

Autorzy	<b>Arkadiusz Rasz 242493</b>
Zespół	2
Grupa zajęć	Poniedziałek 9:15

## Realizowane przypadki użycia

Przeglądaj dane dostępnych odcinków punktowanych

<b>Nazwa</b>	<b>Przeglądaj dane dostępnych odcinków punktowanych</b>
<b>Aktorzy</b>	Administrator
<b>Krótki opis</b>	Przypadek użycia służy do przeglądania dostępnych już odcinków punktowanych przez Administratora w celu weryfikacji danych z rzeczywistymi lub znalezienia możliwości ulepszenia bazy.
<b>Warunki wstępne</b>	Administrator jest zalogowany do systemu.
<b>Warunki końcowe</b>	Każdy odcinek punktowany może być wyświetlony przez Administratora.
<b>Punkty rozszerzeń</b>	-
<b>Scenariusz główny – edycja istniejącego odcinka</b>	<ol style="list-style-type: none"> <li>1. Administrator chce wyświetlić dostępne odcinki punktowane</li> <li>2. System wyświetla stronę główną systemu</li> <li>3. Administrator wybiera z menu nawigacyjnego opcję odcinków punktowanych</li> <li>4. System pobiera dane o odcinkach.</li> <li>5. System wyświetla dane o odcinkach w postaci tabeli.</li> </ol>

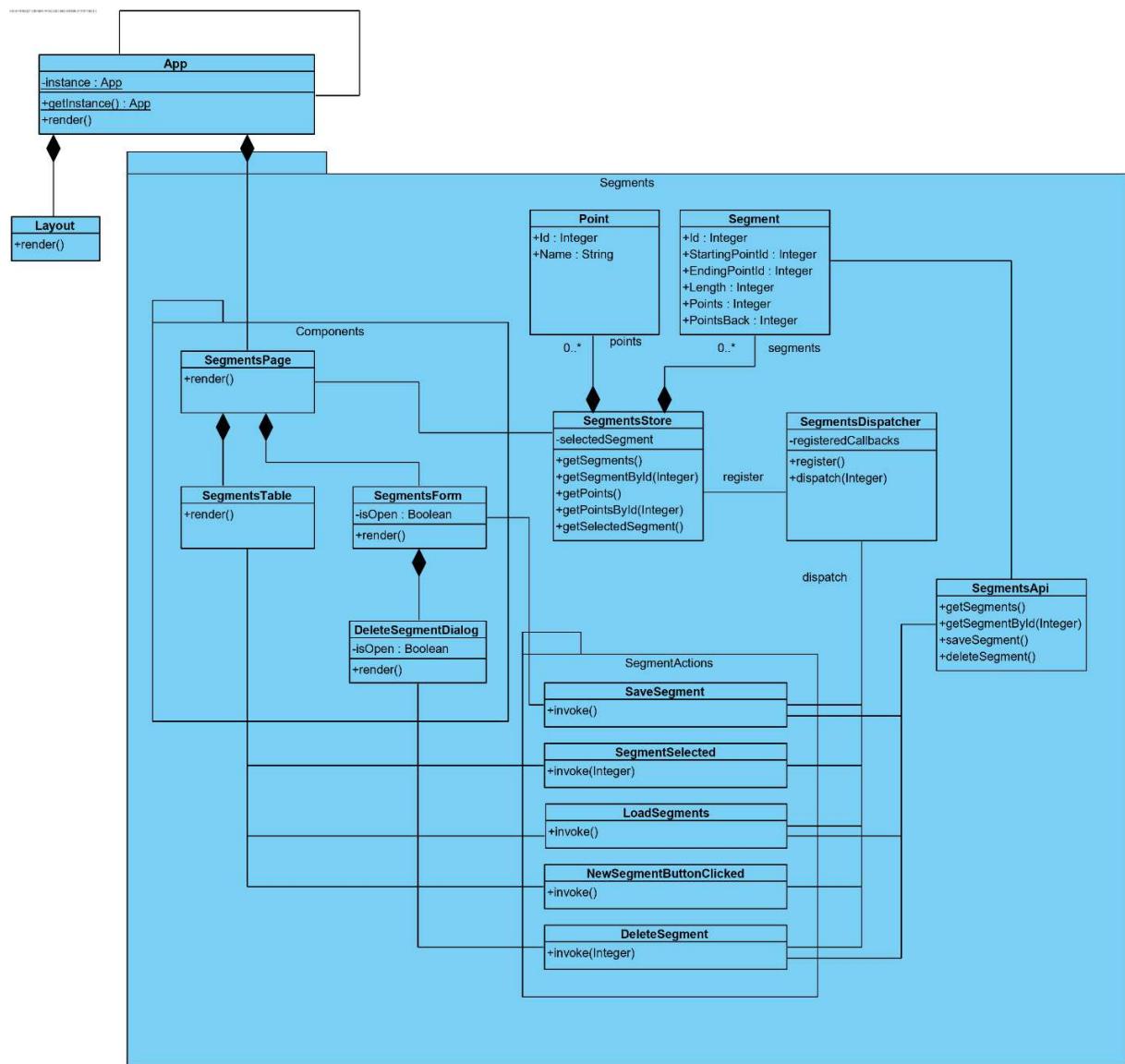
# Realizowane przypadki użycia

## Zarządzaj bazą odcinków punktowanych

<b>Nazwa</b>	<b>Zarządzaj bazą odcinków punktowanych</b>
<b>Aktorzy</b>	Administrator
<b>Krótki opis</b>	Przypadek użycia służy do zarządzania odcinkami punktowanymi tj. dodawaniem nowych odcinków, usuwaniem odcinków czy edycją istniejących odcinków lub punktów za nie należnych. Administrator dokonuje wszystkich zmian na skutek zmiany spisu odcinków punktowanych przez Komisję Turystyki Górskiej.
<b>Warunki wstępne</b>	Administrator jest zalogowany do systemu na konto z uprawnieniami do modyfikacji bazy odcinków punktowanych.
<b>Warunki końcowe</b>	Zmiana wprowadzonych przez administratora danych dotyczących odcinków punktowanych jest zatwierdzona w bazie danych.
<b>Punkty rozszerzeń</b>	-
<b>Scenariusz główny – edycja istniejącego odcinka</b>	<ol style="list-style-type: none"><li>Administrator chce dokonać modyfikacji w bazie.</li><li>System przedstawia tabelę dostępnych odcinków.</li><li>Administrator wybiera jeden z dostępnych już odcinków.</li><li>System zaznacza wybrany odcinek i umożliwia wprowadzanie nowych danych</li><li>Administrator wprowadza dane odcinka.</li><li>System stwierdza poprawność danych.</li><li>System zapisuje zmiany w bazie danych.</li></ol>
<b>Scenariusz alternatywny 1 – dodanie nowego odcinka</b>	<ol style="list-style-type: none"><li>Administrator dodaje nowy odcinek.<ol style="list-style-type: none"><li>Administrator wybiera opcję reprezentującą utworzenie nowego odcinka.</li><li>System tymczasowo tworzy nowy odcinek bez danych.</li></ol>Wejście do kroku 4. scenariusza głównego.</li></ol>
<b>Scenariusz alternatywny 3 – usunięcie istniejącego odcinka</b>	<ol style="list-style-type: none"><li>Administrator wybiera opcję usunięcia wybranego odcinka.<ol style="list-style-type: none"><li>System usuwa odcinek</li></ol>Wejście do kroku 7. scenariusza głównego.</li></ol>
<b>Wyjątek 1 – niepoprawna modyfikacja</b>	<ol style="list-style-type: none"><li>System stwierdza niepoprawność modyfikacji.<ol style="list-style-type: none"><li>System informuje administratora o niemożności wykonania żądanej modyfikacji wraz z informacją o przyczynie niepowodzenia.</li></ol>Wejście do kroku 5. scenariusza głównego.</li></ol>

