Dokumentacja techniczna

Panel zarządzania produktami dedykowany dla rozwiązań B2B

Uczelnia:

Politechnika Łódzka

Aktualizacja:

Standaryzacja dokumentacji [24.06.2020] autor: Mateusz Domalążek

Temat: Panel zarządzania produktami dedykowany dla rozwiązań B2B z wykorzystaniem Heroku – Cloud Application Platform

Słowa kluczowe: heroku, b2b, sklep internetowy, zarządzanie platformą zakupową,

---

Politechnika Łódzka ul. Żeromskiego 116, 90-924 Łódź, NIP: 727 002 18 95

Daniel Duczymiński, Leon Wojtczak, Mateusz Domalążek, Łukasz Zalewski

Mechanizm kontroli wersji

Numer porządkowy	Imię autora zmiany	Nazwisko autora zmiany	Data i godzina zmiany	Wersja po zmianie	Opis
1	Mateusz	Domalążek	4.06.2020 12:40	1.1	Utworzenie dokumentu
2	Mateusz	Domalążek	5.06.2020 16:40	1.2	Opis środowiska pracy programisty
3	Mateusz	Domalążek	5.06.2020 18:50	1.3	Utworzenie schematu ideowego aplikacji
4	Mateusz	Domalążek	6.06.2020 17:10	1.5	Dodanie diagramu klas
5	Mateusz	Domalążek	6.06.2020 20:30	1.6	Dodanie opisu klas
6	Mateusz	Domalążek	8.06.2020 9:35	1.7	Aktualizacja opisu klas
7	Mateusz	Domalążek	12.06.2020 19:00	1.8	Dodanie metod i pól klas
8	Mateusz	Domalążek	13.06.2020 17:30	1.9	Dodanie opisu metod

9	Daniel	Duczymiński	13.06.2020 19.00	2.0	Dodanie opisu bazy danych (śr.prog.)
10	Mateusz	Domalążek	18.06.2020 20:30	2.1	Podział dokumentu na backend i frontend
11	Mateusz	Domalążek	20.06.2020 21:20	2.2	Utworzenie sekcji testowej
12	Mateusz	Domalążek	21.06.2020 17:40	2.3	Opis plików i funkcji frontendu
13	Mateusz	Domalążek	23.06.2020 20:40	2.4	Aktualizacja testów
14	Mateusz	Domalążek	24.06.2020 3:22	2.5	Analiza statyczna kodu
15					
16					
17					

Spis treści

Mechanizm kontroli wersji **2**

Środowisko pracy **6**

Oprogramowanie aplikacji 6

Środowisko programisty 6

Schemat ideowy aplikacji i hierarchiczny funkcji **7**

Opis ogólny 7

Schemat bazy danych 8

Opis klas metod i funkcji aplikacji **9**

Backend 9

public class TaxesInState 9

public double getLogisticCost() 9

public String getName() 9

public double getBaseTaxValue() 9

public String getTaxForCategory(String category) 10

public class ProfitData 10

public String getNameOfState() 10

public void setNameOfState(String nameOfState) 10

public double getPriceWithoutTaxes() 10

public double getProfit() 10

public void setProfit() 10

public String toString() 11

public class ProfitCalculator 11

public String CalculateForAllStates(String id, String category, String basePrice, String finalPrice) 11

