GymChain – API para Saúde e Bem-Estar através da Musculação

Matheus de Jesus Antunes

Análise e Desenvolvimento de Sistemas

Junho de 2025

Resumo

Este documento apresenta a descrição completa do projeto **GymChain**, uma API REST voltada ao monitoramento de treinos de musculação e ao gerenciamento de exercícios, atividades físicas e usuários. Desenvolvida em Java com Spring Boot, a aplicação tem como objetivo oferecer funcionalidades robustas de **Cadastro, Leitura, Atualização e Exclusão** (CRUD) de usuarios, exercícios, treinos e associações entre

treinos e exercícios, além de implementar um mecanismo de segurança baseado em JWT para autenticação e autorização. A API utiliza persistência em banco de dados MySQL, migrações Flyway para versionamento do esquema, validação de dados com Bean Validation (Hibernate Validator) e anotação de métodos protegidos com **@PreAuthorize**. Ferramentas como Spring Tool Suite, MySQL Workbench e Postman foram empregadas nas fases de desenvolvimento, modelagem e testes. Seguindo normas da ABNT, este documento detalha o propósito, as características técnicas, a metodologia empregada e as referências bibliográficas que fundamentaram cada parte do desenvolvimento.

Introdução

A crescente preocupação com a saúde e o bem-estar impulsionou o desenvolvimento de aplicativos que auxiliam na prática de atividades físicas (SILVA; SOUZA, 2021). No contexto de musculação, a organização e registro dos treinos, bem como o acompanhamento de exercícios, são fundamentais para garantir eficiência, segurança e evolução dos praticantes (FERREIRA; OLIVEIRA, 2020).

GymChain é projetado para servir como back-end de um sistema completo de monitoramento de treinos, permitindo que usuários criem perfis, registrem exercícios, montem treinos (workouts) relacionando exercícios específicos a cada sessão e acompanhem métricas como duração e pontos de experiência (HP – Health Points) ganhos por atividade. A aplicação adota o modelo arquitetural REST (FIELDING, 2000), fornecendo endpoints padronizados para operação sobre recursos como /users, /exercises, /workouts e /workout-exercises.

Esta documentação, seguindo normas ABNT, descreve detalhadamente o desenvolvimento do GymChain: desde a modelagem conceitual até a implementação de segurança com JWT (JSON Web Token), passando por validações, migrações de banco e arquitetura de pacotes.

Objetivos

Objetivo Geral

Desenvolver uma API REST completa para gerenciar usuários, exercícios e treinos de musculação, integrando conceitos de sustentabilidade tecnológica, usabilidade e segurança da informação.

Objetivos Específicos

1. Modelar o domínio do problema com entidades: **User**, **Exercise**, **Workout**, **WorkoutExercise** e **Permission**.

2. Implementar persistência em banco relacional MySQL, versionando o esquema com Flyway.

3. Validar dados de entrada usando Bean Validation (Hibernate Validator).

4. Fornecer endpoints RESTful para cadastro, consulta, atualização e remoção de recursos.

5. Aplicar segurança com autenticação e autorização baseada em JWT, garantindo controle granular de acesso a cada recurso conforme papéis (roles).

6. Documentar e testar a API utilizando Postman e relatórios de logs de execução.

Fundamentação Teórica

Arquitetura REST

O modelo arquitetural REST (Representational State Transfer), proposto por Fielding (2000), define seis restrições essenciais:

1. **Cliente-Servidor**: separação entre interface do usuário e armazenamento de dados.

2. **Sem Estado (Stateless)**: cada requisição deve conter todas as informações necessárias.

3. **Cacheable**: respostas podem ser armazenadas em cache para otimizar performance.

4. **Interface Uniforme**: padronização de como os recursos são identificados e manipulados (usar HTTP verbs: GET, POST, PUT, DELETE).

5. **Sistema em Camadas**: a arquitetura pode ter intermediários (load balancers, caches, etc.).

