Dokumentacja aplikacji

1. Informacje ogólne.

Aplikacja stworzona przy wykorzystaniu Spring FrameWork wykorzystująca silnik bazy danych postgresql, Aplikacja służy do obsługi sklepu interneowego, umożliwia użutkownikom sprzedawanie i kupowanie produktów. Aplikacja udostępnia użytkownikom koszyk dzięki czemu mogą oni bez konieczności każdorazowego wybierania mieć dostęp do produktów, które chcą kupić. W celu odświerzenia tokena konieczne jest wysłanie przez klienta refresh tokena na podstawie którego tworzony jest nowy access token. Aplikacja posiada panel administratora w którym to administrator może zarządzać użykownikami i ich zasobami, a także dodawać zasoby wymagające jego uprwnień. Po rejestracji aplikacja wymaga potwierdzenia adresu email w innym przypadku konto jest zablokowane.

1. Opis API.

API zostało zrealizowane według wzorca REST. Komunikacja polega na wysłaniu żądania HTTP pod adres API i ewentualnym przesłaniu wymaganych danych. Odpowiedź jest zwracana w formacie JSON

1. Uwierzytelnienie

Do autoryzaji aplikacja wykorzystuje JSON Web Token (JWT). Logika jego działania opiera siię na dwóch tokenaach tzw. acces token i refresh token są one tworzone podczas logowania użytkownika do aplikacji i zwracane na frontend. Następnie po stronie frontu trzeba przechwycić oba tokeny i zapisać je lokalnie w przeglądarce. Przy wykonaniu requetu wymagającego autoryzacji należy dodać acces token do nagłówka Authorization poprzedzając jego wartość przedrostkiem „Bearer ”. Dwa tokeny są potrzebne poniewarz acces token w celu większego bezpieczeństwa posiada krótki termin ważności i kiedy on się skończy przy wykorzystaniu refresh tokena tworzony jest nowy acces token i tak w kółko.

Każde wysłane żądanie po API wymaga uwierzytelnienia. Uwierzytelnienie następuje poprzez wysłanie nagłówku HTTP o nazwie Authorization.

Przykład autoryzacji w interakcji z klientem w React.js

Metoda logowania po stronie kienta

#1

login = (form)=>{

    let loginFormError = false

#1

Przesyłane dane

    let sendLogin = {

      "username": form.login,

      "password": form.password,

    }

Metoda fetch(wysłanie danych na odpowiedzni endpoint)

    fetch("http://localhost:8080/montanashop/login", {

      method: 'POST',

      headers: {

        'Accept': 'application/json, text/plain, \*/\*',

        'Content-type': 'application/json',

        'Access-Control-Allow-Origin': 'http://localhost:8080',

        'X-Requested-With': 'XMLHttpRequest',

      }, body:JSON.stringify(sendLogin)

    }

   ).then(response=>{

Sprawdzenie odpowiedzi serwera

    if(response.status!==200){

      loginFormError = true;

      return response.json();

    }

    else{

      return response.json()

    }

   }).then(response=>{

    if(loginFormError){

      throw response

    }else{

      return response

    }

  }).then(response=>

    {

Po ;pozytywnym zalogowaniu zapisanie w local storage zwróconych tokenów

#2

      localStorage.setItem('accesToken', response["acces-token"]);

      localStorage.setItem('refreshtoken', response["refreshtoken"]);

Przekierowanie na stronę główną po udanym logowaniu

      return fetch("http://localhost:8080/montanashop/login-page",{

        method: 'GET',

        headers: {

          'Accept': 'application/json, text/plain, \*/\*',

          'Content-type': 'application/json',

          'X-Requested-With': 'XMLHttpRequest',

          'Access-Control-Allow-Origin': '\*',

          'Authorization': "Bearer "+localStorage.getItem('accesToken')

        },

      }).then(

        res=>{

          if(res.status===200){

            return res.json()

          }

        }

      )

      .then(res=>{

        localStorage.setItem("in","1")

        this.getShoppibngCartQuantity()

        this.getCategoryProduct()

        if(res["role-user"]==="[ROLE\_USER]"){

          localStorage.setItem("role","1")

        }else if(res["role-user"]==="[ROLE\_ADMIN]"){

          localStorage.setItem("role","2")

        }

        this.setState({

          logIn:1,

          roleUser:res["role-user"]

        })

      })

      }

  )

      .catch(err=>{

        this.validFormAfterBackend(form,err)

      })

