Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

WHERESHOPPING.COM

PIOTR ANTOSZ

Kraków 2011

Spis treści

1. Projekt konceptualny

1.1. Sformułowanie zadania projektowego

Projekt zakłada stworzenie aplikacji internetowej umożliwiającej uzyskanie odpowiedzi na pytanie: „Gdzie najtaniej zrobię zakupy w mojej okolicy?”. Aplikacja ma pomagać zmniejszyć wydatki na produkty codziennego użytku, poprzez dostarczenie informacji o najbardziej korzystnej cenowo ofercie w danym czasie i miejscu. Rozwiązanie pozwoli zaoszczędzić czas poświęcony na poszukiwanie konkretnych produktów. Portal przyciągnie także uwagę ze strony oferujących produkty, pozwalając im na reklamę swojej oferty i zainteresowanie potencjalnych klientów. Serwis umożliwi stworzenie własnej listy zakupów, następnie wskazanie obszaru poszukiwań, w którym chcemy się poruszać w trakcie zakupów. Na podstawie tych informacji system wyznaczy miejsce w jakim najlepiej zakupić wybrane produkty. Umożliwi to uniknięcie pułapek cenowych zastawianych przez część sklepów, kuszących niskimi cenami jedynie części towaru. Wiarygodność i aktualność ofert zamieszczanych przez właścicieli sklepów będzie podlegać weryfikacji ze strony klientów. Dostaną oni możliwość wypowiedzenia się na temat danego sklepu, jego oceny. Oprócz serwisu pozwalającego na zaplanowanie swoich zakupów, zostanie stworzony panel umożliwiający tworzenie ofert sklepów przez ich przedstawicieli. Oferty sklepów jak i obszary przeszukiwań będą obejmować dowolną lokalizację. Wybór najkorzystniejszej oferty będzie określany na podstawie różnych kryteriów, np. (najniższa cena sumaryczna w jednym sklepie, najniższe ceny produktów we wszystkich sklepach, najniższa cena sumaryczna we wszystkich sklepach).

1.2. Analiza stanu wyjściowego

Informatyzowany system do tej pory realizowany był poprzez zrobienie zakupów w wielu sklepach, a następnie przeanalizowanie cen w każdym z nich. Po takim rozpoznaniu klient wiedział, w którym sklepie spodziewać się najniższych cen, jeśli często nabywa podobne produkty. Tworzony serwis umożliwi przyspieszenie i udoskonalenie tego procesu, oraz dostarczy nowych funkcjonalności.

Prezentowana oferta sklepów nie będzie stanowić oferty w rozumieniu prawa handlowego. Jednak w celu utrzymania dobrej opinii, właściciele sklepów będą musieli zadbać o jej odwzorowanie w rzeczywistości.

W Internecie znaleźć można kilka podobnych rozwiązań, jednak dotyczą one innego rodzaju produktów, analizując obszar bardziej globalny. Nie pozwalają one także na porównanie cen grupy produktów, a jedynie konkretnego artykułu. W typowych porównywarkach cen zwykle nie występują kategorie obejmujące asortyment codziennego użytku, np. spożywczy. Serwisy takie jak ceneo.pl, skapiec.pl, kupujemy.pl, pozwalają uzyskać informację o pojedynczym produkcie, zawężając jego poszukiwania do obrębu, minimalnie, miasta. Model biznesowy wymienionych portali bazuje na pobieraniu opłat od agregowanych sklepów.

1.3. Analiza wymagań użytkownika

Z serwisu będą korzystać użytkownicy publikujący swoją ofertę, przedstawiciele sklepów (Sprzedawcy), a także osoby chcące zbudować swoją listę zakupów (Klienci). Klienci zostaną pogrupowani

na Anonimowych i Zalogowanych. Konta Sprzedawców będą zakładane poza funkcjonalnością serwisu, aby zapewnić większą wiarygodność oferentów.

1.3.1. Sprzedawca

- wskazanie lokalizacji sklepu
- dodawanie produktu do oferty
- zmiana ceny produktów, stanu magazynowego
- tworzenie wizytówki

1.3.2. Klient Anonimowy

- rejestracja konta
- przegląd mapy z lokalizacjami sklepów
- podgląd wizytówek sklepów
- dodawanie produktów do listy zakupów
- określanie ilości produktów na liście (sztuki/waga/objętość)

1.3.3. Klient Zalogowany

- zaznaczenie nieregularnego obszaru przeszukiwań
- poszukiwanie najkorzystniejszej oferty
- wybór kategorii poszukiwań
- wydruk listy zakupów
- zapis listy zakupów w historii
- przeglądanie archiwum zakupów
- zmiana danych personalnych
- wydanie opinii o sklepie

1.4. Określenie scenariuszy użycia

1.4.1. Rejestracja nowego Klienta

Klient Anonimowy chce zarejestrować nowe konto w serwisie, aby mieć dostęp do pełnej funkcjonalności.

- Klient Anonimowy wybiera funkcję „Rejestruj”
- system wyświetla formularz tworzenia nowego konta
- Klient Anonimowy wprowadza swoje dane (login, imię, nazwisko, adres, datę urodzenia)
- Klient Anonimowy wprowadza dwa razy ten sam adres E-mail

- Klient Anonimowy wprowadza dwa razy to samo hasło (min. 8 znaków, min. jedna cyfra, min. jedna duża i mała litera)
- Klient Anonimowy akceptuje regulamin serwisu
- Klient Anonimowy poprawnie przepisuje symbol Captcha
- Klient Anonimowy wybiera funkcję „Zapisz”
- system dodaje nowego użytkownika
- system wyświetla stronę główną serwisu

1.4.2. Przeglądanie dostępnych sklepów

Klient przegląda mapę z naniesionymi lokalizacjami sklepów, chce zapoznać się z wizytówkami oglądanych sklepów.

- Klient wskazuje na mapie obszar, który chce oglądnąć
- system wyświetla mapę wraz z lokalizacjami sklepów w danym obszarze
- Klient wybiera sklep, którego wizytówkę chce zobaczyć
- system wyświetla wizytówkę sklepu (nazwa, opis, godziny otwarcia, adres, lokalizacja na mapie, dane kontaktowe, opinie o sklepie)

1.4.3. Zarządzanie ofertą przez Sprzedawcę

Sprzedawca chce zmodyfikować swoją wizytówkę, rozbudować/zaktualizować swoją ofertę produktów.

- Sprzedawca loguje się do swojego panelu
- system wyświetla jego profil i ofertę
- Sprzedawca uzupełnia swoją wizytówkę (nazwa, opis, godziny otwarcia, adres, lokalizacja na mapie, dane kontaktowe)
- Sprzedawca dodaje produkt do swojej oferty (kategoria, podkategoria, producent, nazwa, opis, jednostka miary, ilość)
- Sprzedawca określa cenę produktów, zapas magazynowy
- Sprzedawca zapisuje swój profil

1.4.4. Tworzenie listy zakupów

Klient chce stworzyć listę zakupów w interesującym go obszarze przeszukiwań, aby wyznaczyć najkorzystniejszą ofertę.

- system wyświetla listę produktów
- Klient przegląda katalog dostępnych produktów
- Klient wybiera produkty do listy zakupów, określa ich ilość
- Klient loguje się do systemu

- Klient zaznacza na mapie obszar poszukiwania
- Klient uruchamia symulację
- system zwraca najkorzystniejszą ofertę
- Klient określa kryterium korzyści (najniższa cena sumaryczna w jednym sklepie, najniższa cena sumaryczna we wszystkich sklepach)
- Klient zapisuje wynik symulacji

1.4.5. Przeglądanie historii list zakupów

Klient chce przeglądnąć wygenerowane listy zakupów w celu ich wydrukowania albo dokonania oceny wiarygodności Sprzedawcy.

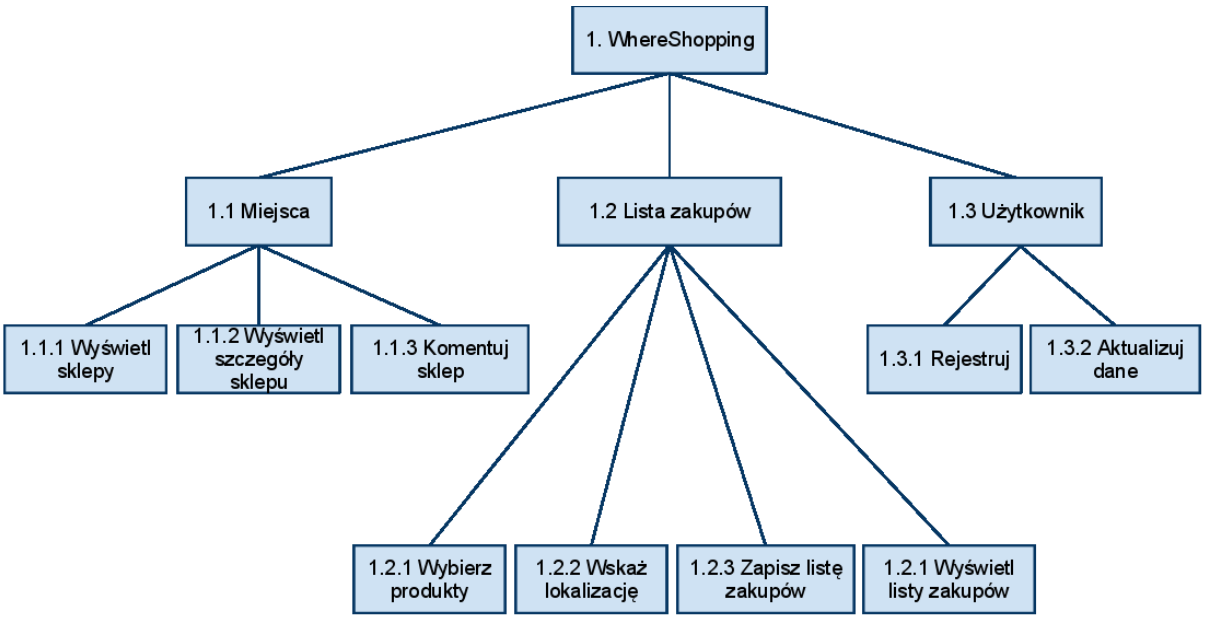
- Klient loguje się do swojego panelu
- system wyświetla stronę główną
- Klient przegląda historię swoich list zakupów
- Klient wybiera listę zakupów do wydrukowania
- Klient wybiera Sprzedawcę związanego z daną listą zakupów
- Klient ocenia sklep wskazując rating (0-7), podając komentarz tekstowy

1.5. Identyfikacja funkcji

Funkcje realizowane w bazie danych zidentyfikowane na podstawie specyfikacji wymagań systemu:

