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, 21 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

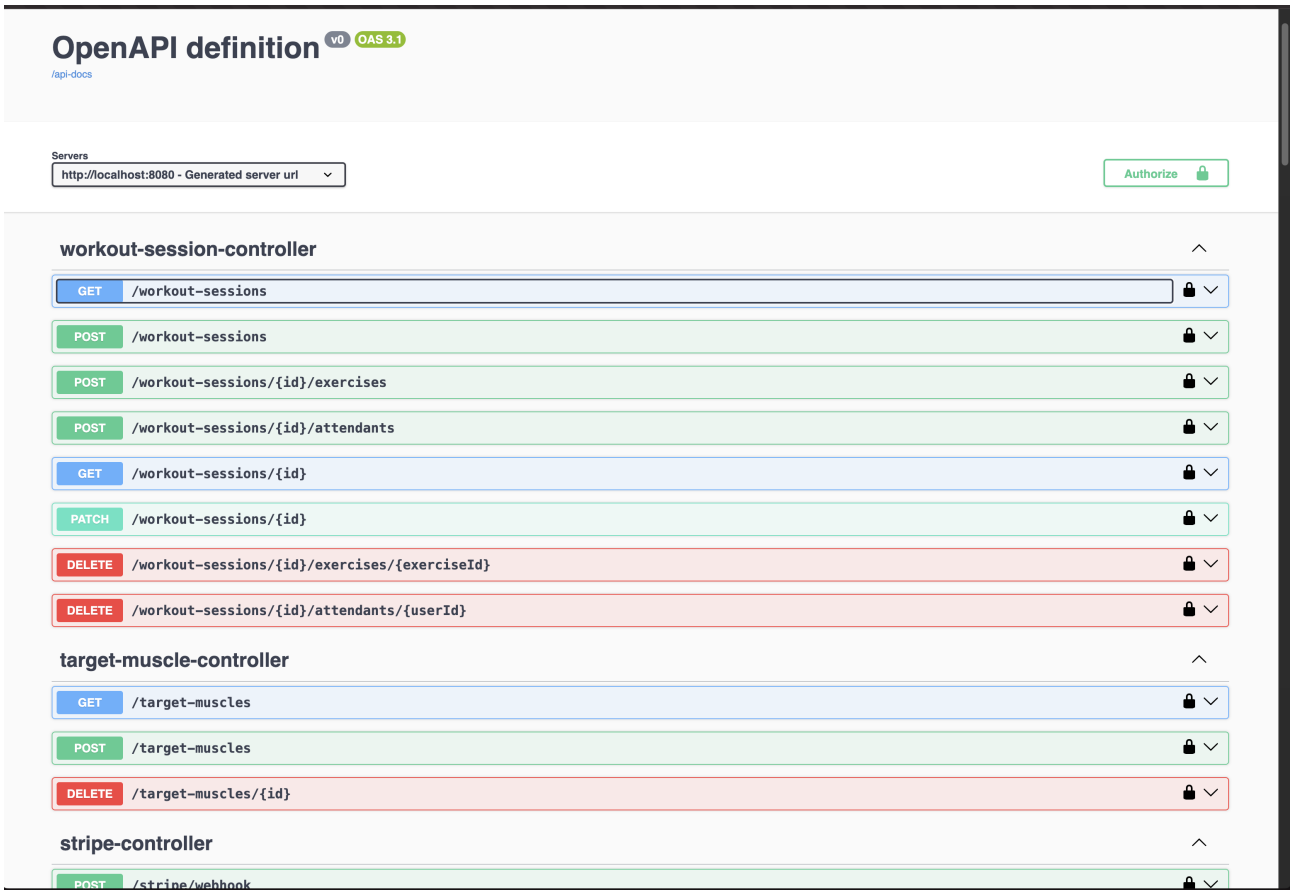
- **Nix** - System do zarządzania pakietami i środowiskami – pozwala tworzyć powtarzalne konfiguracje projektów.
- **git** - system kontroli wersji
- **GitHub Actions** - System CI/CD w GitHubie do automatyzacji testów, buildów i wdrożeń przy pomocy workflowów.
- **Spring Boot** - Framework w Javie do szybkiego tworzenia aplikacji webowych i mikroserwisów.
- **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 uruchominiowego (baza danych, keycloak, stripe)
-