  }

Metoda login zwracająca tokeny acces i refresh

#1

@PostMapping("login")  
public HashMap<String,String> login(@RequestBody Client client){  
 HashMap<String,String> tokens = new HashMap<>();  
 if (clientManager.existByUserName(client.getUsername())){  
 Client client1 = clientManager.findByUserName(client.getUsername());  
 boolean result = passwordEncoder.matches(client.getPassword(),client1.getPassword());  
 if(result){  
 if (client1.isEnabled()){  
 AccesToken accesToken = jwTokenManager.createAccesToken(client1);  
 RefreshToken refreshToken = jwTokenManager.createRefreshToken(client1);  
 tokens.put("acces-token",accesToken.getToken());  
 tokens.put("refreshtoken",refreshToken.getRefreshToken());  
 }  
 else {  
 throw new InactiveAccountException();  
 }  
 }else {  
 throw new BadPasswordException();  
 }  
 }else {  
 throw new UsernameNotExistException();  
 }

#2 Zwrócenie tokenów  
 return tokens;  
}

Metoda przekierowująca na stronę główną po pozytywnym logowaniu.

@GetMapping("login-page")  
public ResponseEntity <Map<String,String>> home(){  
 Map<String, String> roleUser = new HashMap<>();  
 roleUser.put("role-user", clientManager.getAuthenticatedClient().getAuthorities().toString());  
 return new ResponseEntity<Map<String,String>>(roleUser, new HttpHeaders(), HttpStatus.*OK*);  
}

1. **Odpowiedzi API.**

Odpowiedź na żądania wysyłane do API jest w formacie JSON.

Jeśli zapytanie powiedzie się wtedy pod kluczem „status” będziemy mieli kod oznaczający prwidłowo zrealizowane zapytanie np. 200, 201... w zależności od rodzaju zapytania 'OK'. Aby odczytać dane zwócone przez metodę należy skonwertować odpowiedź jeszcze raz na JSON. Natomiast jeśli wystąpi błąd pod kluczem „status” będzie kod błędu, a pod kluczem znajdzie się opis błędów jakie wystąpiły wraz z kodami błędów. Dodatkowo dla niektórych requestów pod kluczem message będzie dokładny opis błędu.

1. **Ścieżka dostępu**

Każda z akcji jaką można wykonać ma swoją osobną ścieżkę dostępu. Poza logowaniem i rejestracją ścieżka zbudowana jest w następujący sposób:

/host/[nazwaaplikacji]/[nazwa zasoby np, product]/[akcja np. delete]

Scierzka z parametrem

/host/[nazwaaplikacji]/[nazwa zasoby np, product]/[akcja np. delete]/np. 2

Lub

/host/[nazwaaplikacji]/[nazwa zasoby np, product]/[akcja np. delete]?[nazwa parametru]=np.2

1. **Opis metod używanych w API**

**Dodawanie/rejestracja użytkownika.**

Kontroler służy do wysłania da danych podanych przz użykownika w celu utworzenia konta.

Wywołanie POST host/nazwaaplikacji/register.

Dane w POST w fomacie json {username : nazwa użytkownika,password : hasło,email : adres email}

Przykład

@PostMapping("register")  
public ResponseEntity<Token> add(@RequestBody Client client){

HttpHeaders responseHeaders = new HttpHeaders();

clientManager.addClient(client, passwordEncoder);

Token token = clientManager.crreateToken(client);

return new ResponseEntity<Token>(token, new HttpHeaders(), HttpStatus.*CREATED*);  
}

**Update użytkownika.**

Kontroler służy do wysłania da danych podanych perz użykownika w celu aktualizaji danych dotyczących konta.

Wywołanie PUT host/nazwaaplikacji/updater.

Dane w PUT w formacie json w zależności jakie dane chcemy aktualizować.

{username : nazwa użytkownika,password : hasło,email : adres email}

Przykład.

@PutMapping("client/update")  
public ResponseEntity update(@RequestBody Client client){

ResponseEntity entity = null;

if (clientRepository.existsByUsername(client.getUsername())){

throw new UsernameExistException(client.getUsername());

}else if (clientRepository.existsByEmail(client.getEmail())){

throw new EmailExsistException(client.getEmail());

}  
 else {  
 Client client1 = clientManager.getById(client.getId()).get();  
   
 if (clientManager.updateClient(client1)){  
 entity = new ResponseEntity(HttpStatus.*OK*);  
 }else {  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
 }  
}

**Pobranie wszystkich użytkowników.**

Kontroler służy do wyszukania wszystkich użytkowników

Wywołanie GET host/nazwaaplikacji/client/all.