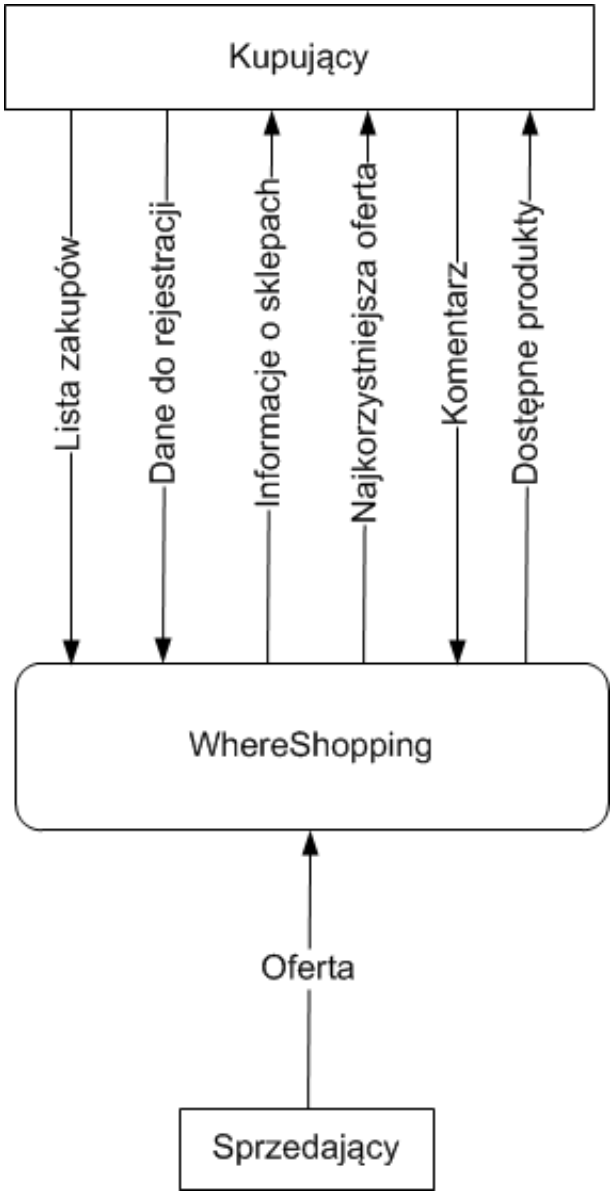
- pobranie nazw sklepów mieszczących się na wybranym obszarze
- pobranie szczegółów odnośnie wybranego sklepu (wizytówka + komentarze)
- pobranie listy dostępnych produktów (w danej kategorii i podkategorii)
- wyszukanie oferty sklepu (ustalony obszar poszukiwań, ilość i rodzaj produktów)
- zapisanie listy zakupów
- pobranie list zakupów danego użytkownika
- pobranie szczegółów dotyczących listy zakupów
- zapisanie komentarza
- zapisanie nowego użytkownika, zmiana danych osobowych

1.6. Analiza hierarchii funkcji projektowanej aplikacji (ang. Functional Hierarchy Diagram - FHD)

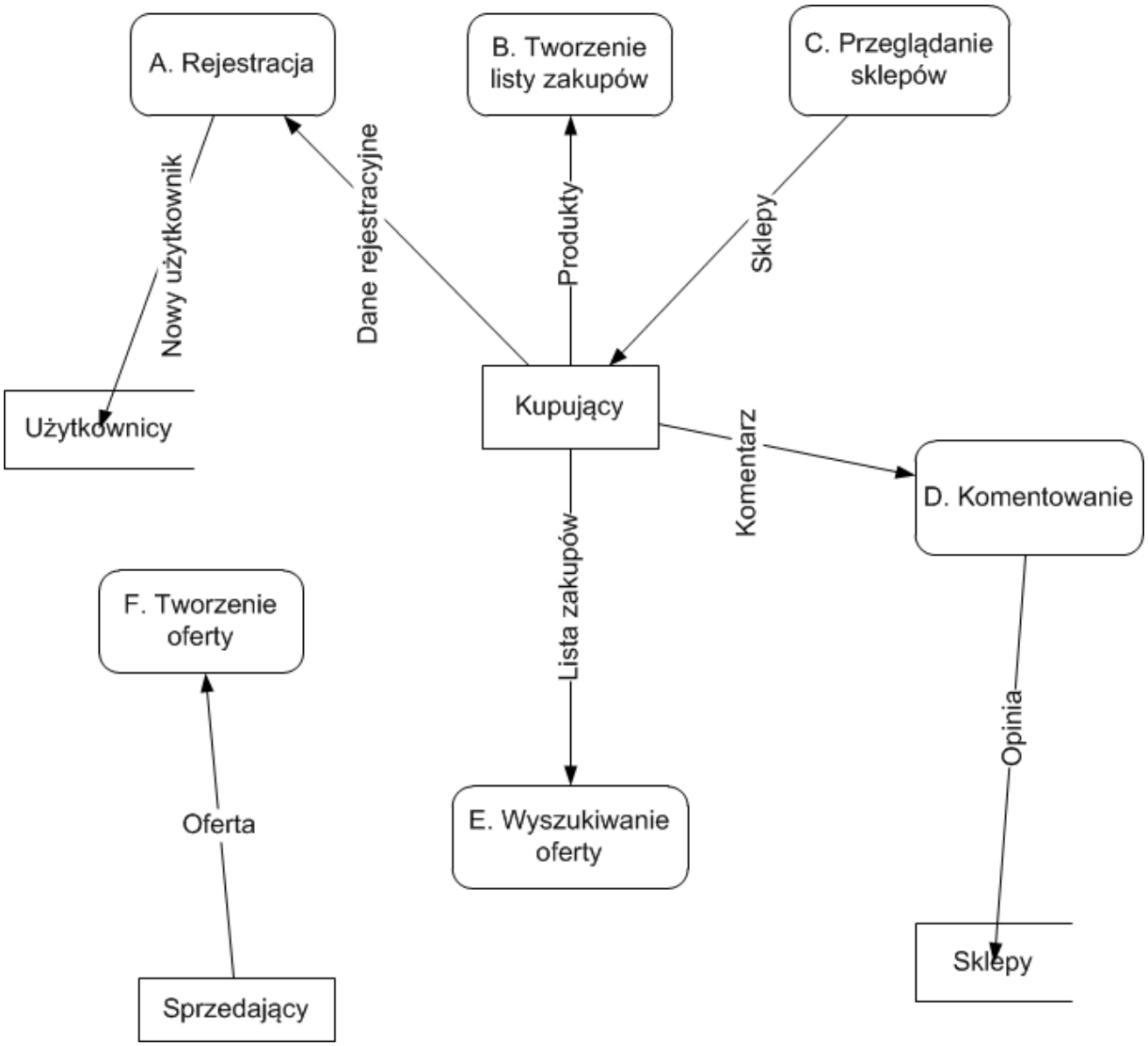


Rysunek 1.1: Analiza hierarchii funkcji

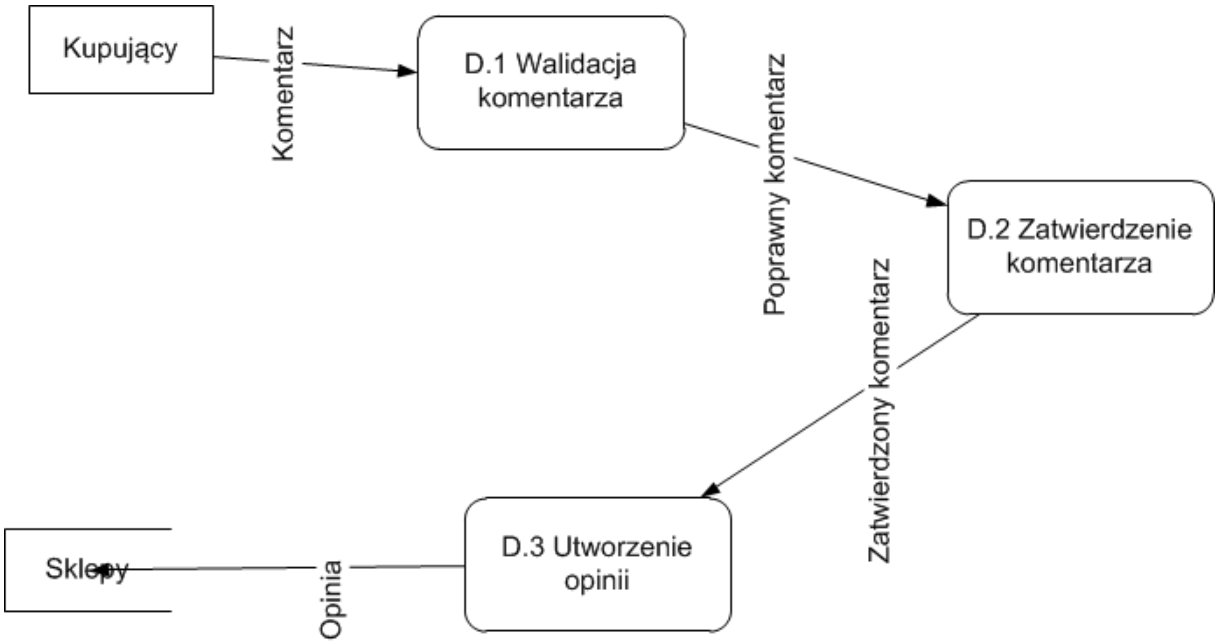
1.7. Budowa i analiza diagramu przepływu danych (ang. Data Flow Diagram - DFD)