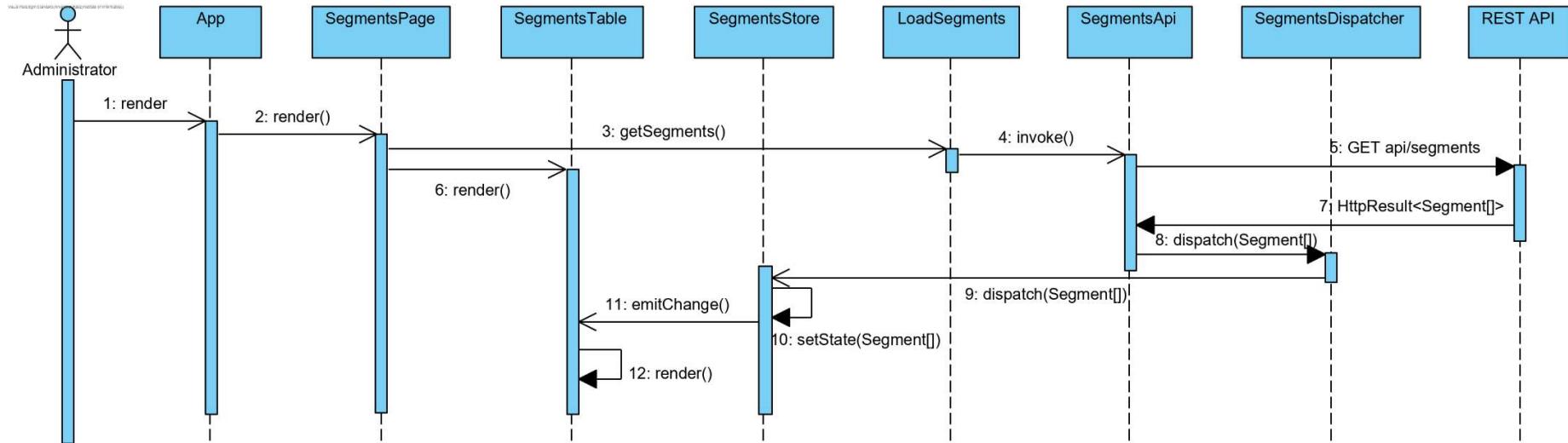
# Diagram klas

Aplikacja webowa do zarządzania bazą odcinków



# Diagram sekwencji

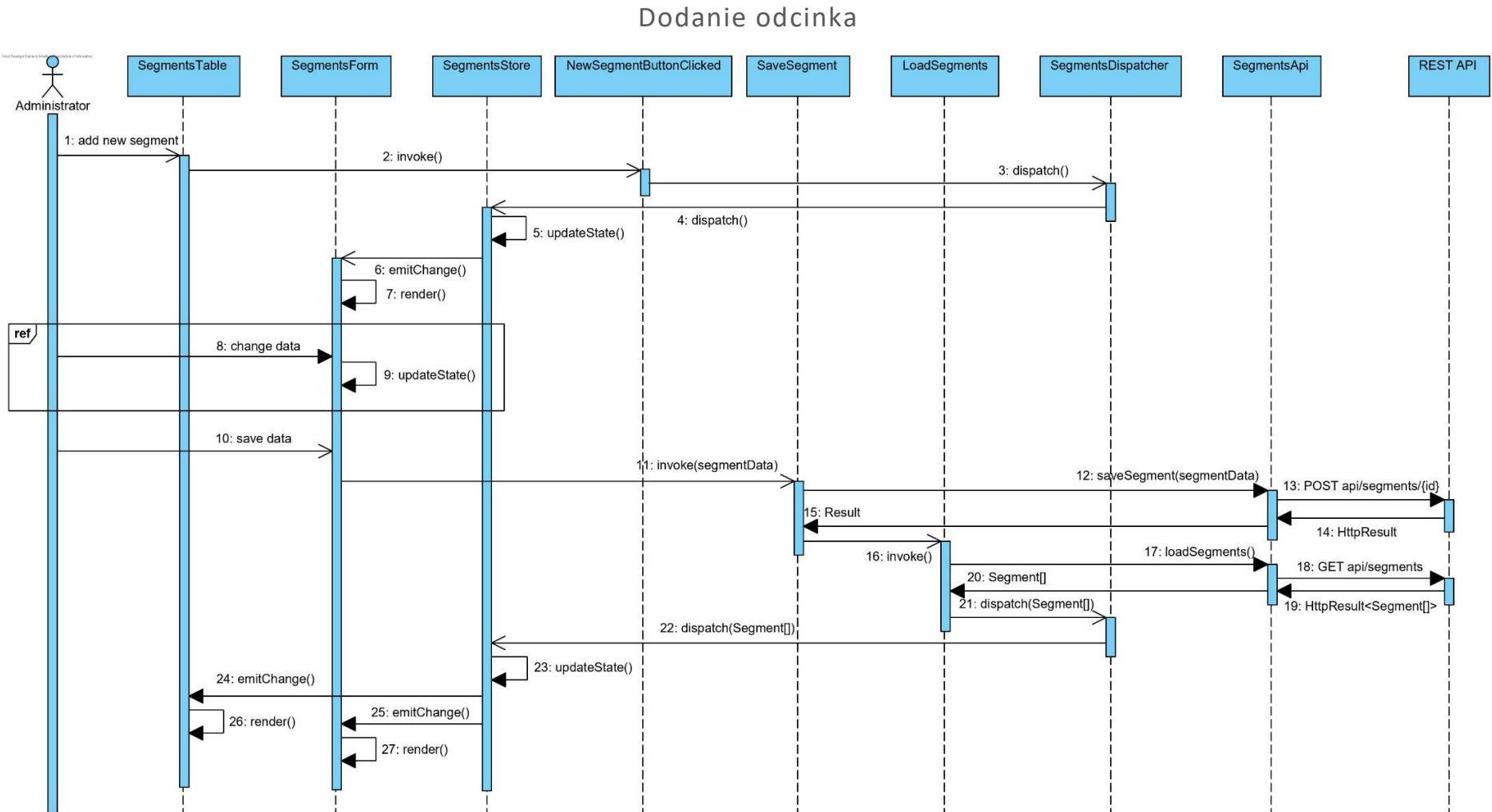
Przeglądaj dane dostępnych odcinków punktowanych