Przykład.

@GetMapping("client/all")  
public HttpEntity<List<Client>> getAll(){

return new ResponseEntity<List<Client>>(clientManager.getAll(),new HttpHeaders(), HttpStatus.*OK*);  
}

**Pobranie użytkownika po id.**

Kontroler służy do wyszukania użytkownika po id

Wywołanie GET host/nazwaaplikacji/client?id=id\_użytkownika.

Przykład.

@GetMapping("client")  
public ResponseEntity<Optional> getById(@RequestParam long id){

HttpHeaders httpHeaders = new HttpHeaders();

return new ResponseEntity<Optional>( clientManager.getById(id),httpHeaders,HttpStatus.*OK*);  
}

**Usuwanie użytkownika po id.**

Kontroler służy do usuwania użytkownika po id

Wywołanie DELETE host/nazwaaplikacji/client/delete/id\_użytkownika.

Przykład.

@DeleteMapping("client/delete/{id}")  
public ResponseEntity delete(@PathVariable(name = "id") long id){

ResponseEntity entity = null;  
 if (clientManager.deleteClient(id)){  
 entity = new ResponseEntity(HttpStatus.*OK*);  
  
 }else {  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
}

**Usuwanie konta użytkownika.**

Kontroler służy do usuwania konta użytkownika

Wywołanie DELETE host/nazwaaplikacji/client/delete.

Przykład.

@DeleteMapping("client/delete")  
public ResponseEntity deleteAccount(){

ResponseEntity entity = null;

if (clientManager.deleteClient(clientManager.getAuthenticatedClient().getId())){  
 entity = new ResponseEntity(HttpStatus.*OK*);

}else {  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
}

**Pobranie wszystkich kategorii**

Kontroler służy do pobrania wszystkich kategorii

Wywołanie GET host/nazwaaplikacji/category/all.

Przykład.

@GetMapping("/all")  
public Iterable<Category> getAll(){  
 return categoryManager.getAll();  
}

**Dodawanie kategorii**

Kontroler służy do dodania kategorii

Wywołanie POST host/nazwaaplikacji/category/update.

Dane w POST w formacie json {name : nazwa}

Przykład.

@PostMapping("/add")  
public ResponseEntity add(@RequestBody Category category){  
 ResponseEntity entity;  
 if(categoryManager.addCategory(category)){  
 entity = new ResponseEntity(HttpStatus.*CREATED*);  
 }else {  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
}

**Usuwanie kategorii po id.**

Kontroler służy do usunięcia kategorii po id

Wywołanie DELETE host/nazwaaplikacji/category/delete?id=id\_kategorii.

Przykład.

@DeleteMapping("/delete")  
public ResponseEntity delete(@RequestParam long id){  
 {  
 ResponseEntity entity;  
 if(categoryManager.deleteCategory(id)){  
 entity = new ResponseEntity(HttpStatus.*OK*);  
 }else {  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
 }  
}

**Update kategorii**

Kontroler służy do aktualizacji kategorii

Wywołanie PUT host/nazwaaplikacji/category/update.

Dane w PUT w formacie json {name : nazwa}

Przykład.

@PutMapping("/update")  
public ResponseEntity update(@RequestBody Category category)  
{  
 ResponseEntity entity;  
 if(categoryManager.updateCategory(category)){  
 entity = new ResponseEntity(HttpStatus.*OK*);  
 }else {  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
}

**Pobranie wszystkich produktów**

Kontroler służy do pobrania wszystkich produktów

Wywołanie GET host/nazwaaplikacji/product/all.

Przykład.

@GetMapping("/all")  
 public HttpEntity<List<Prodyct>> geetAll(){  
 return new ResponseEntity<List<Prodyct>>((List<Prodyct>) productManager.getAll(),new HttpHeaders(), HttpStatus.*OK*);  
}

**Pobranie produktu po nazwie.**

Kontroler służy do pobrania produktu po id.

Wywołanie GET host/nazwaaplikacji/product/all-by-name.

Przykład.

@GetMapping("/all-by-name")  
public HttpEntity<List<Prodyct>> getByName(@RequestParam String name){  
 return new ResponseEntity<List<Prodyct>>((List<Prodyct>) productManager.getByName(name),new HttpHeaders(), HttpStatus.*OK*);  
}

**Pobranie produktów po id kategorii**

Kontroler służy do pobrania produktów po id kategorii.

Wywołanie GET host/nazwaaplikacji/product/all-by-category?id=id\_kategorii.

Przykład.

