

Politechnika Świętokrzyska Wydział Elektrotechniki, Automatyki i Informatyki	
Studia: Stacjonarne II stopnia	Kierunek: Informatyka
Technologie obiektowe - Projekt	Grupa: 1ID21B Bartłomiej Kozieł
REST Web Services CRUD	

1. Założenia projektu

Automatyczne generowanie na podstawie metadanych pobranych z bazy danych usług REST Web Services pozwalających na tworzenie, odczytywanie, aktualizację i kasowanie danych (CRUD). Po uruchomieniu aplikacja łączy się z bazą danych i pozwala wybrać różne tabele oraz rodzaje operacji, dla których usługi mają zostać wygenerowane. Należy obsłużyć związki 1:1, 1:* oraz *:*. Należy opracować stronę internetową pozwalającą na uruchamianie tych usług.

Do implementacji tego tematu został stworzony prosty serwis API napisany przy użyciu DataNucleus, Spring i MySQL oraz klient napisany w Vue.js. Serwis implementuje proste funkcjonalności sklepu, takie jak dodawanie produktów, przypisywanie kategorii produktom, dodawanie sklepów, klientów oraz obsługę koszyka. Nie jest to w pełni funkcjonalny sklep, a jedynie idea do której została stworzona aplikacja oparta o JDO.

2. Specyfikacja projektu

2.1. JDO

JDO (ang. Java Data Object) jest standardem zapewnienia trwałości, który udostępnia niezależny interfejs API pomiędzy bazą danych a aplikacją, przeznaczony do utrwalania danych obiektowych w dowolnej bazie danych. JDO może współpracować z relacyjnymi bazami danych, obiektowymi bazami danych, repozytoriami XML, systemami plików itp.

Java Data Objects to standardowy sposób uzyskiwania dostępu do trwałych danych w bazach danych, wykorzystujący zwykle stare obiekty Java (POJO) do reprezentowania trwałych danych. Podejście to oddziela manipulację danymi (wykonywaną przez dostęp do elementów danych Java w obiektach domeny Java) od manipulacji bazą danych (wykonywaną przez wywołanie metod interfejsu JDO). To rozdzielenie obaw prowadzi do wysokiego stopnia niezależności widoku danych w języku Java od widoku danych w bazie danych.

Interfejsy są zdefiniowane z myślą o trwałości użytkownika:

- **PersistenceManager**: składnik odpowiedzialny za cykl życia instancji trwałych, fabrykę zapytań i dostęp do transakcji
- **Zapytanie**: komponent odpowiedzialny za wysyłanie zapytań do magazynu danych i zwracanie trwałych instancji lub wartości
- **Transakcja**: komponent odpowiedzialny za inicjowanie i kończenie transakcji

Klasę zdolną do trwałości można stworzyć trzema metodami. W tym projekcie zastosowałem metodę poprzez rozszerzenie kodu bajtowego – stosując odpowiednie narzędzia kod bajtowy powstałym w wyniku kompilacji jest rozszerzany tak, aby był zgodny z kontraktem trwałości. W tej metodzie klasy, którym chcemy nadać cechę zdolności do trwałości najpierw muszą zostać skompilowane. W następnym kroku odpowiednie narzędzie rozszerzania kodu bajtowego (enhancer) rozszerza na podstawie metadanych skompilowany kod bajtowy. Przy specyfikacji zostało wykorzystane Metadata API.

2.2. MetaData API

JDO API udostępnia dynamiczny interfejs API do definiowania metadanych dla klas, jako alternatywę dla używania adnotacji lub metadanych XML.

Podstawowa idea interfejsu API metadanych polega na tym, że programista uzyskuje obiekt metadanych z pliku **PersistenceManagerFactory** i dodaje do niego definicję zgodnie z wymaganiami przed zarejestrowaniem go w celu wykorzystania w procesie utrwalania.

