
	Politechnika Bydgoska im. J. J. Śniadeckich Wydział Telekomunikacji, Informatyki i Elektrotechniki	
<b>Przedmiot</b>	Zaawansowane Programowanie Obiektowe (Projekt)	
<b>Prowadzący</b>	dr inż. Marcin Szczegielniak	
<b>Temat</b>	<i>Strona Internetowa / Integracja z PayPal</i>	
<b>Student</b>	Cezary Tytko 118873	
<b>Ocena</b>		<b>Data oddania spr.</b>

### ***Specyfikacja:***

Aplikacja webowa napisana we frameworku spring boot z wykorzystaniem technologii frontendowych takich jak VUE JS.

Strona internetowa przeznaczona dla pszczelarzy, której głównym założeniem będzie dostarczanie cyfrowych narzędzi pomagających w pracy i zarządzaniu pasieką, gdzie kluczowym elementem będzie:

-Rozbudowany Kalkulator pokarmu i innych produktów potrzebnych na pasiece

Dodatkowo strona może zostać poszerzona o takie elementy jak:

-Sekcja z artykułami na różne tematy, gdzie treści będą mogli zamieszczać zweryfikowani użytkownicy

-Sekcja dla pytań i odpowiedzi użytkowników

Drugim elementem będzie sklep właściciela strony gdzie będzie można zakupić wybrane produkty (miody), wraz z integracją systemu płatności PayPal.

Aplikacja częściowo zostanie napisana jako REST API, ale główne zarządzanie szablonami i routinguem będzie odbywała się za pomocą kontrolerów spring boota.

## ***Dokumentacja:***

### ***Ogólne Założenia:***

Aplikacja została napisana z wykorzystaniem języka java i frameworku SpringBoot, oraz skryptów VueJS. Domyślna konwencja to MVC, a aplikacja została podzielona na kilka głównych elementów:

- Połączenie z bazą danych i zarządzanie nią odbywa się z wykorzystaniem Hibernate i bazy danych MySQL uruchomionej w docker. Kod został napisany w konwencji Code First i z wykorzystaniem wzorca projektowego repozytorium.

- Mechanizm routingu, czyli obsługi linków i przemieszczania się po odpowiednich podstronach za pomocą języka java i SpringBoot'a. Wykorzystuję również mechanizm sesji, ale tylko w stosownych sytuacjach.

- RestApi, obsługiwanego przez SpringBoot'a. api jest wykorzystane w sytuacjach kiedy nie chcemy bądź nie możemy posłużyć się tylko i wyłącznie modelem MVC i jego mechanizmami, np. kiedy chcemy bez przeładowania strony dokonywać zmian bądź odczytu z bazy danych. Api nie zawsze jest pisane w konwencji rest, np. dla paypal gdzie mamy 2 endpointy: do tworzenia płatności i jej autoryzacji, tak samo dla kalkulatora gdzie mamy tylko jeden endpoint, który oblicza i zwraca nam wynik.

- Obsługa generowania html na podstawie modelu i przekazanych do niego obiektów, czyli silnik szablonów jest zrobiony w oparciu o Thymeleaf, który pozwala na przekazanie obiektów z kontrolera do modelu przy wywołaniu funkcji obsługującej odpowiedni endpoint.

- FrontEnd, został stworzony w oparciu o VueJS w zakresie obsługi interakcji z użytkownikiem, korzystania z RestApi, jak i składania szablonów VueJS do ostatecznej strony.

## Implementacja:

### Baza Danych:

- Połączenie z bazą danych jest realizowane automatycznie przez hibernate i spring data z wykorzystaniem connection string:

```
1. spring.datasource.url=jdbc:mysql://localhost:3306/PasiekaZPO?useSSL=false&allowPublicKeyRe-  
trieval=true  
2. spring.datasource.username=root  
3. spring.datasource.password=*****  
4.
```

- Implementacja bazy danych jest realizowana, po pierwsze z pomocą klas Entity będących odzwierciedlaniem tabel i relacji w bazie danych, następnie interfejsów repository odpowiadających klasom encji (w których wykorzystuje również mechanizm dostarczony przez bibliotekę lombok) i pozwalającymi na wykonywanie operacji sql na odpowiadających im tabelach, kończąc na klasie service oznaczonym odpowiednią adnotacją w springBoot aby utworzyć kontener serwisu, obiekt ten skupia wszystkie repozytoria i udostępnia konkretne metody wykorzystane w aplikacji internetowej potrzebne do prowadzenia takiego serwisu :

