Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Elektryczny

Łukasz Aniszewski

nr albumu: 36705
kierunek studiów: Automatyka i Robotyka
forma studiów: studia stacjonarne

APLIKACJA NA SMARTPHONE PREZENTUJĄCA PARAMETRY REJSU JEDNOSTKI PŁYWAJĄCEJ

SMARTPHONE APPLICATION THAT PRESENTS THE PARAMETERS OF THE VESSEL'S VOYAGE

Praca dyplomowa inżynierska

napisana pod kierunkiem:

dr inż. Robert Krupiński

Katedra Przetwarzania Sygnałów i Inżynierii Multimedialnej

Data wydania tematu pracy: 28.02.2020 r.

Data dopuszczenia pracy do egzaminu: .....

Szczecin, 2020

OŚWIADCZENIE
AUTORA PRACY DYPLOMOWEJ

Oświadczam, że praca dyplomowa inżynierska pn.
„Aplikacja na smartphone prezentująca parametry rejsu jednostki pływającej”
napisana pod kierunkiem:
dr inż. Robert Krupiński
jest w całości moim samodzielnym autorskim opracowaniem, sporządzonym przy wykorzystaniu wykazanej w pracy literatury przedmiotu i materiałów źródłowych.
Złożona w Dziekanacie Wydziału Elektrycznego treść mojej pracy dyplomowej w formie elektronicznej jest zgodna z treścią w formie pisemnej.

Oświadczam ponadto, że złożona w Dziekanacie praca dyplomowa, ani jej fragmenty nie były wcześniej przedmiotem procedur procesu dyplomowania związanych z uzyskaniem tytułu zawodowego w uczelniach wyższych.

.....
podpis dyplomanta

Szczecin, dn. ..... r.

Streszczenie pracy

Celem pracy były stworzenie aplikacji prezentującej aktualne parametry jednostki pływającej. Zakres wykonanych prac obejmował przegląd aktualnie stosowanych rozwiązań, zapoznanie się i opis protokołu NMEA (National Marine Electronics Associa-tion) oraz część oprogramowania aplikacji mobilnej. Dla testów skorzystano z dostęp-nego w sieci symulatora jednostki wysyłającej dane zakodowane protokołem NMEA. Głównymi z napotkanych problemów były – ilość oraz dokumentacja zdań protokołu NMEA, przygotowanie oprogramowania pod łatwy rozwój o wsparcie dla kolejnych zdań. Podczas wyboru technologii kierowano się głównie osobistym doświadczeniem oraz umiejętnościami.

Słowa kluczowe

słowo kluczowe 1, słowo kluczowe 2, ... maksymalnie 5 słów kluczowych

Abstract

The aim ofthe work was to create a mobile application for presenting actual parameters ofthe vessel's voyage. The scope of work included an overview of actually usedso-lutions, acquiring the knowledge about NMEA (National Marine Electronics Associa-tion)protocol and describing it, creating software for mobile application. Fortesting purposes, NMEA Simulator (available on the internet) which sends NMEA encodedda-ta was used. The main encountered issues were – the amount and documentationof NMEA protocol sentences, preparing the software for easy expansion by additional-sentences. The choice of the technological stack was mainly based on personalexpe-rience and skills.

Keywords

first keyword, second keyword, ... max. 5 keywords

Spis treści

Wykaz ważniejszych oznaczeń i skrótów

FFT	Fast Fourier Transform — Szybka Transformata Fouriera
IFFT	Inverse Fast Fourier Transform — Odwrotna Szybka Transformata Fouriera
$H(S)$	Transmitancja
$h(t)$	Okno czasowe
α	Współczynnik rozszerzalności termicznej

Wprowadzenie

Komunikacja między urządzeniami elektronicznymi jest obecnie standardem. Zależnie od obszaru działań wspomnianych urządzeń stosowane są odmienne protokoły. Jed-nostki pływające, których dotyczy ta praca korzystają ze standardu *NMEA*. Brak jednak narzędzi pozwalających w łatwy sposób przedstawić człowiekowi dane zakodowane tym protokołem. Obecnie aplikacje mobilne stanowią ogromną część rynku oprogramowania. Niemal każdy posiada smartphone i korzysta z aplikacji dla systemów takich jak *Android* czy *iOS*. W kolejnych rozdziałach został przedstawiony proces tworzenia takiej aplikacji. Równie dużą, a nawet większą, rolę odgrywa obecnie komunikacja sieciowa, większość znanych nam stron i aplikacji bazuje na architekturze *REST*, popularność zyskuje *GraphQL*, a do określonych zadań stosuje się protokoły takie jak websocket. Właśnie ten ostatni został wykorzystany w pracy, o czym dowiemy się w kolejnych rozdziałach.

Cel pracy

Celem pracy jest stworzenie aplikacji mobilnej prezentującej aktualne parametry jednostki pływającej.

Zakres pracy

Przegląd aktualnie stosowanych rozwiązań w zakresie parametrów jednostek pływających oraz metod ich prezentacji, zapoznanie się i opisanie standardu dla protokołu *NMEA* (National Marine Electronics Association) oraz stworzenie aplikacji mobilnej na Smartphone wraz z GUI (Graphical User Interface) prezentującej aktualne parametry jednostki pływającej z wykorzystaniem sieci *WiFi* jednostki pływającej. Celem dodatkowym jest przetestowanie aplikacji w warunkach rzeczywistych na jednostce pływającej.

ROZDZIAŁ 1

Protokół NMEA.

Komunikacja między morskimi urządzeniami elektronicznymi wykorzystuje protokół *NMEA*. Wyróżniamy dwie wersje tego protokołu – *NMEA 0183* oraz *NMEA 2000*. Obecnie interfejs *NMEA 0183* zastępowany jest przez nowszy *NMEA 2000*, jednak praca dotyczy będącego przez lata standardem oraz ciągle wykorzystywanego starszego protokołu. Odniesienia do protokołu NMEA w dalszej części pracy oznaczać będą wersję 0183.

1.1. Format zdania NMEA

Należy wydzielić podstawowe typy zdań: talker sentences (źródeł danych), proprietary sentences (zdefiniowanych przez poszczególnych producentów), query sentences (zapytań). Pierwszy typ, na którym skupia się ta praca, obejmuje sygnały GPS, dane z urządzeń wbudowanych w jednostkę, czujników. Typ drugi pozwala indywidualnym producentom zdefiniować własne formaty, zdania tego typu zaczynają się od znaków „$P" poprzedzających trzy literowy identyfikator producenta. Następnie zawarte są dane, o których decyduje producent. Ostatni typ pozwala urządzeniu zażądać wybranego rodzaju zdania od innego urządzenia. Ogólne formaty zdań wyglądają następująco:

1. Talker Sentence - `$ttsss,d1,d2,....<CR><LF>`, gdzie „tt" oznacza identyfikator nadawcy, „sss" identyfikator zdania, a „d1, d2..." kolejne pola danych.

2. Proprietary Sentence - `$Psss,d1,d,2,...<CR><LF>`, gdzie „sss" oznacza identyfikator producenta, a „d1,d2..." kolejne pola danych.

3. Query Sentence - `$ttllQ,sss,<CR><LF>`, gdzie „tt" oznacza identyfikator żądającego, „ll" identyfikator urządzenia, do którego zostało skierowane zapytanie, a „sss" identyfikator żądanego zdania.

Każde zdanie nadane w protokole NMEA zaczyna się od znaku „$", dane oddzielone są przecinkami, a zdanie kończy się sumą kontrolną poprzedzoną znakiem „*".

1.2. Identyfikatory nadawców

Identyfikatory nadawców rozróżniają sentencje NMEA ze względu na źródło danych, o których mowa w danym zdaniu.

Kod źródłowy 1.1. Identyfikatory nadawców
Źródło: Opracowanie własne

```
enum TalkerIdentifiers {
    AG= 'AG', AP= 'AP', CD= 'CD', CR= 'CR', CS= 'CS', CT= 'CT', CV= 'CV',
    CX= 'CX', DF= 'DF', EC= 'EC', EP= 'EP', ER= 'ER', GP= 'GP', HC= 'HC',
    HE= 'HE', HN= 'HN', II= 'II', IN= 'IN', LC= 'LC', RA= 'RA', SD= 'SD',
    SN= 'SN', SS= 'SS', TI= 'TI', VD= 'VD', DM= 'DM', VW= 'VW', WI= 'WI',
    YX= 'YX', ZA= 'ZA', ZC= 'ZC', ZQ= 'ZQ', ZV= 'ZV'
}
```

Kod źródłowy 1.2. Deskryptory nadawców
Źródło: Opracowanie własne

```
const TalkerDescriptors: Record<TalkerIdentifiers, string> = {
    [TalkerIdentifiers.AG]: 'Autopilot – General',
    [TalkerIdentifiers.AP]: 'Autopilot – Magnetic',
    [TalkerIdentifiers.CD]: 'Communications – Digital Selective Calling (DSC)',
    // deskryptor uległa skróceniu w celach edytorskich
    [TalkerIdentifiers.ZC]: 'Timekeeper – Chronometer',
    [TalkerIdentifiers.ZQ]: 'Timekeeper – Quartz',
    [TalkerIdentifiers.ZV]: 'Radio Update, WWV or WWVH'
};
```

1.3. Identyfikatory zdań

Lista wspieranych przez aplikację identyfikatorów zdań.

