Politechnika Świętokrzyska w Kielcach

Wydział Elektrotechniki, Automatyki i Informatyki

Politechnika Świętokrzyska
Kielce University of Technology

Inzynieria Systemow 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, 19 czerwca 2025

1 Wprowadzenie/Cel laboratorium

1.1 Krótki opis aplikacji

1.2 Wykorzystane technologie oraz narzędzia

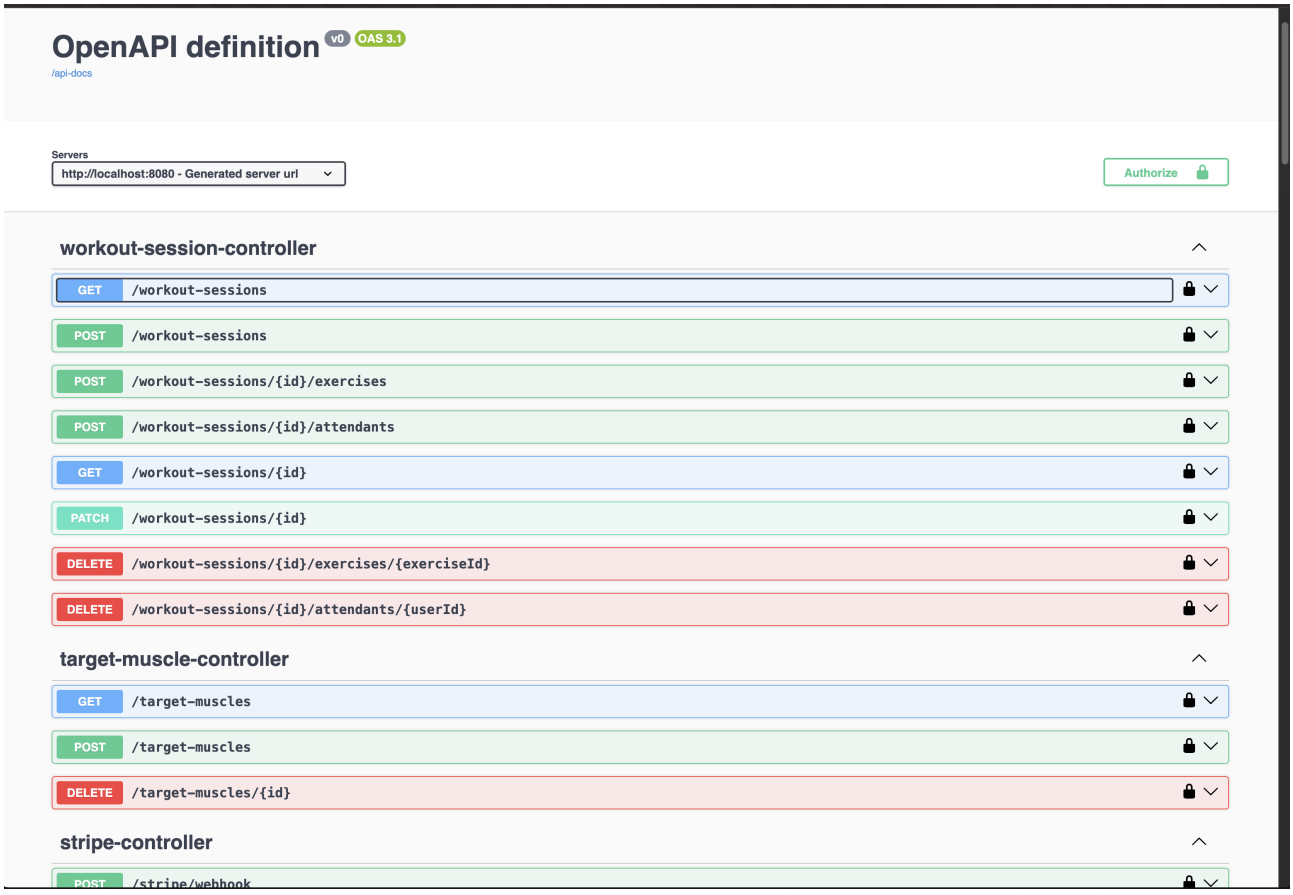
- nix
- gitub actions
- spring boot
- swagger
- OIDC
- keycloak
- postgres
- stripe
- taskfile

2 Implementacja

2.1 Opis głównych funkcjonalności aplikacji

- autoryzacja z wykorzystaniem OIDC.
- Płatności.
- zarządzanie siłownnią

2.2 Prezentacja zrzutów ekranu (screeny) prezentujących działanie aplikacji



Rysunek 1: Swagger UI z dokumentacją API

2.3 Wybrane fragmenty kodu z kluczowymi funkcjonalnościami

2.3.1 Autoryzacja z tokena JWT

```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.JwtAuthentication
↪  Token;
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 1: Autoryzacja z tokena JWT (OIDC)

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

import jakarta.validation.Valid;
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 2: Przykładowy kontroler

