



Gustavo Mourato, Leonardo Gutzeit,  
Paulo Rosado, Thomaz Lima & Vinícius de Andrade



Veo



# O QUE É A FORGEFIT?

- Uma plataforma integrada que **unifica a gestão da academia & a jornada do aluno**, com uma abordagem focada em três pilares estratégicos:
  - **1. Gestão da Rotina Fitness:**
    - Organização inteligente de Aulas, espaços & Professores;
    - Criação & acompanhamento de Treinos personalizados.
  - **2. Acompanhamento da Evolução Individual:**
    - Monitoramento de progresso físico através de Bioimpedância & medidas;
    - Ciclo de feedback contínuo com Avaliação de Professores.
  - **3. Estímulo ao Engajamento e Comunidade:**
    - Mecanismos de gamificação com Rankings e Pontuações;
    - Criação de Guildas para fortalecer o senso de comunidade & a motivação em grupo.



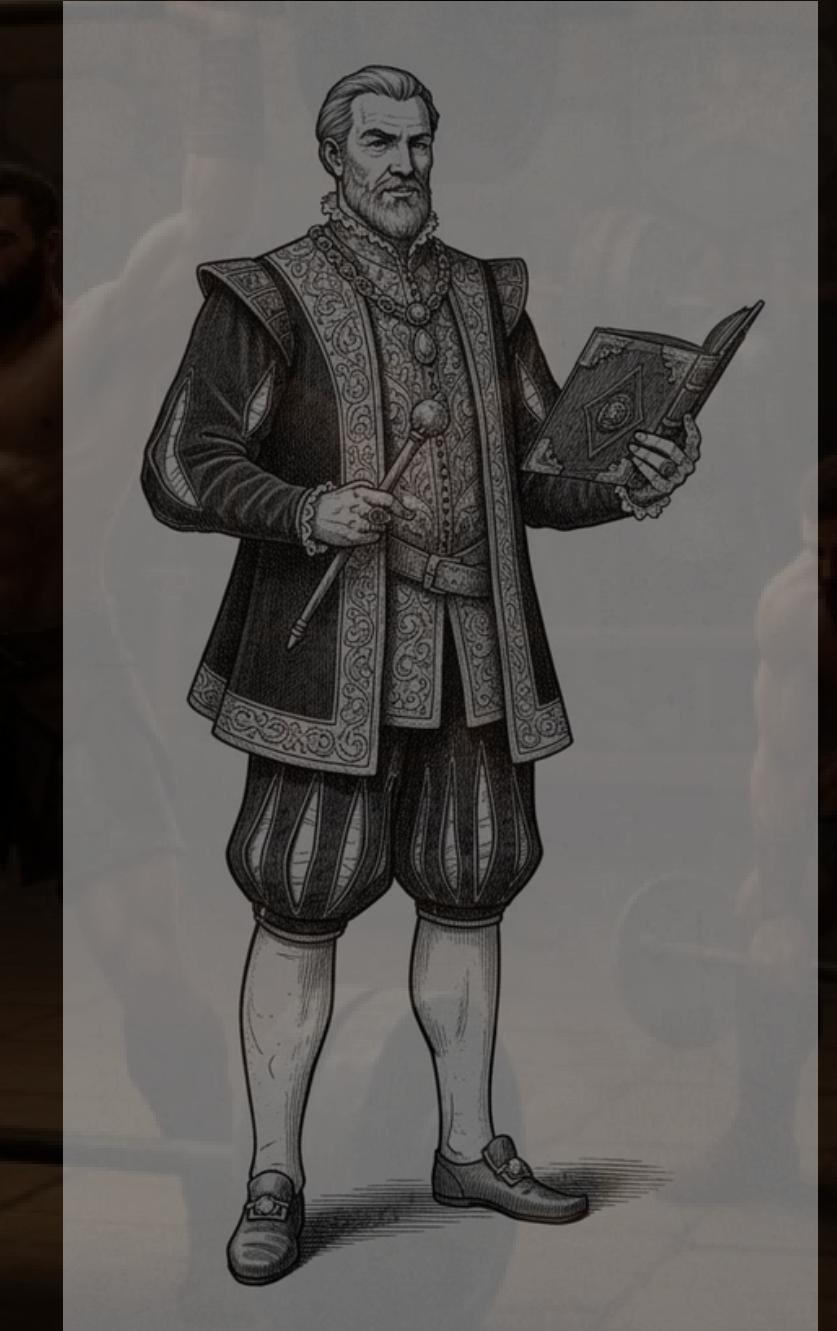
# PERSONAS



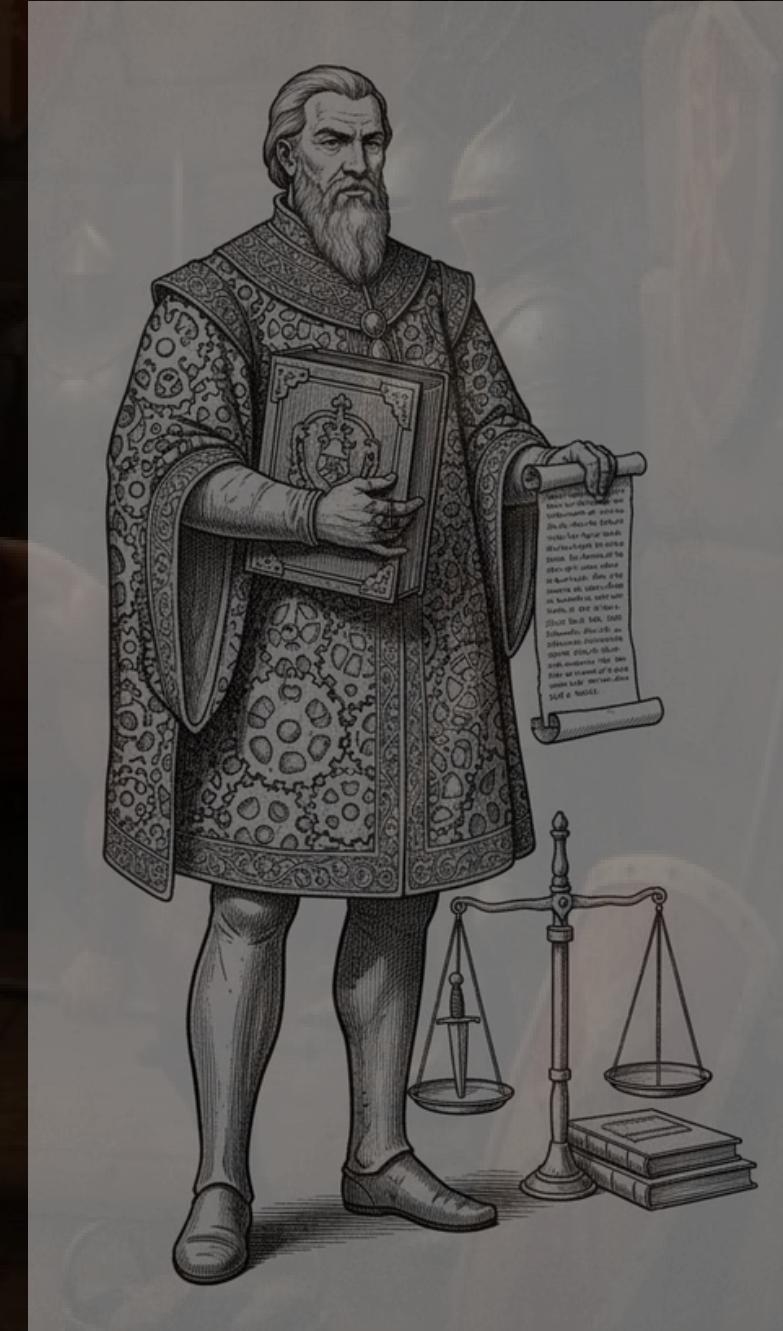
ALUNO



PROFESSOR



GERENTE



SISTEMA

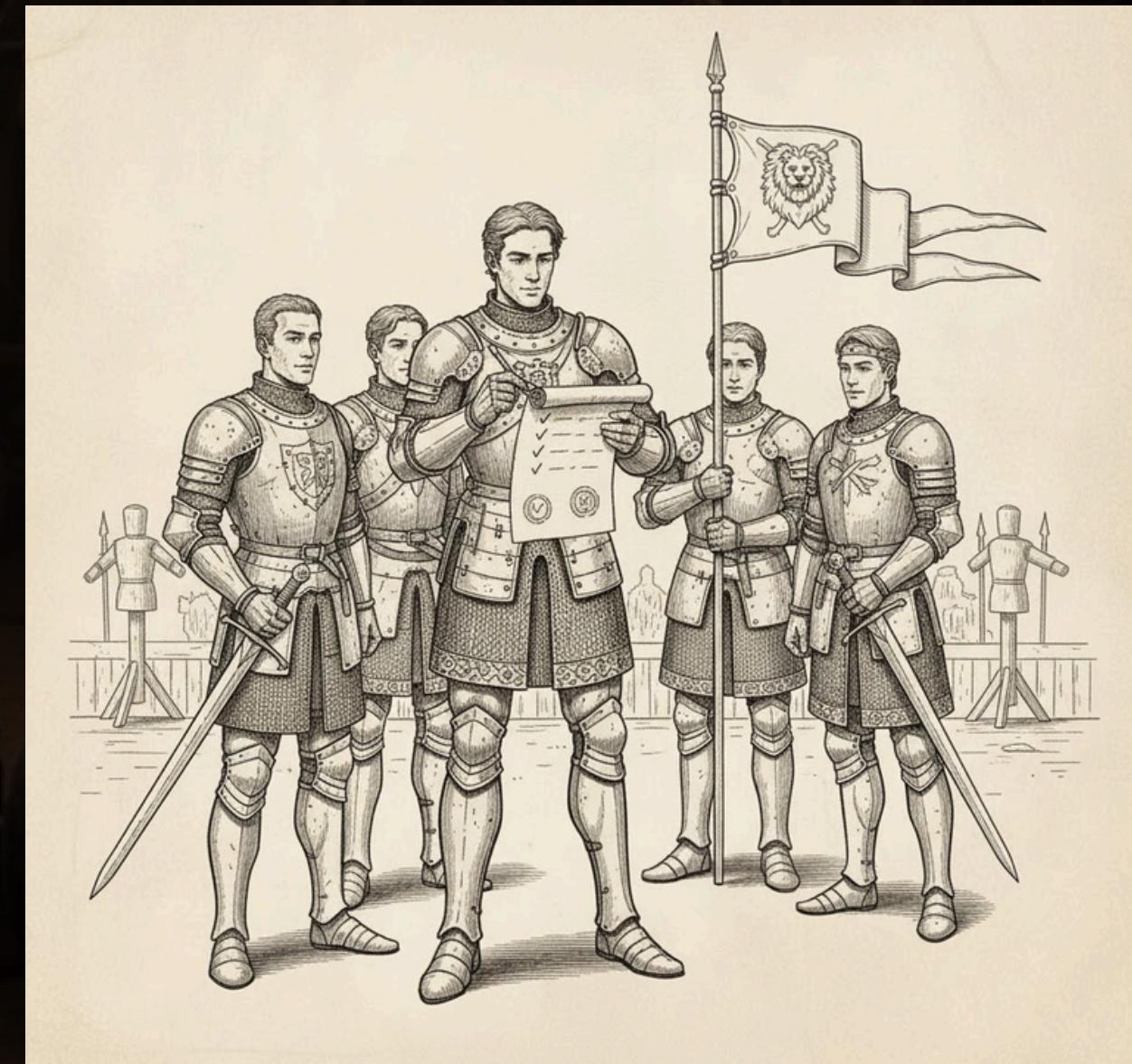


# PERSONAS - ALUNO

- **Jornada:** Participar de competições com amigos
- **Objetivo:** Engajar-se com outros Alunos por meio de desafios em grupo.

- **Passos:**

- Criar uma Guilda;
- Alterar dados de uma Guilda;
- Excluir uma Guilda;
- Analisar o Rank de Pontuação;
- Entrar em uma Guilda com Código de Convite;
- Realizar Check-in diário nos Treinos;
- Acompanhar o Ranking da academia.