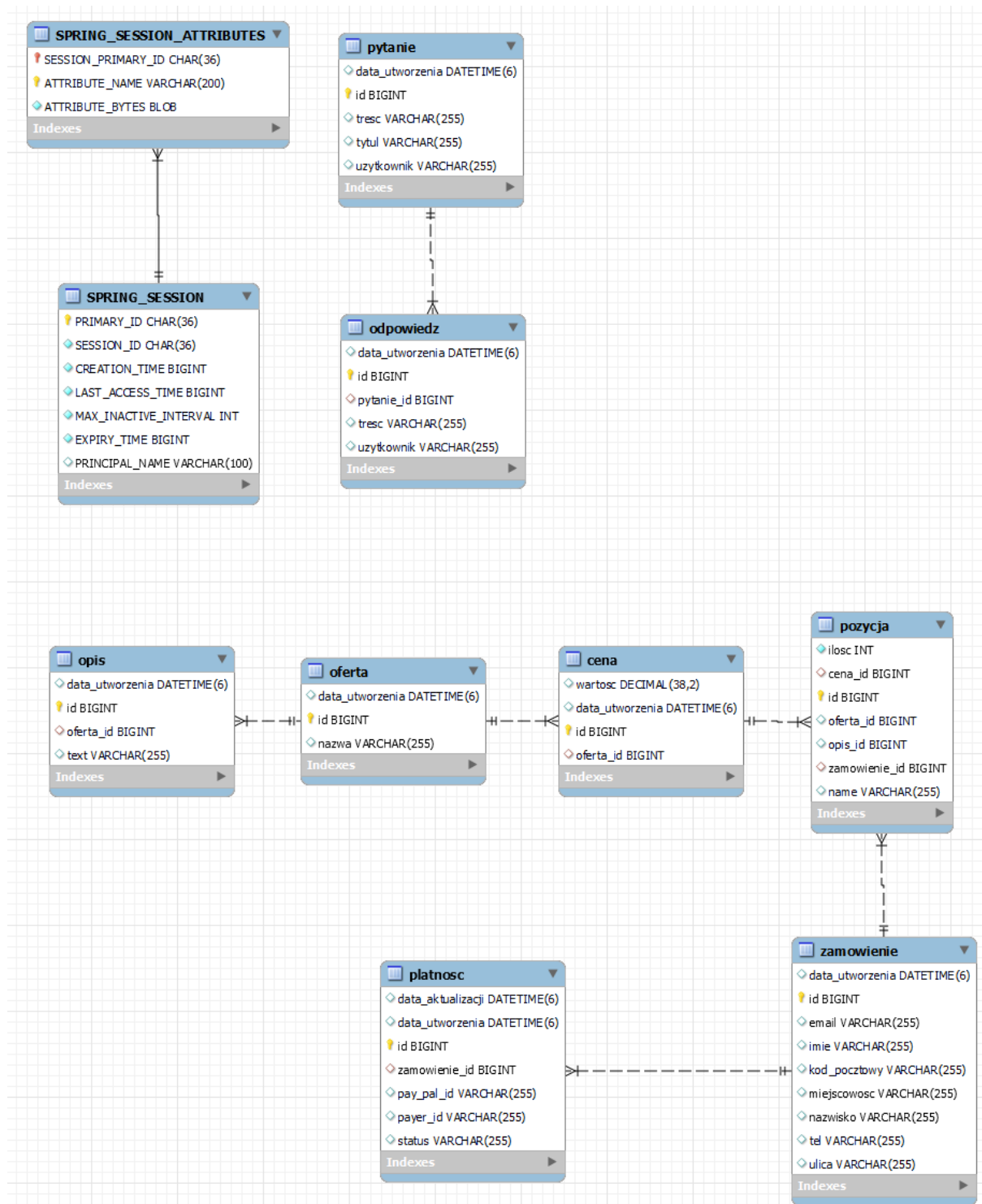
### Encja:

```
1. @Entity  
2. @Data  
3. public class Cena {  
4.     @Id  
5.     @GeneratedValue(strategy = GenerationType.IDENTITY)  
6.     private Long id;  
7.  
8.     private BigDecimal wartosc;  
9.  
10.    @ManyToOne  
11.    @JoinColumn(name = "oferta_id")  
12.    private Oferta oferta;  
13.  
14.    @Temporal(TemporalType.TIMESTAMP)  
15.    private Date dataUtworzenia;  
16.    @PrePersist  
17.    protected void onCreate() {  
18.        dataUtworzenia = new Date();  
19.    }  
20. }  
21.
```

### Repozytorium:

```
1. public interface CenaRepository extends JpaRepository<Cena, Long> {  
2.     List<Cena> findTop1ByOfertaOrderByDataUtworzeniaDesc(Oferta oferta);  
3. }  
4.
```

W wyniku połączenia relacji powstał taki schemat, na którym widać również tabele odpowiedzialne za sesję, która to sesja jest wykorzystana dla mechanizmu koszyka zażądanego przez Rest API:



## Serwis:

```
1. @Service
2. public class PasiekaService {
3.     private final CenaRepository cenaRepository;
4.     private final OfertaRepository ofertaRepository;
5.     private final PozycjaRepository pozycjaRepository;
6.     private final ZamowienieRepository zamowienieRepository;
7.     private final PytanieRepository pytanieRepository;
8.     private final OdpowiedzRepository odpowiedzRepository;
9.     private final PlatnoscRepository platnoscRepository;
10.
11.
12.     public PasiekaService(CenaRepository cenaRepository, OfertaRepository ofertaRepository,
13.         PozycjaRepository pozycjaRepository, ZamowienieRepository zamowienieRepository, PytanieRepository
14.         pytanieRepository, OdpowiedzRepository odpowiedzRepository, PlatnoscRepository platnoscRepository) {
15.
16.         this.cenaRepository = cenaRepository;
17.         this.ofertaRepository = ofertaRepository;
18.         this.pozycjaRepository = pozycjaRepository;
19.         this.zamowienieRepository = zamowienieRepository;
20.         this.pytanieRepository = pytanieRepository;
21.         this.odpowiedzRepository = odpowiedzRepository;
22.         this.platnoscRepository = platnoscRepository;
23.     }
24.
25.     public List<Odpowiedz> getAllAnswersForQuestionSortedById(Long questionId) {
26.         return odpowiedzRepository.findByPytanieIdOrderByDataUtworzeniaAsc(questionId);
27.     }
28.
29.     public Odpowiedz createOdpowiedz(Long questionId, AnswerModel answerModel) {
30.         Pytanie question = new Pytanie();
31.         question.setId(questionId);
32.         Odpowiedz odpowiedz = new Odpowiedz(answerModel);
33.         odpowiedz.setPytanie(question);
34.         odpowiedzRepository.save(odpowiedz);
35.
36.         return odpowiedz;
37.     }
38.
39.     public List<Oferta> getAllOferty() {
40.         return ofertaRepository.findAll();
41.     }
42.
43.     public List<Pytanie> getAllPytania() {
44.         return pytanieRepository.findAll();
45.     }
46.
47.     public Pytanie createPytanie(QuestionModel questionModel) {
48.         var pytanie = new Pytanie(questionModel);
49.         return pytanieRepository.save(pytanie);
50.     }
51.
52.     public Zamowienie CreateZamowienie(OrderModel orderModel, List<Product> productList){
53.         var zam = new Zamowienie(orderModel);
54.         List<Pozycja> pozList = new ArrayList<Pozycja>();
55.         productList.forEach(product -> {
56.             var poz = new Pozycja(product, zam);
57.             pozList.add(poz);
58.             System.out.println(poz);
59.         });
60.         zam.setPozycje(pozList);
61.         zamowienieRepository.save(zam);
62.         return zam;
63.     }
64.
65.     public Zamowienie GetZamowienieById(Long id){
66.         return zamowienieRepository.getReferenceById(id);
67.     }
68.
69.     public BigDecimal GetPriceByPriceId(Long id){
70.         return cenaRepository.getReferenceById(id).getWartosc();
71.     }
72.
73.     public Zamowienie GetZamowieniebyId(Long zamId) {
74.         return zamowienieRepository.getReferenceById(zamId);
75.     }
76.
77.     public Platnosc SavePlatnosc(Platnosc platnosc){
78.         Platnosc exist = platnoscRepository.findByZamowienie(platnosc.getZamowienie());
```

```

75.         if (exist != null)
76.             platnosc.setId(exist.getId());
77.         return platnoscRepository.save(platnosc);
78.     }
79.     public Platnosc GetPlatnoscByPayPalId(String paypalId){
80.         return platnoscRepository.findPlatnoscByPayPalId(paypalId);
81.     }
83.     public BigDecimal GetCenaValueById(Long cenaId) {
84.         return cenaRepository.getReferenceById(cenaId).getWartosc();
85.     }
87.     public Platnosc GetPlatnoscByOrderId(Long orderId) {
88.         return platnoscRepository.findByZamowienie_Id(orderId);
89.     }
90. }
91.