public class Product 11

public int getId() 11

public void setId(int id) 12

public String getName() 12

public void setName(String name) 12

public String getCategory() 12

public void setCategory(String category) 12

public double getPrice() 12

public void setPrice(double price) 12

public class Main 12

public static void main(String[] args) 13

private static String renderContent(String htmlFile) 13

private static String getProductsJsonString() 13

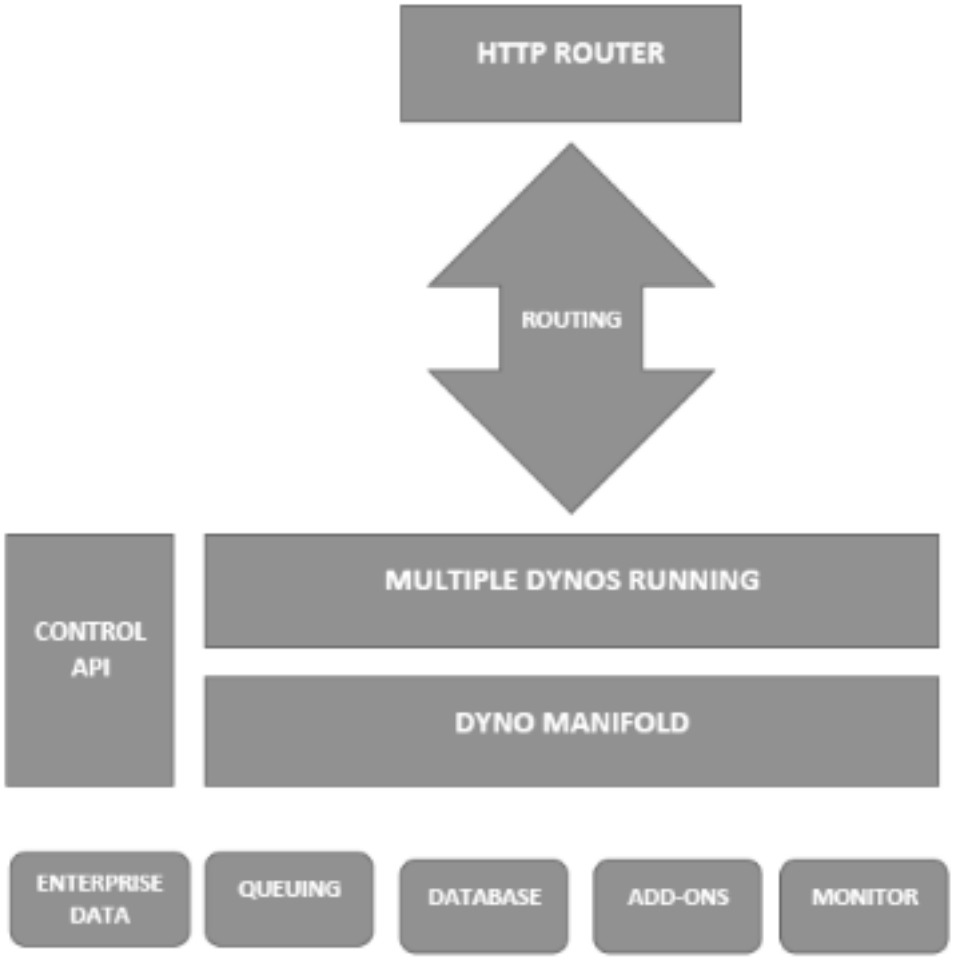
static int getHerokuAssignedPort() 13

public class DataBase 13
public static Connection getConnection() 13
public static ArrayList<Product> getProductById(int id) 13
public static ArrayList<Product> getProductsFromDB() 13
public static void putProductIntoDB(Product pr) 14
public void deleteProductById(int id) 14
public static ArrayList<Taxes> getTaxesInStatesFromDB() 14
public static ArrayList<Taxes> getInterCostsFromDB() 14
public class CsvReader 14
static ArrayList<Product> readProductsFromFile(String path) 14
public class InterCosts 14
Frontend 14
HTML 14
index.html 15
CSS 15
style.css 15
bootstrap.css 15
mdb.css 15
JavaScripts 15
main.js 15
function main() 15
function calculatePriceForEveryProduct() 15
function getPricesFromServer(id) 15
function updateModalContent(data) 16
table-creation.js 16
function createTableFromJSON(data) 16
function createRemoveButton(id) 16
function createShowPricesButton(id) 16
config.js 16

1. Środowisko pracy

Oprogramowanie aplikacji

Heroku – platforma chmurowa stworzona w modelu Platform as a Service obsługująca kilka języków programowania. Aktualnie obsługuje języki Java, JavaScript (Node.js), Scala, Clojure, Python i PHP oraz nieudokumentowany Perl. Podstawowym systemem operacyjnym jest Debian lub bazujący na Debianie Ubuntu.



rys. 1 - architektura Heroku

Środowisko programisty

IntelliJ IDEA – komercyjne zintegrowane środowisko programistyczne (IDE) dla Javy firmy JetBrains.

