

**Politechnika Świętokrzyska w Kielcach**

Wydział Elektrotechniki, Automatyki i Informatyki

Politechnika Świętokrzyska  
Kielce University of Technology

## **Inżynieria Systemów Informacyjnych**

### **Projekt**

## **Aplikacja do zarządzania siłownią**

**Skład zespołu:**

Rafał Grot

Filip Stępień

Bartłomiej Karkoszka

Mateusz Karbowniczak

Kierunek/specjalność: Systemy informacyjne

Studia: stacjonarne

Numer grupy: 3ID11B

Kielce, 20 czerwca 2025

# 1 Wprowadzenie/Cel laboratorium

## 1.1 Krótki opis aplikacji

Aplikacja to system do zarządzania siłownią. Umożliwia kompleksowe zarządzanie obiektem fitness, w tym: rejestrację użytkowników, zarządzanie członkostwami, planowanie sesji treningowych oraz obsługę płatności. System zapewnia bezpieczną autoryzację poprzez integrację z Keycloak (OIDC) i oferuje REST API z pełną dokumentacją w Swaggerze.

## 1.2 Wykorzystane technologie oraz narzędzia

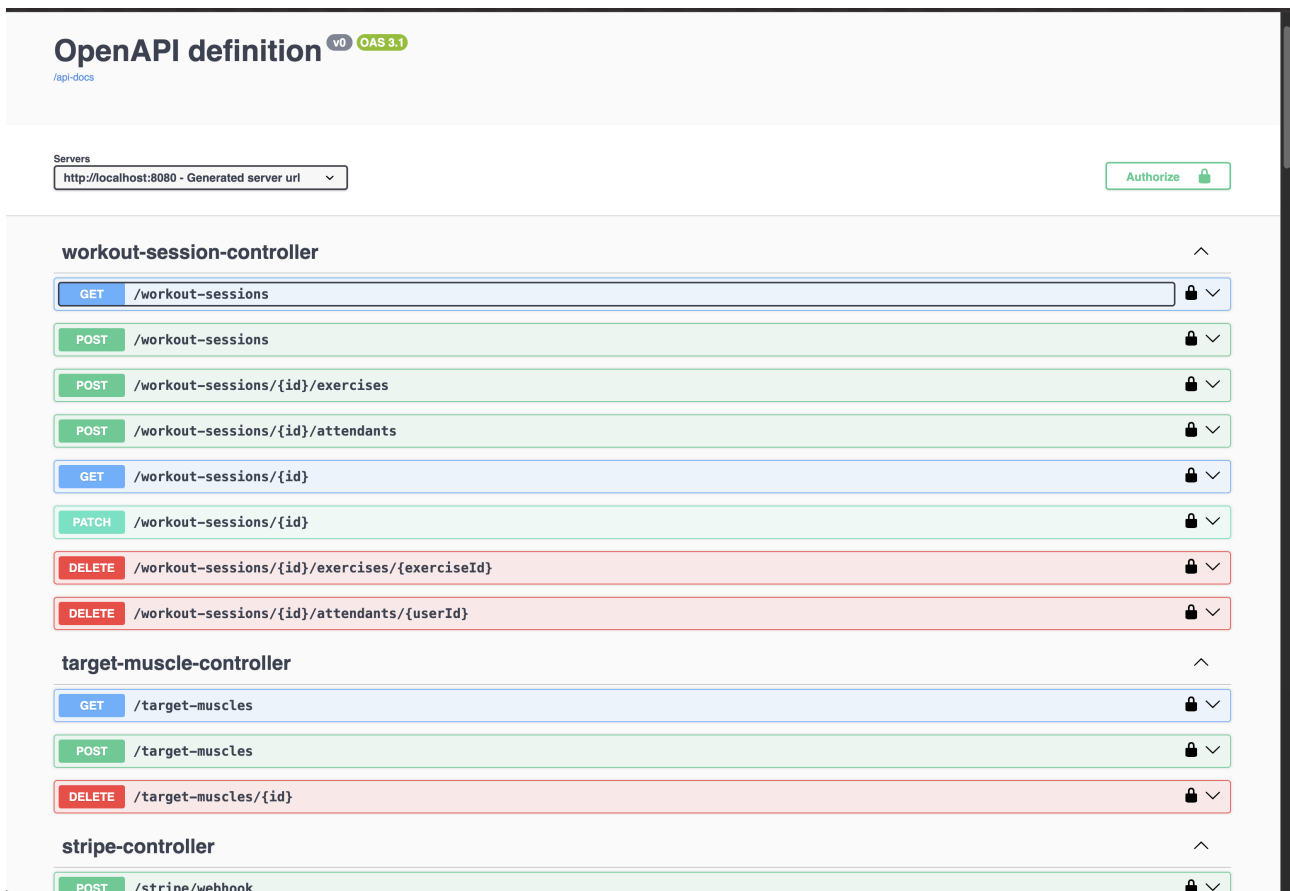
- **Nix** - System do zarządzania pakietami i środowiskami – pozwala tworzyć powtarzalne konfiguracje projektów.
- **git** - system kontroli wersji
- **GitHub Actions** - System CI/CD w GitHubie do automatyzacji testów, buildów i wdrożeń przy pomocy workflowów.
- **Spring Boot** - Framework w Javie do szybkiego tworzenia aplikacji webowych i mikroservisów.
- **Swagger (OpenAPI)** - Narzędzie do generowania i testowania dokumentacji REST API w sposób interaktywny.
- **OIDC** - Protokół uwierzytelniania oparty na OAuth 2.0 – pozwala na bezpieczne logowanie użytkowników.
- **Keycloak** - Open-source'owy serwer tożsamości z obsługą SSO, OIDC i integracją z LDAP.
- **PostgreSQL** - Zaawansowany relacyjny system baz danych, otwartoźródłowy, skalowalny i wydajny.
- **Stripe** - Platforma płatnicza do obsługi płatności online – łatwa integracja przez REST API.
- **Taskfile** - Lekki zamiennik Makefile – automatyzuje zadania developerskie w plikach YAML.
- **adminer** - system zarządzania bazą danych
- **lombok** - redukcja kodu typu boilerplate
- **docker-compose** - do jednolitego środowiska uruchomieniowego (baza danych, keycloak, stripe)
-

