

**Politechnika Świętokrzyska**

Wydział Elektrotechniki, Automatyki i Informatyki

**Technologie obiektowe**

**Projekt**

**Temat: Narzędzia odwzorowania obiektowo - relacyjnego np. Hibernate, Oracle TopLink, Microsoft ADO.NET Entity Framework, ActiveRecord lub innego.**

Wykonał: Parzniewski Daniel

Kierunek: Informatyka (niestacjonarnie)

Grupa: 1IZ21A

1. **Wstęp**

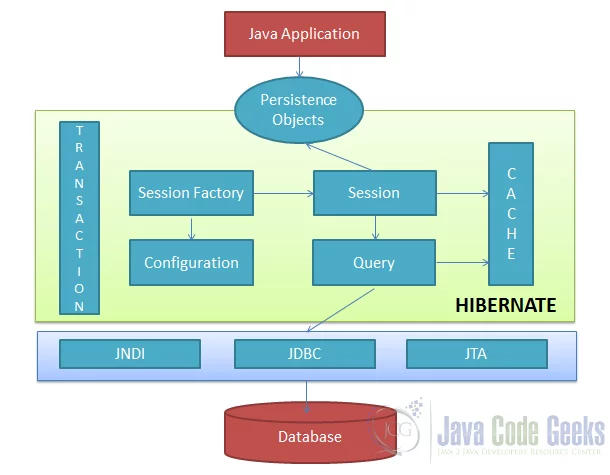
Odwzorowanie obiektowo – relacyjne, nazywane inaczej mapowaniem relacyjno – obiektowym (z ang. Object – relational mapping ORM, O/RM, O/R mapping tool), jest to technika programowania służąca do konwertowania danych przy użyciu języków programowania obiektowego. Tworzona jest wirtualna baza danych obiektów, której można używać z poziomu języka programowania. W projekcie wykorzystano do tego celu narzędzie Hibernate[[1]](#footnote-1) oraz relacyjną bazę danych MySQL[[2]](#footnote-2).

1. **Hibernate**

Implementacją odwzorowania obiektowo – relacyjnego dla języka Java uznawaną powszechnie za najpopularniejszą jest Hibernate. Cechuje się ona wysoką wydajnością, skalowalnością. Obsługuje asocjacje, kompozycje, dziedziczenie, polimorfizm, mechanizm kolekcji. Jest jedną z implementacji standardu Java Persistence.

* 1. **Architektura Hibernate**

Na rys. 1 przedstawiono architekturę Hibernate.



*Rys. 1. Architektura Hibernate.[[3]](#footnote-3)*

**Configuration** – jest to pierwszy obiekt, który zostanie utworzony w aplikacji Hibernate i jest tworzony tylko raz. Zapewnia połączenie bazy danych i struktury mapowania klas;

**Session Factory** – obiekt bezpieczny wątkowo i tworzący instancje sesji Hibernate. Wymaga obiektu „Configuration”. Tworzony podczas uruchamiania aplikacji;

**Session** – obiekt uzyskiwany każdorazowo w celu wykonania pracy z bazą danych. Nie jest bezpieczny wątkowo;

**Querry** – reprezentuje zapytania SQL, służące do pobierania lub modyfikowania danych. Obiekt zapytania służy do powiązania parametrów;

**Transaction –** transakcja reprezentuje pojedynczą jednostkę pracy bazy danych. To jest obiekt opcjonalny;

**Cache** – Hibernate obsługuje pamięci podręczne na poziomie zapytań i sesji, co podnosi wydajność;

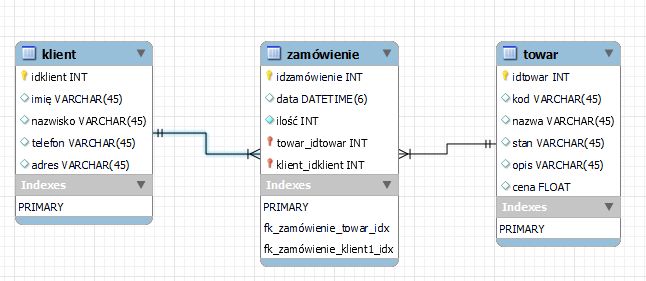
**Persistence Objects –** obiekty Java, które są odwzorowywane na tabelę relacyjnej bazy danych.