6. **Código Sob Demanda (opcional)**: permitir que o servidor devolva scripts executáveis no cliente.

A adoção de REST garante interoperabilidade, escalabilidade e simplicidade na comunicação entre front-end (SPA em Angular) e back-end (GymChain API).

Framework Spring Boot

Spring Boot (PIVOTAL, 2024) oferece convenções e autoconfiguração para acelerar o desenvolvimento de aplicações Java, especialmente APIs RESTful. O uso de **Spring Data JPA** simplifica o mapeamento objeto-relacional, enquanto **Spring Security** facilita a implementação de autenticação e autorização.

Bean Validation (Hibernate Validator)

Bean Validation (JSR 380) e sua implementação Hibernate Validator validam automaticamente dados de entrada anotando campos de entidades com anotações como @NotNull, @Size, @Email, entre outras (BALESTRIN, 2018).

Migrações Flyway

Flyway (2024) fornece versionamento automatizado do esquema de banco de dados usando scripts SQL nomeados sequencialmente (V01__, V02__, …) em src/main/resources/db/migration. Isso garante replicabilidade e rastreabilidade das alterações de tabelas.

JSON Web Token (JWT)

JSON Web Token (JWT) é um padrão aberto (RFC 7519) para troca segura de informações em formato JSON entre partes. Cada token é assinado digitalmente, permitindo ao servidor validar a integridade e autenticidade do token sem necessitar de sessão no servidor. (BANDO; FILHO, 2022).

Metodologia

Ferramentas e Tecnologias

Ferramenta / Tecnologia	Versão / Descrição
Java	21 (Oracle/OpenJDK)
Spring Boot	3.5.0
Spring Data JPA	3.0.0 (Hibernate 6.x)
Spring Security	6.x (OAuth2 Resource Server / JOSE)
MySQL	8.0 (Banco de Dados Relacional)
Flyway	9.16.0 (Migrações de esquema)
Hibernate Validator	8.x (Bean Validation – JSR 380)
Jackson	2.15.0 (Serialização JSON)
JJWT (io.jsonwebtoken)	0.11.5 (Geração e validação de JWT)
Maven	3.x (Gerenciamento de dependências e build)
Spring Tool Suite (STS)	Eclipse-based IDE para Spring
MySQL Workbench	Ferramenta para modelagem e administração do banco MySQL
Postman	Cliente REST para testes de API
Angular CLI	16.x (Front-end SPA – escopo futuro, não detalhado neste documento)

Estrutura de Pacotes e Componentes

A organização do código segue convenções que separam responsabilidades em pacotes:

```
br.com.gymchain.api
 ├── GymchainApiApplication.java          # Classe principal (ponto de partida)
 ├── domain.model
 │    ├── User.java                    # Entidade Usuário
 │    ├── Exercise.java                  # Entidade Exercício
 │    ├── Workout.java                   # Entidade Treino
 │    ├── WorkoutExercise.java               # Entidade Associação Treino-Exercício
 │    ├── Permission.java                 # Entidade Permissão
 │    ├── Gender.java                    # Enum Gênero (MASCULINO, FEMININO, etc.)
 │
```

```
├── repository
│   ├── UserRepository.java              # Repositório JPA para User
│   ├── ExerciseRepository.java          # Repositório JPA para Exercise
│   ├── WorkoutRepository.java           # Repositório JPA para Workout
│   └── WorkoutExerciseRepository.java   # Repositório JPA para WorkoutExercise
│
├── service
│   └── NonExistentOrInactiveUserException.java # Exceção customizada
│
├── exceptionhandler
│   ├── IfitnessExceptionHandler.java    # Handler global de exceções (ControllerAdvice)
│   └── NonExistentOrInactiveUserException.java
│
├── resource
│   ├── ExerciseResource.java            # Controller REST para Exercícios
│   ├── UserResource.java                # Controller REST para Usuários
│   ├── WorkoutResource.java             # Controller REST para Treinos
│   ├── WorkoutExerciseResource.java     # Controller REST para Associação Treino-Exercício
│   └── auth
│       ├── AuthController.java          # Controller REST para autenticação (JWT)
│       ├── LoginRequest.java            # DTO de login (email e senha)
│       └── TokenResponse.java           # DTO de resposta com JWT
│
└── security
    ├── CustomUserDetailsService.java    # Implementação de UserDetailsService para Spring Security
    ├── JwtTokenProvider.java            # Geração e validação de JWT
    ├── JwtAuthenticationFilter.java     # Filtro de interceptação e validação do token
    └── SecurityConfig.java              # Configurações de segurança (JWT, endpoints permitidos, CORS, etc.)
```