Kod źródłowy 1.3. Identyfikatory zdań
Źródło: Opracowanie własne

```
enum SentenceIdentifiers {
    AAM= 'AAM', ALM= 'ALM', APA= 'APA', APB= 'APB', ASD= 'ASD', BEC= 'BEC',
    BOD= 'BOD', BWC= 'BWC', BWR= 'BWR', BWW= 'BWW', DBK= 'DBK', DBS= 'DBS',
    DBT= 'DBT', DCN= 'DCN', DPT= 'DPT', DSC= 'DSC', DSE= 'DSE', DSI= 'DSI',
    DSR= 'DSR', DTM= 'DTM', FSI= 'FSI', GBS= 'GBS', GGA= 'GGA', GLC= 'GLC',
    GLL= 'GLL', GRS= 'GRS', GST= 'GST', GSA= 'GSA', GSV= 'GSV', GTD= 'GTD',
    GXA= 'GXA', HDG= 'HDG', HDM= 'HDM', HDT= 'HDT', HSC= 'HSC', LCD= 'LCD',
    MSK= 'MSK', MSS= 'MSS', MWD= 'MWD', MTW= 'MTW', MWV= 'MWV', OLN= 'OLN',
    OSD= 'OSD', ROO= 'ROO', RMA= 'RMA', RMB= 'RMB', RMC= 'RMC', ROT= 'ROT',
    RPM= 'RPM', RSA= 'RSA', RSD= 'RSD', RTE= 'RTE', SFI= 'SFI', STN= 'STN',
    TLL= 'TLL', TRF= 'TRF', TTM= 'TTM', VBW= 'VBW', VDR= 'VDR', VHW= 'VHW',
    VLW= 'VLW', VPW= 'VPW', VTG= 'VTG', VWR= 'VWR', WCV= 'WCV', WDC= 'WDC',
    WDR= 'WDR', WNC= 'WNC', WPL= 'WPL', XDR= 'XDR', XTE= 'XTE', XTR= 'XTR',
    ZDA= 'ZDA', ZDL= 'ZDL', ZFO= 'ZFO', ZTG= 'ZTG'
```

```
}
```

Kod źródłowy 1.4. Deskryptory zdań
Źródło: Opracowanie własne

```
const SentencesDescriptions: Record<SentenceIdentifiers, string> = {
    [SentenceIdentifiers.AAM]: 'Waypoint Arrival Alarm',
    [SentenceIdentifiers.ALM]: 'GPS Almanac Data',
    [SentenceIdentifiers.APB]: 'Autopilot Sentence "B"',
    // deskryptor uległa skróceniu w celach edytorskich
    [SentenceIdentifiers.ZDL]: 'Time & Distance to Variable Point',
    [SentenceIdentifiers.ZFO]: 'UTC & Time from Origin Waypoint',
    [SentenceIdentifiers.ZTG]: 'UTC & Time to Destination Waypoint'
};
```

1.4. Zdania ujęte w aplikacji

Z racji bardzo dużej ilości istniejących zdań protokołu NMEA do realizacji pracy inżynierskiej zostały wybrane zdania uwzględnione w używanym symulatorze NMEA Simulator. Oznacza to pełne dekodowanie oraz wyświetlanie danych w nich zawartych, jednak aplikacja została stworzona w taki sposób, aby dodawanie kolejnych zdań było jak najprostsze. Szczegóły zostaną omówione w dalszym rozdziale. Dane dostarczone przez symulator opisują zarówno położenie jak i dane na temat samej jednostki. Na ich podstawie zbudowany został interfejs graficzny aplikacji. Pełna lista znajduje się w tabeli 1.1.

Tabela 1.1. Zdania NMEA w pełni dekodowane przez aplikacje
Źródło: Opracowanie własne

Identyfikator nadawcy	Nadawca	Identyfikator zdania	Zdanie
SD	Sounder	DBT	Depth below transducer
GP	Global Positioning System (GPS)	GGA	Global Positioning System Fix Data
GP	Global Positioning System (GPS)	GSA	GPS DOP and active satellites
II	Integrated instrumentation	MTW	Mean Temperature of Water
WI	Weather Instruments	MWD	Wind Direction and Speed
WI	Weather Instruments	MWV	Wind Speed and Angle
GP	Global Positioning System (GPS)	RMC	Recommended Minimum Navigation Information
II	Integrated instrumentation	VHW	Water speed and heading

// 1. Protokół NMEA: - format - talker IDs - sentence IDs - sentences decoded by app
// 2. Przegląd dostępnych rozwiązań - problem z parsowaniem tylko GPS w większości rozwiązań - problem z integracją generycznych parserów z GUI (dają z każdej ramki gene-

ryczny obiekt, gdzie nie wiemy co znaczy jakie pole) // 3. Aplikacja: - język JS (Typescript)
- framework (React Native, expo) - Zarządzanie stanem (context, akcje, reducery, hooki)
- komunikacja z serwerem (web socket w contexcie) - generyczny parser i kodeki

ROZDZIAŁ 2

Aplikacja Mobilna

2.1. Język programowania

Domena tworzenia oprogramowania jest bardzo obszerna i może być dzielona wedle różnych kryteriów. Pierwszą przychodzącą na myśl kategoryzacją jest podział ze względu na różne języki programowania, te najbardziej popularne z nich, czyli C, C++, Python, Java, C#, JavaScript są wystarczająco odmienne, aby było to adekwatne kryterium. Innym, prawdopodobnie najbardziej szczegółowym zbiorem kryteriów są kryteria techniczne. Najważniejsze z nich to:

1. Sposób egzekucji (czyli sposób, w jaki wykonywany jest dany program komputerowy):
    - Języki interpretowalne, czyli takie, w których środowiskiem działania jest wirtualna maszyna a kod w sposób dynamiczny wykonywany jest linijka po linijce bez translacji na kod maszynowy danego procesora
    - Języki kompilowalne, czyli takie, w których do uruchomienia programu wymagana jest zamiana jego postaci na kod wykonywalny, zrozumiały dla danej architektury
    - Języki hybrydowe, czyli będącę połączeniem powyższych dwóch - przykładowo, niektóre fragmenty kodu są kompilowane (najczęściej tam gdzie zależy nam na optymalności rozwiązania)
2. Wspierany paradygmat programowania, głównie:
    - Structural Programming (SP), czyli paradygmat, który za podstawowy element budulcowy uznaje funkcje (procedury), które są zbiorem instrukcji, które mają wpływ na stan programu. Całość jest hierarchiczna oraz sekwencyjna. Pierwszym najszerzej rozpowszechnionym, uznanym i do dzisiaj używanym językiem strukturalnym jest język C

- Object Oriented Programming (OOP), czyli próba opisania oprogramowania w sposób jak najbardziej oddający otaczający nas świat. Klasy oraz obiekty stanowią warstwę abstrakcji reprezentującą dane procesy oraz podmioty. Wedle definicji możemy wymienić cztery filary programowania obiektowego: abstrakcja, hermetyzacja, polimorfizm oraz dziedziczenie. Obecnie jest to dominujący na rynku paradygmat, cechujący przede wszystkim C++, C# i Java. Nie jest jednak pozbawiony wad oraz problemów wynikających z samych jego założeń.
- Functional Programming (FP), czyli paradygmat można rzec inspirowany matematyką. W jego podstawowych założeniach kod źródłowy składałby się z wielu przewidywalnych funkcji, nazywanych 'pure functions'. Takie funkcje muszą zawsze zwracać dokładnie taki sam rezultat przy takich samych danych wejściowych, niezależnie od jakichkolwiek innych czynników. Tak restrykcyjne założenie uniemożliwia wywoływanie jakichkolwiek efektów ubocznych (ang. side effects) w kodzie, czego najbardziej dobitnym przykładem są operacje I/O będące podstawą znakomitej części oprogramowań. Chciałbym tutaj zaznaczyć, że wbrew rozumowaniu wielu inżynierów oprogramowania, nie oznacza niemożliwości wykorzystania efektów ubocznych. W językach programowania funkcyjnego jest to w pełni możliwe, a sam paradygmat dopuszcza łamanie założeń w niezbędnych operacjach. Programowanie funkcyjne posiada wysoki próg wejścia, może wydawać się nieintuicyjne i skomplikowane oraz przestarzałe. Zyskuje jednak ponownie na popularności, powoli stając się konkurentem dla programowania obiektowego. Przykładowe języki reprezentujące ten paradygmat to Haskell, Lisp.

3. Typowanie:

    - Statyczne, inaczej zwane twardo typowanym, języki tego typu wymagają od programisty znajomości i deklaracji typów danych. Cechujące wszystkie najstarsze języki programowania, a także większość obecnie używanych. Zapewnia możliwość znalezienia błędów ludzkich na poziomie kompilacji, czyli bez konieczności uruchamiania i sprawdzania oprogramowania. Przykładowe języki: C, C++, Java, C#
    - Dynamiczne, inaczej zwane miękko typowanym, języki tego typu nie posiadają oznaczeń typów danych, takich jak np. 'string' czy 'float', lub nie wymagają jawnego ich podawania. Zaletą takiej konwencji jest mniejsza ilość kodu oraz większa przystępność dla początkujących programistów. Przykładowe języki: Python, JavaScript.

Kolejnym logicznym kryterium jest docelowa platforma, dzięki czemu wyróżnimy programowanie wbudowane, web'owe, desktop'owe, mobilne. Należy zauważyć, że w tym wypadku niektóre z wyżej wymienionych języków trafią do wspólnego zbioru. Dla przykładu zarówno Python jak i JavaScript można nazwać językami web'owymi (lecz nie tylko). Pojawia się tutaj pewien problem, którym jest skalowanie oprogramowania. Jeśli firma posiada obiecujące oprogramowanie to ograniczenie grona odbiorców do danej platformy jest

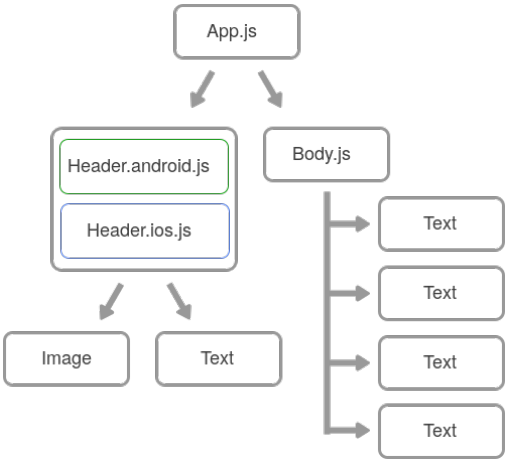
bardzo niekorzystne. Z między innymi tego powodu, języki programowania ewoluowały stając się wieloplatformowe. Dla przykładu Java, będąca jednym z najpopularniejszych języków na rynku, została stworzona z założeniem wsparcia różnych platform bez potrzeby ingerowania w kod.