Windows 10 – system operacyjny wyprodukowany przez amerykańską międzynarodową firmę technologiczną Microsoft i wydany jako część rodziny systemów operacyjnych Windows NT.

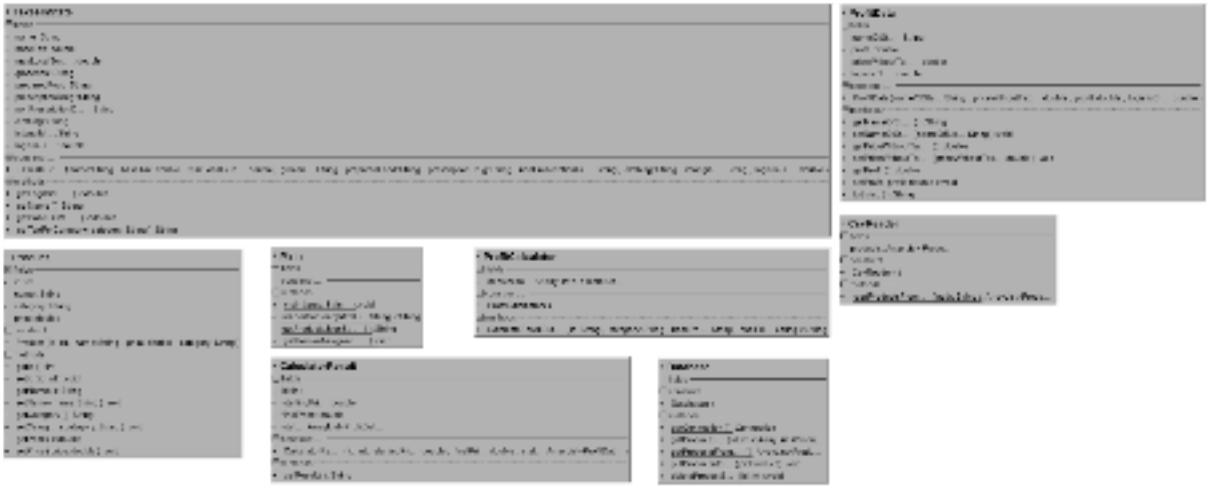
Google Chrome (Wersja 83.0.4103.97) – przeglądarka internetowa rozwijana przez Google. Jej kod został napisany w oparciu o rozwiązania open source częściowo oparte na innych aplikacjach, z wyjątkiem wersji na iOS

PostgreSQL - jeden trzech najpopularniejszych systemów do zarządzania relacyjnymi bazami danych zaraz obok MySQL i SQLite. Wybrany, ponieważ tylko z tym systemem współpracuje Heroku.