Dostęp do bazy danych uzyskujemy za pomocą fabryki JDO, która udostępnia nam menedżera transakcji odpowiedzialnego za wykonywanie transakcji:

```
PersistenceManagerFactory pmf = new JDOPersistenceManagerFactory(pumd, null);
PersistenceManager pm = pmf.getPersistenceManager();
```

W projekcie jest to odpowiednio linia kodu:

```
private static final PersistenceManagerFactory PERSISTENCE_MANAGER_FACTORY =
JDOHelper.getPersistenceManagerFactory("PersistenceUnit");
```

W pliku Config. W której do dodaliśmy do fabryki plik **META-INF/persistence.xml**.

Kolejnym krokiem jest pobranie transakcji za pośrednictwem klasy PersistenceManager, oraz rozpoczęcie jej poprzez wywołanie metody begin(). W celu utrwalenia obiektu w bazie danych wywoływana jest metoda makePersistent() klasy PersistenceManager, której parametrem jest obiekt, który chcemy utrwalić. W projekcie metody są wywoływane w plikach repozytoriów np. CartItemRepository w folderze repository.

Aby połączyć nasz program z bazą danych, musimy utworzyć jednostkę *trwałości* w czasie wykonywania, aby określić trwałe klasy, typ bazy danych i parametry połączenia:

```
persistence-unit name="PersistenceUnit">
  <class>com.bartek.projekt.jdo.model.Product</class>
  <class>com.bartek.projekt.jdo.model.ShoppingCart</class>
  <class>com.bartek.projekt.jdo.model.CartItem</class>
  <class>com.bartek.projekt.jdo.model.Person</class>
  <class>com.bartek.projekt.jdo.model.Category</class>
  <class>com.bartek.projekt.jdo.model.Shop</class>
  <class>com.bartek.projekt.jdo.model.ShopProduct</class>
  <exclude-unlisted-classes />
  <properties>
    <property name="datanucleus.ConnectionURL" value="jdbc:mysql://localhost/projekt_jdo" />
    <property name="javax.jdo.option.ConnectionDriverName" value="com.mysql.jdbc.Driver" />
    <property name="javax.jdo.option.ConnectionUserName" value="root" />
    <property name="javax.jdo.option.ConnectionPassword" value="" />
    <property name="datanucleus.schema.autoCreateAll" value="true" />
    <property name="datanucleus.schema.validateTables" value="false" />
    <property name="datanucleus.schema.validateConstraints" value="false" />
  </properties>
</persistence-unit>
```

3. Opis projektu

3.1. DataNucleus, Spring Application

Implementacja projektu została oparta o **DataNucleus Access Platform**, której implementacja jest w pełni zgodną implementacją specyfikacji Java Data Objects oraz Java Persistence API, zapewniając przezroczystą trwałość obiektów Java. Obsługuje trwałość w najszerzej gamie magazynów danych dowolnego oprogramowania trwałego Java, obsługując wszystkie główne wzorce mapowania obiektowo-relacyjnego (ORM), umożliwia wykonywanie zapytań za pomocą JDOQL, JPQL lub SQL i zawiera własny wzmacniacz kodu bajtowego. Pozwala na trwałość w relacyjnych magazynach danych (RDBMS), obiektowych bazach danych.

Pliki konfiguracyjne zostały umieszczone w pliku persistence.xml w folderze META-INF.

```
<persistence-unit name="PersistenceUnit">
  <class>com.bartek.projekt.jdo.model.Product</class>
  <class>com.bartek.projekt.jdo.model.ShoppingCart</class>
  <class>com.bartek.projekt.jdo.model.CartItem</class>
  <class>com.bartek.projekt.jdo.model.Person</class>
```