Rysunek 1.2: Diagram kontekstowy

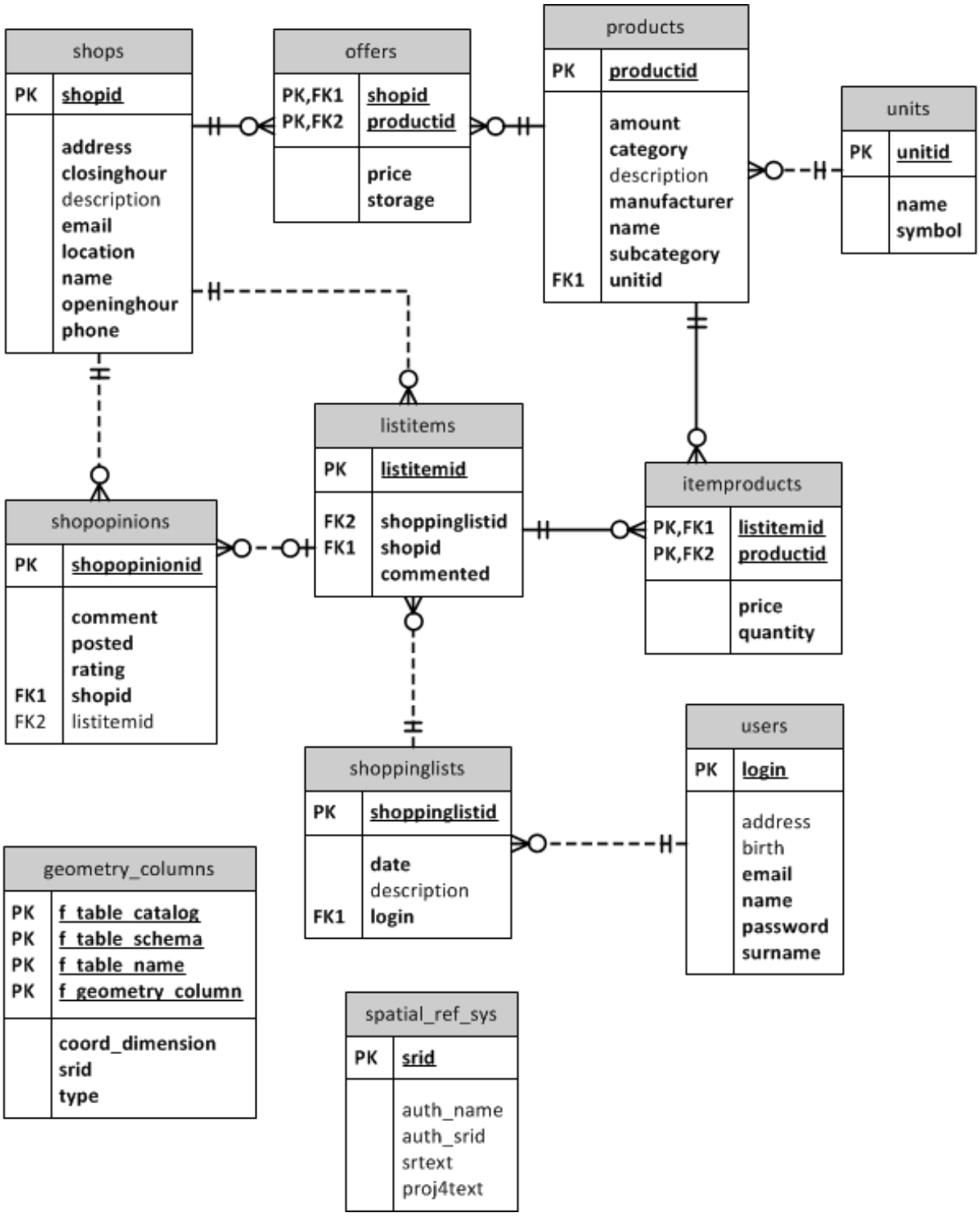


Rysunek 1.3: Diagram przepływu danych



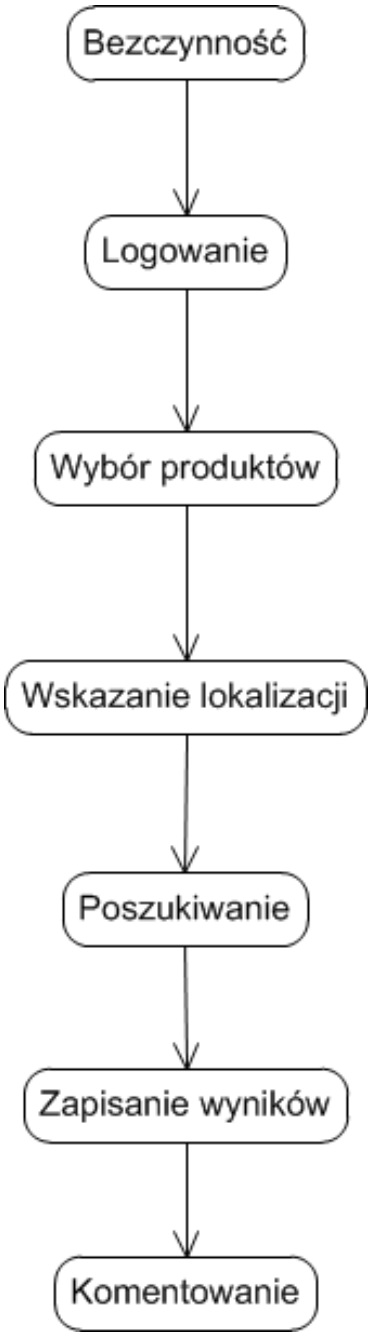
Rysunek 1.4: Diagram przepływu danych - dekompozycja

1.8. Encje, atrybuty, powiązania.



Rysunek 1.5: Diagram ERD

1.9. Projekt diagramów STD (ang. State Transition Diagram)



Rysunek 1.6: Diagram STD

2. Projekt logiczny

2.1. Projektowanie tabel, kluczy, kluczy obcych, powiązań między tabelami, indeksów.

```
@Entity
@Table(name = "itemproducts")
class ItemProduct implements Serializable{

    private Double price;
    private Integer quantity;
    private Product product;
    private Integer listItemId;

    public ItemProduct(){
    }

    @NotNull
    @Column
    public Double getPrice() {
        return price;
    }

    public void setPrice(Double price) {
        this.price = price;
    }

    @NotNull
    @Column
    public Integer getQuantity() {
        return quantity;
    }

    public void setQuantity(Integer quantity) {
        this.quantity = quantity;
    }

    @Id
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "productid")
    public Product getProduct() {
        return product;
```

```
    }

    public void setProduct(Product product) {
        this.product = product;
    }

    @Id
    @Column
    public Integer getListItemId() {
        return listItemId;
    }

    public void setListItemId(Integer listItemId) {
        this.listItemId = listItemId;
    }

}

@Entity
@Table(name = "listitems")
class ListItem {

    private Integer listItemId;
    private Integer shoppingListId;
    private Shop shop;
    private Boolean commented = false;
    private Set<ItemProduct> itemProducts = new HashSet<ItemProduct>();

    public ListItem(){
    }

    @Id
    @GeneratedValue
    public Integer getListItemId() {
        return listItemId;
    }

    public void setListItemId(Integer listItemId) {
        this.listItemId = listItemId;
    }

    @NotNull
    @Column
    public Integer getShoppingListId() {
        return shoppingListId;
    }

    public void setShoppingListId(Integer shoppingListId) {
        this.shoppingListId = shoppingListId;
    }
```

```
    @NotNull
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "shopid")
    public Shop getShop() {
        return shop;
    }

    public void setShop(Shop shop) {
        this.shop = shop;
    }

    @NotNull
    @Column
    public Boolean getCommented() {
        return commented;
    }

    public void setCommented(Boolean commented) {
        this.commented = commented;
    }

    @OneToMany(mappedBy = "listItemId", fetch = FetchType.EAGER)
    @Cascade({CascadeType.SAVE_UPDATE, CascadeType.DELETE})
    public Set<ItemProduct> getItemProducts() {
        return itemProducts;
    }

    public void setItemProducts(Set<ItemProduct> itemProducts) {
        this.itemProducts = itemProducts;
    }

    public void addItemProduct(ItemProduct itemProduct){
        itemProduct.setListItemId(getListItemId());
        itemProducts.add(itemProduct);
    }

}

@Entity
@Table(name = "offers")
class Offer implements Serializable{

    private Integer shopId;
    private Double price;
    private Integer storage;
    private Product product;

    public Offer(){
    }

    @Id
```

```
    public Integer getShopId() {
        return shopId;
    }

    public void setShopId(Integer shopId) {
        this.shopId = shopId;
    }

    @NotNull
    @Column
    public Double getPrice() {
        return price;
    }

    public void setPrice(Double price) {
        this.price = price;
    }

    @NotNull
    @Column
    public Integer getStorage() {
        return storage;
    }

    public void setStorage(Integer storage) {
        this.storage = storage;
    }

    @Id
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "productId")
    public Product getProduct() {
        return product;
    }

    public void setProduct(Product product) {
        this.product = product;
    }

}

@Entity
@Table(name = "products")
class Product {

    private Integer productId;
    private Double amount;
    private String category;
    private String description;
    private String manufacturer;
    private String name;
```

```
private String subcategory;
private Unit unit;

public Product(){
}

@Id
@GeneratedValue
public Integer getProductId() {
    return productId;
}

public void setProductId(Integer productId) {
    this.productId = productId;
}

@NotNull
@Column
public Double getAmount() {
    return amount;
}

public void setAmount(Double amount) {
    this.amount = amount;
}

@NotNull
@Column
public String getCategory() {
    return category;
}

public void setCategory(String category) {
    this.category = category;
}

@Column
public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

@NotNull
@Column
public String getManufacturer() {
    return manufacturer;
}
```

```
    public void setManufacturer(String manufacturer) {
        this.manufacturer = manufacturer;
    }

    @NotNull
    @Column
    public String getName() {
        return name;
    }

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

    @NotNull
    @Column
    public String getSubcategory() {
        return subcategory;
    }

    public void setSubcategory(String subcategory) {
        this.subcategory = subcategory;
    }

    @NotNull
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "unitid")
    @Cascade(CascadeType.SAVE_UPDATE)
    public Unit getUnit() {
        return unit;
    }

    public void setUnit(Unit unit) {
        this.unit = unit;
    }

}

@Entity
@Table(name = "shops")
class Shop {

    private Integer shopId;
    private String address;
    private Time[] closingHour;
    private String description;
    private String email;
    private Point location;
    private String name;
    private Time[] openingHour;
    private String phone;
```

```
private Set<ShopOpinion> opinions = new HashSet<ShopOpinion>();
private Set<Offer> offers = new HashSet<Offer>();

public Shop(){

}

@Id
@GeneratedValue
public Integer getShopId() {
    return shopId;
}

public void setShopId(Integer shopId) {
    this.shopId = shopId;
}

@NotNull
@Column
public String getAddress() {
    return address;
}

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

@NotNull
@Column
public Time[] getClosingHour() {
    return closingHour;
}

public void setClosingHour(Time[] closingHour) {
    this.closingHour = closingHour;
}

@Column
public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

@NotNull
@Column
public String getEmail() {
    return email;
}
```

```
public void setEmail(String email) {
    this.email = email;
}

@NotNull
@Column
@Type(type = "org.hibernatespatial.GeometryUserType")
public Point getLocation() {
    return location;
}

public void setLocation(Point location) {
    this.location = location;
}

@NotNull
@Column
public String getName() {
    return name;
}

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

@NotNull
@Column
public Time[] getOpeningHour() {
    return openingHour;
}

public void setOpeningHour(Time[] openingHour) {
    this.openingHour = openingHour;
}

@NotNull
@Column
public String getPhone() {
    return phone;
}

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

@OneToMany(mappedBy = "shopId", fetch = FetchType.EAGER)
@Cascade({CascadeType.SAVE_UPDATE, CascadeType.DELETE})
public Set<ShopOpinion> getOpinions() {
    return opinions;
}
```

```
    public void setOpinions(Set<ShopOpinion> opinions) {
        this.opinions = opinions;
    }

    @OneToMany(mappedBy = "shopId", fetch = FetchType.EAGER)
    @Cascade({CascadeType.SAVE_UPDATE, CascadeType.DELETE})
    public Set<Offer> getOffers() {
        return offers;
    }

    public void setOffers(Set<Offer> offers) {
        this.offers = offers;
    }

}

@Entity
@Table(name = "shopopinions")
class ShopOpinion{

    private Integer shopOpinionId;
    private String comment;
    private Integer rating;
    private ListItem listItem;
    private Timestamp posted;
    private Integer shopId;

    public ShopOpinion(){
    }

    @Id
    @GeneratedValue
    public Integer getShopOpinionId() {
        return shopOpinionId;
    }

    public void setShopOpinionId(Integer shopOpinionId) {
        this.shopOpinionId = shopOpinionId;
    }

    @NotNull
    @Column
    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }
```

```
    @NotNull
    @Column
    public Integer getRating() {
        return rating;
    }

    public void setRating(Integer rating) {
        this.rating = rating;
    }

    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "listitemid")
    public ListItem getListItem() {
        return listItem;
    }

    public void setListItem(ListItem listItem) {
        this.listItem = listItem;
    }

    @NotNull
    @Column
    @Type(type = "timestamp")
    public Timestamp getPosted() {
        return posted;
    }

    public void setPosted(Timestamp posted) {
        this.posted = posted;
    }

    @NotNull
    @Column
    public Integer getShopId() {
        return shopId;
    }

    public void setShopId(Integer shopId) {
        this.shopId = shopId;
    }

}

@Entity
@Table(name = "shoppinglists")
class ShoppingList {

    private Integer shoppingListId;
    private Date date;
    private String description;
    private Set<ListItem> listItems = new HashSet<ListItem>();
```

```
    private String login;

    public ShoppingList(){
    }

    @Id
    @GeneratedValue
    public Integer getShoppingListId() {
        return shoppingListId;
    }

    public void setShoppingListId(Integer shoppingListId) {
        this.shoppingListId = shoppingListId;
    }

    @NotNull
    @Column
    @Type(type = "date")
    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    @Column
    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    @OneToMany(mappedBy = "shoppingListId", fetch = FetchType.EAGER)
    @Cascade({CascadeType.SAVE_UPDATE, CascadeType.DELETE})
    public Set<ListItem> getListItems() {
        return listItems;
    }

    public void setListItems(Set<ListItem> listItems) {
        this.listItems = listItems;
    }

    @NotNull
    @Column
    public String getLogin() {
        return login;
    }
```

```
    public void setLogin(String login) {
        this.login = login;
    }

}

@Entity
@Table(name = "units")
class Unit {

    private Integer unitId;
    private String name;
    private String symbol;

    public Unit(){
    }

    @Id
    @GeneratedValue
    public Integer getUnitId() {
        return unitId;
    }

    public void setUnitId(Integer unitId) {
        this.unitId = unitId;
    }

    @NotNull
    @Column
    public String getName() {
        return name;
    }

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

    @NotNull
    @Column
    public String getSymbol() {
        return symbol;
    }

    public void setSymbol(String symbol) {
        this.symbol = symbol;
    }

}

@Entity
@Table(name = "users")
```

```
class User {

    private String address;
    private Date birth;
    private String email;
    private String login;
    private String name;
    private String password;
    private String surname;
    private Set<ShoppingList> shoppingLists = new HashSet<ShoppingList>();

    public User(){
    }

    @Column
    public String getAddress() {
        return address;
    }

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

    @Column
    @Type(type = "date")
    public Date getBirth() {
        return birth;
    }

    public void setBirth(Date birth) {
        this.birth = birth;
    }

    @NotNull
    @Column
    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Id
    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }
```

```
    @NotNull
    @Column
    public String getName() {
        return name;
    }

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

    @NotNull
    @Column
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @NotNull
    @Column
    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    @OneToMany(mappedBy = "login", fetch = FetchType.LAZY)
    @Cascade({CascadeType.SAVE_UPDATE, CascadeType.DELETE})
    public Set<ShoppingList> getShoppingLists() {
        return shoppingLists;
    }

    public void setShoppingLists(Set<ShoppingList> shoppingLists) {
        this.shoppingLists = shoppingLists;
    }

}
```