1. **MySQL**

MySQL to system zarządzania relacyjnymi bazami danych (relational database management system - RDBMS), opracowany i wydany w 1995 roku. Został opracowany i sponsorowany przez firmę MySQL AB, która została następnie przejęta przez firmę Sun Microsystems znaną dziś, jako Oracle Corporation. MySQL to połączenie „My” (córka współzałożyciela) i „SQL”.

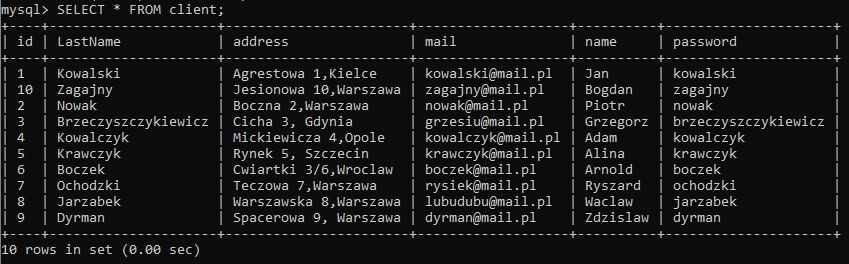
MySQL był jednym z pierwszych RDBMS o otwartym kodzie źródłowym, jakie kiedykolwiek opracowano i uruchomiono. Obecnie istnieje wiele wariantów MySQL. Jednak podstawowa składnia wszystkich wariacji pozostaje taka sama. Zaprojektowany i napisany w językach programowania C i C++, MySQL jest kompatybilny ze wszystkimi podstawowymi systemami operacyjnymi. Jest to podstawowy składnik szeroko popularnego stosu oprogramowania aplikacji internetowych o otwartym kodzie źródłowym o nazwie LAMP, co oznacza Linux, Apache, MySQL, PHP/Perl/Python.

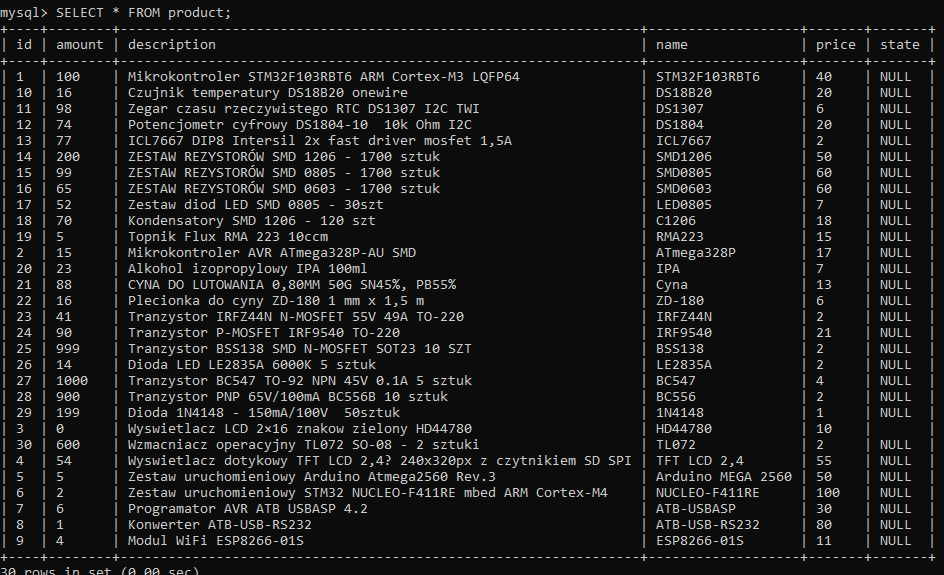
Poniżej schemat bazy danych użyty w projekcie (rys. 2).



*Rys.2. Schemat bazy danych użytej w projekcie.*

W projekcie wykorzystano bazę MySQL, nazwaną „shop”, zawierającą tabele „*client”* (rysunek 3) oraz „*product*” (rysunek 4).

*Rys. 3. Tabela MySQL „client”.*

*Rys. 4. Tabela MySQL „product”.*

1. **Mapowanie obiektowo-relacyjne z wykorzystaniem Hibernate**

Mapowanie obiektowo-relacyjne (ang. Object-relational mapping ORM) jest to konwertowanie danych z tabel w relacyjnej bazie danych na obiekty aplikacji klienckiej i na odwrót. Większość dzisiejszych aplikacji jest zorientowanych obiektowo, ale dane przechowuje się wciąż w relacyjnych bazach danych. ORM pozwala na używanie obiektów java, jako reprezentacji relacyjnej bazy danych. To mapuje dwie koncepcje (zorientowane obiektowo i relacyjne). Mapowanie pozwala na:

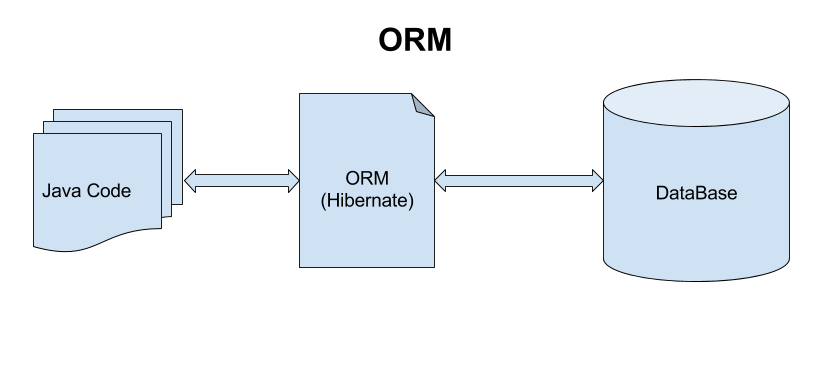
- zredukowanie ilości pracy związanej z oprogramowaniem dostępu do danych;

- skorzystanie z zalet relacyjnych baz danych jednocześnie nie rezygnując z obiektowości programowania;

- automatyczną obsługę transakcji i połączenia z bazą;

Hibernate jest faworkiem ORM – służy do opisu, w jaki sposób obiekty są reprezentowane w bazie danych, a Hibernate zajmuje się konwersją. Hibernate zwiększa wydajność operacji na bazie danych dzięki buforowaniu i minimalizacji liczby przesyłanych zapytań. Jest to projekt rozwijany jako open source.

JDBC jest API dostępu do bazy danych i działa "w sposób relacyjny" – poprzez odpytywanie tabel i odczyt zawartości wierszy i kolumn. Hibernate używa JDBC pod maską, aby pobrać dane, a następnie przekonwertować je do obiektów. Blokowy schemat mapowania przedstawia rysunek 5.



*Rys. 5. Mapowanie ORM.[[4]](#footnote-4)*

* 1. **Plik konfiguracyjny Hibernate**

Konfiguracja zawarta jest w pliku *„hibernate.cfg.xml”*. W nim zawarte są dane konfiguracyjne dotyczące bazy danych na której działamy. W projekcie przedstawia się on następująco:

*<?***xml version**="1.0" **encoding**="UTF-8"*?>***<!DOCTYPE hibernate-configuration PUBLIC** "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd"***>***<**hibernate-configuration**>  
 <**session-factory**>  
 <**property name**="hibernate.connection.driver\_class">

com.mysql.jdbc.Driver</**property**>

<**property name**="hibernate.connection.url">

jdbc:mysql://127.0.0.1:3306/shop</**property**>  
 <**property name**="hibernate.connection.username">shop</**property**>  
 <**property name**="hibernate.connection.password">shoppass</**property**>  
 <**property name**="hibernate.show\_sql">true</**property**>  
 <**property name**="hibernate.dialect">

org.hibernate.dialect.MySQL8Dialect</**property**>  
 <!-- <property name="hibernate.hbm2ddl.auto">update</property> -->  
 </**session-factory**>  
</**hibernate-configuration**>

Poniżej opis najważniejszych elementów:

connection.driver\_class, connection.url, connection.username, connection.password – informacje o połączeniu z bazą danych: sterownik, adres i port, nazwę, nazwę użytkownika i hasło.

dialect – informacja o rodzaju użytej bazy danych.

<mapping resource="nazwa\_pliku.hbm.xml"/> – deklaracja wszystkich plików mapujących.

* 1. **Sterowanie procesem odwzorowania.**

Aktualnie Hibernate wspiera odwzorowanie za pomocą zarówno adnotacji, jak i XML (pliki .hbm.xml). Jednakże ten drugi sposób jest nie jest już zalecany.[[5]](#footnote-5)

* Pliki mapujące XML

Te pliki zawierają właściwy opis mapowania, czyli tłumaczenia relacyjnego opisu danych na obiektowy i na odwrót. Zazwyczaj jeden plik \*.hbm.xml odpowiada jednej klasie bądź tabeli. Dzieje się tak zwłaszcza wtedy, gdy mamy prostą strukturę, gdzie tabele odpowiadają bezpośrednio klasom. Podstawowe elementy pliku mapującego:

<?xml version=*"1.0"*?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping 3.0.dtd">

<hibernate-mapping>

<class name=*" "* table=*" "*>

<id name=*"id"* type=*"int"* column=*"ID"*>

<generator class=*"increment"*/>

</id>

<property name=*" "*/>

<property name=*" "* column=*" "*/>

<property name=*" "* column=*"* ”

</class>

</hibernate-mapping>

*<class*

*name=" "*

*table=" "*

*>*

Określa klasę i tabelę której dotyczy mapowanie.