# PERSONAS - ALUNO

- Jornada: Participar de competições com amigos

The wireframe illustrates the student's journey through the application:

- TELA DE GUILDA**: Shows a sidebar with 'Aula', 'Treino', 'Guilda' (highlighted), 'Torneio', 'Ranking', and 'Evolução'. The main area says "Participe de competições com seus amigos" with buttons for "Entrar em uma guilda" and "Criar guilda". It contains placeholder text aboutLorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur at dolor lorem. Nullam non ex sagittis.
- Entrar / Criar uma guilda**: Shows the same sidebar. The main area says "Entrar em uma guilda" with a "Inserir o código:" input field, an "ENTRAR" button, and a "CANCELAR" button.
- Página inicial da guilda com histórico de check-ins**: Shows the guilda's name "#COD3" and a history of check-ins:
  - Gustavo Mourato concluiu o treino de Peito "VAMOOOO, O DE HOJE TA PAGO" 2 dias atrás
  - Leonardo participou da aula de pilates "GALERIA MUITO FODA" 3 horas atrásA large red 'X' is overlaid on the bottom right of the history section.
- RANKING DOS ALUNOS**: Shows the sidebar. The main area displays a ranking table:

Rank	Aluno	Pontos
#1	Erico Chen	9600 pts
#2	João Marcelo	670pts
#3	Matheus Domingos	500pts
#4	José Braz	- 120pts
#5	Júlia Sales	- 100pts
#6	Demétrius	- 50pts

A large red 'X' is overlaid on the bottom right of the ranking table.
- Crie a sua guilda**: Shows the sidebar. The main area says "Crie a sua guilda" with a "Nome da guilda:" input field, a "Detalhes:" input field, an "CRIAR" button, and a "CANCELAR" button.

- Histórias:

- Listar a Pontuação individual dos Alunos da academia;
- Criar uma Guilda;
- Visualizar Código de Convite;
- Entrar em uma Guilda com o Código de Convite.

# PERSONAS - ALUNO

- **Jornada:** Participar de competições com amigos

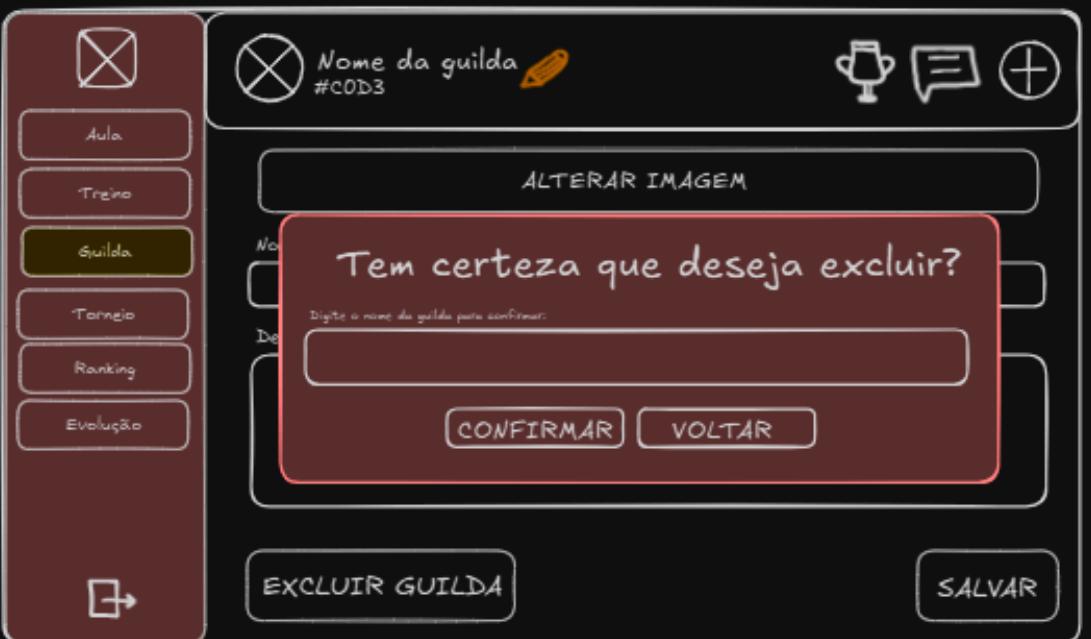
The image displays three mobile application screens for managing a guild:

- Ranking dos membros da guilda:** Shows a ranking of guild members. At the top, there are buttons for "Nome da guilda" (#COD3), "Aula", "Treino", "Guilda" (highlighted in green), "Torneio", "Ranking", and "Evolução". Below this is a section titled "Ranking de membros" with filters for "Esta semana", "Este mês", and "Sempre". The ranking table shows:

Rank	Membro	Pontos
#1	Gustavo Mourato	1200pts
#2	Paulo Rosado	350pts
#3	Vinícius de Andrade	200pts
#4	Leonardo Matos	- 120pts
#5	Sophia Gallindo	- 100pts
#6	Ana Clara	- 50pts
- Realizar check-in de treino:** A screen for performing a training check-in. It includes a "Selecionar o treino" dropdown, a date field set to "hoje (30/09/2025)", a message input field, and a "ENVIAR" button.
- Editar informações da guilda:** A screen for editing guild information. It has fields for "Nome da guilda" (#COD3), "ALTERAR IMAGEM", "Nome da guilda" (input field), "Descrição" (input field), and "EXCLUIR GUILDA". A "SALVAR" button is located at the bottom right.

- **Histórias:**

- Alterar dados de uma Guilda;
- Excluir uma Guilda criada;
- Listar Pontuação de membros da Guilda por semana, mês e sempre;
- Realizar Check-in de Treino.





# PERSONAS - ALUNO

- **Jornada:** Realizar atividades físicas
- **Objetivo:** Acompanhar suas Aulas, desempenho & evolução corporal.
- **Passos:**
  - Inscrever-se em Aulas;
  - Cancelar Inscrição em uma Aula;
  - Avaliar Professor;
  - Acompanhar Lista de Espera;
  - Acompanhar evolução física.



# PERSONAS - ALUNO

- Jornada: Realizar atividades físicas

The screenshots illustrate the student's journey through the application:

- LISTAR AULA:** Shows a list of available classes across different modalities: Boxe, Judo, Dança, Pilates, Karatê, and Yoga, each with its respective time slot.
- RESERVAR AULA:** Allows the student to reserve a spot in a specific class, such as Boxe, taught by Professor Vínius de Andrade, from 9:00-10:30.
- CANCELAR AULA:** Provides a confirmation dialog for canceling a class reservation. It asks if the user really wants to cancel and lists the class details (Boxe, Professor Vínius de Andrade, 9:00-10:30).
- LISTAR AULA:** Another view of the class listing screen.
- Cancelar presença:** A button located on the class listing screen, likely used to cancel attendance for a specific class.