Wyżej wymienione kryteria mają olbrzymie znaczenie podczas wyboru języka dla danego projektu. W przypadku aplikacji będącej podmiotem pracy inżynierskiej zdecydowano się na TypeScript, czyli nadzbiór języka JavaScript. Podobnie jak JavaScript wspiera on zarówno programowanie funkcyjne jak i obiektowe. Natomiast najistotniejszymi różnicami są kompilacja oraz wsparcie dla twardego typowania. Kompilowany jest on do czystego JavaScriptu, który następnie interpretowany jest przez środowisko klienckie, czyli np. node.js lub przeglądarka internetowa. Dzięki temu w przypadku, gdy nie jesteśmy w stanie lub nie chcemy określać typu danych - jest to całkowicie dozwolone. Przykładowo - dane dostarczane z zewnętrznej biblioteki, której twórca nie zapewnia ich struktury. W przypadku niektórych frameworków wymuszone jest użycie języka TypeScript, w innych jest to opcjonalne. Przykładowo jeden z najpopularniejszych frameworków do tworzenia GUI o nazwie Angular wymusza używanie tej technologii. Podobnie jak zyskujący coraz większą popularność NestJS, służący do tworzenia aplikacji serwerowych. Po wybraniu języka programowania przychodzi czas na wybór frameworka, który usprawni pracę.

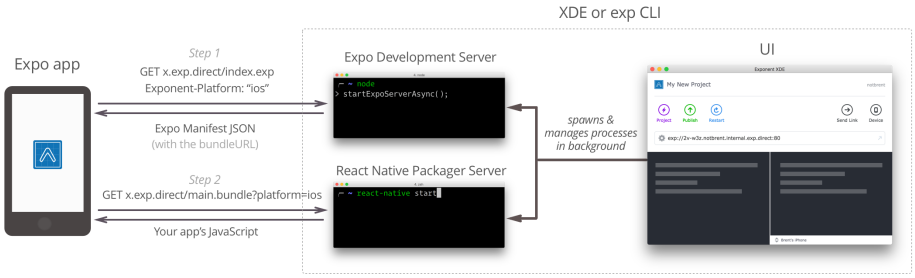
2.2. Framework

(TUTAJ DODAJ STATYSTYKE UDZIALU FRAMEWORKOW W RYNKU) Framework dostarcza programiście szkielet aplikacji, a także metody, komponenty, serwisy oraz składnię do ich wykorzystywania. Najpopularniejszym z frameworków do tworzenia aplikacji webowych po stronie klienta jest ReactJs , konkurujący z frameworkami nazwanymi Angular oraz Vue. Nie udostępnia on jednak możliwości tworzenia natywnych aplikacji mobilnych na systemy Android oraz iOS. Na te potrzeby powstał ReactNative. Jest to odpowiednik wspomnianego ReactJs dedykowany dla platform mobilnych. Podstawowe założenia takie jak kompozycja komponentów, przesyłanie danych, czy cykl życia komponentu pozostają bez zmian. Znaczące różnice wynikają z faktu, że w odróżnieniu do przeglądarek, aplikacje mobilne nie wspierają składni HTML oraz CSS. W miejsce pierwszego ReactNative dostarcza podstawowe komponenty pozwalające na budowanie widoku. Odpowiednikiem elementu w składni HTML jest w tym wypadku element <View />. Olbrzymią zaletą tego frameworka jest fakt, że w podanym przykładzie element <View /> zostanie wyrenderowany przy użyciu natywnego API platformy, co zapewnia dużo lepszą wydajność w porównaniu do przykładowo frameworku Cordowa, który do renderowania wykorzystuje WebView. Rysunek 2.1 Arkuszy Styli CSS zastąpione są natomiast przez obiekty Stylesheets. Zaletą takiego podejścia jest możliwość przeniesienia aplikacji napisanej w ReactNative na ReactJS przy stosunkowo niewielkim nakładzie pracy, gdyż znaczna część kodu może zostać użyta ponownie. Dodatkowo został wykorzystany framework oraz platforma Expo. Jest to zbiór narzędzi oraz serwisów zbudowanych wokół ReactNative, mający na celu przyspieszenie procesu tworzenia aplikacji. Dostarcza on między innymi przydatne warstwy abstrakcji nałożone na natywne API, przykładowo dostęp do kamery, czy żyro-

skopu. Dodatkowo zapewnia rozbudowane CLI (Command Line Interface). Jedną z jego możliwości jest niezwykle proste budowanie kodu na określoną platformę oraz serwer deweloperski z funkcją 'hot-reload'. Oznacza to, że zapisanie zmiany w kodzie natychmiast wywoła proces budowy odpowiednich plików, a programista ma możliwość obserwowania zmian na bieżąco używając symulatora urządzenia mobilnego lub rzeczywistego urządzenia. Sam proces połączenia z serwerem deweloperskim jest niesłychanie prosty. Expo dostarcza adres url zawierający pliki źródłowe. Dla ułatwienia jest on również przedstawiony jak kod QR. Dowolne urządzenie mobilne posiadające aplikację kliencką Expo po zeskanowaniu tego kodu uruchomi wersję deweloperską aplikacji. Rysunek 2.2 Oczywiście takie udogodnienia nakładają pewne ograniczenia, przez co dla dużych projektów z rozbudowaną logiką biznesową Expo może okazać się niewłaściwym wyborem. Jednak analiza pod kątem potrzeb omawianej pracy pozwoliła na uzasadniony wybór tego narzędzia.



Rysunek 2.1. Przekształcenia kodu ReactNative na natywne API platformy
Źródło: https://reactnative.dev/



Rysunek 2.2. Schemat działania Expo lokalnie
Źródło: https://docs.expo.io/versions/v36.0.0/workflow/how-expo-works

2.3. Narzędzia wspomagające

JavaScript jest niezwykle dynamicznie rozwijającym się językiem, co sprawia że wiele środowisk nie wspiera najnowszej składni. Aby rozwiązać ten problem skorzystano z transpilera Babel. Przekształca on najnowszą składnię języka na wspierany przez środowisko odpowiednik. Dzięki temu rozwiązaniu programista może korzystać z wszelkich udogodnień wprowadzonych wraz z kolejnymi wersjami języka nie martwiąc się przy tym o wsparcie starszych platform. Narzędzie to posiada wsparcie dla TypeScript'u oraz składni React-Native.

Kod źródłowy 2.1. Babel - przykładowe transformacje kodu
Źródło: Opracowanie własne

```
// input
enum myEnum {
  first=1,
  second=2,
  third=3
}
//output
var myEnum;

(function (myEnum) {
  myEnum[myEnum["first"] = 1] = "first";
  myEnum[myEnum["second"] = 2] = "second";
  myEnum[myEnum["third"] = 3] = "third";
})(myEnum || (myEnum = {}));

//input
const arr: number[] = [1,2,3]
arr.map(el => el * 2);
//output
var arr = [1, 2, 3];
arr.map(function (el) {
  return el * 2;
});

//input
const obj: Record<string, string> = {prop1: 'value', prop2: 'value'};
const { prop1, prop2 } = obj;
//output
var obj = {
  prop1: 'value',
  prop2: 'value'
};
var prop1 = obj.prop1,
    prop2 = obj.prop2;

//input
```

```
37
   const templateString: string = `First variable is ${prop1} and second is ${prop2}`;
38 //output
39 var templateString = "First variable is ".concat(prop1, " and second is
   ").concat(prop2);
40
41 //input
42 const ReactComponent: React.FC = () => <div>Text</div>
43 //output
44 var ReactComponent = function ReactComponent() {
45   return /*#__PURE__*/React.createElement("div", null, "Text");
46 };
```

Na potrzeby projektu została stworzona minimalistyczna konfiguracja zalecana przez twórców Expo.

Kod źródłowy 2.2. babel.config.js - plik konfiguracyjny
Źródło: Opracowanie własne

```
1 module.exports = function(api) {
2   api.cache(true);
3   return {
4     presets: ['babel-preset-expo'],
5   };
6 };
```

Dla zachowania standardów jakości oraz jednolitego stylu kodu zostało wykorzystane narzędzie do analizy statycznej. Za powszechny standard uznaje się oprogramowanie ESLint. Zapewnia ono olbrzymią ilość konfigurowalnych zasad, dotyczących zarówno stylu kodu takich jak maksymalna długość linii czy sposób indentacji, jak i jego logiki. Warto wspomnieć, że narzędzie to posiada wsparcie najpopularniejszych środowisk programistycznych. W przypadku języka JavaScript/TypeScript są to Visual Studio Code, WebStorm, IntelliJ IDEA. Dzięki temu błędy wykrywane są automatycznie podczas pisania kodu. W przypadku projektów komercyjnych tworzonych przez zespół programistów częstą praktyką jest tworzenie tak zwanych 'git hooków', które mają na celu sprawdzenie kodu przez między innymi linter przed utworzeniem commita, a w przypadku błędów uniemożliwienie go. Omawiany projekt korzysta ze zbioru zasad dostarczonego przez firmę Airbnb skonfigurowanych wedle osobistych preferencji.

2.4. Komunikacja z serwerem

Sposób komunikacji z serwisami sieciowymi zależy od typu API danego serwisu. Najczęściej spotykamy się z REST API opartymi na protokole http. Taki serwis opierania się na schemacie, w którym klient wysyła zapytanie i oczekuje na odpowiedź. Zapytanie http w momencie wywołania otwiera połączenie, a po przesłaniu odpowiedzi lub określonym (krótkim) czasie je zamyka. Ciągłe otwieranie i zamykanie połączenia jest olbrzymim problemem dla aplikacji wymagających stałej aktualizacji danych. W przypadku systemu wi-

zualizującego możliwie aktualny stan modułów konieczna jest ciągła aktualizacja danych. Dla takich potrzeb wykorzystywany jest protokół WebSocket, umożliwiający otworzenie połączenia między serwerem a klientem oraz komunikację w obie strony. Dzięki temu klient jest w stanie nasłuchiwać dostarczanych wiadomości bez konieczności tworzenia i wysyłania zapytania. Korzystając z tego samego połączenia klient ma możliwość wysłania wiadomości do serwera. Wymienione cechy czynią protokół WebSocket idealnym wyborem dla aplikacji wymagającej możliwie najbardziej aktualnych danych. Wykorzystany do testów NNMEA Simulator zapewnia funkcję serwera tego typu. Klient nie posiada konieczności wysyłania informacji do nadawcy, obsługuje jedynie ich odbiór. Wiersze od siódmego do dziewiątego (kod źródłowy **??**) zawierają implementację uchytwu (ang. handler) obsługującego odbiór wiadomości.