2. Schemat ideowy aplikacji i hierarchiczny funkcji

Opis ogólny

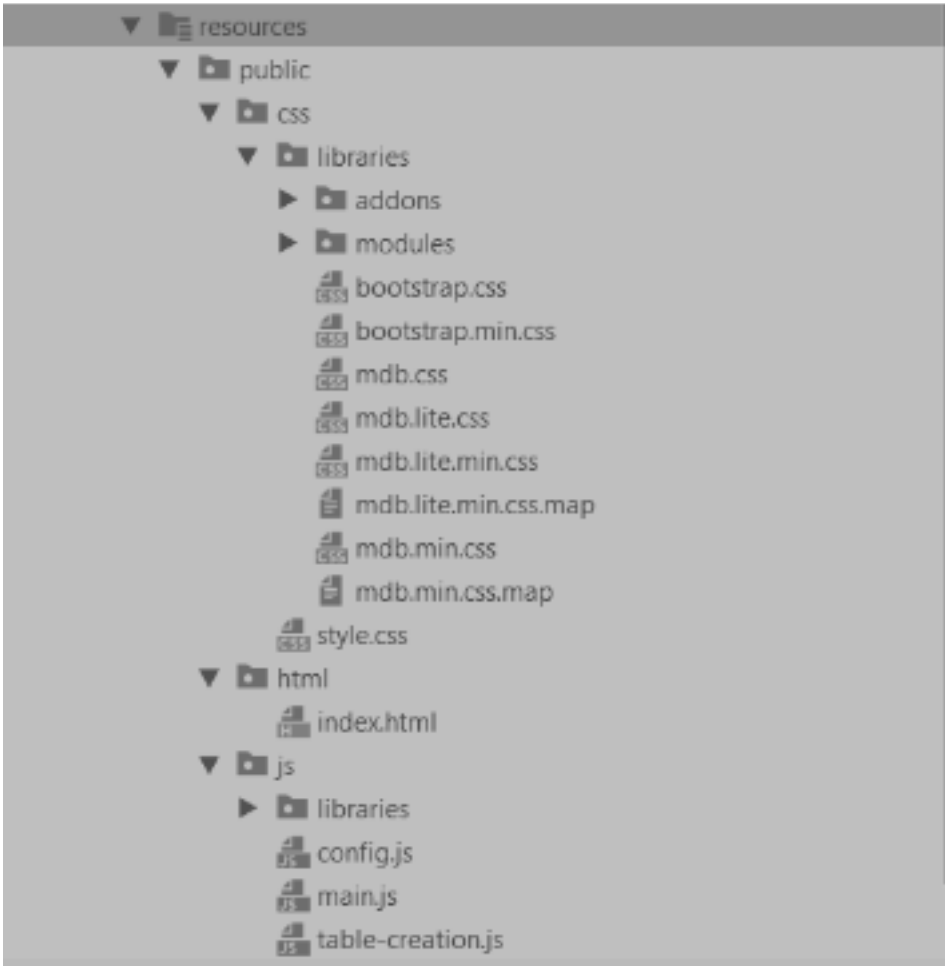
Aplikacja składa się z 8 klas: TaxesInState, ProfitData, Product, Main, ProfitCalculator, CsvReader, CalculatorResult oraz Database.



rys. 2 - diagram klas

Platforma wykorzystuje bibliotekę Bootstrap v4.5.0 – zestaw przydatnych narzędzi ułatwiających tworzenie interfejsu graficznego stron oraz aplikacji internetowych. Bazuje głównie na gotowych rozwiązaniach HTML oraz CSS.

Generowanie dynamicznych elementów witryny odbywa się za pośrednictwem JavaScript – skryptowy język programowania, stworzony przez firmę Netscape, najczęściej stosowany na platformach webowych. W celu usprawnienia działania użyto jQuery – lekka biblioteka programistyczna dla języka JavaScript.



rys. 3 - zasoby platformy

Schemat bazy danych

Baza danych wykorzystuje system PostgreSQL. Znajdują się w niej 4 tabele:

a) TaxesInStates - przedstawia dane na temat wysokości lub braku podatków we wszystkich stanach Ameryki Północnej zgodnie ze stroną: https://en.wikipedia.org/wiki/Sales_taxes_in_the_United_States

b) Products - reprezentuje wszystkie produkty, którymi mógłby być zainteresowany klient. W przyszłości ma zostać zaimplementowana możliwość dodawania własnych produktów.

c) CommonCosts - zawiera informacje o dodatkowych kosztach logistycznych w danych stanach.

d) InternationalTransportationCosts - znajdują się tu dane o zainteresowanych współpracą krajach europejskich, koszta transportu oraz koszta importowania produktów z poszczególnych kategorii.

Wszystkie informacje (oprócz pierwszej tabeli) zostały dostarczone na potrzeby aplikacji przez Product Ownera.