## 2 Implementacja

### 2.1 Opis głównych funkcjonalności aplikacji

- **Autoryzacja z wykorzystaniem OIDC** – aplikacja wykorzystuje protokół OpenID Connect (OIDC) do bezpiecznej autoryzacji i uwierzytelniania użytkowników. Dzięki temu możliwe jest logowanie przy użyciu zewnętrznych dostawców tożsamości (np. Google, Facebook), a także bezpieczne zarządzanie sesją użytkownika.
- **Płatności** – integracja z zewnętrznym providerem płatności (Stripe) umożliwia użytkownikom szybkie i bezpieczne opłacanie karnetów.
- **Zarządzanie siłownią** – kompleksowy panel administracyjny pozwala na:
  - zarządzanie karnetami (tworzenie, edycja),
  - zarządzanie pracownikami,
  - organizowanie sesji treningowych,
  - monitorowanie statystyk użytkowników (liczba odbytych treningów, aktywność, postępy),
  - obsługę zgłaszania konserwacji sal.

## 2.2 Prezentacja zrzutów ekranu (screeny) prezentujących działanie aplikacji.



Rysunek 1: Swagger UI z dokumentacją API.

GET /workout-sessions

Parameters

Try it out

Name	Description
page integer (query)	Zero-based page index (0..N) Default value : 0
size integer (query)	The size of the page to be returned Default value : 20
sort array<string> (query)	Sorting criteria in the format: property,(asc desc). Default sort order is ascending. Multiple sort criteria are supported.

Responses

Code	Description	Links
200	OK	No links

Media type: \*/\*

Controls Accept header.

Example Value | Schema

```
{
  "roles": [
    {
      "uid": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "roleName": "string"
    }
  ],
  "membership": {
    "uid": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "validFrom": "2025-06-19T19:02:35.392Z",
    "validUntil": "2025-06-19T19:02:35.392Z",
    "membershipType": {
      "uid": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "type": "string",
      "currency": {
        "currencyCode": "string",
        "displayName": "string",
        "symbol": "string",
        "defaultFractionDigits": 1073741824,
        "numericCode": 1073741824,
        "numericCodeAsString": "string"
      },
      "price": 0
    }
  },
  "firstName": "string",
  "lastName": "string"
}
```

Rysunek 2: Swagger UI z endpointem GET.

