metodę inject konkretne usługi są wstrzykiwane do ciał funkcji co pozwala na skorzystanie z nich. Jest to zastosowanie mechanizmu dependency injection wspomnianego wyżej przez metodę inject()<sup>[15]</sup>.

W celu przechowania danych użytkownika autor wykorzystał usługi. Działają one jako twory typu 'Singleton' czyli posiadające jedną instancję dla całej aplikacji oraz mogą być wstrzyknięte do innych komponentów lub funkcji celem wykorzystania danych w nich zawartych. Usługa TokenContainerService wstrzyknięta do funkcji TokenGuard zwróci takie same dane jak wstrzyknięta do komponentu Dashboard ze względu na bycie tworem typu Singleton.

```

1  import { Injectable } from '@angular/core';
2  import { User } from '../Models/user';
3
4  @Injectable({
5    providedIn: 'root'
6  })
7  export class LoggedUserDataServiceService {
8
9    public LoggedUserId : string = "default value";
10   public LoggedUserRole : string = "default value";
11   public LoggedUser! : User;
12   public UserToProfileView : User = {
13     Id:"def",
14   }
15
16   constructor() {
17   }
18
19   SetLoggedUserId(Id : string){
20     this.LoggedUserId = Id;
21   }
22
23   GetLoggedUserId(){
24     return this.LoggedUserId;
25   }
26
27   SetLoggedUserRole(Role : string){
28     this.LoggedUserRole = Role;
29   }
30
31   GetLoggedUserRole(){
32     return this.LoggedUserRole;
33   }
34 }
35

```

Ilustracja 13. Zawartość klasy definiującej usługę przechowywania danych użytkownika.

Na powyższej ilustracji pokazano klasę definiującą usługę LoggedUserDataServiceService. Pozwala ona na przechowywanie danych o obecnie zalogowanym użytkowniku. Może być wstrzyknięta do innych komponentów celem wykorzystania w nich przechowywanych w usłudze danych. Zawiera 4 właściwości - dwie typu string oraz dwie typu User (jest to model stworzony przez autora na potrzeby pracy), pusty konstruktor oraz cztery metody. Metody owe to metody typu Get oraz Set dla dwóch właściwości typu string.



Framework Angular pozwala na tworzenie własnych modeli danych. Są to złożone typy danych tworzone z innych typów podstawowych oraz logiki z nimi związanej<sup>[16]</sup>. Podczas prac i badań autor utworzył 4 takie modele. Przytoczony zostanie tutaj oraz wyjaśniony jeden z nich.

```
1  export interface User {  
2      Id?: string;  
3      Username?: string;  
4      Email?: string;  
5      Region?: string;  
6      Role?: string;  
7      Age?: number;  
8      City?: string;  
9      ProfilePhoto?: any;  
10 }  
11
```

Ilustracja 14. Definicja modelu User.

Na powyższej ilustracji pokazano definicję modelu User. Jest to interfejs łączący tylko podstawowe typy danych w jeden - odpowiadający niezbędnym do funkcjonowania aplikacji właściwościom encji User z API. Służy do mapowania odpowiedzi z API będącej encją bazy danych na typ User wewnętrzny dla części Front-End. Pozwala na wykorzystanie wartości z danych kolumn tej encji w aplikacji.

```
21  export class LoginPageComponent {  
50  
51      GetApiDataFirstUser() {  
52          this.API_COMM.login(this.myForm.get('email')?.value, this.myForm.get('password')?.value)  
53              .pipe(  
54                  catchError(error => {  
55                      if (error.status === 404) {  
56                          this.areCredentialsvalid = false;  
57                      }  
58                      return throwError(() => new Error("Error occurred"));  
59                  })  
60              ),  
61              map((response) => {  
62                  const data = response.body;  
63                  const user : User = {  
64                      Id: data.user.id,  
65                      Username: data.user.username,  
66                      Email: data.user.email,  
67                      Role: data.user.role,  
68                      Age: data.user.age,  
69                      Region: data.user.region,  
70                      City: data.user.city,  
71                      ProfilePhoto: data.user.ProfilePhoto  
72                  };  
73                  this.verificationPassed = true;  
74                  return { user, token: data.token };  
75              })  
76          ).subscribe({  
77              next: (result) => {  
78                  sessionStorage.setItem("Token", result.token);  
79                  this.LoggedUserData.LoggedUser = result.user;  
80                  this.LoggedUserData.SetLoggedUserId(result.user.Id!);  
81                  this.LoggedUserData.SetLoggedUserRole(result.user.Role!);  
82                  this.userRecived = result.user;  
83                  this.JWTToken = result.token;  
84                  this.ValidateUser();  
85              },  
86              error: (error) => {  
87                  console.error('API Error:', error);  
88              }  
89          });  
90      }  
91  }
```

Ilustracja 15. Praktyczne użycie modelu User.

Na powyższej ilustracji pokazano praktyczne wykorzystanie modelu User. Podczas mapowania odpowiedzi z API na żądanie zalogowania użytkownika ciało odpowiedzi jest przetwarzane na obiekt typu User (zadeklarowany przez autora interfejs). Tworzony jest nowy obiekt typu User oraz poszczególne jego właściwości są uzupełniane danymi z odpowiedzi API, następnie razem z tokenem jest przekazywany do dalszego przetworzenia.

## 1.4 Komponent główny

W wyniku badań nad widocznością części systemu przez pryzmat obecności zalogowanego użytkownika, autor zawarł elementy takie jak stopka oraz pasek nawigacyjny w komponencie głównym. W gotowym kodzie strony, zostanie wygenerowany kod komponentu, do którego przekieruje użytkownika router, a miejsce znacznika `<router-outlet>` zajmie szablon strony HTML komponentu docelowego. Obecność stopki oraz paska nawigacji są warunkowe. Uzyskano efekt, w którym pasek nawigacji oraz stopka pokazują się tylko, gdy użytkownik jest zalogowany do systemu.

```
1 <nav class="navbar is-white" role="navigation" aria-label="main navigation" *ngIf="visible_nav">
2   <a role="button" class="navbar-burger" aria-label="menu" aria-expanded="false" data-target="navig" id="burger" (click)="ShowMenu()">
3     <span aria-hidden="true"></span>
4     <span aria-hidden="true"></span>
5     <span aria-hidden="true"></span>
6     <span aria-hidden="true"></span>
7   </a>
8
9   <div id="navig" class="navbar-menu">
10    <div class="navbar-start">
11      <a class="navbar-item" [routerLink]="['/Profile']">Your profile</a>
12
13      <a class="navbar-item" [routerLink]="['/Notifications']">Recent Notifications</a>
14
15      <a class="navbar-item" [routerLink]="['/Search']">Search for people</a>
16
17      <div class="navbar-brand">
18        <a class="navbar-item" [routerLink]="['/Dashboard']">
19          
20        </a>
21      </div>
22
23      <a class="navbar-item" [routerLink]="['/Messages']">Your Messages</a>
24
25      <a class="navbar-item" [routerLink]="['/Payments']">Transactions</a>
26
27      <a class="navbar-item" [routerLink]="['/Contact']">Contact Us</a>
28
29      <a class="navbar-item" (click)="Logout()">Logout</a>
30    </div>
31  </div>
32 </nav>
33
34 <router-outlet></router-outlet>
35
36
37 <footer class="footer" *ngIf="visible_nav">
38   <div class="content has-text-centered">
39     <p>
40       <strong>Orchard</strong> by Adam Wielogórski.
41       <a [routerLink]="['/Contact']">Contact Us</a> if you have any questions
42     </p>
43   </div>
44 </footer>
```

Ilustracja 16. Kod HTML komponentu głównego.

Na powyższej ilustracji pokazano kod HTML komponentu głównego. Ilustracja pokazuje zastosowanie tzw. ‘Burger Menu’ czyli zdarzenia, w którym okno przeglądarki zmniejszone do odpowiednio małych rozmiarów zmieni swój styl z długiego paska nawigacji na rozwijane menu złożone z trzech kresek ułożonych jedna pod drugą stąd

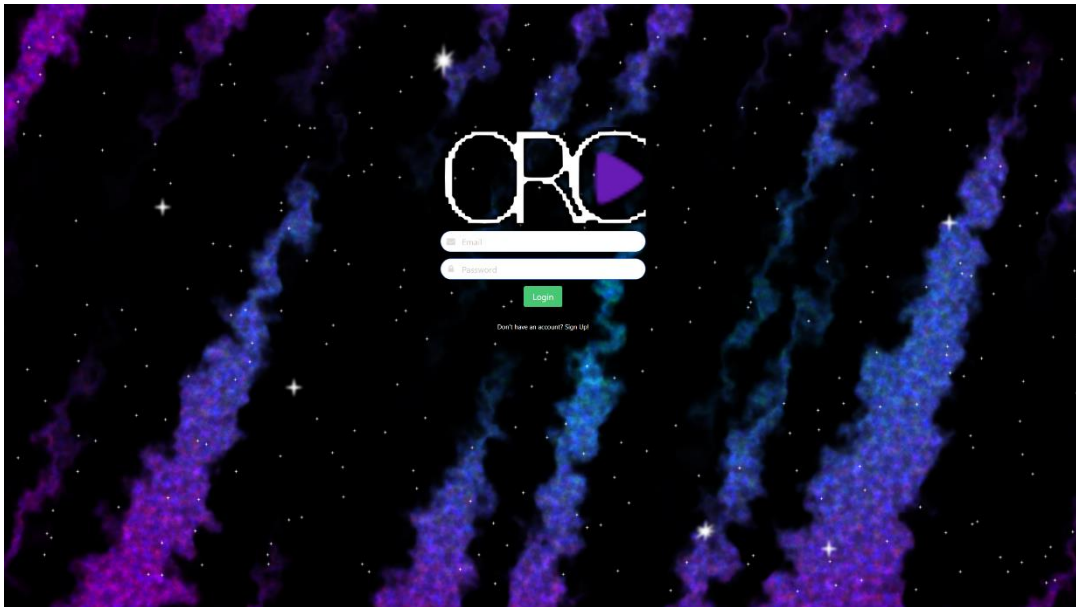
nazwa 'Burger Menu'. Widoczne są także konkretne odnośniki do innych komponentów przez właściwość [routerLink], która przekieruje przez odpowiednią ścieżkę (rozdział 1.4) do komponentu docelowego, którego kod zostanie wygenerowany w miejscu znacznika <router-outlet>. Na dole umieszczono stopkę.

Jak wspomniano wyżej, widoczność stopki oraz paska nawigacji jest warunkowa. Określa to właściwość \*ngIf. Jeżeli wartość w tej właściwości będzie równa true, to zarówno stopka jak i pasek nawigacji pokażą się. W przeciwnym razie nie zostaną wygenerowane<sup>[17]</sup>.

```
1  import { Component } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { Router, RouterOutlet } from '@angular/router';
4  import { APIConnectionService } from '../APIConnectionService/api-connection.service';
5  import { RouterLink } from '@angular/router';
6  import { RouterModule } from '@angular/router';
7  import { LoggedUserDataServiceService } from '../LoggedInUserData/logged-user-data-service.service';
8
9  @Component({
10   selector: 'app-root',
11   standalone: true,
12   imports: [CommonModule, RouterOutlet, RouterLink, RouterModule],
13   templateUrl: './app.component.html',
14   styleUrls: ['./app.component.css']
15 })
16 export class AppComponent {
17   isMenuOpen: boolean = false;
18
19   title = 'Orch_FrontAngu_Praca';
20   visible_nav: boolean = true;
21   constructor(private API_COMM : APIConnectionService, private loggeduserdata : LoggedUserDataServiceService, private router : Router) {}
22
23   ShowMenu(){
24     let menu = document.getElementById('navig')
25     menu?.classList.toggle('is-active');
26   }
27
28   Logout(){
29     sessionStorage.removeItem("Token")
30     this.router.navigate(['/']);
31   }
32 }
```

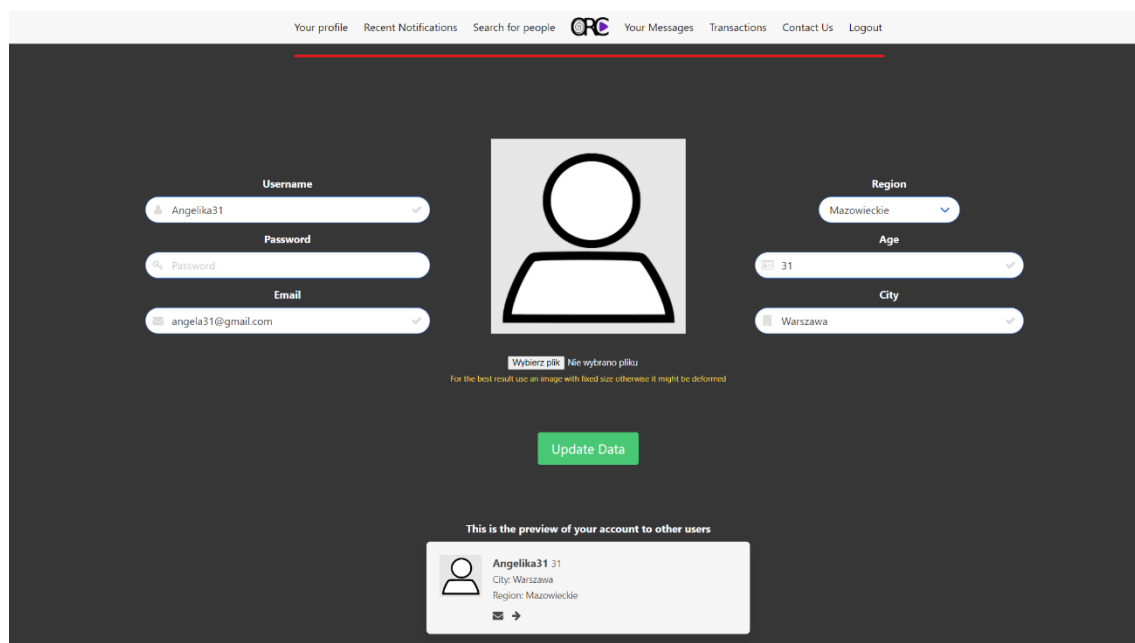
Ilustracja 17. Klasa definiująca zachowanie komponentu głównego.