2.2. Słowniki danych

2.2.1. ItemProduct

- Double price - cena produktu w momencie tworzenia listy
- Integer quantity - ilość sztuk produktu dodanych listy
- Integer productId - identyfikator produktu
- Integer listItemId - identyfikator pozycji na liście zakupów

2.2.2. ListItem

- Integer listItemId - identyfikator pozycji na liście zakupów
- Integer shoppingListId - identyfikator listy zakupów
- Integer shopId - identyfikator sklepu
- Boolean commented - informacja czy sklep związany z pozycją listy zakupów został skomentowany

2.2.3. Offer

- Integer shopId - identyfikator sklepu
- Double price - cena danego produktu w danym sklepie
- Integer storage - zapas magazynowy danego produktu w danym sklepie
- Integer productId - identyfikator produktu

2.2.4. Product

- Integer productId - identyfikator produktu
- Double amount - ilość produktu w odniesieniu do jednostki
- String category - kategoria produktu
- String description - opis produktu
- String manufacturer - producent produktu
- String name - nazwa produktu
- String subcategory - podkategoria produktu
- Integer unitId - jednostka produktu

2.2.5. Shop

- Integer shopId - identyfikator sklepu
- String address - adres sklepu
- Time[] closingHour - godziny zamknięcia sklepu
- String description - opis sklepu
- String email - email sklepu
- Point location- lokalizacja sklepu (współrzędne geograficzne)
- String name - nazwa sklepu
- Time[] openingHour - godziny otwarcia sklepu
- String phone - telefon sklepu

2.2.6. ShopOpinion

- Integer shopOpinionId - identyfikator opinii sklepu
- String comment - komentarz słowny
- Integer rating - wartość 1 - 7, 7 najwyższa
- Integer listItemId - identyfikator pozycji listy zakupów
- Timestamp posted - czas wystawienia komentarza
- Integer shopId - identyfikator sklepu

2.2.7. ShoppingList

- Integer shoppingListId - identyfikator listy zakupów
- Date date - data utworzenia
- String description - opis listy zakupów
- String login - twórca listy zakupów

2.2.8. Unit

- Integer unitId - identyfikator jednostki
- String name - nazwa jednostki
- String symbol - symbol jednostki

2.2.9. User

- String address - adres użytkownika
- Date birth - data urodzenia użytkownika
- String email - E-mail użytkownika
- String login - login użytkownika
- String name - imię użytkownika
- String password - hasło użytkownika (skrót MD5)
- String surname - nazwisko użytkownika

2.3. Analiza zależności funkcyjnych

2.3.1. Pierwsza postać normalna (1NF)

Wszystkie atrybuty encji w przedstawionym modelu są atomiczne, a więc baza danych spełnia warunek pierwszej postaci normalnej.

2.3.2. Druga postać normalna (2NF)

Wszystkie atrybuty niekluczowe encji zależą od całego klucza głównego danej encji.

2.3.3. Trzecia postać normalna (3NF)

W żadnej z encji opracowanego modelu nie występują relacje tranzytywne, wszystkie atrybuty zależą bezpośrednio od klucza głównego. Wyjątkiem jest enja Unit, symbol zależy od nazwy.

2.4. Projektowanie operacji na danych

2.4.1. pobranie szczegółów odnośnie wybranego sklepu (wizytówka + komentarze)

```
@Autowired
private SessionFactory sessionFactory;

Session session = sessionFactory.openSession();
Shop shop = (Shop)session.get(Shop.class, shopId);
session.close();
```

2.4.2. pobranie listy dostępnych produktów (w danej kategorii i podkategorii)

```
@Autowired
private SessionFactory sessionFactory;

Session session = sessionFactory.openSession();
Criteria productCriteria = session.createCriteria(Product.class);
productCriteria.add(Restrictions.eq("category", category));
productCriteria.add(Restrictions.eq("subcategory", subcategory));
Order order1 = Order.asc("manufacturer");
productCriteria.addOrder(order1);
List<Product> products = (List<Product>)productCriteria.list();
session.close();
```

2.4.3. wyszukanie oferty sklepu (ustalony obszar poszukiwań, ilość i rodzaj produktów)

```
@Autowired
private SessionFactory sessionFactory;

Session session = sessionFactory.openSession();

WKTReader fromText = new WKTReader();
Geometry filter;
String polygon = "POLYGON((" + convertCoordinates(coordinates) + "))"
    ;
try {
    filter = fromText.read(polygon);
    filter.setSRID(4326);
} catch (ParseException e) {
    throw new RuntimeException("Not a POLYGON WKT String");
}
Criteria shopCriteria = session.createCriteria(Shop.class);
shopCriteria.add(SpatialRestrictions.within("location", filter));

ProjectionList projectionList = Projections.projectionList();
```

```
projectionList.add(Projections.property("shopId"));
projectionList.add(Projections.property("offer.product.productId"));
projectionList.add(Projections.property("offer.price"));
shopCriteria.setProjection(projectionList);

Order order1 = Order.asc("shopId");
Order order2 = Order.asc("offer.product.productId");
shopCriteria.addOrder(order1);
shopCriteria.addOrder(order2);

Criteria offerCriteria = shopCriteria.createAlias("offers", "offer");
Disjunction disjunction = Restrictions.disjunction();
for (ItemProduct itemProduct : cartProducts) {
    Criterion idCriterion = Restrictions.eq("offer.product.productId"
        , itemProduct.getProduct().getProductId());
    Criterion storageCriterion = Restrictions.ge("offer.storage",
        itemProduct.getQuantity());
    Conjunction conjunction = Restrictions.conjunction();
    conjunction.add(idCriterion);
    conjunction.add(storageCriterion);
    disjunction.add(conjunction);
}
offerCriteria.add(disjunction);
List result = shopCriteria.list();
session.close();
```

2.4.4. pobranie list zakupów danego użytkownika

```
@Autowired
private SessionFactory sessionFactory;

Session session = sessionFactory.openSession();
User user = (User) session.get(User.class, login);
Set<ShoppingList> shoppingLists = user.getShoppingLists();
session.close();
```

2.4.5. zapisanie komentarza

```
@Autowired
private SessionFactory sessionFactory;

shopOpinion.setPosted(new Timestamp((new Date()).getTime()));
Session session = sessionFactory.openSession();
session.beginTransaction();
Shop shop = (Shop) session.get(Shop.class, shopOpinion.getShopId());
session.getTransaction().commit();
session.beginTransaction();
shop.addShopOpinion(shopOpinion);
ListItem listItem = (ListItem) session.get(ListItem.class,
    shopOpinion.getListItem().getListItemId());
listItem.setCommented(true);
session.getTransaction().commit();
session.close();
```

3. Raport końcowy

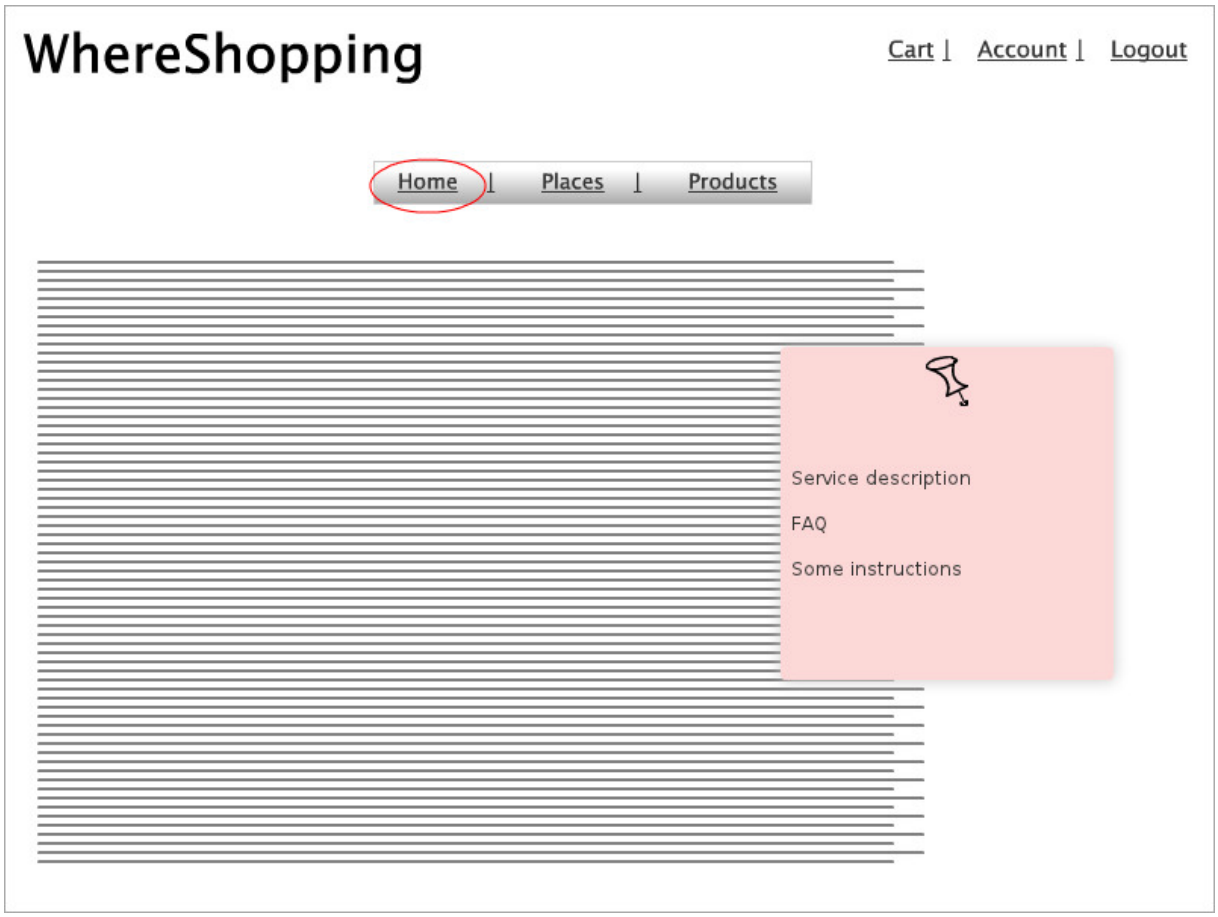
3.1. Implementacja bazy danych

Utworzeniem bazy danych zajmuje się Hibernate. Wymagane zewnętrzne biblioteki dla współpracy z dodatkiem PostGIS to „hibernate-spatial-1.1.jar”, „hibernate-spatial-postgis-1.1.jar”, „postgis-jdbc-1.3.3.jar”, „postgis-stubs-1.3.3.jar”. Więcej na http://www.hibernatespatial.org/hibernate-spatial-postgis. Plik konfiguracyjny:

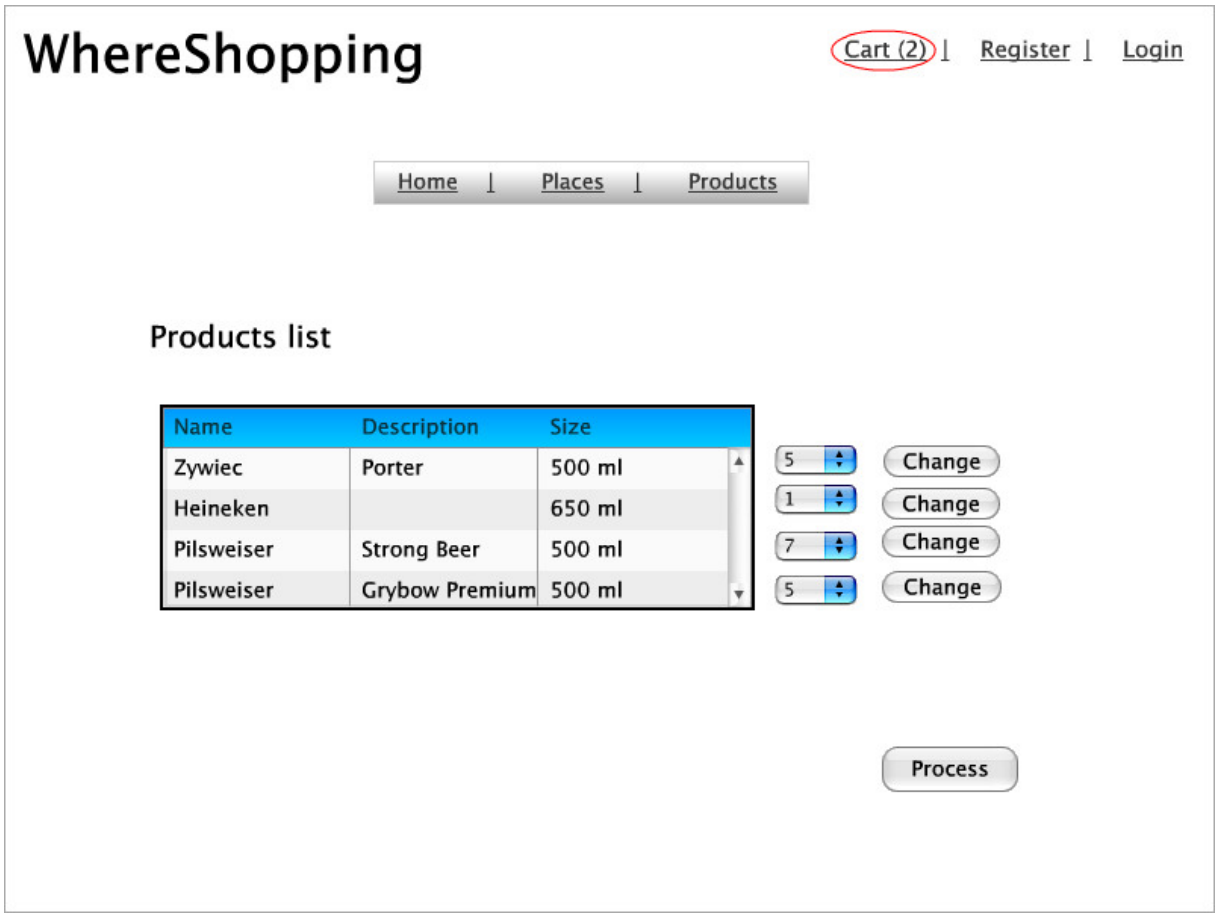
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
                    "-//Hibernate/Hibernate␣Configuration␣DTD␣3.0//EN"
                    "http://hibernate.sourceforge.net/hibernate-
                        configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">org.
            postgresql.Driver</property>
        <property name="hibernate.connection.password">password</
            property>
        <property name="hibernate.connection.url">jdbc:postgresql://
            hostname:port/baseName</property>
        <property name="hibernate.connection.username">username</
            property>
        <property name="hibernate.default_schema">public</property>
        <property name="hibernate.dialect">org.hibernatespatial.
            postgis.PostgisDialect</property>
        <property name="show_sql">false</property>
        <property name="format_sql">true</property>
        <property name="hbm2ddl.auto">update</property>