```

## Controller:

- Controllery podstron zostały rozbite klasa per endpoint, a odpowiednie definicje elementów ścieżek zostały wydzielone do wspólnej klasy z elementami statycznymi, ta aby zachować spójność i zminimalizować możliwość wystąpienia błędu spowodowanego literówką

```

1. @Controller
2. public class BeeSyrupCalculatorController {
3.
4.     @GetMapping(StaticRoutesName.BEE_SYRUP_CALCULATOR)
5.     public String showCalculatorForm(Model model) {
6.         model.addAttribute("BaseElementFromSpring", MyStaticUtilities.GetJsonString(EnumSyrup-
pBaseElement.values()));
7.         model.addAttribute("RatiosFromSpring", MyStaticUtilities.GetJsonString(EnumSugarWa-
terRatio.values()));
8.         return "bee_syrup_calculator";
9.     }
10. }
11.

```

## Definicje ścieżek:

```

1. public class StaticRoutesName {
2.     public static final String HOME = "/";
3.     public static final String PAGE_1 = "/page1";
4.     public static final String PAGE_2 = "/page2";
5.     public static final String PAGE_3 = "/page3";
6.     public static final String BEE_SYRUP_CALCULATOR = "/bee-syrup-calculator";
7.     public static final String SHOP = "/shop";
8.     public static final String FORUM = "/forum";
9.     public static final String CART = "/cart";
10.    public static final String ORDER = "/order";
11.    public static final String PAYMENT = "/payment/{orderId}";
12.    public static final String FINALIZE_ORDER = "/finalize-order/{orderId}/";
13.    public static final String GOOD = "good";
14.    public static final String BAD = "bad";
15.    public static String GetPaymentURL(Long id){
16.        return PAYMENT.replace("{orderId}",id.toString());
17.    }
18.    public static String GetFinalizeURL(Long id){
19.        return FINALIZE_ORDER.replace("{orderId}",id.toString());
20.    }

```

```

21.     public StaticRoutesName() {
22.     }
23. }
24.

```

- RestApi również zostało podzielone na 3 mniejsze klasy w zależności od problemu z jakim sobie radzą, w moim projekcie utworzyłem „MyRestController” do obsługi ogólnych elementów strony związanych z interakcją z użytkownikiem, jak obliczenia kalkulatora pasieki, czy odczytywanie i dodawanie postów na forum. „CartRestController” do obsługi koszyka z wykorzystaniem sesji http. „PayPalApiController” do połączenia z systemem płatniczym paypal:

```

1. @RestController
2. @RequestMapping(APIRoutesName.PREFIX)
3. public class MyRestController {
4.
5.     @Autowired
6.     private PasiekaService pasiekaService;
7.     @PostMapping(APIRoutesName.CALCULATE)
10.    public ResponseEntity<CalculationResponse> calculate(@RequestBody CalculationRequest
request) {
11.        CalculationResponse response = MyStaticUtilities.GetCalculationResponse(request);
12.        return ResponseEntity.ok(response);
13.    }
15.    @GetMapping(APIRoutesName.OFFER)
16.    public ResponseEntity<List<Product>> oferta() {
17.
18.        var oferta = pasiekaService.getAllOferty();
19.        var produkty = Product.GetProductsFromOferta(oferta);
20.        return ResponseEntity.ok(produkty);
21.    }
23.    @GetMapping(APIRoutesName.ANSWER + APIRoutesName.PATHPARAM_QUESTIONID)
24.    public ResponseEntity<List<AnswerModel>> answer(@PathVariable Long questionId) {
25.        var odpowiedzi = pasiekaService.getAllAnswersForQuestionSortedByDate(questionId);
26.        var answer = AnswerModel.GetAnswersFromOdpowiedzi(odpowiedzi);
27.        return ResponseEntity.ok(answer);
28.    }
29.
30.    @PostMapping(APIRoutesName.QUESTION)
31.    public ResponseEntity<QuestionModel> addQuestion(@RequestBody QuestionModel question) {
32.
33.        Pytanie savedQuestion = pasiekaService.createPytanie(question);
34.
35.        return ResponseEntity.ok(new QuestionModel(savedQuestion));
36.    }
38.    @PostMapping(APIRoutesName.ANSWER + APIRoutesName.PATHPARAM_QUESTIONID)
39.    public ResponseEntity<AnswerModel> addAnswer(@PathVariable Long questionId, @RequestBody
AnswerModel answer) {
40.        Odpowiedz savedAnswer = pasiekaService.createOdpowiedz(questionId, answer);
41.        return ResponseEntity.ok(new AnswerModel(savedAnswer));
42.    }
44.    //Zapisujemy zamowienie do bazy i zwracamy link do platnosci
45.    @PostMapping(APIRoutesName.ORDER_CREATE)
46.    public ResponseEntity<CreateOrderResponse> CreateOrder(@RequestBody OrderModel orderMo-
del, HttpSession httpSession) {
47.        ShoppingCart shoppingCart = SessionController.SafeGetObject(httpSession, SessionCon-
troller.cartName);
48.        var zam = pasiekaService.CreateZamowienie(orderModel, shoppingCart.getProducts());
49.        httpSession.removeAttribute(SessionController.cartName);
50.        return ResponseEntity.ok(new CreateOrderResponse(StaticRoutesName.GetPaymen-
tURL(zam.getId())));
51.    }
52. }
53.

```

Vidoki html są przygotowywane na podstawie szablonów przez silnik Thymeleaf. Ten język wykorzystywany jest nie tylko do przekazywania elementów z modelu, ale także ma dostęp do wybranych klas np. z informacjami o routingu, udało mi się również wypracować schemat postępowanie przy przekazywaniu zmiennych dla VueJS pochodzących z controllera i modeli, który jest widoczny w kodzie źródłowym, obsługują również takie mechanizmy jak dołączanie skryptów arkuszy stylów i składania wielu szablonów w jeden:

```
1. <!DOCTYPE html>
2. <html lang="pl" xmlns:th="http://www.thymeleaf.org">
3. <head>
4.     <meta charset="UTF-8">
5.     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6.     <title>Strona Domowa</title>
7.     <link th:href="@{styles.css}" th:rel="stylesheet"/>
8.     <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
9. </head>
10. <body>
11. <div id="index"><div th:replace="Header :: #header"></div></div>
12.
13. <h1>Witaj na stronie domowej!</h1>
14.
15. <div th:replace="Footer :: footer"></div>
16.
17.     <script th:inline="javascript" type="module">
18.         import ShoppingCart from "./ShoppingCart.js"
19.         const vueApp = new Vue({
20.             el: '#index',
21.             components: {
22.                 'shoppingCart': ShoppingCart,
23.             },
24.
25.         });
26.     </script>
27. </body>
28. </html>
29.
```