Essa estrutura modular facilita manutenção e leitura do código (KRUTCH, 2019).

Modelagem de Banco de Dados e Migrações Flyway

Modelagem Conceitual

1. **User**
    - **Atributos:**
        - id (PK, BIGINT)
        - name (String)

        - email (String, único)
        - password (String, Bcrypt)
        - birth_date (Date)
        - gender (Enum: MASCULINO, FEMININO)
        - active (Boolean)
        - creation_date (Timestamp)
        - last_update_date (Timestamp)
        - web3_address (String) – campo opcional para “gamificação no blockchain”.
    - **Relacionamentos:**
        - M:N com Permission (tabela intermediária user_permissions).

2. **Permission**
    - **Atributos:**
        - id (PK)
        - description (String, ex.: ROLE_REGISTER_USER, ROLE_REMOVE_WORKOUT, etc.)

3. **Exercise**
    - **Atributos:**
        - id (PK)
        - name (String)
        - description (Text)
        - muscle_group (String)
        - difficulty_level (String)
        - video_url (String, opcional)

4. **Workout**
    - **Atributos:**
        - id (PK)
        - user_id (FK → User)
        - workout_date (Date)
        - duration_minutes (Integer)
        - notes (Text, opcional)
        - total_hp_earned (Integer) – pontos de experiência (gamificação)
    - **Relacionamentos:**
        - 1:N com WorkoutExercise
5. **WorkoutExercise**
    - **Atributos:**
        - id (PK)
        - workout_id (FK → Workout)
        - exercise_id (FK → Exercise)
        - sets (Integer)
        - reps (String, ex.: “12-10-8”)
        - weight_kg (Decimal)
        - rest_seconds (Integer)

Scripts de Migração Flyway

Em src/main/resources/db/migration, foram criados quatro arquivos SQL:

V01__create_user_table.sql

```
CREATE TABLE `user` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
```

```
  `name` VARCHAR(50) NOT NULL,
  `email` VARCHAR(50) NOT NULL UNIQUE,
  `password` VARCHAR(150) NOT NULL,
  `birth_date` DATE NOT NULL,
  `gender` VARCHAR(30) NOT NULL,
  `active` TINYINT(1) NOT NULL DEFAULT 1,
  `creation_date` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    `last_update_date` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `web3_address` VARCHAR(255),
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

V02__create_exercise_and_workout_tables.sql

```
CREATE TABLE `exercise` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(100) NOT NULL,
  `description` TEXT,
  `muscle_group` VARCHAR(50) NOT NULL,
  `difficulty_level` VARCHAR(20),
  `video_url` VARCHAR(255),
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `workout` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `user_id` BIGINT(20) NOT NULL,
  `workout_date` DATE NOT NULL,
  `duration_minutes` INT NOT NULL,
  `notes` TEXT,
  `total_hp_earned` INT DEFAULT 0,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`user_id`) REFERENCES `user`(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `workout_exercise` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `workout_id` BIGINT(20) NOT NULL,
  `exercise_id` BIGINT(20) NOT NULL,
  `sets` INT NOT NULL,
  `reps` VARCHAR(50) NOT NULL,
  `weight_kg` DECIMAL(5,2),
  `rest_seconds` INT,
```

