

Dokumentacja projektu e-commerce

1. WSTĘP	3
1.1 CEL PROJEKTU:.....	3
2. STRUKTURA APLIKACJI	3
2.1 WZORZEĆ MVC	3
2.2 STRUKTURA KATALOGÓW.....	4
3. OPIS PLIKÓW	5
3.1 ODPOWIEDZIALNOŚĆ PLIKÓW I KATALOGÓW:	5
4. FUNKCJONALNOŚCI.....	8
4.1 REJESTRACJA	8
4.1.1 FRAGMENT KODU ODPOWIEDZIALNY ZA REJESTRACJE	8
4.2 LOGOWANIE	9
4.2.1 FRAGMENT KODU ODPOWIEDZIALNY ZA LOGOWANIE.....	9
4.3 PRZEGŁĄDANIE PRODUKTÓW	10
4.3.1 FRAGMENT KODU ODPOWIEDZIALNY ZA PRZEGŁĄDANIE PRODUKTÓW	10
4.3.2 FRAGMENT KODU ODPOWIEDZIALNY ZA SZCZEGÓŁY PRODUKTÓW.....	11
4.4 DODAWANIE DO KOSZYKA	12
4.4.1 FRAGMENT ODPOWIEDZIALNY ZA DODAWANIE PRODUKTU DO KOSZYKA.....	12
4.5 SKŁADANIE ZAMÓWIENIA.....	13
4.5.1 FRAGMENT KODU ODPOWIEDZIALNEGO ZA SKŁADANIE ZAMÓWIENIA	13
4.6 OBSŁUGA PŁATNOŚCI	14
4.6.1 FRAGMENT KODU ODPOWIEDZIALNY ZA OBSŁUGĘ PŁATNOŚCI.....	14
4.7 ZARZĄDZANIE KATEGORIAMI PRODUKTÓW	15
4.7.1 FRAGMENT KODU ODPOWIEDZIALNY ZA DODAWANIE KATEGORII.....	15
4.7.2 FRAGMENT KODU ODPOWIEDZIALNY ZA USUWANIE KATEGORII.....	16
4.8 ZARZĄDZANIE ZAMÓWIENIAMI UŻYTKOWNIKÓW.	17
4.8.1 FRAGMENT KODU ODPOWIEDZIALNY ZA WYSZWIETLANIE SZCZEGÓŁÓW ZAMÓWIENIA	17
4.8.2 FRAGMENT KODU ODPOWIEDZIALNY ZA EDYTOWANIE ZAMÓWIENIA.....	18
4.8.3 FRAGMENT KODU ODPOWIEDZIALNY ZA USUWANIE ZAMÓWIENIA.....	19
4.9 ZARZĄDZANIE PRODUKTAMI.....	20
4.9.1 FRAGMENT KODU ODPOWIEDZIALNY ZA DODAWANIE PRODUKTU	20
4.9.2 FRAGMENT KODU ODPOWIEDZIALNY ZA EDYTOWANIE PRODUKTU	21
4.9.3 FRAGMENT KODU ODPOWIEDZIALNY ZA WYSZWIETLANIE SZCZEGÓŁÓW PRODUKTU	22
4.9.4 FRAGMENT KODU ODPOWIEDZIALNY ZA USUWANIE PRODUKTU	22
4.10 ZARZĄDZANIE UŻYTKOWNIKAMI	23
4.10.1 FRAGMENT KODU ODPOWIEDZIALNY ZA WYSZWIETLANIE SZCZEGÓŁÓW UŻYTKOWNIKA.	23
5. JĘZYKI PROGRAMOWANIA, FRAMEWORKI, USŁUGI ORAZ BIBLIOTEKI	24
5.1 JĘZYKI PROGRAMOWANIA	24
5.2 FRAMEWORKI I BIBLIOTEKI	25
6. BAZA DANYCH	26
6.1 TECHNOLOGIA	26
6.2 SCHEMAT BAZY DANYCH	26
6.3 RELACJE MIĘDZYM TABELAMI.....	26
6.4 DIAGRAM ERD	27
7. BEZPIECZEŃSTWO	28
7.1 HASZOWANIE HASEŁ	28
7.1.1 FRAGMENT KODU Z UŻYCIMI PASSWORD_HASH().....	28
7.1.2 FRAGMENT Z UŻYCIMI PASSWORD_VERIFY().	28

7.2 OCHRONA PRZED SQL INJECTION	28
7.3 WALIDACJA DANYCH WEJŚCIOWYCH	29

1. Wstęp

1.1 Cel projektu:

Projekt e-commerce ma na celu umożliwienie użytkownikom przeglądanie oraz zakupy produktów. A administratorowi umożliwienie zarządzania zamówieniami oraz produktami na stronie poprzez panel administracyjny.