-VueJs wykorzystuję do front endu na praktycznie wszystkich podstronach, ale najważniejszy element jaki udało mi się zaprojektować, jest wydzielony szablon ze skryptem js obsługujący wyświetlanie koszyka. Dzięki takiemu szablonowi ten sam koszyk (obsługiwany tym samym kodem) może zostać dołączony do praktycznie dowolnej podstrony przy jak najmniejszej dodatkowej konfiguracji (której nie da się wyeliminować, ale starałem się ją jak najbardziej ograniczyć i przenieść ją częściowo do nagłówka strony będącego elementem wspólnym dla wielu podstron):

```
1. // ShoppingCart
2. export default {
3.     el: '#shopping-cart',
4.     data() {
5.         return {
6.             cartItems: [],
7.             getCartUrl: '',
8.             addCartUrl: '',
9.             orderUrl: '',
10.            orderSummary: {
11.                totalItems: 0,
```



```

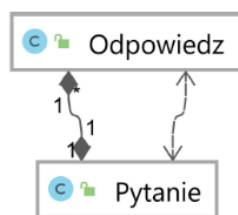
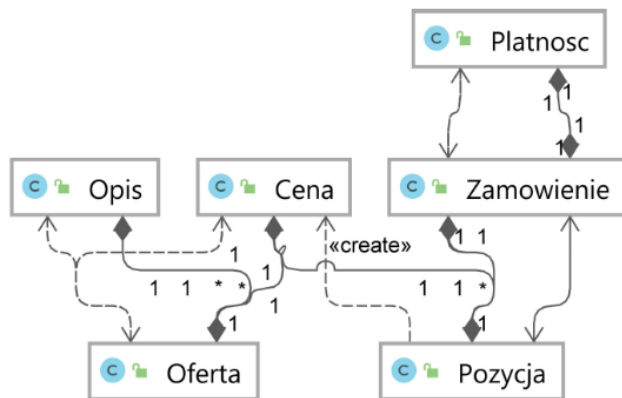
12.         totalPrice: 0,
13.     },
14.     };
15. },
16. template: `
17.     <div>
18.         <div v-if="cartItems.length > 0">
19.             <span>Twój koszyk ({{ orderSummary.totalItems }} produktów)</span>
20.             <ul>
21.                 <li v-for="product in cartItems" :key="product.id">
22.                     <span> {{ product.nazwa }} - {{ formatPrice(product.cena) }} zł </span>
23.                     <span> Ilość: {{ product.ilosc }} - {{ formatPrice(product.cena * pro-
duct.ilosc) }} zł </span>
24.                     <button @click="increaseQuantity(product)">+</button>
25.                     <button @click="decreaseQuantity(product)">-</button>
26.                     <button @click="deleteQuantity(product)">X</button>
27.                 </li>
28.             </ul>
29.             <div>Podsumowanie: {{ formatPrice(orderSummary.totalPrice) }} zł</div>
30.             <button @click="placeOrder">Złóż zamówienie</button>
31.         </div>
32.         <div v-else>
33.             <span>Koszyk jest pusty</span>
34.         </div>
35.     </div>
36. `
37. },
38. mounted() {
39.     this.getCartUrl = document.getElementById('shoppingCart').getAttribute('getCar-
tUrl');
40.     this.addCartUrl = document.getElementById('shoppingCart').getAttribute('addCar-
tUrl');
41.     this.orderUrl = this.$el.getAttribute('orderUrl');
42.     // Pobierz zawartość koszyka po załadowaniu komponentu
43.     this.fetchCartItems();
44. },
45. methods: {
46.     fetchCartItems() {
47.         fetch(this.getCartUrl)
48.             .then(response => response.json())
49.             .then(data => {
50.                 this.cartItems = data;
51.                 console.log(data);
52.                 this.calculateOrderSummary();
53.             })
54.     },
55.     addCartItems(produkt) {
56.         return fetch(this.addCartUrl, {
57.             method: 'POST',
58.             headers: {
59.                 'Content-Type': 'application/json',
60.             },
61.             body: JSON.stringify(produkt),
62.         })
63.             .then(response => {
64.                 if (response.ok) {
65.                     // response.text().then(t => alert(t));
66.                 } else {
67.                     console.error('Błąd dodawania do koszyka:', response.statusText);
68.                 }
69.             })
70.             .catch(error => console.error('Błąd dodawania do koszyka:', error));
71.     },
72.     calculateOrderSummary() {
73.         this.orderSummary.totalItems = this.cartItems.reduce((total, product) => total +
product.ilosc, 0);
74.         this.orderSummary.totalPrice = this.cartItems.reduce((total, product) => total +
(product.cena * product.ilosc), 0);
75.     },
76.     deleteQuantity(product) {
77.         product.ilosc = 0;

```

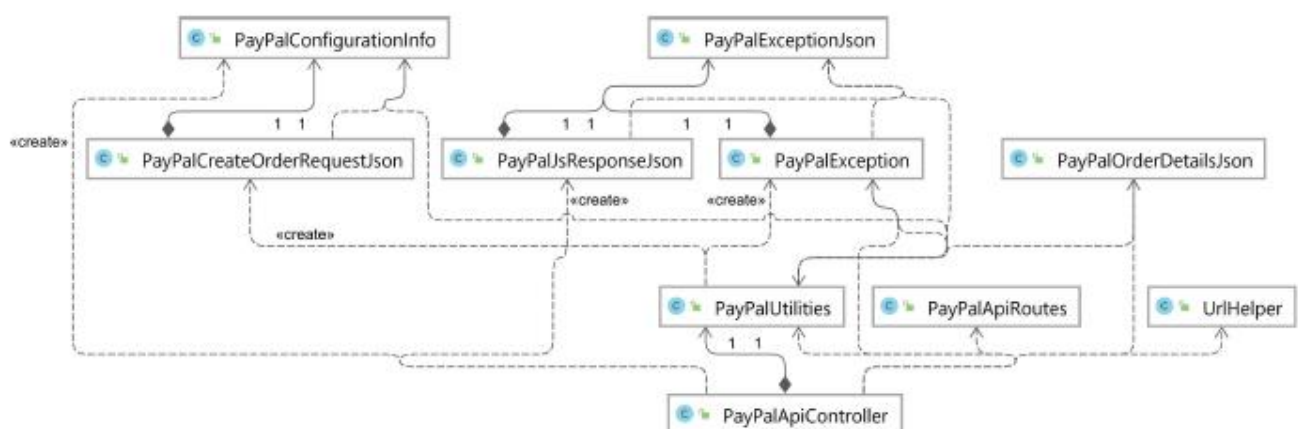
```
77.         this.addCartItems(product).then(value => this.fetchCartItems());
78.     },
79.     increaseQuantity(product) {
80.         if (product.ilosc < 10) {
81.             product.ilosc++;
82.             this.addCartItems(product);
83.             this.calculateOrderSummary();
84.         }
85.     },
86.     decreaseQuantity(product) {
87.         if (product.ilosc > 1) {
88.             product.ilosc--;
89.             this.addCartItems(product);
90.             this.calculateOrderSummary();
91.         }
92.     },
93.     formatPrice(price) {
94.         return price.toFixed(2); // Sformatuj cenę do dwóch miejsc po przecinku
95.     },
96.     placeOrder(price) {
97.         if (this.orderUrl != '')
98.             window.location.href = this.orderUrl;
99.     }
100. },
101. });
102.
```