```
  PRIMARY KEY (`id`),
  FOREIGN KEY (`workout_id`) REFERENCES `workout`(`id`),
  FOREIGN KEY (`exercise_id`) REFERENCES `exercise`(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

V03__insert_initial_data.sql

```
INSERT INTO `user` (id, name, email, password, birth_date, gender, active, web3_address)
  VALUES
                                (1, 'Fernando Duarte', 'fernandoduarte@ifsp.edu.br',
'$2a$10$Ot4XGuyPP7r82nN3WXA0bOL1Qk9gShKDlVuPoyp89HoFnHcwO4Tji', '1975-11-16',
'MASCULINO', 1, '0xFernandoEthAddress'),
                                (2, 'Juliana Silva', 'julianasilva@ifsp.edu.br',
'$2a$10$Ot4XGuyPP7r82nN3WXA0bOL1Qk9gShKDlVuPoyp89HoFnHcwO4Tji', '1980-01-01',
'FEMININO', 1, '0xJulianaEthAddress');

INSERT INTO `exercise` (id, name, description, muscle_group, difficulty_level) VALUES
  (1, 'Supino Reto com Barra', 'Exercício fundamental para o peito.', 'PEITO', 'INTERMEDIARIO'),
    (2, 'Agachamento Livre', 'Rei dos exercícios, trabalha todo o corpo inferior.', 'PERNA',
'AVANCADO'),
  (3, 'Remada Curvada', 'Ótimo para as costas e postura.', 'COSTAS', 'INTERMEDIARIO'),
  (4, 'Elevação Lateral', 'Isolamento para ombros.', 'OMBROS', 'INICIANTE');
```

V04__create_permissions_and_roles_tables.sql

```
CREATE TABLE permission (
    id BIGINT(20) PRIMARY KEY,
    description VARCHAR(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE user_permission (
    id_user BIGINT(20) NOT NULL,
    id_permission BIGINT(20) NOT NULL,
    PRIMARY KEY (id_user, id_permission),
    FOREIGN KEY (id_user) REFERENCES user(id),
    FOREIGN KEY (id_permission) REFERENCES permission(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Admin
INSERT INTO user (id, name, email, password, birth_date, gender, active, web3_address)
```

```
VALUES (3, 'Administrador', 'admin@ifsp.edu.br',
'$2a$10$X607ZPhQ4EgGNaYKt3n4SONjIv9zc.VMWdEuhCuba7oLAL5IvcL5.', '2000-01-01',
'MASCULINO', 1, NULL);

-- Permissions
INSERT INTO permission (id, description) VALUES
  (1, 'ROLE_REGISTER_USER'),
  (2, 'ROLE_REMOVE_USER'),
  (3, 'ROLE_SEARCH_USER'),
  (4, 'ROLE_REGISTER_ACTIVITY'),
  (5, 'ROLE_REMOVE_ACTIVITY'),
  (6, 'ROLE_SEARCH_ACTIVITY'),
  (7, 'ROLE_REGISTER_EXERCISE'),
  (8, 'ROLE_REMOVE_EXERCISE'),
  (9, 'ROLE_SEARCH_EXERCISE'),
  (10, 'ROLE_REGISTER_WORKOUT'),
  (11, 'ROLE_REMOVE_WORKOUT'),
  (12, 'ROLE_SEARCH_WORKOUT');

-- User-Permission Mapping
-- Admin (id=3) recebe todas as permissões
INSERT INTO user_permission (id_user, id_permission) VALUES
  (3,1),(3,2),(3,3),(3,4),(3,5),(3,6),(3,7),(3,8),(3,9),(3,10),(3,11),(3,12);

-- Fernando (id=1)
INSERT INTO user_permission (id_user, id_permission) VALUES
  (1,1),(1,3),(1,4),(1,6),(1,9),(1,12);

-- Juliana (id=2)
INSERT INTO user_permission (id_user, id_permission) VALUES
  (2,1),(2,3),(2,4),(2,6),(2,9),(2,12);
```

> **Observação:** após inserir esses scripts em src/main/resources/db/migration, o Flyway executará as migrações na ordem: V01 → V02 → V03 → V04, criando e populando as tabelas.