Kod źródłowy 2.3. Metoda tworząca połączenie WebSocket
Źródło: Opracowanie własne

```
1   const connectWebsocket = useCallback(() => {
2         const ws = new WebSocket(state.url);
3         ws.onopen = () => {
4             dispatch(NmeaConnectorActions.setConnected(true));
5         };
6
7         ws.onmessage = (e) => {
8             dispatch(NmeaConnectorActions.setData(e.data));
9         };
10
11        ws.onerror = () => {
12            dispatch(NmeaConnectorActions.setConnected(false));
13            ws.close();
14        };
15
16        ws.onclose = () => {
17            // refresh connection after 10s
18            setTimeout(() => connectWebsocket(), 10000);
19        };
20        return (() => {
21            ws.close();
22        });
23    }, [state.url]);
```

2.5. Dekoder zdań NMEA

Otrzymywane z serwera dane są pojedynczym stringiem zawierającym wszystkie zdania NMEA oddzielone znakiem końca linii. Pierwszym krokiem niezbędnym dalszego przetwarzania jest więc rozbicie otrzymanego ciągu znaków na tablicę zawierającą tekstową reprezentację pojedynczego zdania. Następnie dla każdego z elementów wywoływana jest metoda zwracająca obiekt reprezentujący ramkę NMEA o określonej strukturze. Dołożono wszelkich starań, aby rozwiązanie było jak najbardziej generyczne i łatwe w rozbudo-

wie. Proces dekodowania rozpoczyna się od walidacji sumy kontrolnej zdania (wiersze od drugiego do czwartego (kod źródłowy **??**). W przpadku powodzenia wyodrębniane są identyfikatory nadawcy oraz zdania (wiersze jedenaście oraz dwanaście (kod źródłowy **??**)). Na podstawie identyfikatora zdania uzyskiwany jest jego format (wiersz czternasty kod źródłowy **??** oraz kod źródłowy **??**). Format określa typ danych każdego pola w zdaniu, co umożliwiło stworzenie struktury określającej sposób dekodowania każdego pola (kod źródłowy **??**). Dzięki takiemu rozwiązaniu programista w łatwy sposób jest w stanie rozszerzyć obsługiwane typy pól. Posiadając informację o typach pól zawartych w zdaniu oraz o ich wartości możliwe jest stworzenie obiektu opisującego zdanie NMEA. Oznacza to także zakończenie części generycznej dla wszystkich typów zdań, gdyż format danych w poszczególnych polach nie określa ich znaczenia. Aby otrzymać w pełni wartościowe informacje należy opisać strukturę poszczególnych zdań oraz metodę przenoszącą surowe dane do właściwych pól tej struktury. Taki interfejs oraz metoda są wykorzystywane do utworzenia obiektu pakietu (wiersz dwudziesty drugi kod źródłowy **??**) reprezentującego w pełni odkodowane zdanie NMEA. Przykładowy plik dla zdania DBT przedstawia kod źródłowy 2.7. Rozszerzenie aplikacji o kolejny typ zdania wymaga jedynie utworzenia takiego pliku oraz dodanie utworzonego interfejsu i metody do kolekcji wykorzystywanych przy parsowaniu.

Kod źródłowy 2.4. Metoda parsująca zdania NMEA
Źródło: Opracowanie własne

```
1  export const parseNmeaSentence = (sentence: string): Packet => {
2      if (!validateNmeaChecksum(sentence)) {
3          throw Error(`Invalid sentence: "${sentence}".`);
4      }
5      const fields = sentence.split('*')[0].split(',');
6
7      if (fields[0].charAt(1) === 'P') {
8          throw Error('Proprietary sentences not suppoeted');
9      }
10
11     const talkerId = TalkerIdentifiers[fields[0].substr(1, 2)];
12     const sentenceId = SentenceIdentifiers[fields[0].substr(3)];
13
14     const formatter = SentencesFormats[sentenceId];
15     const sentenceProperties = formatter.split(',').map((format, index) => {
16         if (parsers[format]) {
17             return parsers[format](fields[index + 1]);
18         }
19
20         return fields[index + 1];
21     });
22     const packet = decoders[sentenceId](sentenceProperties);
23
24     packet.talkerId = talkerId;
25
26     return packet;
```

```
};
```

Kod źródłowy 2.5. Obiekt definiujący formaty zdań
Źródło: Opracowanie własne

```
const SentencesFormats: Record<SentenceIdentifiers, string> = {
    //obiekt skrócony w celach edytorskich
    [SentenceIdentifiers.DBT]: 'x.x,f=ft|M=m,x.x,f=ft|M=m,x.x,F',
    [SentenceIdentifiers.GGA]:
'hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx',
    [SentenceIdentifiers.MWD]: 'x.x,T,x.x,M,x.x,N,x.x,M',
    [SentenceIdentifiers.VHW]: 'x.x,T,x.x,M,x.x,N,x.x,K',
    }
```

Kod źródłowy 2.6. Obiekt definiujący sposób przetwarzania danych zależnie od ich rodzaju
Źródło: Opracowanie własne

```
const parsers: Object<(x: string) => any> = {
    x: (x) => parseIntSafe(x),
    xx: (x) => parseIntSafe(x),
    xxx: (x) => parseIntSafe(x),
    xxxx: (x) => parseIntSafe(x),
    xxxxx: (x) => parseIntSafe(x),
    xxxxxx: (x) => parseIntSafe(x),
    hh: (x) => parseIntSafe(x, 16),
    hhhh: (x) => parseIntSafe(x, 16),
    hhhhhh: (x) => parseIntSafe(x, 16),
    hhhhhhhh: (x) => parseIntSafe(x, 16),
    'h--h': (x) => hexWithPrefixToBytes(x),
    'x.x': (x) => parseFloatSafe(x),
    'c--c': (x) => x,
    'llll.ll': (x) => ParseCommonDegrees(x),
    'yyyyy.yy': (x) => ParseCommonDegrees(x),
    hhmmss: (x) => moment(x, 'HHmmss').format('HH:mm:ss'),
    'hhmmss.ss': (x) => moment(x, 'HHmmss').format('HH:mm:ss'),
    ddmmyy: (x) => moment(x, 'DD-MM-YY').format('DD/MM/YYYY'),
    'dd/mm/yy': (x) => moment(x, 'DD/MM/YY').format('DD/MM/YYYY'),
    'dddmm.mmm': (x) => ParseCommonDegrees(x)
};
```

Kod źródłowy 2.7. Kodek DBT
Źródło: Opracowanie własne

```
export interface DBTPacket {
    sentenceId: SentenceIdentifiers.DBT;
    sentenceName?: string;
    talkerId?: string;
    depthFeet: number;
    depthMeters: number;
    depthFathoms: number;
```

```
}

export const decodeDBT = (fields: Array<number>): DBTPacket => ({
    sentenceId: SentenceIdentifiers.DBT,
    sentenceName: SentencesDescriptions.DBT,
    depthFeet: fields[0],
    depthMeters: fields[2],
    depthFathoms: fields[4]
});
```

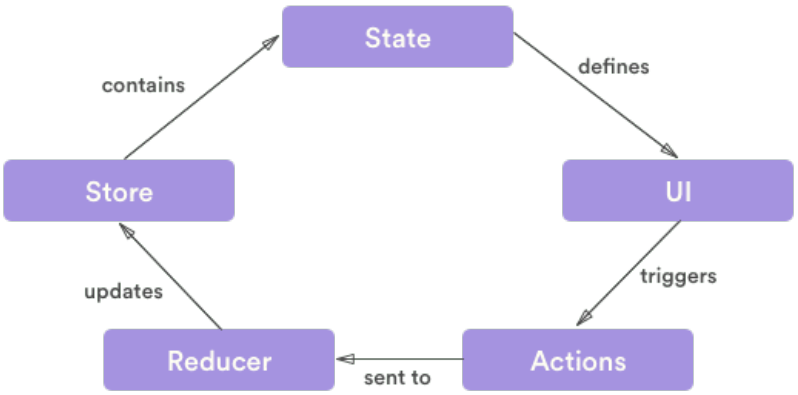
2.6. Zarządzanie danymi

Uzyskane dane oczywiście należy przechowywać w aplikacji. Komponenty tworzące widok zorganizowane są w hierarchicznej strukturze. Błędnym rozwiązaniem byłoby przekazywanie danych w dół hierarchii przez wszystkie komponenty pośredniczące, gdyż dostęp do nich powinny mieć jedynie elementy je wykorzystujące. W takich sytuacjach, gdy dane muszą być dostępne dla wielu komponentów na różnych poziomach zagnieżdżenia, stosuje się globalny stan aplikacji. Dużym problemem dotyczącym zagadnienia globalnie przechowywanych informacji jest możliwość ich mutacji. JavaScript przechowuje złożone struktury danych przez referencję, co umożliwia modyfikację obiektów (kod źródłowy **??**), które powinny pozostać stałe. Może to prowadzić do ciężkich w lokalizacji błędów. Aby tego uniknąć skorzystano z wzorca stanu. Rozbudowane projekty wykorzystują w tym celu biblioteki takie jak Redux i MobX, jednak potrzeby omawianej aplikacji nie uzasadniają ich wykorzystania. ReactNative zapewnia możliwość utworzenia kontekstu, dostępnego dla każdego komponentu w nim zagnieżdżonego. Korzystając z tej funkcjonalności utworzone zostały trzy konteksty:

1. nmeaConnectorContext odpowiedzialny za utworzenie połączenia WebSocket, odbieranie informacji ze strony serwera oraz przechowywanie ich.
2. unitContext odpowiedzialny za sformatowane dane dotyczące zdań NMEA opisujących parametry jednostki pływającej.
3. positionContext odpowiedzialny za sformatowane dane dotyczące zdań NMEA opisujących położenie jednostki pływającej.

Do ich obsługi wykorzystano wzorzec stanu narzucany przez bibliotekę Redux (Rysunek 2.3) Podstawowe elementy składniowe tego wzorca to:

1. Action, czyli obiekt składający się z typu oraz ładunku (ang. payload). Każda akcja musi zostać wyemitowana poprzez wywołanie metody dispatch(action).
2. Reducer, czyli funkcja odpowiadająca za zmianę stanu w odpowiedzi na wyemitowanie akcji. Zaleca się, aby funkcja pełniąca rolę reducera była 'pure function'.
3. Store, czyli obiekt przechowujący stan aplikacji. Jego zmiany oraz ich przebieg definiowany jest przez reducery.



