



# Manual do przedmiotu „Zaawansowane projektowanie aplikacji mobilnych”

## Wprowadzenie

Celem przedmiotu jest nabycie przez studentów wiedzy i umiejętności z zakresu przygotowania aplikacji hybrydowej, tzn. dostępnej jako:

- aplikacja webowa mobilna – witryna dedykowana dla urządzeń mobilnych wyglądem przypominająca aplikacje mobilne oraz dobrze obsługująca zdarzenia dotyku;
- aplikacja mobilna wieloplatformowa – możliwość kompilacji rozwiązania jako aplikacji mobilnej dla platform Android, iOS i Windows Mobile.

W tym zakresie na zajęciach przedstawiany jest Ionic Framework w powiązaniu z Apache Cordova. Studenci na wstępie poznają procedurę przygotowania środowiska, podstawy tworzenia aplikacji mobilnych, w tym frameworku w uwzględnieniu architektury rozwiązań. Następnie poprzez komponenty, kontrolki i zdarzenia zdefiniują interfejs i podstawową funkcjonalność aplikacji. W oparciu o preprocesory CSS SASS customizacji poddany zostanie wygląd. W kolejnych tematach przygotowana zostanie warstwa REST dostępu do funkcjonalności systemu Back-end w zakresie pobierania i przesyłania danych. Ostatnią funkcjonalnością aplikacji będzie szereg funkcjonalności natywnych urządzeń, jak robienie zdjęć. Przedmiot kończy temat przygotowania pakietów wdrożeniowych i instalacji aplikacji w telefonach.

## Spis tematów

Wprowadzenie .....	1
<b>CZĘŚĆ WYKŁADOWCY</b>	
Temat 1: Przygotowanie środowiska i utworzenie projektu aplikacji wieloplatformowej .....	5
Temat 2: Analiza struktury projektu SPA .....	5
Temat 3: Komponenty budowania nawigacji projektu .....	5
Temat 4: Postawy tworzenia interfejsu użytkownika i funkcjonalności aplikacji .....	5
Temat 5: Kontrolki i mechanizmy projektowania interfejsu użytkownika .....	6
Temat 6: Adaptacja wyglądu aplikacji .....	6
Temat 7: Projektowanie warstwy dostępu do danych .....	7
Temat 8: Komunikacja front-end back-end aplikacji wieloplatformowej .....	7
Temat 9: Programowanie funkcji natywnych dla urządzeń mobilnych .....	7
Temat 10: Wdrażanie aplikacji w urządzeniach mobilnych .....	8
<b>CZĘŚĆ STUDENTA</b>	
Temat 1: Przygotowanie środowiska i utworzenie projektu aplikacji wieloplatformowej .....	9
Wprowadzenie teoretyczne .....	9
Przykłady .....	10
Zadania do realizacji .....	11
Pytania kontrolne .....	11
Praca własna studenta .....	11
Literatura .....	11
Temat 2: Analiza struktury projektu SPA .....	12
Wprowadzenie teoretyczne i przykłady .....	12
Zadania do realizacji .....	14
Pytania kontrolne .....	15
Praca własna studenta .....	15
Literatura .....	15
Temat 3: Komponenty budowania nawigacji projektu .....	16
Wprowadzenie teoretyczne i przykłady .....	16
Zadania do realizacji .....	17
Pytania kontrolne .....	17
Praca własna studenta .....	17
Literatura .....	18



Temat 4: Postawy tworzenia interfejsu użytkownika i funkcjonalności aplikacji.....	19
Wprowadzenie teoretyczne .....	19
Przykłady .....	21
Zadania do realizacji.....	21
Pytania kontrolne .....	22
Praca własna studenta .....	22
Literatura .....	22
Temat 5: Kontrolki i mechanizmy projektowania interfejsu użytkownika .....	23
Wprowadzenie teoretyczne i przykłady .....	23
Zadania do realizacji.....	27
Pytania kontrolne .....	28
Praca własna studenta .....	28
Literatura .....	28
Temat 6: Adaptacja wyglądu aplikacji .....	29
Wprowadzenie teoretyczne i przykłady .....	29
Zadania do realizacji.....	33
Pytania kontrolne .....	33
Praca własna studenta .....	34
Literatura .....	34
Temat 7: Projektowanie warstwy dostępu do danych .....	35
Wprowadzenie teoretyczne i przykłady .....	35
Zadania do realizacji.....	36
Pytania kontrolne .....	37
Praca własna studenta .....	37
Literatura .....	37
Temat 8: Komunikacja front-end back-end aplikacji wieloplatformowej .....	38
Wprowadzenie teoretyczne i przykłady .....	38
Zadania do realizacji.....	40
Pytania kontrolne .....	42
Praca własna studenta .....	42
Literatura .....	42
Temat 9: Programowanie funkcji natywnych dla urządzeń mobilnych .....	43
Wprowadzenie teoretyczne i przykłady .....	43
Zadania do realizacji.....	44



Pytania kontrolne.....	45
Praca własna studenta .....	45
Literatura .....	45
Temat 10: Wdrażanie aplikacji w urządzeniach mobilnych .....	46
Wprowadzenie teoretyczne .....	46
Zadania do realizacji.....	48
Pytania kontrolne.....	48
Praca własna studenta .....	48
Literatura .....	48

## CZĘŚĆ DLA WYKŁADOWCY

### **Temat 1: Przygotowanie środowiska i utworzenie projektu aplikacji wieloplatformowej**

W ramach tematu należy przedstawić koncepcję projektowania aplikacji hybrydowych. Ponadto przygotować środowisko dla programowania mobilnych aplikacji hybrydowych we frameworku Ionic, a także dodać i uruchomić przykładowe projekty.

Realizacja tematu:

1. Wprowadzenie teoretyczne do zagadnień – 30 minut.
2. Realizacja zadania 1 – 20 minut – wspólna realizacja zadania.
3. Realizacja zadania 2 – 20 minut – wspólna realizacja zadania. Zadanie może być zrealizowane jako praca własna studentów.
4. Realizacja zadania 3 – 15 minut – samodzielna realizacja zadania przez studentów.

### **Temat 2: Analiza struktury projektu SPA**

Należy przeprowadzić analizę poszczególnych typów projektów SPA w Ionic Framework zarówno jako struktur katalogów, jak i kodu w poszczególnych rodzajach plików.

Realizacja tematu:

1. Wprowadzenie teoretyczne do zagadnień – 20 minut.
2. Realizacja zadania 1 – 45 minut – wspólna realizacja zadania.
3. Realizacja zadania 2 – 25 minut – wspólna realizacja zadania. Zadanie może być zrealizowane jako praca własna studentów.
4. Realizacja zadania 3 – 25 minut – wspólna realizacja zadania. Zadanie może być zrealizowane jako praca własna studentów.

### **Temat 3: Komponenty budowania nawigacji projektu**

W ramach tematu nastąpi dodanie do projektu aplikacji kolejnej strony/ekranu oraz jej integracja w strukturze nawigacji.

Realizacja tematu:

1. Wprowadzenie teoretyczne do zagadnień – 15 minut.
2. Realizacja zadania 1 – 20 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Zadanie może być zrealizowane jako praca własna studentów.
3. Realizacja zadania 2 – 20 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Zadanie może być zrealizowane jako praca własna studentów.

### **Temat 4: Postawy tworzenia interfejsu użytkownika i funkcjonalności aplikacji**

Studenci poznają podstawowe aspekty tworzenia aplikacji hybrydowych jako projektowania ich interfejsu, obsługi zdarzeń oraz bindingu danych, na przykładzie projektu do wyliczania kosztów dojazdu.

Realizacja tematu:

1. Wprowadzenie teoretyczne do zagadnień – 25 minut.
2. Realizacja zadania 1 – 40 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.
3. Realizacja zadania 2 – 30 minut przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.



4. Realizacja zadania 3 – 30 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.

## **Temat 5: Kontrolki i mechanizmy projektowania interfejsu użytkownika**

Przedmiotem zajęć jest przygotowanie interfejsu użytkownika aplikacji dla przeprowadzania procesu ankietyzacji zajęć poprzez szereg typów kontrolek dla projektowania aplikacji Ionic, integracji kodu HTML i CSS oraz systemu wieloplatformowych ikon Ionic. Ponadto zdefiniowanie bindingu danych z wypełnionej ankiety.

Realizacja tematu:

1. Wprowadzenie teoretyczne do zagadnień – 40 minut.
2. Realizacja zadania 1 – 40 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.
3. Realizacja zadania 2 – 20 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.
4. Realizacja zadania 3 – 20 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.
5. Realizacja zadania 4 – 30 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.

## **Temat 6: Adaptacja wyglądu aplikacji**

Studenci poznają adaptację interfejsu użytkownika aplikacji do ankietyzacji zajęć poprzez zmienne SCSS wzorców szablonów Ionic oraz predefiniowane style. Zakres zajęć obejmuje również pełną customizację wyglądu komponentów kontrolek poprzez style SCSS poszczególnych stron oraz rozszerzenie i adaptację interakcji aplikacji z użytkownikiem poprzez powiadomienia i alerty.

Realizacja tematu:

1. Wprowadzenie teoretyczne do zagadnień – 45 minut.
2. Realizacja zadania 1 – 30 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.
3. Realizacja zadania 2 – 30 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.
4. Realizacja zadania 3 – 20 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.
5. Realizacja zadania 4 – 30 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.
6. Realizacja zadania 5 – 30 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.
7. Realizacja zadania 6 – 20 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.
8. Realizacja zadania 7 – 25 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.



9. Realizacja zadania 8 – 20 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.

## Temat 7: Projektowanie warstwy dostępu do danych

Przedmiotem tematu jest przygotowanie warstwy back-end dostępu do danych jako kontrolerów opartych o podejście REST API, przygotowanych we frameworku .NET jako ASP .NET Web API, a także wdrożenie dla kontrolerów dostępu z wielu domen – CORS.

Realizacja tematu:

1. Wprowadzenie teoretyczne do zagadnień – 45 minut.
2. Realizacja zadania 1 – 30 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.
3. Realizacja zadania 2 – 30 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.
4. Realizacja zadania 3 – 40 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.

## Temat 8: Komunikacja front-end back-end aplikacji wieloplatformowej

Należy przygotować warstwę front-end dla przesłania wypełnionej ankiety na serwer i napisać rozwiązania wyświetlające na czas przesyłania danych stosowny o tym komunikat. Przedmiotem zajęć jest też modyfikacja strony jako przejścia slajdów – dwu-widokowości strony – dla wyświetlania w jednym widoku danych ankiety po podaniu poprawnego kodu, a na drugiej - interfejsu dla wypełnienia i wysłania ankiety.

Realizacja tematu:

1. Wprowadzenie teoretyczne do zagadnień – 45 minut.
2. Realizacja zadania 1 – 60 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.
3. Realizacja zadania 2 – 40 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.
4. Realizacja zadania 3 – 90 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.

## Temat 9: Programowanie funkcji natywnych dla urządzeń mobilnych

Zajęcia pozwalają na zrozumienie wdrażania wsparcia dla funkcji natywnych urządzeń mobilnych. Przedmiotem tematu jest też opracowanie funkcjonalności w aplikacji ankietowania dla kilku funkcji natywnych, jak: wibracje, kamera, dostęp do plików, lokalizacja i kalendarz.

Realizacja tematu:

1. Wprowadzenie teoretyczne do zagadnień – 45 minut.
2. Realizacja zadania 1 – 40 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.
3. Realizacja zadania 2 – 60 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.



4. Realizacja zadania 3 – 40 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.
5. Realizacja zadania 4 – 80 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.
6. Realizacja zadania 5 – 45 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.

## **Temat 10: Wdrażanie aplikacji w urządzeniach mobilnych**

Temat prezentuje wdrożenie aplikacji dla ankietyzacji zajęć na telefonie z systemem Android poprzez kabel USB, a także pakiet do publikowania w sklepie.

Realizacja tematu:

1. Wprowadzenie teoretyczne do zagadnień – 25 minut.
2. Realizacja zadania 1 – 45 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania.
3. Realizacja zadania 2 – 25 minut – samodzielna realizacja zadania przez studentów przez dwie trzecie czasu. Wspólne sprawdzenie/wypracowanie rozwiązania. Zadanie może być zrealizowane jako praca własna studentów.



## CZĘŚĆ DLA STUDENTA

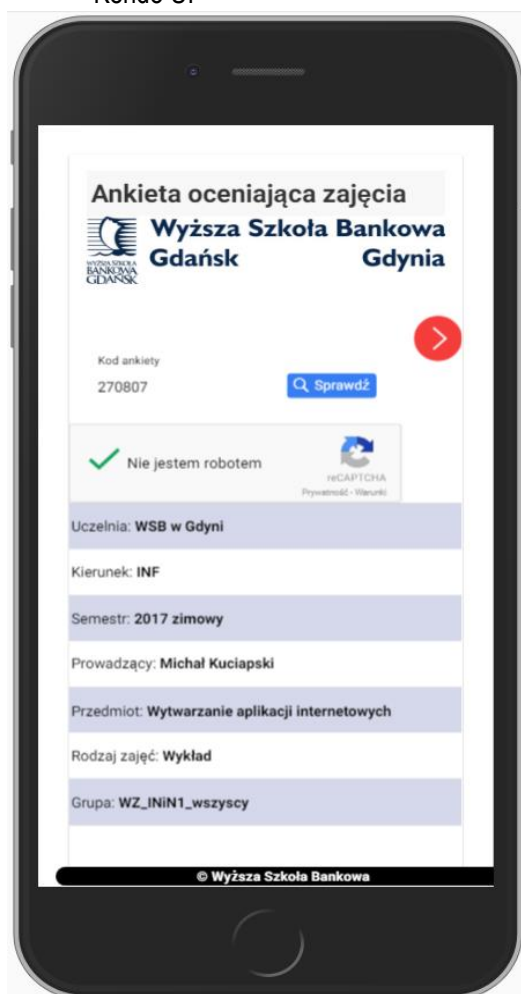
# Temat 1: Przygotowanie środowiska i utworzenie projektu aplikacji wieloplatformowej

## Wprowadzenie teoretyczne

Mobilne aplikacje hybrydowe są połączeniem rozwiązań aplikacji webowych oraz natywnych. Ich projektowanie jest podobne do tworzenia stron internetowych z wykorzystaniem technologii, takich jak HTML5, CSS oraz JavaScript. Jednak w porównaniu do dynamicznych witryn, za pomocą specjalnych frameworków, można zapewnić wygląd wysoce zbliżony do typowego dla aplikacji mobilnych, obsługę zdarzeń dotyku jak i kompilację do aplikacji natywnych. W przypadku kompilacji dla aplikacji natywnych, można oferować aplikacje w specjalnych sklepach jak Google Play.