Implementação dos Endpoints (Controllers/Resources)

Cada recurso REST segue convenções de nomenclatura:

- **/users** – manipula usuários

- **/exercises** – manipula exercícios
- **/workouts** – manipula treinos
- **/workout-exercises** – associa treino e exercício
- **/auth/login** – gera JWT

Exemplo de anotações usadas:

- @RestController e @RequestMapping("/users")
- Métodos HTTP:
    - @GetMapping → Listar todos
    - @GetMapping("/{id}") → Buscar por ID
    - @PostMapping → Criar novo
    - @PutMapping("/{id}") → Atualizar (Substituir parciamente com BeanUtils.copyProperties)
    - @DeleteMapping("/{id}") → Excluir
- Validações com @Valid e Bean Validation nos DTOs/entidades.
- Tratamento de exceções de negócio (ex.: usuário inativo) lança ResponseStatusException.
- Métodos protegidos com @PreAuthorize("hasAuthority('ROLE_X') and #oauth2.hasScope('read' ou 'write')").

Segurança e Autenticação JWT

Configuração Geral

Em src/main/resources/application.properties:

```
# Configurações de datasource
spring.datasource.url=jdbc:mysql://localhost:3306/gym_chain_db?createDatabaseIfNotExist=true&u
seSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=pretinha22
```

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

# Propriedades de JWT
security.jwt.secret=MinhaChaveSecretaMuitoSegura123!@#
security.jwt.expiration=3600000

# Internationalization
spring.messages.basename=messages
spring.messages.encoding=UTF-8
```

SecurityConfig.java

```
package br.com.gymchain.api.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.*;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.*;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

   @Autowired
   private JwtAuthenticationFilter jwtAuthenticationFilter;

   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http
        .csrf().disable()
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and()
        .authorizeRequests()
          .antMatchers("/auth/login").permitAll()
```

```
            .anyRequest().authenticated();

        http.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

- @EnableGlobalMethodSecurity(prePostEnabled = true) permite usar @PreAuthorize nos controllers.

- O filtro JwtAuthenticationFilter intercepta cada requisição, verifica se o header “Authorization” contém “Bearer <token>” e, se válido, insere o UsernamePasswordAuthenticationToken no contexto de segurança (SecurityContext) do Spring.

JwtTokenProvider.java

Gerencia geração, análise e validação de tokens JWT:

```
package br.com.gymchain.api.security;

import io.jsonwebtoken.*;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Component;
import java.util.Date;

@Component
public class JwtTokenProvider {

    @Value("${security.jwt.secret}")
    private String secret;

    @Value("${security.jwt.expiration}")
    private long expiration;
```

```
    public String generateToken(Authentication authentication) {
        String username = authentication.getName();
        Date now = new Date();
        Date expiryDate = new Date(now.getTime() + expiration);

        return Jwts.builder()
            .setSubject(username)
            .setIssuedAt(now)
            .setExpiration(expiryDate)
            .signWith(SignatureAlgorithm.HS512, secret)
            .compact();
    }

    public String getUsernameFromToken(String token) {
        return Jwts.parser()
            .setSigningKey(secret)
            .parseClaimsJws(token)
            .getBody()
            .getSubject();
    }

    public boolean validateToken(String token) {
        try {
            Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
            return true;
        } catch (JwtException | IllegalArgumentException e) {
            return false;
        }
    }
}
```