# PERSONAS - ALUNO

- Jornada: Realizar atividades físicas

The image shows a sequence of five mobile application screens illustrating the user flow for evaluating a teacher:

- AVALIAR PROFESSOR**: A navigation bar with icons for Aula (green), Treino (blue), Squilda (orange), Torneio (yellow), Ranking (purple), and Evolução (red). Below the bar is a back arrow icon.
- Nome do Professor**: Fields for Didática (Didactic), Atenção (Attention), and Pontualidade (Punctuality), each represented by a row of diamond icons. A text input field labeled "Comentários" (Comments) is at the bottom, and a red "Enviar" (Send) button is at the bottom right.
- Nome do Professor**: The same evaluation fields as the previous screen, but with different diamond icons filled with yellow. A text input field labeled "Foi muito atencioso em relação à minha forma" (Was very attentive regarding my form) is at the bottom, and a red "Enviar" (Send) button is at the bottom right.
- Avaliação enviada com sucesso.**: A confirmation message in a blue box with a "Confirmar" (Confirm) button. The evaluation fields show yellow-filled diamonds. A text input field labeled "Foi muito atencioso em relação à minha forma" (Was very attentive regarding my form) is at the bottom, and a red "Enviar" (Send) button is at the bottom right.
- AVALIAR PROFESSOR**: A navigation bar with icons for Aula (green), Treino (blue), Squilda (orange), Torneio (yellow), Ranking (purple), and Evolução (red). Below the bar is a back arrow icon.

- Histórias:
  - Avaliar Professor;
  - Listar a pontuação por atributo.

# PERSONAS - ALUNO

- Jornada: Realizar atividades físicas

The screenshots illustrate a student's interaction with a mobile application for physical activities:

- Screenshot 1: ACEITAR PARTICIPAÇÃO EM AULA**  
Shows a navigation menu on the left with options: Aula (highlighted), Treino, Guilda, Torneio, Ranking, and Evolução. In the center, there is a "Boxe" section with a red invite icon. Below it, details for a class with Professor: Vinicius de Andrade, time 9:00-10:30, and two lists: "Lista de presença" (1: Thomas Lima, 2: Pedro, 3: Joabe, 4: Marcos) and "Lista de espera" (empty). At the bottom are buttons "Quero entrar na lista de espera" (blue) and "Quero sair da lista de espera" (pink).
- Screenshot 2: Boxe**  
Shows the same "Boxe" section after acceptance. The "Lista de espera" now contains names: 1: Evaldo, 2: Saulo.
- Screenshot 3: Aceitando vaga**  
Shows a confirmation dialog: "Uma vaga foi liberada! Deseja confirmar sua presença?" with "Sim" (green) and "Não" (pink) buttons.
- Screenshot 4: ACOMPANHAMENTO DE BIOIMPEDÂNCIA**  
Shows a navigation menu on the left with options: Aula, Treino, Guilda, Torneio, Ranking, and Evolução. In the center, there is a "Demetrius" section with a circular profile picture. Below it, a table displays bioimpedance data for Demetrius:

Massa muscular	55kg
33kg	Massa magra
% de gordura	20%
20%	Comprimento da cintura
Comprimento do braço direito	80cm
61cm	Comprimento da perna direita
Comprimento do braço esquerdo	92cm
60cm	Comprimento da perna esquerda
	92cm

At the bottom is a "Quero sair da lista de espera" button.
- Screenshot 5: Boxe**  
Shows the "Boxe" section again, identical to Screenshot 2, with the "Lista de espera" containing 1: Evaldo, 2: Saulo.



# PERSONAS



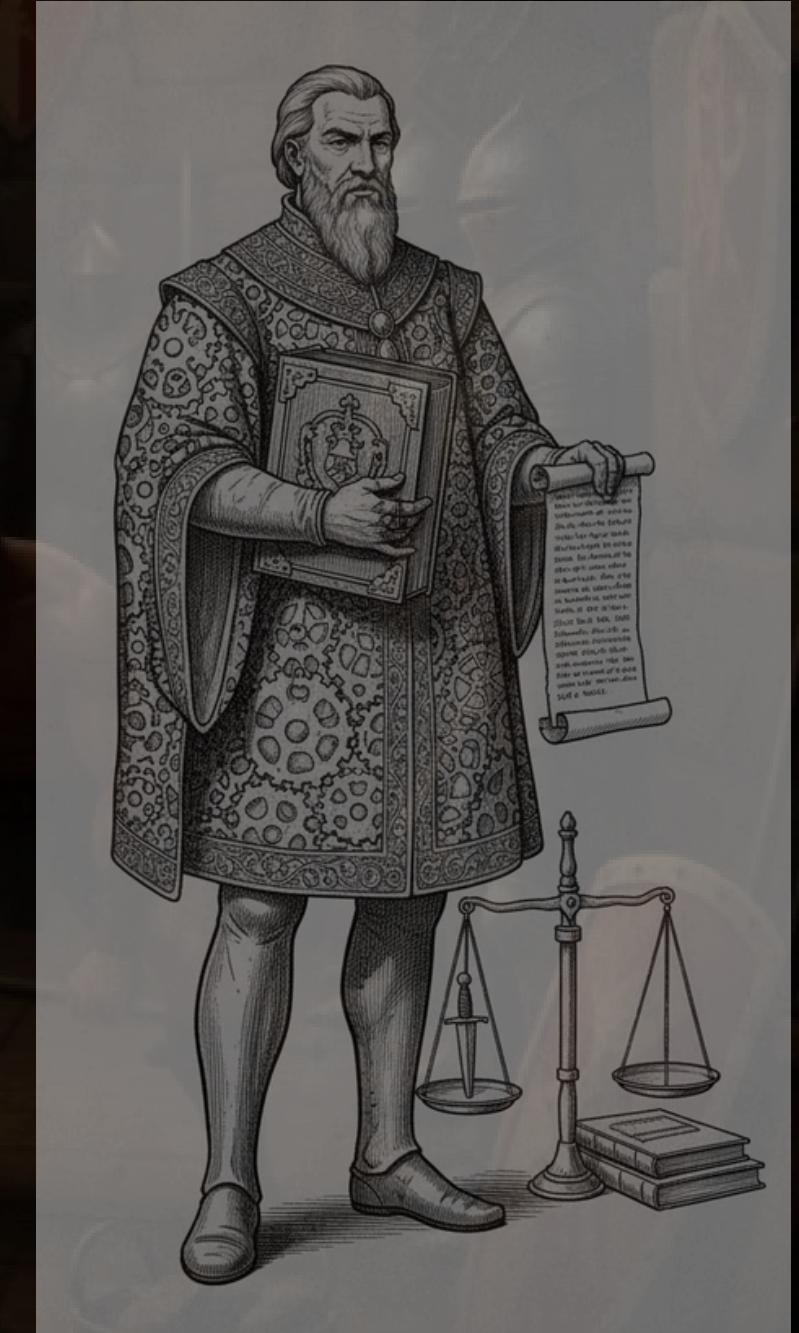
ALUNO



PROFESSOR



GERENTE



SISTEMA

# PERSONAS – PROFESSOR

- **Jornada:** Treinar Alunos
- **Objetivo:** Organizar Aulas, Treinos & acompanhar o progresso dos Alunos.