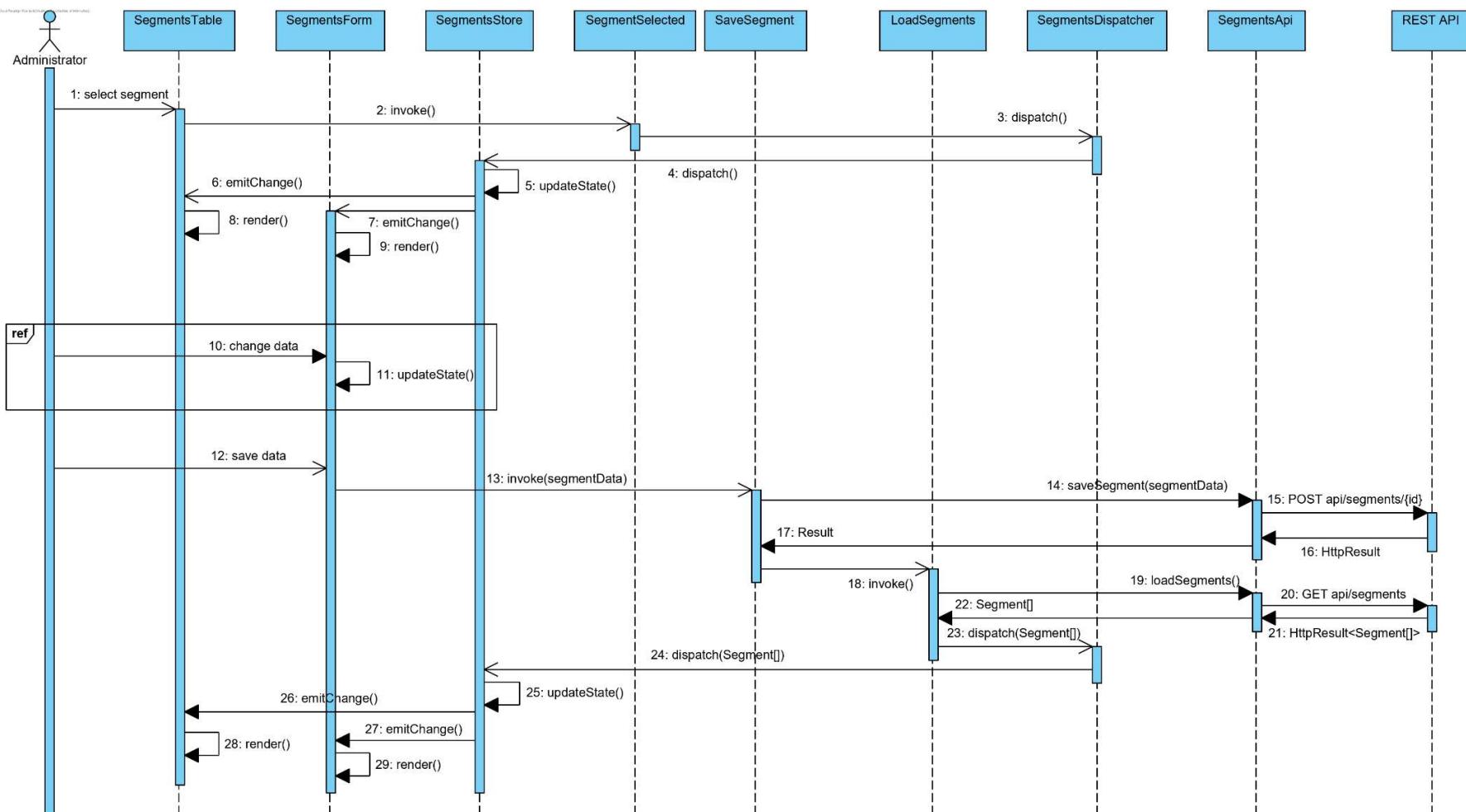
# Diagram sekwencji

Zarządzaj bazą odcinków punktowanych

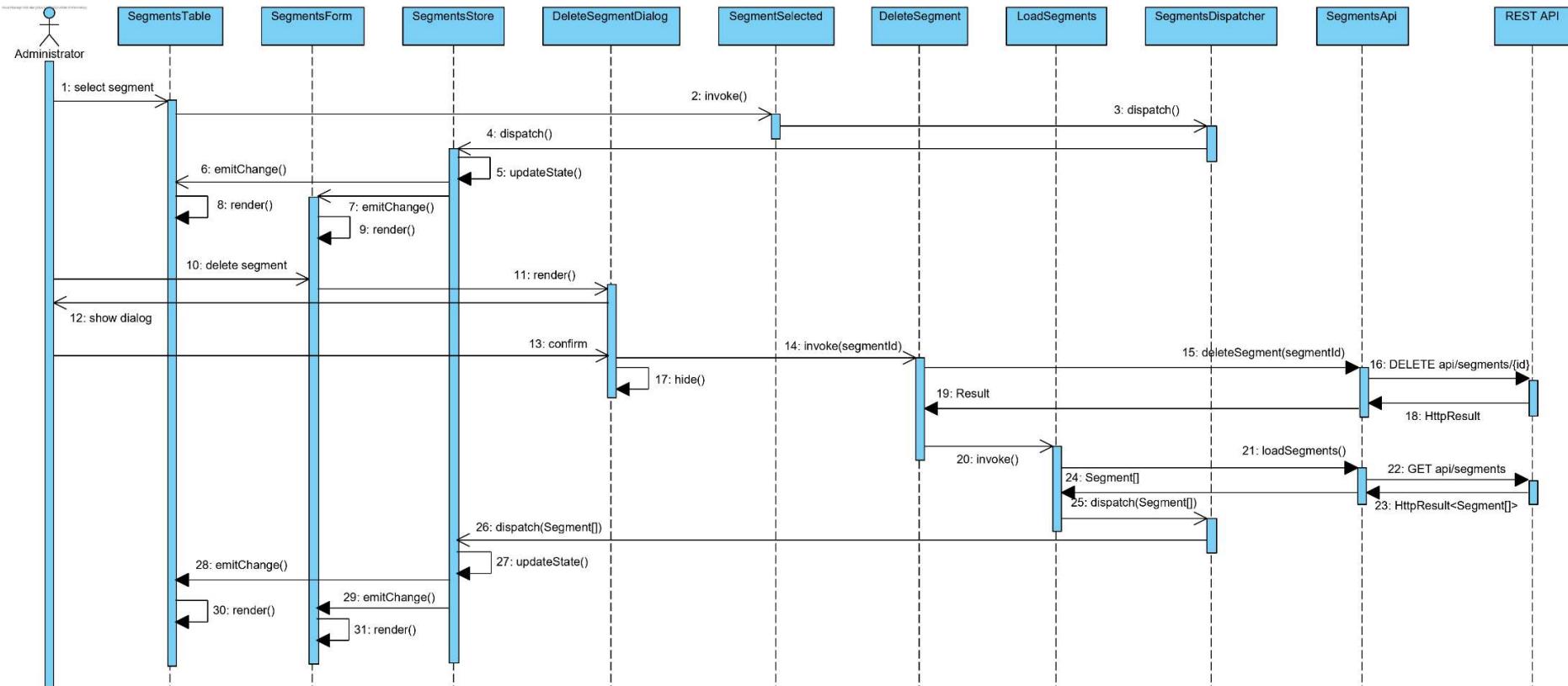
Ze względu na złożoność diagramu sekwencji opisywanego przypadku użycia oraz różności klas uczestniczącym w nich w poszczególnych etapach, diagram został podzielony na trzy części reprezentujące każdy przepływ: dodanie odcinka, edycja odcinka oraz usunięcie odcinka.