POST /maintenance-tasks

Parameters

Try it out

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "maintainerUid": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "hallUid": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "plannedStartDate": "2025-06-19T19:03:22.606Z",
  "plannedEndDate": "2025-06-19T19:03:22.606Z",
  "description": "string"
}
```

Responses

Code	Description	Links
200	OK	No links

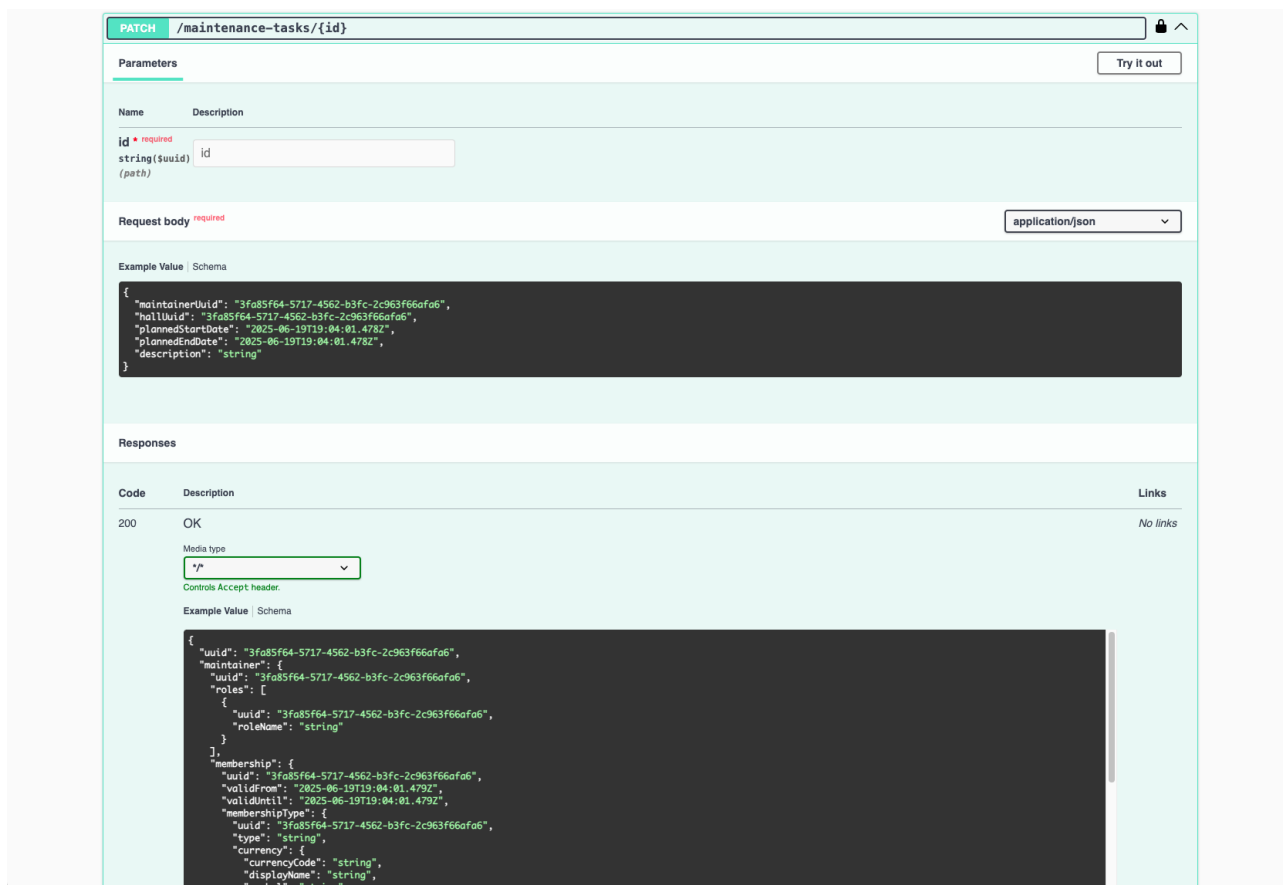
Media type: \*/\*

Controls Accept header.

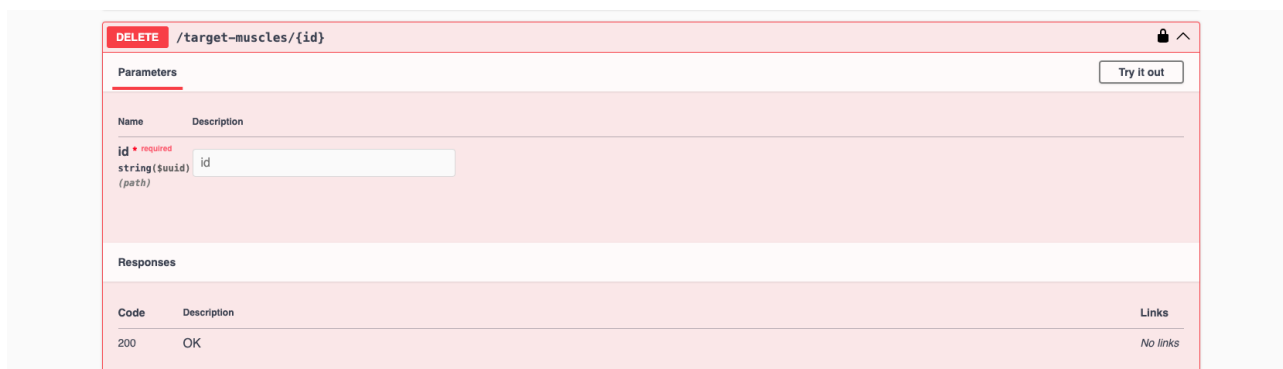
Example Value | Schema