Rysunek 2.3. Cykl życia Redux
Źródło: https://dev.to/radiumsharma06/abc-of-redux-5461

Dla każdego z kontekstów oznacza to, że stan może zostać jedynie odczytany. Jakiekolwiek zmiany w stanie wywołuje jedynie reducer w odpowiedzi na określoną akcję. W funkcji reducera koniecznie należy zwrócić całkowicie nowy obiekt stanu po każdej modyfikacji. Dzięki temu możliwe jest wykonanie procedur powiązanych ze zmianą, w szczególności przerenderowanie powiązanych komponentów. Konieczność ta wynika ze wspomnianej natury języka, mutacja obiektu nie zmieni jego referencji. Co za tym idzie, porównanie zmienionego obiektu stanu z poprzednim zwróci wartość 'true' sugerującą brak zmian. Zgodnie z założeniami wzorca stanu dla każdego kontekstu (w tym wypadku pełniącego rolę 'store') utworzono wymagane akcje (kod źródłowy 2.9) oraz funckję reducera (kod źródłowy 2.10).

Kod źródłowy 2.8. Mutacja obiektu
Źródło: Opracowanie własne

```
1  const myObj = {x: 1, y: 2};
2  myObj.y = 4;
3  // myObj is now {x: 1, y:4}
```

Kod źródłowy 2.9. Akcje kontekstu konektora NMEA
Źródło: Opracowanie własne

```
1  export const NmeaConnectorActions = {
2
       setConnected: (flag: boolean) => createAction(ActionTypes.SET_CONNECTED, flag),
3      setData: (data: any) => createAction(ActionTypes.SET_DATA, data),
4      setUrl: (url: string) => createAction(ActionTypes.SET_URL, url)
5  };
6
7  export type NmeaConnectorActions = ActionsUnion<typeof NmeaConnectorActions>;
```

Kod źródłowy 2.10. Funkcja reducera dla konektora NMEA

Źródło: Opracowanie własne

```
const nmeaReducer = (state: INmeaConnectorState, action: NmeaConnectorActions):
INmeaConnectorState => {
    switch (action.type) {
        case ActionTypes.SET_CONNECTED: {
            return { ...state, connected: action.payload };
        }
        case ActionTypes.SET_DATA: {
            return { ...state, data: action.payload };
        }
        case ActionTypes.SET_URL: {
            return { ...state, url: action.payload };
        }
        default: {
            throw new Error(`Unhandled action type: ${action}`);
        }
    }
};
```

Dzięki takiemu rozwiązaniu warstwa abstrakcji definiująca strukturę globalnych danych oraz przebieg ich zmian oddzielona jest od warstwy widoku. Dane dostarczane przez konteksty są konsumowane przez zależne od nich komponenty. W tym celu utworzono tak zwane 'hooki'. Jest to wzorzec wprowadzony przez twórców biblioteki React oraz frameworka ReactNative, określający metody pozwalające na skorzystanie z wewnętrznego stanu oraz cyklu życia komponentu w ciele funkcji. Wspomniane metody dostarczane są przez framework, a ich cechą charakterystyczną jest przedrostek 'use' występujący na początku każdej funkcji (przykładowo 'useState', 'useEffect'). Docelowym zamysłem twórców jest wykorzystanie ich do tworzenia własnych 'hooków' będących funkcjami zawierającymi współdzieloną logikę. Taką logiką w omawianej aplikacji jest użycie kontekstów. Każdy z nich zbudowany jest przy użyciu natywnych 'useContext' oraz 'useReducer'. Użycie pierwszego z nich przedstawia wiersz dwudziesty czwarty kod źródłowy 2.11, gdzie 'PositionContext' zdefiniowany jest w wierszu dwunastym 2.11. Wykorzystanie 'useReducer' przedstawia wiersz piętnasty 2.11. Zwraca on obiekt stanu oraz metodę 'dispatch' służące do odczytu stanu oraz emisji akcji. Wiersze od dwudziestego trzeciego do dwudziestego dziewiątego 2.11 zawierają definicję własnego hooka.

Kod źródłowy 2.11. Definicja kontekstu oraz hooka pozycji jednostki
Źródło: Opracowanie własne

```
export interface IPositionState {
    GGA: GGAPacket | null;
    GLL: GLLPacket | null;
    GSA: GSAPacket | null;
}

export interface IPositionContext {
    state: IPositionState,
    dispatch: Dispatch<PositionActions>
```

```
}

const PositionContext = createContext<IPositionContext | undefined>(undefined);

const PositionProvider = ({ children }) => {
    const [state, dispatch] = React.useReducer(positionReducer, initialState);
    return (
        <PositionContext.Provider value={{ state, dispatch }}>
            {children}
        </PositionContext.Provider>
    );
};

const usePosition = (): IPositionContext => {
    const context = React.useContext(PositionContext);
    if (context === undefined) {
        throw new Error('usePosition must be used within a PositionProvider');
    }
    return context;
};
```

Posiadając w ten sposób zdefiniowany kontekst programista zyskuje prosty i intuicyjny sposób na jego wykorzystanie w dowolnym komponencie niezależnie od jego umiejscowienia w hierarchii. Przykład użycia przedstawia 2.12

Kod źródłowy 2.12. Przykładowy komponent konsumujący hook kontekstu
Źródło: Opracowanie własne

```
// funkcja stworzona w celach edytorskich
function ConsumerComponent() {
    const unit = useUnit();

    // access state
    const MTWPacket = unit.state.MTW;

    // dispatch action
    unit.dispatch(UnitActions.setFrameData({ frameType: 'MTW', frameData: data })

    return <View><ChildComponents /></View>
```

Konteksty dotyczące jednostki oraz pozycji zawierają sformatowane zdania (wiersze od pierwszego do piątego 2.11), które użyte zostały do zbudowania GUI. Za przekształcenie otrzymywanego ciągu znaków do struktur pakietów zawartych w kontekstach odpowiada metoda przedstawiona w kodzie źródłowym 2.13). Jest ona wywoływana przez każdą zmianę stanu kontekstu 'nmeaConnectorContext', czyli przy odbiorze danych z serwera. Otrzymany ciąg znaków ulega podziałowi względem znacznika końca linii. Każdy element tablicy wynikowej tego podziału poddawany jest próbie dekodowania. Uzyskane dane przypisane zostają do ładunku akcji dla określonego kontekstu w zależności od identyfikatora zdania. Dzięki temu komponenty reprezentujące widok mogą bazować na spreparowanych danych zawartych w kontekstach 'unitContext' oraz 'positionContext',

gdyż są one aktualizowane natychmiastowo po otrzymaniu wiadomości z serwera.

Kod źródłowy 2.13. Dekodowanie i dystrybucja otrzymanych danych
Źródło: Opracowanie własne

```
    const nmeaConnector = useNmeaConnector();
    const position = usePosition();
    const unit = useUnit();

    useDeepCompareEffect(() => {
        if (nmeaConnector.state.connected &&
Object.keys(nmeaConnector.state.data).length > 0) {
            const frames = nmeaConnector.state.data.split(/\r?\n/);
            frames.map((frame: string) => {
                try {
                    const response = parseNmeaSentence(frame);
                    if (isPositionPacket(response)) {
                        position.dispatch(PositionActions.setFrameData({
frameType: response.sentenceId, frameData: response }))
                    }
                    else if(isUnitPacket(response)) {
                        unit.dispatch(UnitActions.setFrameData({ frameType:
response.sentenceId, frameData: response }))
                    }
                }
                catch(error) {
                    console.error('Parsing error, invalid input data: ', error)
                }
            })
        }
    }, [nmeaConnector.state])
```

2.7. Graphial User Interface (GUI)

1. Autor potrafi poprawnie formułować problemy badawcze i inżynierskie oraz metodycznie dążyć do ich rozwiązania;

2. Dyplomant posiada zdolność sprawnego korzystania z dostępnych zasobów wiedzy naukowej i technicznej, w związku z czym potrafi odpowiednio wyselekcjonować literaturę z danego zakresu oraz dokonać odpowiednich, wstępnych opracowań na jej podstawie. Umiejętność selekcji literatury, tzn. posługiwanie się przejrzystym i racjonalnym kryterium doboru (pozwalającym odrzucić pozycje pseudonaukowe lub o nikłym związku z tematem) jest szczególnie istotna dla oceny pracy. „Bibliografia” zamieszczona na końcu pracy pozwala bowiem w dużym stopniu wyrobić sobie pogląd na temat zakresu zapoznania się autora z daną problematyką. Autor powinien pamiętać, że nie liczba pozycji literaturowych lecz ich jakość podwyższa wartość merytoryczną i ocenę pracy.

3. Dyplomant potrafi na podstawie zgromadzonych i odpowiednio wyselekcjonowanych oraz skatalogowanych materiałów stworzyć tekst o charakterze naukowym lub technicznym. Tekst powinien być poprawny pod względem:

   rzeczowo-merytorycznym tzn. bezbłędnie i treściwie przedstawiać wszystkie istotne fakty, podać ich opis zgodny ze stanem rzeczywistym, oraz zwracać uwagę na prawidłowości występujące w obrębie analizowanych zjawisk;

   metodologicznym tzn. powinien zawierać opis umiejętnie zastosowanych metod, technik i narzędzi badawczych, właściwych dla danej dziedziny wiedzy i adekwatnych do podjętego problemu naukowego lub technicznego;

   logiczno-stylistycznym tzn. powinien zawierać tezy stawiane w sposób wzajemnie niesprzeczny, ściśle używane pojęcia — nieużywanie tego samego pojęcia na określenie różnych stanów rzeczywistości, poprawnie formułowane myśli i wnioski wynikające z przeprowadzonych analiz itd.

   Tekst pracy nie może w żadnym przypadku stanowić wyłącznie kompilacji innych tekstów!

4. Autor potrafi wykorzystać przedstawioną wiedzę teoretyczną do rozwiązania problemów inżynierskich;

5. Dyplomant nabył podstawowe umiejętności redakcyjne w zakresie pisania prac o charakterze naukowym lub technicznym; oznacza to opanowanie powszechnie przyjętych zasad w zakresie konstruowania struktury pracy, jej języka i stylu, wykonywania przypisów i bibliografii, dokumentacji technicznej itd.