## Edycja odcinka



## Usunięcie odcinka



# Interfejs użytkownika

## Na podstawie prototypu

Górsko Odznaka Turystyczna - Administrator

example.user@gmail.com

Odcinki

Punkty

Odcinki punktowane Górskiej Odznaki Turystycznej					DODAJ
Punkt Początkowy	Punkt końcowy	Długość (m)	Punkty GOT	Punkty GOT Powrót	
Rusinowa Polana	Gęśia Szyja	1200	4	1	
Psia Trawka	Rówień Waksmundzka	1500	5	3	
Wodogrzmoty Mickiewicza	Schronisko PTTK nad Morskim Okiem	3400	8	5	
Gęśia Szyja	Schronisko PTTK nad Morskim Okiem	950	3	1	
Schronisko PTTK nad Morskim Okiem	Rusinowa Polana	2300	7	5	

Rows per page: 5 ▾ 1-5 of 7 < >

Interfejs użytkownika został zaimplementowany na podstawie utworzonych wcześniej prototypów z wykorzystaniem komponentów dostępnych w bibliotece material.ui. Elementy te minimalnie różnią się od tych, które dostępne były podczas tworzenia prototypów. Jedyną funkcjonalną różnicą jest przycisk dodawania nowego odcinka widniejący w prawym górnym rogu tabeli z odcinkami. Na poprzednim prototypie użyty został wiersz z pustymi danymi, który pełnił rolę tego przycisku.

Górsko Odznaka Turystyczna - Administrator

example.user@gmail.com

Odcinki

Punkty

Odcinki punktowane Górskiej Odznaki Turystycznej					DODAJ
Punkt Początkowy	Punkt końcowy	Długość (m)	Punkty GOT	Punkty GOT Powrót	
Rusinowa Polana	Gęśia Szyja	1200	4	1	
Psia Trawka	Rówień Waksmundzka	1500	5	3	
Wodogrzmoty Mickiewicza	Schronisko PTTK nad Morskim Okiem	3400	8	5	
Gęśia Szyja	Schronisko PTTK nad Morskim Okiem	950	3	1	
Schronisko PTTK nad Morskim Okiem	Rusinowa Polana	2300	7	5	

Rows per page: 5 ▾ 1-5 of 7 < >

Punkt Początkowy: Wodogrzmoty Mickiewicza Punkt Końcowy: Schronisko PTTK nad Morskim Okiem ZAPISZ

Punkty GOT: 8 Punkty GOT powrót: 5 Długość (m): 3400 ANULUJ

USUŃ ODCINEK

Po wybraniu jednego z dostępnych odcinków, jest on widocznie podświetlony, aby użytkownik miał pewność, który z nich jest edytowany. Punkt początkowy oraz końcowy wybierane są z opcji dostępnych w rozwijanych listach. Pozostałe dane wprowadzane są w odpowiednich polach tekstowych.

Górsko Odznaka Turystyczna - Administrator

Punkt Początkowy	Punkt końcowy	Długość (m)	Punkty GOT	Punkty GOT Powrót
Rusinowa Polana	Gęśla Szyja	1200	4	1
Psia Trawka		1500	5	3
Wodogrzmoty Mickiewicza		3400	8	5
Gęśla Szyja		950	3	1
Schronisko PTTK nad Morskim Okiem		2300	7	5

Rows per page: 5 ▾ 1-5 of 7 < >

Punkt Początkowy: Wodogrzmoty Mickiewicza Punkt Końcowy: Schronisko PTTK nad Morskim Okiem

ZAPISZ ANULUJ USUŃ ODCINEK

Po wybraniu jednego z odcinku jest również opcja usunięcia go. Po wybraniu takiej akcji, użytkownikowi ukazany jest komunikat z potwierdzeniem.

Górsko Odznaka Turystyczna - Administrator

Punkt Początkowy	Punkt końcowy	Długość (m)	Punkty GOT	Punkty GOT Powrót
Rusinowa Polana	Gęśla Szyja	1200	4	1
Psia Trawka	Rówień Waksmundzka	1500	5	3
Gęśla Szyja	Schronisko PTTK nad Morskim Okiem	950	3	1
Rusinowa Polana	Rówień Waksmundzka	2300	7	5

Rows per page: 5 ▾ 1-5 of 7 < >

Punkt Końcowy: Schronisko PTTK nad Morskim Okiem

ZAPISZ ANULUJ USUŃ ODCINEK

Po edycji i zapisaniu danych, są one automatycznie aktualizowane w tabeli bez konieczności odświeżania strony.

# Implementacja aplikacji

## Interfejs użytkownika

Interfejs użytkownika aplikacji został napisany w technologii React z użyciem wzorca architektonicznego Flux. We wzorcu tym przepływ danych jest jednostronny, co pozwala na łatwe rozszerzanie funkcjonalności aplikacji oraz utrzymanie czytelnego kodu.

### Akcje

Wszystkie akcje wywoływane przez interakcję użytkownika z interfejsem reprezentowane są przez funkcje w pliku SegmentActions.js.

```
export function saveSegment(segment) {
  segmentSelected(-1);
  SegmentsApi.saveSegment(segment).then(() => loadSegments());
}

export function segmentSelected(id) {
  Dispatcher.dispatch({
    actionTypes: ActionTypes.SEGMENT_SELECTED,
    id: id
  });
}

export function loadSegments() {
  SegmentsApi.getSegments()
    .then(response => response.json())
    .then(segments => {
      newSegmentId = Math.max(...segments.map(seg => seg.id)) + 1;
      Dispatcher.dispatch({
        actionTypes: ActionTypes.LOAD_SEGMENTS,
        segments: segments
      });
    });
}

export function deleteSegment(id) {
  SegmentsApi.deleteSegment(id).then(response => {
    segmentSelected(-1);
    loadSegments();
  });
}

export function newSegmentButtonClicked() {
  Dispatcher.dispatch({
    actionTypes: ActionTypes.NEW_SEGMENT,
    newSegmentId: newSegmentId
  });
}
```

Funkcje te, w razie konieczności, wywołują połączenie z API umieszczonym na serwerze za pomocą funkcjonalności zawartej w SegmentsApi.js. Po otrzymaniu niezbędnych danych, przekazują informację o trwaniu akcji i zawartej z nią danych do Dispatchera.