Do najpopularniejszych frameworków dla budowy aplikacji hybrydowych zaliczane są:

- Ionic (Rys. 1)
- PhoneGap
- Mobile Angular UI
- Appcelerator Titanium
- Sencha Touch
- Kendo UI



Rys. 1 Przykład aplikacji hybrydowej w Ionic Framework



### Instalacja środowiska

W ramach przedmiotu przedstawiany jest wysoce popularny framework dla projektowania mobilnych aplikacji hybrydowych Ionic (<https://ionicframework.com>). Procedura zainstalowania Framework-a Ionic:

1. Instalacja Node.js jako serwera dla uruchamiania i monitorowania aplikacji Ionic - <https://nodejs.org/en>
2. Instalacja Framework-ów Ionic (aplikacje hybrydowe) i Cordova (programowanie funkcjonalności natywnej urządzeń) –  
w Node.js command prompt: `npm install -g ionic cordova`

### Utworzenie przykładowego projektu

Dodawanie projektów, ich monitorowanie i uruchamianie odbywa się poprzez rozszerzony o polecenia Ionic Node.js command prompt. **Utworzenie nowej aplikacji Ionic** wykonywane jest poprzez polecenie `ionic start`, np.

`ionic start kalkulatorKosztowDojazdu`

Polecenie przyjmuje dodatkowy parametr typu projektu:

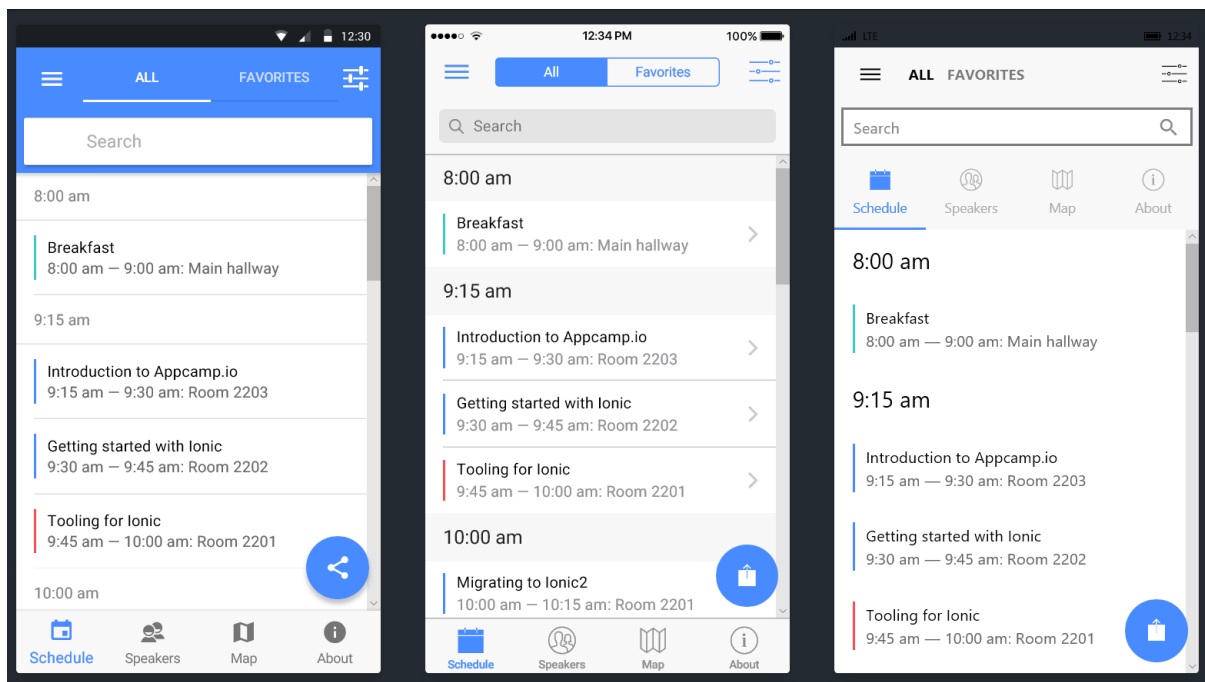
- `tabs` - projekt z interfejsem opartym o zakładki,
- `blank` – pusty projekt,
- `sidemenu` - projektu z menu bocznym i nawigacją,
- `super` - projekt ze wstępnie przygotowanymi stronami, dostawcami oraz wdrożoną nawigacją.
- `conference` – przykład w pełni opracowanej aplikacji,
- `tutorial` – projekt dla tutoriali w wirtualnie Ionic,
- `aws` - AWS Mobile Hub Starter.

### Uruchomienie projektu

**Uruchomienie projektu** dla wyświetlenia aplikacji w wersji dla przeglądarki możliwe jest poprzez polecenie `ionic serve`. Przydatne są flagi pozwalające na wyświetlanie logów działania aplikacji, a w szczególności równoległą prezentację aplikacji dla platform Android, iOS i Windows Mobile - `ionic serve --lab -c`.

### Przykłady

Rys. 2 prezentuje aplikację opartą o template `conference`, uruchomioną w trybie prezentacji na wiele platform - `ionic serve --lab -c`.



Rys. 2 Uruchomiony projekt oparty o template conference w trybie prezentacji na wiele platform

## Zadania do realizacji

1. Zainstaluj środowiska Ionic i Apache Cordova w najnowszych wersjach.
2. Utwórz przykładowy (tutorialowy) projekt Ionic, a następnie uruchom go.
3. Utwórz dwa puste projekty Ionic o nazwach: kosztDojazdu oraz ankieta.

## Pytania kontrolne

1. Jakie korzyści daje przygotowanie hybrydowych aplikacji mobilnych?
2. Jakie znasz frameworki dla tworzenia hybrydowych aplikacji mobilnych?
3. Opisz procedurę i składowe niezbędne dla przygotowania projektu we frameworku Ionic.

## Praca własna studenta

1. Zadanie 2.

## Literatura

1. Griffith C., Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova, O'Reilly Media 2017
2. Instalacja Ionic - <https://ionicframework.com/docs/intro/installation>
3. Dodanie nowej aplikacji - <https://ionicframework.com/docs/cli/starters.html>

## Temat 2: Analiza struktury projektu SPA

### Wprowadzenie teoretyczne i przykłady

Aplikacje przygotowywane we frameworku Ionic opracowywane są zgodnie z koncepcją **Single Page Application** (SPA). SPA to aplikacja internetowa, która pozwala na dynamiczną podmianę bieżącej strony, zamiast pełnego wczytywania nowej strony z serwera. To podejście eliminuje negatywny efekt przeładowywania kolejnych stron, co czyni aplikację bardziej podobną do aplikacji stacjonarnej, gdzie jest jeden dynamicznie zmieniający się widok. W systemie SPA pobierany jest dowolny niezbędny kod HTML, JavaScript i CSS w ramach żądania w tle, lub odpowiednie zasoby są dynamicznie ładowane i dodawane do strony w razie potrzeby.

### Struktura utworzonego projektu Ionic

Po utworzeniu projektu Ionic tworzonych jest szereg katalogów, z których kluczowe są:

- **src** – zawierający projekt aplikacji mobilnej, zatem będący katalogiem, w ramach którego w poszczególnych plikach tworzony jest kod aplikacji;
- **www** – powstaje na skutek kompilacji aplikacji, zawierając wynikową mobilną witrynę;
- **resources** – pliki pomocnicze dla kompilacji projektu do różnych platform;
- **node\_modules** – zawiera biblioteki dostępne w trakcie tworzenia aplikacji Ionic.

### Struktura projektu aplikacji Ionic

Tworzony projekt aplikacji webowej dla urządzeń mobilnych, która może być kompilowana jako natywna aplikacja na urządzenia mobilne znajduje się w katalogu **src**. Plikiem startowym aplikacji jest [src/index.html](#), którego celem jest załadowanie skryptów JavaScript i CSS wymaganych przez aplikację oraz uruchomienie aplikacji. Plikami skryptów są:

- [build/main.js](#) jest skompilowanym plikiem zawierającym kod js związany z frameworkiem Ionic oraz kod js będący wynikiem kompilacji poszczególnych stron, przez co jego rozmiar to kilka megabajtów.
- [cordova.js](#) – jest plikiem stosowanym wyłącznie w przypadku kompilacji projektu do natywnej aplikacji mobilnej, ponieważ zawiera funkcjonalność dostępu do funkcjonalności telefonu jak np. wibracje.

Startowy skrypt [build/main.js](#) w pliku [src/index.html](#) szuka wpisu `<ion-app></ion-app>`, w który ładowana jest aplikacja (interfejs użytkownika i zdarzenia).

Plik [src/app/app.module.ts](#) zawiera kod startowy aplikacji – kompilowany później do pliku [build/main.js](#). Na początku pliku znajduje się kod zbliżony z poniższym:

```
@NgModule({
  declarations: [MyApp, HelloIonicPage, ItemDetailsPage, ListPage],
  imports: [ BrowserModule, IonicModule.forRoot(MyApp)],
  bootstrap: [IonicApp],
  entryComponents: [MyApp, HelloIonicPage, ItemDetailsPage, ListPage],
  providers: []
})
export class AppModule {}
```

Najważniejsze dyrektywy w pliku to :

- **imports**: ... - wskazanie stosowanych modułów dla funkcjonalności;
- **entryComponents**: ... - lista modułów (czyt. stron), które zawiera aplikacja;
- **IonModule.forRoot(MyApp)** wskazuje na główny moduł, od którego rozpoczyna się ładowanie aplikacji.

W pliku [src/app/app.component.ts](#) znajduje się wskazana klasa „startowa” – **MyApp**.

```
import { Component } from '@angular/core';
import { Platform } from 'ionic-angular';
import { StatusBar } from '@ionic-native/status-bar';
import { SplashScreen } from '@ionic-native/splash-screen';

import { HomePage } from '../pages/home/home';
@Component({
  templateUrl: 'app.html'
})
export class MyApp {
  rootPage:any = HomePage;

  constructor(platform: Platform, statusBar: StatusBar, splashScreen: SplashScreen) {
    platform.ready().then(() => {
      statusBar.styleDefault();
      splashScreen.hide();
    });
  }
}
```

W pliku tym wskazane są ładowane moduły niezbędne dla zapewnienia funkcjonalności aplikacji. Najważniejszymi wpisami są jednak:

- wskazanie ekranu/strony startowej, która jest ładowana - `rootPage:any = HomePage;`
- wskazanie ładowanej strony - `templateUrl: 'app.html'`.

Strona `src/app/app.html` jest głównym szablonem aplikacji, który dla szablonu tutorial wygląda następująco:

```
<ion-nav id="nav" [root]="rootPage" #nav swipeBackEnabled="false"></ion-nav>
<ion-menu [content]="nav">
  <ion-header>
    <ion-toolbar>
      <ion-title>Pages</ion-title>
    </ion-toolbar>
  </ion-header>
  <ion-content>
    <ion-list>
      <button ion-item *ngFor="let p of pages" (click)="openPage(p)">
        {{p.title}}
      </button>
    </ion-list>
  </ion-content>
</ion-menu>
```

Powyższy szablon ładuje menu nawigacyjne poprzez `ion-menu`, a w centrum ekranu wyświetlana jest strona wskazana jako `rootPage`, czyli `HomePage`. Strona `HomePage` znajduje się w katalogu `src/pages/hello`.

### Struktura ekranu/strony w projekcie aplikacji Ionic

Każda ze stron aplikacji znajduje się w osobnym podkatalogu o swojej nazwie, w katalogu `src/pages`. Każda ze stron składa się z 3 plików:



- \*.html – zawartość strony,
- \*.scss – wygląd strony,
- \*.ts – funkcjonalność strony, w tym obsługa zdarzeń.

**Plik obsługi strony** z kodem w języku TypeScript ma formę analogiczną do poniższej:

```
import { Component } from '@angular/core';
@Component({
  selector: 'page-hello-ionic',
  templateUrl: 'hello-ionic.html'
})
export class HelloIonicPage {
  constructor() {
  }
}
```

W pliku wskazane są stosowane moduły w sekcji import. Ponadto `templateUrl`: określa plik strony. Funkcjonalność strony programowana jest poprzez kod wewnątrz klasy (class).

**Plik strony** opiera się o notację HTML oraz Ionic, np.

```
<ion-header>
  <ion-navbar>
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>Hello Ionic</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <h3>Moja pierwsza aplikacja Ionic!</h3>
  <p>
    Dopiero rozpoczynam tworzenie aplikacji.
  </p>
  <p>
    <button ion-button color="primary" menuToggle>Toggle Menu</button>
  </p>
</ion-content>
```

**Plik ze stylami \*.scss** jest domyślnie pusty, ponieważ aplikacje Ionic korzystają z predefiniowanych kilku szablonów stylów. Zatem w pliku \*.scss strony definiuje się jako ewentualne zmiany standardowego wyglądu komponentów i kontrolek.

## Zadania do realizacji

1. Sprawdź strukturę projektu utworzonego jako blank.
2. Sprawdź strukturę projektu utworzonego jako przykładowy (tutorial).
3. Sprawdź strukturę projektu utworzonego jako szablon conference.



## Pytania kontrolne

1. Z jakich komponentów (sekcji) składa się nowy projekt Ionic?
2. Jaka jest procedura uruchamiania plików projektu Ionic zanim załadowany zostanie ekran startowy?
3. Strony/ekrany/widoki aplikacji SPA w Ionic w jakich znajdują się katalogach?
4. Strona/ekran/widok aplikacji SPA w Ionic definiowana jest w których plikach?