Powyższe pięć wymagań w istotny sposób wpływa na ocenę uzyskiwaną w recenzjach pracy dyplomowej. Ponadto na ocenę pracy wpływa:

oryginalność rozumiana jako np. nowe ujęcie badanego problemu, interesujące, oryginalne zaprezentowanie wiedzy nienowej, ale np. mało znanej, udane wieloźródłowe (również z wykorzystaniem źródeł obcojęzycznych) opracowanie czy usystematyzowanie danej wiedzy, własna interpretacja problemu, pogłębiona analiza przedstawionych zagadnień itp.;

przydatność praktyczna tzn. opracowanie nowego urządzenia, programu komputerowego, dokonanie eksperymentu oraz zebranie i opracowanie danych pomiarowych itp., walory dydaktyczne przedstawionego opracowania, studia literaturowe przydatne w dalszej pracy naukowej itp.

Jak wspomniano wcześniej podczas redagowania pracy należy w sposób szczególny dbać o poszanowanie cudzych praw autorskich. Czytelnik powinien być w jednoznaczny sposób informowany, poprzez stosowanie odpowiednich odnośników do bibliografii, które z fragmentów pracy stworzone zostały w oparciu o literaturę przedmiotu, a które stanowią oryginalne przemyślenia i osiągnięcia autora. Należy również pamiętać, że odwzorowywanie

w pracy zdjęć, rysunków, tabel itp. bez podania źródła stanowi pogwałcenie cudzych praw autorskich. Dlatego zaleca się, aby tego typu elementy pracy były wykonane samodzielnie z uwzględnieniem własnych przemyśleń i modyfikacji. Autor pracy dyplomowej musi być świadomy, że konsekwencją udowodnienia pogwałcenia cudzych praw autorskich może być nawet cofnięcie decyzji o przyznaniu tytułu zawodowego[1].

2.8. Podział treści pracy

Tekst pracy dyplomowej zwyczajowo składa się z wprowadzenia, rozdziałów zasadniczych oraz zakończenia. Tekst zasadniczy powinien obejmować 60–80 stron. W spisie treści numerujemy rozdziały zasadnicze i dodatki natomiast wprowadzenie, zakończenie, bibliografia i inne mogą nie być numerowane.

2.8.1. Streszczenie

Streszczenia pracy w języku polskim i angielskim o długości ok. 10 wierszy każde (ok. 800 znaków) umieszcza się na trzeciej stronie pracy. Streszczenie powinno zwięźle przedstawiać podstawowe tezy i cele pracy, precyzować problem i metody badawcze oraz prezentować najważniejsze wyniki.

2.8.2. Wprowadzenie

We wprowadzeniu pracy dyplomowej powinny być zawarte następujące treści:

1. ogólne wprowadzenie do problematyki poruszanej w pracy;
2. omówienie koncepcji pracy i opis podjętego problemu badawczego:
    - obiektywne i subiektywne motywy podjęcia tematu,
    - cele pracy,
    - zakres pracy,
    - metody, techniki i narzędzia badawcze.

Cele i zakres pracy należy zredagować i wyróżnić w sposób pokazany na stronie 8.

2.8.3. Konstrukcja rozdziału pracy

Każdy rozdział powinien być **logicznie powiązany z resztą pracy**. Nie powinien on stanowić autonomicznej części związanej ze strukturą pracy jedynie samym tytułem. Treść rozdziału powinna logicznie wynikać z treści poprzedniego i implikować porządek rozdziału

[1] Zgodnie z art. 77. ust. 5 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce: *W przypadku gdy w pracy dyplomowej stanowiącej podstawę nadania tytułu zawodowego osoba ubiegająca się o ten tytuł przypisała sobie autorstwo istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego, rektor, w drodze decyzji administracyjnej, stwierdza nieważność dyplomu.*

następującego po nim. W pracy dyplomowej powinny w zasadzie znaleźć się co najmniej trzy rozdziały, ale nie powinno ich być też zbyt dużo. Pierwszy rozdział pracy powinien prezentować dotychczasowy stan wiedzy, wyniki oraz rozwiązania techniczne dotyczące poruszanej tematyki wraz z krytyczną analizą tego stanu i uzasadnieniem dokonanego doboru literatury. Każdy rozdział powinien — podobnie jak cała praca — posiadać swoją strukturę, formalny układ treści. I tak, rozpoczynając dany rozdział należy — zanim zacznie się omawianie poszczególnych podrozdziałów i paragrafów — **scharakteryzować pokrótce treść rozdziału.** Należy tutaj wymienić wstępnie tematykę podrozdziałów, wskazując na ich istotny związek z tematem rozdziału. Należy pamiętać, że zadaniem każdego rozdziału jest ustosunkowanie się do określonej części (aspektu) postawionego we wprowadzeniu problemu badawczego. W dalszej części rozdziału powinny mieć miejsce szczegółowe rozważania nad wybranymi aspektami problemu zdefiniowanego we wstępie do rozdziału. Rozdział powinien kończyć się podsumowaniem wniosków wypływających z przeprowadzonych analiz.

2.8.4. Zakończenie

Zakończenie jest niezbędnym elementem pracy dyplomowej. Powinny się w nim znaleźć następujące elementy:

1. podsumowanie uzyskanych wyników (niekoniecznie pozytywnych);
2. odpowiedzi na postawione we wprowadzeniu pytania problemowe (w świetle wyników, przeprowadzonych analiz lub przemyśleń);
3. omówienie obiektywnych trudności występujących w trakcie badań, wskazanie ich przyczyn i konsekwencji;
4. przedstawienie możliwości kontynuowania rozpoczętych badań, nowych pomysłów, które pojawiły się podczas pisania pracy, wskazanie wątków i obszarów problemowych wymagających dodatkowej naukowej analizy, lecz wychodzących poza ramy danej pracy itd.

Powyższe wymagania dotyczące treści pracy dyplomowej zredagowano na podstawie dostępnej literatury [**Boc2003**, **Honczarenko2000**, **Opoka2001**, **Pioterek1997**, **Zenderowski2004**].

ROZDZIAŁ 3

Strona formalno-edytorska pracy

Jak wspomniano we wprowadzeniu, struktura i strona edytorska niniejszego dokumentu zgodne są z zaleceniami dotyczącymi formatowania podstawowych elementów składowych pracy dyplomowej na Wydziale Elektrycznym Zachodniopomorskiego Uniwersytetu Technologicznego w Szczecinie oraz Zarządzeniem nr 26 Rektora ZUT z dnia 24 lutego 2020 r. (z późn. zm.) w sprawie Procedury procesu dyplomowania w ZUT, które ściśle regulują zasady edycji prac dyplomowych. Poniżej przedstawione zostały wytyczne jak należy formatować poszczególne, najistotniejsze elementy występujące w pracy dyplomowej. Redagując pracę dyplomant powinien w maksymalnym stopniu wzorować się na niniejszym dokumencie i w miarę możliwości przestrzegać poniższych zaleceń.

Podczas edycji można posługiwać się dowolnym, legalnie posiadanym przez siebie edytorem lub systemem do składu tekstu, np. typu WYSIWYG (MS Word, OpenOffice, LibreOffice, itp.) lub LaTeX. Wiele z tych narzędzi edytorskich umożliwia użycie specjalnie przygotowanych szablonów, zapewniających automatyczne formatowanie istotnych elementów pracy (rysunki, tabele, wzory, numeracje, wyliczenia, odnośniki literaturowe itp.) oraz generację spisów: treści, tabel, rysunków, ważniejszych oznaczeń i skrótów, kodów źródłowych, skorowidza oraz bibliografii. Szczególnie bogate funkcje w tym zakresie ma bezpłatny system składu tekstu LaTeX, przy pomocy którego przygotowany został niniejszy dokument.

Warto zauważyć, że wykorzystanie odpowiedniego szablonu w znacznym stopniu odciąża studenta od czynności edytorskich, a tym samym pozwala mu się skupić na stronie merytorycznej pracy. W systemie SIWE dostępny jest szablon dla systemu LaTeX, a także w formacie MS Word.

3.1. Podstawowe wymiary i układ pracy

Pracę należy drukować dwustronnie z wykorzystaniem drukarki laserowej lub atramentowej stosując druk czarno-biały lub kolorowy (strona tytułowa musi być kolorowa ze względu na logotypy Uczelni praz Wydziału). Wymaganą czcionką tekstu głównego jest Franklin

Gothic Book lub Helvetica o rozmiarze 12 pt.

Zalecane marginesy: lewy 20 mm[1], prawy 20 mm, górny 20 mm, dolny 30 mm (ze względu na numerację stron i przypisy) oraz margines na oprawę 15 mm. Odległość pomiędzy wierszami — 1,2 wiersza bez dodatkowych odstępów. Numeracja podrozdziałów nie powinna przekroczyć czterech stopni zagnieżdżenia tj. X.X.X.X. Jeśli występuje konieczność utworzenia dalszego zagłębienia należy przemyśleć strukturę dokumentu, gdyż najprawdopodobniej jest ona źle opracowana.

3.1.1. Strona tytułowa, strona streszczeń i słów kluczowych

Strona tytułowa zawiera kolorowe logotypy Uczelni oraz Wydziału o wysokości 1,8 cm. Poniżej znajdują się wyśrodkowane informacje o pracy tj. imię i nazwisko dyplomanta (14 pt, bold) a w liniach poniżej podajemy numer albumu, kierunek studiów, specjalność (dla studiów II stopnia) oraz formę studiów (12 pt) wpisując odpowiednio słowa „nr albumu:" „kierunek studiów:", „specjalność:" oraz „forma studiów:". Następnie piszemy tytuł pracy oraz (w kolejnych liniach) tytuł pracy w języku angielskim (14 pt, bold, kapitaliki). Poniżej wpisujemy informację o rodzaju pracy („Praca dyplomowa inżynierska" lub „Praca dyplomowa magisterska"), słowa „napisana pod kierunkiem:"(12 pt) i w kolejnym wierszu podajemy tytuł lub stopień naukowy, imię i nazwisko opiekuna pracy (14 pt, bold) oraz w linii poniżej jednostkę organizacyjną opiekuna (Katedra lub Pracownia). Warto zwrócić uwagę, iż jeśli promotor jest mężczyzną, po skrócie "dr" w dopełniaczu występuje kropka (alternatywny zapis to "dra"). Poniżej umieszczamy informacje o datach wydania tematu („Data wydania tematu pracy:") oraz w kolejnym wierszu pozostawiamy miejsce na datę dopuszczenia pracy do egzaminu do wypełnienia przez Dziekanat („Data dopuszczenia pracy do egzaminu:") wpisane czcionką o rozmiarze 10 pt dosunięte do lewego marginesu. Ostatnim elementem strony tytułowej jest podanie miejscowości „Szczecin" (12 pt) oraz roku (12 pt), w którym praca została przedstawiona do obrony.