## Dispatcher

```
//SegmentsDispatcher.js
import { Dispatcher } from "flux";
const dispatcher = new Dispatcher();
export default dispatcher;
```

Plik SegmentsDispatcher.js tworzy nowy obiekt klasy Dispatcher dostępnej w bibliotece flux. Cała logika potrzebna do zastosowania wzorca flux z zastosowaniem Dispatchera jest dostępna w tej klasie.

Funkcje Dispatchera, które wykorzystywane są w aplikacji to:

- **dispatch(payload: object): void** – wykorzystany wcześniej, służy do przekazania informacji o rozpoczętej akcji i danych z nią związanych do komponentów, które zarejestrowały w Dispatcherze callback.
- **register(callback: function): void** – służy do zarejestrowania funkcji, która ma się wywołać po funkcji **dispatch**, z zewnętrznych komponentów aplikacji (w architekturze flux są to Stores).

## Store

Klasa SegmentsStore odpowiedzialna jest za przechowywanie stanu aplikacji podczas jej działania. Zawiera kolekcje odcinków i punktów oraz dane odcinka zaznaczonego przez użytkownika (lub jego brak). Funkcje addChangeListener, removeChangeListener oraz emitChange są wymagane w każdym Storze przy korzystaniu z architektury flux, udostępniają one funkcjonalność klasy dla innych komponentów, które chcą mieć dostęp do stanu aplikacji oraz ich zmian. Klasa udostępnia publiczne metody związane z danymi o odcinkach i punktach, przez co komponenty interfejsowe mogą z nich korzystać. Konstruktor klasy wywołuje subskrypcję do Dispatchera oraz definiuje, co ma się dziać z otrzymanymi danymi. Najczęściej zmieniają one stan aplikacji, po czym wywoływana jest funkcja emitChange, aby nasłuchujące komponenty zostały odpowiednio zaktualizowane.

```

//SegmentsStore.js
class SegmentsStore extends EventEmitter {
    _segments = [];
    _points = [];
    _selectedSegmentId = -1;

    constructor() {
        this._subscribeToDispatcher();
    }

    addChangeListener(callback) {
        this.on("change", callback);
    }

    removeChangeListener(callback) {
        this.removeListener("change", callback);
    }

    emitChange() {
        this.emit("change");
    }

    getSegments = () => _segments;

    getSegmentById = id =>
        _segments.find(segment => segment.id === id) || { id: id };

    getPoints = () => _points;

    getPointById = id => _points.find(point => point.id === id);

    getSelectedSegmentId() {
        return _selectedSegmentId;
    }

    getSelectedSegment() {
        return _selectedSegmentId !== -1
            ? this.getSegmentById(_selectedSegmentId)
            : null;
    }

    _subscribeToDispatcher() {...}
}

```

```
_subscribeToDispatcher() {
    Dispatcher.register(action => {
        switch (action.actionType) {
            case ActionTypes.SAVE_SEGMENT: {
                if (!_segments.find(segment => segment.id === action.segment.id)) {
                    _segments.push(action.segment);
                } else {
                    _segments = _segments.map(segment =>
                        segment.id === action.segment.id ? action.segment : segment
                    );
                }
                this.emitChange();
                break;
            }
            case ActionTypes.SEGMENT_SELECTED: {
                _selectedSegmentId = action.id;
                this.emitChange();
                break;
            }
            case ActionTypes.LOAD_SEGMENTS: {
                _segments = action.segments;
                this.emitChange();
                break;
            }
            case ActionTypes.LOAD_POINTS: {
                _points = action.points;
                this.emitChange();
                break;
            }
            case ActionTypes.NEW_SEGMENT: {
                _selectedSegmentId = action.newSegmentId;
                this.emitChange();
                break;
            }
            default:
        }
    });
}
```

## Widoki

Interfejs użytkownika składa się z wielu komponentów (widoków) zagnieżdzonych w sobie. Komponenty są funkcjami, które po wywołaniu tworzą swoje wewnętrzne stany, rejestrują nasłuchiwacze w SegmentsStore oraz zwracają część interfejsu użytkownika, która ma zostać wyświetlona. Zagnieżdżone w nich elementy to głównie komponenty dostępne w bibliotece material.ui. Dostępne w nich callbacki interakcji użytkownika nadpisane zostały wywołaniami odpowiednich akcji (opisanych wcześniej). Dane pomiędzy zagnieżdżonymi komponentami przekazywane są za pomocą obiektu props.

Utworzono komponenty:

- **SegmentsPage** – Zawiera w sobie tabelę oraz formularz. Formularz jest wyświetlany w zależności od stanu dostępnego w SegmentsStore.
- **SegmentsTable** – Wyświetla tabelę z dostępnymi odcinkami punktowanymi oraz przycisk dodawania nowego odcinka.
- **SegmentsForm** – Wyświetlana po wybraniu jednego z odcinków lub przycisku „Dodaj”. Zawiera pola tekstowe/wyboru na zmianę danych odcinka oraz przyciski do zapisania lub odrzucenia zmian oraz usunięcia odcinka.
- **DeleteSegmentDialog** – Dialog wyświetlany po wybraniu opcji usunięcia odcinka, posiadający opcje Zatwierdź oraz Anuluj.
- **SelectInput** – Współdzielony komponent zawierający powtarzającą się logikę pola wyboru.

Poniższe zrzuty ekranu przedstawiają tylko część interfejsowa wybranych klas, bez uwzględnienia otaczającej ich logiki.

```
//SegmentsPage
return (
  <>
  <SegmentsTable
    segments={segments}
    points={points}
    selectedSegmentId={selectedSegment ? selectedSegment.id : -1}
  />
  {selectedSegment && (
    <SegmentForm points={points} selectedSegment={selectedSegment} />
  )}
</>
```

```
//DeleteSegmentDialog
return (
  <div>
    <Dialog
      open={isDialogOpen}
      onClose={handleDialogDiscard}
      aria-labelledby="alert-dialog-title"
      aria-describedby="alert-dialog-description"
    >
      <DialogTitle id="alert-dialog-title">Usunąć odcinek?</DialogTitle>
      <DialogContent>
        <DialogContentText id="alert-dialog-description">
          Jesteś pewien, że chcesz usunąć odcinek?
        </DialogContentText>
      </DialogContent>
      <DialogActions>
        <Button onClick={handleDialogDiscard} color="primary">
          Anuluj
        </Button>
        <Button onClick={handleDialogConfirm} color="primary">
          Potwierdź
        </Button>
      </DialogActions>
    </Dialog>
  </div>
);
```

```
//SelectInput
return (
  <FormControl className={classes.margin}>
    <InputLabel>{props.name}</InputLabel>
    <Select
      name={props.name}
      onChange={props.handleChange}
      inputProps={{ name: props.name, id: props.id }}
      value={props.value}
      width={2000}
    >
      {props.data.map((point, index) => (
        <option value={point.id} key={point.id}>
          {point.name}
        </option>
      ))}
    </Select>
  </FormControl>
);
```