        <mapping class="com.whereshopping.entity.Unit"/>
        <mapping class="com.whereshopping.entity.Product"/>
        <mapping class="com.whereshopping.entity.Shop"/>
        <mapping class="com.whereshopping.entity.Offer"/>
        <mapping class="com.whereshopping.entity.User"/>
        <mapping class="com.whereshopping.entity.ShoppingList"/>
        <mapping class="com.whereshopping.entity.ListItem"/>
        <mapping class="com.whereshopping.entity.ShopOpinion"/>
        <mapping class="com.whereshopping.entity.ItemProduct"/>
    </session-factory>
</hibernate-configuration>
```

3.2. Zdefiniowanie interfejsów, panelów

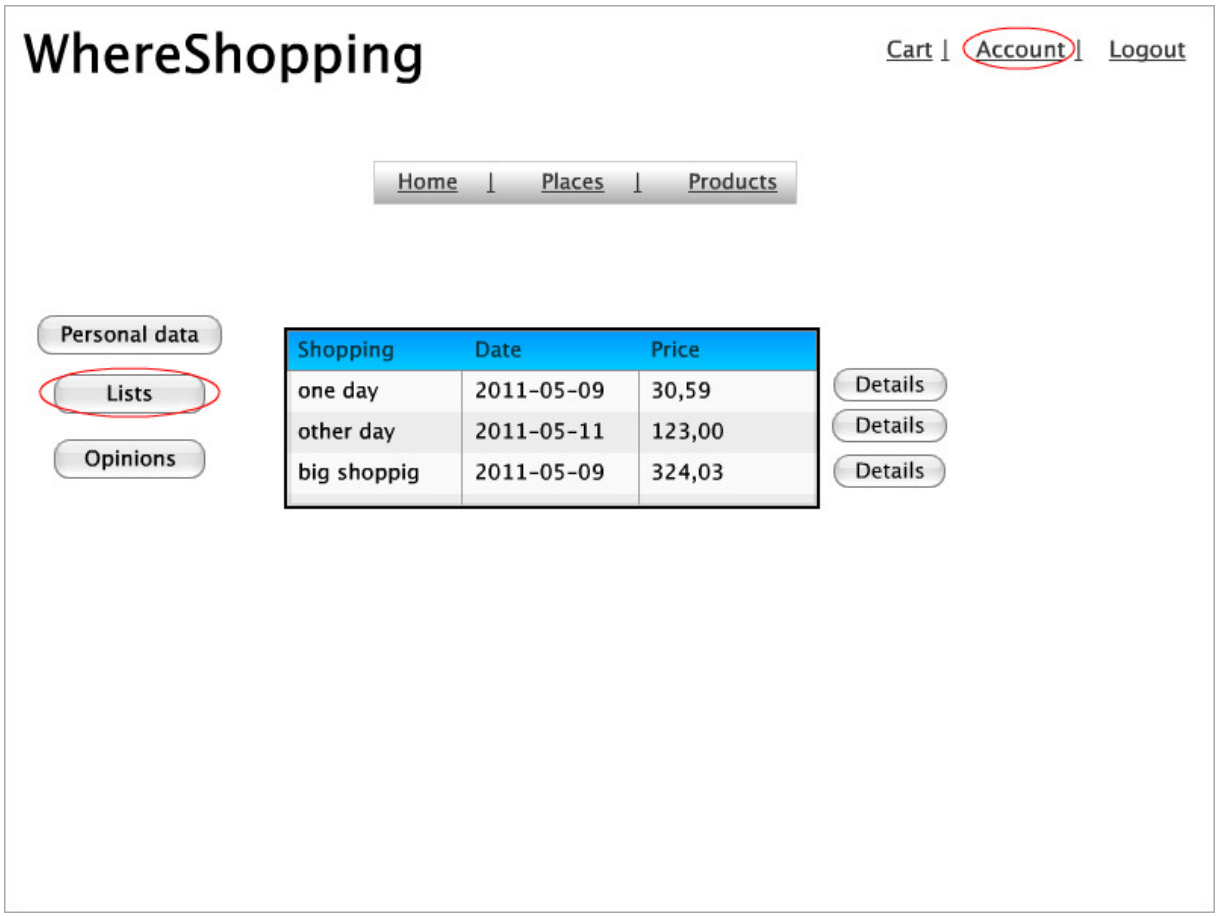
Rysunek 3.1: Interfejs użytkownika 1



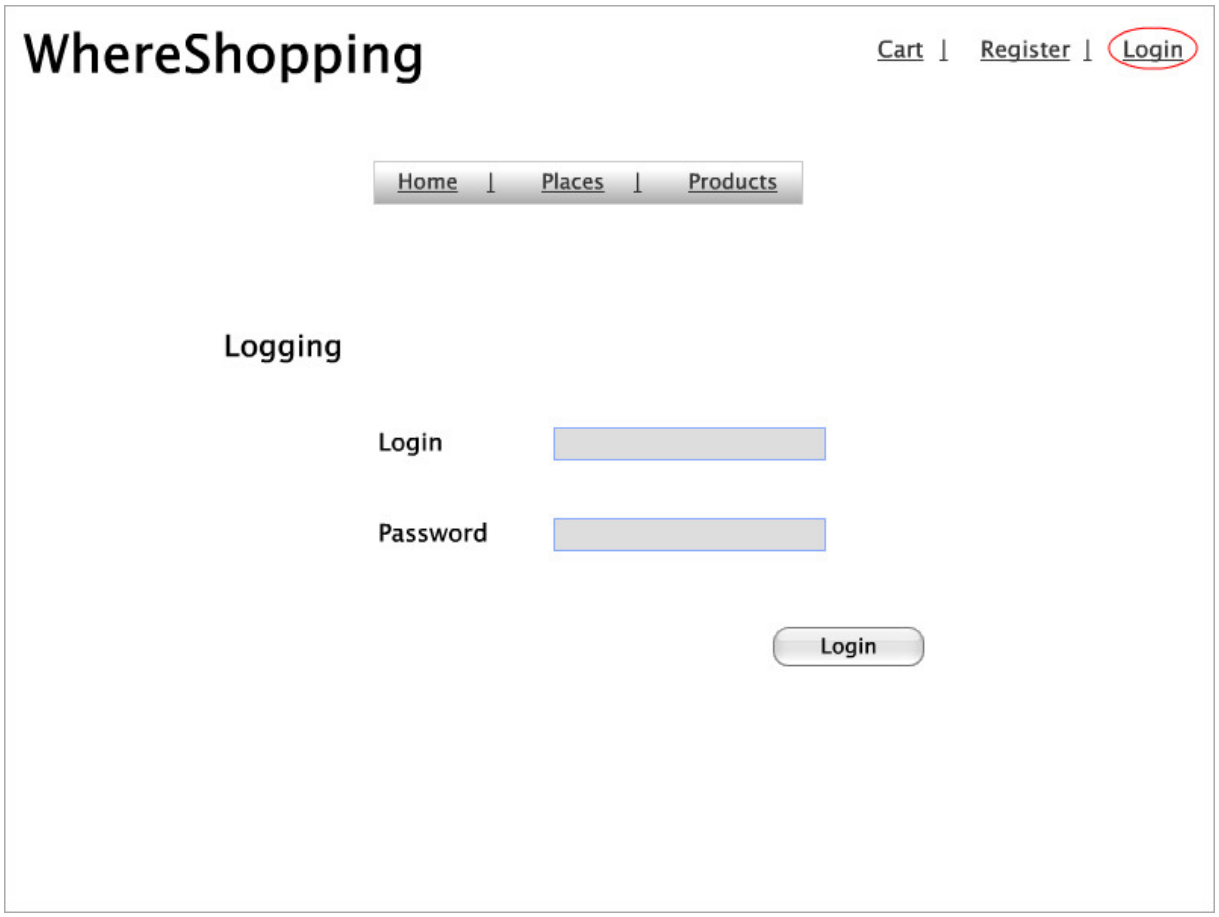
Rysunek 3.2: Interfejs użytkownika 2



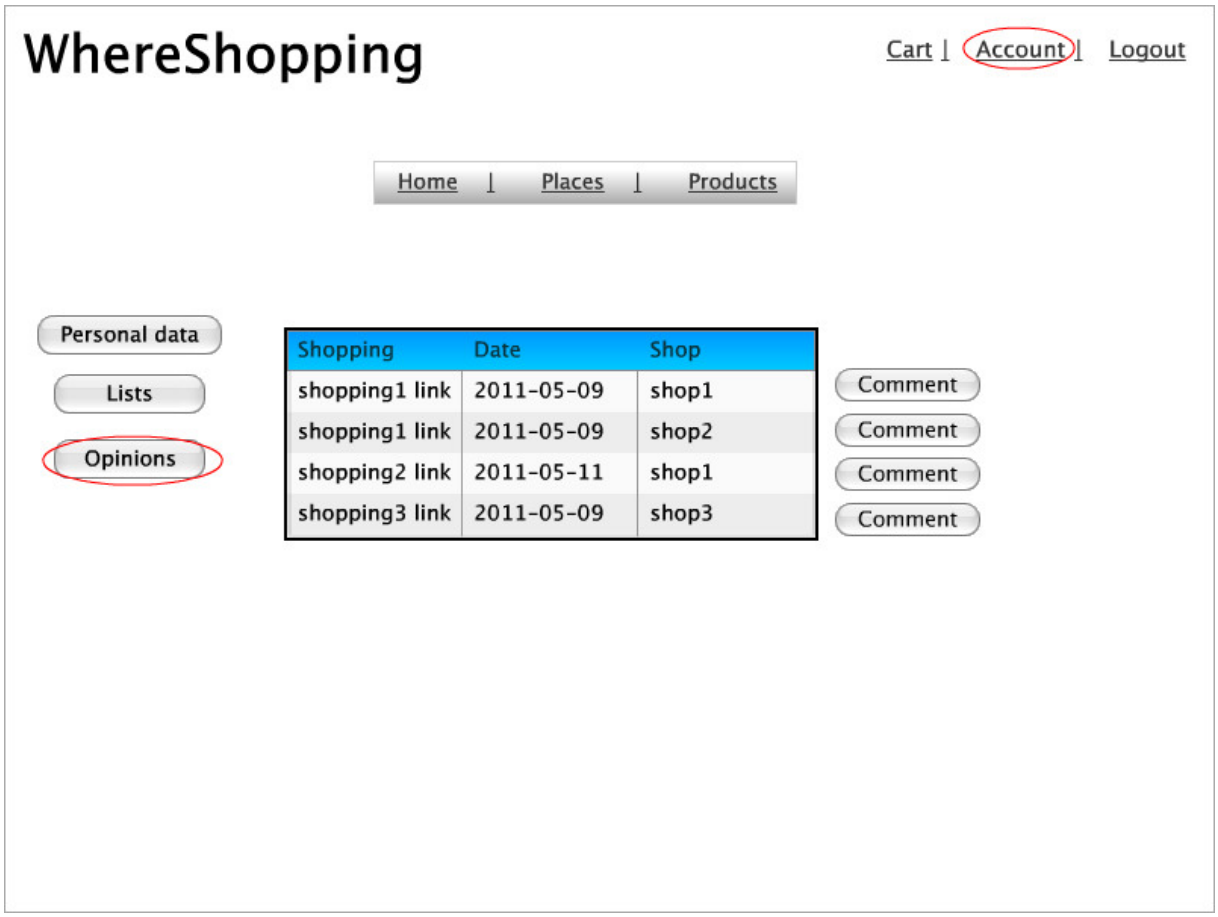
Rysunek 3.3: Interfejs użytkownika 3



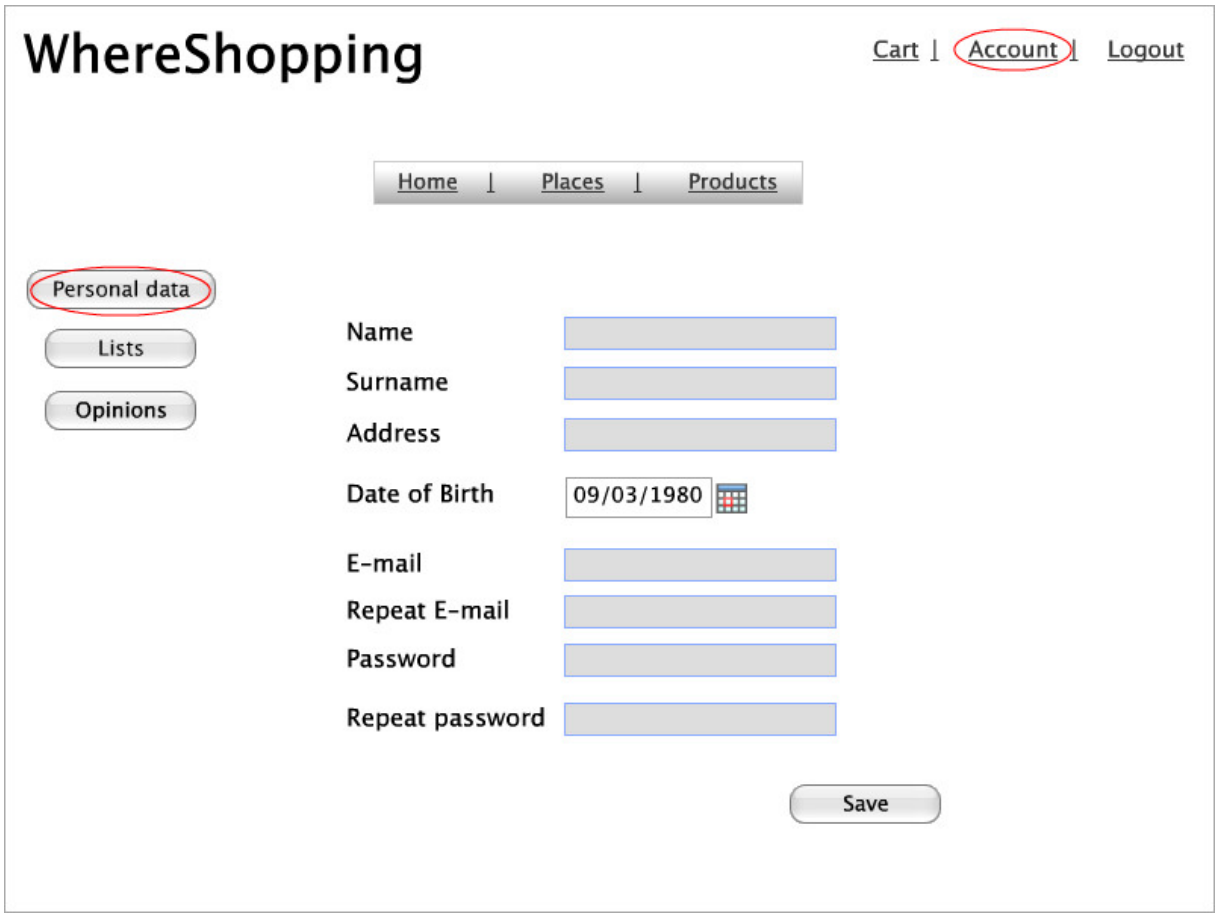
Rysunek 3.4: Interfejs użytkownika 4