Na powyższej ilustracji pokazano kod klasy definiującej zachowanie komponentu głównego. To właśnie tutaj zdefiniowana jest właściwość visible\_nav kluczowa do określania widoczności stopki oraz paska nawigacyjnego. Jest tutaj także metoda odpowiadająca za wylogowanie użytkownika. Usunięcie tokenu JWT jest kluczowe dla tego wydarzenia, ponieważ przy jego nieobecności nie jest możliwy dostęp do żadnej części systemu, nie licząc rejestracji oraz logowania. Ten mechanizm zapewnia dostęp do nawigacji tylko zalogowanemu użytkownikowi. Wyniki badań nad widocznością paska nawigacji oraz działaniem usługi Router we frameworku Angular pozwoliły osiągnąć pożądany przez autora rezultat.



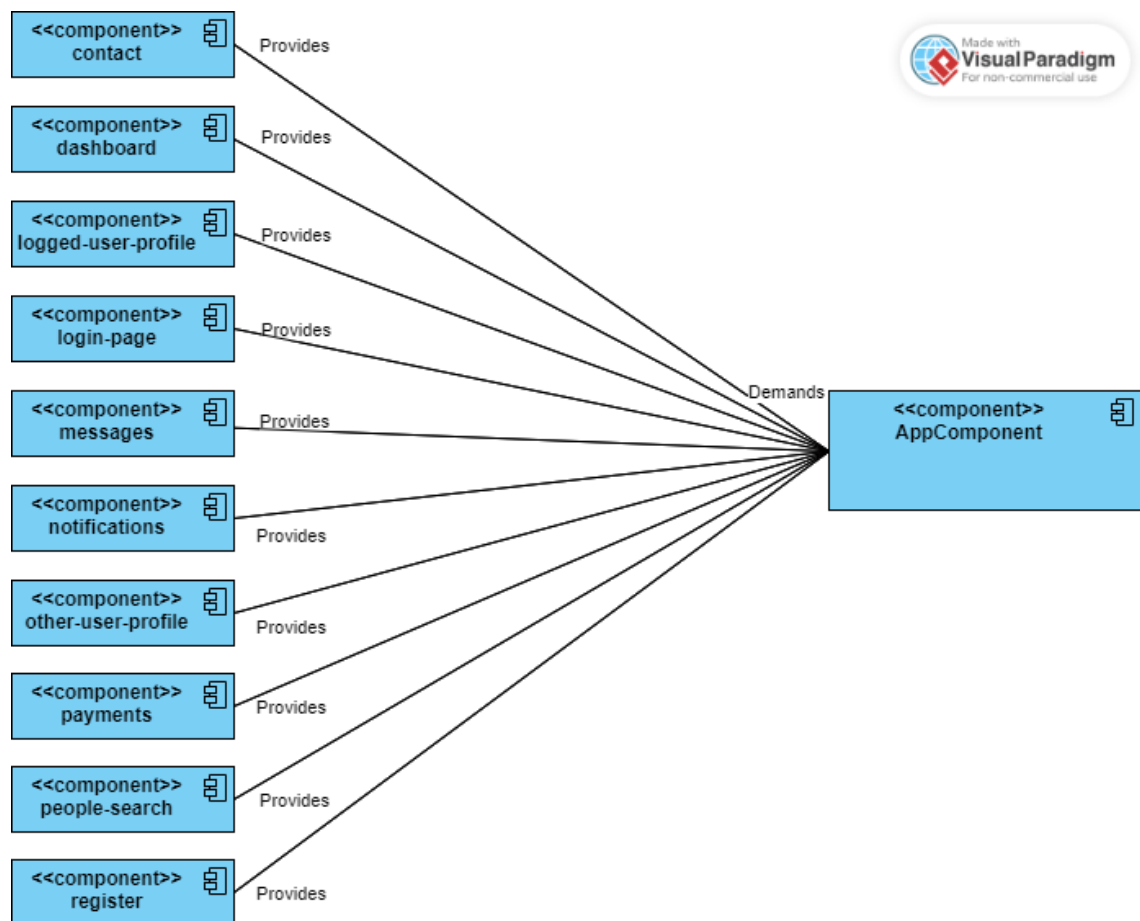
Ilustracja 18. Widok logowania do systemu.

Na ilustracji 18. pokazano ekran logowania do systemu. Zgodnie z założeniem widoczności paska nawigacji tylko dla użytkownika zalogowanego, nie jest on widoczny.



Ilustracja 19. Widoczny pasek nawigacji po zalogowaniu.

Na powyższej ilustracji pokazano widoczny pasek nawigacji po zalogowaniu. W przytoczonym przykładzie, widzimy widok edycji profilu zalogowanego użytkownika oraz podkreślony został pasek nawigacji widoczny ze względu na jego obecność.



Ilustracja 20. Diagram komponentów Front-End.

Na powyższej ilustracji pokazano diagram komponentów<sup>[18]</sup> Front-End. Pokazuje on wszystkie komponenty wchodzące w skład części Front-End systemu oraz zależności między nimi, a komponentem głównym. AppComponent może posłużyć się każdym z nich do wygenerowania kodu zamiast znacznika <router-outlet>, a każdy z komponentów dostarcza te dane do komponentu głównego.

## 2. INTERFEJS PROGRAMISTYCZNY APLIKACJI

W celu opracowania warstwy Back-End systemu autor wykorzystał technologię .NET. Jest to utworzona przez Microsoft platforma programistyczna<sup>[19]</sup>. Jej dokumentacja jest olbrzymia, a sama platforma jest bezpłatna oraz typu Open Source<sup>[20]</sup>. Autor wykorzystał język C# w celu realizacji warstwy Back-End na owej platformie tworząc Web API. Jest ono niezbędnym pośrednikiem między warstwą Front-End a bazą danych.

Odpowiada za podstawowe bezpieczeństwo systemu, generuje token JWT dla logującego się użytkownika oraz pozwala pobierać informacje z bazy danych. Zapobiega duplikacji danych w bazie danych oraz czyści ją z niepotrzebnych starych powiadomień użytkowników. Współpracuje z nią ORM wykorzystany przez autora – Entity Framework, który był niezbędnym elementem tworzenia bazy danych. Pozwolił on autorowi na utworzenie bazy danych z poziomu środowiska programistycznego oraz wedle potrzeby dokonuje translacji metod z kodu źródłowego na zapytania SQL wykonywane w bazie danych. Pozwala na sprawną pracę z bazą danych z poziomu środowiska programistycznego API.

### 2.1 Mechanizmy zabezpieczające system

Na początkowym etapie badań i prac nad API zbadano Cross-Origin Resource Sharing – w skrócie CORS. Jest to mechanizm, który zapewnia bezpieczny dostęp do zasobów z różnymi źródłami<sup>[21]</sup>. W przypadku pracy autora jest to element niezbędny do poprawnego skonfigurowania, gdyż API jak i Front-End działają na innych portach i bez odpowiednio skonfigurowanej polityki CORS zablokuje ona komunikację między nimi, właśnie z powodów innego źródła.

```
builder.Services.AddCors(options =>
{
    options.AddPolicy("CORSPolicy", policy =>
    {
        policy.WithOrigins("http://localhost:4200")
            .AllowAnyMethod()
            .AllowAnyHeader()
            .AllowCredentials();
    });
});
```

Ilustracja 21. Konfiguracja mechanizmu CORS w API.

Na powyższej ilustracji pokazano fragment kodu API, w którym konfigurowany jest mechanizm CORS. Jest on dodawany jako usługa, a dokładna konfiguracja zakłada umożliwienie komunikacji z aplikacją pod adresem <http://localhost:4200>. Taka

konfiguracja zapewnia odblokowanie komunikacji między API oraz warstwą Front-End systemu.

Poprawę bezpieczeństwa systemu zapewnia zastosowanie mechanizmu token JWT (Json Web Token). Jest to mechanizm, który autor zastosował w sposób następujący - użytkownik podczas logowania otrzymuje takowy token ważny przez 60 minut, a wszystkie punkty końcowe (tzw. Endpoint) API, nie licząc logowania i rejestracji, są objęte autoryzacją tokenu JWT, więc nie jest możliwe wykorzystanie ich bez ważnego tokenu. Zapewnia to, że tylko zalogowany użytkownik ma dostęp do pobierania danych przez API takich jak chociażby danych o jego profilu wyświetlanych w jednym z komponentów.

```
builder.Services.AddAuthentication(auth =>
{
    auth.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    auth.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    auth.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(jwt =>
{
    jwt.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters
    {
        /// <summary>Konfiguracja parametrów używanych do validacji tokenów</summary>
        ValidateIssuer = true,
        ValidateLifetime = true,
        ValidateAudience = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = builder.Configuration["Jwt:Issuer"],
        ValidAudience = builder.Configuration["Jwt:Audience"],
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secret))
    };
});
```

Ilustracja 22. Konfiguracja autoryzacji JWT.

Na powyższej ilustracji pokazano konfigurację autoryzacji JWT. Pierwsza część kodu to zdefiniowanie autoryzacji jako autoryzacja JWT, natomiast część druga jest konfiguracją konkretnych parametrów walidacji tokenu. Autor ustawił 4 weryfikacje oraz 3 inne parametry. Zadano weryfikowanie wydawcy tokenu, jego żywotność, jego odbiorcę oraz klucz podpisujący wydawcy. Pokazano także ustawienie wydawcy oraz odbiorcy, a także klucz podpisujący wydawcy.

```

1 odwołanie
public string GenerateToken(Users user)
{
    var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Secret"]));
    var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
    var claims = new[]
    {
        new Claim(ClaimTypes.NameIdentifier, user.Username),
        new Claim(ClaimTypes.Role, user.Role)
    };
    var token = new JwtSecurityToken(_configuration["Jwt:Issuer"],
        _configuration["Jwt:Audience"],
        claims,
        expires: DateTime.Now.AddMinutes(60),
        signingCredentials: credentials);

    return new JwtSecurityTokenHandler().WriteToken(token);
}

```

Ilustracja 23. Metoda generująca token JWT.

Na powyższej ilustracji pokazano metodę, która odpowiada za wygenerowanie konkretnego tokenu. Ustawiane są konkretne parametry takie jak klucz bezpieczeństwa oraz poświadczenie, a także twierdzenia. Na samym końcu, na podstawie wcześniejszych przesłanek generowany jest rzeczywisty token JWT, który jest zwracany po słowie kluczowym return.

```

[AllowAnonymous]
[HttpPost]
Odwołania: 0
public async Task<IActionResult> Login([FromBody] Users users)
{
    PasswordHasher<Users> passwordHasher = new();
    var Shared = new Shared(this._context, this._config);
    var user = await Shared.Authenticate(users);
    if (user != null)
    {
        var token = Shared.GenerateToken(user);
        return Ok(new { token, user });
    }

    return NotFound("user not found");
}

```

Ilustracja 24. Endpoint Logowania.

Na powyższej ilustracji pokazano Endpoint Login. Ukazanie tego fragmentu kodu w tym momencie pracy służy ukazaniu, gdzie w praktyce jest wykorzystywane generowanie tokenu JWT. Dzieje się to po autentykacji użytkownika, próbującego się zalogować. W przypadku poprawnych danych i możliwości zalogowania się do systemu takowy token jest generowany, przyznawany oraz zwracany do Front-End'u celem dalszego wykorzystywania go jak nagłówek autoryzacyjny w żądaniach HTTP.