Wzorcem strony tytułowej pracy jednoosobowej jest pierwsza strona niniejszego dokumentu, w wypadku krótszego lub dłuższego tytułu pracy zmianie ulegnie wyłącznie odstęp pomiędzy ostatnim wierszem tytułu w języku angielskim a informacjami o opiekunie. Druga strona pracy pozostaje pusta, a trzecia zawiera oświadczenie autora określone Zarządzeniem Rektora.

Na czwartej stronie pracy należy umieścić streszczenia oraz słowa kluczowe w języku polskim i angielskim, a także tytuł pracy w języku angielskim (w wypadku pracy pisanej w całości w języku angielskim należy w tym miejscu podać tytuł w języku polskim). Poniżej nagłówka (ok. 25 pt) piszemy odpowiednią treść streszczenia oraz słów kluczowych i tytuł angielski.

3.1.2. Podział na rozdziały

Każdy rozdział powinien zaczynać się od nowej strony (najlepiej nieparzystej). W odległości ok. 50 mm od górnej granicy strony wpisujemy dosunięty do lewego marginesu napis

[1] 1 mm = 2,85 pt

„ROZDZIAŁ” (14 pt, bold) oraz numer rozdziału (cyfry arabskie, 14 pt, bold). W odległości ok. 18 mm piszemy tytuł rozdziału[2] (20 pt, bold), a następnie w odległości ok. 20 mm rozpoczynamy pisanie tekstu pracy. Zaleca się, aby pierwszy akapit po tytule rozdziału czy sekcji nie zawierał wcięcia akapitowego (w innym przypadku wcięcie akapitowe jest wielkości ok. 10 mm).

Tytuł podrozdziału (poziom typu X.X., np. 2.1. – 17 pt, bold) powinien posiadać odstęp przed (ok. 16 mm) i po (ok. 9 mm). Poziom niższy śródtytułu (X.X.X., np. 2.1.1. – 14 pt, bold) powinien posiadać odstępy przed (ok. 10 mm) i po (ok. 9 mm).

3.1.3. Wzory matematyczne

Wielkość czcionki wzoru powinna być taka sama jak tekstu głównego. Można rozróżnić trzy typowe rozwiązania podczas pisania wzorów matematycznych wewnątrz dokumentu.

Pierwszym sposobem jest pisanie wzoru w tekście bez stosowania eksponowania i bez numeracji. Dotyczyć to powinno wzorów mniej znaczących, które nie wymagają eksponowania, a tym bardziej zastosowania numeracji w celu późniejszego odniesienia się do nich. Oto przykład wzoru tak składanego: $K = \sum_{i=0}^{n} k_i^3$.

Drugi sposób to wzór eksponowany:

$$K = \sum_{i=0}^{n} k_i^3$$

Jest to przypadek stosowany podczas wyprowadzeń jako etap pośredni, mniej istotny. Najistotniejsze wzory, szczególnie takie, które będą pomocne w dalszej części pracy, powinny zostać wyeksponowane i numerowane (sposób trzeci):

$$K = \sum_{i=0}^{n} k_i^3 \tag{3.1}$$

$$\phi = \prod_{\alpha=0}^{n} \alpha^{3\gamma} + \int_{-\infty}^{\infty} \alpha^{t\sqrt{\gamma}} \, \mathrm{d}\gamma \tag{3.2}$$

Elementy wzorów umieszczane w tekście należy wpisywać tym samym stylem czcionki co we wzorze np. „zmienna i oznacza indeks”. Numerację wzoru umieszcza się po prawej stronie i jest ona ciągła w ramach rozdziału np. (3.2). Odnośniki do wzoru można realizować według następujących przykładów: „ze wzoru (3.1) na stronie 33 wynika...” lub „uwzględniając (3.1) oraz (3.2) otrzymujemy...” itp.

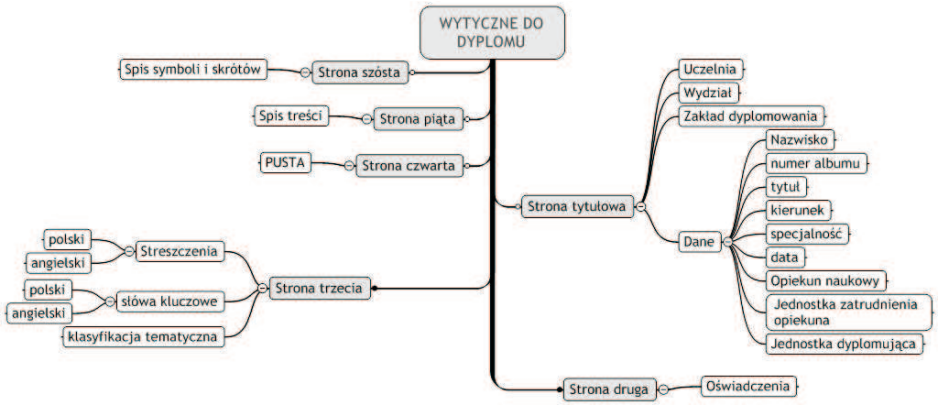
3.1.4. Rysunki

Każdy rysunek umieszczony w tekście (w odległości ok. 7 mm od poprzedzającego wiersza) powinien zawierać opis **pod** rysunkiem (w odległości ok. 7 mm poniżej), zaczynający się od „Rysunek X.X” (11 pt, bold) a następnie opis rysunku złożony czcionką 11 pt.

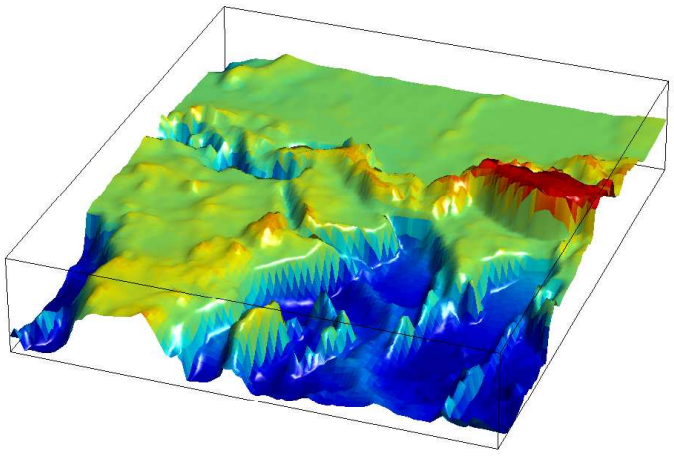
[2] po tytułach rozdziałów i podrozdziałów nie stawia się kropki

W wypadku rysunków zaczerpniętych z literatury pod opisem rysunku powinna znaleźć się informacja „Źródło: (11 pt)” i podajemy opis wraz ze wskazaniem źródła bibliograficznego (np. „Na podstawie [**Opoka2001**]”).

Zaleca się, aby podpis pod rysunkiem był dosunięty do lewego marginesu. Numeracja rysunków powinna być ciągła w ramach rozdziału – np. Rysunek 3.2. W tekście pracy odnośniki do numeru rysunku wykonujemy wpisując słowo „Rysunek” i podając jego numer np. „patrz Rysunek 3.2 na stronie 34”. Podanie numeru strony, na której ten rysunek się znajduje, nie jest konieczne, jednak ułatwia czytelnikowi poruszanie się po tekście pracy. Rysunki umieszczamy jako wyśrodkowane bezpośrednio lub możliwie blisko **po** fragmencie tekstu, w którym nastąpiło pierwsze odwołanie się do danego rysunku.



Rysunek 3.1. Podpis pod rysunkiem (bez kropki)
Źródło: Na podstawie [**Opoka2001**]



Rysunek 3.2. Rysunek z Matlaba (print -depsc2 name.eps)
Źródło: Opracowanie na podstawie dokumentacji elektronicznej MATLAB 7.0 [**Mathworks2004**]

3.1.5. Tabele

Każda tabela powinna zawierać opis umieszczony **nad** tabelą. Piszemy „**Tabela X.X**" (11 pt, bold, 11 mm poniżej tekstu głównego) i dalej treść opisu (11 pt). Zaleca się, aby podpis nad tabelą był dosunięty do lewego marginesu, natomiast sama tabela może być wyśrodkowana. Skład tabeli rozpoczynamy ok. 4 mm poniżej jej opisu.

Tabela 3.1. Przykładowy opis tabeli (bez kropki na końcu)
Źródło: Na podstawie [źródło bibliograficzne]

Współczynnik	$\alpha = 10{,}8$	$\alpha = 7{,}9$
a_0	1,0000	1,0000
a_2	-6,3710	-4,9320
a_4	18,4145	10,6350
a_6	-32,1102	-13,3432
a_8	37,7124	10,9568

Odnośniki do tabel umieszczamy według podobnej zasady jak w wypadku rysunków np.: „... patrz Tabela 3.1 na stronie 35". Tekst główny kontynuujemy ok. 10 mm poniżej tabeli. W wypadku tabeli zaczerpniętych z literatury pod opisem tabeli powinna znaleźć się informacja „Źródło: (11 pt)" – podajemy tu opis wraz ze wskazaniem źródła bibliograficznego (np. „Na podstawie [7]").

3.1.6. Kody źródłowe

Kody programów składamy czcionką maszynową Inconsolata (10 pt) z uwzględnieniem, w razie potrzeby, kolorowania składni. W treści pracy należy podawać jedynie **najważniejsze** klasy, procedury lub funkcje a pozostałe kody źródłowe można zawrzeć i omówić w dodatku (Dodatek A na stronie 41) lub umieścić tylko na dołączonej płycie CD-ROM.