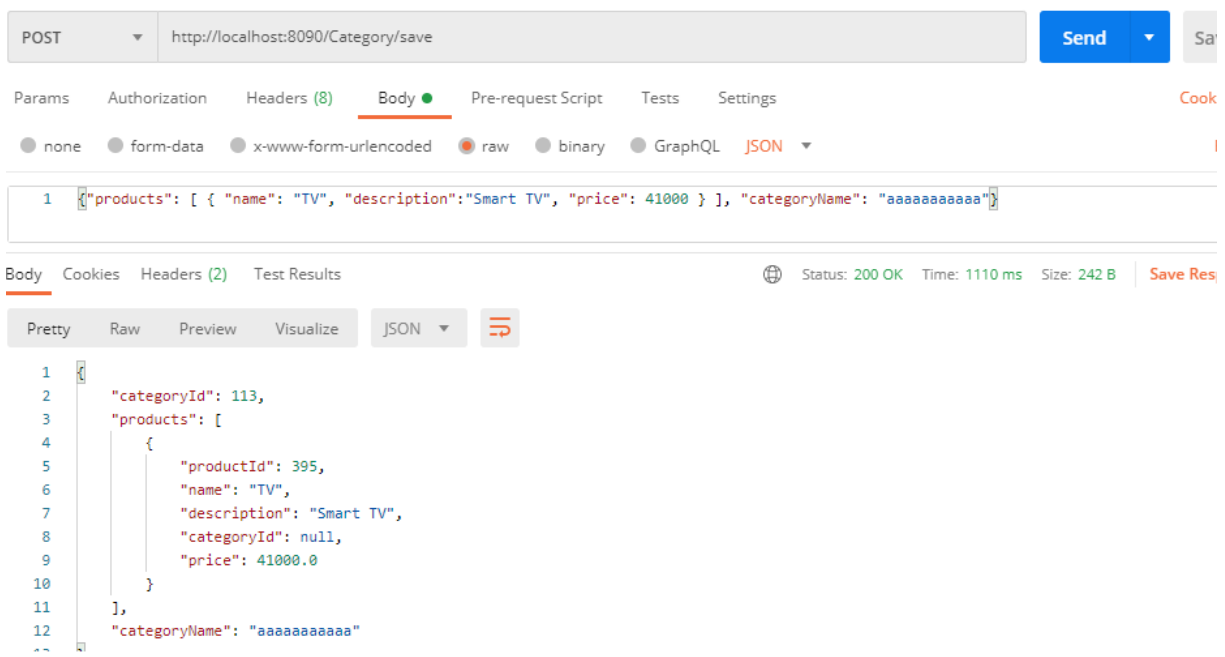
```

<class>com.bartek.projekt.jdo.model.Category</class>
<class>com.bartek.projekt.jdo.model.Shop</class>
<class>com.bartek.projekt.jdo.model.ShopProduct</class>
<exclude-unlisted-classes />
<properties>
  <property name="datanucleus.ConnectionURL" value="jdbc:mysql://localhost/projekt_jdo" />
  <property name="javax.jdo.option.ConnectionDriverName" value="com.mysql.jdbc.Driver" />
  <property name="javax.jdo.option.ConnectionUserName" value="root" />
  <property name="javax.jdo.option.ConnectionPassword" value="" />
  <property name="datanucleus.schema.autoCreateAll" value="true" />
  <property name="datanucleus.schema.validateTables" value="false" />
  <property name="datanucleus.schema.validateConstraints" value="false" />
</properties>
</persistence-unit>