*<property*

*name=" "*

*type=" "*

*column=" "*

*length=" "*

*/>*

Opisuje pojedyncze pole i odpowiadającą mu kolumnę w tabeli.

*<many-to-one*

*name=" "*

*class=" "*

*not-null="true"*

*>*

*<column name=" " />*

*</many-to-one>*

Opisu relacji, w tym przypadku wiele-do-jednego.

* Odwzorowanie poprzez adnotacje.

Encja (entity) jest to obiekt służący do reprezentacji trwałych danych. Reprezentuje tabelę z relacyjnej bazy danych. Definiowany jest w formie klasy encji i klas pomocniczych.

Dla klas encji wymagane jest:

- Plain Old Java Object (POJO) z adnotacją @Entity;

- bezargumentowy konstruktor (public lub protected);

- implementacja Serializable, jeśli obiekty będą odłączane;

W projekcie wykorzystano ten sposób mapowania. Poniżej przykład dla tabeli „client”.

@Entity  
@Table(name = "client")  
**public class** Client **implements** Serializable {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.***AUTO***)  
 **private** String id;  
  
 **private** String name;  
 **private** String password;  
 **private** String mail;  
 **private** String LastName;  
 **private** String address;  
  
 **public** String getId() {  
 **return** id;  
 }  
  
 **public void** setId(String id) {  
 **this**.id = id;  
 }

//gettery i settery

}

* 1. **Związki między encjami.**
* Liczność

- 1:1 (@OneToOne);

- 1:N (@OneToMany);

- N:1 (@ManyToOne);

- N:M (@ManyToMany);

* Kierunkowość

- dwukierunkowe;

- jednokierunkowe;

* Kaskada operacji

- PERSIST;

- MERGE;

- REMOVE;

- REFRESH;

Poniżej przedstawiono przykłady asocjacji @ManyToOne i @ManyToMany:

@Entity  
@Table(name = "shopping\_cart")  
**public class** ShoppingCart {  
  
 @Id @GeneratedValue(generator="system-uuid")  
 @GenericGenerator(name="system-uuid", strategy = "org.hibernate.id.UUIDGenerator")  
 **private** String id;  
 **private** String state;  
  
 @ManyToOne(fetch = FetchType.***EAGER***)  
 @JoinColumn(name = "client\_id")  
 **private** Client client;  
  
 @ManyToMany(cascade = {CascadeType.***ALL***})  
 @JoinTable(  
 name = "product\_shopping",  
 joinColumns = {@JoinColumn(name = "s\_id")},  
 inverseJoinColumns = {@JoinColumn(name = "p\_id")})  
 List<Product> products = **new** ArrayList<>();

* 1. **Dziedziczenie encji.**

Encje są zwykłymi klasami Javy. Tym samym jak najbardziej mogą rozszerzać inne klasy, a także być rozszerzane. Model obiektowy pozwala bardzo efektywnie przedstawić złożoność rozwiązywanych problemów biznesowych poprzez sensowne zastosowanie dziedziczenia. Z drugiej zaś strony mamy relacyjną bazę danych, która opiera się o tabele i relacje. Nie za wiele wie ona o dziedziczeniu. Istnieje kilka sposobów na rozwiązanie tego problemu.

- Single – Table Strategy

Zgodnie z nazwą dziedziczenie jest realizowane w oparciu o jedną tabelę w bazie danych.

Automatycznie implikuje to istnienie w niej kolumn, które będą w stanie przechować stan wszystkich encji, które realizują dziedziczenie w oparciu o tę metodę.