@GetMapping("/all-by-category-and-name")  
public HttpEntity<List<Prodyct>> getAllByNameAndCategory(@RequestParam String name, @RequestParam long id){  
 System.*out*.println(name+ " " +id);  
 return new ResponseEntity<List<Prodyct>>((List<Prodyct>) productManager.getAllByNameAndCategory(id,name),new HttpHeaders(), HttpStatus.*OK*);  
}

**Dodawanie produktu**

Wywołanie POST host/nazwaaplikacji/product/add.

Dane w POST w formacie form-data product: {name : nazwa\_produktu, price : cena, quantity- ilość, productDescription : opis produktu}, idCategotu – id kategorii, file – ścierzka do pliku ze zdjęciem

Przykład.

@PostMapping("/add")  
public ResponseEntity add(@RequestParam("product") String prodyct, @RequestParam("idCategory") String id,  
 @RequestPart("file") MultipartFile file) throws JsonProcessingException {  
  
 ResponseEntity entity = null;  
 ObjectMapper mapper = new ObjectMapper();  
 Prodyct prodyct1 = mapper.readValue(prodyct, Prodyct.class);  
 long idCategory = Long.*parseLong*(id);  
 Category category = categoryManager.getById(idCategory);  
 prodyct1.setCategory(category);  
 String path = uploadManager.uploadFile(file);  
 prodyct1.setClient(clientManager.getAuthenticatedClient());  
 prodyct1.setPhoto(path);  
 if (productManager.add(prodyct1)){  
 entity = new ResponseEntity(HttpStatus.*CREATED*);  
 }  
 else{  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
}

**Usuwanie produktu po id.**

Kontroler służy do usunięcia produktui po id

Wywołanie DELETE host/nazwaaplikacji/producty/delete?id=id\_produktu.

Przykład.

@DeleteMapping("/delete")  
public ResponseEntity delete(@RequestParam long id){  
 ResponseEntity entity = null;  
 if( productManager.delete(id)){  
 entity = new ResponseEntity(HttpStatus.*OK*);  
 }else {  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
  
 }  
 return entity;  
}

**Update produktu**

Kontroler służy do aktualizacji produktu

Wywołanie PUT host/nazwaaplikacji/product/add.

Dane w PUT w formacie form-data product: {name : nazwa\_produktu, price : cena, quantity- ilość, productDescription : opis produktu}, idCategotu – id kategorii, file – ścierzka do pliku ze zdjęciem

Przykład.

@PutMapping("/update")  
public ResponseEntity update(@RequestParam("product") String prodyct, @RequestParam("idCategory") String id,  
 @RequestPart("file") MultipartFile file) throws JsonProcessingException {  
  
 ResponseEntity entity = null;  
 ObjectMapper mapper = new ObjectMapper();  
 Prodyct prodyct1 = mapper.readValue(prodyct, Prodyct.class);  
 Prodyct updateProduct = productManager.getById(prodyct1.getId()).get();  
   
 long idCategory = Long.*parseLong*(id);  
 Category category = categoryManager.getById(idCategory);  
 updateProduct.setName(prodyct1.getName());  
 updateProduct.setQuantity(prodyct1.getQuantity());  
 updateProduct.setPrice(prodyct1.getPrice());  
 updateProduct.setCategory(category);  
 String path = uploadManager.uploadFile(file);  
 updateProduct.setPhoto(path);  
 if (productManager.update(updateProduct)){  
 entity = new ResponseEntity(HttpStatus.*OK*);  
 }  
 else{  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
}

**Pobranie wszystkich koszyków**

Kontroler służy do pobrania wszystkich koszyków

Wywołanie GET host/nazwaaplikacji/shoppingcart/all.

Przykład.

@GetMapping("/all")  
public Iterable<ShoppingCart> getAll(){  
 return shoppingCartManager.getAll();  
}

**Dodawanie koszyka**

Wywołanie POST host/nazwaaplikacji/ shoppingcart /add.

Dane w POST w formacie json {client:{ id:id\_klienta}}

Przykład.

@PostMapping("/add")  
public ResponseEntity add(@RequestBody String shoppingCart) throws IOException { ;  
 ObjectMapper mapper = new ObjectMapper();  
 ShoppingCart shoppingCart1 = mapper.readValue(shoppingCart, ShoppingCart.class);  
 Client client = clientManager.getById(shoppingCart1.getClient().getId()).get();  
 shoppingCart1.setClient(client);  
  
 ResponseEntity entity;  
 if (shoppingCartManager.add(shoppingCart1)){  
 entity = new ResponseEntity(HttpStatus.*OK*);  
 }else {  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
}

**Usuwanie koszyka po id.**

Kontroler służy do usunięcia koszyka i po id

Wywołanie DELETE host/nazwaaplikacji/ shoppingcart /delete?id=id\_produktu.