Autor zastosował hashowanie haseł użytkowników. Zwiększa to bezpieczeństwo danych, gdyż nawet w przypadku wycieku danych, zahashowane hasła są bezużyteczne. API odpowiada za zabezpieczenie haseł w ten sposób wykorzystując klasę PasswordHasher.

```
PasswordHasher<Users> passwordHasher = new();
Users userToAdd = userCame;

var users = await _dbContext.Users.ToListAsync();
foreach(var user in users)
{
    if(user.Username == userToAdd.Username || user.Email == userToAdd.Email)
    {
        return NotFound("User already exists");
    }
}

userToAdd.Role = "NFUA";
userToAdd.Id = new Guid();
userToAdd.Password = passwordHasher.HashPassword(userToAdd, userToAdd.Password);
```

Ilustracja 25. Hashowanie hasła z wykorzystaniem klasy PasswordHasher.

Na powyższej ilustracji pokazano fragment endpoint'u, w którym ukazano utworzenie nowego obiektu klasy PasswordHasher z parametrem generycznym Users. Pokazano ustawienie hasła w obiekcie klasy Users, który zostanie dodany do bazy danych jako wynik reprezentacji w postaci zahashowanej hasła podanego przez użytkownika systemu.

```
1 odwołanie
public async Task<Users> Authenticate(Users userLogin)
{
    PasswordHasher<Users> passwordHasher = new();
    var currentUser = await _context.Users.FirstOrDefaultAsync(x => x.Email.ToLower() ==
        userLogin.Email.ToLower());
    if(currentUser != null)
    {
        if (passwordHasher.VerifyHashedPassword(userLogin, currentUser.Password, userLogin.Password) == PasswordVerificationResult.Success)
        {
            return currentUser;
        }
    }
    return null;
}
```

Ilustracja 26. Metoda uwierzytelniająca używająca odhashowywania haseł

Na powyższej ilustracji pokazano metodę uwierzytelniającą wykorzystującą odhashowywanie haseł. Wykorzystując tą samą klasę tj. PasswordHasher, metoda ta weryfikuje hasło podane przez użytkownika z tym zahashowanym. Następnie w postaci jednej z wartości obiektu typu enum zwraca wynik tej weryfikacji.

## 2.2 Wykorzystanie Entity Framework

W celu dostępu do danych z bazy danych z poziomu API autor wykorzystał Entity Framework. Jest to narzędzie do mapowania obiektowo-relacyjnego, które pozwala

zbudować wysokopoziomową warstwę dostępu do danych z różnych baz danych takich jak dla przykładu PostgreSQL lub MSSQL na platformie .NET<sup>[22]</sup>. Omawiane narzędzie oferuje takie funkcjonalności jak:

- Kwerendy LINQ
- Śledzenie zmian
- Aktualizowanie danych w bazie
- Migracje schematów bazodanowych

Entity Framework automatycznie tworzy zapytania SQL z kwerend LINQ oraz wykonuje je w bazie danych wedle potrzeby. Zapewnia bardzo sprawną pracę z bazą danych z poziomu API.

Kluczową częścią Entity Framework jest DbContext. Jest to klasa, której instancja reprezentuje sesję z bazą danych<sup>[23]</sup>, za pomocą której można chociażby wysłać zapytanie o dane lub zaktualizować dane w bazie. Autor musiał utworzyć klasę kontekstu swojej bazy danych, dziedziczącej po DbContext, aby utworzyć schemat swojej bazy danych, które następnie, dzięki migracji, utworzy pożądaną bazę danych.

```
namespace Orch_back_API.Entities
{
    Odwołania: 29
    public class MyJDBContext : DbContext
    {
        Odwołania: 0
        public MyJDBContext() { }
        Odwołania: 0
        public MyJDBContext(DbContextOptions<MyJDBContext> options) : base(options)
        {
        }

        Odwołania: 22
        public DbSet<Users> Users { get; set; }
        Odwołania: 5
        public DbSet<Messages> Messages { get; set; }
        Odwołania: 5
        public DbSet<Notifications> Notifications { get; set; }

        Odwołania: 0
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfigurationsFromAssembly(this.GetType().Assembly);
            base.OnModelCreating(modelBuilder);
        }
    }
}
```

Ilustracja 27. Kod klasy kontekstu bazy danych.

Na powyższej ilustracji pokazano kod klasy kontekstu bazy danych. Dziedziczy ona po klasie DbContext, posiada dwa konstruktory, trzy DbSet'y (każdy DbSet to odpowiednik

jednej tabeli w bazie danych, a parametr generyczny ukazuje klasę, która ma być encjami tej tabeli oraz której właściwości będą kolumnami) oraz przeciążenie metody `OnModelCreating`, w której aplikowana jest konfiguracja z lokalnego zasobnika.

```
builder.Services.AddDbContext<MyJDBContext>(options =>
{
    options.UseSqlServer("Server=(localdb)\\MSSQLLocalDB;Database=Orchard_INZDB;Trusted_Connection=True;");
    options.EnableSensitiveDataLogging();
});
```

Ilustracja 28. Dodanie DbContext jako usługi.

Na powyższej ilustracji pokazano dodanie do usług aplikacji (`IServiceCollection`) kontekstu bazy danych, którym jest klasa dziedzicząca po macierzystym `DbContext` zawierająca schemat bazy danych (Ilustracja 27.). Metoda `UseSqlServer` pozwala dodać `ConnectionString`. Jest to ciąg znaków, za pomocą którego Entity Framework będzie mógł nawiązać połączenie z odpowiednią bazą danych.

Parametry generyczne właściwości klas `DbSet` z ilustracji 27. to klasy, które zostaną encjami bazy danych, a ich właściwości kolumnami. Baza danych zawiera trzy `DbSet`'y reprezentujące:

- Użytkowników
- Powiadomienia
- Wiadomości

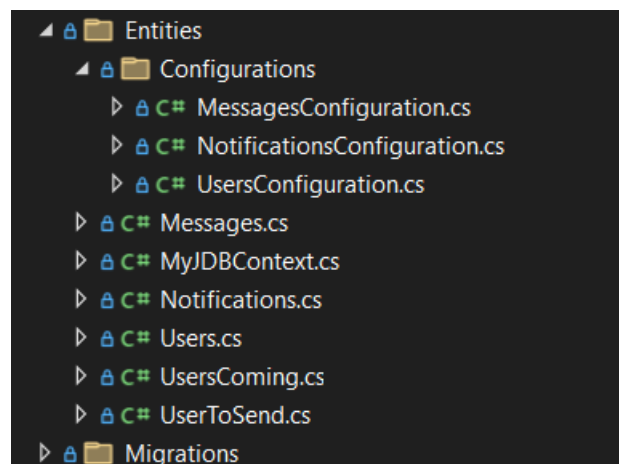
```
namespace Orch_back_API.Entities
{
    Odwołania: 18
    public class Notifications
    {
        Odwołania: 5
        public Guid? Id { get; set; }
        Odwołania: 3
        public string? Content { get; set; }
        Odwołania: 7
        public DateTime? SendDate { get; set; }
        Odwołania: 4
        public Guid? DeliveryId { get; set; }
        Odwołania: 5
        public Users? Author { get; set; }
        Odwołania: 4
        public Guid? AuthorId { get; set; }
    }
}
```

Ilustracja 29. Kod klasy `Notifications` będącej tabelą w bazie danych.

Na powyższej ilustracji pokazano kod klasy będącej tabelą w bazie danych. Każda właściwość, prócz właściwości `Author`, jest kolumną w bazie danych. Dwie ostatnie

właściwości to właściwości relacyjne. Odpowiadają za nawiązanie relacji z tabelą Users. Owa tabela oraz tabela Notifications są połączone relacją jeden do wielu, gdyż użytkownik może mieć wiele powiadomień, ale powiadomienie jest przypisane tylko do danego użytkownika. W taki sposób w Entity Framework w podejściu Code-First, wyrażana jest relacja między tabelami po stronie “wielu” tej relacji. Dokładne jej określenie nastąpi w konfiguracji jednej z tabel.

Pracując z Entity Framework autor postanowił zastosować osobny folder na konfigurację encji w lokalnym zasobniku (Assembly).



Ilustracja 30. Struktura plików zastosowanych przy Entity Framework.

Na powyższej ilustracji pokazano strukturę plików wykorzystanych przy pracy z Entity Framework (oraz dwie klasy pomocnicze). Konfiguracje encji znajdują się w folderze Configurations.

```
Odwołania: 0
public class NotificationsConfiguration : IEntityTypeConfiguration<Notifications>
{
    Odwołania: 0
    public void Configure(EntityTypeBuilder<Notifications> eb)
    {
        eb.Property(x => x.SendDate).HasDefaultValueSql("getutcddate()");

        eb.HasOne(c => c.Author)
            .WithMany(x => x.Notifications)
            .HasForeignKey(x => x.AuthorId)
            .OnDelete(DeleteBehavior.ClientSetNull);
    }
}
```

Ilustracja 31. Konfiguracja encji Notifications.

Na powyższej ilustracji pokazano konfigurację encji Notifications jako przykład konfiguracji z lokalnego zasobnika. Klasy konfiguracyjne muszą dziedziczyć po

interfejsie `IEntityTypeConfiguartion` z parametrem generycznym będącym klasą, która jest konfigurowaną tabelą. Metoda `Configure` zawiera całą konfigurację. Stosując parametr `'eb'` typu `EntityTypeBuilder` z parametrem generycznym, będącym klasą konfigurowaną, autor skonfigurował domyślną wartość kolumny `SendDate` dla każdego nowego rekordu oraz zdefiniował relację jeden do wielu między użytkownikiem a powiadomieniami. Na ilustracji 27. pokazane zostało zaaplikowanie wszystkich konfiguracji z tego zasobnika na encje (metoda: `ApplyConfigurationsFromAssembly`).

## 2.3 Architektura kontrollerów API

Tworząc warstwę Back-End systemu autor zastosował architekturę komponentów API. Polega ona na podzieleniu API na kontrollery, będące klasami odpowiedzialnymi za wyznaczone części systemu oraz obsługujące konkretne endpoint'y API, wykorzystujące wbudowane mechanizmy .NET (choćby kwerendy LINQ) oraz Entity Framework, w celu realizacji zadań w systemie. Większość endpointów całego API jest objęta autoryzacją JWT poprzez atrybut `[Authorize]`. Endpointy przyjmują postać metod asynchronicznych typu `Task` (służący do wykonania operacji asynchronicznej mogącej zwrócić wartość) z parametrem generycznym typu `IActionResult` pozwalającym na zwrócenie odpowiedzi na żądanie HTTP o kodzie OK (200) lub 404 (Not Found) z możliwością zwrócenia danych do Front-End'u.

```
[Authorize]
[Route("api/[controller]")]
[ApiController]
1 odwołanie
public class UsersController : ControllerBase
{
    private readonly MyJDBContext _context;
    Odwołania: 0
    public UsersController(MyJDBContext context)
    {
        this._context = context;
    }
}
```

Ilustracja 32. Fragment kodu controllera UsersController.

Na powyższej ilustracji pokazano część kodu controllera UsersController. Wszystkie jego endpointy są objęte autoryzacją JWT (atrybut `[Authorize]`), ścieżka do jego endpointów jest określona w atrybucie `Route`, a także atrybut `[ApiController]` definiuje klasę jako controller API. Powyższa ilustracja ukazuje także mechanizm dependency injection. Pozwala on na wstrzyknięcie niezbędnej temu controllerowi zależności, w tym wypadku przez konstruktor<sup>[24]</sup>, co pozwala na korzystanie z niej bez potrzeby tworzenia obiektu tej klasy wewnątrz controllera. Wstrzykiwany jest tutaj kontekst bazy danych. Za jego pomocą autor wykonywać będzie operacje bazodanowe pokazane w następnych ilustracjach.

```

[HttpPost]
[Route("getuserbyid")]
Odwolania: 0
public async Task<IActionResult> GetUserById([FromBody] UsersComing user)
{
    var id = user.Id;
    var userToReturn = await _context.Users.Where(eb => eb.Id.Equals(id)).FirstOrDefaultAsync();
    if(userToReturn != null)
    {
        return Ok(new { userToReturn });
    }
    else
    {
        return NotFound();
    }
}

```

Ilustracja 33. Endpoint 'getuserbyid' kontrolera Users.

Na powyższej ilustracji pokazano endpoint zwracający użytkownika z przekazanym w ciele żądania Id. Klasa UserComing to klasa pomocnicza, za pomocą, której autor czasem określa typ parametru przychodzącego. Ze względu na dopuszczanie wartości null jako wartość właściwości klasy UsersComing jako parametr odbierany jest obiekt tej klasy z wartością inną niż null tylko w właściwości 'Id', która potem jest wykorzystywana do asynchronicznej operacji bazodanowej. Z użyciem właściwości \_context dokonywane jest przeszukanie tabeli Users, aby znaleźć takiego użytkownika, którego Id będzie równe temu z parametru metody. Następnie w zależności od tego, czy szukany użytkownik istnieje w bazie danych czy nie, odpowiednia odpowiedź jest zwracana do warstwy Front-End systemu. Atrybut [FromBody] parametru określa pochodzenia parametru jako ciało żądania HTTP.