- **Passos:**

- Criar Aula;
- Alterar data/hora da Aula;
- Cancelar Aula;
- Personalizar Treino de Aluno;
- Avaliar Aluno;
- Realizar Bioimpedância.



# PERSONAS - PROFESSOR

- Jornada: Treinar Alunos

The image displays three wireframe prototypes for a mobile application interface, likely for a professor's digital dashboard. The prototypes are arranged vertically.

- Top Prototype (Left): CRIAR AULA (Create Class)**
  - Left sidebar: Buttons for 'Aula' (selected), 'Treino', and 'Evolução'.
  - Main area: A large circular '+' button labeled 'criar aula'.
  - Preview section: Shows a class named 'Boxe - Seg 9:00-10:30' with a red diamond icon.
- Top Prototype (Right): EXCLUIR AULA (Delete Class)**
  - Left sidebar: Buttons for 'Aula' (selected), 'Treino', and 'Evolução'.
  - Main area: A preview of a class named 'Boxe' with a red diamond icon.
  - Confirmation dialog: 'GOSTARIA MESMO DE CANCELAR A AULA?' with 'NÃO' (No) and 'CANCELAR AULA' (Cancel Class) buttons.
- Bottom Prototype: EDITAR AULA (Edit Class)**
  - Left sidebar: Buttons for 'Aula' (selected), 'Treino', and 'Evolução'.
  - Main area: A preview of a class named 'aula de boxe' with a red diamond icon.
  - Form fields:
    - name: 'aula de boxe'
    - horario: '00:00'
    - data: 'xx/xx/xxx'
    - descrição: Text input field
    - matriculados: List of students: ALAN PATRICK, NEYMAR JUNIOR, LUCAS LIMA, DERIK LACERDA, RAFAEL THYERE. Each student has a small red diamond icon next to their name.
  - Switch: 'TORNAR RECORRENTE?' (Make Recurrent?) with 'SIM' (Yes) and 'NÃO' (No) options.

- Histórias:

- Criar Aula / Aulas recorrentes;
- Editar data/hora das Aulas;
- Cancelar Aula;
- Encerrar recorrência de Aulas.

# PERSONAS - PROFESSOR

- Jornada: Treinar Alunos

The interface consists of four main panels:

- Panel 1 (Top Left):** Shows a list of registered students ("matriculados") with edit icons. The list includes ALAN PATRICK, NEYMAR JUNIOR, LUCAS LIMA, DERIK LACERDA, and RAFAEL THYERE. A search bar at the bottom is labeled "pesquise".
- Panel 2 (Top Middle):** Shows the "Acompanhamento de Evolução" (Monitoring of Progress) for ALAN PATRICK. It displays various physical measurements and body composition data. Buttons for "Enviar" (Send) and "Confirmar" (Confirm) are present.
- Panel 3 (Top Right):** Shows the "Acompanhamento de Evolução" for ALAN PATRICK after data has been sent. A message says "Dados de acompanhamento enviados com sucesso" (Monitoring data sent successfully). Buttons for "Enviar" (Send) and "Confirmar" (Confirm) are present.
- Panel 4 (Bottom):** Shows the "Treino" (Training) screen for ALAN PATRICK. It includes buttons for exercises A, B, C, and a plus sign. Below are categories for muscle groups: perna, costas, biceps, inferiores, regenerativo, superiores, and a "SALVAR" (Save) button.

- Histórias:
  - Criar Treino de Aluno;
  - Atualizar Treino de Aluno;
  - Salvar dados corporais de um Aluno.