```
{
  "uid": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "maintainer": {
    "uid": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "roles": [
      {
        "uid": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
        "roleName": "string"
      }
    ]
  },
  "membership": {
    "uid": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "validFrom": "2025-06-19T19:03:22.607Z",
    "validUntil": "2025-06-19T19:03:22.607Z",
    "membershipType": {
      "uid": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "type": "string",
      "currency": {
        "currencyCode": "string",
        "displayName": "string",
        "symbol": "string",
        "defaultFractionDigits": 1073741824,
        "numericCode": 1073741824,
        "numericCodeAsString": "string"
      }
    }
  }
}
```

Rysunek 3: Swagger UI z endpointem POST.



Rysunek 4: Swagger UI z endpointem PATCH.



Rysunek 5: Swagger UI z endpointem DELETE.

## 2.3 Wybrane fragmenty kodu z kluczowymi funkcjonalnościami

### 2.3.1 Autoryzacja z tokena JWT

```
1 package xyz.cursedman.gym_api.security;
2
3 import lombok.RequiredArgsConstructor;
4 import org.springframework.core.convert.converter.Converter;
5 import org.springframework.security.authentication.AbstractAuthenticationToken;
6 import org.springframework.security.core.GrantedAuthority;
7 import org.springframework.security.core.authority.SimpleGrantedAuthority;
8 import org.springframework.security.oauth2.jwt.Jwt;
9 import org.springframework.security.oauth2.server.resource.authentication.JwtAuthenticationToken;
10 import org.springframework.stereotype.Component;
11 import xyz.cursedman.gym_api.domain.dtos.user.UserDto;
12 import xyz.cursedman.gym_api.domain.dtos.user.UserRequest;
13 import xyz.cursedman.gym_api.services.UserService;
14
15 import java.util.*;
16 import java.util.stream.Collectors;
17
18 @Component
19 @RequiredArgsConstructor
20 public class GymJwtAuthenticationConverter implements Converter<Jwt,
21     ↳ AbstractAuthenticationToken> {
22     private final UserService userService;
23     private final String externalAuthProvider = "OIDC";
24     private final String userRolesClaimName = "roles";
25
26     private UserRequest jwtToUserRequest(Jwt jwt) {
27         UserRequest userRequest = new UserRequest();
28         userRequest.setFirstName(jwt.getClaim("name"));
29         userRequest.setLastName(jwt.getClaim("family_name"));
30         userRequest.setEmail(jwt.getClaim("email"));
31         userRequest.setUsername(
32             jwt.getClaim("preferred_username")
33         );
34
35         Set<String> userRoleNames = new HashSet<>(
36             // read roles claim
37             Optional.ofNullable(
38                 jwt.getClaimAsStringList(userRolesClaimName)
39             )
40             // if roles claim doesn't set no roles
41             .orElse(new ArrayList<>())
42         );
43         userRequest.setRoles(userRoleNames);
44
45         return userRequest;
46     }
47
48     @Override
49     public AbstractAuthenticationToken convert(Jwt jwt) {
50         UUID keycloakId = UUID.fromString(jwt.getSubject());
51
52         UserDto user = userService.createOrUpdateLinkedUser(
53             jwtToUserRequest(jwt), externalAuthProvider, keycloakId.toString()
54         );
55
56         Collection<String> roles = userService.getUserRoles(user.getUuid());
```

```

57         Collection<GrantedAuthority> authorities
58             = roles.stream()
59                 .map(SimpleGrantedAuthority::new)
60                 .collect(Collectors.toList());
61
62         return new JwtAuthenticationToken(jwt, authorities,
63             ↪ user.getUuid().toString());
64     }

```

Listing 1: Autoryzacja z tokena JWT (OIDC).