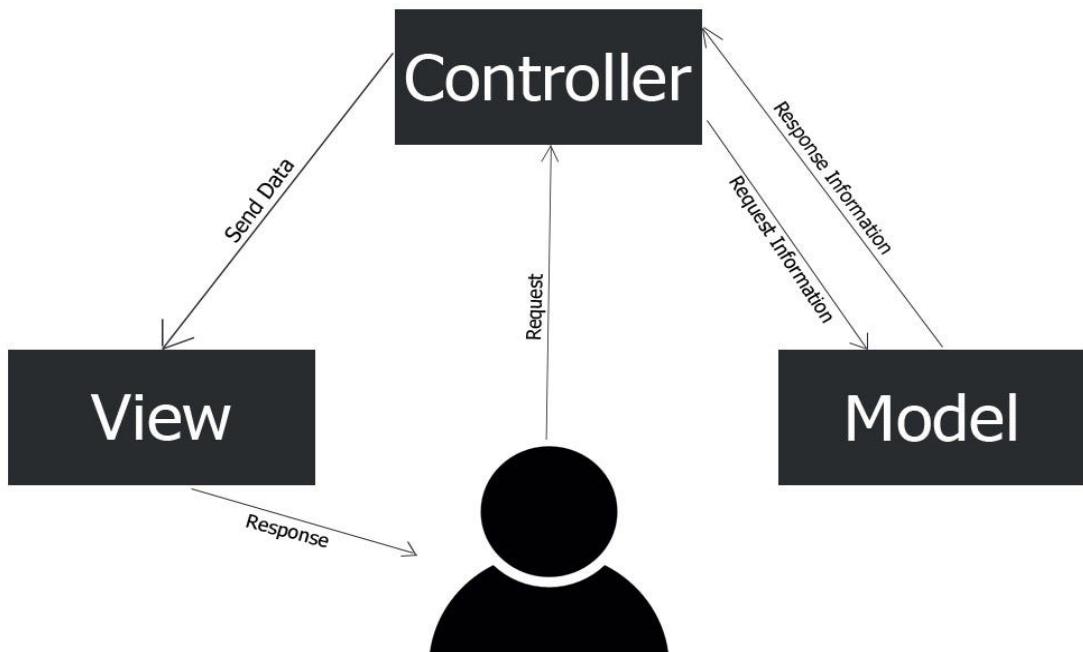
2. Struktura Aplikacji

2.1 Wzorzec MVC

Aplikacja opiera się na wzorcu **Model-View-Controller (MVC)**

- Model – Obsługuje interakcje z bazą danych (AbstractModel.php, StoreModel.php, UserModel.php).
- View – Odpowiada za wyświetlanie odpowiedniego layoutu użytkownikowi (view.php).
- Controller - obsługuje logikę aplikacji i przekazuje dane między modelem a widokiem (AbstractController.php, StoreController.php, UserController.php).

Model-View-Controller



2.2 Struktura katalogów

```
/PROJECT
|---/config/
|---/node_modules/
|---/public/
|   ---/css/
|   ---/images/
|       ---/products/
|---/resources/
|   ---/css/
|   ---/templates/
|       ---/dashboard/
|           ---/category/
|           ---/orders/
|           ---/products/
|           ---/start/
|           ---/users/
|       ---/store
|   ---/components
|---/src/
|   ---/Controllers/
|   ---/Models/
|   ---/Services/
|   ---/Views/
|---/vendor
```

3. Opis Plików

3.1 Odpowiedzialność plików i katalogów:

- `/config/config.php` - plik zawierając dane do logowania z bazą danych
- `/node_modules` – zawiera wszystkie zależności (biblioteki moduły) zainstalowane przez npm na podstawie pliku package.json
- `/public/css/styles.css` - to wygenerowany plik TailwindCSS zawierający zmienne, reset stylów oraz klasy utility do stylizacji elementów w projekcie.
- `/resources/css/styles.css` - o źródłowy plik TailwindCSS, który importuje podstawowe style, komponenty i klasy narzędziowe.
- `/resources/templates/store/components/category.php` - zawiera szablon komponentu kategorii.
- `/resources/templates/store/components/footer.php` - zawiera szablon komponentu stopki.
- `/resources/templates/store/components/header.php` - zawiera szablon komponentu nagłówka.
- `/resources/templates/store/components/navigation.php` - zawiera szablon komponentu nawigacji.
- `/resources/templates/store/layout.php` - definiuje główny szablon strony
- `/resources/templates/store/order.php` - definiuje szablon dla zamówienia
- `/resources/templates/store/product_details.php` – definiuje szablon dla szczegółów produktu.
- `/resources/templates/store/products.php` – definiuje szablon dla listy produktów.
- `/resources/templates/store/shopping_cart.php` – definiuje szablon dla koszyka.
- `/resources/templates/store/sign_in.php` – definiuje szablon logowania.
- `/resources/templates/store/sign_up.php` – definiuje szablon rejestracji.
- `/resources/templates/dashboard/category` – W tym katalogu znajdują się formularze do operacji (create, delete) oraz szablony do wyświetlania wszystkich kategorii produktów w dashboardzie.

- **/resources/templates/dashboard/orders** – W tym katalogu znajdują się formularze do operacji (edit, delete) oraz szablony do wyświetlania wszystkich zamówień i wyświetlania szczegółów danego zamówienia w dashboardzie.
- **/resources/templates/dashboard/products** – W tym katalogu znajdują się formularze do operacji (create, edit) oraz szablony do wyświetlania wszystkich produktów oraz ich szczegółów w dashboardzie.
- **/resources/templates/dashboard/start** – W tym katalogu znajdują się strona startowa dashboardu.
- **/resources/templates/dashboard/users** – W tym katalogu znajdują się szablony do wyświetlania wszystkich użytkowników oraz szczegółów ich zamówień w dashboardzie.
- **/resources/templates/dashboard/layout.php** – Jest to plik, który zawiera główny szablon dashboardu.
- **/src/Controllers/AbstractBaseController.php** - zawiera klasę bazową dla kontrolerów abstract.
- **/src/Controllers/AbstractController.php** - zawiera klasę bazową dla kontrolerów odpowiedzialnych za stronę sklepu (StoreController, UserController).
- **/src/Controllers/AbstractDashboardController.php** - zawiera klasę bazową dla kontrolerów odpowiedzialnych za panel administracyjny (CategoryController, DashboardController, OrderController, ProductController, UserDashboardController).
- **/src/Controllers/StoreController.php** - zawiera klasę kontrolera odpowiedzialnego za obsługę sklepu.
- **/src/Controllers/UserController.php** - zawiera klasę kontrolera odpowiedzialnego za obsługę użytkownika.
- **/src/Controllers/CategoryController.php** - zawiera klasę kontrolera odpowiedzialnego za obsługę operacji dla kategorii w dashboardzie.
- **/src/Controllers/DashboardController.php** - zawiera klasę kontrolera odpowiedzialnego za obsługę strony głównej w dashboardzie.
- **/src/Controllers/OrderController.php** - zawiera klasę kontrolera odpowiedzialnego za obsługę operacji dla zamówień w dashboardzie.
- **/src/Controllers/ProductController.php** - zawiera klasę kontrolera odpowiedzialnego za obsługę operacji dla produktów w dashboardzie.

- `/src/Controllers/UserDashboardController.php` - zawiera klasę kontrolera odpowiedzialnego za obsługę operacji dla użytkowników w dashboardzie.
- `/src/Models/AbstractModel.php` - zawiera klasę bazową dla modeli, definiuje połączenie z bazą danych.
- `/src/Models/DashboardModel.php` - zawiera model odpowiedzialny za zarządzanie danymi na dashboardie w bazie danych.
- `/src/Models/StoreModel.php` - zawiera model odpowiedzialny za zarządzanie danymi sklepu w bazie danych.
- `/src/Models/UserModel.php` - zawiera model odpowiedzialny za operacje użytkowników.
- `/src/Services/StripeService.php` - odpowiada za obsługę płatności
- `/src/Views/view.php` - odpowiada za renderowanie widoków sklepu.
- `/src/Views/view.php` - odpowiada za renderowanie widoków dla dashboardu.
- `/src/Request.php` - odpowiada za obsługę żądań HTTP.
- `/src/Validator.php` - zawiera klasę do walidacji danych wejściowych.
- `/vendor/` - zawiera zależności PHP zarządzane przez Composer.
- `composer.json` - zawiera konfigurację menedżera pakietów Composer
- `composer.lock` - zapisuje dokładne wersje zainstalowanych pakietów określonych w composer.json.
- `index.php` - główny punkt inicjalizacji aplikacji.
- `package-lock.json` - zapisuje dokładne wersje zależności zainstalowanych przez npm.
- `postcss.config.js` - zawiera konfigurację dla PostCSS, który jest narzędziem do przetwarzania CSS.
- `tailwind.config.js` - zawiera konfigurację TailwindCSS.