```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);
        }

        @Override
        public HallDto getHall(UUID id) {
                return hallRepository.findById(id).map(hallMapper::toDtoFromEntity).orElseT
                ↪  hrow(NotFoundException::new);
        }

        @Override
        public HallDto createHall(HallRequest request) {
                Hall hall = hallMapper.toEntityFromRequest(request);
                Hall result = hallRepository.save(hall);
                return hallMapper.toDtoFromEntity(result);
        }

        @Override
        public HallDto patchHall(UUID id, HallRequest request) {
                Hall hall = hallRepository.findById(id).orElseThrow(NotFoundException::new);
                hallMapper.updateFromRequest(request, hall);

                Hall result = hallRepository.save(hall);
                return hallMapper.toDtoFromEntity(result);
        }

        @Override
        public Hall findHallByUuid(UUID id) {
                if (id == null) return null;
                return hallRepository.findById(id).orElse(null);
        }
}
```

Listing 3: Przykładowy serwis

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

import org.hamcrest.Matchers;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.security.test.context.support.WithMockUser;
import org.springframework.test.annotation.DirtiesContext;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;
import xyz.cursedman.gym_api.domain.dtos.hall.HallRequest;
import xyz.cursedman.gym_api.helpers.TestJsonHelper;

import java.util.UUID;

@SpringBootTest
@ExtendWith(SpringExtension.class)
@AutoConfigureMockMvc
@ActiveProfiles("test")
@WithMockUser(roles = {"MANAGER", "EMPLOYEE"})
@DirtiesContext(classMode = DirtiesContext.ClassMode.AFTER_EACH_TEST_METHOD)
class HallControllerTest {
        private final String endpointUri = "/halls";
        private final String validHallUuid = "ce5f8d01-6fa8-4226-97fc-51d3e9cd91e5";
        private final String validHallTypeUuid = "2a2fa2ba-2381-4cb6-86f4-282bdbf18e81";
        private final HallRequest validHallRequest = HallRequest.builder()
                .hallName("hall name")
                .hallDescription("hall desc")
                .hallTypeUuid(UUID.fromString(validHallTypeUuid))
                .build();
        @Autowired
        private MockMvc mockMvc;

        // GET

        @Test
        void checkIfGetReturnsHttp200AndAllRecords() throws Exception {
                mockMvc.perform(MockMvcRequestBuilders.get(endpointUri))
                        .andExpect(MockMvcResultMatchers.status().isOk())
                        .andExpect(MockMvcResultMatchers.jsonPath("$.length()",
                        ↪  Matchers.greaterThan(0)));
        }

        @Test
        void checkIfGetByIdReturnsHttp200AndRequestedRecord() throws Exception {
                mockMvc.perform(MockMvcRequestBuilders.get(endpointUri + "/" +
                ↪  validHallUuid))
                        .andExpect(MockMvcResultMatchers.status().isOk())
                        .andExpect(MockMvcResultMatchers.jsonPath("$").exists());
        }

        @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 }
```

Listing 4: Przykładowe testy jednego z kontrolerów

2.3.2

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.(fig. 2). Testy GET na endpointach np. /hall-types, /memberships, /membership-types i /workout-sessions zwracały kod HTTP 200 oraz listy rekordów, co świadczy o poprawnej implementacji pobierania danych. Testy POST, takie jak tworzenie nowych sesji treningowych czy członkostw, zwracały kod HTTP 201, potwierdzając sukces tworzenia rekordów. Operacje PATCH, np. aktualizacja członkostwa, również działały poprawnie, zwracając kod 200 i zaktualizowane dane. Testy DELETE, np. usuwanie uczestników lub ćwiczeń z sesji, kończyły się sukcesem z kodem 204, a następnie weryfikacja GET potwierdzała usunięcie rekordów. Testy negatywnych scenariuszy, takich jak pobieranie nieistniejących rekordów czy wysyłanie nieprawidłowych danych, zwracały odpowiednio kody 404 i 400, co jest zgodne z oczekiwaniami. Ogólnie, testy wykazały wysoką stabilność API, jeśli byłyby jakieś błędy, to CI nie wygenerowałby artefaktów, a testy nie przechodziłyby.