```

1  package xyz.cursedman.gym_api.controllers;
2
3  import jakarta.validation.Valid;
4  import lombok.RequiredArgsConstructor;
5  import org.springdoc.core.annotations.ParameterObject;
6  import org.springframework.data.domain.Page;
7  import org.springframework.data.domain.Pageable;
8  import org.springframework.data.web.config.EnableSpringDataWebSupport;
9  import org.springframework.http.HttpStatus;
10 import org.springframework.http.ResponseEntity;
11 import org.springframework.web.bind.annotation.*;
12 import xyz.cursedman.gym_api.domain.dtos.hall.HallDto;
13 import xyz.cursedman.gym_api.domain.dtos.hall.HallRequest;
14 import xyz.cursedman.gym_api.services.HallService;
15
16 import java.util.UUID;
17
18 @RestController
19 @RequestMapping(path = "/halls")
20 @RequiredArgsConstructor
21 @EnableSpringDataWebSupport
22 public class HallController {
23     private final HallService hallService;
24
25     @GetMapping
26     public ResponseEntity<Page<HallDto>> listHalls(@ParameterObject Pageable pageable) {
27         return ResponseEntity.ok(hallService.listHalls(pageable));
28     }
29
30     @GetMapping(path =("/{id}")
31     public ResponseEntity<HallDto> getHall(@Valid @PathVariable UUID id) {
32         HallDto hallDto = hallService.getHall(id);
33         return ResponseEntity.ok(hallDto);
34     }
35
36     @PostMapping
37     public ResponseEntity<HallDto> createHall(@Valid @RequestBody HallRequest request) {
38         HallDto createdHall = hallService.createHall(request);
39         return ResponseEntity.status(HttpStatus.CREATED).body(createdHall);
40     }
41
42     @PatchMapping(path =("/{id}")
43     public ResponseEntity<HallDto> updateHall(
44         @Valid @PathVariable UUID id,
45         @RequestBody HallRequest request
46     ) {
47         return ResponseEntity.ok(hallService.patchHall(id, request));
48     }
49 }

```



Listing 2: Przykładowy kontroler obsługujący sale treningowe.

```
1 package xyz.cursedman.gym_api.services.impl;
2
3 import lombok.AllArgsConstructor;
4 import org.springframework.data.domain.Page;
5 import org.springframework.data.domain.Pageable;
6 import org.springframework.stereotype.Service;
7 import xyz.cursedman.gym_api.domain.dtos.hall.HallDto;
8 import xyz.cursedman.gym_api.domain.dtos.hall.HallRequest;
9 import xyz.cursedman.gym_api.domain.entities.Hall;
10 import xyz.cursedman.gym_api.exceptions.NotFoundException;
11 import xyz.cursedman.gym_api.mappers.HallMapper;
12 import xyz.cursedman.gym_api.repositories.HallRepository;
13 import xyz.cursedman.gym_api.services.HallService;
14
15 import java.util.UUID;
16
17 @Service
18 @AllArgsConstructor
19 public class HallServiceImpl implements HallService {
20
21     private final HallRepository hallRepository;
22
23     private final HallMapper hallMapper;
24
25     @Override
26     public Page<HallDto> listHalls(Pageable pageable) {
27         return hallRepository.findAll(pageable)
28             .map(hallMapper::toDtoFromEntity);
29     }
30
31     @Override
32     public HallDto getHall(UUID id) {
33         return hallRepository.findById(id).map(hallMapper::toDtoFromEntity).orElseThrow(
34             ↪ hrow(NotFoundException::new);
35     }
36
37     @Override
38     public HallDto createHall(HallRequest request) {
39         Hall hall = hallMapper.toEntityFromRequest(request);
40         Hall result = hallRepository.save(hall);
41         return hallMapper.toDtoFromEntity(result);
42     }
43
44     @Override
45     public HallDto patchHall(UUID id, HallRequest request) {
46         Hall hall = hallRepository.findById(id).orElseThrow(NotFoundException::new);
47         hallMapper.updateFromRequest(request, hall);
48
49         Hall result = hallRepository.save(hall);
50         return hallMapper.toDtoFromEntity(result);
51     }
52
53     @Override
54     public Hall findHallByUuid(UUID id) {
55         if (id == null) return null;
56         return hallRepository.findById(id).orElse(null);
57     }
58 }
```

Listing 3: Przykładowy serwis obsługujący sale treningowe.