3. Opis klas metod i funkcji aplikacji

Backend

public class TaxesInState

Klasa przechowująca informacje o stawkach podatkowych w danym stanie.

```
private String name;
private double baseTax;
private double maxLocalSurtax;
private String groceries;
private String preparedFood;
private String prescriptionDrug;
private String nonPrescriptionDrug;
private String clothing;
private String intangibles;
private double logisticCost;
```

public double getLogisticCost()

Metoda zwracająca procentową reprezentację kosztu logistycznego, jako liczbę zmiennoprzecinkową podwójnej precyzji.

public String getName()

Metoda zwracająca nazwę stanu.

public double getBaseTaxValue()

Metoda zwracająca procentową reprezentację podstawowej stawki podatkowej, jako liczbę zmiennoprzecinkową podwójnej precyzji.

public String getTaxForCategory(String category)

Metoda zwracająca kategorię opodatkowania na podstawie argumentu.

public class ProfitData

Klasa magazynująca dane o zysku w oparciu o stan

private String nameOfState;

private double profit;

private double priceWithoutTaxes;

private double logisticCost;

public String getNameOfState()

Metoda zwracająca nazwę stanu.

public void setNameOfState(String nameOfState)

Metoda ustawiająca nazwę stanu na nameOfState.

public double getPriceWithoutTaxes()

Metoda pobierająca cenę bez podatku.

public double getProfit()

Metoda pobierająca zysk.

public void setProfit()

Metoda ustawiająca zysk.

public String toString()

Metoda zwracająca wszystkie parametry klasy jako łańcuch znaków.

public class ProfitCalculator

Klasa generująca informacje o cenach produktu w oparciu o stawki podatkowe według określonej kategorii.

private ArrayList<TaxesInState> taxesInStates;

Tablica przechowywująca elementy klasy TaxesInState.

public String CalculateForAllStates(String id, String category, String basePrice, String finalPrice)

Metoda zwracająca obliczenia dla stanów.

public class Product

Klasa przechowująca informacje o numerze identyfikacyjnym produktu, nazwie, kategorii oraz cenie.

private int id;

private String name;

private String category;

private double price;

public int getId()

Metoda zwracająca id produktu.

public void setId(int id)

Metoda ustawiająca id produktu.

public String getName()

Metoda pobierająca nazwę produktu.

public void setName(String name)

Metoda ustawiająca nazwę produktu.

public String getCategory()

Metoda pobierająca kategorię produktu.

public void setCategory(String category)

Metoda ustawiająca kategorię produktu.

public double getPrice()

Metoda pobierająca cenę produktu.

public void setPrice(double price)

Metoda ustawiająca cenę produktu.

public class Main

Główna klasa aplikacji generująca kontent oraz konfigurująca połączenie z platformą Heroku.

public static void main(String[] args)

Główna metoda aplikacji pobierająca argumenty.

private static String renderContent(String htmlFile)

Metoda renderująca kontent na podstawie htmlFile.

private static String getProductsJsonString()

Metoda pobierająca produkty.

static int getHerokuAssignedPort()

Metoda pobierająca przydzielony port Heroku.

public class DataBase

Klasa odpowiedzialna za połączenie z bazą danych oraz obsługująca zapytania SQL.

public static Connection getConnection()

Metoda zawierająca połączenie z bazą danych na serwerze.

public static ArrayList<Product> getProductById(int id)

Metoda pobierająca produkty na podstawie id z bazy danych. Elementy zapisywane są do tablicy reprezentującej elementy klasy Product.

public static ArrayList<Product> getProductsFromDB()

Metoda pobierająca produkty z bazy danych. Elementy zapisywane są do tablicy reprezentującej elementy klasy Product.

public static void putProductIntoDB(Product pr)

Metoda dodająca produkt do bazy danych.

public void deleteProductById(int id)

Metoda usuwająca produkt z bazy danych o podanym w argumencie id.

public static ArrayList<Taxes> getTaxesInStatesFromDB()

Metoda zwracająca informacje o podatkach w Stanach Zjednoczonych.

public static ArrayList<Taxes> getInterCostsFromDB()

Metoda zwracająca rekordy z kosztami handlu międzykontynentalnego.

public class CsvReader

Klasa przetwarzająca dane z pliku .csv oraz wczytująca produkty z zewnętrznego pliku do tablicy produktów.

private ArrayList<Product> products;

Tablica przechowująca produkty.

static ArrayList<Product> readProductsFromFile(String path)

Metoda pobierająca produkty z pliku .csv do tablicy produktów.

public class InterCosts

Klasa przechowująca informację na temat międzynarodowych kosztów transportu.