4. Funkcjonalności

4.1 Rejestracja

Użytkownicy mają możliwość utworzenia nowego konta poprzez proces rejestracji. Poprzez podania informacji takich jak email, password, powtórzenie hasła, nazwa użytkownika.

4.1.1 Fragment kodu odpowiedzialny za rejestracje

```
38     public function sign_upAction(): void {
39         if(!empty($this->request->session(param: 'user'))) header(header: "Location:/?page=start");
40         if($this->request->hasPost()) {
41             $password = $this->request->post(param: 'password');
42             $data = [
43                 'username' => $this->request->post(param: 'username'),
44                 'email' => $this->request->post(param: 'email'),
45                 'password' => $password,
46                 'confirm_password' => $this->request->post(param: 'confirm_password'),
47                 'numberOfUsers' => $this->userModel->getByEmail(email: $this->request->post(param: 'email'))
48             ];
49
50             if($this->checkValidation(data: $data)) {
51                 $data['password'] = password_hash(password: $password, algo: PASSWORD_DEFAULT);
52                 $this->userModel->create(data: $data);
53                 $this->view->renderView(params: ['page' => 'sign_in'],
54                                         message: ['messageTop' => "Udało się założyć konto !!!"]);
55             } else {
56                 $this->view->renderView(
57                     params: ['page' => 'sign_up'],
58                     message: [
59                         'username' => $this->validator->username(username: $data['username']),
60                         'email' => $this->validator->email(email: $data['email']),
61                         'password' => $this->validator->checkPassword(password: $password),
62                         'confirm_password' => $this->validator->confirmPassword(password: $password,
63                                         confirm_password: $data['confirm_password']),
64                         'userExist' => $this->validator->userExist(numberOfUsers: $data['numberOfUsers'])
65                     ]
66                 );
67             }
68         }
69
70         $this->view->renderView(params: ['page' => 'sign_up']);
71     }
72
73     public function create(array $data):void {
74         try {
75             $username = $this->conn->quote(string: $data['username']);
76             $email = $this->conn->quote(string: $data['email']);
77             $password = $data['password'];
78
79             $sql = "INSERT INTO users(name, email, password)
80                   VALUES($username, $email, '$password')";
81
82             $this->conn->exec(statement: $sql);
83         } catch(Throwable $e) {
84             throw new Exception(message: "Nie udało się utworzyć użytkownika");
85         }
86     }
87 }
```

4.2 Logowanie

Użytkownicy mają możliwości zalogowania się do serwisu podając email oraz hasło. Po zalogowaniu użytkownik może realizować zamówienie.

4.2.1 Fragment kodu odpowiedzialny za logowanie

```
18     public function sign_inAction():void {
19         if(!empty($this->request->session(param: 'user'))) header(header: "Location:/?page=start");
20         $error = [];
21         if($this->request->hasPost()) {
22             $email = $this->request->post(param: 'email');
23             $password = $this->request->post(param: 'password');
24             if($this->userModel->getByEmail(email: $email) !== 0) {
25                 $user = $this->userModel->getUser(email: $email);
26                 if(password_verify(password: $password, hash: $user['password'])) {
27                     $_SESSION['user'] = $user;
28                     $this->view->renderView(params: ['page' => 'start'],
29                     | message: ["messageTop" => "Udało się zalogować"]);
30                 } else {
31                     $error['loginError'] = "Nie poprawne hasło";
32                 }
33             } else $error['loginError'] = "Nie poprawny E-mail";
34         }
35         $this->view->renderView(params: ['page' => 'sign_in'], message: $error);
36     }
37 }
```

4.3 Przeglądanie Produktów

Użytkownik ma możliwość przeglądania listy produktów, może ją wyświetlać po kategorii oraz wyświetlać szczegóły produktów.

4.3.1 Fragment kodu odpowiedzialny za przeglądanie produktów

```
16  ↵ public function productsAction():void {
17  ↵     if($this->request->isPost()) {
18  ↵         $this->AddProductToCart();
19  ↵     }
20
21     $category = $this->request->get(param: 'category');
22  ↵     $this->view->renderView(params: [
23  ↵         'page' => 'products',
24  ↵         'content' => $this->model->GetProducts(category: $category)
25  ↵     ]);|
26 }
```

```
12  ↵     public function GetProducts(?string $category): array {
13  ↵         try {
14  ↵             $sql = "";
15  ↵             if(in_array(needle: $category, haystack: ['okna_aluminiowe', 'okna_pcv', 'okna_drewniane'])) {
16  ↵                 $sql = "SELECT products.*, categories.name AS categoryName
17  ↵                         FROM products
18  ↵                         LEFT JOIN categories ON products.category_id = categories.id
19  ↵                         WHERE categories.name = '$category'";
20  ↵             } else {
21  ↵                 $sql = "SELECT * FROM products";
22  ↵             }
23
24  ↵             $result = $this->conn->query($sql);
25  ↵             return $result->fetchAll(PDO::FETCH_ASSOC);
26  ↵         }catch(Throwable $e) {
27  ↵             throw new Exception(message: "Nie udało się pobrać listy produktów.", code: 400, previous: $e);
28  ↵         }
29 }
```

4.3.2 Fragment kodu odpowiedzialny za szczegóły produktów.

```
28     public function product_detailsAction(): void {
29         $id = (int) $this->request->get(param: 'id');
30         if(!$id) {
31             $this->startAction();
32         }
33
34         if($this->request->isPost()) {
35             $quantity = (int) $this->request->post(param: "quantity");
36             $this->AddProductToCart(quantity: $quantity);
37         }
38
39         $this->view->renderView(params: [
40             'page' => 'product_details',
41             'content' => $this->model->GetProductDetails(id: $id)
42         ]);
43     }
44
45     public function GetProductDetails(int $id): array {
46         try {
47             $sql = "SELECT products.id, products.name, products.image_url,
48                   products.size, products.description, products.price,products.stock, categories.name
49                   AS categoryName
50                   FROM products
51                   LEFT JOIN categories ON products.category_id = categories.id
52                   WHERE products.id = $id";
53             $result = $this->conn->query($sql);
54
55             $product = $result->fetch(mode: PDO::FETCH_ASSOC);
56         }catch(Throwable $e) {
57             throw new Exception(message: "Nie udało się pobrać produktu", code: 400, previous: $e);
58         }
59
60         if(!$product) {
61             throw new Exception(message: "Nie znaleziono produktu");
62         }
63
64         return $product;
65     }
66 }
```

4.4 Dodawanie do koszyka

Użytkownik po zalogowaniu ma możliwość dodawania produktów do koszyka poprzez wybranie ilości produktu i kliknięciu przycisku „*Dodaj do Koszyka*” z podstrony o szczegółach produktu albo bezpośrednio z podstrony z listą produktów.