## Praca własna studenta

1. Zadanie 2 i 3.

## Literatura

1. Griffith C., Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova, O'Reilly Media 2017
2. Ionic Tutorial - <https://ionicframework.com/docs/intro/tutorial>
3. Nawigacja - <http://ionicframework.com/docs/v2/getting-started/tutorial/navigation>

## Temat 3: Komponenty budowania nawigacji projektu

### Wprowadzenie teoretyczne i przykłady

#### Dodanie nowej strony (ekranu)

Dla dodania nowej strony do projektu wystarczy skopiować katalog innej strony z katalogu `src/pages` i nazwać go zgodnie z nazwą, którą ma mieć nowa strona. Następnie pliki w katalogu (\*.html, \*.ts i \*.scss) należy nazwać tak, jak ma się nazywać strona, czyli identycznie jak nazwa katalogu. Kluczowym plikiem strony jest plik \*.ts, który ładuje stronę oraz zawiera jej funkcjonalność, np.

```
import {Component} from '@angular/core';
import {NavController, NavParams} from 'ionic-angular';
import {ItemDetailsPage} from '../item-details/item-details';

@Component({
  templateUrl: 'list.html'
})
export class ListPage {
  selectedItem: any;
  icons: string[];
  items: Array<{ title: string, note: string, icon: string }>;

  constructor(public navCtrl: NavController, public navParams: NavParams) {
    this.selectedItem = navParams.get('item');

    this.icons = ['flask', 'wifi', 'beer', 'football', 'basketball', 'paper-plane',
      'american-football', 'boat', 'bluetooth', 'build'];

    this.items = [];
    for (let i = 1; i < 11; i++) {
      this.items.push({
        title: 'Item ' + i,
        note: 'This is item #' + i,
        icon: this.icons[Math.floor(Math.random() * this.icons.length)]
      });
    }
  }

  itemTapped(event, item) {
    this.navCtrl.push(ItemDetailsPage, {
      item: item
    });
  }
}
```

Przykładowa klasa strony w konstruktorze (`constructor(public navCtrl: NavController, public navParams: NavParams)`) przygotowuje listę do wyświetlenia na stronie. Najważniejsze instrukcje to:

- import – np. `import {ItemDetailsPage} from '../item-details/item-details'`; służące zaimportowaniu modułów z funkcjonalnością lub innych stron jak w przykładzie, do których można się odwołać w kodzie.
- @Component – np.  

```
@Component({
  templateUrl: 'list.html'
})
```



- Wskazuje nazwę pliku, w którym znajduje się ładowana strona, jaka zostanie wyświetlona.
- `export class` - **export class** `ListPage` {... - klasa dla obsługi strony, tzn. wczytywania danych czy obsługi zdarzeń

Klasa odpowiedzialna za obsługę strony posiada konstruktor, np. **constructor**(`public navCtrl: NavController`, `public navParams: NavParams`). Parametry w nim są deklaracjami obiektów zapewniających stosowną funkcjonalność, np. **public navCtrl: NavController** odnosi się do możliwości poprzez `navCtrl` nawigacji do innych stron. Kod w konstruktorze

```
this.items = [];  
for (let i = 1; i < 11; i++) {  
  this.items.push({  
    title: 'Item ' + i,  
    note: 'This is item #' + i,  
    icon: this.icons[Math.floor(Math.random() * this.icons.length)]  
  });  
}
```

Generuje listę elementów, gdzie każdy z nich ma tytuł (`title`), opis (`note`) i losową nazwę ikony (`icon`).

Klasa oprócz danych i konstruktora posiadać może metody obsługi zdarzeń – czyli metody, które są wywoływane po wystąpieniu zdarzenia np. wybrania palcem (`ang. tap`) elementu listy.

### Definiowanie nawigacji

Metody będące wynikiem zdarzeń mogą powodować przejście (nawigację) do innej strony, np.

```
itemTapped(event, item) {  
  this.navCtrl.push(ItemDetailsPage, {  
    item: item  
  });  
}
```

W tym przypadku poprzez zadeklarowany w konstruktorze obiekt (**public navCtrl: NavController**) otwierana jest inna strona (`this.navCtrl.push(ItemDetailsPage)`) – ze szczegółami, do której przekazywany jest obiekt z danymi (`item: item`) – wybrany element listy.

Strona, do której następuje przekierowanie – `ItemDetailsPage` – musi być uprzednio zaimportowana do projektu w początkowej sekcji pliku:

```
import { ItemDetailsPage } from '../pages/item-details/item-details';
```

### Zadania do realizacji

1. Dodaj nową stronę do projektu utworzonego jako przykładowy (tutorialowy).
2. Zmodyfikuj nawigację dla strony, tak aby uwzględniała ona nową stronę i możliwe było do niej przejście.

### Pytania kontrolne

1. Jaka jest procedura dodania nowej strony do projektu na podstawie już istniejącej?
2. Zanim będzie możliwe przejście do nowej strony, co należy wykonać, aby było to możliwe?
3. Jakie parametry i w jakim celu są podawane w konstruktorze klasy obsługującej stronę?
4. Jakie cele pełnią metody definiowane w klasie obsługującej stronę?

### Praca własna studenta

1. Zadania 1 i 2



## Literatura

1. Griffith C., Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova, O'Reilly Media 2017
2. <https://ionicframework.com/docs/intro/tutorial/adding-pages>
3. <https://ionicframework.com/docs/intro/tutorial/navigation/>
4. <https://ionicframework.com/docs/components/#navigation>

## Temat 4: Podstawy tworzenia interfejsu użytkownika i funkcjonalności aplikacji

### Wprowadzenie teoretyczne

Mobilne aplikacje hybrydowe we frameworku Ionic przygotowywane są w oparciu o:

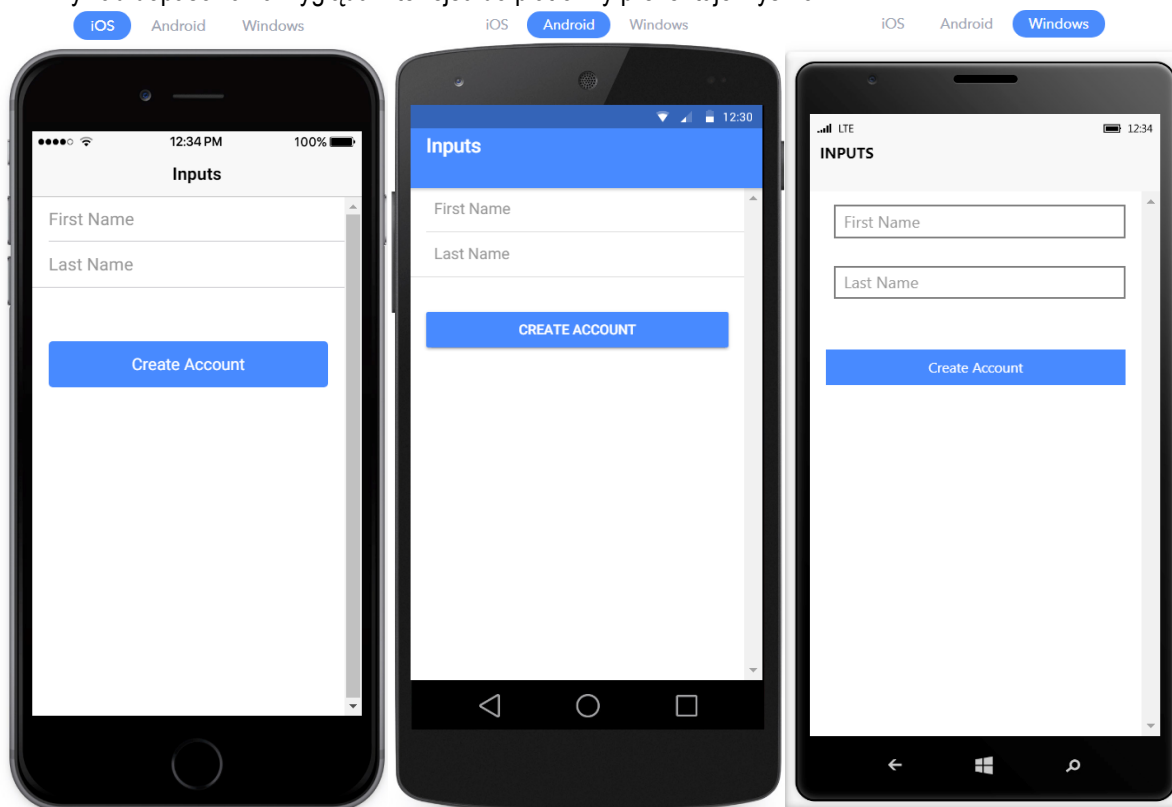
- kontrolki interfejsu użytkownika – dostępnych jest ich kilkadziesiąt, a wygląd oraz interakcja z nimi jest identyczna jak dla typowych kontrolek mobilnych;
- komponenty – pozwalające zwiększyć interaktywność aplikacji oraz user experience poprzez okna alertów czy powiadamiania o ładowaniu danych;
- obsługę interakcji – opartą o kod obsługi strony, pisany w języku Type Script kompilowanym do Java Script;
- obsługę funkcjonalności natywnych urządzeń – dostępnych jest kilkadziesiąt pluginów pozwalających na m. in.: korzystanie z wibracji, latarki, GPS czy kalendarza.

### Podstawy projektowania interfejsu użytkownika

W ramach frameworku Ionic dostępnych jest kilkadziesiąt kontrolek użytkownika, a wygląd oraz interakcja z nimi jest identyczna jak dla typowych kontrolek mobilnych. Ponadto ich wygląd jest domyślnie dostosowywany do platformy, a zatem zależy od:

- urządzenia, na którym wyświetlona została webowa aplikacja mobilna;
- platformy, na którą kompilowany jest projekt, jeśli ma zostać przygotowany natywny pakiet wdrożeniowy aplikacji mobilnej.

Przykład dopasowania wyglądu interfejsu do platformy prezentuje Rys. 3.



Rys. 3 Dopasowanie interfejsu użytkownika aplikacji Ionic do poszczególnych platform

Kontrolki tworzone są za pomocą specyficznej składni Ionic wprowadzającej nowe tagi oraz właściwości np. poniższy kod spowoduje wyświetlenie szeregu przycisków o różnych kolorach

Projekt: „Zaprogramowani na sukces – program kształcenia praktycznego na kierunku Informatyka w Wyższej Szkole Bankowej w Gdańsku”  
współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego

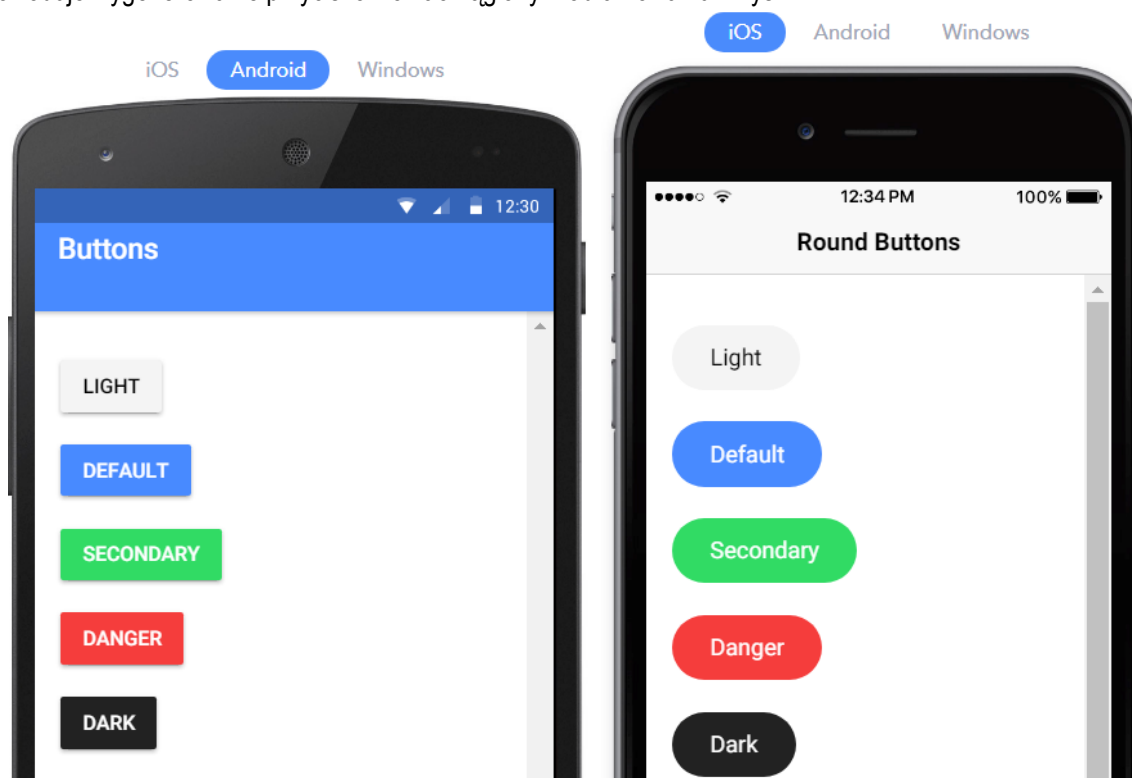
**POWR.03.01.00-00-N010/16**

```
<button ion-button color="light">Light</button>
<button ion-button>Default</button>
<button ion-button color="secondary">Secondary</button>
<button ion-button color="danger">Danger</button>
<button ion-button color="dark">Dark</button>
```

Liczba atrybutów i dostępnych dla nich wartości pozwala na tworzenie złożonych i zróżnicowanych interfejsów użytkownika. Samych przycisków istnieje wiele rodzajów, np. kod:

```
<button ion-button color="light" round>Light Round</button>
<button ion-button round>Primary Round</button>
<button ion-button color="secondary" round>Secondary Round</button>
<button ion-button color="danger" round>Danger Round</button>
<button ion-button color="dark" round>Dark Round</button>
```

spowoduje wygenerowanie przycisków o zaokrąglonym obramowaniu - Rys. 4.



Rys. 4 Generowanie różnego rodzaju przycisków

## Obsługa zdarzeń

Zdarzenia podczepiane są bezpośrednio na stronach poprzez notację Angular-a, który jest wykorzystywany przez framework Ionic. Nazwa zdarzenia podawana jest w nawiasie, a nazwa metody jej obsługującej, przypisywana jest poprzez operator „=”. Poniższy kod powoduje dodanie zdarzenia kliknięcia na przycisk:

```
<button (click)="clicked()"></button>
```

Metoda z obsługą zdarzenia powinna się znajdować w pliku \*.ts odpowiedzialnym za obsługę strony. Dla powyższego kodu byłoby to:

```
class KalkulatorKredytu {
  kwotaKredytu : number = 0.0;
  oprocentowanieKredytu : number = 7.8;
```



```
RRSO: number = 0.0;
clicked() {
  this.RRSO = this.kwotaKredytu * ....
}
}
```

Zgodnie z powyższym kodem metoda może korzystać z danych, które są dostępne dla strony – np. KwotaKredytu.