2 Implementacja

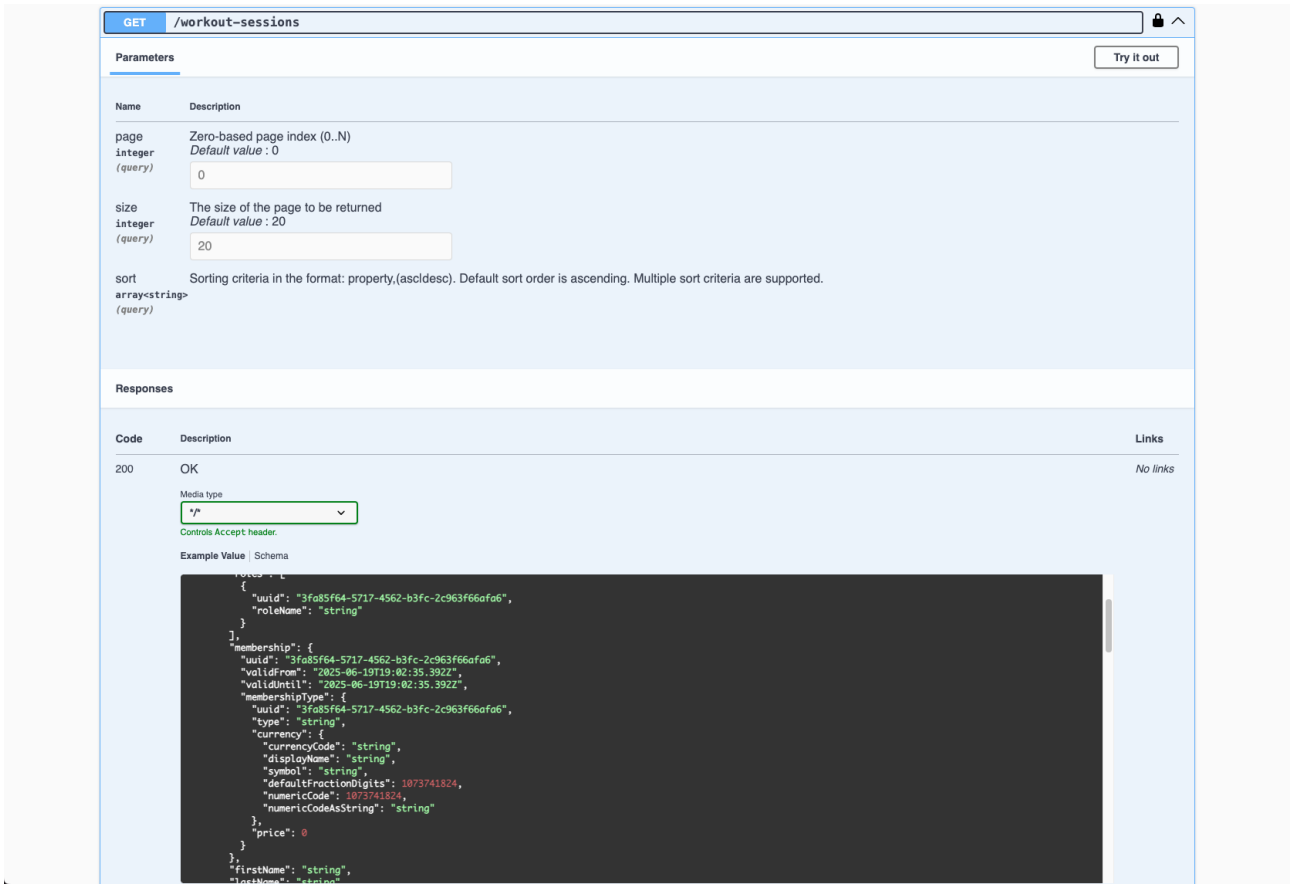
2.1 Opis głównych funkcjonalności aplikacji

- **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.