W celu automatyzacji utrzymania porządku w bazie danych autor posłużył się specjalnym kontrolerem porządkowym przedstawionym na poniższej ilustracji.

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Orch_back_API.Entities;

namespace Orch_back_API.Controllers
{
    [Authorize]
    [Route("api/[controller]")]
    [ApiController]
    1 odwołanie
    public class SystemCleaningController : ControllerBase
    {
        private readonly MyJDBContext _context;
        Odwołania: 0
        public SystemCleaningController(MyJDBContext context)
        {
            this._context = context;
        }

        [HttpPost]
        Odwołania: 0
        public async Task<IActionResult> CleanOldNotifications()
        {
            await _context.Notifications.Where(eb => eb.SendDate < DateTime.UtcNow.AddDays(-3)).ExecuteDeleteAsync();
            return Ok();
        }
    }
}

```

Ilustracja 34. Kontroler porządkowy.

Zasadą działania jest wywołanie tej metody przez żądanie HTTP przychodzące w momencie uruchomienia komponentu Dashboard w warstwie Front-End'u. Zapewnia to,

że zwyczajnie użytkując system, będzie on automatycznie czyścił bazę danych ze zbędnych starych powiadomień. Autor założył, że nie jest rozsądne, dla celów zachowania miejsca w bazie danych oraz skalowalności, przechowywanie danych o powiadomieniach starszych niż 3 dni, toteż są one usuwane automatycznie.

```
export class DashboardComponent implements OnInit{

    messagess : any;
    users : any;
    objectsToDisplay = new Array();
    MessageToSendForm: FormGroup;
    formDataMessage: FormData;
    currentUserToSendMessageTo : any;
    ethereum : any;

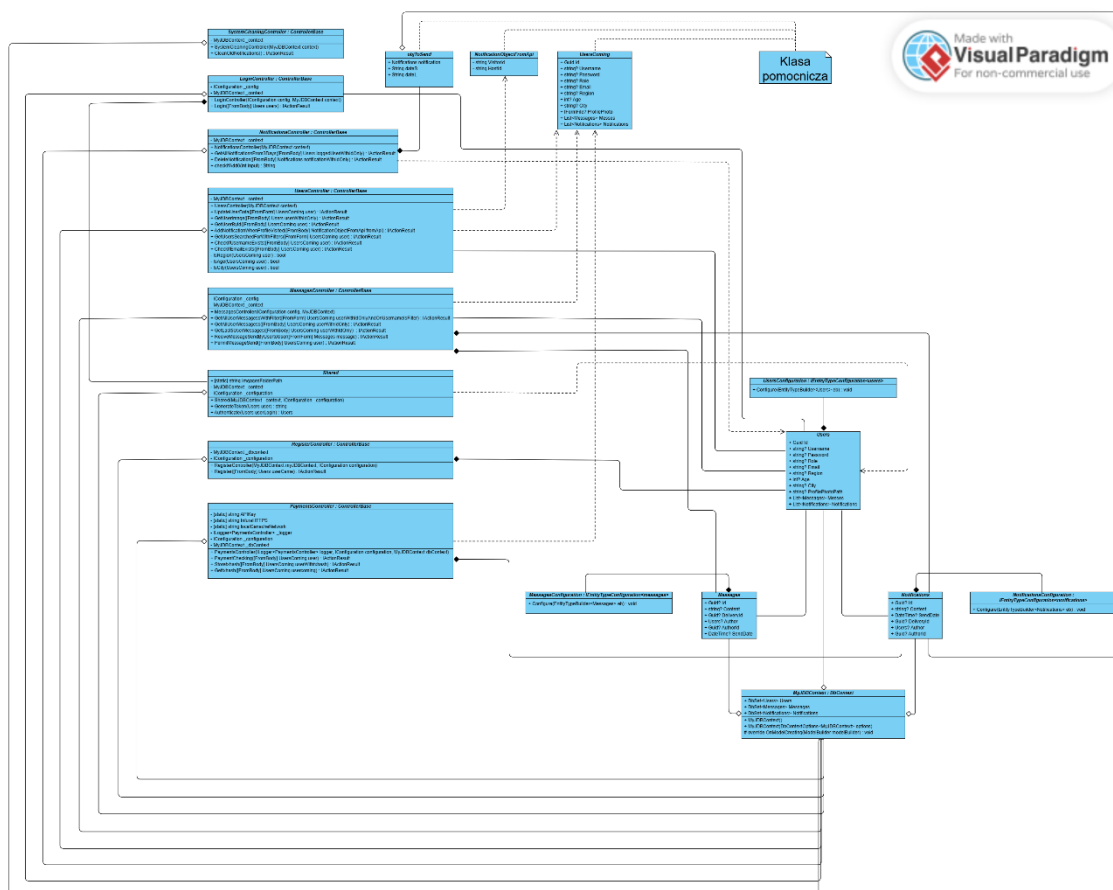
    constructor(private router1 : Router, private fb : Form
        appC.visible nav = true;
        apiComm.cleanExcessNotifications().subscribe();

        this.MessageToSendForm = this.fb.group({
            Message:"Type message here"
        })