## API

```
import { Component } from "react";

class SegmentsApi extends Component {
    getSegments() {
        return fetch("api/segments");
    }

    getSegment(id) {
        return fetch(`api/segments/${id}`);
    }

    saveSegment(segment) {
        segment.points = parseInt(segment.points);
        segment.pointsBack = parseInt(segment.pointsBack);
        segment.length = parseInt(segment.length);
        return fetch(`api/segments`, {
            method: "post",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify(segment)
        }).catch(err => console.log(err));
    }

    deleteSegment(id) {
        return fetch(`api/segments/${id}`, { method: "delete" });
    }

    getPoints() {
        return fetch(`api/points`);
    }
}

const api = new SegmentsApi();
export default api;
```

SegmentsApi jest klasą odpowiadającą za pobranie informacji od API umieszczonego na serwerze i przekazanie ich do pozostałych komponentów. Zapytania wykonywane są za pomocą asynchronicznej metody fetch.

# Implementacja aplikacji

## Baza Danych oraz API

Baza danych została napisana w Microsoft SQL Server. Na potrzeby realizowanego przypadku użycia, potrzebne były jedynie tabele odcinków oraz punktów.

```
CREATE TABLE [dbo].[Segment] (
    [Id]             INT NOT NULL,
    [StartingPointId] INT NOT NULL,
    [EndingPointId]   INT NOT NULL,
    [Length]          INT NOT NULL,
    [Points]          INT NOT NULL,
    [PointsBack]       INT NOT NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC),
    CONSTRAINT [FK_Segments_StartEndPointId]
        FOREIGN KEY ([StartingPointId])
            REFERENCES [dbo].[Point] ([Id]),
    CONSTRAINT [FK_Segments_EndingPointId]
        FOREIGN KEY ([EndingPointId])
            REFERENCES [dbo].[Point] ([Id])
);
```

```
CREATE TABLE [dbo].[Point] (
    [Id]             INT NOT NULL,
    [Name]           NVARCHAR (50) NOT NULL,
    [Latitude]        REAL NOT NULL,
    [Longitude]       REAL NOT NULL,
    [Altitude]        REAL NOT NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC)
);
```

Na postawie powstałej bazy danych zostały utworzone modele w aplikacji z użyciem Entity Framework Core. Utworzono modele:

```
public partial class Segment
{
    public int Id { get; set; }
    public int StartingPointId { get; set; }
    public int EndingPointId { get; set; }
    public int Length { get; set; }
    public int Points { get; set; }
    public int PointsBack { get; set; }

    public virtual Point EndingPoint { get; set; }
    public virtual Point StartingPoint { get; set; }
}
```

```
public partial class Point
{
    public Point()
    {
        SegmentEndingPoint = new HashSet<Segment>();
        SegmentStartingPoint = new HashSet<Segment>();
    }

    public int Id { get; set; }
    public string Name { get; set; }
    public float Latitude { get; set; }
    public float Longitude { get; set; }
    public float Altitude { get; set; }

    public virtual ICollection<Segment> SegmentEndingPoint { get; set; }
    public virtual ICollection<Segment> SegmentStartingPoint { get; set; }
}
```

Została również utworzona klasa GOTMobileContext zawierająca logikę łączenia się z bazą oraz wydobywania z nich danych oraz mapowania ich na kolekcje utworzonych modeli. Klasa została skonfigurowana w pliku startowym aplikacji, aby mogła być wstrzykiwana przez konstruktor do innych klas.

Dla utworzonych modeli zostało utworzone RESTowe API, pozwalające za pomocą zapytań protokołu http na operacje create, read, update, delete obiektów znajdujących się w bazie danych. Dla punktów zaimplementowana została jedynie metoda GET, gdyż inne nie były potrzebne dla realizowanego przypadku użycia. Utworzonye endpointy w kontrolerach posiadają walidację danych dla operacji POST.

```
//PointsController.cs
[Route("api/[controller]")]
[ApiController]
public class PointsController : ControllerBase
{
    private readonly GOTMobileContext _gotMobileContext;

    public PointsController(GOTMobileContext gotMobileContext)
    {
        _gotMobileContext = gotMobileContext;
    }

    [HttpGet]
    public ICollection<Point> Get()
    {
        return _gotMobileContext.Point.ToList();
    }
}
```

```

//SegmentsController.cs
[Route("api/[controller]")]
[ApiController]
public class SegmentsController : ControllerBase
{
    private readonly GOTMobileContext _gotMobileContext;

    public SegmentsController(GOTMobileContext gotMobileContext)
    {
        _gotMobileContext = gotMobileContext;
    }

    [HttpGet]
    public ICollection<Segment> GetAllSegments()
    {
        return _gotMobileContext.Segment.ToList();
    }

    [HttpGet("{id}")]
    public IActionResult GetSegmentById(int id)
    {
        var segment = _gotMobileContext.Segment.SingleOrDefault(seg => seg.Id == id);
        if (segment == null)
            return NotFound($"No segment with id {id}.");
        return Ok(segment);
    }

    [HttpPost]
    public IActionResult PostSegment([FromBody] Segment segment)
    {
        var res = _gotMobileContext.Segment.SingleOrDefault(seg => seg.Id == segment.Id);
        if (res != null)
            _gotMobileContext.Segment.Remove(res);
        _gotMobileContext.Segment.Add(segment);
        _gotMobileContext.SaveChanges();
        return Ok();
    }

    [HttpDelete("{id}")]
    public ActionResult DeleteSegment(int id)
    {
        var found = _gotMobileContext.Segment.SingleOrDefault(seg => seg.Id == id);
        if (found == null)
            return BadRequest($"No segment with id {id} found");

        _gotMobileContext.Segment.Remove(found);
        _gotMobileContext.SaveChanges();
        return Ok();
    }
}