JwtAuthenticationFilter.java

Intercepta requisições e carrega detalhes do usuário:

```
package br.com.gymchain.api.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.*;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import jakarta.servlet.FilterChain;
import jakarta.servlet.http.*;
```

```
@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    @Autowired
    private JwtTokenProvider tokenProvider;

    @Autowired
    private CustomUserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(
            HttpServletRequest request,
            HttpServletResponse response,
            FilterChain filterChain) {

        try {
            String jwt = extractToken(request);
            if (jwt != null && tokenProvider.validateToken(jwt)) {
                String username = tokenProvider.getUsernameFromToken(jwt);
                UserDetails userDetails = userDetailsService.loadUserByUsername(username);

                UsernamePasswordAuthenticationToken authToken =
                    new UsernamePasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities());

                authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        } catch (Exception ex) {
            // Logar ou ignorar — falha de autenticação
        }
        filterChain.doFilter(request, response);
    }

    private String extractToken(HttpServletRequest request) {
        String bearer = request.getHeader("Authorization");
        if (bearer != null && bearer.startsWith("Bearer ")) {
            return bearer.substring(7);
        }
        return null;
    }
}
```

CustomUserDetailsService.java

Carrega usuário e suas permissões:

```
package br.com.gymchain.api.security;

import br.com.gymchain.api.domain.model.User;
import br.com.gymchain.api.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.*;
import org.springframework.stereotype.Service;

import java.util.stream.Collectors;

@Service
public class CustomUserDetailsService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(email)
            .orElseThrow(() -> new UsernameNotFoundException("Usuário não encontrado"));

        return org.springframework.security.core.userdetails.User
            .withUsername(user.getEmail())
            .password(user.getPassword())
            .authorities(
                user.getPermissions()
                    .stream()
                    .map(p -> new SimpleGrantedAuthority(p.getDescription()))
                    .collect(Collectors.toList())
            )
            .build();
    }
}
```

Testes e Validação

- **Postman**: utilizados para validar todas as rotas, incluindo autenticação (/auth/login) e endpoints protegidos com token “Bearer <JWT>”.

- **Testes de Integração**: (padrão Spring Boot Starter Test) para cobrir operações CRUD e fluxos de autenticação/erro.

- **Mensagens Internacionalizadas**:

    - messages.properties e ValidationMessages.properties armazenam as chaves de texto para mensagens de erro e validação em português (UTF-8).

Exemplo:

invalid.message=Mensagem inválida
resource.not-found=Recurso não encontrado
user.non-existent-or-inactive=Usuário inexistente ou inativo

    -
    - O IfitnessExceptionHandler utiliza MessageSource e LocaleContextHolder para retornar mensagens localizadas.

Resultados Esperados

1. **Cadastro de Usuários:**

    - Rotas:

        - POST /users – cria usuário com dados válidos.

        - GET /users – lista todos os usuários (Role = ROLE_SEARCH_USER).

        - GET /users/{id} – busca usuário por ID.

        - PUT /users/{id} – atualiza dados do usuário (exceto ID e datas).

        - DELETE /users/{id} – remove usuário (Role = ROLE_REMOVE_USER).

    - Campo active permite ativar/desativar usuários sem excluí-los.

2. **Gerenciamento de Exercícios:**

    - POST /exercises – cria exercício (Role = ROLE_REGISTER_EXERCISE).

    - GET /exercises – lista exercícios (Role = ROLE_SEARCH_EXERCISE).

- PUT /exercises/{id} – altera dados do exercício (Role = ROLE_REGISTER_EXERCISE).
- DELETE /exercises/{id} – remove exercício (Role = ROLE_REMOVE_EXERCISE).

3. **Gerenciamento de Treinos (Workouts):**
   - POST /workouts – cria treino vinculado a usuário ativo (Role = ROLE_REGISTER_WORKOUT).
   - GET /workouts – lista todos os treinos (Role = ROLE_SEARCH_WORKOUT).
   - GET /workouts/{id} – busca treino por ID (Role = ROLE_SEARCH_WORKOUT).
   - PUT /workouts/{id} – atualiza treino (Role = ROLE_REGISTER_WORKOUT).
   - DELETE /workouts/{id} – remove treino (Role = ROLE_REMOVE_WORKOUT).