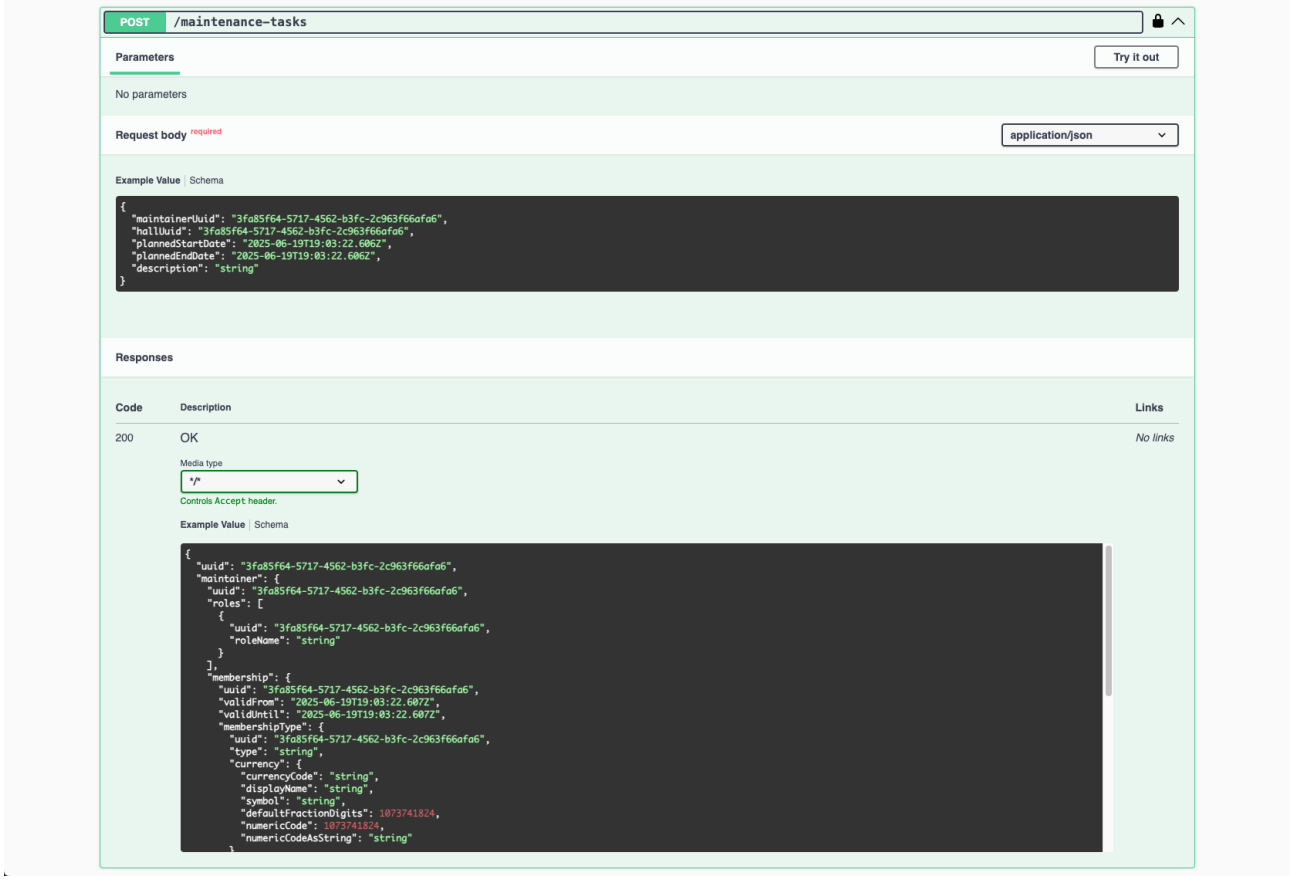
2.2 Prezentacja zrzutów ekranu (screeny) prezentujących działanie aplikacji.



