Dokument: Projekt aplikacji – BookSwap   
v.2024-03-18

**Projekt aplikacji**

Software Architecture Design (SAD)

Spis treści

[1. Wstęp 2](#_Toc170261193)

[2. Cel i zakres dokumentu 2](#_Toc170261194)

[3. Definicja architektury aplikacji 2](#_Toc170261195)

[4. Cele i ograniczenia architektury 2](#_Toc170261196)

[5. Obraz logiczny aplikacji 2](#_Toc170261197)

[5.1 Charakterystyka pakietów 2](#_Toc170261198)

[5.2 Diagram klas aplikacji 2](#_Toc170261199)

[5.3 Specyfikacja funkcji i metod aplikacji 3](#_Toc170261200)

[6. Dynamiczny obraz modelowanej aplikacji 3](#_Toc170261201)

[6.1 Diagram sekwencji UML dla obiektów 3](#_Toc170261202)

[6.2 Diagram aktywności UML dla obiektów 4](#_Toc170261203)

[7. Statyczny obraz modelowanej aplikacji 4](#_Toc170261204)

[8. Projekt bazy danych 5](#_Toc170261205)

# Wstęp

Dokument opisuje architekturę aplikacji BookSwap, systemu umożliwiającego użytkownikom wymianę książek online. Aplikacja pozwala na przeglądanie, oferowanie i wymianę książek między zarejestrowanymi użytkownikami.

# Cel i zakres dokumentu

Celem dokumentu jest przedstawienie szczegółowej architektury aplikacji BookSwap,   
w tym logicznego projektu aplikacji, struktury bazy danych oraz interakcji między komponentami systemu. Zakres dokumentu obejmuje projekt architektury oprogramowania, projekt interfejsów użytkownika, oraz struktury danych.

# Definicja architektury aplikacji

Architektura aplikacji BookSwap opiera się na modelu klient-serwer. Frontend aplikacji został zaprojektowany z wykorzystaniem HTML, CSS i JavaScript, z biblioteką React. Backend aplikacji zaimplementowano w Java, używając frameworka Spring Boot, co zapewnia potężne narzędzia do budowania REST API oraz zarządzania aplikacją, w tym bezpieczne połączenia z bazą danych i zarządzanie transakcjami.

# Cele i ograniczenia architektury

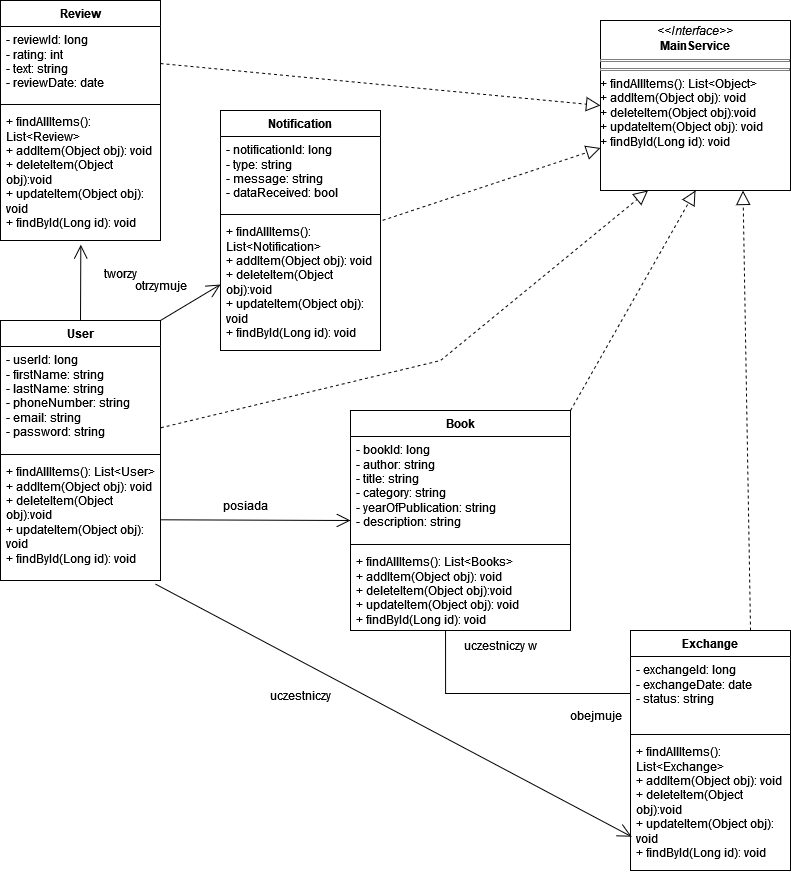
Celem architektury jest zapewnienie skalowalności, bezpieczeństwa i łatwości użytkowania. Ograniczenia to między innymi zapewnienie ochrony danych osobowych oraz zapewnienie wysokiej dostępności systemu.