        this.formDataMessage = new FormData();
    }
```

Ilustracja 35. Wysłanie żądania oczyszczenia bazy danych ze zbędnych powiadomień

Na powyższej ilustracji pokazano fragment klasy definiującej zachowanie komponentu Dashboard z warstwy Front-End oraz zaznaczył linię kodu odpowiedzialną za wysłanie żądania do API, które wywoła oczyszczenie bazy danych ze zbędnych powiadomień. Celem ukazania tej ilustracji jest pokazanie miejsca, w którym owe żądanie jest wysyłane - w konstruktorze komponentu Dashboard.



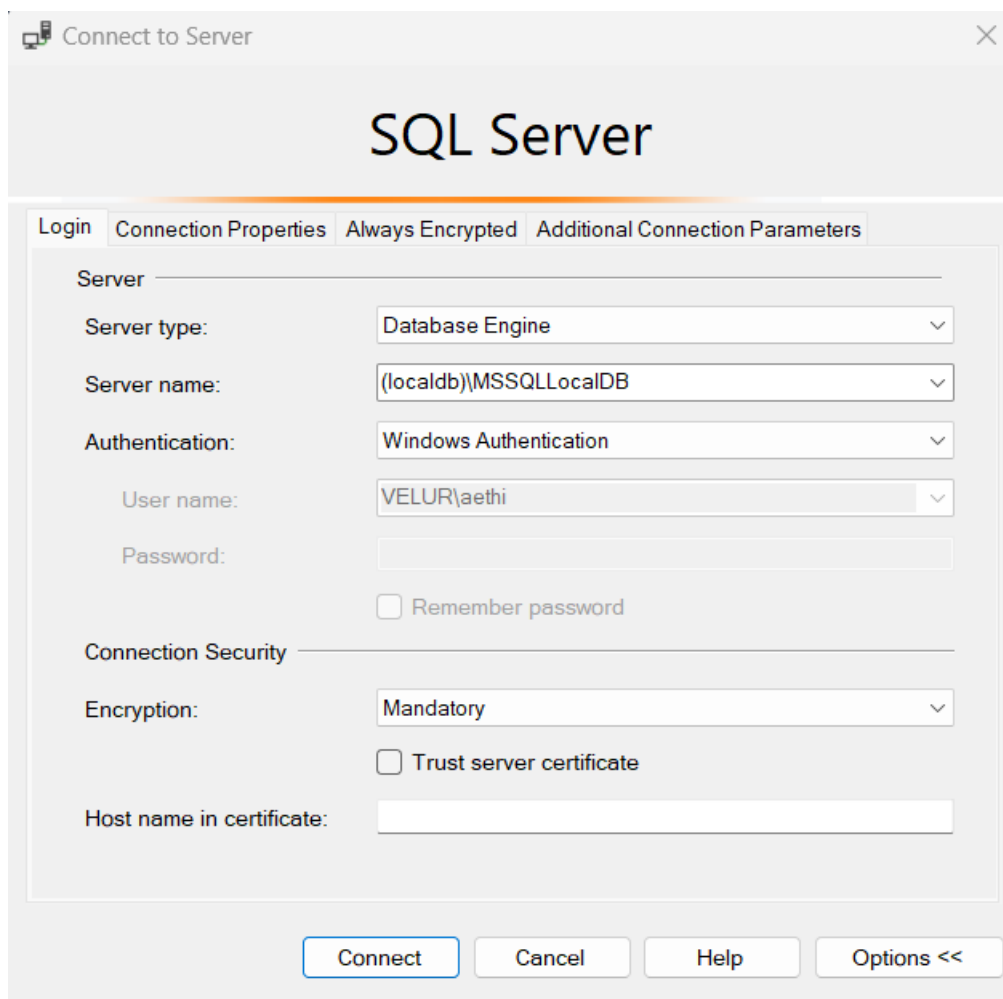
Ilustracja 36. Diagram klas API.

Na powyższej ilustracji pokazano diagram klas API. Pokazuje on zależności między klasami, które je tworzą. Notatka z treścią “Klasa pomocnicza” jest powiązana z klasami, które tylko pomagają w poprawnym funkcjonowaniu API, ale nie są kontrollerami, ani klasami, na podstawie których stworzone zostały tabele w bazie danych. Wszystkie klasy dziedziczące po klasie `ControlerBase` są kontrollerami API. Klasa `shared` służy jako klasa z elementami, które mogą być użyte przez inne klasy. Zarówno ona jak i kontrollery połączone są z klasą `MyJDBContext` relacją agregacji częściowej ze względu na mechanizm wstrzykiwania zależności, w którym to zależność `MyJDBContext` jest wstrzykiwana przez konstruktor do danej klasy<sup>[25]</sup>. Klasy reprezentujące encje bazy danych są połączone z klasami, które zawierają ich konfigurację agregacją całkowitą, ponieważ tworzą one całość - encja i jej konfiguracja.



### **3. BAZA DANYCH SYSTEMU**

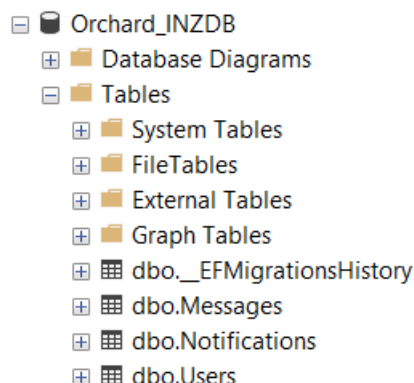
Autor przewidział, że system będzie wymagał integracji z bazą danych. Przechowywanie danych użytkowników, haseł, adresów mailowych, informacji o wiadomościach jak i powiadomieniach jest zadaniem, które autor zlecił systemowi zarządzania bazą danych MSSQL. Jest to wytworzony oraz utrzymywany przez Microsoft system zarządzania bazą danych z graficznym interfejsem<sup>[26]</sup>. Jako wariant języka SQL, wykorzystywany jest tutaj Transact-Sql rozszerzający język SQL o funkcjonalności, takie jak instrukcje warunkowe IF, pętlę WHILE, zmienne oraz synonimy<sup>[27]</sup>. Autor wykorzystał lokalną bazę danych, a środowisko użyte do zarządzania nią to SQL Server Management Studio 20.



Ilustracja 37. Logowanie do lokalnej bazy danych z poziomu środowiska SQL Server Management Studio 20.

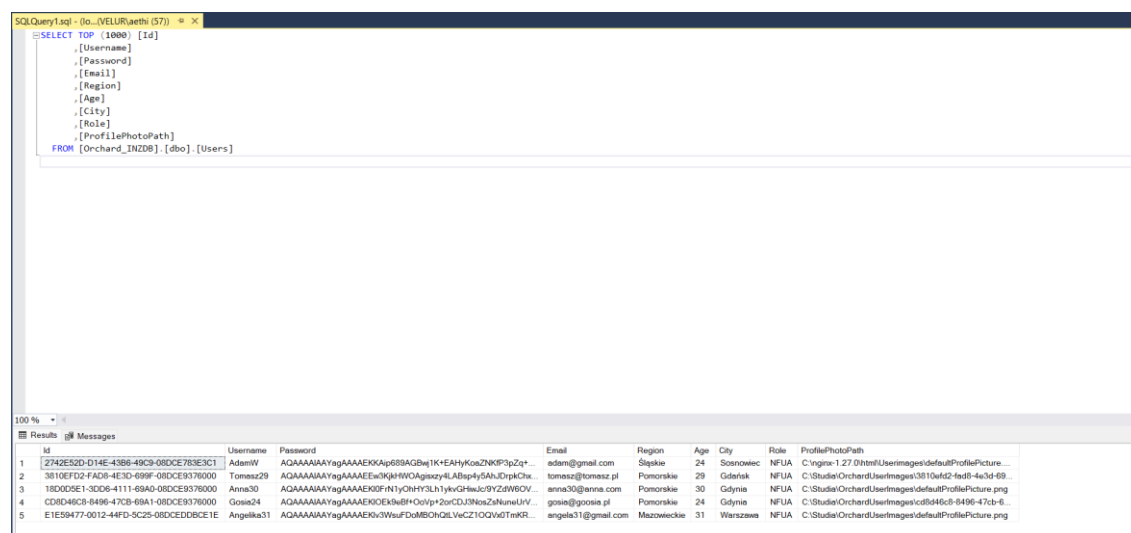
Na powyższej ilustracji pokazano ekran logowania się do lokalnej bazy danych z wykorzystaniem środowiska SQL Server Management Studio 20. Atrybut Server name zawiera łańcuch znaków, który pozwoli połączyć się z lokalną bazą danych – z tą samą, z którą łączy się API przez Entity Framework (ilustracja 28.).

Środowisko wykorzystane przez autora ukazuje strukturę bazy danych w postaci graficznej.



Ilustracja 38. Struktura bazy danych.

Na powyższej ilustracji pokazano strukturę bazy danych. Zgodnie z łańcuchem połączeniowym ukazanym na ilustracji 28., baza danych jest nazwana zgodnie z nazwą przekazaną przez ten łańcuch, a tabele nazywają się dokładnie tak, jak nazywał je autor definiując DbSet'y w klasie dziedziczącej po kontekście bazy danych (ilustracja 27.). Dodany został jedynie przedrostek "dbo.".

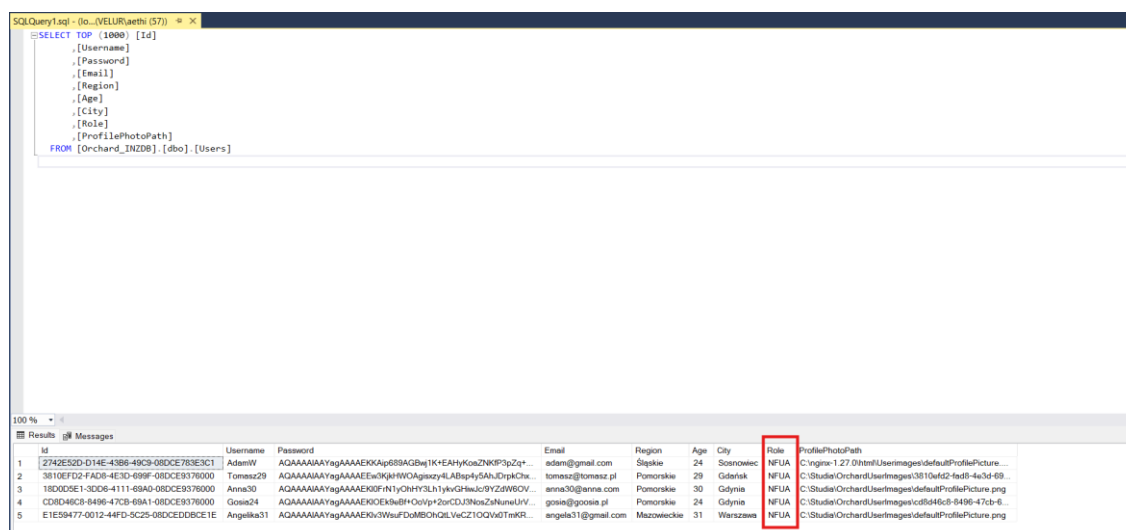


Ilustracja 39. Widok danych z tabeli Users.

Na powyższej ilustracji pokazano wynik działania zapytania SQL widocznego na powyższej ilustracji, ukazującego pierwszych 1000 użytkowników (ze względu na to, że ich ilość nie przekracza 1000 ukazano tylko 5 - są to wszyscy użytkownicy systemu). Hasła użytkowników nie są widoczne ze względu na zahashowanie.

## 4. TRANSAKCJE KRYPTOWALUTOWE

Użytkownicy systemu, od momentu utworzenia konta, nie mają pełnego dostępu do funkcjonalności systemu. Oznacza to, że dostęp do niektórych z nich będzie dla nich zamknięty. Są to powiadomienia (informujące chociażby o odwiedzinach innego użytkownika na profilu odbiorcy powiadomienia), specjalna ikonka pojawiająca się przy wieku osoby, podczas gdy inni wyszukają jej profil przez wyszukiwarkę oraz zniesienie limitu 5 wiadomości dziennie (domyślnie użytkownik nie może wysłać większej liczby wiadomości). Te ograniczenia nie uniemożliwiają korzystania z serwisu, ale zdjęte mogą poprawić komfort korzystania z systemu oraz podnieść szanse na spełnienie celu, w jakim jest on wykorzystywany jak chociażby poznanie partnera życiowego. Za przechowywanie informacji o pełnym dostępie do systemu odpowiada kolumna Role tabeli Users.



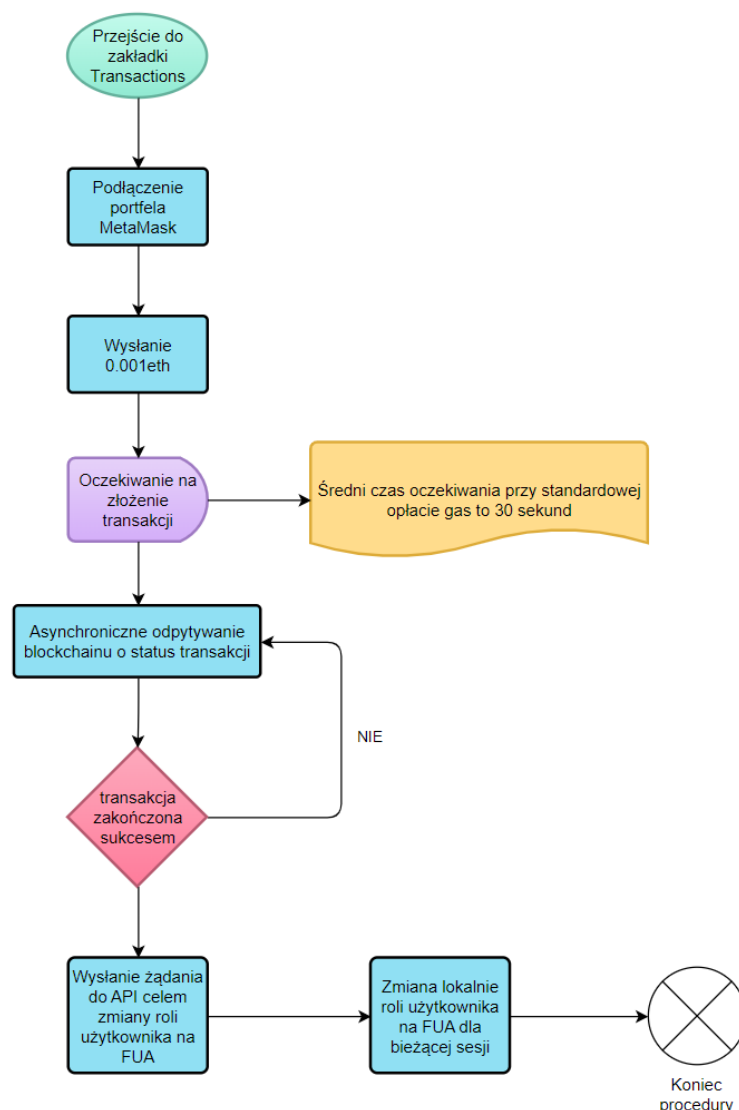
```
SELECT TOP (1000) [Id]
,[Username]
,[Password]
,[Email]
,[Region]
,[Age]
,[City]
,[Role]
,[ProfilePhotoPath]
FROM [Orchard_INZDB].[dbo].[Users]
```

Id	Username	Password	Email	Region	Age	City	Role	ProfilePhotoPath
1	Adam	AQAAAAAAYagAAAAEKXAp8BAGBj1K*E4HyKoaZNMf5pZt...	adam@gmail.com	Śląskie	24	Sosnowiec	NFUA	C:\Inetpub\wwwroot\aspnet\images\defaultProfilePicture...
2	Tomasz	AQAAAAAAYagAAAAEKXAp8BAGBj1K*E4HyKoaZNMf5pZt...	tomasz@tomek.pl	Pomorskie	29	Gdańsk	NFUA	C:\Inetpub\wwwroot\aspnet\images\defaultProfilePicture...
3	Anna	AQAAAAAAYagAAAAEKXAp8BAGBj1K*E4HyKoaZNMf5pZt...	anna30@anna.com	Pomorskie	30	Gdynia	NFUA	C:\Inetpub\wwwroot\aspnet\images\defaultProfilePicture...
4	Gosia	AQAAAAAAYagAAAAEKXAp8BAGBj1K*E4HyKoaZNMf5pZt...	gosia@gosia.pl	Pomorskie	24	Gdynia	NFUA	C:\Inetpub\wwwroot\aspnet\images\defaultProfilePicture...
5	Angelika	AQAAAAAAYagAAAAEKXAp8BAGBj1K*E4HyKoaZNMf5pZt...	angelika31@gmail.com	Mazowieckie	31	Warszawa	NFUA	C:\Inetpub\wwwroot\aspnet\images\defaultProfilePicture...

Ilustracja 40. Zaznaczenie kolumny Role odpowiadającej za przechowywanie danych o pełnym dostępie użytkowników do systemu.

Na powyższej ilustracji pokazano zaznaczoną kolumnę Role tabeli Users odpowiadającą za przechowywanie danych o pełnym dostępie użytkowników do systemu. Jeżeli wartość w kolumnie byłaby równa “NFUA” - skrót od Not Full Access User, oznacza to, że dany użytkownik nie ma pełnego dostępu do systemu. Jeżeli wartość w kolumnie byłaby równa “FUA” - skrót od Full Access User, oznacza to, że dany użytkownik ma pełny dostęp do systemu.

Celem zdjęcia omówionych przez autora ograniczeń należy zapłacić przy pomocy kryptowaluty ethereum cenę 0.001eth portfelem MetaMask. Po zatwierdzeniu transakcji na blockchainie ethereum pełny dostęp zostanie przyznany.



Ilustracja 41. Schemat procedury zakupu pełnego dostępu.

Na powyższej ilustracji pokazano schemat procedury zakupu pełnego dostępu do systemu. W poniższych podrozdziałach omówione zostaną poszczególne kroki tej procedury.

#### 4.1 Przygotowanie portfela MetaMask

Portfel MetaMask jest wirtualnym portfelem kryptowalutowym. Oznacza to, że użytkownik wchodzi w posiadanie pary klucza publicznego (Public Key) oraz klucza prywatnego (Private Key). Klucz publiczny jest używany jako widoczny adres portfela. Z jego wykorzystaniem można odbierać transakcje wysyłane do własnego portfela. Jest on widoczny na blockchainie dla wszystkich użytkowników. Klucz prywatny służy do podpisywania transakcji. Niemożliwe jest potwierdzenie transakcji w sieci, jeżeli nie jest ona podpisana kluczem prywatnym wysyłającego. Podłączenie portfela do strony

odbywa się za pomocą MetaMaskSDK. Jest to biblioteka, która pozwala wpieść w tworzone oprogramowanie takie funkcjonalności jak chociażby:

- Podłączenie portfela MetaMask
- Wysłanie transakcji kryptowalutowej
- Sprawdzanie statusu transakcji na blockchainie

Autor rozpoczął proces od zainstalowania MetaMaskSDK używając npm. Wykorzystał w tym celu polecenie “npm i @metamask/sdk”<sup>[28]</sup>. Następnie, gdy odpowiedni moduł został już dodany, autor zaimportował go do klas, które jego użycia wymagały.

```
import { MetaMaskSDK } from '@metamask/sdk';  
import { Router } from '@angular/router';  
import { catchError, map, throwError } from 'rxjs';  
  
@Component({  
  selector: 'app-payments',  
  standalone: true,  
  imports: [CommonModule],  
  templateUrl: './payments.component.html',  
  styleUrls: ['./payments.component.css']  
})  
export class PaymentsComponent implements OnInit{
```

Ilustracja 42. Część kodu klasy definiującej zachowanie komponentu odpowiedzialnego za płatności z zaznaczonym importem MetaMaskSDK.

Na powyższej ilustracji pokazano fragment kodu klasy definiującej zachowanie komponentu odpowiedzialnego za płatności oraz zaznaczył konkretną linię kodu, która odpowiada za import omawianego SDK. Od tego momentu możliwe jest posłużenie się tym modulem w kodzie klasy.