## Binding

Aby metody obsługujące zdarzenia mogły poprawnie działać, muszą mieć dostęp do danych wprowadzanych przez użytkowników na stronie. W tym celu stosowany jest mechanizm bindingu, polegający nie na pobieraniu danych poprzez kod, ale deklaracyjnym wskazaniu ich źródła – ich pobranie oraz wypisywanie odbywać się będzie automatycznie dzięki wsparciu frameworka.

Pobranie danych z formatki możliwe jest poprzez atrybut [(ngModel)] z przypisania nazwy pola w kodzie - np.

```
<ion-input type="number" value="0.0" [(ngModel)]="kwotaKredytu"></ion-input>
```

Wyświetlenie danych jest możliwe poprzez interpolację danych jako tzw. symbol wąsów (ang. mustache) - np.

```
<ion-label>RRSO kredytu wynosi: {{RRSO}}</ion-label>
```

Nazwa, do której jako binding danych występuje odwołanie - np. kwotaKredytu w przykładzie powyżej - musi zostać uprzednio zadeklarowana w klasie obsługującej stronę, np.

```
class KalkulatorKredytu {
  kwotaKredytu : number = 0.0;
}
```

## Przykłady

Przykład strony wykorzystującej mechanizm bindingu.

\*.html

```
<ion-input type="number" [(ngModel)]="km" step="1"></ion-input>
```

....

```
<ion-label>{{wynik}}</ion-label>
```

x.ts

```
export class HomePage {
```

```
  km: number = 1;
```

```
...}
```

## Zadania do realizacji

1. W utworzonym pustym projekcie kosztDojazdu zdefiniuj interfejs użytkownika zbliżony do poniższego. Zastosuj do tego celu elementy: List, Inputs, Labels, Button oraz inne zgodnie z uznaniem.



### Kalkulator kosztów dojazdu

Liczba kilometrów do przejechania

1321

Spalanie na 100

6,2

Koszt za 1 litr (zł)

4,32

WYLICZ

353,82 zł

2. Dla przycisku „wylicz” dodaj obsługę zdarzenia wyliczając koszt dojazdu i wypisując go w etykiecie poniżej.
3. Zdefiniuj binding danych w ten sposób, aby koszt dojazdu wyliczał się automatycznie na skutek zmian wartości w innych kontrolkach, bez konieczności klikania na przycisk wylicz.

### Pytania kontrolne

1. Jaka notacja jest stosowana dla dodawania kontrolek do strony?
2. W jaki sposób do przycisku można dodać zdarzenie kliknięcia?
3. W którym pliku definiuje się metody obsługi zdarzeń strony?
4. Aby kontrolka mogła się odwoływać do wartości pod nazwą liczbaBiletów, gdzie i w jaki sposób musi zostać ona zdefiniowana?

### Praca własna studenta

1. Zadanie 3.

### Literatura

1. Griffith C., Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova, O'Reilly Media 2017
2. Kontrolki wprowadzania danych w pola tekstowe - <https://ionicframework.com/docs/components/#inputs>
3. Przyciski - <https://ionicframework.com/docs/components/#buttons>
4. Podczepianie zdarzenia do kontrolek - <http://learnangular2.com/events>
5. Pobieranie i wypisywanie danych poprzez binding:
  - <https://angular.io/docs/ts/latest/guide/template-syntax.html>
  - <https://angular.io/docs/ts/latest/guide/forms.html>
  - <https://angular.io/docs/ts/latest/guide/template-syntax.html#!#binding-syntax>

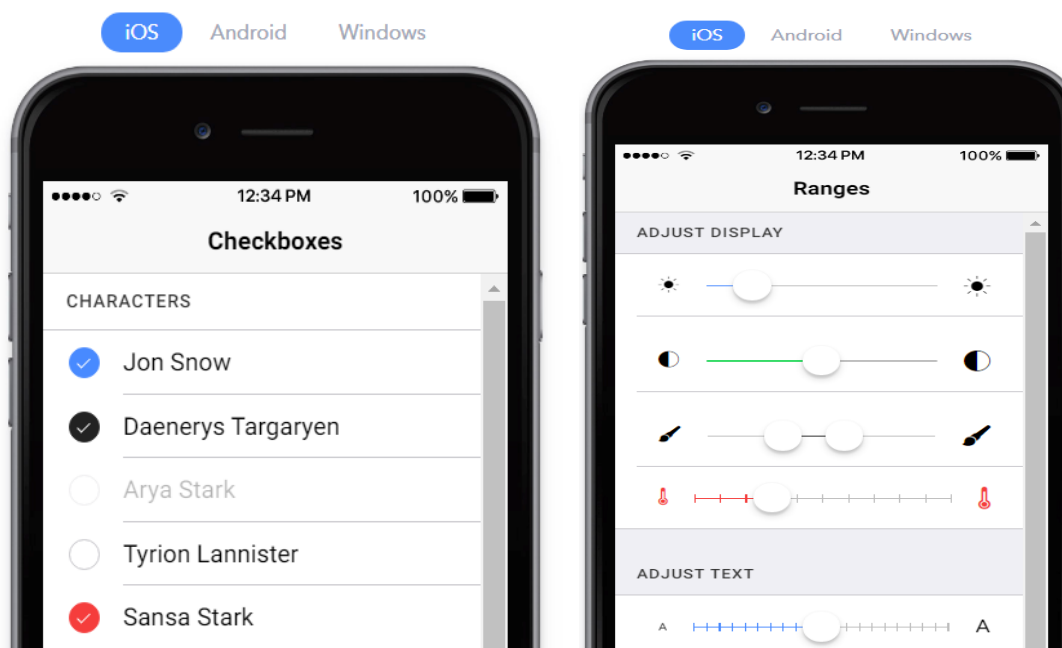
## Temat 5: Kontrolki i mechanizmy projektowania interfejsu użytkownika

### Wprowadzenie teoretyczne i przykłady

#### Podstawowe kontrolki

Do popularnych kontrolek należą (Rys. 5):

- przyciski (ang. buttons)
- listy (rys. poniżej) – wybrania elementów (ang. lists) jak i pól dla zaznaczenia (ang. checkbox)
- daty (ang. DateTime)
- pola tekstowe (ang. inputs)
- menu (ang. menus)
- zakresu (ang. range) (rys. 5)
- wybory z rozwijane listy (ang. listbox)

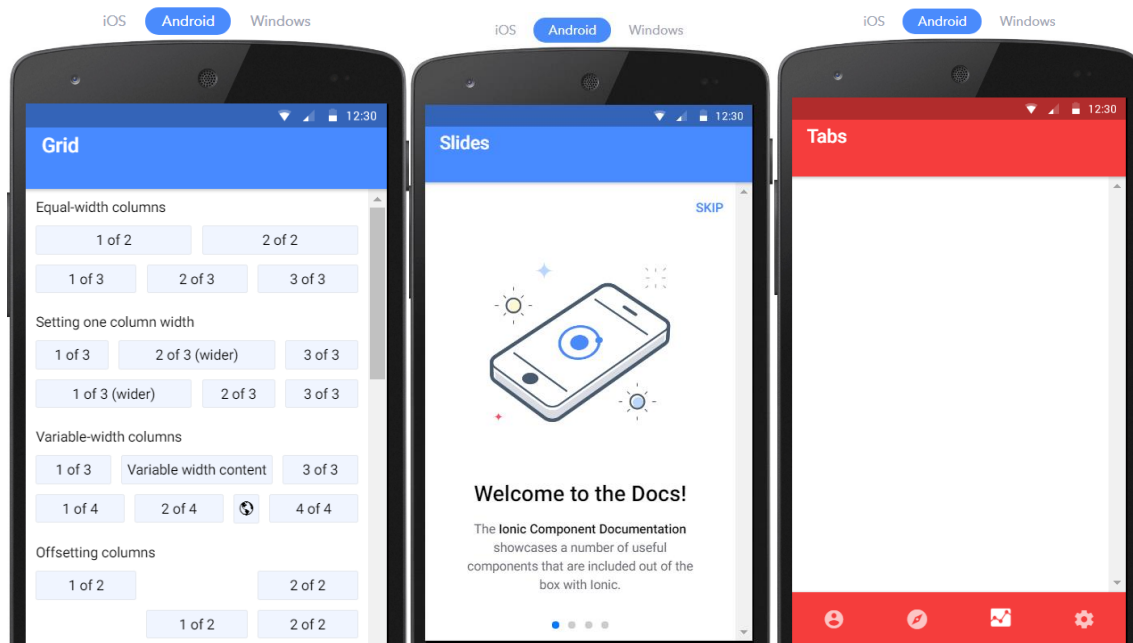


Rys. 5 Przykłady wyglądu kontrolki pola zaznaczenia oraz zakresu w Ionic Framework

Do wypisywania informacji służą etykiety, np. Etykieta - `<ion-label fixed> zł</ion-label>`.

Pozycjonowanie elementów na stronie odbywa się poprzez sekcje HTML, jak: header, section, article, aside, footer czy div – zatem zwykle - jak div - oraz semantyczne - jak pozostałe wymienione. Przydatne są również style CSS dla pozycjonowania poprzez: flex, float, display, width, height, padding czy float. Ponadto Ionic framework udostępnia wysoce przydatne komponenty dla pozycjonowania (Rys. 6), takie jak:

- Grids (rys. 6) – system pozycjonowania oparty o CSS-owy flexbox;
- Tabs – możliwość przejścia pomiędzy widokami strony poprzez tekstowe lub graficzne (rys. 6) zakładki
- Slides (rys. 6) – mechanizm przechodzenia pomiędzy częściami strony jak pomiędzy slajdami, pozwalający zwiększyć łączną liczbę wyświetlanych elementów na stronie poprzez podział ich na slajdy.



Rys. 6 Przykłady komponentów dla pozycjonowania interfejsu użytkownika

## HTML, CSS i JavaScript

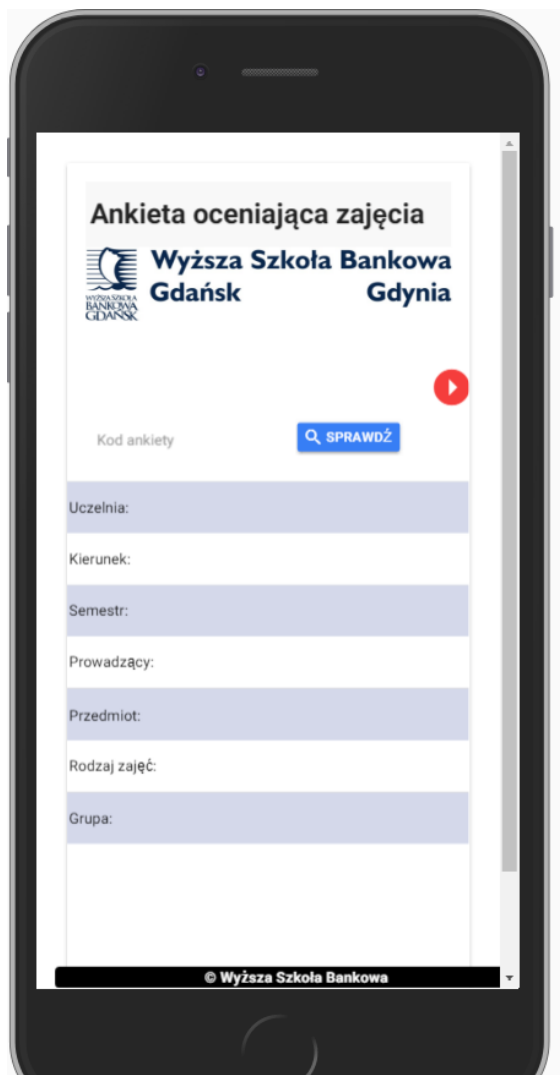
Mobilne aplikacje webowe mogą stosować tagi HTML w zakresie wyświetlania: tekstu, linków, kontrolki czy multimediów jak obrazki. Kod poniżej spowoduje wyświetlenie obrazka na stronie:

```

```

Kod HTML, wraz z notacją Ionic definiowania kontrolki i komponentów, stanowi podstawę tworzenia webowych aplikacji mobilnych (Rys. 7). Po kompilacji webowej aplikacji mobilnej jako natywnej mobilnej jest on w odpowiedni sposób zamieniany na kod natywny, tak aby elementy HTML były również w odpowiedni sposób wyświetlane na ekranach aplikacji mobilnej – np.





Rys. 7 Integracja HTML w hybrydowej aplikacji Ionic

Arkusze stylów CSS mogą być stosowane do zmiany wyglądu elementów: rozmiaru, czcionek, marginesów, obramowań czy kolorystyki. Analogicznie jako kod HTML zostaną one odpowiednio skompilowane do kodu wynikowego (np. Java) w przypadku przygotowania pakietu dla natywnej aplikacji mobilnej.

W aplikacjach Ionic można programować interakcje za pomocą języków TypeScript (jest kompilowany do JavaScript), jak i bezpośrednio w JavaScript. Co ważne korzystać można z różnych użytecznych bibliotek JS, w tym takich jak jQuery, które wprowadzają własne symbole notacji dla kodu. W trakcie kompilacji kodu do wersji natywnej będzie on odpowiednio transformowany do kodu natywnego platformy – Javy w przypadku Android-a czy Swift dla iOS.






















### Ikony Ionic

W aplikacjach często stosowane są ikony, chociażby dla przycisków – na rysunku powyżej są to lupa dla przycisku wyszukiwania oraz strzałka w prawo dla przejścia do slajdu obok. W tym celu można wstawić pobrane ikony z Internetu. Podstawowymi ograniczeniami takiego rozwiązania mogą być:

- spójność wizualna poszczególnych ikon,
- koszty zakupu stosownych pakietów ikon,

- adaptacja wyglądu ikon w zależności od platformy, na której wyświetlana jest mobilna aplikacja webowa, albo do której kompilowany jest projekt jako natywna aplikacja mobilna.

Szczególnie trzeci z problemów wymaga znacznych nakładów pracy dla jego wyeliminowania, czyli posiadania wersji ikon zgodnych z typografią poszczególnych platform oraz kodu, który zapewniłby wyświetlanie odpowiedniego zestawu graficznego. Dlatego pomocne są ikony dostępne w ramach Ionic Icons, gdzie dostępnych jest kilkaset ikon w różnych wersjach kolorystycznych z automatycznym wyborem wersji w zależności od platformy, co przedstawia Rys. 8.