# PERSONAS



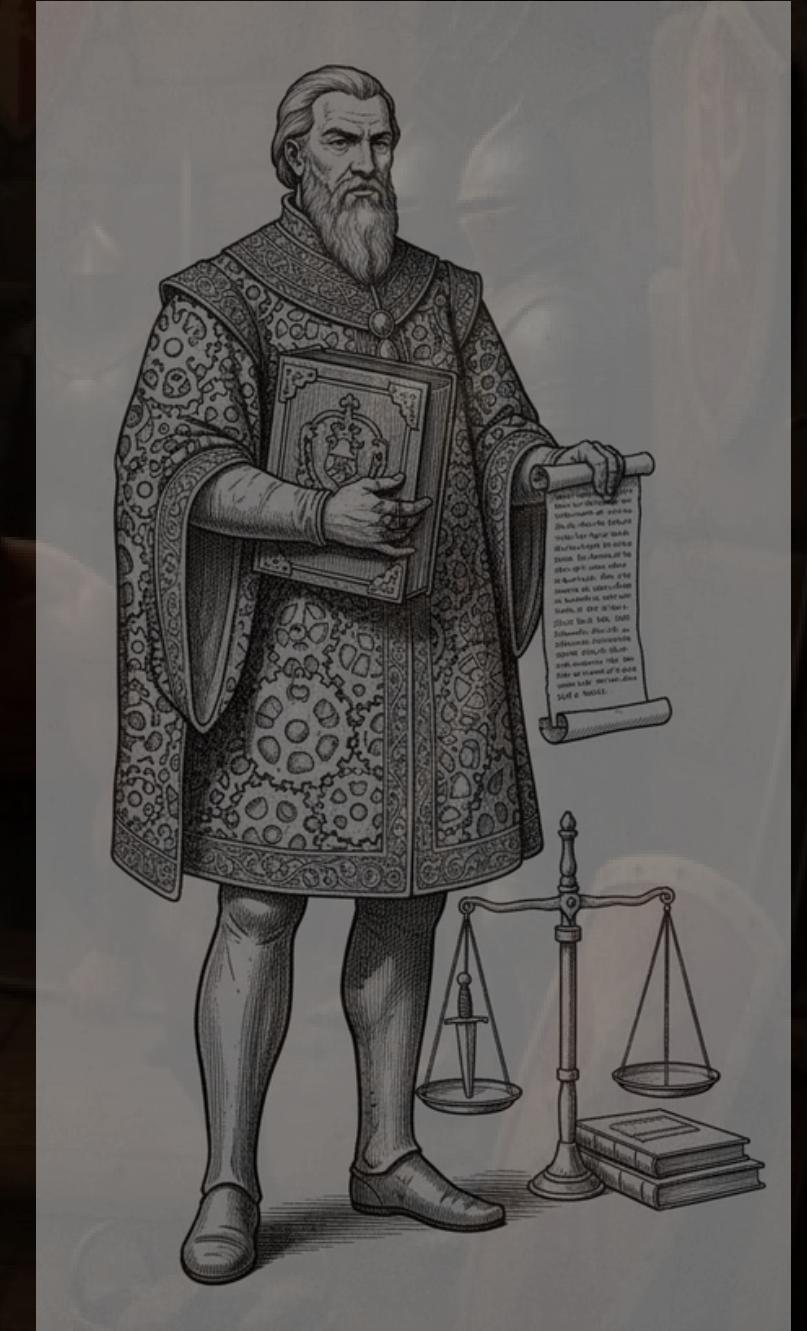
ALUNO



PROFESSOR



GERENTE



SISTEMA



# PERSONAS - GERENTE

- **Jornada:** Proporcionar competições que incentivem os Alunos
- **Objetivo:** Criar competições & Torneios entre Guildas para aumentar o engajamento.
- **Passos:**
  - Iniciar um Torneio entre Guildas;
  - Alterar dados de um Torneio;
  - Cancelar um Torneio.





# PERSONAS - GERENTE

- **Jornada:** Proporcionar competições que incentivem os Alunos

The image displays six wireframe prototypes for a mobile application, likely for managing tournaments. The prototypes are arranged in two rows of three. Each prototype includes a navigation bar on the left with icons for Aula, Treino, Guilda, Torneio (highlighted in green), Ranking, and Evolução.

- Create Tournament:** Shows a large button "Criar torneio" and a "Visualizar anteriores" button.
- Define Prizes:** Shows a tournament titled "Torneio de São João" ending in 12 days. It lists the first prize as "#1 Kit 1kg Whey + 600g Creatina". There are two empty fields for additional prizes.
- Edit Tournament:** Shows the same tournament details. It includes fields for "Name do torneio" (Torneio de São João), "Data de inicio" (22/10/25), and "Data de Fim" (29/10/25). Buttons for "SALVAR ALTERAÇÕES" and "CANCELAR TORNEIO" are present.
- Start Tournament:** Shows a button "Iniciar torneio". Below it, fields for "Name do torneio", "Data de inicio", and "Data de Fim" are shown, along with "CRIAR" and "CANCELAR" buttons.
- Prize Details:** Shows a detailed view of the second-place prize: "Premiação do 2º lugar". It includes fields for "Premiação" and "URL da Imagem". Buttons for "CONFIRMAR" and "VOLTAR" are at the bottom.
- Cancel Confirmation:** Shows a confirmation dialog: "Tem certeza que deseja cancelar?". It includes a field "Digite o nome do torneio para confirmar" and buttons "CONFIRMAR" and "VOLTAR".

- **Histórias:**

- Iniciar um Torneio;
- Definir os Prêmios do Torneio;
- Alterar dados de um Torneio;
- Cancelar um Torneio.