**Przygotowałem schematy przedstawiające ogólną strukturę aplikacji bez wchodzenia w szczegóły z wykorzystaniem narzędzi środowiska IntelliJ:**

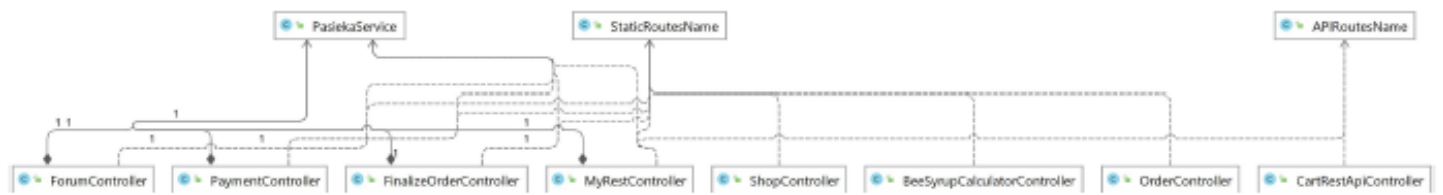
### Schemat klas Entities:



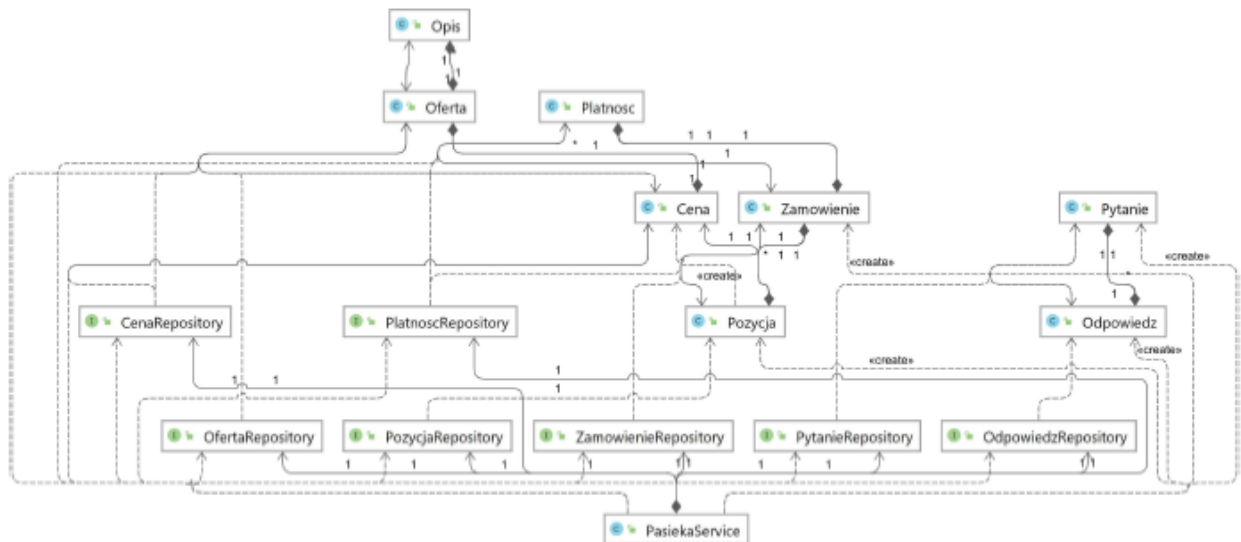
### Schemat paypal:



## Schemat dla routingu:



## Schemat dla serwisu:



## Widoki:

Aby obsługiwać responsywność przygotowałem jedynie podstawę opartą o media query, gdzie mamy miejsce w którym możemy zdefiniować nadpisanie zachowania css dla mniejszych rozdzielczości:

```
1. /* Media Query dla urządzeń o szerokości mniejszej niż 768px (tablet i telefon) */
2. @media screen and (max-width: 768px) {
3.     /* Dodatkowe dostosowania dla wersji mobilnej */
4.     .button-kup,
5.     .button-dodaj-do-koszyka,
6.     .button-dodaj,
7.     .button-oblicz {
8.         width: auto; /* Zwróć szerokość do domyślnej */
9.     }
10.
11.     .page-title {
12.         font-size: 28px; /* Mniejsza czcionka dla wersji mobilnej */
13.     }
14.
15.     .page-description {
16.         font-size: 16px; /* Mniejsza czcionka dla wersji mobilnej */
17.         margin-bottom: 15px;
18.     }
19. }
20.
```

### Strona główna.

- Strona główna
  - Kalkulator Syropu
  - Sklep miodowy
  - Forum dla pszczelarzy
- Koszyk jest pusty

## Witaj na stronie domowej!

[Polityka prywatności](#) [Regulamin](#) [Kontakt](#)

© 2024 Moja Strona. Wszelkie prawa zastrzeżone.

Podstrona z kalkulatorem pasiecznym z możliwością wyboru elementu bazowego, ilości i proporcji, podstawowy element pracy z pszczołami.

- 
- 
- 
-

Strona główna

Kalkulator Syropu

Sklep miodowy

Forum dla pszczelarzy

Koszyk jest pusty

# Kalkulator Syropu dla Pszczół

Rodzaj: 

Cukier

 Ilość:

12

Stosunek cukru do wody: 

TrzyDoDwoch

Oblicz

Woda: 8

Cukier: 12

Syrop: 16

Polityka prywatności

Regulamin

Kontakt

© 2024 Moja Strona. Wszelkie prawa zastrzeżone.