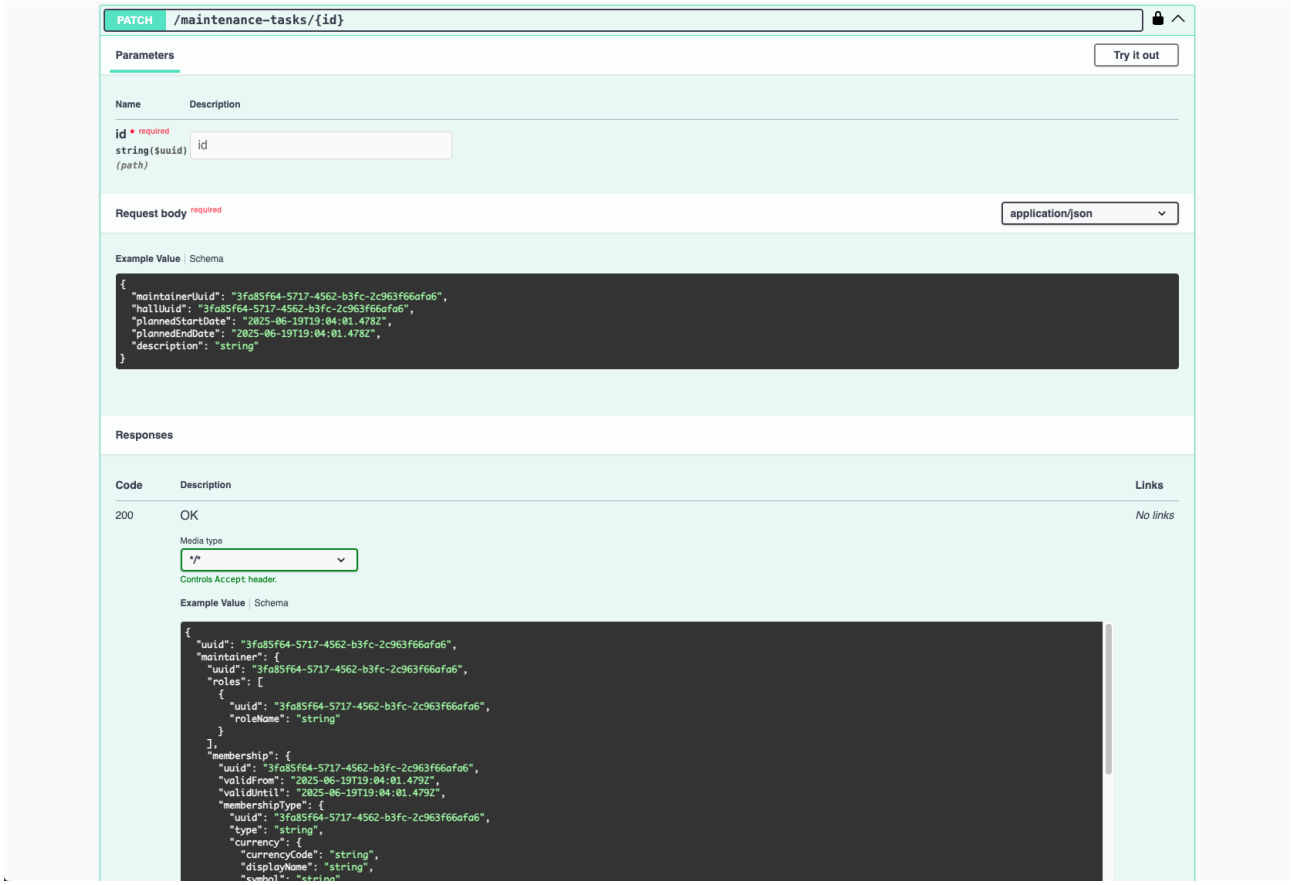
Rysunek 1: Swagger UI z dokumentacją API.