4.4.1 Fragment odpowiedzialny za dodawanie produktu do koszyka.

```
102     public function AddProductToCart(int $quantity = 1): void {
103         if(empty($this->request->session(param: 'user'))) header(header: "Location: /?page=products");
104         $productId = $this->request->post(param: "product_id");
105         if($quantity == 0) $quantity = 1;
106         $data = [
107             'userId' => $this->request->session(param: 'user')['id'],
108             'productId' => $productId,
109             'quantity' => $quantity
110         ];
111         $this->userModel->AddProductToCart(cartData: $data);
112     }
113     public function AddProductToCart(array $cartData): void {
114         try {
115             $sql = "INSERT INTO cart (user_id, product_id, quantity)
116             VALUES('$cartData[userId]', '$cartData[productId]', '$cartData[quantity]')";
117             $this->conn->exec(statement: $sql);
118         }catch(Throwable $e) {
119             throw new Exception(message: "Nie udało się dodać produktu do koszyka");
120         }
121     }
122 }
```

4.5 Składanie zamówienia

Użytkownik po zalogowaniu ma możliwość składania zamówienia po przejściu w koszyk i kliknięciu „Zamów”. Następnie użytkownik jest przekierowany na podstronę, gdzie jest proszony o uzupełnienie danych adresowych.

4.5.1 Fragment kodu odpowiedzialnego za składanie zamówienia

```
114     public function orderAction():void{
115         if(empty($this->request->session(param: 'user'))) header(header: "Location: /?page=start");
116         $userId = $this->request->session(param: 'user')['id'];
117
118         $orderProducts = $this->userModel->GetUserCart(userId: $userId);
119         $total_amount = $this->GetTotalAmount(content: $orderProducts);
120
121         if($this->request->isPost()) {
122             $data = [
123                 "city" => $this->request->post(param: "city"),
124                 "street" => $this->request->post(param: "street"),
125                 "postal_code" => $this->request->post(param: "postal_code"),
126                 "building_number" => $this->request->post(param: "building_number"),
127                 'firstname' => $this->request->post(param: "firstname"),
128                 'lastname' => $this->request->post(param: "lastname"),
129                 'userId' => $userId
130             ];
131
132             $addressId = $this->userModel->AddAddress(data: $data);
133
134             $orderId = (int) $this->userModel->CreateOrder(data: [
135                 "total_amount" => $total_amount,
136                 "addressId" => $addressId,
137                 "userId" => $userId
138             ]);
139
140             $this->userModel->AddProductsToOrder(data: $orderProducts, orderId: $orderId);
141             $this->StripeAction(orderId: $orderId, orderProducts: $orderProducts);
142         }
143
144         $this->view->renderView([
145             'page' => 'order',
146             'content' => $orderProducts,
147             'total_amount' => $total_amount
148         ]);
149     }
150
151     public function CreateOrder(array $data): string {
152         try {
153             $sql = "INSERT INTO orders(user_id, total_price, address_id)
154             VALUES('$data[userId]', '$data[total_amount]', '$data[addressId]')";
155             $this->conn->exec(statement: $sql);
156             return $this->conn->lastInsertId();
157         }catch(Throwable $e) {
158             throw new Exception(message: "Nie udało się utworzyć zamówienia");
159         }
160     }
161 }
```

4.6 Obsługa płatności

Użytkownik po złożeniu zamówienia zostaje przekierowany na stronę odpowiedzialną za płatność (Stripe) gdzie jest proszony o wybranie metody płatności oraz dane.

4.6.1 Fragment kodu odpowiedzialny za obsługę płatności

```
151     public function StripeAction(int $orderId, array $orderProducts): void {
152         if(empty($this->request->session(param: 'user'))) header(header: "Location: /?page=start");
153
154         $StripsProductList = [];
155         foreach($orderProducts as $orderProduct) {
156             array_push(array: &$StripsProductList ,
157                         values: ["quantity" => (int) $orderProduct['quantity'],
158                                  "price_data"=>["currency"=>"pln",
159                                  "unit_amount" => (int) $orderProduct['productPrice'] * 100,
160                                  "product_data" => [ "name" => $orderProduct['productName']
161                                         []]);
162         }
163
164         $checkout_session = \Stripe\Checkout\Session::create(params: [
165             "mode" => "payment",
166             "success_url" => "http://localhost/?page=success&orderId=$orderId&session_id={CHECKOUT_SESSION_ID}",
167             "cancel_url" => "http://localhost/?page=fail&orderId=$orderId&session_id={CHECKOUT_SESSION_ID}",
168             "locale" => "pl",
169             "line_items" => [
170                 $StripsProductList
171             ]
172         ]);
173
174         http_response_code(response_code: 303);
175         header(header: "Location: " . $checkout_session->url);
176     }
177 }
```

4.7 Zarządzanie kategoriami produktów

Administrator ma możliwość dodawania oraz usuwania (tylko w momencie, w którym żaden produkt nie ma przypisanej danej kategorii) kategorii produktu

4.7.1 Fragment kodu odpowiedzialny za dodawanie kategorii.

```
15     public function createAction(): void {
16         $this->dashboardView->renderView(params: [
17             "page" => "category/create.php",
18         ]);
19     }
20
21     0 references | 0 overrides
22     public function storeAction():void {
23         if($this->request->isPost()){
24             $this->dashboardModel->createCategory(name: $this->request->post['name']);
25             header(header: "Location: /?module=category");
26         }
27
28     }
29
30     public function createCategory(string $name): void {
31         try {
32             $result = $this->conn->prepare(query: "INSERT INTO categories (name) VALUES(:name)");
33             $result->execute(params: [":name" => $name]);
34         }catch(Throwable $e) {
35             throw new Exception(message: "Nie udało się utworzyć nowej kategorii");
36         }
37     }
38
39 
```