Name	iOS	iOS-Outline	Material Design
add			
add-circle			
alarm			
albums			
alert			
american-football			
analytics			

Rys. 8 Ikony Ionic Framework

### Pobieranie danych z formatki

Aby metody obsługujące zdarzenia mogły poprawnie działać, muszą mieć dostęp do danych wprowadzanych przez użytkowników na stronie. W tym celu stosowany jest mechanizm bindingu, polegający nie na pobieraniu danych poprzez kod, ale deklaracyjnym wskazaniu ich źródła – ich pobranie oraz wypisywanie odbywać się będzie automatycznie dzięki wsparciu frameworka. Dostępne są 2 formy bindingu różniące się od siebie kierunkiem „komunikacji”:

- Jednostronny do odczytu (ang. one-way) – kontrolki pobierają dane i wyświetlają je, bez przesłania zmian wartości do źródła. Można w tym zakresie zastosować 2 notacje:

- Poprzez atrybut ng-bind:

```
<span ng-bind="RRSO"></span>
```

- Poprzez interpolację jako tzw. symbol wąsów (ang. mustache) – np.

```
<ion-label>RRSO kredytu wynosi: {{RRSO}}</ion-label>
```

```

```

Zgodnie z przykładami powyżej wartości mogą być zarówno wyświetlane jako treść, jak i przypisywane do atrybutów. Interpolacja może stosować również wyrażenia matematyczne np.

```
<ion-label>Wynagrodzenie wynosi: {{pensja + procPremii * pensja}}</ion-label>
```



- Obustronny (ang. two-way) – kontrolki pobierają dane i wyświetlają je, a zmiany wartości w nich, powodują aktualizację źródła danych. W tym celu w kontrolce stosuje się atrybut [(ngModel)] z przypisanie nazwy pola w kodzie, np.

```
<ion-list>
  <ion-item>
    <ion-label fixed>Username</ion-label>
    <ion-input type="number" value="0.0" [(ngModel)]="kwotaKredytu"></ion-input>
  </ion-item>
</ion-list>
```

Nazwa, do której jako binding danych występuje odwołanie - np. kwotaKredytu w przykładzie powyżej - musi zostać uprzednio zadeklarowana w klasie obsługującej stronę, jako pole globalne, tzn. zmienna o najwyższym poziomie widoczności, np.

```
class KalkulatorKredytu {
  kwotaKredytu : number = 0.0;
  oprocentowanieKredytu : number = 7.8;
  RRSO: number = 0.0;
}
```

## Zadania do realizacji

1. W utworzonym pustym projekcie zdefiniuj, aby strona domyślna wyświetlała zbliżony do poniższego interfejs użytkownika – ankieta ewaluacji zajęć.

1. Na zajęciach była przekazywana wiedza zarówno teoretyczna, jak i praktyczna. ▼

2. Zajęcia miały precyzyjne określone założenia i cele. ▼

3. Zajęcia były prowadzone w sposób jasny i zrozumiały. ▼

4. Forma zajęć, wykorzystywane ćwiczenia oraz przykłady, angażowały uczestników. ▼

5. Wykładowca dokonywał syntez i podsumowań partii materiału. ▼

6. Wykładowca potrafił zinteresować słuchaczy omawianym tematem. ▼

7. Zajęcia realizowane były w przyjaznej i budującej atmosferze. ▼

[Uwagi dodatkowe i komentarze \(opcjonalne\).](#)

Maksymalnie 140 znaków.

/

WYŚLIJ

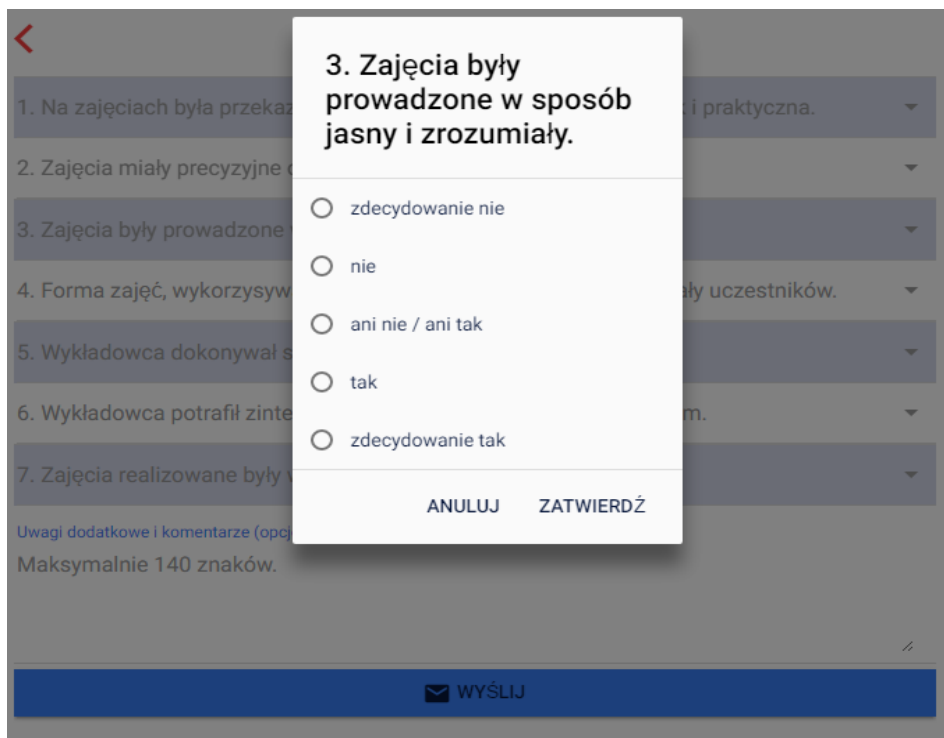
2. Dla przycisku „wyślij” dodaj ikonę zgodnie z rysunkiem powyżej.

Projekt: „Zaprogramowani na sukces – program kształcenia praktycznego na kierunku Informatyka w Wyższej Szkole Bankowej w Gdańsku”  
współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego

POWR.03.01.00-00-N010/16



3. Dodaj, aby strona wyświetlała logo uczelni, w której studiujesz.
4. Napisz kod powodujący pobieranie danych z wypełnionej ankiety poprzez binding.



3. Zajęcia były prowadzone w sposób jasny i zrozumiały.

☐ zdecydowanie nie

☐ nie

☐ ani nie / ani tak

☐ tak

☐ zdecydowanie tak

ANULUJ ZATWIERDŹ

WYŚLIJ

### Pytania kontrolne

1. Wymień komponenty dostępne we framework-u Ionic ułatwiające definiowanie układu witryny.
2. Jakie znasz typy bindingów i jak można je zdefiniować dla kontrolek.

### Praca własna studenta

1. Zadanie 2 i 3.

### Literatura

1. Griffith C., Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova, O'Reilly Media 2017
2. Kontrolki i komponenty Ionic - <http://ionicframework.com/docs/components>
3. HTML - <http://www.w3schools.com/html>
4. Kurs CSS - <https://www.w3schools.com/css>
5. Kurs JavaScript - <https://www.w3schools.com/js>
6. Kurs TypeScript - <https://www.tutorialspoint.com/typescript>
7. Kurs jQuery - <https://www.w3schools.com/jquery>
8. Angular - <https://angular.io/docs/ts/latest/guide/template-syntax.html>
9. Projektowanie aplikacji w Ionic - <http://gonehybrid.com/build-your-first-mobile-app-with-ionic-2-angular-2-part-4>

## Temat 6: Adaptacja wyglądu aplikacji

### Wprowadzenie teoretyczne i przykłady

Wygląd aplikacji można adaptować w zakresie:

- statycznym – zmiana kolorystyki względem standardowych szablonów,
- dynamicznym – wyświetlanie powiadomień, alertów czy blokowanie interfejsu na czas ładowania danych.

### Modyfikacja szablonów wizualnych

Wygląd aplikacji mobilnych tworzonych we frameworku Ionic oparty jest o preprocesory SASS. Szybkim sposobem wdrożenia nowego wzorca wizualnego jest jego zakup w ramach witryny <https://market.ionic.io/themes>, co wiąże się jednak z poniesieniem kosztów. Innym rozwiązaniem jest modyfikacja domyślnego wzorca oraz poszczególnych stron. W zakresie formatowania tekstu dostępnych jest szereg atrybutów w tym zakresie, do: wyrównania, przekształcania, pozycjonowania czy zapewnienia responsywności, np.

```
<div text-uppercase>
  text-uppercase
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed ac vehicula lorem.
</div>
```

Powyższy kod powoduje wyświetlenie wszystkich liter z dużej:

```
TEXT-UPPERCASE
LOREM IPSUM DOLOR SIT AMET, CONSECTETUR
ADIPISCING ELIT. SED AC VEHICULA LOREM.
```

Podstawowym sposobem modyfikacji domyślnego szablonu wizualnego jest zmiana wartości poszczególnych zmiennych odpowiedzialnych m. in. za kolorystykę. W tym zakresie w pliku [src/theme/variables.scss](#) wartości domyślne należy zmodyfikować na odpowiednie, np.

```
$colors: (
  primary: #488aff,
  secondary: #32db64,
  danger: #f53d3d,
  light: #f4f4f4,
  dark: #222
);
```

Zmienna primary określa podstawowy kolor, w którego tonacji będzie wyświetlany interfejs.

Ponadto można zdefiniować własne kolory, np.

```
$colors: (
  primary: #488aff,
  ...
  twitter: (
    base: #55acee,
    contrast: #ffffff
  )
);
```

Nowa zmienna jest nazwą koloru, który można przypisać w kontrolce, np.

```
<button ion-button color="twitter">I'm a button</button>
```

Zmiennych, poprzez które w pliku `src/theme/variables.scss` można określić wygląd witryny, jest kilkadziesiąt, np. `$alert-ios-button-text-color` określa kolor przycisku w oknie alert w systemie iOS, gdzie domyślna wartość to `color($colors-ios, primary)`.

Ponadto zgodnie z zasadami SCSS można stworzyć własne zmienne i wielokrotnie wykorzystać je dla określania stylu poszczególnych elementów, np.

`$control-height: 40px;`

```
.header {
  height: $control-height;
}
.sub-header {
  height: $control-height;
}
```

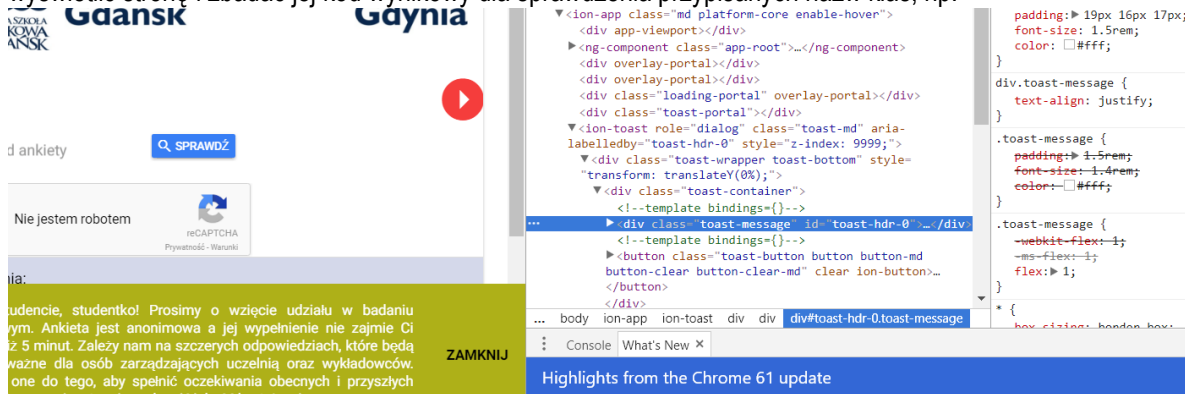
Zmiany w pliku `src/theme/variables.scss` dotyczą całej aplikacji, a zatem każdej ze stron. Dla określenia wyglądu pojedynczej strony należy dodać stosowne wpisy SCSS/CSS w jej pliku scss.

Framework Ionic zapewnia dopasowanie wyglądu witryny/aplikacji zależnie od platformy, w której jest wyświetlana/uruchomiona. Jeśli wygląd na wszystkich platformach ma być identyczny, można przypisać wartość dla strony głównej poprzez `<ion-app class="...">` gdzie możliwe wartości to:

- ios – iOS
- md – Android
- wp – Windows

## Pełna adaptacja wyglądu kontrolki i komponentów

Jeśli wygląd komponentu lub kontrolki nie jest możliwy do adaptacji poprzez zmienne SCSS, zmiana wyglądu możliwa jest w pliku scss strony z nimi jako wpisy CSS. W tym celu uprzednio należy w przeglądarce wyświetlić stronę i zbadać jej kod wynikowy dla sprawdzenia przypisanych nazw klas, np.



Rys. 9 Przykład analizy kodu dla adaptacji interfejsu użytkownika

## Wyświetlanie powiadomień i alertów

W trakcie użytkowania aplikacji ze względu na ograniczony rozmiar ekranu użytkownika stosowne informacje dobrze jest wyświetlać wyłącznie tymczasowo. Np. zamiast na stronie wyświetlać informacje o konieczności sumiennego wypełnienia ankiety, które zajmowałyby znaczny procent ekranu, lepszym rozwiązaniem jest okresowe wyświetlenie powiadomienia (ang. Toast). W tym celu należy zaimportować klasę `ToastController` i zadeklarować jej obiekt w konstruktorze:

```
import { ToastController } from 'ionic-angular';
```

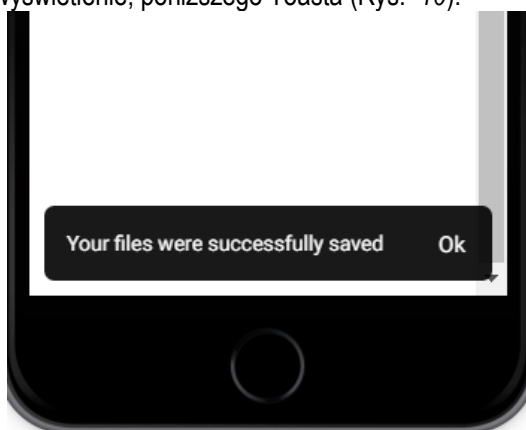


```
export class MyPage {  
  constructor(public toastCtrl: ToastController) {  
  }  
}
```

Toast można utworzyć za pomocą metody create z parametrami jak ma on wyglądać i jaką posiadać treść. Do wyświetlenia służy metoda present:

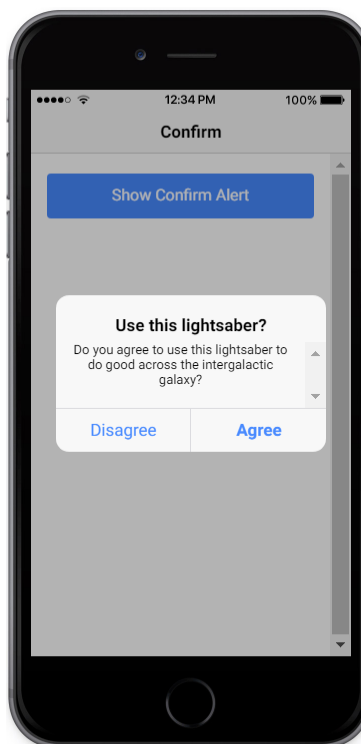
```
presentToast() {  
  let toast = this.toastCtrl.create({  
    message: 'Your files were successfully saved',  
    duration: 3000,  
    showCloseButton: true,  
    position: 'bottom'  
  });  
  toast.present();  
}
```

Powyższy kod powoduje wyświetlenie, poniższego Toasta (Rys. 10).