```
const MMSDK = new MetaMaskSDK({
  dappMetadata: {
    name: "Orchard",
    url: window.location.href,
  },
  infuraAPIKey: "b8af0de6aa8e4d8aae4902937a7386ff",
});
setTimeout(() => {
  MMSDK.init().then(() => {
    this.ethereum = MMSDK.getProvider();
  });
}, 0);
```

Ilustracja 43. Zainstancjonowanie MetaMaskSDK.

Na powyższej ilustracji pokazano fragment kodu odpowiadający za zainstancjonowanie MetaMaskSDK. Owa ilustracja wspomina również o infuraAPIKey. Jest to parametr, w którym autor przekazał istotną część klucza API Infura. Infura jest zewnętrznym węzłem sieci udostępniając (za darmo jeden lub opłacając subskrypcję więcej) klucze API, za pomocą których można wykonywać operacje na blockchainie przez węzeł Infura. W celu uzyskania dostępu do blockchainu trzeba posłużyć się węzłem sieci. Węzeł, w tej terminologii, jest urządzeniem, na którym obecny jest cały blockchain oraz które współpracuje z innymi węzłami odbierając informacje o nowych blokach wydobywanych/wykuwanych (nazewnictwo zależne od algorytmu konsensusu danej sieci) przez innych lub siebie. Utrzymanie węzła sieci wymaga zasobów sprzętowych. W przybliżeniu, do poprawnego funkcjonowania węzła oraz uniknięcia ciężkich obciążeń systemu zalecane jest posiadać urządzenie posiadające<sup>[29]</sup>:

- 4 bądź 8 GB pamięci RAM
- Dysk 2TB SSD
- Procesor Intel 7 generacji lub wyższej

Autor zdecydował się, celem rozwiązania wybranego sobie problemu inżynierskiego, na użycie zewnętrznego węzła sieci.

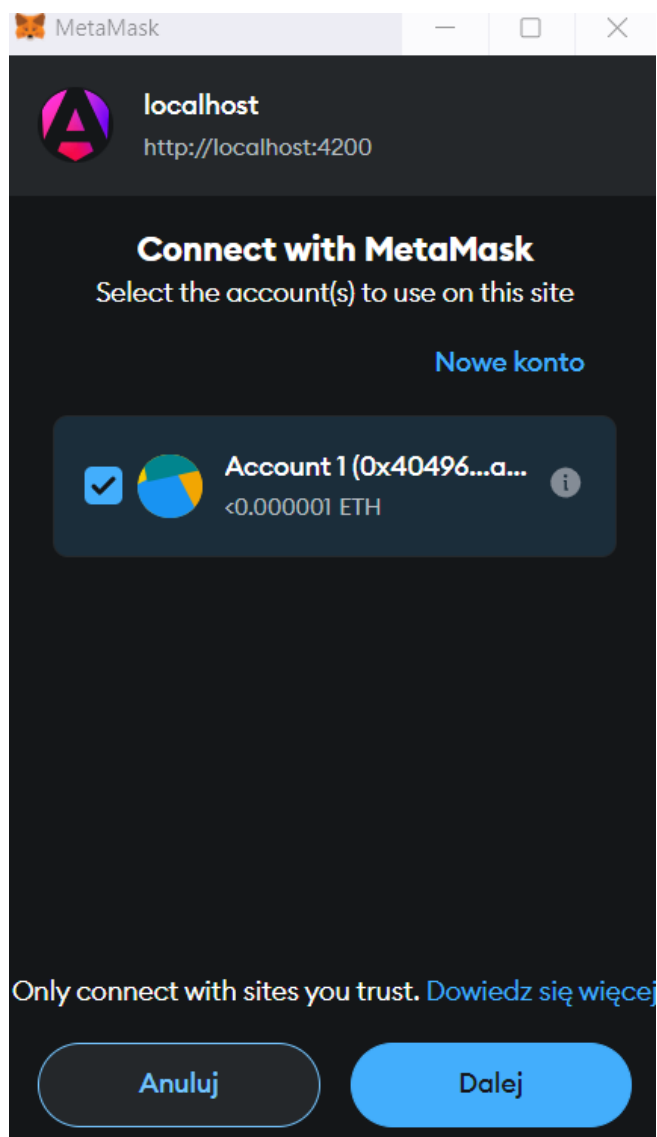
W dalszej części widzimy uzyskanie wartości właściwości ethereum. Odnosząc się w późniejszych etapach, do niej autor będzie wysyłał żądania do sieci Ethereum z jej użyciem. Nim nastąpi przypisanie poprzez instrukcję MMSDK.getProvider() wyczekane jest zainicjalizowanie obiektu MMSDK dzięki instrukcjom init() oraz then(). Zapewnia to, że nie zdarzy się sytuacja, w której aplikacja będzie chciała przypisać wartość do właściwości ethereum z użyciem instrukcji MMSDK.getProvider() zanim obiekt MMSDK zostanie zainicjalizowany.

```
async ConnectMetaMask(){
  await this.ethereum.request({ method: 'eth_requestAccounts' }).then((accounts : any) => {
    this.account = accounts[0];
    console.log(this.account);
  });
};
```

Ilustracja 44. Podłączenie portfela MetaMask.

Na powyższej ilustracji pokazano kod metody odpowiadającej za podłączenie portfela MetaMask do systemu. Używając określonej wcześniej właściwości `ethereum` autor wysłał żądanie przez API portfela MetaMask, które daje użytkownikowi możliwość podłączenia swojego portfela MetaMask. Jest to żądanie typu JSON-RPC (JSON-RPC jest protokołem zdalnego wywoływania procedur, który jako dane transportowe wykorzystuje obiekty JSON<sup>[30]</sup>)





Ilustracja 45. Efekt wywołania metody ConnectMetaMask().

Na powyższej ilustracji pokazano efekt wywołania metody ConnectMetaMask(). Sprawia ona, że okno podłączenia portfela pojawi się i zapyta użytkownika czy na pewno chce się połączyć z tą stroną. Jest to konieczny krok, ponieważ zgodnie z dokumentacją, w celu wykonania transakcji niezbędne są uprawnienia od właściciela portfela na interakcje z jego kontem<sup>[31]</sup>.

## 4.2 Obsługa transakcji

Celem dokonania transakcji autor wysyła inne żądanie z parametrami transakcji. Również jest to żądanie, jak wymienione wyżej, JSON-RPC.

```

let transactionDetails = {
  to: '0x40496e2d5a48779a2721e6effa5be7a7a9caa151',
  from: this.account,
  value: '38D7EA4C68000'
};

await this.ethereum.request({method: 'eth_sendTransaction', params:[transactionDetails]}).then((txhash : any) => {
  this.txhash = txhash;
})

```

Ilustracja 46. Żądanie JSON-RPC wysyłające transakcję z parametrami.

Na powyższej ilustracji pokazano fragment kodu metody, odpowiadający za zdefiniowanie parametrów transakcji oraz wysłanie żądania wysłania transakcji. Jej parametry to<sup>[32]</sup>:

- to – klucz publiczny odbiorcy transakcji
- from – adres konta podłączonego portfela (uzyskane dzięki żądaniu eth\_requestAccounts na ilustracji 44.)
- value - ilość eth przesłana do odbiorcy przez nadawcę

Wartość parametru value zostanie teraz przez autora dokładniej omówiona. Musi być to wartość szesnastkową<sup>[33]</sup>. Przesyłając eth do odbiorcy, celem zdefiniowania jego ilości, niezbędne jest posłużenie się mniejszą jednostką – Wei<sup>[34]</sup>. Jest on najmniejszą jednostką ether'u. 1 eth jest równy  $10^{18}$  Wei<sup>[35]</sup>. Autor posłużył się więc konwerterem jednostek, aby uzyskać interesującą go ilość Wei.

The screenshot shows the 'Ethereum Unit Converter' website. The main heading is 'Simple Unit Converter'. Below it, a paragraph explains the tool's purpose. The converter interface includes three input fields: 'Wei' with the value '1000000000000000', 'Gwei' with '1000000', and 'Ether' with '0.001'. At the bottom, it displays 'Total Price \$ 2.62' and '(2616.53 \$ Per Ether)'. The footer mentions 'Made in 2017' and provides an email contact: 'contact@eth-converter.com'.

Ilustracja 47. Konwersja jednostek<sup>[36]</sup> Ether na Wei.

Na powyższej ilustracji pokazano wyżej wymienioną konwersję jednostek. Następnym krokiem była zamiana liczby określającą ilość Wei na wartość szesnastkową.

### Decimal to Hexadecimal converter

From	To
Decimal	Hexadecimal

Enter decimal number

1000000000000000	10
------------------	----

**= Convert**   **× Reset**   **↕ Swap**

Hex number (13 digits)

38D7EA4C68000	16
---------------	----

Ilustracja 48. Konwersja wartości dziesiętnej na szesnastkową<sup>[37]</sup>.

Na powyższej ilustracji pokazano wyżej wspomnianą konwersję. Otrzymana wartość szesnastkowa zostanie wpisana jako wartość właściwości value w parametrach transakcji. Ostatnim krokiem, zaprezentowanym na ilustracji 46. jest wysłanie żądania wysłania transakcji. Całość zawarta jest w metodzie asynchronicznej więc dodane słowo kluczowe await sprawi, że wykonanie dalszej części kodu zostanie wstrzymane do momentu zakończenia obsługi żądania. Ethereum wykorzystuje algorytm konsensusu Proof of Stake. Wybiera on z pseudolosowego procesu jeden węzeł, który będzie odpowiadał za weryfikację następnego bloku<sup>[38]</sup>. Sprawdzi on poprawność transakcji, które mają być w nim zapisane, złoży swój “podpis” pod nimi i zatwierdzi je. W zamian za to, wszystkie opłaty gas (drobna opłata, którą płacą użytkownicy za wykonywanie transakcji na blockchainie oraz wykonywanie Smart Contract’ów), związane z tymi transakcjami trafią do weryfikatora jako nagroda. Oczekiwanie na wysłanie transakcji potrwa, przy obecnie zgodnej z ceną rynkową opłacie gas, około 30 sekund. Uiszczając mniejszą opłatę gas potrwa to dłużej lub jeżeli większą, krócej. Jako odpowiedź odbierany jest hash transakcji będący jej unikalnym identyfikatorem<sup>[39]</sup>.

Następnym krokiem jest odpytywanie blockchainu, przez żądanie JSON-RPC (`eth_getTransactionReceipt`) w pętli o status transakcji. W przypadku jej potwierdzenia pełny dostęp zostanie przyznany obecnie zalogowanemu użytkownikowi oraz wysłane żądanie do API celem zmiany wartości w kolumnie Role w bazie danych, dla obecnie zalogowanego użytkownika, na "FUA".

```
async WaitForTransactionConfirmation(txhash : string){
    let loop = () => {
        return this.ethereum.request({method: 'eth_getTransactionReceipt', params:[txhash]}).then((ev: any) => {
            if(ev != null){
                return 'confirmed';
            }
            else{
                return loop();
            }
        })
    }
    return await loop();
}
```

Ilustracja 49. Asynchroniczna metoda odpytująca blockchain o status transakcji.

Na powyższej ilustracji pokazano kod asynchronicznej metody odpytującej blockchain o dane transakcji o hashu podanym jako parametr metody. Jej działanie polega na zapętleniu wysyłania żądania JSON-RPC o dane transakcji. Jeżeli zwracaną odpowiedzią jest null znaczy to, że nie została ona jeszcze zatwierdzona. Jeżeli odpowiedzią będzie obiekt JSON, który nie będzie nullem oznacza to, że transakcja została zatwierdzona<sup>[40]</sup>. W przeciwnym zwrócenia odpowiedzi null instrukcje z pętli wykonają się jeszcze raz i będą się wykonywać ponownie do czasu, aż transakcja zostanie zatwierdzona (całość potrwa w przybliżeniu pół sekundy).

```
this.apiconn.PaymentStoretxhash(this.loggeduserdata.LoggedUser.Id, this.txhash).subscribe( //to słu
{
    next: (result) => {
        this.WaitForTransactionConfirmation(this.txhash).then(ev => { //Ponieważ metoda odpytywania j
            if(ev == 'confirmed'){
                this.apiconn.PaymentStoretxhash(this.loggeduserdata.LoggedUser.Id, "FUA").subscribe();
                this.loggeduserdata.LoggedUserRole = "FUA";
                this.loggeduserdata.LoggedUser.Role = "FUA";
                this.loggedUserRole = "FUA";
            }
        })
    }
});
this.loggeduserdata.LoggedUserRole = this.txhash;
this.loggeduserdata.LoggedUser.Role = this.txhash;
this.loggedUserRole = this.txhash;
```

Ilustracja 50. Operacje finalizujące proces transakcji.

Na powyższej ilustracji pokazano operacje finalizujące proces transakcji. Do API wysyłane jest żądanie celem zmiany w bazie danych wartości kolumny Role zalogowanego użytkownika na wartość hashu transakcji. Pełni to funkcję

zabezpieczającą. W dalszej części odbędzie się oczekiwanie na zatwierdzenie transakcji. Jeżeli zostałoby ono przerwane, to użytkownik zostałby z hashem transakcji w bazie danych, lecz bez pełnego dostępu, mimo że za niego zapłacił. Implementując interfejs OnInit, autor umieścił w metodzie onInit (umożliwia zlecenie dodatkowych instrukcji do zrealizowania przy inicjalizacji komponentu) fragment kodu odpowiadający za wznowienie odpytywania blockchainu celem uzyskania zatwierdzenia transakcji, jeżeli nastąpiłaby sytuacja, w której w kolumnie Role w bazie danych znajdowałby się hash transakcji (ilustracja 51.).