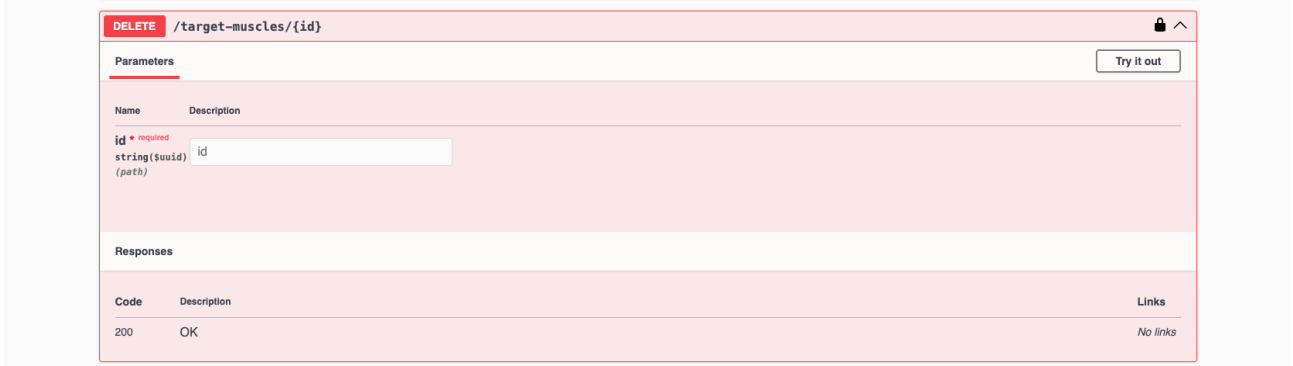
Rysunek 2: Swagger UI z endpointem GET.



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

Listing 1: Autoryzacja z tokena JWT (OIDC).

```java
package xyz.cursedman.gym_api.security;

import lombok.RequiredArgsConstructor;
import org.springframework.core.convert.converter.Converter;
import org.springframework.security.authentication.AbstractAuthenticationToken;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.oauth2.jwt.Jwt;
import org.springframework.security.oauth2.server.resource.authentication.JwtAuthenticationToken;
import org.springframework.stereotype.Component;
import xyz.cursedman.gym_api.domain.dtos.user.UserDto;
import xyz.cursedman.gym_api.domain.dtos.user.UserRequest;
import xyz.cursedman.gym_api.services.UserService;

import java.util.*;
import java.util.stream.Collectors;

@Component
@RequiredArgsConstructor
public class GymJwtAuthenticationConverter implements Converter<Jwt, AbstractAuthenticationToken> {
        private final UserService userService;
        private final String externalAuthProvider = "OIDC";
        private final String userRolesClaimName = "roles";


        private UserRequest jwtToUserRequest(Jwt jwt) {
                UserRequest userRequest = new UserRequest();
                userRequest.setFirstName(jwt.getClaim("name"));
                userRequest.setLastName(jwt.getClaim("family_name"));
                userRequest.setEmail(jwt.getClaim("email"));
                userRequest.setUsername(
                        jwt.getClaim("preferred_username")
                );

                Set<String> userRoleNames = new HashSet<>(
                        // read roles claim
                        Optional.ofNullable(
                                        jwt.getClaimAsStringList(userRolesClaimName)
                                )
                                // if roles claim doesn't set no roles
                                .orElse(new ArrayList<>())
                );
                userRequest.setRoles(userRoleNames);

                return userRequest;
        }

        @Override
        public AbstractAuthenticationToken convert(Jwt jwt) {
                UUID keycloakId = UUID.fromString(jwt.getSubject());

                UserDto user = userService.createOrUpdateLinkedUser(
                        jwtToUserRequest(jwt), externalAuthProvider, keycloakId.toString()
                );

                Collection<String> roles = userService.getUserRoles(user.getUuid());
                Collection<GrantedAuthority> authorities
                        = roles.stream()
                        .map(SimpleGrantedAuthority::new)
                        .collect(Collectors.toList());

                return new JwtAuthenticationToken(jwt, authorities, user.getUuid().toString());
        }
}
```