Przykład.

@DeleteMapping("/delete")  
public ResponseEntity delete(@RequestParam long id){  
 ResponseEntity entity = null;  
 if( productManager.delete(id)){  
 entity = new ResponseEntity(HttpStatus.*OK*);  
 }else {  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
  
 }  
 return entity;  
}

**Update koszyka**

Kontroler służy do aktualizacji koszyka

Wywołanie PUT host/nazwaaplikacji/ shoppingcart /UPDATE.

Dane w PUT {client:{ id:id\_klienta}}

Przykład.

@PutMapping("/update")  
public ResponseEntity update(@RequestBody String shoppingCart) throws JsonProcessingException {  
 ResponseEntity entity = null;  
 ObjectMapper mapper = new ObjectMapper();  
 ShoppingCart shoppingCart1 = mapper.readValue(shoppingCart, ShoppingCart.class);  
 ShoppingCart shoppingCart2 = shoppingCartManager.getById(shoppingCart1.getId());  
 System.*out*.println(shoppingCart2.getClient().getId());  
 shoppingCartManager.add(new ShoppingCart(shoppingCart2.getClient()));  
 Client client = clientManager.getById(shoppingCart1.getClient().getId()).get();  
 if (client.getShoppingCart()!=null){  
 shoppingCartManager.delete(client.getShoppingCart().getId());  
 }  
 shoppingCart1.setClient(client);  
 if (shoppingCartManager.update(shoppingCart1)){  
 entity = new ResponseEntity(HttpStatus.*OK*);  
 }else {  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
}

**Pobranie wszystkich pozycji w koszyku**

Kontroler służy do pobrania wszystkich pozycji w koszyku

Wywołanie GET host/nazwaaplikacji/positionshopping/all.

Przykład.

@GetMapping("/all")  
public HttpEntity<List<PositionShopingJoinProduct>> getAll(){  
  
 return new ResponseEntity<List<PositionShopingJoinProduct>>((List<PositionShopingJoinProduct>) positionShoppingManager.getAll(),new HttpHeaders(), HttpStatus.*OK*);  
}

**Dodawanie pozycji w koszyku**

Wywołanie POST host/nazwaaplikacji/positionshopping/add.

Dane w POST w formacie json {shoppingcart:{ id:id\_koszyka}, prodyct:{id:id\_produktu}}}

Przykład.

@PostMapping("/add")  
public ResponseEntity add(@RequestBody PositionShopping positionShopping) throws JsonProcessingException {  
 Prodyct prodyct = productManager.getById(positionShopping.getProdyct().getId()).get();  
 prodyct.setQuantity(prodyct.getQuantity()-1);  
 ResponseEntity entity = null;  
 if(positionShoppingManager.add(positionShopping) && productManager.update(prodyct)){  
 entity = new ResponseEntity(HttpStatus.*CREATED*);  
 }else{  
 entity = new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
}

**Usuwanie pozycji w koszyku po id.**

Kontroler służy do usunięcia pozycji w koszyku i po id

Wywołanie DELETE host/nazwaaplikacji/positionshopping/delete?id=id\_pozycji\_w\_koszyku.

Przykład.

@DeleteMapping("/delete")  
public ResponseEntity delete(@RequestParam long id){  
 ResponseEntity entity;  
 if (positionShoppingManager.delete(id)){  
 entity = new ResponseEntity<String>(HttpStatus.*OK*);  
 }else{  
 entity = new ResponseEntity<String>(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
}

**Update pozycji w koszyku.**

Kontroler służy do aktualizacji pozycji w koszyku.

Wywołanie PUT host/nazwaaplikacji/positionshopping/update.

Dane w PUT w formacie json {shoppingcart:{ id:id\_koszyka}, prodyct:{id:id\_produktu}}}

Przykład.

@PutMapping("/update")  
public ResponseEntity update(@RequestBody PositionShopping positionShopping) throws JsonProcessingException {  
 ResponseEntity entity = null;  
 if (positionShoppingManager.update(positionShopping)){  
 entity = new ResponseEntity<String>(HttpStatus.*OK*);  
 }else{  
 entity = new ResponseEntity<String>(HttpStatus.*NOT\_FOUND*);  
 }  
 return entity;  
}