```

1  package xyz.cursedman.gym_api.controllers;
2
3  import org.hamcrest.Matchers;
4  import org.junit.jupiter.api.Test;
5  import org.junit.jupiter.api.extension.ExtendWith;
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
8  import org.springframework.boot.test.context.SpringBootTest;
9  import org.springframework.http.MediaType;
10 import org.springframework.security.test.context.support.WithMockUser;
11 import org.springframework.test.annotation.DirtiesContext;
12 import org.springframework.test.context.ActiveProfiles;
13 import org.springframework.test.context.junit.jupiter.SpringExtension;
14 import org.springframework.test.web.servlet.MockMvc;
15 import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
16 import org.springframework.test.web.servlet.result.MockMvcResultMatchers;
17 import xyz.cursedman.gym_api.domain.dtos.hall.HallRequest;
18 import xyz.cursedman.gym_api.helpers.TestJsonHelper;
19
20 import java.util.UUID;
21
22 @SpringBootTest
23 @ExtendWith(SpringExtension.class)
24 @AutoConfigureMockMvc
25 @ActiveProfiles("test")
26 @WithMockUser(roles = {"MANAGER", "EMPLOYEE"})
27 @DirtiesContext(classMode = DirtiesContext.ClassMode.AFTER_EACH_TEST_METHOD)
28 class HallControllerTest {
29     private final String endpointUri = "/halls";
30     private final String validHallUuid = "ce5f8d01-6fa8-4226-97fc-51d3e9cd91e5";
31     private final String validHallTypeUuid = "2a2fa2ba-2381-4cb6-86f4-282bdbf18e81";
32     private final HallRequest validHallRequest = HallRequest.builder()
33         .hallName("hall name")
34         .hallDescription("hall desc")
35         .hallTypeUuid(UUID.fromString(validHallTypeUuid))
36         .build();
37
38     @Autowired
39     private MockMvc mockMvc;
40
41     // GET
42
43     @Test
44     void checkIfGetReturnsHttp200AndAllRecords() throws Exception {
45         mockMvc.perform(MockMvcRequestBuilders.get(endpointUri))
46             .andExpect(MockMvcResultMatchers.status().isOk())
47             .andExpect(MockMvcResultMatchers.jsonPath("$.length()",
48                 ↪ Matchers.greaterThan(0)));
49     }
50
51     @Test
52     void checkIfGetByIdReturnsHttp200AndRequestedRecord() throws Exception {
53         mockMvc.perform(MockMvcRequestBuilders.get(endpointUri + "/" +
54             ↪ validHallUuid))
55             .andExpect(MockMvcResultMatchers.status().isOk())
56             .andExpect(MockMvcResultMatchers.jsonPath("$").exists());
57     }
58
59     @Test

```