4.7.2 Fragment kodu odpowiedzialny za usuwanie kategorii.

```
28     public function deleteAction(): void {
29         $id = (int) $this->request->get(param: "id");
30         $hasProducts = $this->dashboardModel->hasProductsInCategory(id: $id);
31         $message = "";
32
33         if($hasProducts) {
34             $message = "Istnieją produkty z taką kategorią!!!";
35         }
36
37         if($this->request->isPost()){
38             $id = (int) $this->request->post(param: 'id');
39             $hasProducts = $this->dashboardModel->hasProductsInCategory(id: $id);
40
41             if(!$hasProducts) {
42                 $this->dashboardModel->deleteCategory(id: $id);
43                 header(header: "Location: /?module=category");
44                 exit;
45             }else {
46                 throw new Exception(message: "Kategoria ma produkty");
47             }
48         }
49
50         $this->dashboardView->renderView(params: [
51             "page" => "category/delete.php",
52             "category" => $this->dashboardModel->getSingleElement(table: "categories", id: $id),
53             "message" => $message,
54         ]);
55     }
56 }
57
58     public function deleteCategory(int $id):void {
59         try {
60             $result = $this->conn->prepare(query: "DELETE FROM categories WHERE id = ? ");
61             $result->execute(params: [$id]);
62         }catch(Throwable $e) {
63             throw new Exception(message: "Nie udało się usunąć kategorii");
64         }
65     }
66
67
68     public function hasProductsInCategory(int $id): bool {
69         try{
70             $result = $this->conn->prepare(query: "SELECT COUNT(*) FROM products WHERE category_id = ? ");
71             $result->execute(params: [$id]);
72             $count = (int) $result->fetchColumn();
73             return $count === 0 ? false: true;
74         }catch(Throwable $e){
75             throw new Exception(message: "Nie udało się pobrać danych");
76         }
77     }
78
79 }
```

4.8 Zarządzanie zamówieniami użytkowników.

Administrator ma możliwość zarządzania zamówieniami. Może on wyświetlać szczegóły danego zamówienia, edytować zamówienie oraz ma możliwość usuwania zamówień.

4.8.1 Fragment kodu odpowiedzialny za wyświetlanie szczegółów zamówienia

```
148     public function getOrder(int $id): array {
149         try {
150             $orderQuery = "SELECT o.total_price, o.status, o.created_at, o.payment_status, o.id as orderId , u.id as userId, u.name, u.email, a.*  
151             FROM orders o  
152             JOIN users u ON o.user_id = u.id  
153             JOIN adress a ON o.address_id = a.id  
154             WHERE o.id = :order_id";  
155  
156             $orderDetailsQuery = "SELECT oi.id as orderItemId, oi.quantity, oi.price, p.name, p.image_url, p.size, p.id  
157                 FROM order_items oi  
158                 JOIN products p ON oi.product_id = p.id  
159                 WHERE oi.order_id = :order_id";  
160  
161             $orderResult = $this->conn->prepare(query: $orderQuery);  
162             $orderResult->execute(params: [':order_id' => $id]);  
163  
164             $orderDetailsResult = $this->conn->prepare(query: $orderDetailsQuery);  
165             $orderDetailsResult->execute(params: [':order_id' => $id]);  
166  
167             $orderResult = $orderResult->fetch(mode: PDO::FETCH_ASSOC);  
168             $orderDetailsResult = $orderDetailsResult->fetchAll(mode: PDO::FETCH_ASSOC);  
169  
170             if(!$orderResult) throw new Exception(message: "Nie znaleziono zamówienia");  
171  
172             return [  
173                 'order' => [  
174                     'id' => $orderResult['orderId'],  
175                     'total_price' => $orderResult['total_price'],  
176                     'status' => $orderResult['status'],  
177                     'created_at' => $orderResult['created_at'],  
178                     'payment_status' => $orderResult['payment_status'],  
179                 ],  
180                 'user' => [  
181                     'id' => $orderResult['userId'],  
182                     'name' => $orderResult['name'],  
183                     'email' => $orderResult['email'],  
184                 ],  
185                 'address' => [  
186                     'id' => $orderResult['id'],  
187                     'firstname' => $orderResult['firstname'],  
188                     'lastname' => $orderResult['lastname'],  
189                     'street' => $orderResult['street'],  
190                     'city' => $orderResult['city'],  
191                     'building_number' => $orderResult['building_number'],  
192                     'postal_code' => $orderResult['postal_code'],  
193                     'user_id' => $orderResult['user_id'],  
194                 ],  
195                 'products' => $orderDetailsResult,  
196             ];  
197         }catch(Throwable $e) {  
198             throw new Exception(message: "Nie udało się pobrać zamówienia");  
199         }  
200     }
```

```
14     public function showAction(): void {  
15         $id = (int) $this->request->get(param: 'id');  
16         $this->dashboardView->renderView(params: [  
17             'page' => 'orders/show.php',  
18             'data' => $this->dashboardModel->getOrder(id: $id),  
19         ]);  
20     }  
21 }
```

4.8.2 Fragment kodu odpowiedzialny za edytowanie zamówienia

```

203     public function updateOrder(array $data): void {
204         try {
205             $this->conn->beginTransaction();
206
207             $sql = "UPDATE adress
208                 SET firstname = :firstname,
209                     lastname = :lastname,
210                     street = :street,
211                     city = :city,
212                     building_number = :building_number,
213                     postal_code = :postal_code
214                     WHERE id = :id
215             ";
216             $result = $this->conn->prepare(query: $sql);
217             $result->execute(params: [
218                 ':firstname' => $data['address']['firstname'],
219                 ':lastname' => $data['address']['lastname'],
220                 ':street' => $data['address']['street'],
221                 ':city' => $data['address']['city'],
222                 ':building_number' => $data['address']['building_number'],
223                 ':postal_code' => $data['address']['postal_code'],
224                 ':id' => $data['address']['id'],
225             ]);
226
227             foreach($data['order_items']['quantity'] as $productId => $quantity) {
228                 $sql = "UPDATE order_items
229                         SET quantity = :quantity
230                         WHERE order_id = :order_id AND product_id = :product_id";
231
232                 $result = $this->conn->prepare(query: $sql);
233                 $result->execute(params: [
234                     ':quantity' => (int) $quantity,
235                     ':product_id' => (int) $productId,
236                     ':order_id' => (int) $data['order']['id']
237                 ]);
238             }
239
240             $totalPrice = $this->countTotalPrice(data: $data['order_items']['quantity']);
241             $sql = "UPDATE orders
242                     SET status = :status, payment_status = :payment_status, total_price = :totalPrice
243                     WHERE id = :order_id";
244             $result = $this->conn->prepare(query: $sql);
245             $result->execute(params: [
246                 ':payment_status' => $data['order']['payment_status'],
247                 ':status' => $data['order']['status'],
248                 ':totalPrice' => $totalPrice,
249                 ':order_id' => (int) $data['order']['id']
250             ]);
251             $this->conn->commit();
252         }catch(Throwable $e) {
253             $this->conn->rollBack();
254             throw new Exception(message: "Nie udało się zaktualizować danych".$e->getMessage());
255         }
256     }
257
258     public function editAction(): void {
259         $id = (int) $this->request->get(param: 'id');
260         $this->dashboardView->renderView(params: [
261             'page' => 'orders/edit.php',
262             'data' => $this->dashboardModel->getOrder(id: $id),
263         ]);
264     }
265
266     0 references | 0 overrides
267     public function updateAction(): void {
268         if($this->request->isPost()) {
269             $this->dashboardModel->updateOrder( data: $this->takeDataToUpadteOrder());
270             header(header: 'location: /?module=order');
271         }
272     }