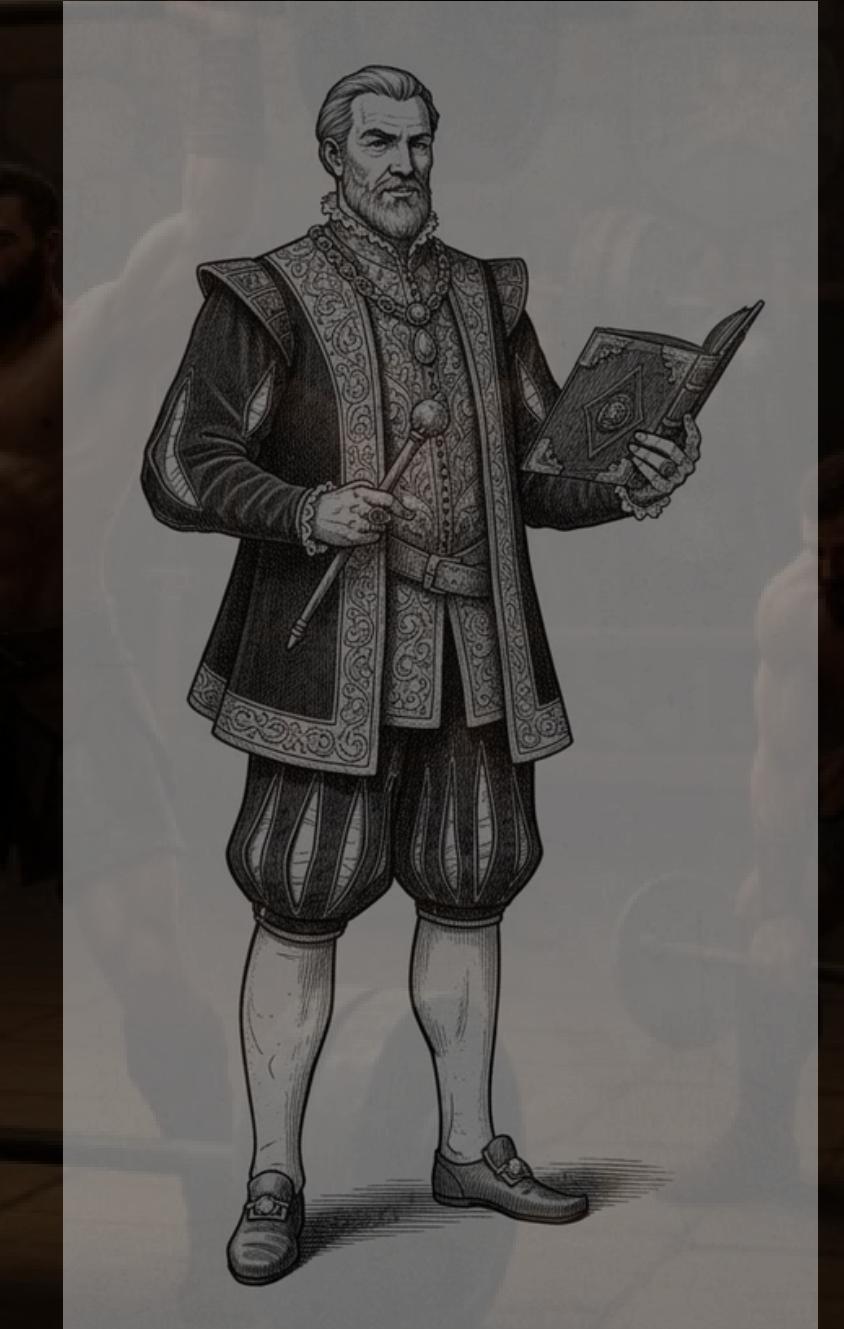
# PERSONAS



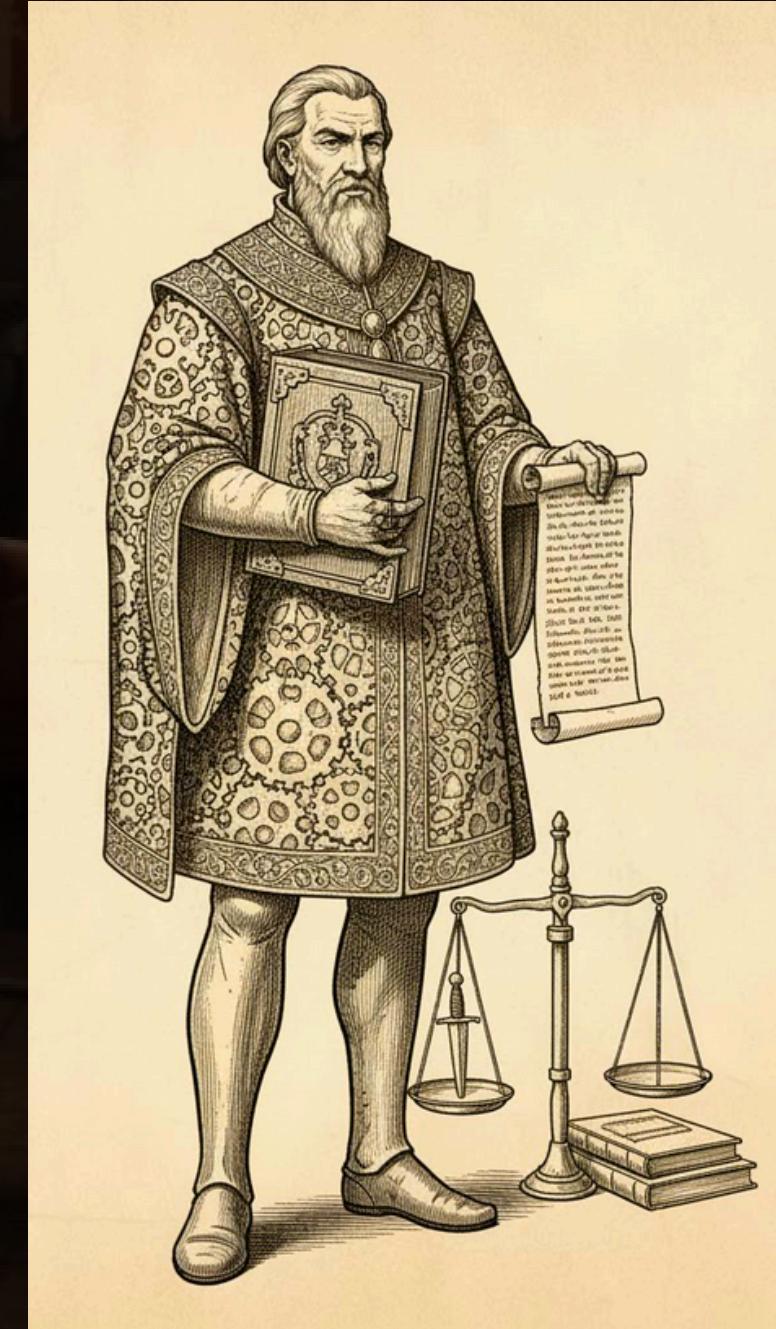
ALUNO



PROFESSOR



GERENTE



SISTEMA

# PERSONAS – SISTEMA

- **Jornada:** Automação & Controle
- **Objetivo:** Garantir o funcionamento automatizado & eficiente do ambiente.

- **Passos:**

- Maximizar a ocupação de uma Aula;
- Controlar Frequência dos Alunos;
- Promover automaticamente Alunos da Lista de Espera;
- Bloquear novas Inscrições por excesso de Faltas;
- Desbloquear Aluno após período determinado;
- Integrar com sistema de catraca inteligente.





# PERSONAS - SISTEMA

- **Funcionalidade:** Cancelamento de Reserva com política de reembolso
- **Cenários de Teste:**
  - Cancelamento com reembolso total processado;
  - Tentativa de cancelamento após prazo de reembolso total.



```
1 Feature: Cancelamento de reserva com política de reembolso
2   # Regra 1 – Cancelamento com antecedência suficiente (reembolso total em até 15 dias)
3
4 Scenario: Cancelamento com reembolso total processado
5   Given existe uma reserva confirmada para o aluno com matrícula "123.456.789-01" no dia "10/09/2025" às "10:00" com duração de "60 minutos"
6   When o aluno solicita o cancelamento em "20/08/2025"
7   Then o sistema informa "cancelamento aprovado, reembolso integral em processamento"
8
9 Scenario: Tentativa de cancelamento após prazo de reembolso total
10  Given existe uma reserva confirmada para o aluno com matrícula "234.567.890-12" no dia "10/09/2025" às "10:00" com duração de "60 minutos"
11  When o aluno solicita o cancelamento em "01/09/2025"
12  Then o sistema informa "cancelamento aprovado, reembolso parcial em processamento"
```