```

57 void checkIfGetNonExistingRecordReturns404() throws Exception {
58     mockMvc.perform(MockMvcRequestBuilders.get(endpointUri + "/" +
59         ↳ UUID.randomUUID()))
60         .andExpect(MockMvcResultMatchers.status().isNotFound());
61 }
62 // POST
63
64 @Test
65 void checkIfCreateReturnsHttp201AndCreatedRecord() throws Exception {
66     mockMvc.perform(
67         MockMvcRequestBuilders.post(endpointUri)
68             .contentType(MediaType.APPLICATION_JSON)
69             .content(TestJsonHelper.stringify(validHallRequest))
70     ).andExpect(MockMvcResultMatchers.status().isCreated())
71     .andExpect(TestJsonHelper.contentEqualsJsonOf(validHallRequest,
72         ↳ "hallTypeUuid"))
73     .andExpect(
74         MockMvcResultMatchers.jsonPath("$.hallType.uuid",
75         ↳ Matchers.is(validHallTypeUuid))
76     );
77 }
78
79 @Test
80 void checkIfInvalidCreateBodyReturnsHttp400() throws Exception {
81     mockMvc.perform(
82         MockMvcRequestBuilders.post(endpointUri)
83             .contentType(MediaType.APPLICATION_JSON)
84             .content("{}")
85     ).andExpect(MockMvcResultMatchers.status().isBadRequest());
86 }
87
88 // PATCH
89
90 @Test
91 void checkIfPatchUpdateReturnsHttp200AndUpdatedRecord() throws Exception {
92     String hallTypeUuidToUpdate = "cc5f2a2a-1248-4e4f-aed9-5aab7c3f577a";
93     mockMvc.perform(
94         MockMvcRequestBuilders.patch(endpointUri + "/" +
95         ↳ validHallUuid)
96         .contentType(MediaType.APPLICATION_JSON)
97         .content(TestJsonHelper.toJSONField("hallTypeUuid",
98         ↳ hallTypeUuidToUpdate))
99     ).andExpect(MockMvcResultMatchers.status().isOk())
100     .andExpect(
101         MockMvcResultMatchers.jsonPath("$.hallType.uuid",
102         ↳ Matchers.is(hallTypeUuidToUpdate))
103     );
104 }
105
106 @Test
107 void checkIfPatchUpdateOfNonExistingRecordReturnsHttp404() throws Exception {
108     mockMvc.perform(
109         MockMvcRequestBuilders.patch(endpointUri + "/" + UUID.randomUUID())
110             .contentType(MediaType.APPLICATION_JSON)
111             .content(TestJsonHelper.stringify(validHallRequest))
112     ).andExpect(MockMvcResultMatchers.status().isNotFound());
113 }

```

Listing 4: Testy dotyczące sal treningowych

## 3 Testy

### 3.1 Opis metod testowania (np. testy manualne i automatyczne)

W projekcie zastosowano automatyczne testy integracyjne warstwy kontrolerów w aplikacji Spring Boot. Testy uruchamiane są na pełnym kontekście aplikacji, co pozwala na weryfikację poprawności działania endpointów HTTP wraz z rzeczywistą logiką biznesową. W celu zapewnienia izolacji testów od zewnętrznych systemów, komponenty komunikujące się z usługami zewnętrznymi są mockowane, natomiast pozostałe serwisy odpowiedzialne za logikę biznesową działają rzeczywiście. Takie podejście umożliwia sprawdzenie zarówno poprawności odpowiedzi HTTP, walidacji danych, jak i integracji kontrolerów z warstwą serwisów. Testy pozwalają na wykrywanie błędów na poziomie integracji, jednocześnie gwarantując stabilność i powtarzalność testów poprzez eliminację zależności od zewnętrznych systemów.

### 3.2 Wyniki testów, napotkane błędy oraz zastosowane rozwiązania

Przeprowadzone testy integracyjne potwierdziły poprawność działania endpointów (zob. 6). Testy typu GET na endpointach takich jak `/hall-types`, `/memberships`, `/membership-types` oraz `/workout-sessions` zwracały kod odpowiedzi 200 oraz listy rekordów, co świadczy o poprawnej implementacji operacji pobierania danych. Testy typu POST, takie jak tworzenie nowych sesji treningowych czy członkostw, zwracały kod 201, potwierdzając sukces operacji tworzenia rekordów. Operacje PATCH, np. aktualizacja danych członkostwa, również działały poprawnie – serwer zwracał kod 200 oraz zaktualizowane dane. Testy typu DELETE, np. usuwanie uczestników lub ćwiczeń z sesji treningowych, kończyły się kodem 204, a następnie testy GET potwierdzały, że rekordy zostały skutecznie usunięte. Testy scenariuszy negatywnych, takich jak próba pobrania nieistniejących zasobów lub wysłanie niepoprawnych danych, skutkowały odpowiednimi kodami błędów: 404 oraz 400, co było zgodne z założeniami. Ogólnie testy wykazały wysoką stabilność API. W przypadku wystąpienia jakichkolwiek krytycznych błędów, system CI nie generowałby artefaktów, a testy nie przechodziłyby poprawnie – co nie miało miejsca.

✓ gym_api (xyz.cursedman)	1 sec 954 ms
✓ MembershipTypeControllerTest	355 ms
✓ checkIfGetByldReturnsHttp200AndRequestedRecord()	250 ms
✓ checkIfInvalidCreateBodyReturnsHttp400()	42 ms
✓ checkIfCreateReturnsHttp201AndCreatedRecord()	30 ms
✓ checkIfGetNonExistingRecordReturns404()	6 ms
✓ checkIfGetReturnsHttp200AndAllRecords()	20 ms