```

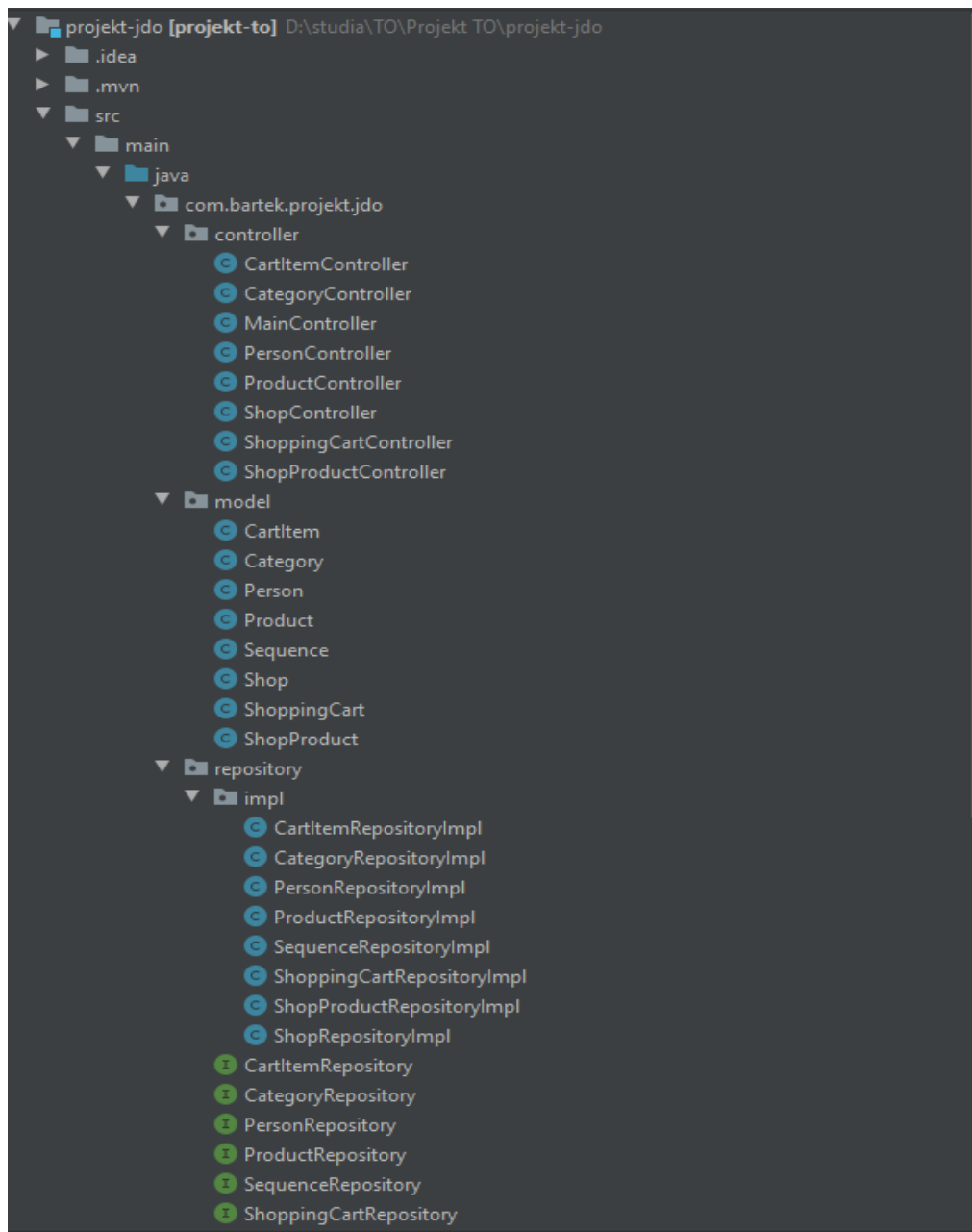
Dane to są wykorzystywane w pliku Config do utworzenia fabryki PERSISTENCE_MANAGER_FACTORY, która jest wykorzystywana późniejszych operacji na danych.

Projekt, a w zasadzie API zostało oparte na Spring Framework. Użycie tego frameworka ułatwiło stworzenie serwisu CRUD, do późniejszej obsługi przez klienta napisanego w Vue.js. Z REST API możemy się także połączyć i wykonywać operację za pomocą Postman. Jest to najprostszy sposób do pokazania zasady działania serwisu.



Po wysłaniu dowolnego zapytania http, w tym przypadku POST z obiektem do zapisania, otrzymujemy odpowiedź zwrotną w postaci utworzonego obiektu, lub błąd. W kliencie Vue zapytania są wykonywane automatycznie na podstawie wybranego modelu. Sam mechanizm generacji zostanie omówiony później.

Za obsługę zapytań po stronie serwisu API odpowiadają Controllery Spring w folderze controller. To tam w poszczególnych plikach odpowiadających utworzonym Modelom zostają przypisane podstawowe operacje umożliwiające dodawanie, edycję odczytywanie i usuwanie obiektów. W każdej metodzie znajduje się odwołanie do repozytoriów tych Modeli i to tam są wykonywane operacje utrwalania