Rys. 10 Przykład Toast-a

Drugim sposobem szybkiego poinformowania użytkownika, w szczególności o sytuacjach niepożądanych, są alerty pokazujące komunikaty na cały ekran urządzenia mobilnego. Alerty, oprócz prezentowania informacji, są przydatnym komponentem dla podejmowania decyzji przez użytkownika jak na Rys. 11.



Rys. 11 Przykład alert-a w Ionic Framework

Analogicznie jak w przypadku toastów, konieczny jest import stosownej klasy, zadeklarowanie jej obiektu, zdefiniowanie wyglądu alertu i jego wyświetlenie, co prezentuje poniższy kod.

```
import { AlertController } from 'ionic-angular';
export class MyPage {
  constructor(public alertCtrl: AlertController) {
  }

  showConfirm() {
    let confirm = this.alertCtrl.create({
      title: 'Use this lightsaber?',
      message: 'Do you agree to use this lightsaber to do good across the intergalactic galaxy?',
      buttons: [
        {
          text: 'Disagree',
          handler: () => {
            console.log('Disagree clicked');
          }
        },
        {
          text: 'Agree',
          handler: () => {
            console.log('Agree clicked');
          }
        }
      ]
    });
    confirm.present();
  }
}
```



## Blokowanie interfejsu użytkownika

W wybranych sytuacjach, jak pobieranie czy przysyłanie danych, użytkownik powinien być o tym informowany, a interfejs na ten czas blokowany. Do tego celu służy komponent loading, opisany w temacie 8.

## Zadania do realizacji

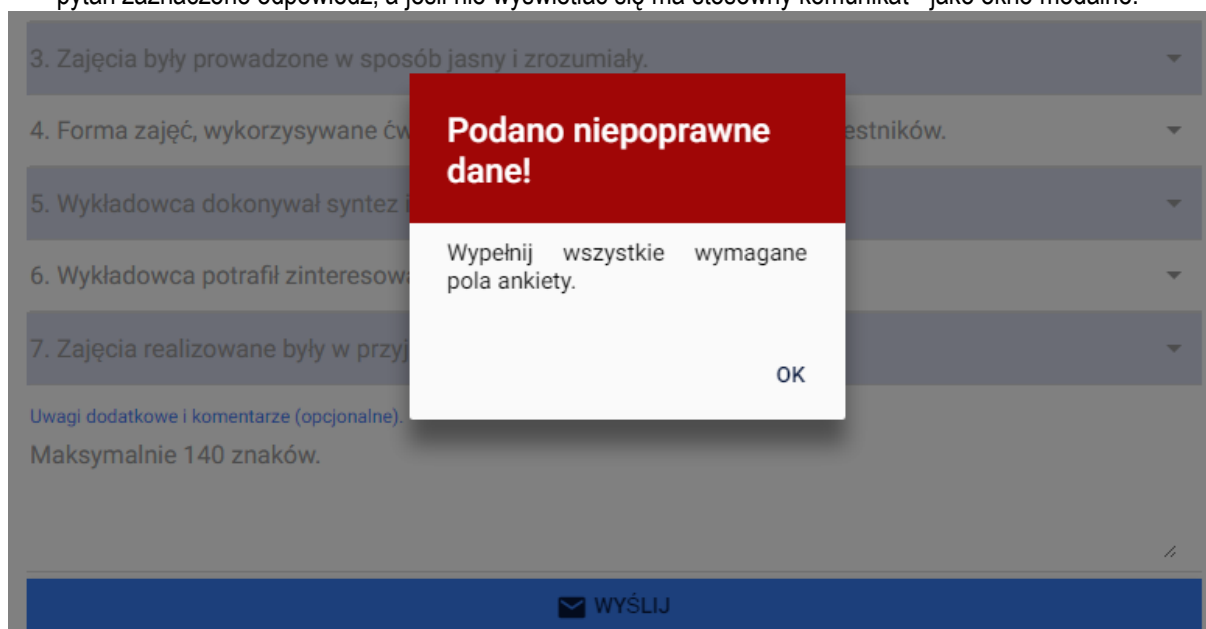
1. Dodaj, aby dla strony ankiety wyświetlało się powiadomienie o poniższej treści.

Proszę sumiennie i rzetelnie ocenić zajęcia, na których się obecnie Pani/Pan znajduje. Udzielone odpowiedzi są ważnym elementem przyczyni się do doskonalenia procesu dydaktycznego w

Nazwa Uczelni

ZAMKNIJ

2. Zdefiniuj obsługę zdarzenia kliknięcia na przycisk „Wyślij”- napisz kod sprawdzający czy dla wszystkich pytań zaznaczono odpowiedź, a jeśli nie wyświetlać się ma stosowny komunikat - jako okno modalne.



3. Zajęcia były prowadzone w sposób jasny i zrozumiały.

4. Forma zajęć, wykorzystywane ćwiczenia i zadania były interesujące dla uczestników.

5. Wykładowca dokonywał syntez i podsumowań.

6. Wykładowca potrafił zinteresować i zaangażować uczestników.

7. Zajęcia realizowane były w przyjaznej atmosferze.

Uwagi dodatkowe i komentarze (opcjonalne):  
Maksymalnie 140 znaków.

**Podano niepoprawne dane!**

Wypełnij wszystkie wymagane pola ankiety.

OK

WYŚLIJ

3. Zdefiniuj w obsłudze zdarzenia kliknięcia na przycisk „Wyślij” kod sprawdzający, że w kontrolce uwagi nie ma tekstu lub ma on mniej niż 140 znaków.
4. Zdefiniuj za pomocą zmiennych SCSS szablon aplikacji w tonacji zielonej.
5. Zdefiniuj własną zmienną w szablonie definiowania stylu elementów, jako określającą kolorystykę w tonacji brązowej. Wdróż własny styl dla przycisku Wyślij.
6. Ze strony <https://github.com/ionic-team/ionic/blob/master/src/themes/ionic.theme.dark.scss> pobierz styl dla wersji w tonacji ciemnej, sprawdź wygląd w niej aplikacji. Następnie wróć do wersji wcześniejszej.
7. Zdefiniuj, aby okno z komunikatem (ang. tile) z zadania 1, miało wygląd zgodny z rysunkiem z tego zadania.
8. Określ, aby okno z alertem z zadania 2, miało wygląd zgodny z rysunkiem z tego zadania.

## Pytania kontrolne

1. W ramach aplikacji często przez dłuższy czas pobierane są dane. Jaki komponent do tego celu powinienes zastosować?
2. W którym pliku i poprzez jaki element modyfikuje się domyślny wygląd aplikacji mobilnej?



3. W jaki sposób można określić dla aplikacji, aby miała zawsze implementowany wygląd material design, nawet jeśli będzie kompilowana i uruchamiana na platformie iOS?
4. W jakim pliku definiowany jest indywidualny wygląd strony?

### **Praca własna studenta**

1. Zadania 3, 5, 6 i 8.

### **Literatura**

1. Griffith C., Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova, O'Reilly Media 2017
2. Toast - <https://ionicframework.com/docs/api/components/toast/ToastController>
3. Alert - <https://ionicframework.com/docs/api/components/alert/AlertController>
4. Projektowanie i modyfikacja wzorców wizualnych - <https://ionicframework.com/docs/theming>
5. SASS - <http://sass-lang.com/guide>

## Temat 7: Projektowanie warstwy dostępu do danych

### Wprowadzenie teoretyczne i przykłady

Wiele aplikacji mobilnych wymaga dostępu do danych. W tym celu konieczne jest przygotowanie stosownego back-endu pozwalającego na pobieranie i przysyłanie danych do centralnej lokalizacji jak baza danych. W tym zakresie można skorzystać z usług mobile services dla chmury lub przygotować własne rozwiązanie oparte o architekturę REST, co przedstawia bieżący temat, gdzie przygotowanie stosowanego rozwiązania oparto o Microsoft SQL Server i ASP .NET Web API.

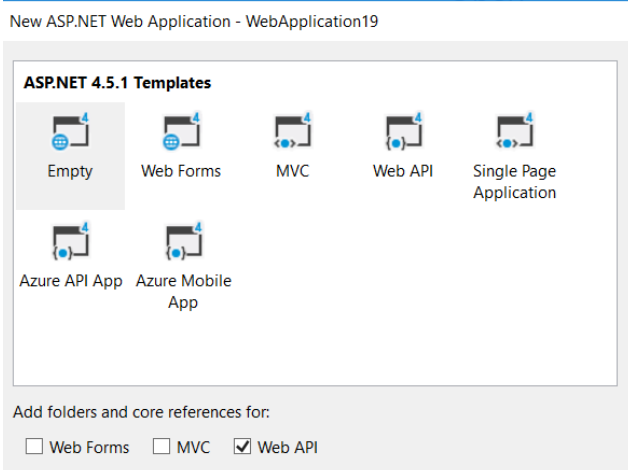
### Utworzenie nowej bazy danych dla aplikacji

Utworzenie bazy danych w MS SQL Server wymaga:

1. Zainstalowania MS SQL Server
2. Dodania nowej bazy danych – w narzędziu Microsoft SQL Server Management Studio operacja New Database
3. Zdefiniowania tabel w bazie danych – w bazie danych, w sekcji Tables, operacja New Table.
4. Dodania danych do tabeli – na nazwie tabeli operacja Edit Top ...

### Zdefiniowanie modelu danych

Aby przygotować warstwę back-end pozwalającą na komunikację z bazą danych aplikacji mobilnej, należy w Visual Studio utworzyć projekt jako ASP .NET Web Application - jako typ Empty z referencjami do bibliotek Web API, zgodnie z Rys. 12.



Rys. 12 Wybór szablonu projektu jako Web API

W projekcie należy utworzyć model danych poprzez Entity Framework – w tym celu w sekcji models dodawany jest nowy element jako ADO .NET Entity Data Model. W kreatorze kolejno konfigurowane są:

1. Połączenie z bazą danych
2. Encje modelu na podstawie tabel w bazie

Po zakończeniu generowania modelu tworzone są klasy - w tym kluczowa - będąca pośrednikiem w dostępie do bazy o kodzie analogicznym do poniższego:

```
using System;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
public partial class ankietyWSBGdyniaEntities : DbContext
{
    public ankietyWSBGdyniaEntities() : base("name=ankietyWSBGdyniaEntities")
    {
    }
}
```



...

### Utworzenie kontrolera komunikacji REST

Poprzez model możliwa jest wyłącznie komunikacja z bazą danych. Aby aplikacja mobilna miała dostęp do bazy, musi zostać przygotowany i uruchomiony jako witryna, kontroler REST (Representational State Transfer). Pozwala on witrynom mobilnym oraz natywnym aplikacjom mobilnym na dostęp poprzez protokół http i stosowne nagłówki:

- GET – pobierać dane
- POST – dodawać nowe dane, np. produkt
- PUT – modyfikować dane
- DELETE – usuwać dane

Kontroler dodawany jest poprzez operację Add Controller w sekcji Controllers projektu. Następnie w odpowiednich metodach w języku C# jako LINQ (Language Integrated Queries) należy napisać kod pozwalający na pobieranie lub przysyłanie danych z/do aplikacji mobilnej. Np.

```
namespace Backend.Controllers
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    public class ZajeciaController : ApiController
    {
        // GET: api/Zajecia/111111
        public Models.Zajecia Get(int id) //id=111111
        {
            var db = new Models.ankietyWSBGdyniaEntities();
            return db.Zajecia.Single(z=>z.kodAnkiety == id);
        }
    }
}
```

Żądanie do serwisu ma formę adresu url, np. `http://.../api/Zajecia/111111`, a zwracane dane w większości przypadków są jako JSON (JavaScript Object notation) – np.

```
{"id":1.0,"przedmiot":"Programowanie sieciowe","prowadzacy":"Michał Kuciapski", "grupa":"ILN4_AS", "kodAnkiety":111111.0}
```

Domyślnie kontrolery Web API (REST) są **dostępne wyłącznie dla aplikacji znajdujących się pod tym samym adresem IP i portem**. W związku z tym, że aplikacje mobilne są uruchamiane na innym urządzeniu, nie będą w stanie komunikować się z warstwą back-end. Dlatego konieczne jest wdrożenie na serwerze z kontrolerami wspierania żądań CORS (Cross-Origin Requests).

### Zadania do realizacji

1. Zdefiniuj bazę danych dla wprowadzania ankiet składającą się z 2 tabel:
  - a. Ankiety – przechowywania danych przesłanych ankiet, czyli odpowiedzi na min. 3 pytania.
  - b. Zajęcia – przechowywania danych zajęć, czyli m. in. kodu ankiety, nazwy przedmiotu, prowadzącego oraz grupy, np. zgodnych z poniższymi



Uczelnia: **WSB w Gdyni**

Kierunek: **INF**

Semestr: **2017 zimowy**

Prowadzący: **Michał Kuciapski**

Przedmiot: **Programowanie rozproszone i w chmurze**

Rodzaj zajęć: **Wykład**

Grupa: **WZ\_INiN4\_wszyscy,WZ\_INiN5\_wszyscy**

2. Zmodyfikuj model danych dostępu do bazy za pomocą Entity Framework-a.
3. Utwórz jako Web API kontroler dla komunikacji z bazą i dodania ankiety.
4. Utwórz jako Web API kontroler dla komunikacji z bazą i pobrania danych dla ankietowanych zajęć na podstawie kodu ankiety.