```

# Testy integracyjne

## Kontroler dostępu do odcinków

Dla kontrolera odpowiedzialnego za dostarczanie danych do strony internetowej użytkownika, zostały napisane testy integracyjne sprawdzające poprawność wykonywanych operacji. Testy te pełnią rolę testów jednostkowych oraz pokrywają większą część funkcjonalności aplikacji. W testach baza danych została symulowana z użyciem biblioteki Moq. Metoda pozwalająca na utworzenie atrapy zbioru bazy danych z użyciem listy:

```
internal static Mock<DbSet<T>> CreateDbSetMock<T>(IList<T> elements) where T : class
{
    var elementsAsQueryable = elements.AsQueryable();
    var dbSetMock = new Mock<DbSet<T>>();

    dbSetMock.As<IQueryable<T>>().Setup(m => m.Provider).Returns(elementsAsQueryable.Provider);
    dbSetMock.As<IQueryable<T>>().Setup(m => m.Expression).Returns(elementsAsQueryable.Expression);
    dbSetMock.As<IQueryable<T>>().Setup(m => m.ElementType).Returns(elementsAsQueryable.ElementType);
    dbSetMock.As<IQueryable<T>>().Setup(m => m.GetEnumerator()).Returns(() => elementsAsQueryable.GetEnumerator());
    dbSetMock.Setup(m => m.Add(It.IsAny<T>())).Callback<T>(elements.Add);
    dbSetMock.Setup(m => m.Remove(It.IsAny<T>())).Callback<T>(s => elements.Remove(s));

    return dbSetMock;
}
```

Następnie dla każdego testu tworzona jest atrapa bazy danych z użyciem powyższej funkcji. Atrapa przekazywana jest do konstruktora SegmentsController (zamiast kontekstu bazy danych skonfigurowanej w pliku startowym).

```
[TestInitialize]
public void TestInit()
{
    IList<Segment> segments = new List<Segment> {
        new Segment {Id = 1, StartingPointId = 1, EndingPointId = 2, Length = 1000, Points = 10, PointsBack = 7 }
    };
    var segmentsMock = Utils.CreateDbSetMock(segments);
    var dbMock = new Mock<GOTMobileContext>();
    dbMock.Setup(x => x.Segment).Returns(segmentsMock.Object);
    _segmentsController = new SegmentsController(dbMock.Object);
}
```

Na utworzonym kontrolerze przeprowadzane są testy integracyjne:

- **PostSegment\_SegmentAdded\_Ok()** – sprawdza, czy po wywołaniu operacji POST na kontrolerze z danymi nowego odcinka, zostanie on poprawnie zapisany w bazie danych.
- **PostSegment\_SegmentUpdated\_Ok()** – sprawdza, czy po wywołaniu operacji POST na kontrolerze z danymi istniejącego odcinka, zostanie on poprawnie zaktualizowany w bazie danych.
- **GetSegmentById\_SegmentReturned\_Ok()** – sprawdza, czy po dodaniu odcinka do bazy, a następnie zapytanie o niego z użyciem Id, poprawnie zwróci odcinek.
- **GetSegmentById\_SegmentDoesNotExist\_NotFound()** – sprawdza, czy kontroler po otrzymaniu zapytania o nieistniejący odcinek, poprawnie zwróci kod 404 Bad Request z odpowiednią wiadomością błędą.

```

[TestMethod]
public void PostSegment_SegmentAdded_Ok()
{
    //Arrange
    var initialSegmentCount = _segmentsController.GetAllSegments().Count;
    var segment = new Segment
    {
        Id = 1000, //New segment id
        StartingPointId = 1,
        EndingPointId = 2,
        Points = 10,
        PointsBack = 10,
        Length = 10
    };

    //Act
    _segmentsController.PostSegment(segment);

    //Assert
    _segmentsController.GetAllSegments()
        .Should().Contain(segment);
    _segmentsController.GetAllSegments()
        .Should().HaveCount(initialSegmentCount + 1);
}

[TestMethod]
public void PostSegment_SegmentUpdated_Ok()
{
    //Arrange
    var initialSegmentCount = _segmentsController.GetAllSegments().Count;
    var segment = new Segment
    {
        Id = 1, //Existing segment id
        StartingPointId = 1,
        EndingPointId = 2,
        Points = 10,
        PointsBack = 10,
        Length = 10
    };

    //Act
    _segmentsController.PostSegment(segment);

    //Assert
    _segmentsController.GetAllSegments()
        .Should().Contain(segment);
    _segmentsController.GetAllSegments()
        .Should().HaveCount(initialSegmentCount);
}

```

```

[TestMethod]
public void GetSegmentById_SegmentReturned_Ok()
{
    //Arrange
    var segment = new Segment
    {
        Id = 1000,
        StartingPointId = 1,
        EndingPointId = 2,
        Points = 10,
        PointsBack = 10,
        Length = 10
    };
    _segmentsController.PostSegment(segment);

    //Act
    var response = _segmentsController.GetSegmentById(segment.Id);

    //Assert
    response.Should().BeOfType(typeof(OkObjectResult));
    (response as OkObjectResult).Value
        .Should().Be(segment);
}

[TestMethod]
public void GetSegmentById_SegmentDoesNotExist_NotFound()
{
    //Arrange

    //Act
    var response = _segmentsController.GetSegmentById(1000);

    //Assert
    response.Should().BeOfType(typeof(NotFoundObjectResult));
    (response as NotFoundObjectResult).Value
        .Should().Be("No segment with id 1000.");
}

```

▲ ✓ C# GOTMobile.Web.Test (4 tests)	Success
▲ ✓ () GOTMobile.Web.Test (4 tests)	Success
▲ ✓ ApiIntegrationTests (4 tests)	Success
✓ GetSegmentById_SegmentDoesNotExist_NotFound	Success
✓ GetSegmentById_SegmentReturned_Ok	Success
✓ PostSegment_SegmentAdded_Ok	Success
✓ PostSegment_SegmentUpdated_Ok	Success

# Przypadki testowe

**ID:** 1.1

**Tytuł:** Dodanie nowego odcinka

**Środowisko:** Windows 10, Google Chrome, [www.gotmobile.pl/admin](http://www.gotmobile.pl/admin)

**Wersja aplikacji:** 1.0

**Warunek wstępny:** Użytkownik jest zalogowany

**Kroki:**

1. Z paska nawigacji wybrać opcję „Odcinki”
2. Wybrać opcję „Dodaj”
3. Wypełnić dane formularza wyświetlanego na dole strony
4. Zatwierdzić dane
5. Odświeżyć stronę

**Oczekiwane wyniki:** Nowy odcinek został utworzony i jest widoczny w tabeli odcinków od momentu zatwierdzenia danych. Po odświeżeniu strony odcinek nie jest utracony.