Jak widać na grafice ogólny zamysł projektu napisanego w Javie możemy podzielić na: Modele implementujące i odzwierciedlające kolumny w bazie danych, repozytoria do obsługi zbiorów obiektów utworzonych na ich podstawie oraz zawierających mechanizmy utrwalania. Następnym elementem są kontrolery służące do obsługi zapytań http.

```

@Override
public Mono<Person> save(Person person) {
    PersistenceManager pm = Config.getPersistenceManagerFactory().getPersistenceManager();
    return Mono.just(pm.makePersistent(person));
}

@Override
public Mono<Person> update(Person person) {
    PersistenceManager pm = Config.getPersistenceManagerFactory().getPersistenceManager();
    Person oldPerson = pm.getObjectById(Person.class, person.getId());
    if (person.getName() != null)
        oldPerson.setName(person.getName());
    if (person.getAddress() != null)
        oldPerson.setAddress(person.getAddress());
    if (person.getPhone() != null)
        oldPerson.setPhone(person.getPhone());
    return Mono.just(pm.makePersistent(oldPerson));
}

@Override
public void delete(Long personId) {
    PersistenceManager pm = Config.getPersistenceManagerFactory().getPersistenceManager();
    Person person = pm.getObjectById(Person.class, personId);
    pm.deletePersistent(person);
}

@Override
public Flux<Person> findAll() {
    PersistenceManager pm = Config.getPersistenceManagerFactory().getPersistenceManager();
    Query query = pm.newQuery(Person.class);
    return Flux.fromIterable((List<Person>) query.execute());
}

```

Przykład metod w repozytorium `PersonRepository`, służących do zapisywania, usuwania, edycji obiektu przekazanego do metody w formie parametru przez kontroler. Na grafice można zobaczyć moment utrwalania/zwalniania obiektu `Person person`.

Poniżej jest zawarty przykład Modelu `Person`. Model zawiera pola, konstruktor oraz proste gettery i setery. W zależności od relacji są bardziej lub mniej rozbudowane poszczególne modele.

```

public class Person {
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.INCREMENT)
    private Long id;

    @Persistent
    private String name;

    @Persistent
    private String address;
}

```

```

@Persistent
private Long phone;

public Person() {
    super();
}

public Long getId() {
    return id;
}

public String getName() {
    return name;
}

public String getAddress() {
    return address;
}

public Long getPhone() {
    return phone;
}

public void setName(String name) {
    this.name = name;
}

public void setAddress(String address) {
    this.address = address;
}

public void setPhone(Long phone) {
    this.phone = phone;
}
}

```

Na koniec przykład Controllera:

```

@RestController
@CrossOrigin(origins = "*")
@RequestMapping(value = "/Category")
public class CategoryController {

    @Autowired
    private CategoryRepository repository;

    @PostMapping("/save")
    public Mono<Category> save(@RequestBody Category category) {

        return repository.save(category);
    }

    @PostMapping("/update")
    public Mono<Category> update(@RequestBody Category category) {
        return repository.update(category);
    }
}

```

```

}

@GetMapping("/findAll")
public Flux<Category> show() {
    return repository.findAll();
}

@GetMapping("/delete/{id}")
public void delete(@PathVariable Long id) {
    repository.delete(id);
}
}

```

Zawiera on podstawowe metody do obsługi repozytorium z obiektami klasy Category. Są to proste metody: add, update, showAll oraz delete.

Diagram Klas:

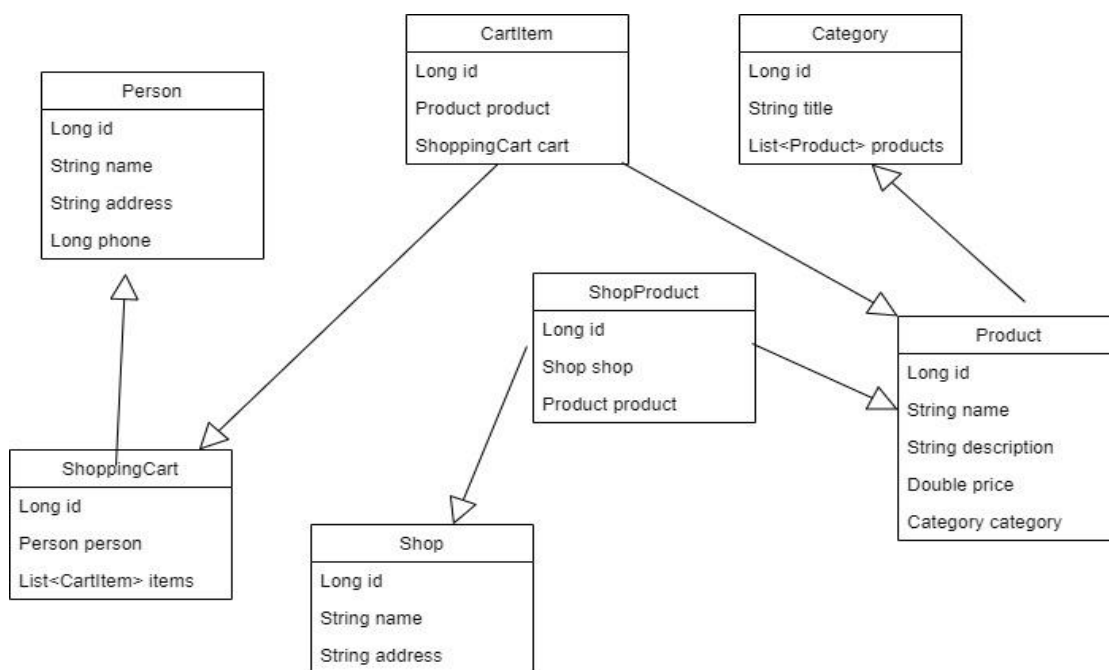
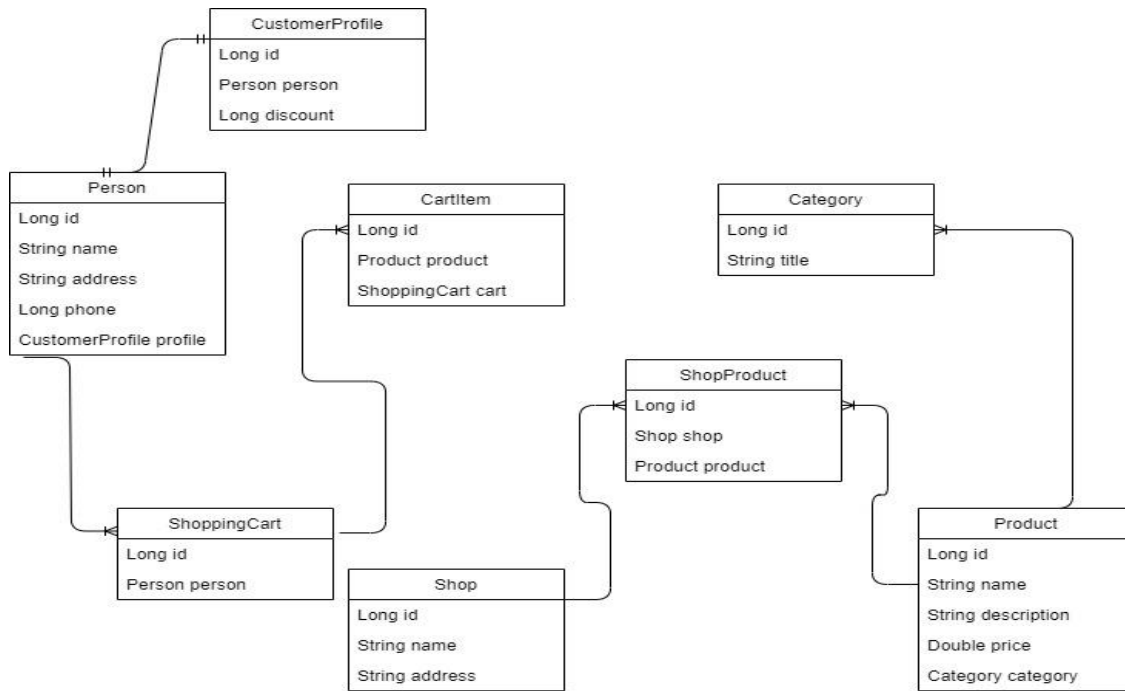


Diagram relacji:



Jak widać zostały zaimplementowane relacje wiele do wielu Shop->ShopProduct<-Product, wiele do jednego i jeden do jednego Person – CustomerProfile.

