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

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

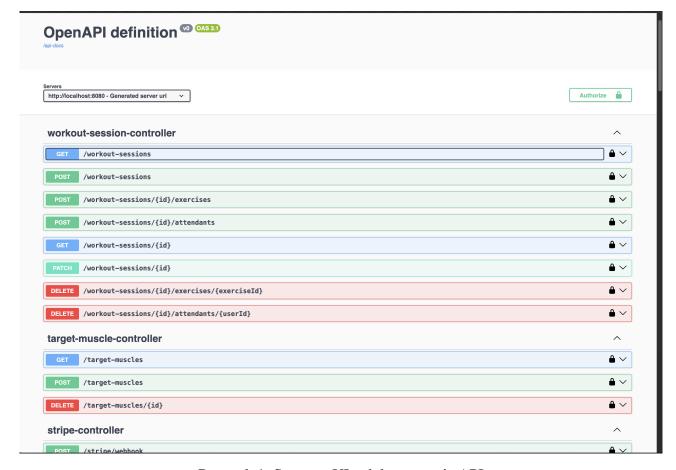
- **Nix** System do zarządzania pakietami i środowiskami pozwala tworzyć powtarzalne konfiguracje projektów.
- **GitHub Actions** System CI/CD w GitHubie do automatyzacji testów, buildów i wdrożeń przy pomocy workflowów.
- Spring Boot Framework w Javie do szybkiego tworzenia aplikacji webowych i mikroserwisów.
- **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.

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

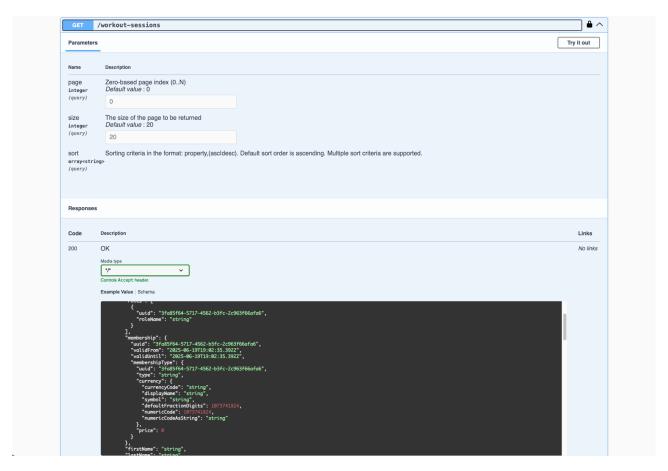
2.3 Wybrane fragmenty kodu z kluczowymi funkcjonalnościami

2.3.1 Autoryzacja z tokena JWT

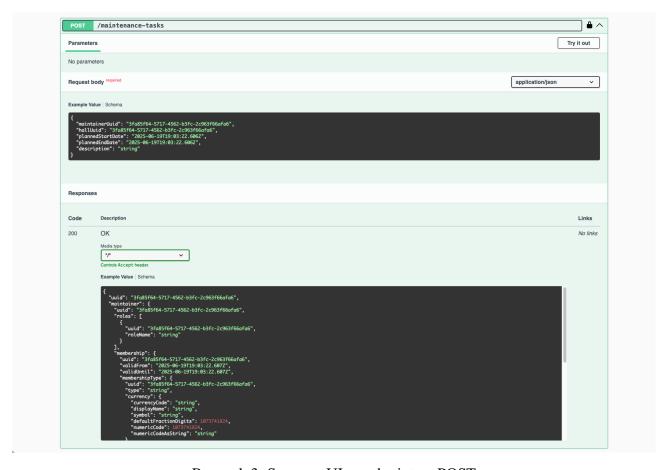
```
package xyz.cursedman.gym_api.security;
   import lombok.RequiredArgsConstructor;
   import org.springframework.core.convert.converter.Converter;
   import org.springframework.security.authentication.AbstractAuthenticationToken;
   import org.springframework.security.core.GrantedAuthority;
   import org.springframework.security.core.authority.SimpleGrantedAuthority;
   import org.springframework.security.oauth2.jwt.Jwt;
   import org.springframework.security.oauth2.server.resource.authentication.JwtAuthentication |

→ Token;

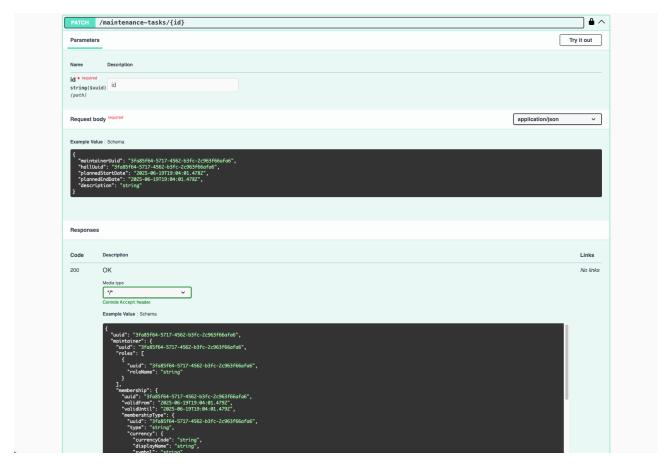
   import org.springframework.stereotype.Component;
10
   import xyz.cursedman.gym_api.domain.dtos.user.UserDto;
11
   import xyz.cursedman.gym_api.domain.dtos.user.UserRequest;
12
   import xyz.cursedman.gym_api.services.UserService;
13
   import java.util.*;
15
   import java.util.stream.Collectors;
16
17
   @Component
18
   @RequiredArgsConstructor
19
   public class GymJwtAuthenticationConverter implements Converter<Jwt,</pre>
20
    → AbstractAuthenticationToken> {
           private final UserService userService;
21
```