# Obraz logiczny aplikacji

## Charakterystyka pakietów

Ze względu na niską złożoność aplikacji nie ma podziału na odrębne pakiety.

## Diagram klas aplikacji



## Specyfikacja funkcji i metod aplikacji

1. @GetMapping("/users")

2. public List<UsersDto> getUsersDto(){

3. return usersService.findAllItems().stream()

4. .map(users -> new UsersDto(

5. users.getUserID(),

6. users.getFirstName(),

7. users.getLastName(),

8. users.getEmail(),

9. users.getPassword(),

10. users.getShippingAddress(),

11. users.getPhoneNumber()))

12. .collect(Collectors.toList());

13. }

14.

15. @GetMapping("/users/notifications/{UserID}")

16. public List<NotificationDto> getUserNotifications(@PathVariable long UserID){

17. Users users = usersService.findById(UserID);

18. return users.getNotifications().stream()

19. .map(notification -> new NotificationDto(

20. notification.getNotificationID(),

21. notification.getType(),

22. notification.getMessage(),

23. notification.isDataReceived()))

24. .collect(Collectors.toList());

25. }

26.

27. @PostMapping("/users")

28. public void createUserDto(@RequestBody UsersDto usersDto) {

29. Users users = new Users();

30. users.setFirstName(usersDto.firstName());

31. users.setLastName(usersDto.lastName());

32. users.setEmail(usersDto.email());

33. users.setShippingAddress(usersDto.shippingAddress());

34. users.setPhoneNumber(usersDto.phoneNumber());

35. //users.setPassword(passwordEncoder.encode(usersDto.password()));

36. users.setPassword(usersDto.password());

37. usersService.addItem(users);

38. }

39.

Ten fragment kodu zawiera trzy główne funkcje: pierwsza funkcja pobiera listę wszystkich użytkowników, przekształca ich dane na odpowiedni format i zwraca tę listę. Druga funkcja pobiera identyfikator użytkownika z adresu internetowego, znajduje tego użytkownika, pobiera jego powiadomienia, przekształca je na odpowiedni format i zwraca jako listę. Trzecia funkcja przyjmuje dane nowego użytkownika, tworzy nowego użytkownika na podstawie tych danych i dodaje go do systemu.

1. @GetMapping("/books")

2. public List<BooksDto> getBooksDto(){

3. return booksService.findAllItems().stream()

4. .map(books -> new BooksDto(books.getBookID(),

5. books.getTitle(),

6. books.getCategory(),

7. books.getYearOfPublication(),

8. books.getDescription()))

9. .collect(Collectors.toList());

10. }

11.

12. @GetMapping("/books/users")

13. public List<BooksUserDto> getBooksUsersDto(){

14. return booksService.findAllItems().stream()

15. .map(books -> new BooksUserDto(books.getBookID(),

16. books.getTitle(),

17. books.getCategory(),

18. books.getYearOfPublication(),

19. books.getDescription(),

20. books.getUser().getUserID(),

21. books.getUser().getFirstName(),

22. books.getUser().getLastName(),

23. books.getUser().getEmail(),

24. books.getUser().getPassword(),

25. books.getUser().getShippingAddress(),

26. books.getUser().getPhoneNumber()))

27. .collect(Collectors.toList());