```

4.8.3 Fragment kodu odpowiedzialny za usuwanie zamówienia

```
37     public function deleteAction():void {
38         $id = (int) $this->request->get(param: 'id');
39
40         if($this->request->isPost()) {
41             $id = (int) $this->request->post(param: 'id');
42             $this->dashboardModel->deleteOrder(id: $id);
43             header(header: "location: /?module=order");
44             exit;
45         }
46
47         $this->dashboardView->renderView(params: [
48             'page' => "orders/delete.php",
49             'data' => $this->dashboardModel->getOrder(id: $id),
50         ]);
51     }
52
53     public function deleteOrder(int $id):void {
54         try {
55             $this->conn->beginTransaction();
56             $result = $this->conn->prepare(query: "DELETE FROM order_items WHERE order_id = :orderId");
57             $result->execute(params: [':orderId' => $id]);
58
59             $result = $this->conn->prepare(query: "SELECT address_id FROM orders WHERE id = :orderId");
60             $result->execute(params: [':orderId' => $id]);
61             $addressId = $result->fetchColumn();
62
63             $result = $this->conn->prepare(query: "DELETE FROM orders WHERE id = :orderId LIMIT 1");
64             $result->execute(params: [':orderId' => $id]);
65
66             $result = $this->conn->prepare(query: "DELETE FROM adress WHERE id = :addressId LIMIT 1");
67             $result->execute(params: [':addressId' => $addressId]);
68
69             $this->conn->commit();
70         }catch(Throwable $e) {
71             $this->conn->rollBack();
72             throw new Exception(message: "Nie udało się usunąć zamówienia");
73         }
74     }
75 }
```

4.9 Zarządzanie produktami

Administrator ma możliwość zarządzania produktami. Może usuwać, dodawać, edytować oraz wyświetlać szczegóły produktu.

4.9.1 Fragment kodu odpowiedzialny za dodawanie produktu

```
0 references | 0 overrides
13     public function createAction():void {
14         $this->dashboardView->renderView(params: [
15             'page' => 'products/create.php',
16             'categories' => $this->dashboardModel->getData(table: 'categories'),
17         ]);
18     }
19
20     0 references | 0 overrides
20     public function storeAction():void {
21         if($this->request->isPost()) {
22             $data = $this->takeProductData();
23             $this->dashboardModel->createProduct(data: $data);
24             header(header: 'location: /?module=product');
25         }else {
26             header(header: 'location: /?module=product');
27         }
28     }
29
30     public function createProduct(array $data):void {
31         try {
32             $imageName = null;
33
34             if($_FILES['image'] && $_FILES['image']['error'] == 0){
35                 $uploadDir = 'public/images/products/';
36
37                 if(!is_dir(filename: $uploadDir)) {
38                     mkdir(directory: $uploadDir, permissions: 0777, recursive: true);
39                 }
40
41                 $imageName = uniqid(prefix: 'product_', more_entropy: true).'.' .pathinfo(path: $_FILES['image']['name'], flags: PATHINFO_EXTENSION);
42                 $imagePath = $uploadDir.$imageName;
43
44                 if(!move_uploaded_file(from: $_FILES['image']['tmp_name'], to: $imagePath)) {
45                     throw new Exception(message: "Nie udało się przesłać obrazka");
46                 }
47             }
48
49             $sql = "INSERT INTO products(name, description, size, price, stock, category_id, image_url, created_at)
50             VALUES(:name, :description, :size, :price, :stock, :category_id, :image_url, NOW())";
51
52             $result = $this->conn->prepare(query: $sql);
53
54             $result->execute(params: [
55                 ':name' => $data['name'],
56                 ':description' => $data['description'],
57                 ':size' => $data['size'],
58                 ':price' => $data['price'],
59                 ':stock' => $data['stock'],
60                 ':category_id' => $data['category'],
61                 ':image_url' => $imageName
62             ]);
63             }catch(Throwable $e) {
64                 throw new Exception(message: "Nie udało się zaktualizować danych". $e->getMessage());
65             }
66         }
67     }
```