```
this.apiconn.PaymentGettxhash(this.loggeduserdata.LoggedUser.Id)
  .pipe(
    catchError(error => {
      return throwError(() => new Error("Error occured"));
    }),
    map((response) => {
      const data = response.body;
      return(data);
    })
  )
  .subscribe({
    next: (result) => {
      const role = result.data.role;
      if(role != "FUA" && role != "NFUA"){
        this.WaitForTransactionConfirmation(role).then(ev => {
          if(ev == 'confirmed'){
            this.apiconn.PaymentStoretxhash(this.loggeduserdata.LoggedUser.Id, "FUA").subscribe(); //GIVE USER FULL ACCESS IF PAYMENT HAS BEED CONFIRMED
            this.loggeduserdata.LoggedUserRole = "FUA";
            this.loggeduserdata.LoggedUser.Role = "FUA";
            this.loggedUserRole = "FUA";
          }
        })
      }
    },
    error: (error) => {
      console.error('API Error:', error);
    }
  });
```

Ilustracja 51. Fragment kodu metody onInit wznowiający odpytywanie blockchainu o dane transakcji.

Na powyższej ilustracji pokazano fragment kodu metody onInit wznowiający odpytywanie blockchainu o dane transakcji z hashu umieszczonego w bazie danych. Wykorzystany mechanizm jest bardzo podobny do mechanizmu z ilustracji 50.