Frontend

HTML

index.html

Główny plik odpowiedzialny za komunikację z użytkownikiem i osadzanie elementów aplikacji. Jest wejściem umożliwiającym eksplorację serwisu.

CSS

style.css

Arkusz ustawiający styl wyświetlania tekstu w kolumnie ceny końcowej w tabeli produktów.

bootstrap.css

Bootstrap v4.5.0
Licensed under MIT

mdb.css

Material Design for Bootstrap 4
Version: MDB FREE 4.19.0

JavaScripts

main.js

Główna plik importujący wszystkie skrypty

function main()

Funkcja generująca tabelę produktów, które umieszczone są w tabeli Produkty w bazie danych oraz uruchamiająca pozostałe skrypty.

function calculatePriceForEveryProduct()

Funkcja pobierająca informacje o cenie dla każdego z produktów.

function getPricesFromServer(id)

Funkcja pobierająca informację o cenie dla produktu o danym id.

function updateModalContent(data)

Funkcja generująca okno przeznaczone do obliczania cen

table-creation.js

Plik generujący tabelę z produktami.

function createTableFromJSON(data)

Funkcja renderująca tabelę produktów.

function createRemoveButton(id)

Funkcja tworząca przycisk do usuwania rekordów z tabeli.

function createShowPricesButton(id)

Funkcja tworząca przycisk do uruchomienia okna przeznaczonego do obliczeń.

config.js

Plik konfiguracyjny dla zaplecza skryptowego i wizualnego. Przechowuje dane tekstowe dla przycisków oraz nagłówków tabel i pozostałych elementów.