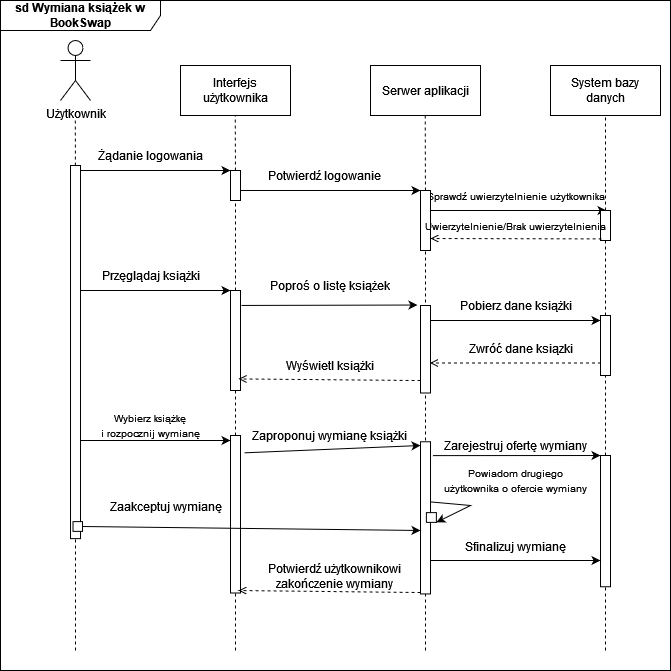
28. }

29.

Ten fragment kodu zawiera dwie główne funkcje: pierwsza funkcja pobiera listę wszystkich książek, przekształca ich dane na odpowiedni format i zwraca tę listę. Druga funkcja pobiera listę wszystkich książek wraz z informacjami o użytkownikach, którzy posiadają te książki, przekształca te dane na odpowiedni format i zwraca jako listę.

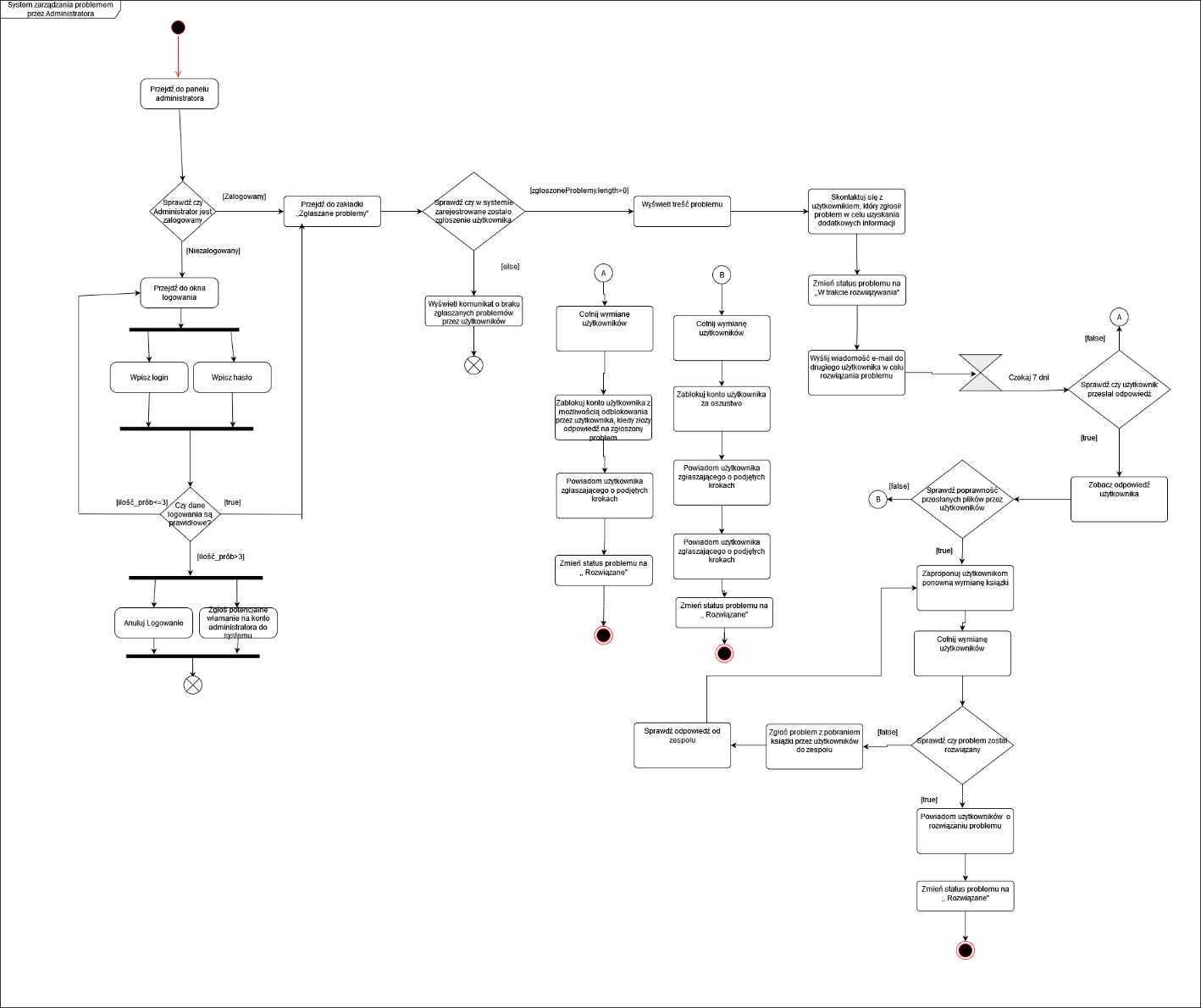
# Dynamiczny obraz modelowanej aplikacji