4.9.2 Fragment kodu odpowiedzialny za edytowanie produktu

```
30     public function editAction():void {
31         $id = (int) $this->request->get(param: 'id');
32         $this->dashboardView->renderView(params: [
33             'page' => 'products/edit.php',
34             'product' => $this->dashboardModel->getSingleElement(table: 'products', id: $id),
35             'categories' => $this->dashboardModel->getData(table: 'categories'),
36         ]);
37     }
38
39     0 references | 0 overrides
40     public function updateAction():void {
41         if($this->request->isPost()) {
42             $data = $this->takeProductData();
43             $this->dashboardModel->updateProduct(data: $data);
44             header(header: 'location: /?module=product');
45         }else {
46             header(header: 'location: /?module=product');
47         }
48
49     }
50
51     0 references | 0 overrides
52     public function updateProduct(array $data): void {
53         try {
54             $imageName = null;
55             $sql = "UPDATE products SET
56                 name = :name,
57                 description = :description,
58                 size = :size,
59                 price = :price,
60                 stock = :stock,
61                 category_id = :category
62                 ";
63
64             if($_FILES['image'] && $_FILES['image']['error'] == 0){
65                 $uploadDir = 'public/images/products/';
66
67                 if(!is_dir(filename: $uploadDir)) {
68                     mkdir(directory: $uploadDir, permissions: 0777, recursive: true);
69                 }
70
71                 $imageName = uniqid(prefix: 'product_', more_entropy: true).'.'.pathinfo(path: $_FILES['image']['name'], flags: PATHINFO_EXTENSION);
72                 $imagePath = $uploadDir.$imageName;
73
74                 if(move_uploaded_file(from: $_FILES['image']['tmp_name'], to: $imagePath)) {
75                     $sql .= ", image_url = :image_url";
76                 }else {
77                     throw new Exception(message: "Nie udało się przesłać obrazka");
78                 }
79             }
80             $sql .= " WHERE id = :id";
81
82             $result = $this->conn->prepare(query: $sql);
83
84             $params = [
85                 ':name' => $data['name'],
86                 ':description' => $data['description'],
87                 ':size' => $data['size'],
88                 ':price' => $data['price'],
89                 ':stock' => $data['stock'],
90                 ':category' => $data['category'],
91                 ':id' => (int) $data['id'],
92             ];
93
94             if($imagePath) {
95                 $params[':image_url'] = $imageName;
96             }
97
98             $result->execute(params: $params);
99         }catch(Throwable $e) {
100             throw new Exception(message: "Nie udało się zaktualizować danych". $e->getMessage());
101         }
102     }
103 }
```

4.9.3 Fragment kodu odpowiedzialny za wyświetlanie szczegółów produktu

```
49     public function showAction(): void {
50         $id = (int) $this->request->get(param: 'id');
51         $this->dashboardView->renderView(params: [
52             'page' => 'products/show.php',
53             'product' => $this->dashboardModel->getSingleElement(table: 'products', id: $id),
54             'categories' => $this->dashboardModel->getData(table: 'categories'),
55         ]);
56     }
57
58     public function getSingleElement(string $table, int $id):array {
59         try {
60             $sql = "SELECT * FROM $table WHERE id = $id";
61             $result = $this->conn->query(query: $sql);
62
63             $result = $result->fetch(mode: PDO::FETCH_ASSOC);
64         }catch(Throwable $e) {
65             throw new Exception(message: "Nie udało się pobrać elementu");
66         }
67
68         if(!$result) {
69             throw new Exception(message: "Nie ma elementu o takim id");
70         }
71
72         return $result;
73     }
74 }
```

4.9.4 Fragment kodu odpowiedzialny za usuwanie produktu

```
58     public function deleteAction(): void {
59         if($this->request->isPost()){
60             $id = (int) $this->request->post(param: 'id');
61             $this->dashboardModel->deleteProduct(id: $id);
62             header(header: 'location: /?module=product');
63         } else {
64             header(header: 'location: /?module=product');
65         }
66     }
67
139     public function deleteProduct(int $id): void {
140         try {
141             $sql = "DELETE FROM products WHERE id = $id LIMIT 1";
142             $this->conn->exec(statement: $sql);
143         } catch(Throwable $e) {
144             throw new Exception(message: "Nie udało się usunąć notatki");
145         }
146     }

```

4.10 Zarządzanie użytkownikami

Administrator ma możliwość przeglądania kont użytkownika (Sprawdzić zamówienia, które użytkownik złożył).

4.10.1 Fragment kodu odpowiedzialny za wyświetlanie szczegółów użytkownika.

```
13     public function showAction():void {
14         $userId = (int) $this->request->get(param: "id");
15
16         $this->dashboardView->renderView(params: [
17             "page" => "users/show.php",
18             "data" => $this->dashboardModel->getUserOrders(id: $userId),
19             "user" => $this->dashboardModel->getSingleElement(table: "users", id: $userId),
20         ]);
21     }
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380 }
```

5. Języki programowania, frameworki, usługi oraz biblioteki

5.1 Języki programowania

- **PHP** – Główny język aplikacji odpowiedzialny za logikę biznesową i interakcję z bazą danych.
 - Zalety:
 - Szerokie wsparcie społeczności i dokumentacji.
 - Dobra integracja z bazami danych.
- **HTML** – Podstawowy język do tworzenia struktury stron internetowych. Wykorzystywany do definiowania elementów takich jak nagłówki, akapity, obrazy, linki i formularze.
- Zalety:
 - Jest standardem dla tworzenia stron internetowych.
 - Łatwy do nauki i szeroko stosowany.
 - Działa na każdej przeglądarce, zapewniając szeroką kompatybilność.
 - Umożliwia tworzenie responsywnych interfejsów użytkownika w połączeniu z CSS i JavaScript.
- **CSS** – Odpowiada za stylizację interfejsu użytkownika, umożliwiając dostosowanie wyglądu aplikacji.
 - Zalety:
 - Pozwala na pełną kontrolę nad wyglądem strony.
 - Wspiera responsywność, dostosowując stronę do różnych ekranów.
 - Umożliwia stosowanie animacji i efektów wizualnych.
 - Może być używany z preprocesorami (itd. SASS) w celu lepszej organizacji kodu.
- **SQL** – Służy do operacji na bazie danych, takich jak pobieranie i modyfikacja danych.
 - Zalety:
 - Umożliwia szybkie przeszukiwanie i filtrowanie danych.
 - Zapewnia integralność i bezpieczeństwo danych.
 - Jest standardem w relacyjnych bazach danych.

5.2 Frameworki i biblioteki

- **TailwindCSS** – Używany do stylizacji interfejsu użytkownika, pozwala na szybkie tworzenie nowoczesnego designu.
 - Zalety:
 - Eliminuje konieczność pisania własnych arkuszy CSS.
 - Modułowa budowa pozwala na łatwą personalizację.
 - Wspiera utility-first approach, co skraca czas stylizacji.
- **PDO (PHP Data Objects)** – Abstrakcyjna warstwa dostępu do bazy danych, umożliwiająca bezpieczne wykonywanie zapytań SQL.
 - Zalety:
 - Obsługuje wiele różnych baz danych (MySQL, PostgreSQL, SQLite itd.).
 - Umożliwia stosowanie zapytań przygotowanych (prepared statements), co zwiększa bezpieczeństwo przed SQL Injection.
 - Łatwiejsza migracja kodu między różnymi systemami bazodanowymi.

5.3 Usługi zewnętrzne

- **Stripe** – Popularna platforma do obsługi płatności online, używana do realizacji transakcji w aplikacji e-commerce.
 - Zalety:
 - Łatwa integracja z PHP i JavaScript.
 - Obsługuje wiele metod płatności, w tym karty kredytowe, Apple Pay i Google Pay.
 - Wbudowane mechanizmy do obsługi subskrypcji i zwrotów.