Testy

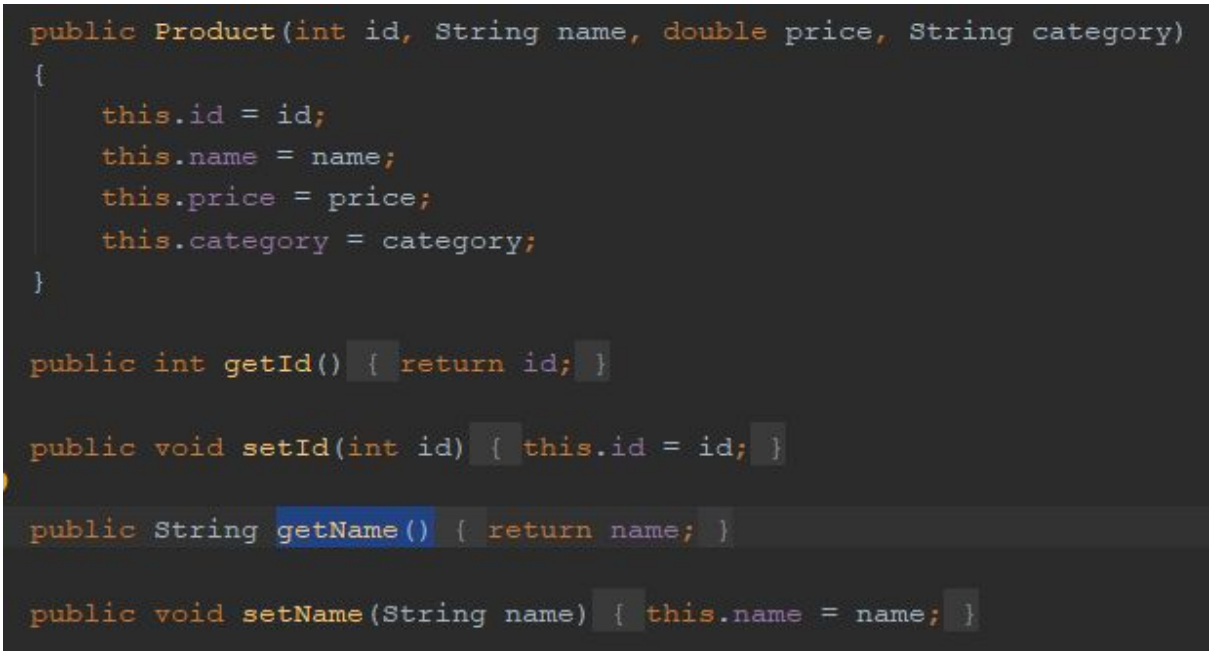
Analiza statyczna

Podczas badania bezpieczeństwa aplikacji udało się wyeliminować wiele złych praktyk w tworzeniu aplikacji. Zabezpieczono przechwytywanie parametrów przekazywanych do metod oraz funkcji, a także dokonano refaktoryzacji kodu.W pobieraniu i dodawaniu elementów do bazy danych wyeliminowano wstrzykiwanie kodu.

```
Connection conn = getConnection();
//Statement statement = conn.createStatement();
//String query = "SELECT * from public.\"Produkty\" WHERE id=" + String.valueOf(id) + ";";
//ResultSet result = statement.executeQuery(query);
ArrayList<Product> products = new ArrayList<>();

PreparedStatement statement = conn.prepareStatement( sql: "SELECT * from public.\"Produkty\" WHERE id= ?");

statement.setInt( parameterIndex: 1, id);

ResultSet result = statement.executeQuery();
```

rys. 4 - eliminacja SQL injection

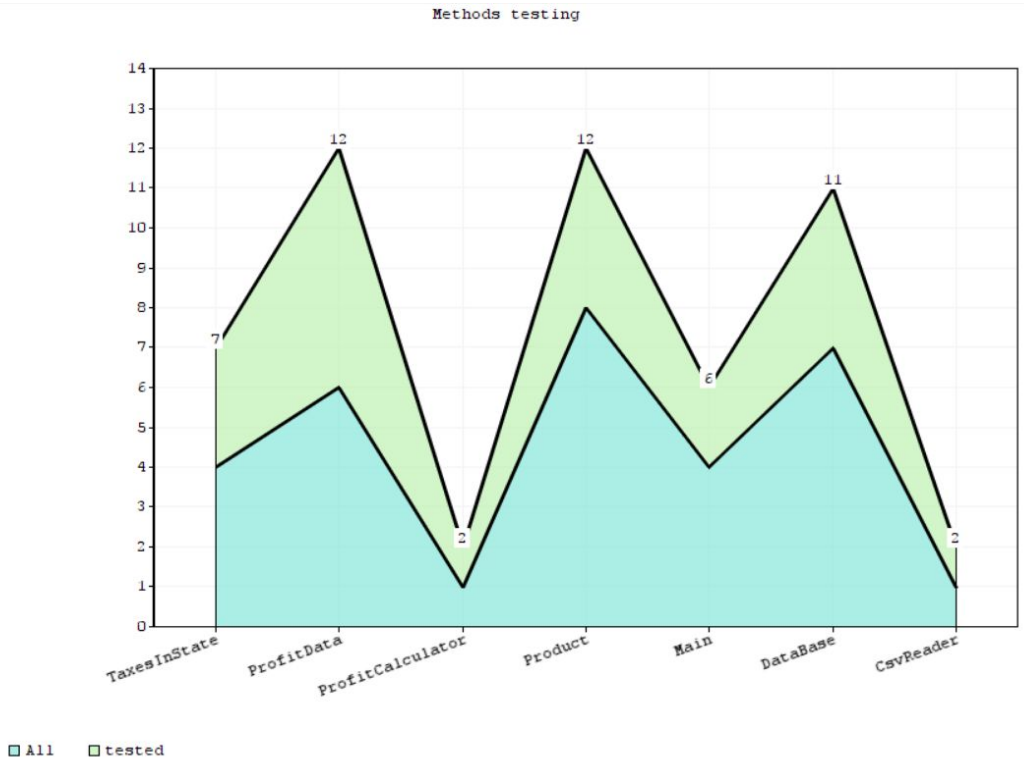
Klasy i metody zostały opisane zgodnie z obowiązującymi standardami. Nazwy są ściśle powiązane z ich przeznaczeniem. Stanowi to zdecydowane ułatwienie w przyszłym rozbudowywaniu aplikacji, a także analizowaniu kodu.