Uruchomienie aplikacji, podstawowe komendy:

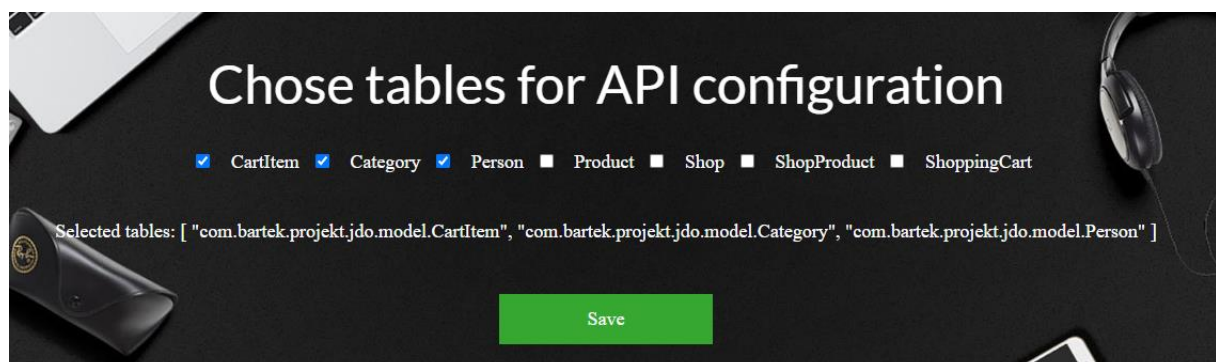
mvn clean compile

mvn -e datanucleus:enhance

mvn spring-boot:run

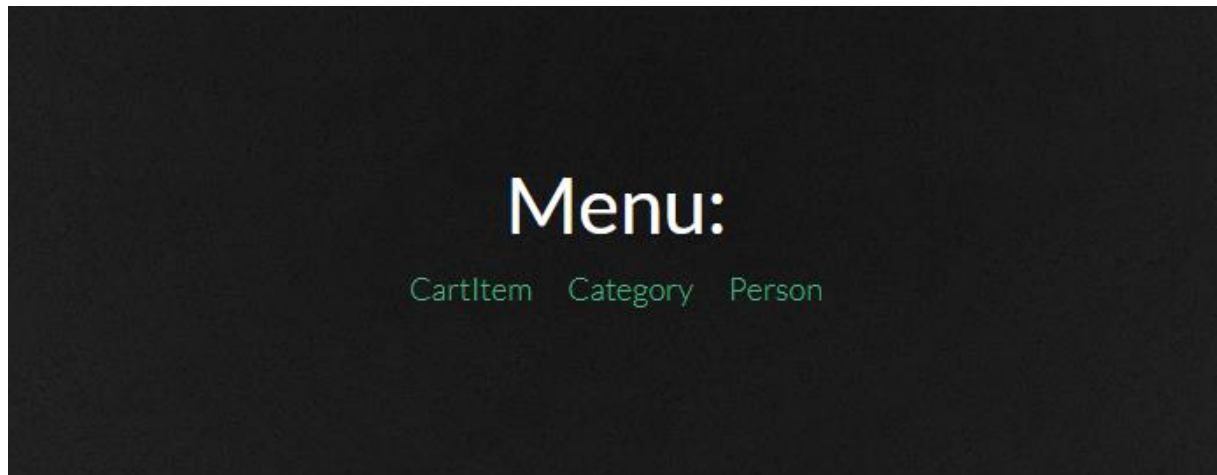
3.2. Vue SinglePage application

Klient został napisany przy użyciu Frameworka Vue.js. Jest to prosta aplikacja SinglePage pozwalająca obsłużyć usługi serwisu CRUD w prost sposób za pomocą http requestów biblioteki Axios zaimportowanej do projektu. Ogólny zamysł aplikacji polega na umożliwieniu wybrania przez użytkownika modeli z bazy danych, na których będzie zbudowany prosty serwis, do wykonywania operacji dodawania, edycji, wyświetlania i usuwania danych.