**Podstrona sklepu z uzupełniony koszykiem o 5 produktów z 3 ofert.**

- 
- 
- 
-

Strona główna

Kalkulator Syropu

Sklep miodowy

Forum dla pszczelarzy

Twój koszyk (5 produktów)

- 
- 
-

Oferta 1 - 10.50 zł Ilość: 1 - 10.50 zł

Oferta 2 - 20.30 zł Ilość: 3 - 60.90 zł

Oferta 3 - 12.70 zł Ilość: 1 - 12.70 zł

+

-

X

+

-

X

+

-

X

Podsumowanie: 84.10 zł

Złóż zamówienie

**Lista Produktów**

**Oferta 1**

Cena: 10.5 zł

Opis Oferty 1

Dodaj do koszyka

**Oferta 2**

Cena: 20.3 zł

Opis Oferty 2

Dodaj do koszyka

**Oferta 3**

Cena: 12.7 zł

**Forum z możliwością dodawania nowych wątków, jak i odpowiadaniem na już istniejące.**

<ul style="list-style-type: none"><li>•</li><li>•</li><li>•</li><li>•</li></ul>	<div>Strona główna</div> <div>Kalkulator Syropu</div> <div>Sklep miodowy</div> <div>Forum dla pszczelarzy</div> <div>Koszyk jest pusty</div>
---	--

Nasze Forum

Dyskusje, pytania i współdzielenie wiedzy

Nazwa Użytkownika

Tytuł pytania

Treść pytania

Dodaj pytanie

UzytkownikA:Tytuł 1:2024-01-07 12:42

Pytanie 1

Nazwa Użytkownika

Nowa odpowiedź

Dodaj odpowiedź

○ UzytkownikF:2024-01-07 12:42

Odpowiedz 1-1



Formularz składania zamówienia, uzupełniony danymi testowymi.

- 
- 
- 
-

Strona główna

Kalkulator Syropu

Sklep miodowy

Forum dla pszczelarzy

Twój koszyk (5 produktów)

- 
- 
-

Oferta 1 - 10.50 zł Ilość: 1 - 10.50 zł

+

-

X

Oferta 2 - 20.30 zł Ilość: 3 - 60.90 zł

+

-

X

Oferta 3 - 12.70 zł Ilość: 1 - 12.70 zł

+

-

X

Podsumowanie: 84.10 zł

Formularz Zamówienia

Imię:

ImięTestowe

Nazwisko:

NazwiskoTestowe

Email:

test@example.com

Ulica z numerem domu:

Testowa 1

Miejscowość:

Testowo

Kod pocztowy:

00-000

Numer telefonu:

123456789

Złóż zamówienie

## Podsumowanie zamówienia z przyciskiem płatności PayPal

- 
- 
- 
- 

Strona główna  
Kalkulator Syropu  
Sklep miodowy  
Forum dla pszczelarzy

### Podsumowanie Zamówienia

Numer Zamówienia: 2

Imię: ImięTestowe

Nazwisko: NazwiskoTestowe

Email: test@example.com

#### Produkty w zamówieniu:

Oferta 1 - 10.5 X 1 = 10.5 PLN
Oferta 2 - 20.3 X 3 = 60.9 PLN
Oferta 3 - 12.7 X 1 = 12.7 PLN

**Łączna cena: 84.1 PLN**

PayPal

Płać wygodnie i bezpiecznie

[Polityka prywatności](#) [Regulamin](#) [Kontakt](#)

© 2024 Moja Strona. Wszelkie prawa zastrzeżone.

**Widok po udanej płatności**

### Status płatności Zamówienia: 2

Płatność zakończona pomyślnie! Oplacono. PayPalId: 9J227724MC082091Y

[Polityka prywatności](#)[Regulamin](#)[Kontakt](#)

© 2024 Moja Strona. Wszelkie prawa zastrzeżone.

**Widok po nie udanej płatności.**

### Status płatności Zamówienia: 2

Płatność nieudana. Spróbuj ponownie. Płatność PayPal nie zaakceptowana

Powrót do płatności

[Polityka prywatności](#)[Regulamin](#)[Kontakt](#)

© 2024 Moja Strona. Wszelkie prawa zastrzeżone.

## Widok po stronie płatności PayPal

PayPal Checkout — Mozilla Firefox

https://www.sandbox.paypal.com/checkoutnow?sessionID=uid\_451e03c2a8\_mtc6mjg6mta&bl

JD

P


zł10.50 PLN

Ship to ImięTestowe NazwiskoTestowe

Testowa 1, 00-000 TESTOWO


Change


Pay with


☒  **PayPal balance** **\$2.82**  
USD

☐ Make this my preferred way to pay

PayPal's conversion rate: 1 USD = 3.72879 PLN

☐  **CREDIT UNION 1 (AK)**  
Checking \*\*\*\*2864

☐  **Visa**  
Credit \*\*\*\*2713

☐  **PayPal Credit**  
Apply for PayPal Credit Pay over time for your purchase of \$2.82 with PayPal Credit. Subject to credit approval. [See terms](#)

✓ [See More](#)

+ [Add debit or credit card](#)

Pay Later

Pay in 4

**Complete Purchase**

[Payment method rights](#)

## Integracja PayPal:

Integracja z systemem PayPal wymagała stworzenia konta developerskiego do uzyskania danych dla sklepu (MID), oraz danych testowych konta klienckiego do testowania przebiegu płatności. Głównym zadaniem było napisanie odpowiedniego RestApi (pokazane w sekcji poświęconej RestApi) które będzie wykorzystywane przez kod JS umożliwiający wywołanie okna płatności, a następnie autoryzację transakcji:

Na obsługę PayPal Składa się wiele obiektów pomocniczych, np. do deserializacji i serializacji przesyłanych obiektów z i do api, narzędzia do obsługi tej metody płatności znajdują się w:

```
1. public class PayPalUtilities {
2.
3.     @Autowired
4.     private PasiekaService pasiekaService;
5.
6.     public PayPalUtilities() {
7.     }
8.
9.     public static String CreateOrder(Zamowienie order, PayPalConfigurationInfo config,
10.                                     String returnUrl, String cancelUrl) throws Exception {
11.         String apiUrl = config.APIServer;
12.         String authUserPass = config.ClientId + ":" + config.Secret;
13.         String authBase = "Basic " + java.util.Base64.getEncoder().encodeToString(authUser-
14. Pass.getBytes());
15.         PayPalCreateOrderRequestJson payPalCreateOrderBody = new PayPalCreateOrderRequestJ-
16. son(order, config, null, null);
17.         URL url = new URL(apiUrl + "/checkout/orders");
18.         HttpURLConnection httpURLConnection = (HttpURLConnection) url.openConnection();
19.         httpURLConnection.setRequestMethod("POST");
20.         httpURLConnection.setRequestProperty("Content-Type", "application/json");
21.         httpURLConnection.setRequestProperty("Authorization", authBase);
22.         httpURLConnection.setDoOutput(true);
23.
24.         String body = SerializeObject(payPalCreateOrderBody);
25.         System.out.println(body);
26.         setBody(httpURLConnection, body);
27.         String response = getBody(httpURLConnection);
28.         return response;
29.     }
30.
31.     public static String GetOrderDetails(String orderId, PayPalConfigurationInfo config)
32.     throws Exception {
33.         String apiUrl = config.APIServer;
34.         String authUserPass = config.ClientId + ":" + config.Secret;
35.         String authBase = "Basic " + java.util.Base64.getEncoder().encodeToString(authUser-
36. Pass.getBytes());
37.         URL url = new URL(apiUrl + "/checkout/orders/" + orderId);
38.         HttpURLConnection httpURLConnection = (HttpURLConnection) url.openConnection();
39.         httpURLConnection.setRequestMethod("GET");
40.         httpURLConnection.setRequestProperty("Authorization", authBase);
41.
42.         String response = getBody(httpURLConnection);
43.         return response;
44.     }
45.
46.     public void SaveNewPayPalPayment(PayPalOrderDetailsJson payPalOrderDetailsJson, Long or-
47. derId) {
```

```

48.         Zamowienie zamowienie = pasiekaService.GetZamowieniebyId(orderId);
49.         Platnosc platnosc = new Platnosc(zamowienie, paypalOrderDetailsJson);
50.         pasiekaService.SavePlatnosc(platnosc);
51.     }
53.     public void UpdatePayPalPayment(PayPalOrderDetailsJson paypalOrderDetailsJson) {
54.         Platnosc platnosc = pasiekaService.GetPlatnoscByPayPalId(paypalOrderDetailsJson.id);
55.         platnosc.UpdateStan(paypalOrderDetailsJson);
56.         pasiekaService.SavePlatnosc(platnosc);
57.     }
59.     public static String ApproveOrder(String orderId, PayPalConfigurationInfo config) throws
Exception {
60.         String apiUrl = config.APIServer;
61.         String authUserPass = config.ClientId + ":" + config.Secret;
62.         String authBase = "Basic " + java.util.Base64.getEncoder().encodeToString(authUser-
Pass.getBytes());
64.         URL url = new URL(apiUrl + "/checkout/orders/" + orderId + "/capture");
65.         System.out.println(url.toString());
66.         HttpURLConnection httpURLConnection = (HttpURLConnection) url.openConnection();
67.         httpURLConnection.setRequestMethod("POST");
68.         httpURLConnection.setRequestProperty("Authorization", authBase);
69.         httpURLConnection.setRequestProperty("Content-Type", "application/json");
70.         httpURLConnection.setDoOutput(true);
71.
72.         String response = getBody(httpURLConnection);
74.         return response;
75.     }
77.     public static boolean isCompleted(Long orderPackId, PayPalOrderDetailsJson detailsJson)
{
78.         return detailsJson.status.equals("COMPLETED")
79.             && detailsJson.purchase_units.get(0).reference_id.equals(String.valueOf(or-
derPackId));
80.     }
81.
82.     public static boolean CheckFinalizePayment(Long orderPackId, String paypalOrderId, Pay-
PalConfigurationInfo config) {
83.         try {
84.             if (paypalOrderId == null || paypalOrderId.isEmpty()) {
85.                 return false;
86.             }
87.             String response = GetOrderDetails(paypalOrderId, config);
88.             PayPalOrderDetailsJson paypalOrderDetails = GetObjectFromJsonString(response,
PayPalOrderDetailsJson.class);
89.             return isCompleted(orderPackId, paypalOrderDetails);
90.         } catch (Exception e) {
91.             return false;
92.         }
93.     }
95.     public static <T> T GetObjectFromJsonString(String jsonString, Class<T> clazz) throws
JsonProcessingException {
96.
97.         ObjectMapper objectMapper = new ObjectMapper();
98.         return objectMapper.readValue(jsonString, clazz);
100.     }
119.     private static void handleException(Exception e) {
120.         e.printStackTrace(); // Handle exception as needed
121.     }
123.     private static void setBody(HttpURLConnection httpURLConnection, String body) throws IO-
Exception {
124.         try (OutputStream os = httpURLConnection.getOutputStream()) {
125.             byte[] input = body.getBytes(StandardCharsets.UTF_8);
126.             os.write(input, 0, input.length);
127.         }
128.     }
130.     private static String getBody(HttpURLConnection httpURLConnection) throws Exception {
132.         int responseCode = httpURLConnection.getResponseCode();
133.         System.out.println("Response Code: " + responseCode);
134.
135.         if (responseCode >= HttpURLConnection.HTTP_OK && responseCode <= 300) {
136.             // Odczytaj odpowiedź, jeśli jest pozytywna (HTTP 200 OK)

```

```

137.         BufferedReader reader = new BufferedReader(new InputStreamReader(httpURLConnection
138.         String line;
139.         StringBuilder response = new StringBuilder();
140.
141.         while ((line = reader.readLine()) != null) {
142.             response.append(line);
143.         }
144.         reader.close();
145.         System.out.println("Response Body: " + response.toString());
146.         return response.toString();
147.     } else {
148.         // Odczytaj informacje o błędzie, jeśli odpowiedź jest inna niż HTTP 200 OK
149.         BufferedReader errorReader = new BufferedReader(new InputStreamReader(httpURL-
150.         Connection.getErrorStream()));
151.         String errorLine;
152.         StringBuilder errorResponse = new StringBuilder();
153.
154.         while ((errorLine = errorReader.readLine()) != null) {
155.             errorResponse.append(errorLine);
156.         }
157.         errorReader.close();
158.         System.out.println("Error Response: " + errorResponse.toString());
159.         try {
160.             var expJson = GetObjectFromJsonString(errorResponse.toString(), PayPalExcep-
161.             tionJson.class);
162.             if (expJson != null)
163.                 throw new PayPalException(expJson);
164.             return null;
165.         } catch (PayPalException ppe) {
166.             throw ppe;
167.         }
168.         catch (Exception e) {
169.             throw new Exception(e.getMessage() + errorResponse.toString());
170.         }
171.     }
172. }
173. public static String SerializeObject(Serializable object) {
174.     try {
175.         // Inicjalizujemy obiekt ObjectMapper
176.         ObjectMapper objectMapper = new ObjectMapper();
177.         // Konwertujemy obiekt na JSON
178.         return objectMapper.writeValueAsString(object);
179.     } catch (JsonProcessingException e) {
180.         e.printStackTrace();
181.         return null;
182.     }
183. }
184. }
185. }
186.