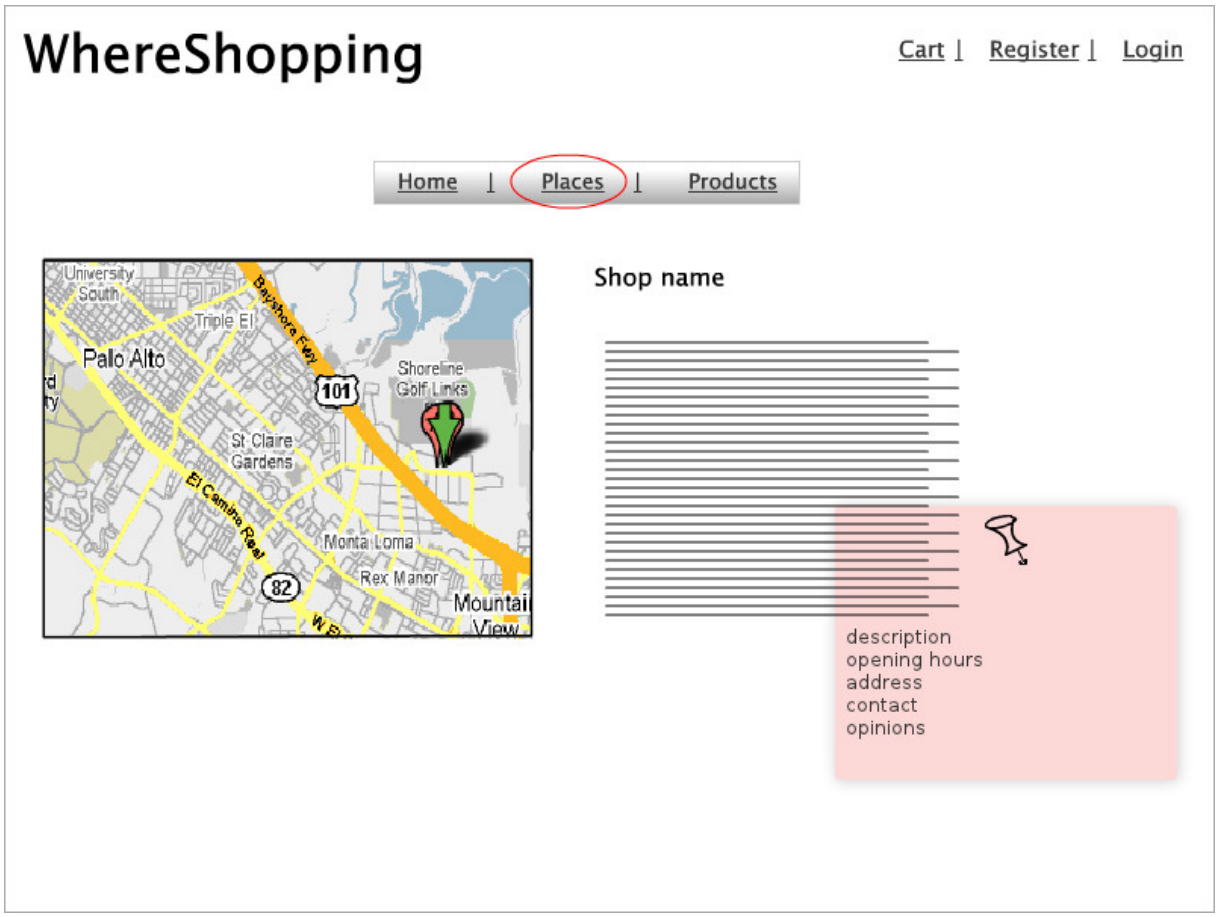
Rysunek 3.5: Interfejs użytkownika 5



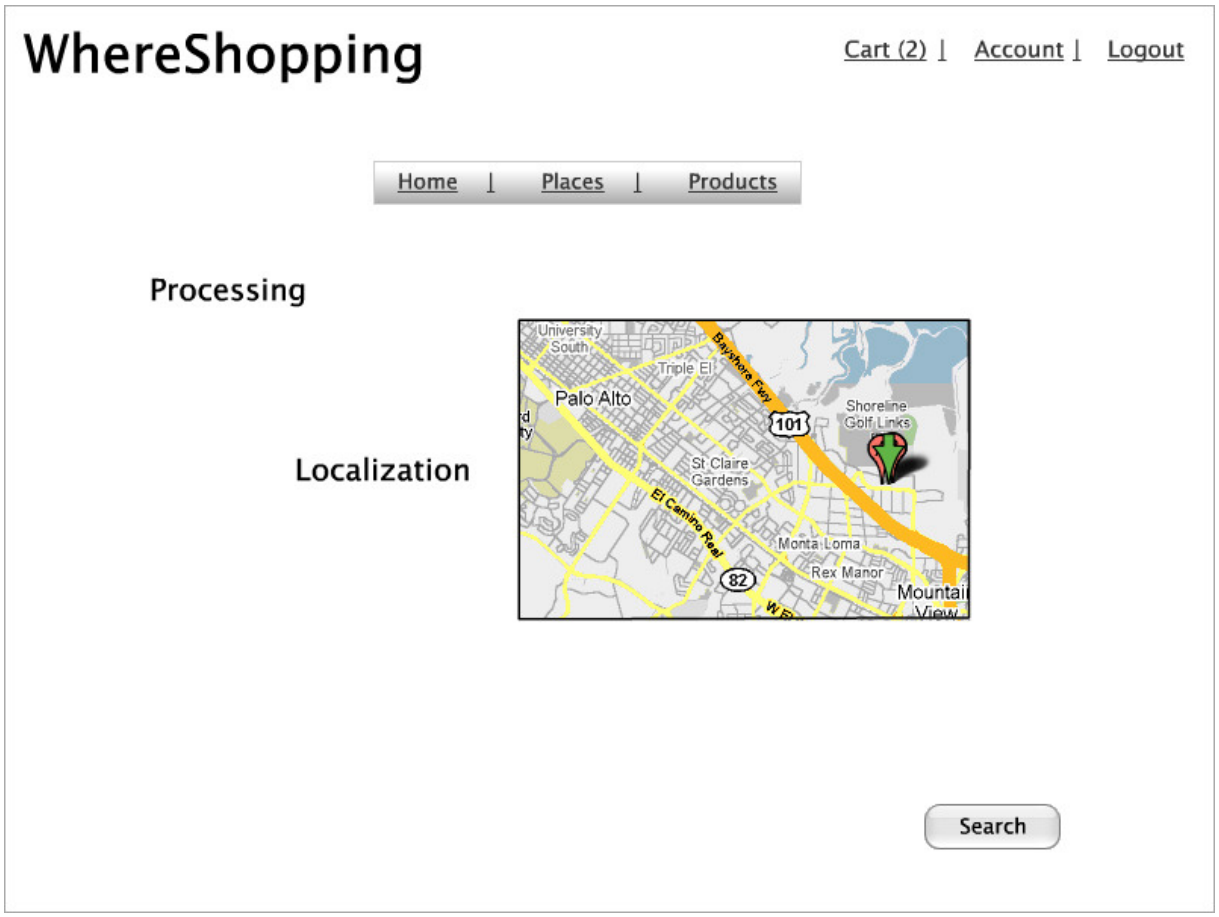
Rysunek 3.6: Interfejs użytkownika 6



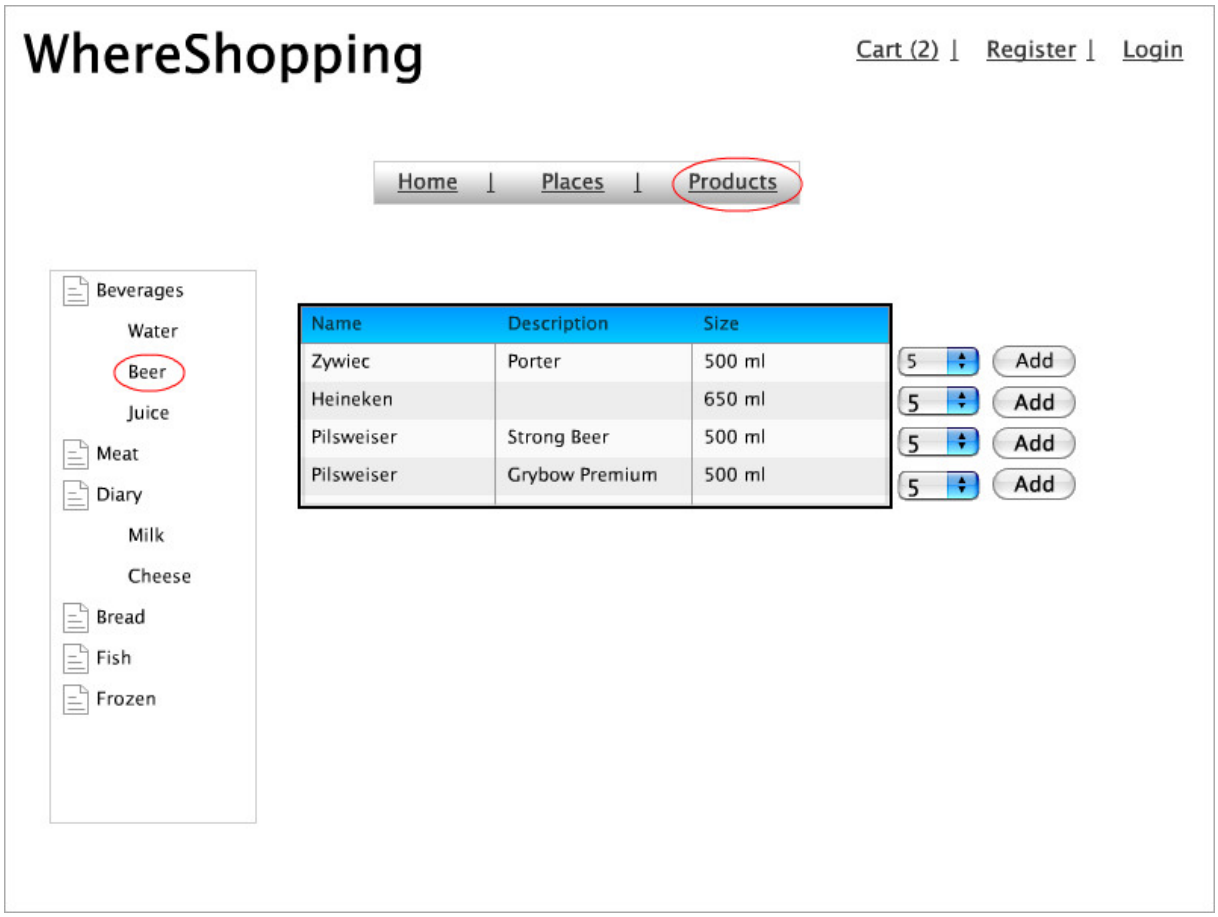
Rysunek 3.7: Interfejs użytkownika 7



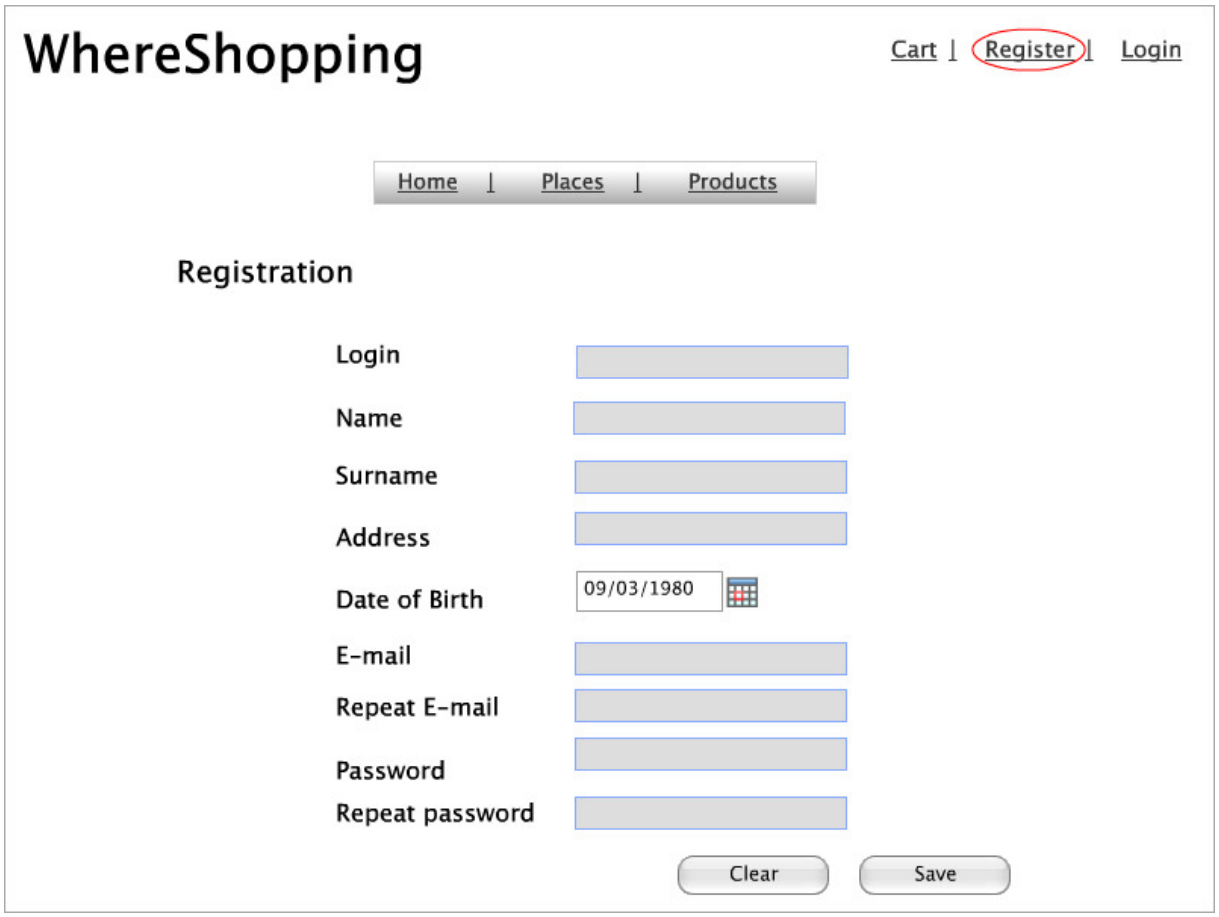
Rysunek 3.8: Interfejs użytkownika 8



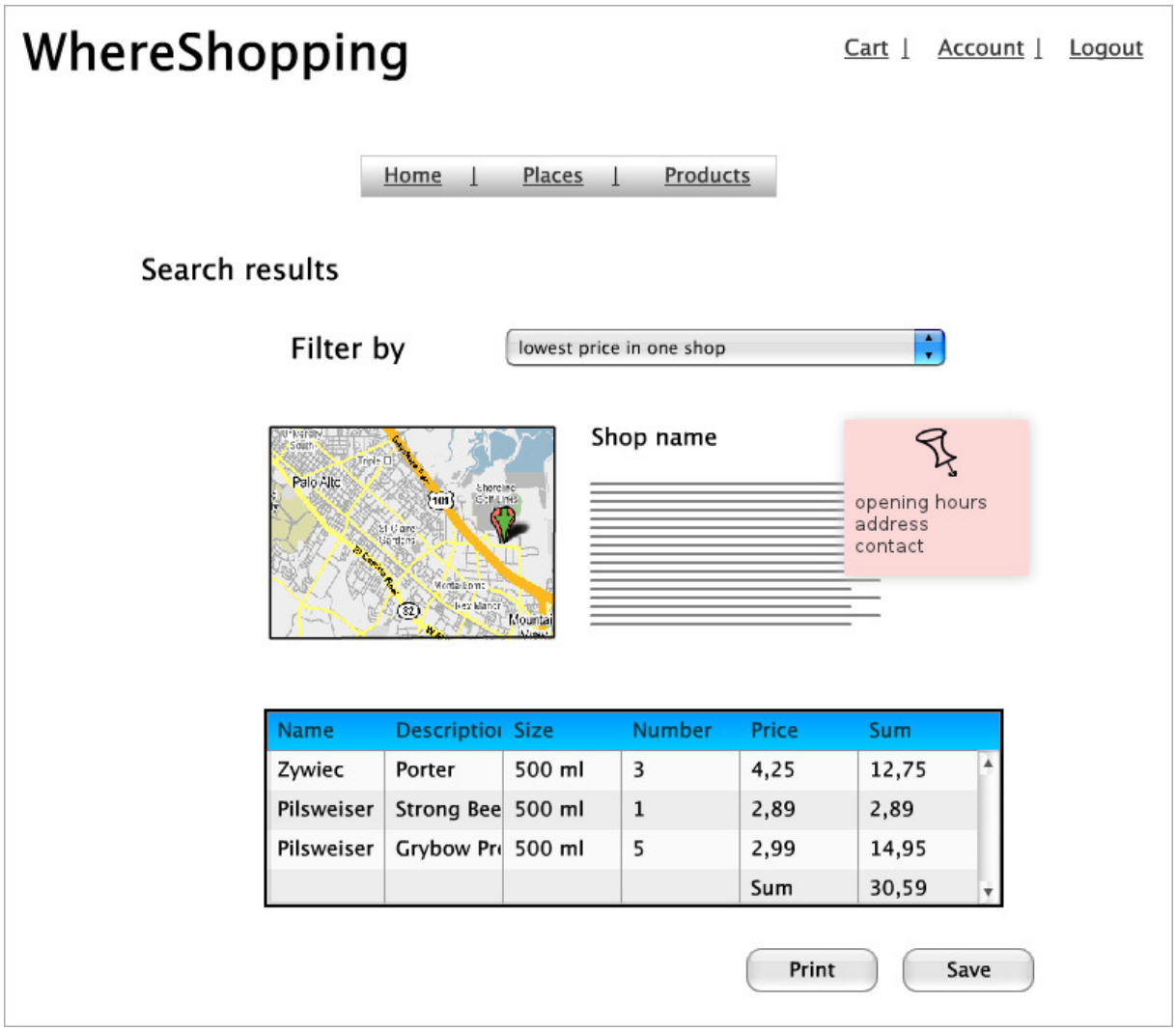
Rysunek 3.9: Interfejs użytkownika 9



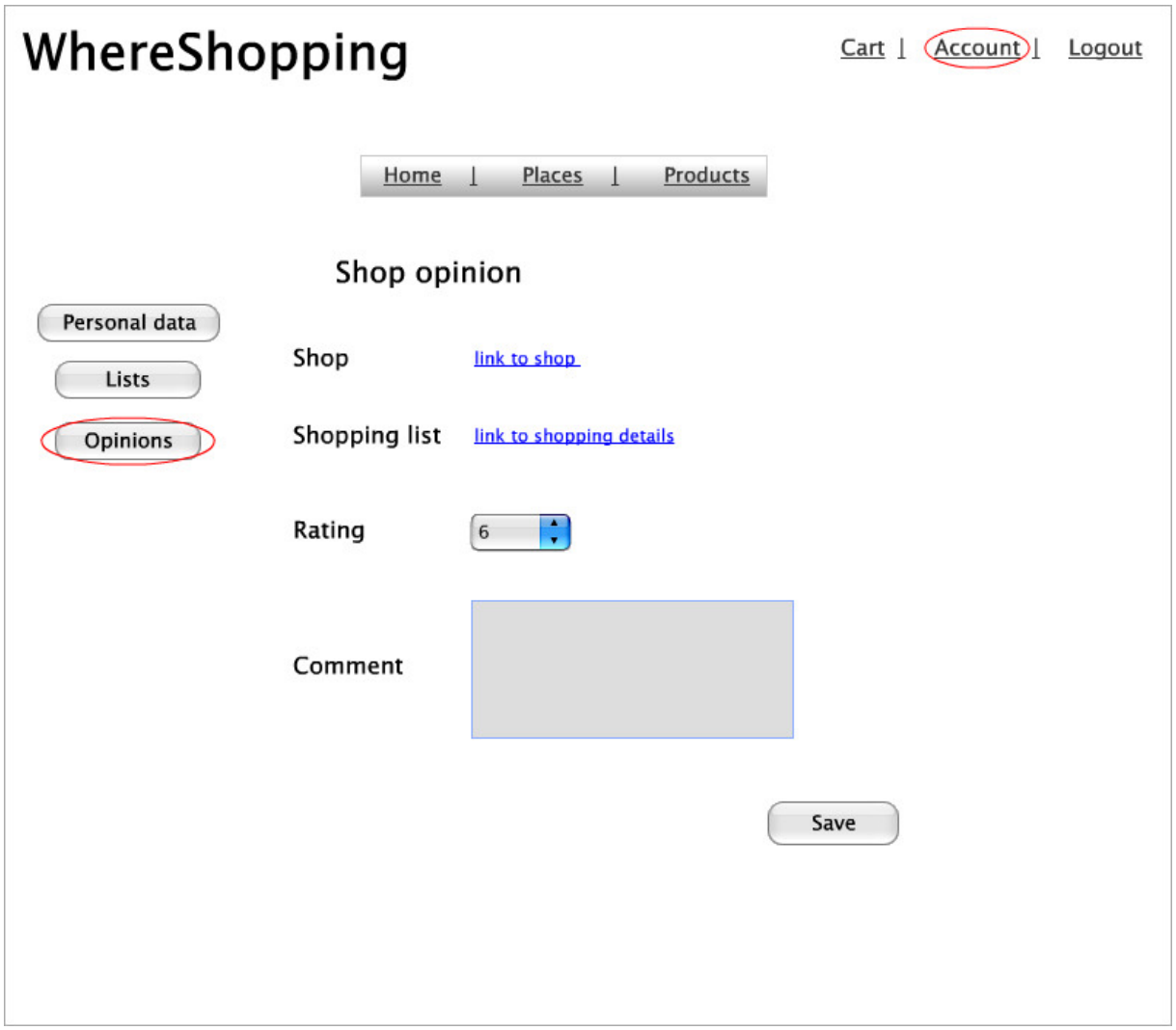
Rysunek 3.10: Interfejs użytkownika 10



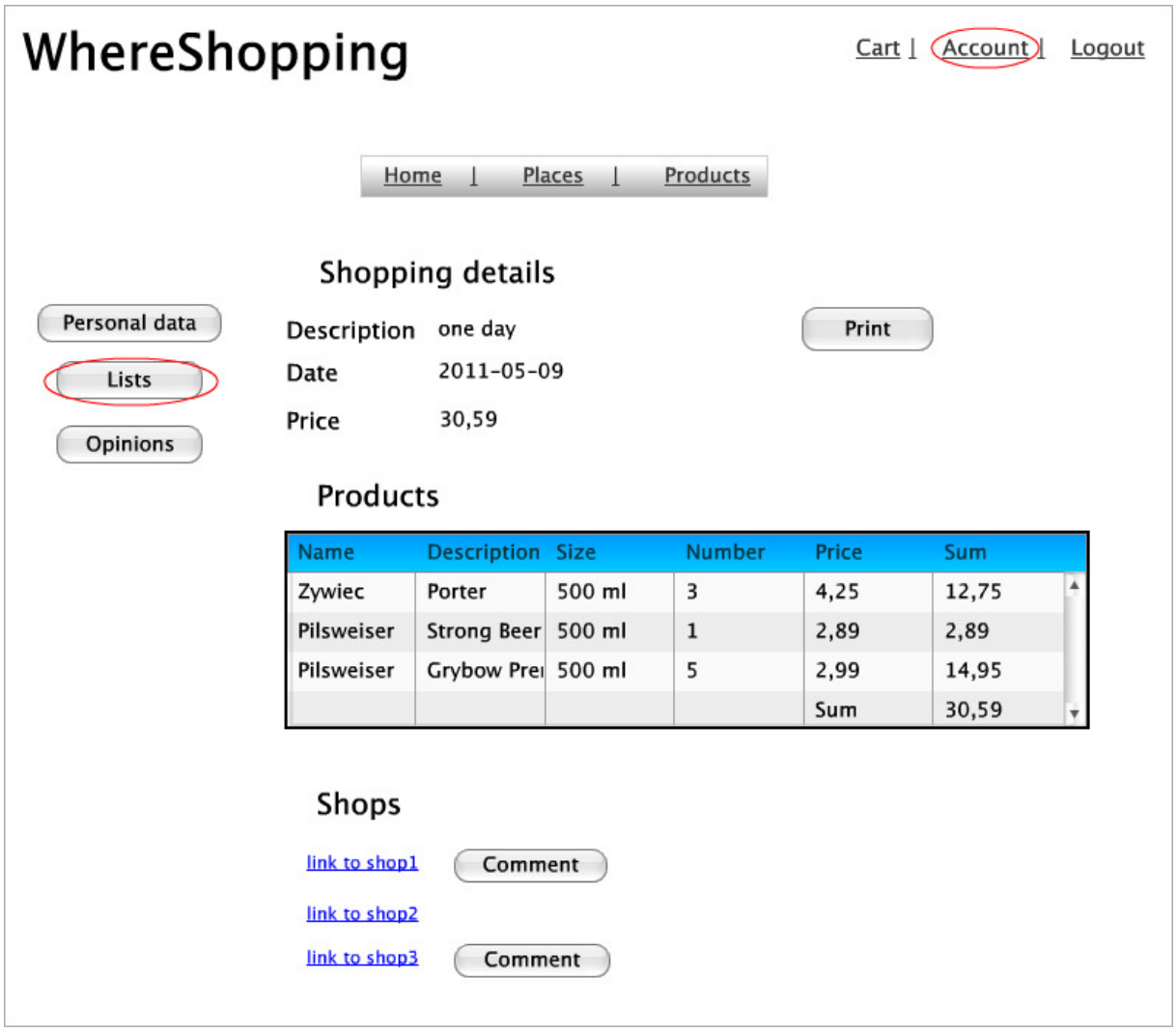
Rysunek 3.11: Interfejs użytkownika 11



Rysunek 3.12: Interfejs użytkownika 12



Rysunek 3.13: Interfejs użytkownika 13



Rysunek 3.14: Interfejs użytkownika 14

3.3. Instalacja

Do działania serwisu potrzebny jest serwer aplikacyjny Javy (np. GlassFish). Na serwerze należy umieścić aplikację „WhereShopping.war”. Dla potrzeb bazy danych należy zainstalować serwer PostgreSQL. Konfiguracja połączenia aplikacji z serwerem znajduje się w pliku „jdbc.properties”:

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://hostname:port/baseName
jdbc.username=username
jdbc.password=password

dbcp.maxActive=100
dbcp.maxIdle=30
dbcp.maxWait=20000

hibernate.generate_statistics=true
hibernate.show_sql=false
hibernate.format_sql=true
hibernate.default_schema=public
hibernate.dialect=org.hibernatespatial.postgis.PostgisDialect
```

```
hbm2ddl.auto=update
```

Zestaw zewnętrznych bibliotek związanych z obsługą Hibernate, Spring, Spring-Security, PostGIS znajduje się w folderze „WEB-INF/lib”.

Bibliografia

[1] A. Hemrajani. *Java : tworzenie aplikacji sieciowych za pomocą Springa, Hibernate i Eclipse*. Helion, Gliwice, 2007.

[2] JBoss Community. *Hibernate Reference Documentation 3.6.3.Final*, 2011.

[3] Spring Source Community. *Spring Framework 3.0 Reference Documentation*, 2010.

[4] Spring Source Community. *Spring Security 3.0.5 Reference Documentation*, 2010.