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

```
private final String externalAuthProvider = "OIDC";
22
            private final String userRolesClaimName = "roles";
23
24
25
            private UserRequest jwtToUserRequest(Jwt jwt) {
26
                    UserRequest userRequest = new UserRequest();
27
                    userRequest.setFirstName(jwt.getClaim("name"));
28
                    userRequest.setLastName(jwt.getClaim("family_name"));
29
                    userRequest.setEmail(jwt.getClaim("email"));
30
                    userRequest.setUsername(
31
                             jwt.getClaim("preferred_username")
32
                    );
33
34
                    Set<String> userRoleNames = new HashSet<>(
35
                             // read roles claim
36
                             Optional.ofNullable(
37
                                              jwt.getClaimAsStringList(userRolesClaimName)
38
39
                                     // if roles claim doesn't set no roles
40
                                     .orElse(new ArrayList<>())
41
                    );
42
                    userRequest.setRoles(userRoleNames);
43
44
                    return userRequest;
45
            }
46
47
            @Override
48
            public AbstractAuthenticationToken convert(Jwt jwt) {
49
                    UUID keycloakId = UUID.fromString(jwt.getSubject());
50
51
                    UserDto user = userService.createOrUpdateLinkedUser(
52
                             jwtToUserRequest(jwt), externalAuthProvider, keycloakId.toString()
53
                    );
54
55
                    Collection<String> roles = userService.getUserRoles(user.getUuid());
56
                    Collection<GrantedAuthority> authorities
57
                             = roles.stream()
58
                             .map(SimpleGrantedAuthority::new)
59
                             .collect(Collectors.toList());
60
61
                    return new JwtAuthenticationToken(jwt, authorities,
62

    user.getUuid().toString());
            }
   }
64
                              Listing 1: Autoryzacja z tokena JWT (OIDC)
   package xyz.cursedman.gym_api.controllers;
2
   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;
10
   import org.springframework.web.bind.annotation.*;
11
   import xyz.cursedman.gym_api.domain.dtos.hall.HallDto;
12
   import xyz.cursedman.gym_api.domain.dtos.hall.HallRequest;
   import xyz.cursedman.gym_api.services.HallService;
```

```
15
   import java.util.UUID;
16
17
   @RestController
18
   @RequestMapping(path = "/halls")
19
   @RequiredArgsConstructor
20
   @EnableSpringDataWebSupport
21
   public class HallController {
22
            private final HallService hallService;
23
24
            @GetMapping
25
            public ResponseEntity<Page<HallDto>> listHalls(@ParameterObject Pageable pageable) {
26
                    return ResponseEntity.ok(hallService.listHalls(pageable));
27
28
29
            @GetMapping(path = "/{id}")
30
            public ResponseEntity<HallDto> getHall(@Valid @PathVariable UUID id) {
31
                    HallDto hallDto = hallService.getHall(id);
32
                    return ResponseEntity.ok(hallDto);
33
            }
34
35
            @PostMapping
36
            public ResponseEntity<HallDto> createHall(@Valid @RequestBody HallRequest request) {
37
                    HallDto createdHall = hallService.createHall(request);
38
                    return ResponseEntity.status(HttpStatus.CREATED).body(createdHall);
39
            }
40
41
            @PatchMapping(path = "/{id}")
42
            public ResponseEntity<HallDto> updateHall(
43
                    @Valid @PathVariable UUID id,
44
                    @RequestBody HallRequest request
45
            ) {
                    return ResponseEntity.ok(hallService.patchHall(id, request));
47
            }
48
   }
49
                                    Listing 2: Przykładowy kontroler
   package xyz.cursedman.gym_api.services.impl;
1
2
   import lombok.AllArgsConstructor;
3
   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;
12
   import xyz.cursedman.gym_api.services.HallService;
13
14
   import java.util.UUID;
15
16
   @Service
17
   @AllArgsConstructor
18
   public class HallServiceImpl implements HallService {
19
20
            private final HallRepository hallRepository;
21
22
23
            private final HallMapper hallMapper;
```

```
24
            @Override
25
            public Page<HallDto> listHalls(Pageable pageable) {
26
                    return hallRepository.findAll(pageable)
27
                             .map(hallMapper::toDtoFromEntity);
28
            }
29
30
            @Override
31
            public HallDto getHall(UUID id) {
32
                    return hallRepository.findById(id).map(hallMapper::toDtoFromEntity).orElseT
33
                     → hrow(NotFoundException::new);
            }
34
35
            @Override
36
            public HallDto createHall(HallRequest request) {
37
                    Hall hall = hallMapper.toEntityFromRequest(request);
38
                    Hall result = hallRepository.save(hall);
39
                    return hallMapper.toDtoFromEntity(result);
40
            }
41
42
            @Override
43
            public HallDto patchHall(UUID id, HallRequest request) {
44
                    Hall hall = hallRepository.findById(id).orElseThrow(NotFoundException::new);
45
                    hallMapper.updateFromRequest(request, hall);
46
47
                    Hall result = hallRepository.save(hall);
48
                    return hallMapper.toDtoFromEntity(result);
49
            }