- Joined Strategy

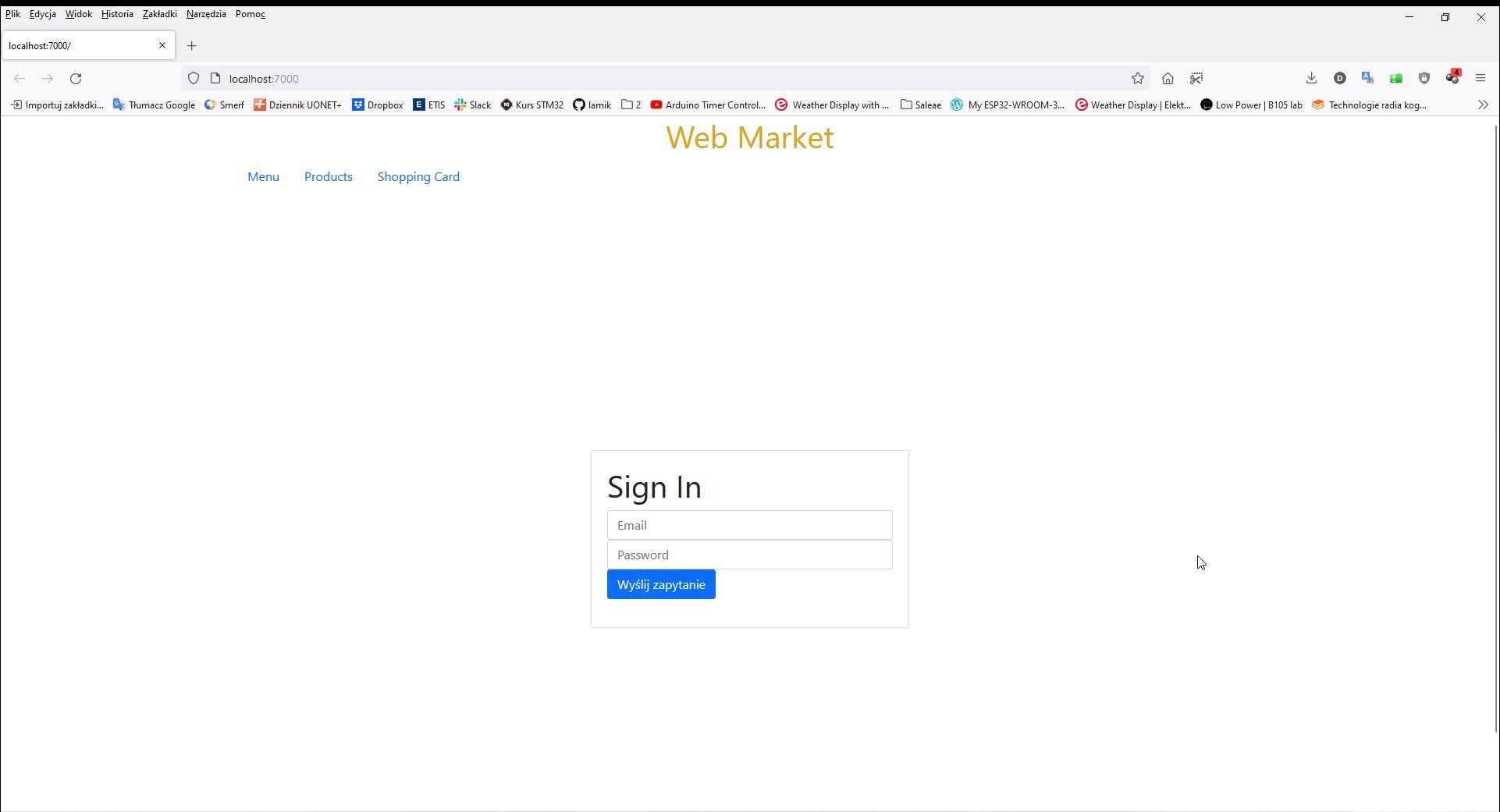
Zastosowanie strategii jednej tabeli może być na dłuższą metę dość zasobożerne. Nagle okazuje się, że w przypadku rozbudowanych hierarchii nasza tabela zawiera dużą ilość kolumn, które w znacznej mierze zawierają wartości NULL. Jeżeli nasza hierarchia dziedziczenia będzie dość rozbudowana można zastosować inną strategię – joined strategy. Wykorzysta ona siłę współczesnych baz danych – relacje.