Listing 2: Przykładowy kontroler obsługujący sale treningowe.

```java
package xyz.cursedman.gym_api.controllers;

import jakarta.validation.Valid;
```

```java
import lombok.RequiredArgsConstructor;
import org.springdoc.core.annotations.ParameterObject;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.web.config.EnableSpringDataWebSupport;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import xyz.cursedman.gym_api.domain.dtos.hall.HallDto;
import xyz.cursedman.gym_api.domain.dtos.hall.HallRequest;
import xyz.cursedman.gym_api.services.HallService;

import java.util.UUID;

@RestController
@RequestMapping(path = "/halls")
@RequiredArgsConstructor
@EnableSpringDataWebSupport
public class HallController {
        private final HallService hallService;

        @GetMapping
        public ResponseEntity<Page<HallDto>> listHalls(@ParameterObject Pageable pageable) {
                return ResponseEntity.ok(hallService.listHalls(pageable));
        }

        @GetMapping(path = "/{id}")
        public ResponseEntity<HallDto> getHall(@Valid @PathVariable UUID id) {
                HallDto hallDto = hallService.getHall(id);
                return ResponseEntity.ok(hallDto);
        }

        @PostMapping
        public ResponseEntity<HallDto> createHall(@Valid @RequestBody HallRequest request) {
                HallDto createdHall = hallService.createHall(request);
                return ResponseEntity.status(HttpStatus.CREATED).body(createdHall);
        }

        @PatchMapping(path = "/{id}")
        public ResponseEntity<HallDto> updateHall(
                @Valid @PathVariable UUID id,
                @RequestBody HallRequest request
        ) {
                return ResponseEntity.ok(hallService.patchHall(id, request));
        }
}
```

Listing 3: Przykładowy serwis obsługujący sale treningowe.

```java
package xyz.cursedman.gym_api.services.impl;

import lombok.AllArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;
import xyz.cursedman.gym_api.domain.dtos.hall.HallDto;
import xyz.cursedman.gym_api.domain.dtos.hall.HallRequest;
import xyz.cursedman.gym_api.domain.entities.Hall;
import xyz.cursedman.gym_api.exceptions.NotFoundException;
import xyz.cursedman.gym_api.mappers.HallMapper;
import xyz.cursedman.gym_api.repositories.HallRepository;
import xyz.cursedman.gym_api.services.HallService;

import java.util.UUID;

@Service
@AllArgsConstructor
public class HallServiceImpl implements HallService {

        private final HallRepository hallRepository;

        private final HallMapper hallMapper;

        @Override
        public Page<HallDto> listHalls(Pageable pageable) {
                return hallRepository.findAll(pageable)
                        .map(hallMapper::toDtoFromEntity);
        }
```

```
30
31          @Override
32          public HallDto getHall(UUID id) {
33                  return hallRepository.findById(id).map(hallMapper::toDtoFromEntity).orElseThrow(NotFoundException
                    ↪  ::new);
34          }
35
36          @Override
37          public HallDto createHall(HallRequest request) {
38                  Hall hall = hallMapper.toEntityFromRequest(request);
39                  Hall result = hallRepository.save(hall);
40                  return hallMapper.toDtoFromEntity(result);
41          }
42
43          @Override
44          public HallDto patchHall(UUID id, HallRequest request) {
45                  Hall hall = hallRepository.findById(id).orElseThrow(NotFoundException::new);
46                  hallMapper.updateFromRequest(request, hall);
47
48                  Hall result = hallRepository.save(hall);
49                  return hallMapper.toDtoFromEntity(result);
50          }
51
52          @Override
53          public Hall findHallByUuid(UUID id) {
54                  if (id == null) return null;
55                  return hallRepository.findById(id).orElse(null);
56          }
57  }
```