50
51
            @Override
52
            public Hall findHallByUuid(UUID id) {
53
                    if (id == null) return null;
54
                    return hallRepository.findById(id).orElse(null);
55
            }
56
   }
57
                                     Listing 3: Przykładowy serwis
   package xyz.cursedman.gym_api.controllers;
1
2
   import org.hamcrest.Matchers;
3
   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;
12
   import org.springframework.test.context.junit.jupiter.SpringExtension;
13
   import org.springframework.test.web.servlet.MockMvc;
14
   import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
15
   import org.springframework.test.web.servlet.result.MockMvcResultMatchers;
   import xyz.cursedman.gym_api.domain.dtos.hall.HallRequest;
17
   import xyz.cursedman.gym_api.helpers.TestJsonHelper;
18
19
   import java.util.UUID;
20
21
22
   @SpringBootTest
   @ExtendWith(SpringExtension.class)
```

```
@AutoConfigureMockMvc
   @ActiveProfiles("test")
25
   @WithMockUser(roles = {"MANAGER", "EMPLOYEE"})
26
   @DirtiesContext(classMode = DirtiesContext.ClassMode.AFTER_EACH_TEST_METHOD)
27
   class HallControllerTest {
28
            private final String endpointUri = "/halls";
29
            private final String validHallUuid = "ce5f8d01-6fa8-4226-97fc-51d3e9cd91e5";
30
            private final String validHallTypeUuid = "2a2fa2ba-2381-4cb6-86f4-282bdbf18e81";
31
            private final HallRequest validHallRequest = HallRequest.builder()
32
                    .hallName("hall name")
33
                    .hallDescription("hall desc")
34
                    .hallTypeUuid(UUID.fromString(validHallTypeUuid))
35
                    .build();
36
            @Autowired
37
            private MockMvc mockMvc;
38
39
            // GET
40
41
            @Test
42
            void checkIfGetReturnsHttp200AndAllRecords() throws Exception {
43
                    mockMvc.perform(MockMvcRequestBuilders.get(endpointUri))
44
                             .andExpect(MockMvcResultMatchers.status().is0k())
45
                             .andExpect(MockMvcResultMatchers.jsonPath("$.length()",
46

    Matchers.greaterThan(0)));
            }
48
            @Test
49
            void checkIfGetByIdReturnsHttp200AndRequestedRecord() throws Exception {
50
                    mockMvc.perform(MockMvcRequestBuilders.get(endpointUri + "/" +
51

    validHallUuid))

                             .andExpect(MockMvcResultMatchers.status().is0k())
52
                             .andExpect(MockMvcResultMatchers.jsonPath("$").exists());
53
            }
54
55
            @Test
56
            void checkIfGetNonExistingRecordReturns404() throws Exception {
                    mockMvc.perform(MockMvcRequestBuilders.get(endpointUri + "/" +
58

    UUID.randomUUID()))

                             .andExpect(MockMvcResultMatchers.status().isNotFound());
59
            }
60
61
            // POST
62
63
            @Test
64
            void checkIfCreateReturnsHttp201AndCreatedRecord() throws Exception {
65
                    mockMvc.perform(
66
                                     MockMvcRequestBuilders.post(endpointUri)
                                              .contentType(MediaType.APPLICATION_JSON)
68
                                              .content(TestJsonHelper.stringify(validHallRequest))
69
                             ).andExpect(MockMvcResultMatchers.status().isCreated())
70
                             .andExpect(TestJsonHelper.contentEqualsJsonOf(validHallRequest,
71
                             .andExpect(
72
                                     MockMvcResultMatchers.jsonPath("$.hallType.uuid",
73

    Matchers.is(validHallTypeUuid))

                            );
74
            }
75
76
            @Test
77
            void checkIfInvalidCreateBodyReturnsHttp400() throws Exception {
78
                    mockMvc.perform(
79
                             MockMvcRequestBuilders.post(endpointUri)
80
```

```
.contentType(MediaType.APPLICATION_JSON)
81
                                      .content("{}")
82
                     ).andExpect(MockMvcResultMatchers.status().isBadRequest());
83
            }
84
85
            // PATCH
86
87
            @Test
88
            void checkIfPatchUpdateReturnsHttp200AndUpdatedRecord() throws Exception {
89
                     String hallTypeUuidToUpdate = "cc5f2a2a-1248-4e4f-aed9-5aab7c3f577a";
90
                    mockMvc.perform(
91
                                     MockMvcRequestBuilders.patch(endpointUri + "/" +
92
                                      .contentType(MediaType.APPLICATION_JSON)
93
                                              .content(TestJsonHelper.toJSONField("hallTypeUuid",
94

    hallTypeUuidToUpdate))
                             ).andExpect(MockMvcResultMatchers.status().is0k())
95
                             .andExpect(
96
                                     MockMvcResultMatchers.jsonPath("$.hallType.uuid",
97
                                         Matchers.is(hallTypeUuidToUpdate))
                             );
98
            }
99
100
101
            void checkIfPatchUpdateOfNonExistingRecordReturnsHttp404() throws Exception {
102
                     mockMvc.perform(
103
                             MockMvcRequestBuilders.patch(endpointUri + "/" + UUID.randomUUID())
104
                                      .contentType(MediaType.APPLICATION_JSON)
105
                                      .content(TestJsonHelper.stringify(validHallRequest))
106
                     ).andExpect(MockMvcResultMatchers.status().isNotFound());
107
            }
108
109
    }
```