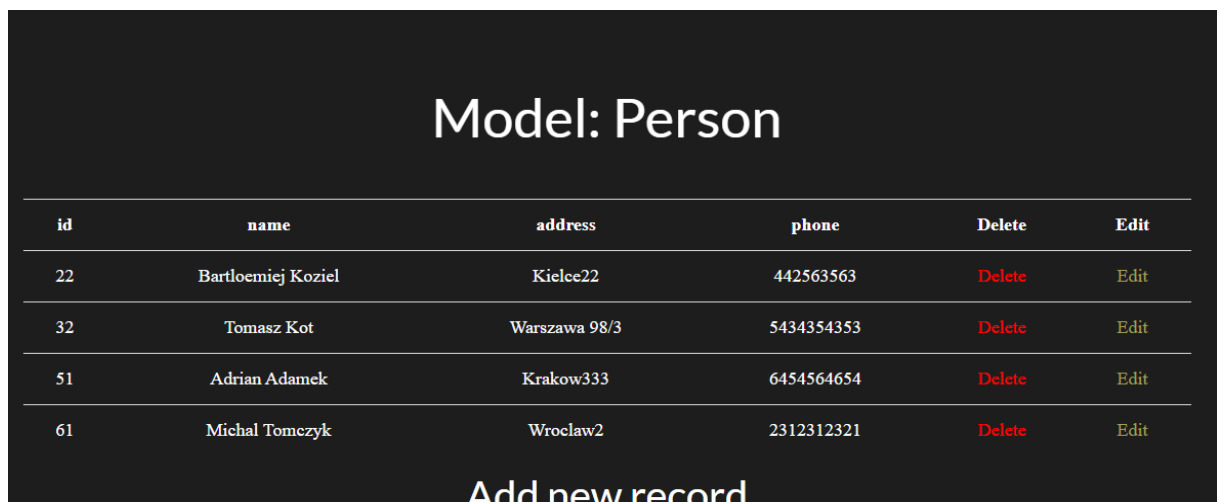


Poniżej znajduje się screen strony startowej Vue. Użytkownik może wybrać modele do wykonywania operacji na nich. Wszystkie modele opierają się na jednym pliku Model.js, który jest uniwersalny i pozwala na obsługę dowolnego modelu wybranego z listy.

Po zapisaniu naszego wyboru, wyświetla się proste menu z wygenerowanym serwisem do wybranych przez użytkownika Modeli.



Po kliknięciu w link z modelem użytkownik zostaje przekierowany do strony Modelu, gdzie wyświetlają się wszystkie zapisane, utrwalone obiekty oraz formularz do dodania nowego rekordu.



Model: Person					
id	name	address	phone	Delete	Edit
22	Bartloemiej Koziel	Kielce22	442563563	Delete	Edit
32	Tomasz Kot	Warszawa 98/3	5434354353	Delete	Edit
51	Adrian Adamek	Krakow333	6454564654	Delete	Edit
61	Michal Tomezyk	Wroclaw2	2312312321	Delete	Edit
Add new record					

Add new record

name

address

phone

ADD

Po kliknięciu w przycisk edytuj, jest możliwa edycja wybranego obiektu:

×

Edit Model:

name

Michał Tomeczyk

address

Wrocław2

phone

2312312321

ADD

Schemat ten jest taki sam dla wszystkich modeli pobranych z Serwisu, generacja odbywa się w czterech plikach js. Start, Menu, Model i EditModel. Pliki te ustawiają uniwersalne operacji i metody na podstawie Metadata otrzymanych z serwera Aplikacji CRUD.

Komendy do uruchomienia aplikacji, po zainstalowaniu Node.js oraz npm.

```
# install dependencies
```

```
npm install
```

```
npm run dev
```

4. Podsumowanie

Próba stworzenia serwisu API(CRUD), opartego na JDO, była trudna w implementacji ale nie niemożliwa. Serwis ten tworzyłem sam i pochłonął on sporą ilość czasu. Przy wykorzystaniu frameworka Spring oraz technologii DataNucleus udało mi się stworzyć prostą aplikację umożliwiającą rozszerzanie kodu bajtowego oraz obsługującą bazę danych MySQL. Dodatkowo stworzona przeze mnie aplikacja w Vue pozwala na uniwersalne wykorzystanie dowolnych modeli stworzonych w tym serwisie i utrwalonych. Umożliwia one wykonywanie podstawowych operacji CRUD, w intuicyjny i prosty dla użytkownika sposób. Do sprawozdania dołączam filmik z działającą aplikacją.