Listing 4: Testy dotyczace sal treninigowych

```
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          @Autowired
38          private MockMvc mockMvc;
39
40          // GET
41
42          @Test
43          void checkIfGetReturnsHttp200AndAllRecords() throws Exception {
44                  mockMvc.perform(MockMvcRequestBuilders.get(endpointUri))
45                          .andExpect(MockMvcResultMatchers.status().isOk())
```

```
46                      .andExpect(MockMvcResultMatchers.jsonPath("$.length()", Matchers.greaterThan(0)));
47          }
48
49          @Test
50          void checkIfGetByIdReturnsHttp200AndRequestedRecord() throws Exception {
51                  mockMvc.perform(MockMvcRequestBuilders.get(endpointUri + "/" + validHallUuid))
52                          .andExpect(MockMvcResultMatchers.status().isOk())
53                          .andExpect(MockMvcResultMatchers.jsonPath("$").exists());
54          }
55
56          @Test
57          void checkIfGetNonExistingRecordReturns404() throws Exception {
58                  mockMvc.perform(MockMvcRequestBuilders.get(endpointUri + "/" + UUID.randomUUID()))
59                          .andExpect(MockMvcResultMatchers.status().isNotFound());
60          }
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, "hallTypeUuid"))
72                          .andExpect(
73                                  MockMvcResultMatchers.jsonPath("$.hallType.uuid", Matchers.is(validHallTypeUuid))
74                          );
75          }
76
77          @Test
78          void checkIfInvalidCreateBodyReturnsHttp400() throws Exception {
79                  mockMvc.perform(
80                          MockMvcRequestBuilders.post(endpointUri)
81                                  .contentType(MediaType.APPLICATION_JSON)
82                                  .content("{}")
83                  ).andExpect(MockMvcResultMatchers.status().isBadRequest());
84          }
85
86          // PATCH
87
88          @Test
89          void checkIfPatchUpdateReturnsHttp200AndUpdatedRecord() throws Exception {
90                  String hallTypeUuidToUpdate = "cc5f2a2a-1248-4e4f-aed9-5aab7c3f577a";
91                  mockMvc.perform(
92                                  MockMvcRequestBuilders.patch(endpointUri + "/" + validHallUuid)
93                                          .contentType(MediaType.APPLICATION_JSON)
94                                          .content(TestJsonHelper.toJSONField("hallTypeUuid", hallTypeUuidToUpdate))
95                          ).andExpect(MockMvcResultMatchers.status().isOk())
96                          .andExpect(
97                                  MockMvcResultMatchers.jsonPath("$.hallType.uuid",
                                    ↪  Matchers.is(hallTypeUuidToUpdate))
98                          );
99          }
100
101         @Test
102         void checkIfPatchUpdateOfNonExistingRecordReturnsHttp404() throws Exception {
103                 mockMvc.perform(
104                         MockMvcRequestBuilders.patch(endpointUri + "/" + UUID.randomUUID())
105                                 .contentType(MediaType.APPLICATION_JSON)
106                                 .content(TestJsonHelper.stringify(validHallRequest))
107                 ).andExpect(MockMvcResultMatchers.status().isNotFound());
108         }
109 }
```