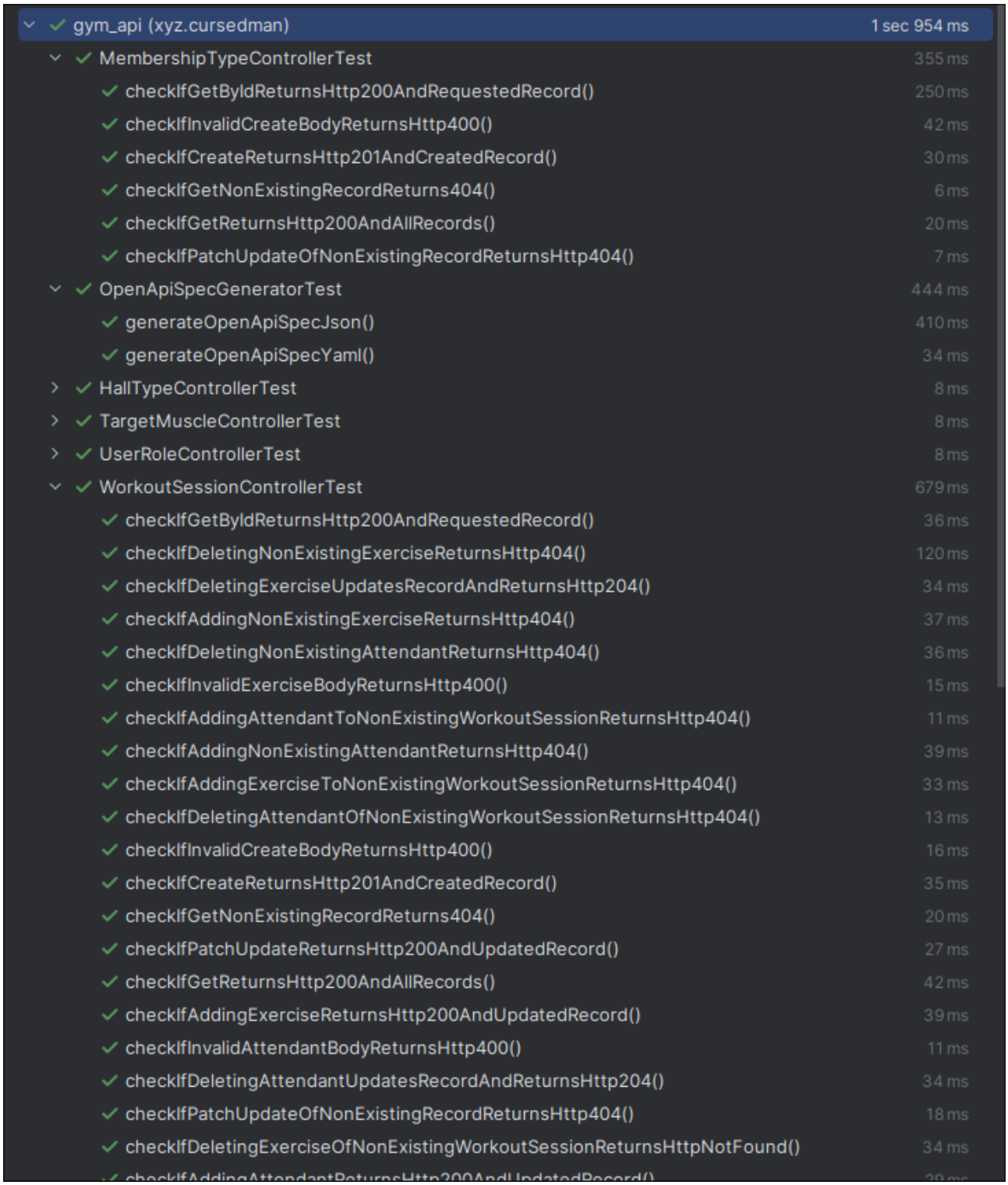
4 Podsumowanie

4.1 Wnioski z realizacji projektu

Możemy stwierdzić, że stworzono solidne API dla aplikacji fitness, oparte na architekturze MVC, umożliwiające zarządzanie sesjami treningowymi, ćwiczeniami, członkostwami użytkowników oraz przetwarzanie płatności. Zaimplementowanie operacji CRUD (GET, POST, PATCH, DELETE) w różnych kontrolerach w ramach wzorca MVC pokazuje kompleksowe rozwiązanie backendowe, gdzie modele obsługują logikę biznesową, widoki prezentują dane, a kontrolery zarządzają przepływem. Użycie zasad SOLID zapewniło modułowość i łatwość rozbudowy kodu. Ponadto zastosowanie paginacji zwiększa skalowalność i efektywną obsługę dużych ilości danych.

4.2 Ocena osiągniętych rezultatów i refleksje na temat procesu implementacji

Oceniając rezultaty, API spełnia podstawowe wymagania platformy fitness. Proces podkreślił znaczenie bezpiecznego uwierzytelniania (OAuth2) oraz elastycznych opcji sortowania i paginacji. Wyzwania obejmowały zapewnienie spójności danych między powiązanymi encjami (np. ćwiczenia i uczestnicy), co zostało rozwiązane dzięki uporządkowanym schematom żądań i odpowiedzi.



Rysunek 2: Wyniki testów integracyjnych

4.3 Propozycje usprawnień lub dalszego rozwoju aplikacji

Proponując usprawnienia, można rozważyć dodanie śledzenia treningów w czasie rzeczywistym.

5 Podział pracy

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

6 Literatura

- standard OIDC https://openid.net/specs/openid-connect-core-1_0.html
- dokumetacja springa https://spring.io/projects/spring-boot
- dokumetacja keycloak https://www.keycloak.org/documentation
- nixos https://nixos.org/
- docker https://www.docker.com/