### Pytania kontrolne

1. Na czym polega podejście architektoniczne REST?
2. Jak jest rola modelu danych w aplikacjach ASP .NET Web API?
3. Aby aplikacja mobilna mogła się łączyć z serwerem z kontrolerami back-end, jakie musi być na nim wdrożone rozwiązanie?

### Praca własna studenta

1. Zadania 1a i 4.

### Literatura

1. Instalacja MS SQL Server - <https://msdn.microsoft.com/pl-pl/library/baza-danych-sql-server--rodzaje-sql-server-zastosowanie-instalacja-i-konfiguracja-w-podstawowym-zakresie.aspx>
2. Utworzenie bazy danych - <https://msdn.microsoft.com/pl-pl/library/tworzenie-bazy-danych-w-sql-server.aspx>
3. Zdefiniowanie tabel - [https://msdn.microsoft.com/pl-pl/library/ms188264\(v=sql.110\).aspx](https://msdn.microsoft.com/pl-pl/library/ms188264(v=sql.110).aspx)
4. Definiowanie warstwy back-end jako ASP .NET MVC WEB API - <https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api>
5. Tutorial Entity Framework - <http://www.entityframeworktutorial.net/EntityFramework5/entity-framework5-introduction.aspx>
6. Wdrażanie CORS - <https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/enabling-cross-origin-requests-in-web-api>

## Temat 8: Komunikacja front-end back-end aplikacji wieloplatformowej

### Wprowadzenie teoretyczne i przykłady

Aplikacja mobilna komunikuje się z warstwą dostępu do danych, czyli REST-owym back-endem, poprzez AJAX (Asynchronous JavaScript and XML). Za pomocą kodu JavaScript w tle (asynchronicznie) przy wsparciu formatu danych jak XML (Extensible Markup Language) czy JSON (JavaScript Object Notation) wykonywana jest komunikacja w zakresie pobierania oraz przesyłania danych.

### Komunikacja aplikacji z warstwą back-end

Napisanie kodu komunikacji z warstwą back-end przyspiesza zastosowanie bibliotek takich jak jQuery. Można ją zaimportować w skrypcie poprzez dodanie do strony index.html wpisu

```
<script src=https://code.jquery.com/jquery-3.2.1.min.js integrity="sha256-  
hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwIAQgxVSNgt4=" crossorigin="anonymous"></script>
```

Biblioteka jQuery Posiada ona m.in. metodę ajax pozwalającą na przesyłanie oraz pobieranie danych, np.

```
$("#sprawdz").click(function () {  
    var kod = $("#kod").val();  
    $.ajax({  
        url: "http://.../api/zajecia/" + kod,  
        success: function (wynik) {  
            $("#grupa").html(wynik.grupa);  
            $("#prowadzacy").html(wynik.prowadzacy);  
            $("#przedmiot").html(wynik.przedmiot);  
        },  
        error: function (x, s, b) {  
            alert(JSON.stringify(b));  
        }  
    });  
});
```

W powyższym kodzie:

- `$("#sprawdz").click` – jest subskrypcją zdarzenia kliknięcia na przycisk
- `$.ajax({` - jest metodą wysłania żądania pobrania danych z serwera (domyślna wersja komunikacji), gdzie metody:
  - `success: function (wynik) {` - wywołuje się w przypadku otrzymania poprawnej odpowiedzi, gdzie w zmiennej wynik znajdują się otrzymane dane;
  - `error: function (x, s, b) {` - jest metodą, która jest wywoływana jeśli zwrócony zostanie błąd.

Jeśli dane mają zostać przesłane na serwer, np. wypełniona ankieta, to konieczne jest określenie metody komunikacji (POST, PUT lub DELETE) oraz przesyłanych danych, np.

```
$("#wysluj").click(function () {  
    var op1 = $("#p1").val();  
    var op2 = $("#p2").val();  
    var ankieta = {  
        Kod: $("#kod").val(),  
        P1: op1,  
        P2: op2  
    };  
    var jsonAnkieta = JSON.stringify(ankieta);
```

```
$ajax({  
  url: "http://localhost:9912/api/ankiety",  
  method: "POST",  
  contentType: "application/json; charset=UTF-8",  
  dataType: "json",  
  data: jsonAnkieta,  
  error: function () { alert("ankieta nie doszła"); },  
  success: function () { alert("ankieta doszła"); }  
});  
});
```

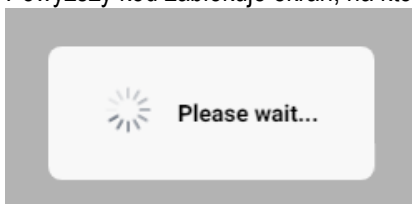
W powyższym kodzie:

- `$("#wyślij").click` – jest subskrypcją zdarzenia kliknięcia na przycisk
- `$ajax({` - jest metodą wysłania żądania wysłania danych do serwera (domyślna wersja komunikacji), gdzie właściwości:
  - `method: "POST"` – oznacza, że wywołanie będzie służyło przesłaniu danych na serwer;
  - `contentType: "application/json; charset=UTF-8"` – wskazuje, że dane są przesyłane w formacie JSON
  - `data: jsonAnkieta` – `jsonAnkieta` jest przesyłanym obiektem z danymi.

W wybranych sytuacjach jak pobieranie czy przysyłanie danych użytkownik powinien być o tym informowany a interfejs na ten czas blokowany. Do tego celu służy komponent `loading`. Analogicznie jak dla `toastów` i `alertów` wymaga obiektów odpowiedniej klasy, w tym przypadku `LoadingController`:

```
import { LoadingController } from 'ionic-angular';  
export class MyPage {  
  constructor(public loadingCtrl: LoadingController) {  
  }  
  presentLoading() {  
    let loader = this.loadingCtrl.create({  
      content: "Please wait...",  
      duration: 3000  
    });  
    loader.present();  
  }  
}
```

Powyższy kod zablokuje ekran, na którym się pojawi, na 3 sekundy, (Rys. 13).



Rys. 13 Komponent Loading

### Strony wielo-widokowe

W przypadku konieczności wyświetlania znacznej liczby danych na stronie internetowej aplikacji mobilnej lub ekranu mobilnej aplikacji natywnej użyteczne są mechanizmy prezentacji ich w ramach przesuwalnych widoków. Ionic framework posiada do tego celu komponent `Slides`, gdzie jeden ekran może się składać z wielu przesuwalnych slajdów – Rys. 14.





```
<ion-slides pager>
  <ion-slide style="background-color: green">
    <h2>Slide 1</h2>
  </ion-slide>
  <ion-slide style="background-color: blue">
    <h2>Slide 2</h2>
  </ion-slide>
  <ion-slide style="background-color: red">
    <h2>Slide 3</h2>
  </ion-slide>
</ion-slides>
```

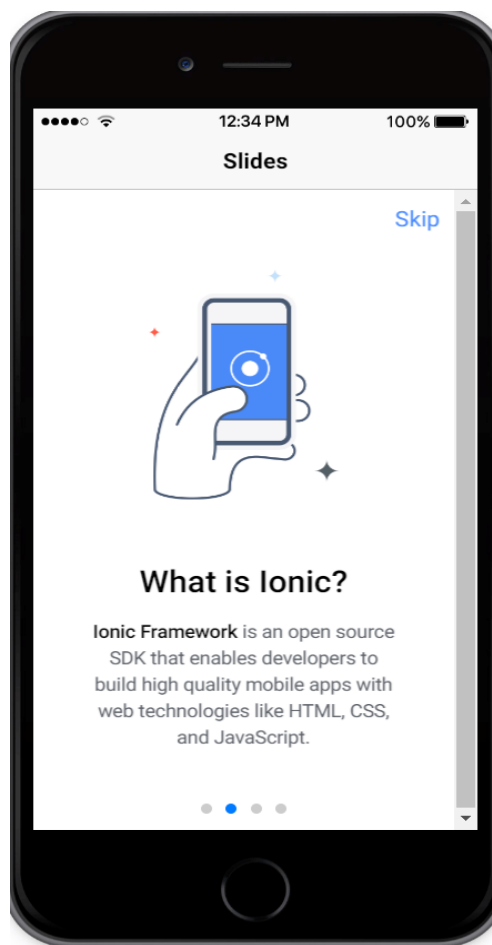
Za pomocą kodu można przechodzić pomiędzy slajdami (czyt. widokami strony) - np. `this.slides.slideTo(2, 500);`

```
import { ViewChild } from '@angular/core';
import { Slides } from 'ionic-angular';

class MyPage {
  @ViewChild(Slides) slides: Slides;

  goToSlide() {
    this.slides.slideTo(2, 500);
  }
}
```

Rys. 14 Komponent Slider



## Zadania do realizacji

1. Dla przycisku „Wyślij” dodaj kod powodujący - jeśli dane są kompletne - przesłanie ich na serwer za pomocą jQuery AJAX. Dane mają zostać przekazane do Web API z tematu 5.
2. Dodaj taką funkcjonalność, aby w trakcie przysyłania danych był wyświetlany stosowny o tym komunikat, który znika po zakończeniu przysyłania danych.
3. Zdefiniuj interfejs użytkownika i funkcjonalność:
  - a. Stronę dla wyświetlania danych o zajęciach, dla których przesyłana jest ankieta z polem podania kodu ankiety oraz przyciskiem sprawdź – wygląd zbliżony z poniższym





Ankieta oceniająca zajęcia

 **Wyższa Szkoła Bankowa  
Gdańsk** **Gdynia**

Kod ankiety  
414670 [SPRAWDŹ](#)

Uczelnia: **WSB w Gdyni**

Kierunek: **INF**

Semestr: **2017 zimowy**

Prowadzący: **Michał Kuciapski**

Przedmiot: **Programowanie rozproszone i w chmurze**


Rodzaj zajęć: **Wykład**

Grupa: **WZ\_INiN4\_wszyscy,WZ\_INiN5\_wszyscy**

- b. Zdarzenie kliknięcia na przycisk „Sprawdź” dodaj kod powodujący pobranie danych ankiety. Na czas pobierania wyświetlać ma się okno powiadamiania o ładowaniu - komponent Loading.
- c. Interfejs przejścia do drugiej strony z możliwością wypełnienia ankiety – komponent Slides. Na pierwszym „slajdzie” możliwość sprawdzenia danych zajęć, a na drugim ankieta do wypełnienia.
- d. Cofnięcie do pierwszego „slajdu” w przypadku jeśli użytkownik chce przejść do wypełniania ankiety, a podał zły jej kod – zgodnie z rys. poniżej.



Ankieta oceniająca zajęcia

 **Wyższa Szkoła Bankowa  
Gdańsk**

**Gdynia**

Kod ankiety  SPRAWDŹ

Uczelnia:

Kierunek:

Semestr:

Prowadzący:

**Podano niepoprawne dane!**

Podaj poprawny kod ankiety (6 cyfr).

OK

### Pytania kontrolne

1. Jaka metoda frameworku jQuery jest przydatna dla komunikacji w tle z serwerem?
2. Aby poprzez jQuery przesłać dane jako POST, jakie właściwości należy określić w obiekcie wywołania żądania?
3. W ramach ekranu aplikacji mobilnej Ionic chcesz wyświetlić szereg elementów interfejsu użytkownika, które nie mieszczą się na jednym ekranie. Jaki komponent dla tego celu będzie przydatny?

### Praca własna studenta

1. Zadania 2 i 3.

### Literatura

1. Griffith C., Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova, O'Reilly Media 2017
2. jQuery AJAX - <http://api.jquery.com/jquery.ajax>
3. Loading - <https://ionicframework.com/docs/api/components/loading/LoadingController>
4. Slides - <https://ionicframework.com/docs/components/#slides> ;  
<https://ionicframework.com/docs/api/components/slides/Slides>

## Temat 9: Programowanie funkcji natywnych dla urządzeń mobilnych

### Wprowadzenie teoretyczne i przykłady

Webowe aplikacje mobilne są wyświetlane w przeglądarce internetowej. Dlatego mogą korzystać z bardzo nielicznych funkcjonalności natywnych urządzeń mobilnych. Są to wyłącznie te, do których dostęp ma przeglądarka, jak np. GPS.

### Instalowanie funkcjonalności obsługi funkcjonalności natywnej

Ionic framework wykorzystuje Ionic Native, który jest wrapperem TypeScript dla wtyczek Cordova / PhoneGap, które ułatwiają dodawanie dowolnej natywnej funkcjonalności do aplikacji mobilnych Ionic. Ionic Native opakowuje wywołania zwrotne (ang. callback) w obiektach Promise oraz Observable, umożliwiając wspólny interfejs dla wszystkich wtyczek i zapewniając, że zdarzenia natywne są wykrywane w kodzie Angular. W ten sposób ułatwia pisanie kodu względem tradycyjnych bibliotek Cordova i PhoneGap.

Aby dodać obsługę Ionic Native do aplikacji, należy uruchomić następujące polecenie w konsoli, które spowoduje zainstalowanie pakietu podstawowego:

```
npm install @ionic-native/core --save
```

Zainstalowanie pakietu podstawowego nie spowoduje dodania do projektu bibliotek wszystkich funkcjonalności natywnych, ze względu na ich liczbę (ok. 100), a przez to i liczbę bibliotek. Przeważnie aplikacja korzysta z kilku z nich. Dlatego zainstalowanie pakietu podstawowego pozwala na dodawanie poszczególnych natywnych funkcjonalności jako wtyczek (ang. plugin).

### Pluginy funkcji natywnych

Aby wdrożyć w projekcie wsparcie dla konkretnej wtyczki – funkcjonalności natywnej urządzenia mobilnego – należy:

1. Zainstalować stosowny pakiet Ionic, np. dla obsługi aparatu:

```
npm install @ionic-native/camera --save
```

2. Zainstalować dla niej bibliotek Apache Cordova, do których odwołuje się plugin Ionic. Dla wskazanej funkcjonalności aparatu:

```
npm install @ionic-native/core --save
```

3. Zarejestrować moduł NgModule w aplikacji, dodając w pliku app.module.ts import biblioteki oraz rejestrując ją na liście provider-ów, np.

```
import { Camera } from '@ionic-native/camera';
```

```
...  
@NgModule({  
  ...  
  providers: [  
    ...  
    Camera  
    ...  
  ]  
})
```

Po wykonaniu powyższych kroków na poszczególnych stronach/ekranach można korzystać z dodanej funkcjonalności mobilnej poprzez:

1. Dodanie importu funkcjonalności natywnej jako klas(y), np.  

```
import { Camera, CameraOptions } from '@ionic-native/camera';
```
2. Utworzenie w konstruktorze obiektu, poprzez który można korzystać z funkcjonalności natywnej, np.  