O cálculo de HP (Health Points) é feito em método interno:

HP = (durationMinutes / 10) * 10

-

4. **Associação Treino-Exercício (WorkoutExercise):**
   - POST /workout-exercises – cria associação Treino + Exercício (Role = ROLE_REGISTER_WORKOUT).
   - GET /workout-exercises – lista todas as associações (Role = ROLE_SEARCH_WORKOUT).
   - GET /workout-exercises/{id} – busca associação por ID (Role = ROLE_SEARCH_WORKOUT).
   - PUT /workout-exercises/{id} – atualiza associação (Role = ROLE_REGISTER_WORKOUT).
   - DELETE /workout-exercises/{id} – exclui associação (Role = ROLE_REMOVE_WORKOUT).

5. **Segurança e Autenticação JWT:**
   - **POST /auth/login** – recebe JSON com email e password; retorna { "token":"<JWT>" }.

- **Filtros de Segurança:** Todas as rotas, exceto /auth/login, exigem cabeçalho HTTP Authorization: Bearer <token>.
- **Roles/Permissões:** Protegem cada endpoint com @PreAuthorize("hasAuthority('ROLE_X') and #oauth2.hasScope('read' ou 'write')").
- **Validação de Token:** O JwtAuthenticationFilter extrai token e preenche o contexto de segurança.

Considerações finais

O **GymChain** atende ao objetivo de disponibilizar uma API robusta para gerenciamento de usuários, exercícios e treinos de musculação, com ênfase em usabilidade, validação de dados e segurança. A escolha de **Spring Boot** acelerou o desenvolvimento, aproveitando autoconfiguração, injeção de dependências, mapeamento objeto-relacional e integrações com mecanismos de segurança (JWT).

A modelagem contou com entidades bem definidas, migrações Flyway que garantem versionamento de banco e inicialização automática, além de controle granular de permissões por usuário. A documentação segue normas ABNT, descrevendo detalhadamente todos os componentes, metodologias e referências.

Em futuras etapas, recomenda-se:

- Desenvolver a interface **SPA em Angular**, consumindo esses endpoints.
- Implementar funcionalidades de histórico de treinos, relatórios e exportação de dados.
- Adicionar testes automatizados de integração e cobertura de segurança.
- Explorar a gamificação via web3_address (blockchain) para recompensas aos usuários.

Referências

- BALESTRIN, F. L. Validação de Dados com Bean Validation (JSR-380) e Hibernate Validator. São Paulo: Novatec, 2018.

- BANDO, L.; FILHO, R. “Implementação de Autenticação JWT em Aplicações Web”. *Revista de Desenvolvimento de Software*, v.15, n.2, p.87-98, jun. 2022.

- FIELDING, R. T.; TAYLOR, R. N. “Principles of Representational State Transfer (REST)”. *Proceedings of the 2000 WWW Conference*, May 2000.

- FERREIRA, M.; OLIVEIRA, A. “A Importância do Registro de Treinos para a Evolução na Musculação”. *Journal of Sports Science*, v.12, n.1, p.45-53, jan. 2020.

- KRUTCH, J. “Good Practices in Package Organization for Java Projects”. *Software Architecture Journal*, v.7, n.4, p.10-19, dez. 2019.

- OWASP Foundation. “JSON Web Token (JWT) Security Considerations”. OWASP Authentication Cheat Sheet, 2020. Disponível em: https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_Cheat_Sheet_for_Java.html (Acesso em 01 jun. 2025).

- PIVOTAL SOFTWARE, INC. *Spring Boot Reference Guide*. Versão 3.5.0, 2024.