```

## Walidacja (BackEnd):

Choć wstępna walidacja może odbywać się na poziomie frontendu, np. za pomocą mechanizmów vue.js, jednak obiekty domenowe, do których mamy dostęp przez publiczne api, należy dodatkowo zabezpieczyć po stronie serwera, tak aby uniemożliwić nawet ponad przeciętnemu użytkownikowi, wprowadzenie wadliwych danych, co przy persystencji może skutkować np. błędem bazy danych

Dla przykładu zabezpieczyłem w ten sposób klasy „Pytanie” i „Odpowiedz”, dodając adnotacje @Size z biblioteki „jakarta.validation”, z przykładowymi parametrami.

```
1. @Entity
2. @Data
3. public class Odpowiedz {
4.     @Id
5.     @GeneratedValue(strategy = GenerationType.IDENTITY)
6.     private Long id;
7.
8.     @Size(min = 10, max = 1023)
9.     private String tresc;
10.
11.     @Size(min = 3, max = 255)
12.     private String uzytkownik;
13.
14.     @ManyToOne
15.     @JoinColumn(name = "pytanie_id")
16.     private Pytanie pytanie;
17.
18.     @Temporal(TemporalType.TIMESTAMP)
19.     private Date dataUtworzenia;
20.
21.     @PrePersist
22.     protected void onCreate() {
23.         dataUtworzenia = new Date();
24.     }
25. }
```

---

```
1. @Entity
2. @Data
3. public class Pytanie {
4.     @Id
5.     @GeneratedValue(strategy = GenerationType.IDENTITY)
6.     private Long id;
7.
8.     @Size(min = 10, max = 4095)
9.     private String tresc;
10.
11.     @Size(min = 3, max = 255)
12.     private String uzytkownik;
13.
14.     @Size(min = 3, max = 255)
15.     private String tytul;
16.
17.     @OneToMany(mappedBy = "pytanie", cascade = CascadeType.ALL)
18.     private List<Odpowiedz> odpowiedzi = new ArrayList<>();
19.
20.     @Temporal(TemporalType.TIMESTAMP)
```



```

22.     private Date dataUtworzenia;
23.     @PrePersist
24.     protected void onCreate() {
25.         dataUtworzenia = new Date();
26.     }
34. }
35.

```

Teraz możemy walidować nasze obiekty w sposób pokazany poniżej:

```

1. @RestController
2. @RequestMapping(APIRoutesName.PREFIX)
3. public class MyRestController {
4.
5.     @Autowired
6.     private PasiekaService pasiekaService;
7.
8.     private Validator validator = Validation.buildDefaultValidatorFactory().getValidator();
9.
10.    @PostMapping(APIRoutesName.QUESTION)
11.    public ResponseEntity<?> addQuestion(@RequestBody QuestionModel question) {
12.        Set<ConstraintViolation<QuestionModel>> violations = validator.validate(question);
13.        if (!violations.isEmpty()) {
14.            return ResponseEntity.badRequest().body(violations.toString());
15.        } else {
16.            Pytanie savedQuestion = pasiekaService.createPytanie(question);
17.            return ResponseEntity.ok(new QuestionModel(savedQuestion));
18.        }
19.    }
20.
21.    .
22.    .
23.    .
24.    .
25.    .
56. }
57.

```

Fragment kodu przedstawiający wykorzystanie walidatora, którym sprawdzamy zgodność instancji obiektu z założeniami narzucanymi przez adnotacje w jego definicji (Klasie). Valideate na obiekcie, zwraca nam zbiór (Set) wykrytych niezgodności, jeżeli zbiór jest pusty możemy uznać że walidacja zakończyła się powodzeniem i dane spełniają narzucone normy, w przeciwnym razie zwracamy kod błędu wraz z komunikatem i nie zapisujemy wadliwych danych w bazie.

## ***Podsumowanie:***

W aplikacji znajdują się o wiele więcej elementów niż te tutaj wymienione, takie jak własne wyjątki, enumy, klasy modeli dla widoków i dla odpowiedzi api, mechanizm serializacji, obsługa sesji, i wiele klas narzędziowych udostępniających metody statyczne jak UrlHelper, ale są one mało ciekawe i nie wnoszą dużo do samego działania aplikacji.

Podczas pisania aplikacji kładłem nacisk i uwzględniałem przyszłą możliwość rozbudowy aplikacji i łatwą skalowalność, dlatego w projekcie znajduje się wiele elementów lub klas wykorzystanych w pojedynczych przypadkach, ale z potencjałem i już gotowym miejscem na rozbudowę, poprzez dodanie kolejnych podstron, lub rozbudowę już istniejących o kolejne elementy.

VueJs wykorzystany po stronie front end przysporzył wiele problemów, ale koniec końców udało mi się wypracować metodologię pozwalającą łączyć SpringBoot'owy model MVC z VueJS poprzez thymeleaf i udało mi się stworzyć szablon tylko w oparciu o VueJS pozwalający dynamicznie wyświetlać zawartość koszyka.

Tworząc aplikację nie skupiałem się na contentcie strony, jest on symboliczny, często wygenerowany automatycznie, miał on służyć tylko i wyłącznie do testowania działania aplikacji, jestem zdania, że dla dobrze napisanego kodu od podstaw nie ma znaczenia czy odpowiada on za obsługę sprzedaży jedzenia, mebli czy opon, zawsze baza jest taka sama i najważniejsze żeby ona była solidna i użytkowniko-odporna. Tak samo nie skupiłem się na ostylowaniu strony i ograniczyłem się tylko do załączeniu pliku css, tak aby się na potencjalne wprowadzenie zmian.

Najbardziej rozbudowanym elementem jest integracja z PayPal, pomysł na to narodził się już w trakcie pisania aplikacji i postanowiłem że to będzie główny i najbardziej dopracowany element, dla którego koniec końców reszta miałaby tylko tłem.

### ***Wnioski:***

Podjąłem się dość pracochłonnego zadania jak na jedną osobę, czyli napisania całej strony internetowej, dlatego później woląłem skupić się tylko na jednym jej elemencie, czyli na BackEnd'owej obsłudze PayPal, i było to całkiem fajne. Dzięki temu projektowi jestem już także pewien, że nigdy nie chcę mieć nic wspólnego z FrontEnd'em i frameworkami JS, warto mieć jakieś rozeznanie w tej kwestii, ale nie jest to nic przyjemnego, dlatego uważam, że sam projekt lepiej jakby się odbywał w parach, powiem więcej on powinien być w parach z podziałem na BackEnd i FrontEnd, dzięki temu każdy może zająć się tym w czym chce się specjalizować, a dodatkowo można mieć namiastkę pracy nad projektem komercyjnym (zespołowym).