```

[HttpPost]
[Route("storetxhash")]
Odwolania: 0
public async Task<IActionResult> Storetxhash([FromBody] UsersComing userWithtxhash)
{
    await _dbContext.Users.Where(eb => eb.Id == userWithtxhash.Id).ExecuteUpdateAsync(setters => setters
        .SetProperty(eb => eb.Role, userWithtxhash.City));
    if(userWithtxhash.City == "FUA")
    {
        Notifications notification = new Notifications();
        notification.Id = Guid.NewGuid();
        var user = await _dbContext.Users.Where(eb => eb.Id == userWithtxhash.Id).FirstOrDefaultAsync();
        notification.Author = null;
        notification.Content = "SYSTEM: Your transaction has been completed. Your are now a Full Access User.";
        notification.SendDate = DateTime.Now;
        notification.DeliveryId = user!.Id;
        notification.AuthorId = null;
        await _dbContext.AddAsync(notification);
        await _dbContext.SaveChangesAsync();
    }
    return Ok();
}

```

Ilustracja 52. Endpoint API obsługujący wprowadzenie do bazy danych zmian w kolumnie Role oraz ewentualnego wysłania powiadomienia w przypadku nadania pełnego dostępu.

Na powyższej ilustracji pokazano kod wykonywany w API jako obsługa żądania kierowana do tego Endpointu. Dokonywana jest zmiana w kolumnie Role, na wartość z właściwości City obiektu z ciała żądania, a także w przypadku nadawania pełnego dostępu wysyłane jest specjalne powiadomienie systemowe informujące użytkownika o nadaniu mu pełnego dostępu do funkcji systemu.

## 5. ANALIZA PODOBNYCH ROZWIĄZAŃ

Aplikacja Badoo jest systemem podobnym do Orchard. Realizuje ona funkcjonalność tworzenia profilu użytkownika oraz pozwala na wymianę wiadomości między użytkownikami. Za dodatkową opłatą daje możliwość skorzystania z dodatkowych funkcjonalności systemu takich jak podgląd przez kogo profil użytkownika został polubiony, korzystanie z polubień bez limitu czy wyłączenie reklam.

Autor wyodrębnił następujące zalety systemu Badoo:

- Przejrzysty i czytelny interfejs
- Szybkie transakcje za pośrednictwem systemu PayPal
- Wysoce dostosowywalny profil
- Weryfikacja profilu będąca również widzialna dla innych użytkowników
- Dodawanie filmów w miejsce zdjęcia profilowego

Zdaniem autora na szczególną uwagę zasługuje weryfikacja profilu. Czyni to Badoo bardziej bezpiecznym oraz zaufanym miejscem. Gwarantuje, że inni użytkownicy nie muszą martwić się o inną tożsamość osoby, z którą nawiązują kontakt, a ikona białego haczyka na niebieskim tle wyróżnia profil na tle innych.

Wyodrębniono następujące wady Badoo:

- Niezrozumiały do nowego użytkownika system kredytów
- Niezwykle drogi status „Premium Plus”
- Brak jakichkolwiek informacji o przybliżonym czasie oczekiwania na pomoc w przypadku wysłania wiadomości przy korzystaniu z pomocy Badoo

Zdaniem autora Badoo jest systemem, który pozwoli na realizację celu jakim jest poznanie innych osób oraz potencjalnego partnera lub partnerki. Jego czysty i przejrzysty interfejs zapewni sprawność podejmowanych działań, a wysoce dostosowywalny profil pozwoli bardzo dokładnie opisać swoje cechy oraz wartości, które mają być dla innych wyeksponowane. Lekką frustracją może napełnić użytkownika początkowe niezrozumienie systemu kredytów, lecz w miarę upływu czasu ustąpi. Zdaniem autora na naganę zasługuje wygórowana cena pakietu „Premium Plus”. Nie oferuje on wiele więcej korzyści niż pakiet „Premium”, a jego koszty są nieporównywalnie większe. Jest to pakiet, w którym cena jest nieproporcjonalnie duża w stosunku do usług i korzyści, do których dostęp otrzymujemy.

Systemem podobnym do Orchard jest Tinder. Jest to bezapelacyjnie najpopularniejsza aplikacja randkowa. Daje on możliwość nawiązania kontaktu z osobami, które użytkownik uważa za atrakcyjne z wzajemnością poprzez sprytny system matchy. Wydarzenie to polega na tym, że gdy dwójka użytkowników przesunie w prawo swoje profile podczas wyszukiwania oznajmiając tym samym, że podoba im się profil osoby, którą przesunęli w prawo otrzymają oni wtedy powiadomienie o matchu oraz będą mogli rozpocząć wymianę wiadomości ze sobą. W zamian za wykupienie jednego z trzech pakietów subskrypcji Tinder oferuje dodatkowe funkcjonalności takie jak nielimitowane polubienia, jeden darmowy Boost na miesiąc lub pogląd kto polubił profil użytkownika.

Autor wyodrębnił następujące zalety systemu Tinder:

- Dostosowanie trybu ciemnego oraz jasnego do ustawień urządzenia
- Możliwość filtrowania osób przez pryzmat zainteresowań lub celów
- Klarownie i zrozumiale okazane korzyści płynące z opłacenia konkretnej subskrypcji
- Weryfikacja profilu będąca widoczna dla innych użytkowników
- Dobór stylizacji interfejsu w trybie nocnym
- Brak niechcianych wiadomości od osób, którymi nie jest zainteresowany użytkownik
- Możliwość powrotu w przypadku omylnego przesunięcia w złą stronę osoby podczas wyszukiwania i oceniania profili

Na wyjątkową uwagę zasługuje funkcjonalność Explore. Pozwala ona na znalezienie osób, z którymi użytkownik dzieli cel użytkowania aplikacji, zainteresowania lub nawyki. Pozwala to na olbrzymią oszczędność czasu, a także zwiększa szanse na osiągnięcie przez użytkowników ich celów. Interfejs tej usługi jest przejrzysty i intuicyjny.

Wyodrębniono następujące wady Tinder:

- Brak możliwości wyboru religii jako filtr podczas wyszukiwania
- Często narzucająca się opcja opłacenia subskrypcji z dodatkowymi funkcjonalnościami

Subiektywna ocena autora dla systemu Tinder jest korzystna i pozytywna. Uważa On, że jest to przemyślane skonstruowana aplikacja, która dobrze sprawdzi się w roli narzędzia do poznawania nowych osób. Jego stylowy interfejs doda przyjemności korzystaniu z niego, a wygodny system przesuwania w lewo i prawo uczyni poszukiwania nowych osób proste i szybkie. Pod kątem funkcjonalności oraz architektury systemu ciężko dopatrzeć się jakichkolwiek wad. Zasadniczą niedogodnością jest jednak brak wyboru religii jako parametru filtrowania przy wyszukiwaniu osób. Dla osób wyznających konkretną religię i szukających osób, które także ją wyznają, przez tą wadę, lepszym narzędziem będzie Badoo, które taką funkcjonalność oferuje.



Aplikacja Singlowanie.pl jest podobnym rozwiązaniem do Orchard. Jest to katolicki portal randkowy. Realizuje on funkcjonalności tworzenia konta użytkownika oraz pozwala na wyszukiwanie profili innych użytkowników. Wyszukiwanie wzbogacone jest tu o wiele filtrów, które znacznie przyspieszą znalezienie profilu odpowiedniej osoby. Za dodatkową opłatą daje możliwość skorzystania z wielu dodatkowych funkcjonalności takich jak wysyłanie wiadomości do innych użytkowników, przeglądanie pełnego profilu innego użytkownika, podgląd kto odwiedził lub polubił profil użytkownika, a także specjalną ikonę na zdjęciu, gdy profil użytkownika z wykupionym dostępem zostanie wyszukany.

Autor wyodrębnił następujące zalety systemu Singlowanie.pl:

- Bardzo niska cena wykupienia dostępu do dodatkowych funkcjonalności w porównaniu do dwóch wyżej wymienionych systemów
- Zwiększenie prawdopodobieństwa natrafienia na osobę o podobnym światopoglądzie przez wzgląd na katolickie nastawienie aplikacji
- Przejrzysty i pozbawiony zbędnego skomplikowania interfejs dający łatwy dostęp do wszystkich potrzebnych funkcjonalności
- Płatność jednorazowa w miejsce modelu subskrypcyjnego

Na szczególną uwagę zasługuje tu bardzo niska cena wykupu dostępu premium. Gwarantuje on olbrzymią ilość dodatkowych funkcjonalności oraz w porównaniu do dwóch poprzednich systemów kosztuje niewiele. System Singlowanie.pl będzie świetnym narzędziem do poznawania nowych ludzi dla osób o katolickim światopoglądzie oraz nie posiadających wielkiego kapitału pieniężnego.

Wyodrębniono następujące wady Singlowanie.pl:

- Niepotrzebna funkcjonalność ‘Aktywności’ pokazująca aktywność wszystkich użytkowników systemu
- Odmowa dostępu do wielu podstawowych funkcjonalności dla użytkowników bez statusu premium
- Brak jakichkolwiek informacji o przybliżonym czasie oczekiwania na informacje zwrotne przy próbie kontaktu z biurem
- Niespójność w opisie korzyści płynących z zakupu dostępu premium (w owych korzyściach istnieje zapis mówiący o możliwości wyszukiwania innych użytkowników z kontem premium, lecz ta funkcjonalność jest dostępna również na użytkowników podstawowych)
- Niepotrzebna funkcjonalność ‘Popularne miasta’ dublująca filtr ‘Miasto’ z wyszukiwania użytkowników

Autor stanowczo potępia brak dostępu do absolutnie podstawowych funkcjonalności takich jak przegląd pełnego profilu użytkownika oraz wysyłanie wiadomości do innych osób dla użytkowników podstawowych. Wymusza to zakup dostępu premium celem normalnego korzystania z systemu w miejsce subtelnej zachęcania.

W ogólnej ocenie stwierdzono, że Singlowanie.pl jest dobrym narzędziem do poznawania nowych osób oferując tani dostęp do płatnych dogodności oraz urzekając prostotą interfejsu. Nie jest on pozbawiony drobnych wad oraz jednej w ocenie autora kardynalnej.

## ZAKOŃCZENIE

Autor rozwiązał zadany problem inżynierski tworząc system nazwany Orchard (w skrócie ORC). Zaprojektowano warstwę Front-End z użyciem frameworku Angular 17 oraz Bulma, a warstwę Back-End z użyciem platformy .NET. Wykorzystano bazę danych MSSQL. Celem realizacji transakcji kryptowalutowych wykorzystano portfel kryptowalutowy MetaMask, MetaMaskSDK, Infura Ethereum Node oraz żądania JSON-RPC.

Orchard umożliwia poznawanie innych osób online, a także, za pomocą kryptowalut, umożliwia wykupienie dostępu do dodatkowych funkcjonalności takich jak powiadomienia, specjalna ikonka przy wieku użytkownika, gdy jego profil zostanie wyszukany oraz zniesienie limitu 5 wiadomości dziennie. Orchard ma znaczenie w rzeczywistym świecie umożliwiając poznawanie innych osób przez internet, oferując płatność jednorazową, przez co wyróżnia się na tle innych aplikacji tego typu działających w modelu subskrypcyjnym, a także może pomóc w popularyzacji płatności kryptowalutowych oraz technologii z nimi związanych, które w czasie pisania tej pracy nie są w Polsce popularne.

## BIBLIOGRAFIA

1. Osoby wnoszące wkład, *Angular (web framework)*, [https://en.wikipedia.org/wiki/Angular\\_\(web\\_framework\)](https://en.wikipedia.org/wiki/Angular_(web_framework)), [dostęp: 24.10.2024]
2. Google, *Built-in directives*, <https://angular.dev/guide/directives>, [dostęp: 24.10.2024]
3. J. Thomas, *Bulma*, <https://bulma.io/>, [dostęp: 24.10.2024]
4. Google, *Angular components overview*, <https://v17.angular.io/guide/component-overview>, [dostęp: 24.10.2024]
5. Google, *Testing*, <https://angular.dev/guide/testing>, [dostęp: 24.10.2024]
6. Google, *Understanding binding*, <https://v17.angular.io/guide/binding-overview>, [dostęp: 24.10.2024]
7. Google, *Angular change detection and runtime optimization*, <https://v17.angular.io/guide/change-detection>, [dostęp: 24.10.2024]
8. Google, *HttpClient*, <https://v17.angular.io/api/common/http/HttpClient>, [dostęp: 24.10.2024]
9. Google, *provideHttpClient*, <https://v17.angular.io/api/common/http/provideHttpClient>, [dostęp: 24.10.2024]
10. Google, *@angular/router*, <https://v17.angular.io/api/router>, [dostęp: 24.10.2024]
11. Optiq, *Stack Overflow answer*, <https://stackoverflow.com/questions/77669269/angular-services-export-class>, [dostęp: 24.10.2024]
12. Monsterlessons Academy, *Angular Guard - How to Use Functional Router Guard*, <https://www.youtube.com/watch?v=Yc93IvrouxY>, [dostęp: 24.10.2024]
13. Google, *Understanding dependency injection*, <https://v17.angular.io/guide/dependency-injection>, [dostęp: 24.10.2024]
14. S. Vyawahare, *Understanding Model, Class, and Interface in Angular*, <https://medium.com/@snehalv.2010/understanding-model-class-and-interface-in-angular-953e9fe668c0>, [dostęp: 24.10.2024]
15. Google, *NgIf*, <https://angular.dev/api/common/NgIf>, [dostęp: 24.10.2024]
16. Visual Paradigm, *Visual Paradigm Online - aplikacja*, <https://online.visual-paradigm.com/pl/>, [dostęp: 24.10.2024]
17. Microsoft, *.NET: bezpłatnie. Wiele platform. Open source.*, <https://learn.microsoft.com/pl-pl/dotnet/>, [dostęp: 24.10.2024]
18. J. Elastyczny, *Czym jest CORS i jak działa?*, <https://dhosting.pl/pomoc/baza-wiedzy/czym-jest-cors-i-jak-dziala/>, [dostęp: 24.10.2024]
19. Google, *Entity Framework documentation hub*, <https://learn.microsoft.com/en-us/ef/>, [dostęp: 25.10.2024]
20. P. Łukasiewicz, *EF Core - DbContext*, <https://www.plukasiewicz.net/EFCore/EFCoreDbContext>, [dostęp: 25.10.2024]

21. P. Łukasiewicz, *C# - Dependency Inversion Principle, Inversion of Control, Dependency Injection*, [https://www.plukasiewicz.net/Artykuly/DIP\\_IoC\\_DI](https://www.plukasiewicz.net/Artykuly/DIP_IoC_DI), [dostęp: 25.10.2024]
22. Contributors, *Download SQL Server Management Studio (SSMS)*, <https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16>, [dostęp: 25.10.2024]
23. Google, *Wprowadzenie do programowania języka Transact-SQL*, <https://learn.microsoft.com/pl-pl/training/modules/get-started-transact-sql-programming/>, [dostęp: 25.10.2024]
24. METAMASK, *Use MetaMask SDK with other web frameworks*, <https://docs.metamask.io/wallet/connect/metamask-sdk/javascript/other-web-frameworks/>, [dostęp: 25.10.2024]
25. ethereum, *run-a-node*, <https://ethereum.org/en/run-a-node/>, [dostęp: 25.10.2024]
26. Contributors, *JSON-RPC*, <https://pl.wikipedia.org/wiki/JSON-RPC>, [dostęp: 25.10.2024]
27. METAMASK, *eth\_sendTransaction*, [https://docs.metamask.io/wallet/reference/json-rpc-methods/eth\\_sendtransaction/](https://docs.metamask.io/wallet/reference/json-rpc-methods/eth_sendtransaction/), [dostęp: 25.10.2024]
28. Red Stapler, *Metamask Javascript API Tutorial Send Transaction Button*, <https://www.youtube.com/watch?v=2m8Ww24X6Yw&t=253s>, [dostęp: 25.10.2024]
29. THE INVESTOPEDIA TEAM, *Wei: Definition in Cryptocurrency, How It Works, and History*, <https://www.investopedia.com/terms/w/wei.asp>, [dostęp: 25.10.2024]
30. Wykorzystany konwerter: <https://eth-converter.com/>, [dostęp: 25.10.2024]
31. Wykorzystany konwerter: <https://www.rapidtables.com/convert/number/decimal-to-hex.html>, [dostęp: 25.10.2024]
32. Slance, *What is Proof of Stake - Explained in Detail (Animation)*, <https://www.youtube.com/watch?v=YudpU58uYuM>, [dostęp: 25.10.2024]
33. coinbase, *What is a transaction hash/hash ID?*, <https://help.coinbase.com/en/coinbase/getting-started/crypto-education/what-is-a-transaction-hash-hash-id>, [dostęp: 25.10.2024]
34. p-programowanie.pl, *Diagramy klas UML*, <https://www.p-programowanie.pl/modelowanie/diagramy-klas-uml>, [dostęp: 28.10.2024]

## PRZYPISY

- 
- [1]. Osoby wnoszące wkład, *Angular (web framework)*, [https://en.wikipedia.org/wiki/Angular\\_\(web\\_framework\)](https://en.wikipedia.org/wiki/Angular_(web_framework)), [dostęp: 24.10.2024]
- [2]. Google, *Built-in directives*, <https://angular.dev/guide/directives>, [dostęp: 24.10.2024]
- [3]. J. Thomas, *Bulma*, <https://bulma.io/>, [dostęp: 24.10.2024]
- [4]. Google, *Angular components overview*, <https://v17.angular.io/guide/component-overview>, [dostęp: 24.10.2024]
- [5]. Google, *Angular components overview*, <https://v17.angular.io/guide/component-overview>, [dostęp: 24.10.2024]
- [6]. Google, *Angular components overview*, <https://v17.angular.io/guide/component-overview>, [dostęp: 24.10.2024]
- [7]. Google, *Testing*, <https://angular.dev/guide/testing>, [dostęp: 24.10.2024]
- [8]. Google, *Understanding binding*, <https://v17.angular.io/guide/binding-overview>, [dostęp: 24.10.2024]
- [9]. Google, *Angular change detection and runtime optimization*, <https://v17.angular.io/guide/change-detection>, [dostęp: 24.10.2024]
- [10]. Google, *HttpClient*, <https://v17.angular.io/api/common/http/HttpClient>, [dostęp: 24.10.2024]
- [11]. Google, *provideHttpClient*, <https://v17.angular.io/api/common/http/provideHttpClient>, [dostęp: 24.10.2024]
- [12]. Google, *@angular/router*, <https://v17.angular.io/api/router>, [dostęp: 24.10.2024]
- [13]. Optiq, *Stack Overflow answer*, <https://stackoverflow.com/questions/77669269/angular-services-export-class>, [dostęp: 24.10.2024]
- [14]. Monsterlessons Academy, *Angular Guard - How to Use Functional Router Guard*, <https://www.youtube.com/watch?v=Yc93lvrouxY>, [dostęp: 24.10.2024]
- [15]. Google, *Understanding dependency injection*, <https://v17.angular.io/guide/dependency-injection>, [dostęp: 24.10.2024]
- [16]. S. Vyawahare, *Understanding Model, Class, and Interface in Angular*, <https://medium.com/@snehalv.2010/understanding-model-class-and-interface-in-angular-953e9fe668c0>, [dostęp: 24.10.2024]
- [17]. Google, *NgIf*, <https://angular.dev/api/common/NgIf>, [dostęp: 24.10.2024]
- [18]. Visual Paradigm, *Visual Paradigm Online - aplikacja*, <https://online.visual-paradigm.com/pl/>, [dostęp: 24.10.2024]
- [19]. Microsoft, *.NET: bezpłatnie. Wiele platform. Open source.*, <https://learn.microsoft.com/pl-pl/dotnet/>, [dostęp: 24.10.2024]
- [20]. Microsoft, *.NET: bezpłatnie. Wiele platform. Open source.*, <https://learn.microsoft.com/pl-pl/dotnet/>, [dostęp: 24.10.2024]
- [21]. J. Elastyczny, *Czym jest CORS i jak działa?*, <https://dhosting.pl/pomoc/baza-wiedzy/czym-jest-cors-i-jak-dziala/>, [dostęp: 24.10.2024]
- [22]. Google, *Entity Framework documentation hub*, <https://learn.microsoft.com/en-us/ef/>, [dostęp: 25.10.2024]
- [23]. P. Łukasiewicz, *EF Core - DbContext*, <https://www.plukasiewicz.net/EFCore/EFCoreDbContext>, [dostęp: 25.10.2024]
- [24]. P. Łukasiewicz, *C# - Dependency Inversion Principle, Inversion of Control, Dependency Injection*, [https://www.plukasiewicz.net/Artykuly/DIP\\_IoC\\_DI](https://www.plukasiewicz.net/Artykuly/DIP_IoC_DI), [dostęp: 25.10.2024]
- [25]. p-programowanie.pl, *Diagramy klas UML*, <https://www.p-programowanie.pl/modelowanie/diagramy-klas-uml>, [dostęp: 28.10.2024]
- [26]. Contributors, *Download SQL Server Management Studio (SSMS)*, <https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16>, [dostęp: 25.10.2024]
- [27]. Google, *Wprowadzenie do programowania języka Transact-SQL*, <https://learn.microsoft.com/pl-pl/training/modules/get-started-transact-sql-programming/>, [dostęp: 25.10.2024]
- [28]. METAMASK, *Use MetaMask SDK with other web frameworks*, <https://docs.metamask.io/wallet/connect/metamask-sdk/javascript/other-web-frameworks/>, [dostęp: 25.10.2024]
- [29]. ethereum, *run-a-node*, <https://ethereum.org/en/run-a-node/>, [dostęp: 25.10.2024]
- [30]. Contributors, *JSON-RPC*, <https://pl.wikipedia.org/wiki/JSON-RPC>, [dostęp: 25.10.2024]
- [31]. METAMASK, *eth\_sendTransaction*, [https://docs.metamask.io/wallet/reference/json-rpc-methods/eth\\_sendtransaction/](https://docs.metamask.io/wallet/reference/json-rpc-methods/eth_sendtransaction/), [dostęp: 25.10.2024]

- 
- [32]. Red Stapler, *Metamask Javascript API Tutorial Send Transaction Button*, <https://www.youtube.com/watch?v=2m8Ww24X6Yw&t=253s>, [dostęp: 25.10.2024]
- [33]. Red Stapler, *Metamask Javascript API Tutorial Send Transaction Button*, <https://www.youtube.com/watch?v=2m8Ww24X6Yw&t=253s>, [dostęp: 25.10.2024]
- [34]. Red Stapler, *Metamask Javascript API Tutorial Send Transaction Button*, <https://www.youtube.com/watch?v=2m8Ww24X6Yw&t=253s>, [dostęp: 25.10.2024]
- [35]. THE INVESTOPEDIA TEAM, *Wei: Definition in Cryptocurrency, How It Works, and History*, <https://www.investopedia.com/terms/w/wei.asp>, [dostęp 25.10.2024]
- [36]. Wykorzystany konwerter: <https://eth-converter.com/>, [dostęp: 25.10.2024]
- [37]. Wykorzystany konwerter: <https://www.rapidtables.com/convert/number/decimal-to-hex.html>, [dostęp: 25.10.2024]
- [38]. Slance, *What is Proof of Stake - Explained in Detail (Animation)*, <https://www.youtube.com/watch?v=YudpU58uYuM>, [dostęp: 25.10.2024]
- [39]. coinbase, *What is a transaction hash/hash ID?*, <https://help.coinbase.com/en/coinbase/getting-started/crypto-education/what-is-a-transaction-hash-hash-id>, [dostęp: 25.10.2024]
- [40]. Red Stapler, *Metamask Javascript API Tutorial Send Transaction Button*, <https://www.youtube.com/watch?v=2m8Ww24X6Yw>, [dostęp: 25.10.2024]