3 Testy

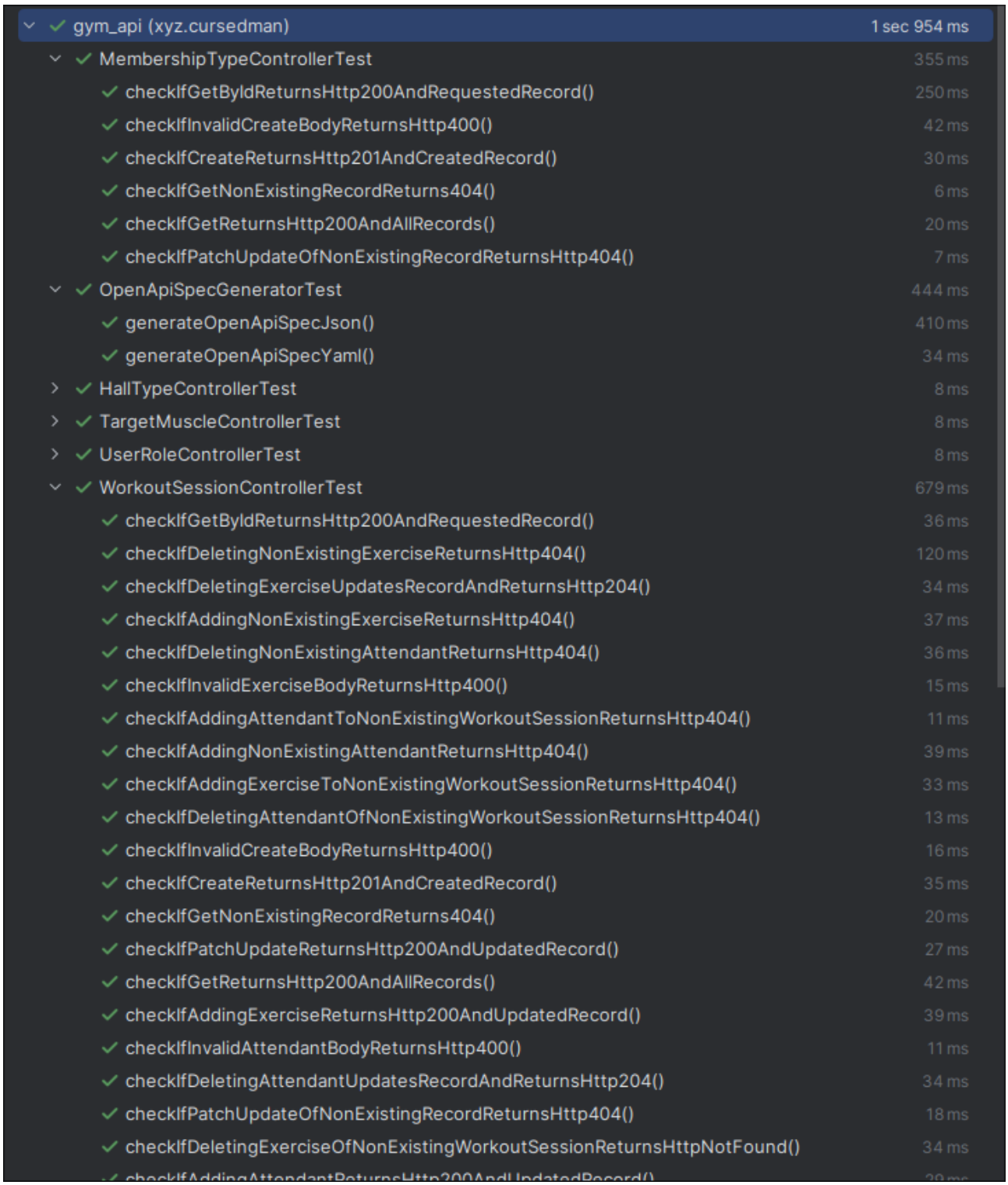
3.1 Opis metod testowania (np. testy manualne i automatyczne)

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.



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 powiązanymi 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.
- Bartłomiej 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, 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.