**Dane testowe:**

- Punkt początkowy, końcowy: dowolne
- Punkty, Punkty powrót, odległość: 10

**ID:** 1.2

**Tytuł:** Edycja istniejącego odcinka

**Środowisko:** Windows 10, Google Chrome, [www.gotmobile.pl/admin](http://www.gotmobile.pl/admin)

**Wersja aplikacji:** 1.0

**Warunek wstępny:** Użytkownik jest zalogowany

**Kroki:**

1. Z paska nawigacji wybrać opcję „Odcinki”
2. Wybrać jeden z dostępnych odcinków
3. Zmienić wszystkie dane dostępne do edycji w formularzu
4. Zatwierdzić dane
5. Odświeżyć stronę

**Oczekiwane wyniki:** Dane odcinka są zaktualizowane. Po odświeżeniu zmiany nie są utracone.

**Dane testowe:** dowolne

**ID:** 1.3

**Tytuł:** Usunięcie odcinka

**Środowisko:** Windows 10, Google Chrome, [www.gotmobile.pl/admin](http://www.gotmobile.pl/admin)

**Wersja aplikacji:** 1.0

**Warunek wstępny:** Użytkownik jest zalogowany

**Kroki:**

1. Z paska nawigacji wybrać opcję „Odcinki”
2. Wybrać jeden z dostępnych odcinków
3. Wybrać opcję „Usuń” na dolnej części ekranu
4. Potwierdzić wyświetlony dialog
5. Odświeżyć stronę

**Oczekiwane wyniki:** Wybrany odcinek został poprawnie usunięty. Po odświeżeniu strony dane odcinka nie powracają do tabeli.

**Dane testowe:**

- Punkt początkowy, końcowy: dowolne
- Punkty, Punkty powrót, odległość: 10

**ID:** 1.4

**Tytuł:** Poprawne działanie tabeli odcinków

**Środowisko:** Windows 10, Google Chrome, [www.gotmobile.pl/admin](http://www.gotmobile.pl/admin)

**Wersja aplikacji:** 1.0

**Warunek wstępny:** Użytkownik jest zalogowany

**Kroki:**

1. Z paska nawigacji wybrać opcję „Odcinki”
2. Wybrać jeden z dostępnych odcinków
3. Wybrać ten sam odcinek

**Oczekiwane wyniki:**

Po wybraniu odcinka z listy, podświetla się on jaskrawym kolorem. Na dolnej części strony wyświetla się formularz edycji danych.

Po ponownym wybraniu odcinka, strona wraca do poprzedniego stanu: Odcinek nie jest już podświetlony oraz formularz nie jest widoczny.

**Dane testowe:** -

# Automatyzacja przypadków testowych

Poprawne działanie tabeli odcinków

```
test("Selecting same segment twice", async () => {
  //Arrange
  const page = create(<SegmentsPage />);
  const instance = page.root;
  const tableRow = instance.find(
    node => (node.id = "enhanced-table-checkbox-2")
  );
  const form = instance.findByType(SegmentsForm);

  //Act, Assert
  act(() => ReactTestUtils.Simulate.click(tableRow));
  expect(form.props.selectedSegment).not.toBe(noSegmentSelected);

  act(() => ReactTestUtils.Simulate.click(tableRow));
  await waitForDomChange();
  expect(form.props.selectedSegment).toBe(noSegmentSelected);
});
```

Test został napisany z użyciem biblioteki react-testing-library. Element SegmentsPage renderuje się w pamięci podręcznej oraz na nim wykonywane są operacje interfejsowe. Na drugim wierszu tabeli symulowane jest go wybranie i sprawdzone, czy do formularza odcinka przekazały się poprawnie dane. Wybranie wiersza symulowane jest ponownie i sprawdzone jest, czy dane odcinka zostały z formularza usunięte.

Testy uruchamiane są z wiersza poleceń komendą **npm test**. Wyjście z potwierdzeniem działania testu:

```
PASS  src/App.test.js (11.747s)
  ✓ renders without crashing (1255ms)
  ✓ Selecting same segment twice (197ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        14.436s
Ran all test suites.

Watch Usage: Press w to show more.
```

# Ocena jakości kodu i wnioski

## Część interfejsowa aplikacji

Architektura Flux zastosowana w aplikacji po stronie interfejsu użytkownika wymaga poświęcenia dużej ilości czasu na wdrożenie architektury. Przynosi jednak duże korzyści. Nowe funkcjonalności aplikacji mogą być dodawane znacznie szybciej bez konieczności zmieniania zaimplementowanego już kodu. Będzie to przydatne dla realizacji pozostałych przypadków użycia, jak edycja punktów.

SegmentsStore w chwili obecnej jest odpowiedzialny za przechowywanie danych o punktach, co przy rozwijaniu aplikacji będzie problemem i dane te będą musiały być wyodrębnione do osobnej klasy PointsStore.

Podczas rozwoju aplikacji komponenty interfejsowe nabierały coraz większej funkcjonalności oraz dodatkowej infrastruktury z nich związaną (np. stylów). Nie jest to problemem w chwili obecnej, ale fragmenty kodu powinny być wyłączone w osobne klasy oraz arkusze stylów w celu poprawy utrzymalności kodu.

## API Aplikacji

Za pomocą wbudowanego narzędzia w Visual Studio wygenerowane zostały metryki kodu dla API:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
▲ □ GOTMobile.Web (Debug)	86	100	4	128	465	98
▲ □ GOTMobile.Web	81	8	1	42	84	23
▷ 📂 Program	91	2	1	8	14	3
▷ 📂 Startup	71	6	1	34	62	20
▲ □ GOTMobile.Web.Controllers	79	12	2	38	105	33
▷ 📂 PointsController	85	2	2	11	17	4
▷ 📂 SegmentsController	72	8	2	25	49	22
▷ 📂 WeatherForecastController	80	2	2	12	29	7
▲ □ GOTMobile.Web.Models	92	41	2	38	97	21
▷ 📂 GOTMobileContext	80	10	2	36	56	19
▷ 📂 Point	98	15	1	3	17	2
▷ 📂 Segment	100	16	1	1	12	0
▲ □ GOTMobile.Web.Pages	91	39	4	21	179	21
▷ 📂 ErrorModel	88	8	2	6	19	6
▷ 📂 Pages_ViewImports	96	12	4	9	49	5
▷ 📂 Pages_Error	91	19	4	13	73	10
▲ □ GOTMobile.Web.Test (Debug)	74	7	1	60	125	41
▷ 📂 AutoGeneratedProgram	100	1	1	1	1	0
▲ □ GOTMobile.Web.Test	62	6	1	59	124	41
▷ 📂 ApplitoolsIntegrationTests	70	5	1	28	99	24
▷ 📂 Utils	54	1	1	32	17	17

Visual Studio podpowiada, że zaimplementowany kod ma duży wskaźnik utrzymania kodu – 86. Wskaźnik projektu zawierającego testy jest zaniżany przez klasę Utils, w której znajduje się metoda generowania atrapy bazy danych. Implementacja tej metody jest zakończona i nie będzie dalej rozwijana, dlatego można ją zignorować.

Kontrolery API eksponują wszystkie potrzebne metody do implementacji opisanych wcześniej przypadków użycia, lecz powinny posiadać dodatkowe zabezpieczenia w postaci walidacji danych, autoryzacji oraz autentykacji. Część metod powinna zwracać odpowiedź http z wynikiem operacji, zamiast wykonywać operację bez odpowiedzi zwrotnej. Pozwoliłby to na dodatkowe informowanie użytkownika o błędach podczas działania aplikacji.