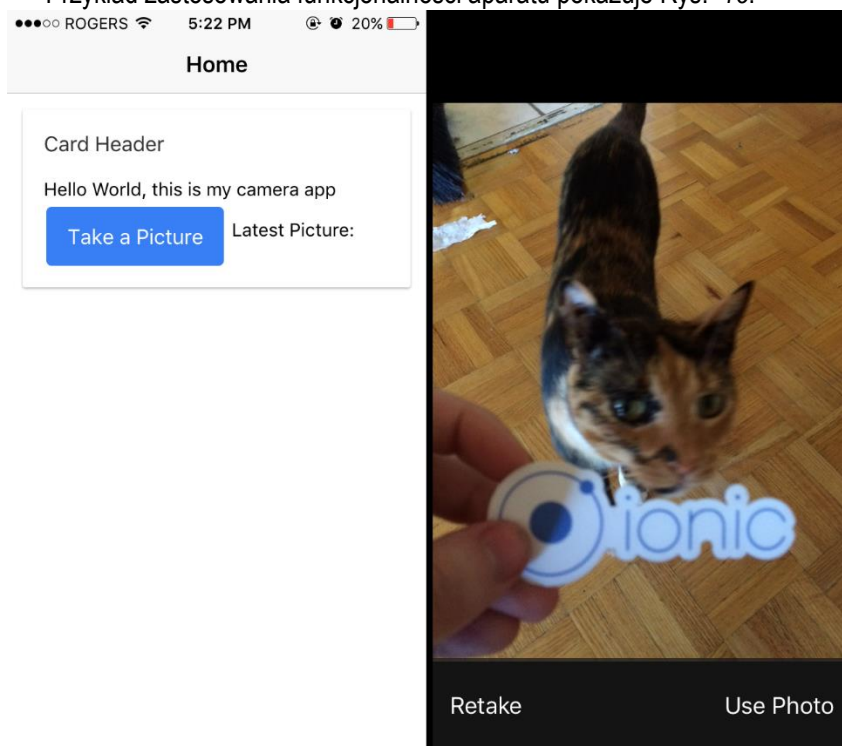
```
constructor(private camera: Camera) { }
```
3. Oprogramowanie funkcjonalności natywnej:

- a. Zdefiniowanie zdarzeń – np. po kliknięciu na przycisk przejście do wykonania zdjęcia
- b. Kod realizacji funkcjonalności natywnej – np. wykonanie zdjęcia i zapisanie do pliku

Poniższy kod powoduje ustawienie parametrów jakości dla robienia zdjęć (`const options: CameraOptions`), przejście do wykonania zdjęcia (`this.camera.getPicture(options).then((imageData) =>)`), a po jego wykonaniu zapisanie go w obiekcie jako danych binarnych (`let base64Image = 'data:image/jpeg;base64,' + imageData;`).

```
const options: CameraOptions = {
  quality: 100,
  destinationType: this.camera.DestinationType.DATA_URL,
  encodingType: this.camera.EncodingType.JPEG,
  mediaType: this.camera.MediaType.PICTURE
}
this.camera.getPicture(options).then((imageData) => {
  let base64Image = 'data:image/jpeg;base64,' + imageData;
}, (err) => {
});
```

Przykład zastosowania funkcjonalności aparatu pokazuje Rys. 15.



Rys. 15 Funkcjonalność natywna w aplikacji hybrydowej - aparat

Źródło: <http://blog.ionic.io/10-minutes-with-ionic-2-using-the-camera-with-ionic-native>

## Zadania do realizacji

1. Za pomocą Ionic i Apache Cordova dodaj funkcjonalność wibracji, jeśli wprowadzony kod ankiety jest błędny.
2. Za pomocą Ionic i Apache Cordova dodaj funkcjonalność wykonania zdjęcia i wyświetlenia go w aplikacji.
3. Za pomocą Ionic i Apache Cordova dodaj funkcjonalność dodania obrazka jako feedbacku ankiety, wybranego z systemu plików.
4. Za pomocą Ionic i Apache Cordova dodaj funkcjonalność wyświetlenia lokalizacji użytkownika.



5. Za pomocą Ionic i Apache Cordova dodaj funkcjonalność dodania informacji do kalendarza o zdarzeniu wypełnienia ankiety.

### **Pytania kontrolne**

1. Jakie funkcjonalności natywne urządzenia może stosować webowa aplikacja mobilna?
2. Co należy zainstalować/wdrożyć/skonfigurować, aby projekt aplikacji Ionic pozwalał na dodanie do niej funkcji mobilnych?
3. Z bibliotek jakiego projektu, korzysta framework Ionic, aby możliwe było dodawanie aplikacji natywnych funkcji urządzeń mobilnych?
4. Wymień pluginy dla natywnych funkcji urządzeń mobilnych oferowane dla projektów Ionic, które znasz.

### **Praca własna studenta**

1. Zadania 3-5.

### **Literatura**

1. Griffith C., Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova, O'Reilly Media 2017
2. Dodanie do projektu wsparcia dla stosowania pluginów funkcji natywnych aplikacji - <https://ionicframework.com/docs/native>, sekcja Overview.
3. Pluginy dla dodawania poszczególnych funkcjonalności natywnych - <https://ionicframework.com/docs/native>
4. Kurs Apache Cordova - <http://www.tutorialspoint.com/cordova>
5. Dokumentacja Apache Cordova - <https://cordova.apache.org/docs/en/latest>

## Temat 10: Wdrażanie aplikacji w urządzeniach mobilnych

### Wprowadzenie teoretyczne

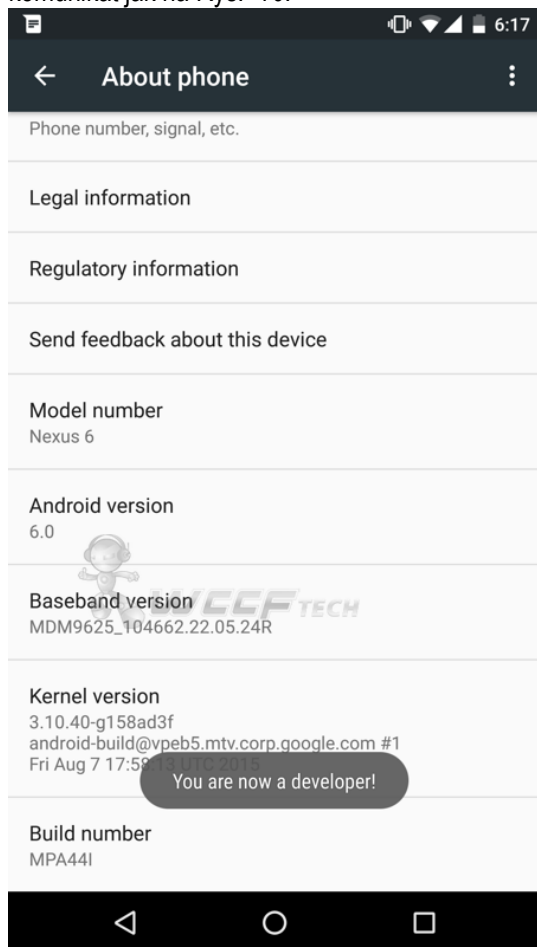
W ramach bieżącego tematu przedstawione zostanie wdrożenie aplikacji na urządzeniach z systemem operacyjnym Android. W podobny sposób możliwe jest przygotowanie pakietu wdrożeniowego aplikacji na systemy operacyjne iOS i Windows Mobile.

### Instalacja SDK dla platformy Android

Aby możliwa była kompilacja aplikacji do systemu urządzenia mobilnego, konieczne jest zainstalowanie jego bibliotek developerskich (ang. Software Development Kit). Nie jest natomiast konieczne instalowanie narzędzi developerskich dla wybranej platformy. W przypadku systemu Android konieczne jest zainstalowanie Android SDK, ze strony <https://developer.android.com/studio/index.html>, gdzie w dolnej jej części możliwe jest wskazanie platformy, dla której chcemy pobrać pakiet instalacyjny. Nie jest zaś konieczne wdrożenie Android Studio.

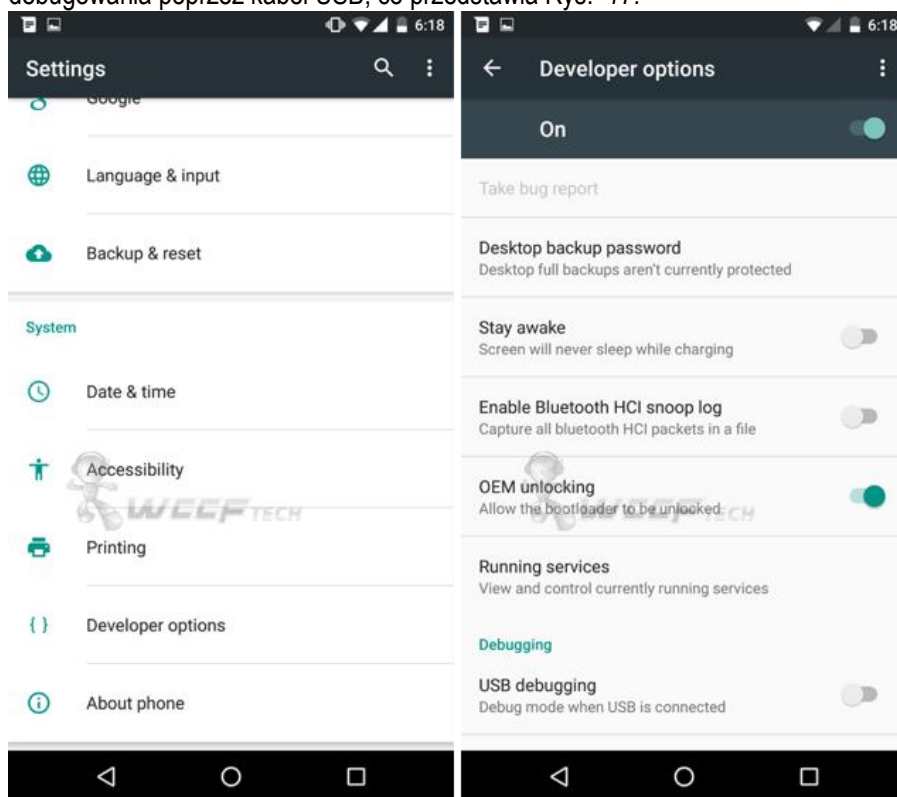
### Wdrożenie aplikacji poprzez kabel USB

Aby możliwe było wdrożenie aplikacji na telefon poprzez kabel USB, w trakcie procesu jej kompilacji konieczne jest włączenie w urządzeniu trybu developerskiego. Sposób zależny jest od wersji systemu Android. W przypadku wersji Android 6.0 Marshmallow, należy w ustawieniach wejść do informacji o telefonie i następnie wielokrotnie (5 lub więcej razy) tapnąć w numer kompilacji. Po włączeniu trybu developera pojawi się stosowny komunikat jak na Rys. 16.



Rys. 16 Włączenie trybu developera w systemie Android

Następnie pojawi się na liście ustawień telefonu sekcja opcji dla developera. W niej należy włączyć tryb debugowania poprzez kabel USB, co przedstawia Rys. 17.



Rys. 17 Włączenie obsługi USB dla wdrożenia aplikacji

Ostatnim krokiem przygotowania się do wdrożenia aplikacji na telefonie poprzez kabel jest instalacja sterowników pozwalających na komunikację komputera z telefonem poprzez kabel USB. Zależy to od systemu operacyjnego oraz jego wersji. Stosowne kroki opisuje dokument „Run Apps on a Hardware Device” (<https://developer.android.com/studio/run/device.html>).

Kompilacja aplikacji do urządzenia mobilnego poprzez kabel USB wykonywana jest poprzez polecenie `ionic cordova run platforma(android/ios)`, np. dla urządzenia z systemem Android:

#### **ionic cordova run android**

W trakcie procesu kompilacji wyświetlane są stosowane informacje o kompilacji, w tym również o błędach, które wystąpiły.

### **Przygotowanie pakietu wdrożeniowego**

Przygotowanie pakietu wdrożeniowego, analogicznie jak bezpośrednie wdrożenie aplikacji w telefonie Android, wymaga uprzedniego zainstalowania Android SDK. Następnie pakiet jest przygotowywany poprzez polecenie `ionic cordova build platforma(android/ios)`, np. dla urządzenia z systemem Android:

#### **ionic cordova build android**

Przygotowany pakiet można opublikować m. in. poprzez:

- Sklep Google Play – wymaga założenia płatnego konta, a publikowane aplikacje poddawane są procesowi weryfikacji;
- Stronę internetową;
- Usługi publikacji w chmurze.





## **Zadania do realizacji**

1. Zainstaluj aplikację w urządzeniu mobilnym poprzez kabel USB.
2. Przygotuj pakiet instalacyjny aplikacji, zamieść go i udostępnij w chmurze, np. Google Drive czy One Drive. Pobierz pakiet na urządzenie i zainstaluj aplikację.

## **Pytania kontrolne**

1. Co należy zrobić, aby w systemie Android włączyć tryb programisty (ang. developer)?
2. Na czym polega różnica w zakresie efektu końcowego, pomiędzy poleceniami ionic cordova run, a ionic cordova build.

## **Praca własna studenta**

1. Zadanie 2.

## **Literatura**

1. Griffith C., Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova, O'Reilly Media 2017
2. Zasoby i dokumentacja dla systemu Android - <https://developer.android.com/index.html>
3. Włączanie trybu developera i ustawianie opcji developerskich - <http://wccfttech.com/enable-developer-options-in-android-6-marshmallow>
4. Kompilacja aplikacji do urządzenie mobilnego poprzez kabel USB – <http://ionicframework.com/docs/cli/cordova/run/>
5. Kompilacja aplikacji jako pakiet dla wdrożenia - <http://ionicframework.com/docs/cli/cordova/build/>
6. Przygotowanie połączenia komputer-telefon poprzez kabel USB - <https://developer.android.com/studio/run/device.html>