Listing 4: Przykładowe testy jednego z kontrolerów

2.3.2

3 Testy

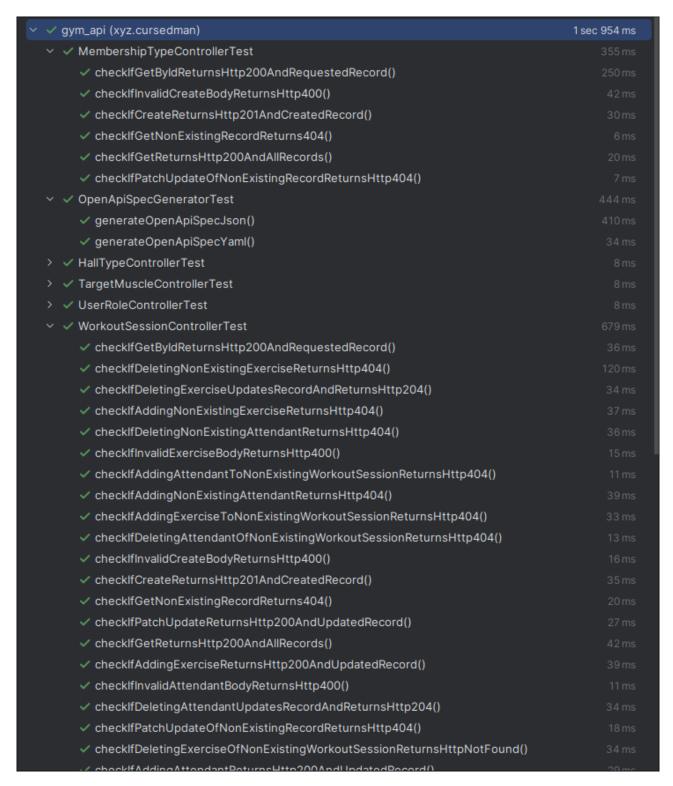
3.1 Opis metod testowania (np. testy manualne i automatyczne)

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 (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 przyszłości można by dodać możliwość śledzenia treningów w czasie rzeczywistym, co uatrakcyjniłoby korzystanie z aplikacji. Warto też pomyśleć o takich funkcjach jak powiadomienia, historia aktywności użytkownika czy integracja z urządzeniami fitness (np. smartwatche). Dodatkowym usprawnieniem mogłoby być wprowadzenie systemu rekomendacji treningów dopasowanych do celów użytkownika (np. redukcja, masa, kondycja). Rozważenie wsparcia dla wielu języków mogłoby rozszerzyć grono odbiorców aplikacji.

5 Podział pracy

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.