## Diagram sekwencji UML dla obiektów

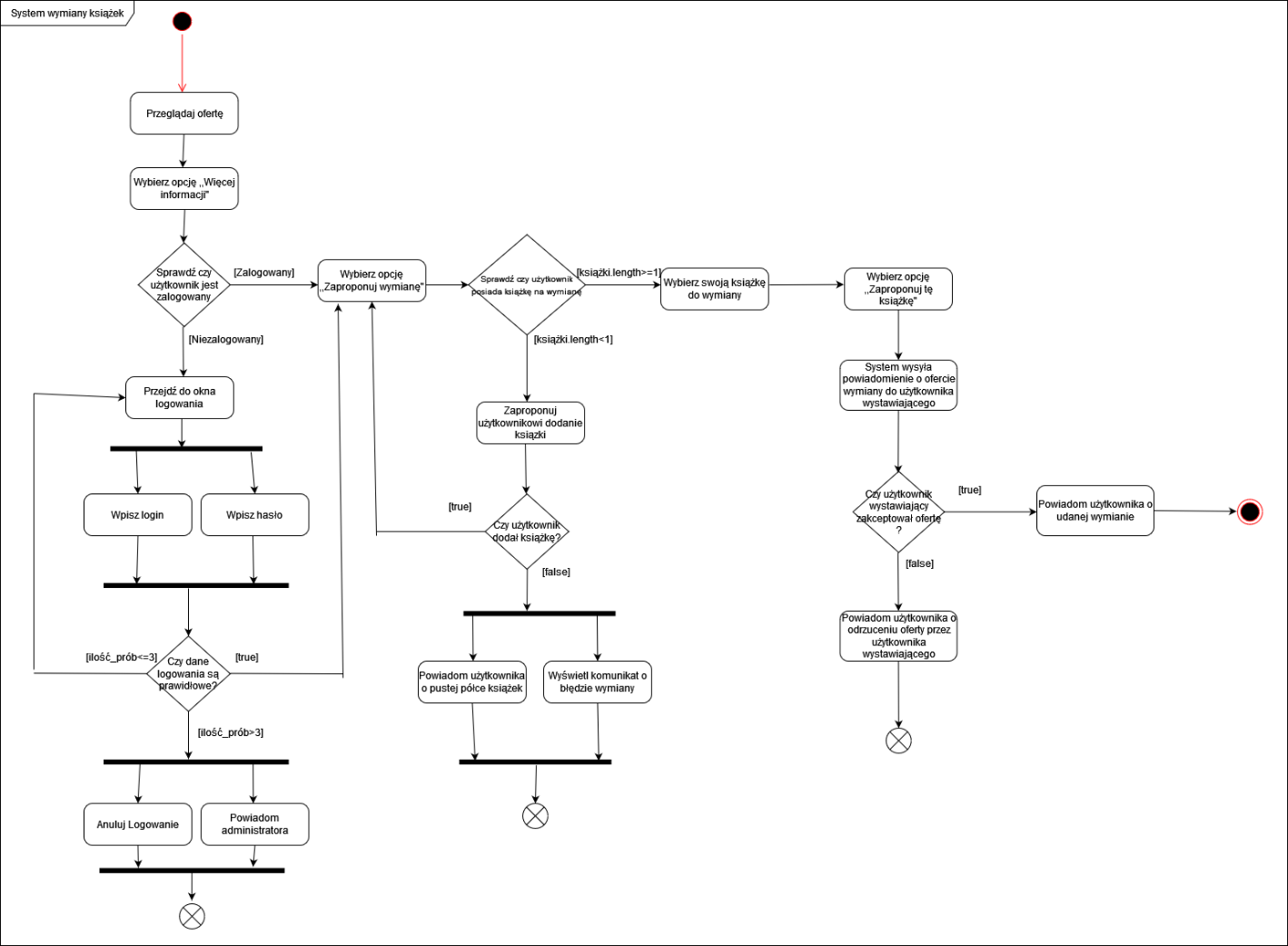


## Diagramy aktywności UML

## Diagram aktywności zarządzanie zgłoszonymi problemami

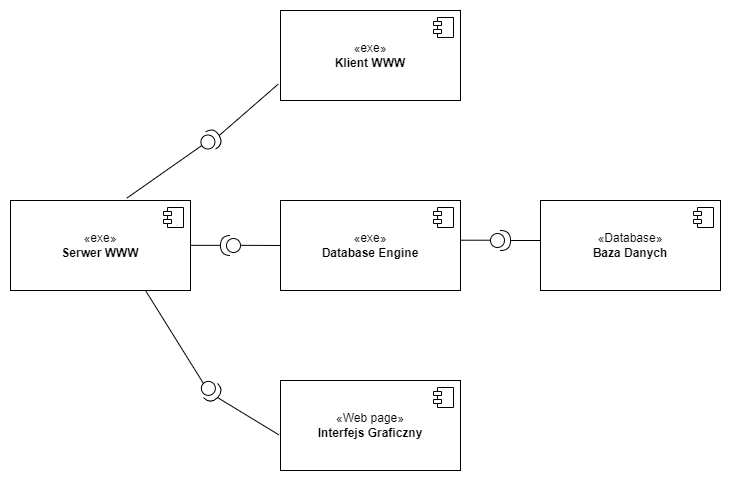


* + 1. Diagram aktywności wymiana książek

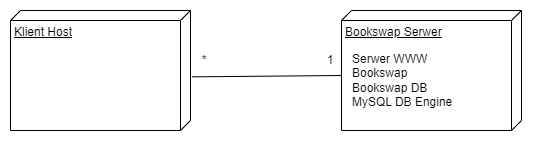