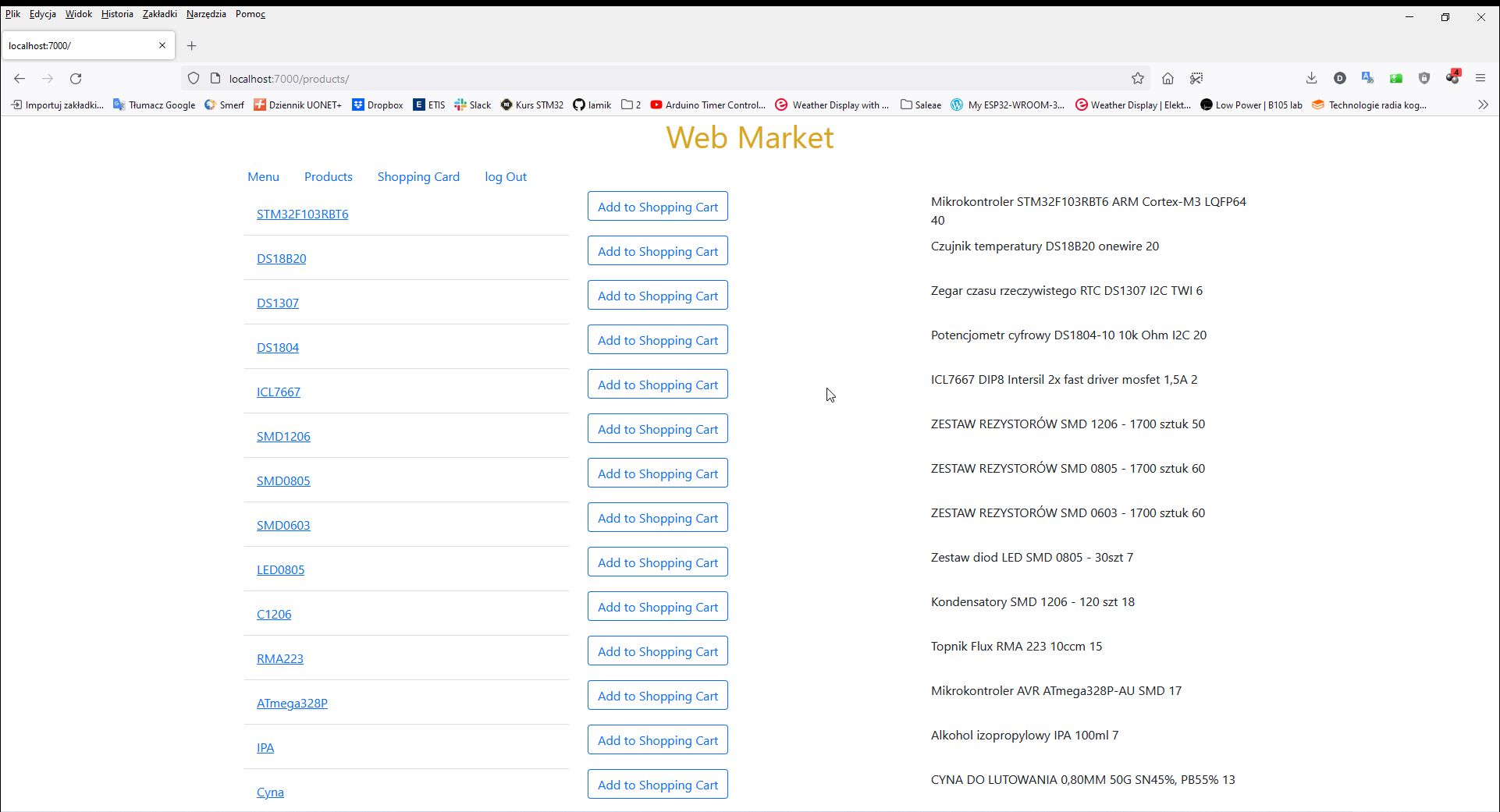
- Table – per – Class Strategy

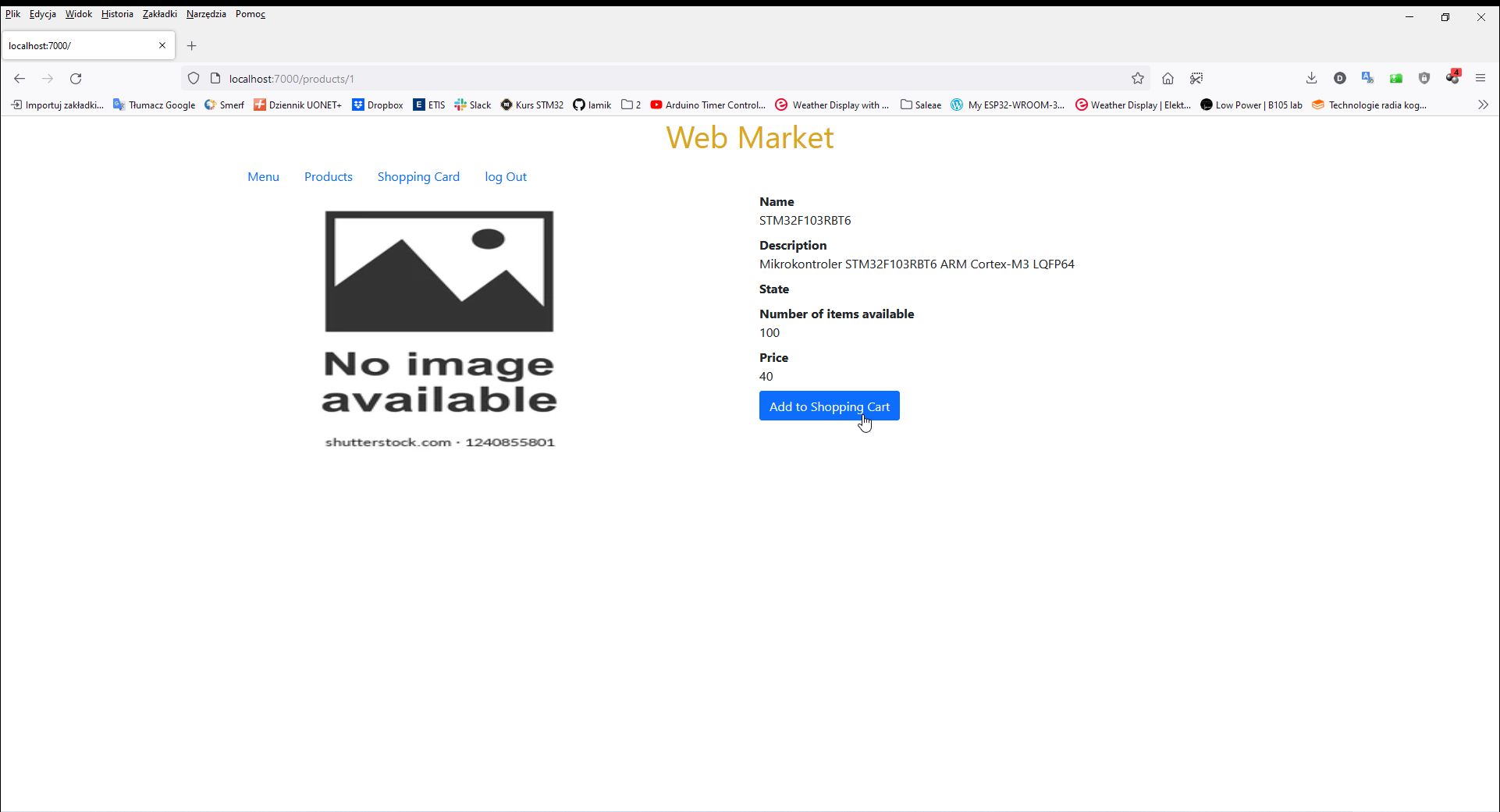
Jest to strategia, w której każda klasa w hierarchii dziedziczenia otrzymuje swoją własną tabelę. Adnotacja: @Inheritance na InheritanceType.TABLE\_PER\_CLASS. W bazie będą występowały informacje w sposób zdublowany. Jednakże rozwiązanie takie jest o wiele szybsze niż strategia relacyjna – zapytanie wybierające rekordy jest kierowane od razu do konkretnej tabeli, z pominięciem złączeń. Zatem o ile miejsce zajmowane przez dane nie jest istotne, albo też dane w bazie są prawie niezmienne, zaś często odpytywane warto zwrócić uwagę na tę strategię.