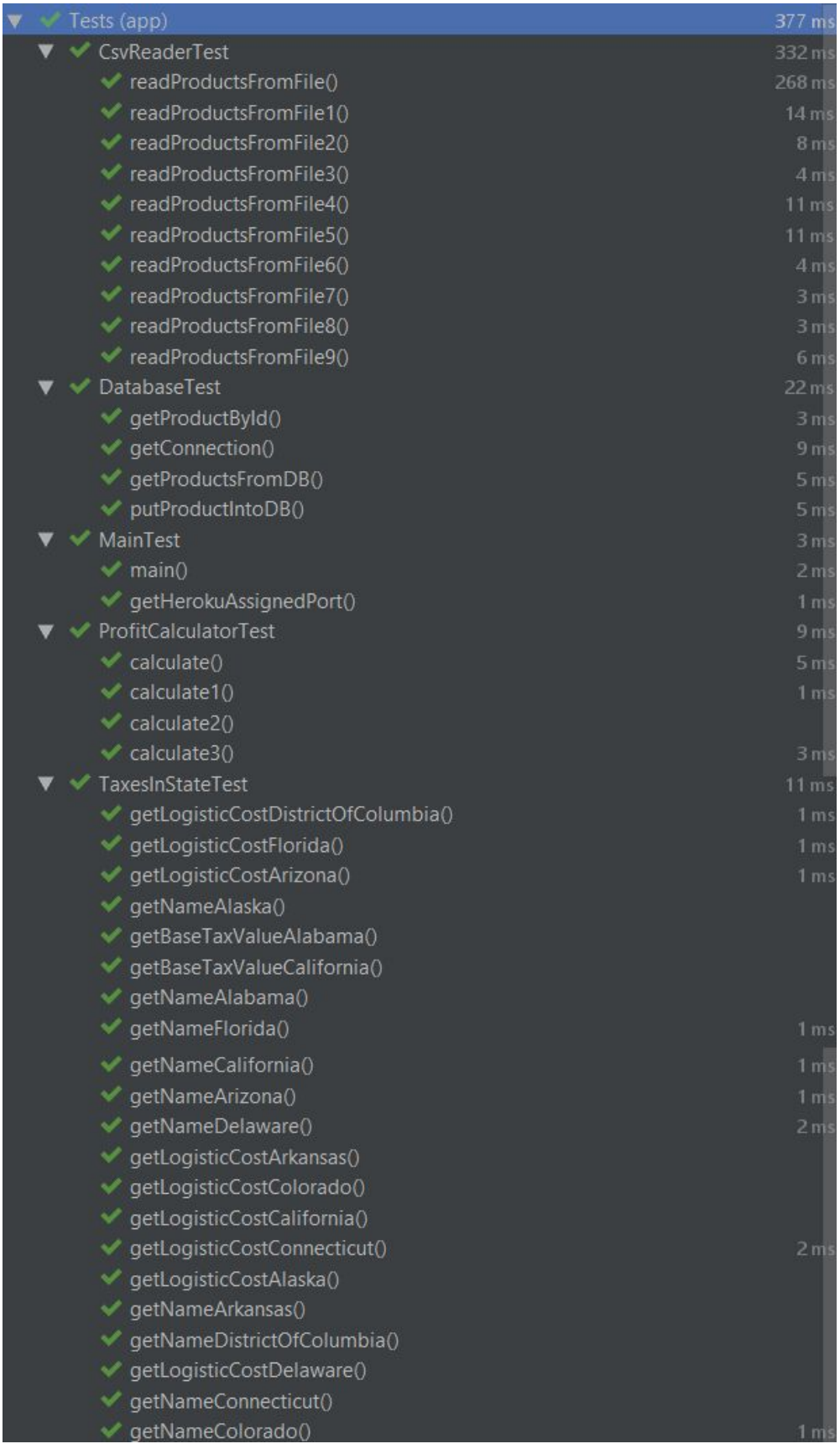
rys. 5 - standaryzacja kodu

Testy jednostkowe

Backend aplikacji został dokładnie przetestowany. Przygotowano 51 niezbędnych testów jednostkowych, których zadaniem było poszukiwanie błędów w najważniejszych klasach aplikacji.



rys. 7 - pokrycie kodu testami



rys. 6 - wyniki testów jednostkowych