Kody źródłowe umieszczamy pomiędzy liniami poziomymi zgodnie z przykładem. Opis powinien znajdować się **nad** umieszczonym kodem. Piszemy „**Kod źródłowy X.X" (12 pt, bold)** a następnie czcionką podstawową (12 pt) treść opisu. W wypadku, gdy dyplomant zamieszcza kody źródłowe nie swojego autorstwa, powinien wskazać ich źródło.

Kod źródłowy 3.1. Kod procedury generowania rysunku
Źródło: Na podstawie [**Mathworks2004**]

```
1 % To jest przykład
2 u = [zeros(1,10) 1 zeros(1,20)];
3 % Plot
4 stem(n,u);
5 xlabel('Time index n');
6 ylabel('Amplitude');
7 title('Unit Sample Sequence');
8 axis([-10 20 0 1.2]);
```

W treści pracy możemy odnosić się do opisów kodu źródłowego w następujący sposób: „... patrz Kod źródłowy 3.1 na stronie 35”. Możemy również odnosić się do konkretnych nazw klas bądź metod w tekście np. „funkcja `title(...)` umożliwia dodanie tytułu do wykresu w środowisku Matlab”.

Zdarzyć się może, że będziemy chcieli omówić pewien fragment kodu źródłowego. Pomocne stają się wtedy znaczniki linii zamieszczone w formatowaniu kodów źródłowych, np. „Wiersze od piątego do ósmego (kod źródłowy 3.1) zawierają przykładowe formatowanie wykresu typu `stem()`”.

Kod źródłowy 3.2 jest przykładem dołączania listingu z pliku poprzez polecenie `\codeinput`. Dokumentacja polecenia zawarta jest w pliku `Pracadyp.tex` w linii 342.

W wypadku wykorzystania oprogramowania specjalistycznego (na przykład popularny Doxygen), które generuje sformatowaną dokumentację przydatną w pracy, można bez dodatkowego formatowania umieścić ją w dodatku.

Kod źródłowy 3.2. Procedura dołączenia do grupy odbiorców SAP
Źródło: Opracowanie własne

```
1  public UdpClient connectToSAPmcastGroup()
2  {
3      UdpClient client = new UdpClient();
4      IPEndPoint localEp = new IPEndPoint(IPAddress.Any, 9875);
5
6      client.ExclusiveAddressUse = false;
7      client.Client.SetSocketOption(
8          SocketOptionLevel.Socket,
9          SocketOptionName.ReuseAddress,
10         true);
11     client.Client.Bind(localEp);
12
13     IPAddress multicastaddress = IPAddress.Parse("224.2.127.254");
14     client.JoinMulticastGroup(multicastaddress);
15
16     return client;
17 }
```

3.1.7. Bibliografia

Wszystkie pozycje bibliograficzne powinny być zacytowane w treści pracy.

Opisy bibliograficzne wynikają ze zwyczajów stosowanych w różnych dziedzinach nauki lub wytycznych różnych wydawnictw. Przygotowując bibliografię powinno się więc **konsekwentnie przestrzegać** kolejności elementów opisu, jak i sposobu ich zredagowania. Opis bibliograficzny powinien być przygotowany tak, aby możliwe było jednoznaczne określenie źródła bibliograficznego. Przy sporządzaniu bibliografii zaleca się **przyjąć kolejność alfabetyczną** względem nazwiska pierwszego autora, natomiast przy kilku pracach tego samego pierwszego autora przyjąć należy ich kolejność według daty publikacji (rosnąco). W wypadku wykorzystania w pracy źródeł drukowanych lub internetowych nie-

posiadających jawnie wskazanych autorów (np. materiały firmowe, dokumentacje), można w spisie bibliografii wydzielić taką grupę źródeł bibliograficznych.

Przykład cytowania źródła internetowego: [**Chwalowski2002**], podręcznika dostępnego w wersji elektronicznej: [**Nowacki1996**, **Reckdahl1997**], artykułu naukowego: [**Iksinski2000**] i książki: [**Honczarenko2000**, **Opoka2001**].

Wygodne narzędzie do posługiwania się wieloma pozycjami literaturowymi stanowić może menedżer bibliografii zgodny z notacją BibTeX np. darmowy JabRef. Plik przechowujący pozycje bibliograficzne zdefiniowane dla dokumentu ma nazwę `bibliografia.bib`. Na podstawie tego pliku LaTeX generuje spis literatury (Bibliografię). Do tego celu służy pakiet **Biber**, stanowiący następcę pakietu BibTeX, zapewniający m.in. pełne wsparcie standardu Unicode. W przypadku użycia edytora TeXstudio współpracującego ze środowiskiem MiKTeX w systemie Windows, może być konieczna zmiana domyślnego narzędzia bibliograficznego w konfiguracji programu (menu: opcje/konfiguruj/zbuduj/metapolecenia), a także jego wywołanie (menu: narzędzia/bibliografia lub skrót klawiszowy F8), a następnie ponowna kompilacja dokumentu.

3.1.8. Spisy rysunków, tabel, symboli i skrótów, kodów źródłowych

Każdy spis powinien rozpoczynać się na oddzielnej stronie. Należy umieścić odpowiedni nagłówek na górze strony (ok. 70 pt od góry) tj. „Spis rysunków", „Spis tabel", „Wykaz ważniejszych oznaczeń i skrótów", „Spis kodów źródłowych" czcionką pogrubioną wielkości 20 pt. Poniżej (ok. 55 pt) należy umieścić właściwy spis pisany czcionką podstawową (12 pt) z podstawową odległością pomiędzy wierszami.

Wykaz ważniejszych oznaczeń i skrótów należy umieścić na początku pracy dyplomowej na nowej stronie po spisie treści. Przy wykonywaniu tego spisu powinno się stosować kolejność sortowania: skróty alfabetycznie, oznaczenia łacińskie - małe litery, wielkie litery, oznaczenia literami alfabetu greckiego - małe litery, wielkie litery, inne. Pozostałe spisy umieszcza się na końcu pracy dyplomowej.

3.1.9. Skorowidz

Skorowidz nie jest elementem obowiązkowym pracy dyplomowej. Jednak w wypadku prac obszernych, przeglądowych, opisujących wiele różnych terminów wykorzystanie skorowidza jest bardzo pomocne. Skorowidz wymusza również na autorze przemyślenie struktury terminów w nim umieszczonych, a co za tym idzie sama struktura treści pracy ulega uporządkowaniu.

Na początku strony (ok. 70 pt od górnego marginesu) należy umieścić napis „Skorowidz" (20 pt, bold). Poniżej (ok. 55 pt) wyliczamy hasła skorowidza wskazując po przecinku na stronę wystąpienia hasła. W wypadku podhasła należy umieścić je jako podrzędne w stosunku do hasła ważniejszego np. hasło „transformata Fouriera" będzie podhasłem hasła „transformata". W takim przypadku nie powtarzamy już słowa „transformata" lecz zastępujemy je myślnikiem. W przypadku użycia edytora TeXstudio współpracującego ze środowiskiem MiKTeX w systemie Windows, w celu umieszczenia skorowidza na końcu

pracy może być konieczne ”ręczne” wywołanie polecenia (menu: narzędzia/indeks).

3.2. Oprawa pracy dyplomowej

Wymaga się stosowania opraw „miękkich” kanałowych z grzbietami metalowymi (tzw. C-BIND) lub termobindowanych. Przednia okładka powinna być przezroczysta, tylna okładka może być plastikowa. Nie są dopuszczalne oprawy „twarde” ani „zwykłe” bindowanie.

ROZDZIAŁ 4

Wymagania dotyczące wersji elektronicznej pracy dyplomowej

Zgodnie z obowiązującym Zarządzeniem nr 26 Rektora ZUT z dnia 24 lutego 2020 r. (z późn. zm.) w sprawie Procedury procesu dyplomowania w Zachodniopomorskim Uniwersytecie Technologicznym w Szczecinie obowiązkiem studenta jest umieszczenie wersji elektronicznej pracy, tożsamej z wersją drukowaną, w systemie informatycznym (e-Dziekanat) celem jej archiwizacji i kontroli w Jednolitym Systemie Antyplagiatowym (JSA).

Zgodnie z Uchwałą Rady Wydziału Elektrycznego oraz Regulaminem Studiów ostateczną wersję pracy dyplomowej student zobowiązany jest złożyć w regulaminowym terminie **w wersji elektronicznej w e-Dziekanacie** (wymagania dotyczące pliku będą sprawdzane podczas transmisji pliku do systemu). Formę papierową pracy student składa najpóźniej w ciągu kolejnych 3 dni u promotora (opiekuna) oraz w Dziekanacie Wydziału, **łącznie w trzech wydrukowanych i podpisanych egzemplarzach**.

Warto podkreślić, iż zgodnie z obowiązującym Zarządzeniem Rektora ZUT w sprawie Procedury procesu dyplomowania w ZUT **datą złożenia pracy jest data jej umieszczenia w systemie informatycznym** a nie data złożenia pracy u promotora (opiekuna).

Zabrania się umieszczania w wersji elektronicznej pracy dyplomowej dodatkowych elementów np. znaków wodnych. Wymagany jest jeden plik w formacie PDF o rozmiarze nieprzekraczającym **15 MB** (zgodnie z wymogami JSA) umożliwiający przeszukiwanie tekstu w całej treści pracy.

Zakończenie

W tym miejscu należy umieścić zakończenie pracy przygotowane zgodnie z wcześniejszymi zaleceniami (patrz rozdział 2.8.4 na stronie 29).

DODATEK A

Tytuł załącznika jeden

Treść załącznika jeden. Mogą to być elementy związane z opisem technicznym wykonanych w pracy urządzeń, programów, dokumentacją projektową itp. Mogą to być również źródła literaturowe w postaci kart katalogowych najistotniejszych elementów elektronicznych użytych w projektowanym urządzeniu itp.

DODATEK B

Zawartość dodatkowej płyty CD-ROM

W tym rozdziale powinno się przedstawić **zawartość DODATKOWEJ płyty CD dołączonej ewentualnie do wydrukowanej pracy**, która zawiera kody programów, wersje elektroniczne dokumentacji, materiały dodatkowe zebrane przez studenta itp.
Uwaga! Nie jest to opis płyty, zawierającej wersję elektroniczną pracy, ale płyty dodatkowej, nieobowiązkowej.

Spis tabel

Spis rysunków

Spis kodów źródłowych