6. Baza Danych

6.1 Technologia

Projekt wykorzystuje relacyjną bazę danych MySQL do przechowywania danych aplikacji, takich jak informacje o użytkownikach, produktach, zamówieniach oraz koszyku zakupowym.

6.2 Schemat bazy danych

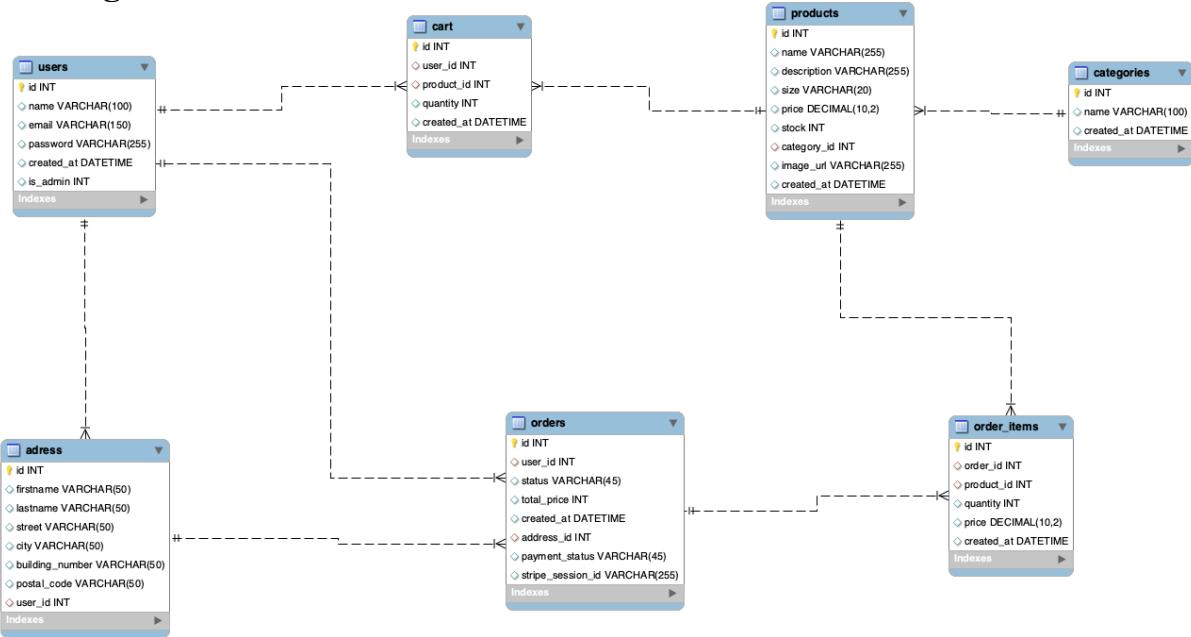
Baza danych składa się z siedmiu tabel:

- **users** – przechowuje dane użytkowników aplikacji.
- **adress** – zawiera adresy powiązane z użytkownikami.
- **products** – katalog produktów dostępnych w sklepie.
- **categories** – kategorie produktów.
- **orders** – zamówienia złożone przez użytkowników.
- **order_items** – szczegóły produktów w konkretnych zamówieniach.
- **cart** – aktualna zawartość koszyka użytkownika.

6.3 Relacje między tabelami

- Jeden użytkownik może mieć wiele adresów, zamówień oraz produktów w koszyku (1:N).
- Każdy adres należy do jednego użytkownika (N:1) i może być powiązany z wieloma zamówieniami (1:N).
- Każdy produkt jest przypisany do jednej kategorii (N:1), a jedna kategoria może mieć wiele produktów (1:N).
- Zamówienie należy do jednego użytkownika i jednego adresu dostawy (N:1).
- Jedno zamówienie może zawierać wiele produktów (1:N) w szczegółach zamówienia.
- Koszyk przechowuje wiele produktów powiązanych z jednym użytkownikiem (1:N).

6.4 Diagram ERD



7. Bezpieczeństwo

Bezpieczeństwo jest kluczowym aspektem każdej aplikacji internetowej, zwłaszcza w kontekście e-commerce, gdzie przetwarzane są dane użytkowników, hasła oraz informacje dotyczące płatności. Wdrożone mechanizmy ochrony obejmują m.in. bezpieczne przechowywanie haseł, ochronę przed atakami SQL Injection oraz odpowiednią walidację danych wejściowych.

7.1 Haszowanie haseł

Przechowywanie haseł w bazie danych w formie jawniej jest jednym z największych błędów, jakie można popełnić. Dlatego w aplikacji każde hasło użytkownika przed zapisaniem jest haszowane z wykorzystaniem funkcji **`password_hash()`** w PHP. Algorytm domyślnie używa **`bcrypt`**, który jest odporny na **ataki brute force** oraz umożliwia dodanie losowego "sola", co sprawia, że nawet identyczne hasła generują różne wartości.

Podczas logowania użytkownika hasło jest weryfikowane przy użyciu funkcji **`password_verify()`**, co zapewnia bezpieczeństwo i uniemożliwia odczytanie oryginalnej wartości hasła.

7.1.1 Fragment kodu z użyciem `password_hash()`.

```
52     $data['password'] = password_hash(password: $password, algo: PASSWORD_DEFAULT);
```

7.1.2 Fragment z użyciem `password_verify()`.

```
26         if(password_verify(password: $password, hash: $user['password'])) {
27             $_SESSION['user'] = $user;
28             $this->view->renderView(params: ['page' => 'start'],
29                                     message: ["messageTop" => "Udało się zalogować"]);
```

7.2 Ochrona przed SQL Injection

Ataki SQL Injection polegają na manipulacji zapytaniami SQL poprzez wstrzyknięcie złośliwego kodu do formularzy lub adresów URL. Aby temu zapobiec, w aplikacji wykorzystywane są przygotowane zapytania (prepared statements) oraz parametryzacja w PDO.

7.3 Walidacja danych wejściowych

Walentacja danych wejściowych ma na celu zapewnienie, że aplikacja przetwarza wyłącznie poprawne i bezpieczne dane. Wprowadzanie niezweryfikowanych wartości może prowadzić do błędów w systemie, a w skrajnych przypadkach umożliwić atakującemu przejęcie kontroli nad aplikacją.

Wszystkie dane podawane przez użytkownika są sprawdzane pod kątem:

- Typu danych – np. adres e-mail musi być poprawnym adresem,
- Zakresu wartości – np. długość hasła musi spełniać określone kryteria.