✓ checkIfPatchUpdateOfNonExistingRecordReturnsHttp404()	7 ms
✓ OpenApiSpecGeneratorTest	444 ms
✓ generateOpenApiSpecJson()	410 ms
✓ generateOpenApiSpecYaml()	34 ms
> ✓ HallTypeControllerTest	8 ms
> ✓ TargetMuscleControllerTest	8 ms
> ✓ UserRoleControllerTest	8 ms
✓ WorkoutSessionControllerTest	679 ms
✓ checkIfGetByldReturnsHttp200AndRequestedRecord()	36 ms
✓ checkIfDeletingNonExistingExerciseReturnsHttp404()	120 ms
✓ checkIfDeletingExerciseUpdatesRecordAndReturnsHttp204()	34 ms
✓ checkIfAddingNonExistingExerciseReturnsHttp404()	37 ms
✓ checkIfDeletingNonExistingAttendantReturnsHttp404()	36 ms
✓ checkIfInvalidExerciseBodyReturnsHttp400()	15 ms
✓ checkIfAddingAttendantToNonExistingWorkoutSessionReturnsHttp404()	11 ms
✓ checkIfAddingNonExistingAttendantReturnsHttp404()	39 ms
✓ checkIfAddingExerciseToNonExistingWorkoutSessionReturnsHttp404()	33 ms
✓ checkIfDeletingAttendantOfNonExistingWorkoutSessionReturnsHttp404()	13 ms
✓ checkIfInvalidCreateBodyReturnsHttp400()	16 ms
✓ checkIfCreateReturnsHttp201AndCreatedRecord()	35 ms
✓ checkIfGetNonExistingRecordReturns404()	20 ms
✓ checkIfPatchUpdateReturnsHttp200AndUpdatedRecord()	27 ms
✓ checkIfGetReturnsHttp200AndAllRecords()	42 ms
✓ checkIfAddingExerciseReturnsHttp200AndUpdatedRecord()	39 ms
✓ checkIfInvalidAttendantBodyReturnsHttp400()	11 ms
✓ checkIfDeletingAttendantUpdatesRecordAndReturnsHttp204()	34 ms
✓ checkIfPatchUpdateOfNonExistingRecordReturnsHttp404()	18 ms
✓ checkIfDeletingExerciseOfNonExistingWorkoutSessionReturnsHttpNotFound()	34 ms
✓ checkIfAddingAttendantReturnsHttp200AndUpdatedRecord()	30 ms

Rysunek 6: Wyniki testów integracyjnych

## 4 Podsumowanie

### 4.1 Wnioski z realizacji projektu

W ramach projektu udało się stworzyć działające i stabilne API dla aplikacji fitness. Zostało ono oparte na architekturze MVC i pozwala na zarządzanie sesjami treningowymi, ćwiczeniami, członkostwami użytkowników oraz obsługę płatności. Wszystkie operacje typu CRUD (GET, POST, PATCH, DELETE) zostały poprawnie zaimplementowane w odpowiednich kontrolerach. Dzięki podziałowi na modele, widoki i kontrolery, kod jest bardziej przejrzysty i łatwiejszy do rozwijania. Dodatkowo zastosowanie zasad SOLID pomogło w utrzymaniu porządku w kodzie i jego modularności. Wprowadzenie paginacji pozwala lepiej obsługiwać większe ilości danych.

### 4.2 Ocena osiągniętych rezultatów i refleksje na temat procesu implementacji

Zrealizowane API spełnia najważniejsze założenia projektu. W trakcie implementacji można było zauważyć, jak ważne są bezpieczne logowanie (*OAuth2*), możliwość sortowania danych i paginacja. Jednym z trudniejszych elementów było zapewnienie spójności danych pomiędzy powiązаныmi tabelami (np. ćwiczenia i uczestnicy). Problem udało się rozwiązać poprzez dobrze zaprojektowane struktury żądań i odpowiedzi oraz ich walidację.

### 4.3 Propozycje usprawnień lub dalszego rozwoju aplikacji

W kolejnych etapach rozwoju systemu warto rozważyć implementację funkcji śledzenia treningów w czasie rzeczywistym, co wiązałoby się z obsługą transmisji danych w trybie ciągłym oraz zarządzaniem stanem aktywnych sesji treningowych. Istotnym rozszerzeniem mogłoby być również dodanie systemu powiadomień, pozwalającego na generowanie i dystrybucję komunikatów w odpowiedzi na zdarzenia zachodzące w systemie, takie jak rozpoczęcie, zakończenie lub przekroczenie określonych parametrów treningu. Uzupełnieniem tych funkcjonalności byłoby wprowadzenie modułu rekomendacji treningów, który na podstawie danych wejściowych, takich jak cel treningowy czy wcześniejsze aktywności, generowałby dopasowane propozycje planów treningowych.

## 5 Podział pracy

- Rafał Grot – konfiguracja środowiska (Docker, Keycloak), wdrożenie procesu automatyzacji (CI), projekt bazy danych i encji, implementacja kontrolerów i serwisów, testy, logika biznesowa płatności za karnety, podstawowa konfiguracja Spring Security, dokumentacja.
- Filip Stępień – projekt bazy danych i encji, implementacja kontrolerów i serwisów, testy i przygotowanie danych testowych, implementacja płatności Stripe, dokumentacja.
- Bartosz Karkoszka – rozszerzenie Spring Security na role użytkowników, dokumentacja.

<https://github.com/rafal11ck/gym-api>

## 6 Literatura

- OpenID Connect – dokumentacja oficjalna, [https://openid.net/specs/openid-connect-core-1\\_0.html](https://openid.net/specs/openid-connect-core-1_0.html), dostęp: 18.06.2025.
- Spring Boot – dokumentacja oficjalna, <https://spring.io/projects/spring-boot>, dostęp: 18.06.2025.
- Keycloak – dokumentacja oficjalna, <https://www.keycloak.org/documentation>, dostęp: 18.06.2025.
- NixOS – dokumentacja oficjalna, <https://nixos.org/>, dostęp: 18.06.2025.
- Docker – dokumentacja oficjalna, <https://www.docker.com/>, dostęp: 18.06.2025.