# Statyczny obraz modelowanej aplikacji

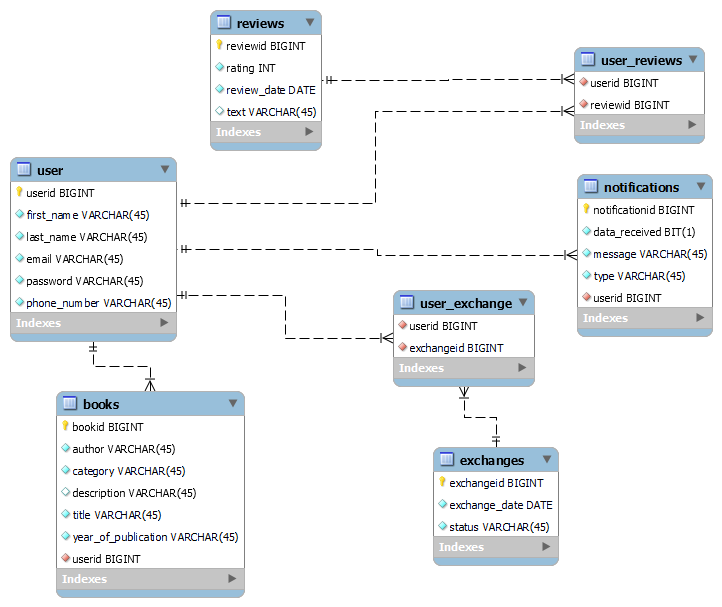
### Diagram komponentów UML



### Diagram instalacji UML



# Projekt bazy danych



# Diagramy UML-based Web Engineering(UWE)

## Diagram prezentacji

## 

## Diagram nawigacji

## 

## Diagram zawartości

## 