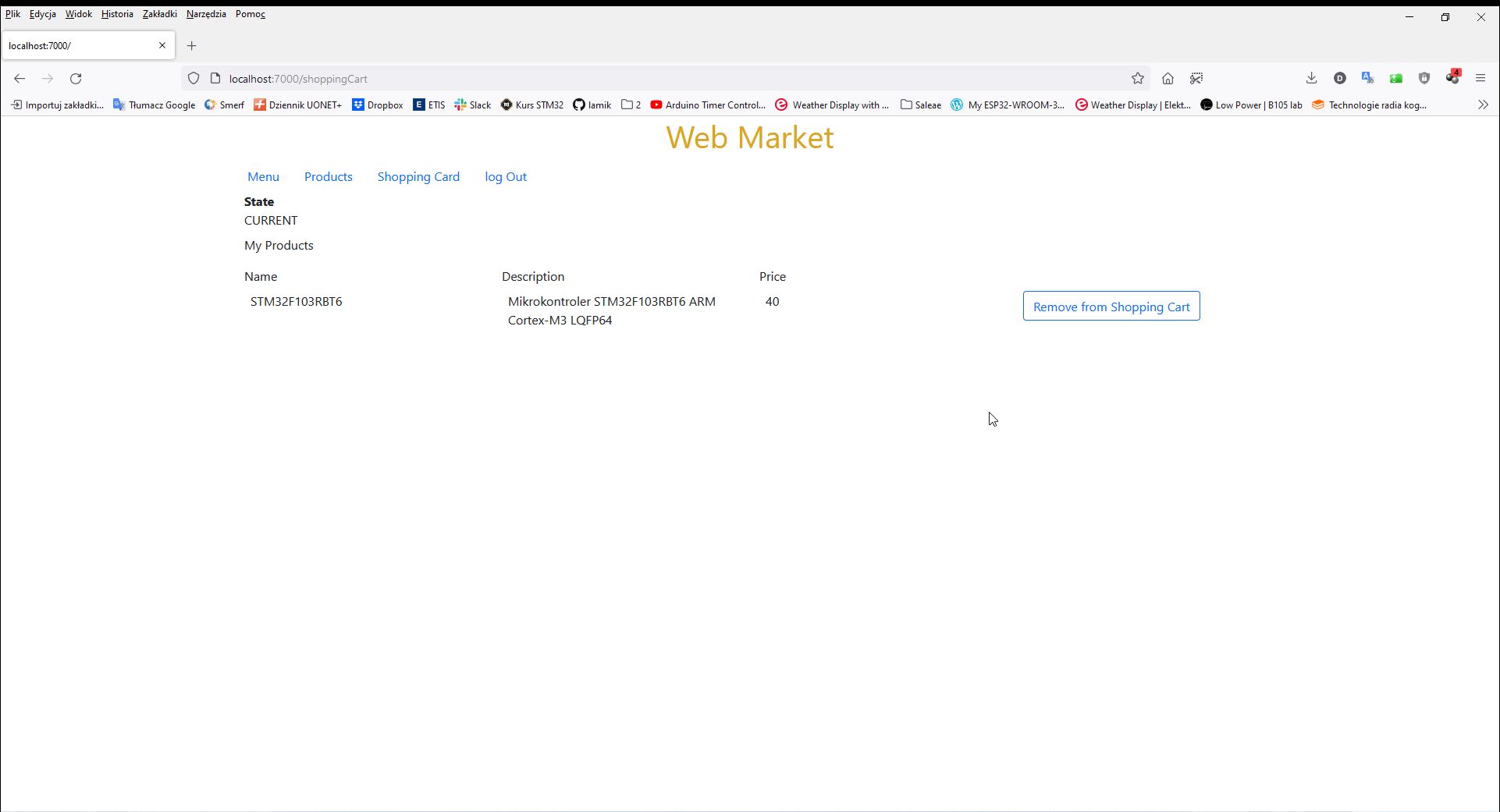
1. **Aplikacja**

Rysunki 6 - 9 przedstawiają okno aplikacji.

*****Rys. 6. Okno aplikacji przed zalogowaniem.*

*****Rys. 7. Okno z listą produktów.*

*****Rys. 8. Okno z opisem produktu.*

*Rys. 9. Okno z zawartością koszyka.*

1. **Wnioski**

Jako zalety Hibernate należy wskazać wygodne mapowanie z wykorzystaniem adnotacji. Mniejsza liczba kodu przekłada się na mniej możliwych błędów. Encje może reprezentować dowolna klasa.

Z drugiej strony dla złożonych danych mapowanie mapowanie z obiektu do tabeli zmniejsza wydajnosć i zwiększa czas konwersji. Hibernate nei pozwala używać niektórych zapytań, które są dostepne w JDBC, np. nie pozwala wstawiać wiele obiekttów do tej samej tabei używając jednego zapytania.

1. <https://hibernate.org/> [↑](#footnote-ref-1)
2. <https://www.mysql.com/> [↑](#footnote-ref-2)
3. <https://examples.javacodegeeks.com/wp-content/uploads/2020/04/Hibernate-Architecture-final.png.webp> [↑](#footnote-ref-3)
4. <https://www.techcret.com/images/resource/orm.png> [↑](#footnote-ref-4)
5. <https://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html> [↑](#footnote-ref-5)