# PERSONAS - SISTEMA

- **Funcionalidade:** Cancelamento de Reserva com política de reembolso
- **Automação dos testes BDD com Cucumber:**
  - CancelamentoDeReservaFuncionalidade - @Given

```
● ● ●  
1  @Given("existe uma reserva confirmada para o aluno com matrícula {string} no dia {string} às {string} com duração de {string}")  
2    public void existe_uma_reserva_confirmada_para_o_aluno_com_matricula_no_dia(String matriculaStr, String dataStr, String horario, String duracao) {  
3      matriculaAluno = new Matricula(matriculaStr);  
4  
5      Cpf cpf = new Cpf(matriculaStr.replaceAll("[^0-9]", "").substring(0, 11));  
6      Aluno aluno = new Aluno(matriculaAluno, cpf, "Aluno Teste", LocalDate.of(1990, 1, 1));  
7      contexto.repositorio.salvar(aluno);  
8  
9      LocalDate data = LocalDate.parse(dataStr, dateFormatter);  
10     String[] partesHorario = horario.split(":");  
11     int hora = Integer.parseInt(partesHorario[0]);  
12     int minuto = Integer.parseInt(partesHorario[1]);  
13  
14     int duracaoMinutos = Integer.parseInt(duracao.replace(" minutos", "").trim());  
15  
16     LocalDateTime inicio = LocalDateTime.of(data, java.time.LocalTime.of(hora, minuto));  
17     LocalDateTime fim = inicio.plusMinutes(duracaoMinutos);  
18  
19     aulaCriada = contexto.aulaService.criarAulaUnica(  
20         new ProfessorId(1),  
21         Modalidade.YOGA,  
22         Espaco.ESTUDIO_PILATES,  
23         20,  
24         inicio,  
25         fim  
26     );  
27  
28     contexto.reservaService.reservarVaga(matriculaAluno, aulaCriada.getId());  
29 }
```



# PERSONAS - SISTEMA

- **Funcionalidade:** Cancelamento de Reserva com política de reembolso
- **Automação dos testes BDD com Cucumber:**
  - CancelamentoDeReservaFuncionalidade - @When

```
● ● ●  
1  @When("o aluno solicita o cancelamento em {string}")  
2      public void o_aluno_solicita_o_cancelamento_em(String dataStr) {  
3          dataCancelamento = LocalDate.parse(dataStr, dateFormatter);  
4          LocalDateTime momentoCancelamento = dataCancelamento.atStartOfDay();  
5  
6          try {  
7              mensagemResposta = contexto.reservaService.cancelarReserva(matriculaAluno, aulaCriada.getId(), momentoCancelamento);  
8  
9          } catch (IllegalArgumentException e) {  
10              contexto.excecao = e;  
11              mensagemResposta = "nenhuma reserva encontrada para cancelamento";  
12          } catch (Exception e) {  
13              contexto.excecao = e;  
14              mensagemResposta = e.getMessage();  
15          }  
16      }
```



# PERSONAS - SISTEMA

- **Funcionalidade:** Cancelamento de Reserva com política de reembolso
- **Automação dos testes BDD com Cucumber:**
  - CancelamentoDeReservaFuncionalidade - @Then



```
1  @Then("o sistema informa {string}")
2      public void o_sistema_informa(String mensagemEsperada) {
3          assertNotNull(mensagemResposta);
4          assertEquals(mensagemEsperada, mensagemResposta);
5
6          if (!mensagemEsperada.equals("nenhuma reserva encontrada para cancelamento")) {
7              Optional<Aula> aulaDoRepositorio = contexto.repositorio.obterPorId(aulaCriada.getId());
8              assertTrue(aulaDoRepositorio.isPresent(), "A aula deveria estar persistida no repositório");
9
10             Aula aulaSalva = aulaDoRepositorio.get();
11             Optional<Reserva> reservaSalva = aulaSalva.obterReserva(matriculaAluno);
12
13             assertTrue(reservaSalva.isPresent(), "A reserva deveria existir no repositório");
14             assertEquals(StatusReserva.CANCELADA_PELO_ALUNO, reservaSalva.get().getStatus(),
15                         "O status da reserva deveria ser CANCELADA_PELO_ALUNO no repositório");
16         }
17     }
```



# PERSONAS – SISTEMA

```
matriculaAluno = new Matricula(matriculaStr);
Cpf cpf = new Cpf(matriculaStr.replaceAll("[^0-9]", "").substring(0, 11));
Aluno aluno = new Aluno(matriculaAluno, cpf, "Aluno Teste", LocalDate.of(1990, 1, 1));
contexto.repositorio.salvar(aluno);
```

- Funcionalidade: Cancelamento de Reserva com política de reembolso
- Código necessário para os testes serem bem sucedidos:
  - Matricula, Aluno, AcademiaFuncionalidade



```
1 public class Matricula {
2     private final String valor;
3
4     public Matricula(String valor) {
5         notNull(valor, "A matrícula não pode ser nula");
6         notBlank(valor, "A matrícula não pode estar em branco");
7         this.valor = valor;
8     }
9
10    public String getValor() {
11        return valor;
12    }
13
14    @Override
15    public boolean equals(Object obj) {
16        if (obj != null && obj instanceof Matricula) {
17            Matricula other = (Matricula) obj;
18            return valor.equals(other.valor);
19        }
20        return false;
21    }
22
23    @Override
24    public int hashCode() {
25        return Objects.hash(valor);
26    }
27
28    @Override
29    public String toString() {
30        return valor;
31    }
32 }
```

```
1 public class Aluno {
2     private static final AtomicLong matriculaCounter = new AtomicLong(1);
3
4     private final Matricula matricula;
5     private final Cpf cpf;
6     private String nome;
7     private LocalDate dataNascimento;
8     private int pontuacaoTotal;
9     private double creditos;
10    private StatusAluno status;
11    private LocalDate bloqueioAte;
12    private PlanoDeTreinoId planoAtivoId;
13    private GuildaId guildaId;
14
15    private final List<AvaliacaoFisica> historicoDeAvaliacoes = new ArrayList<>();
16    private final List<PlanoDeTreinoId> historicoDePlanosIds = new ArrayList<>();
17    private final List<Frequencia> historicoDeFrequencia = new ArrayList<>();
18
19    public Aluno(Matricula matricula, Cpf cpf, String nome, LocalDate dataNascimento) {
20        notNull(matricula, "A matrícula não pode ser nula");
21        notNull(cpf, "O CPF não pode ser nulo");
22        notNull(nome, "O nome não pode ser nulo");
23        notNull(dataNascimento, "A data de nascimento não pode ser nula");
24
25        this.matricula = matricula;
26        this.cpf = cpf;
27        this.nome = nome;
28        this.dataNascimento = dataNascimento;
29        this.status = StatusAluno.ATIVO;
30        this.pontuacaoTotal = 0;
31        this.creditos = 0.0;
32    }
33
34    public Aluno(Cpf cpf, String nome, LocalDate dataNascimento) {
35        this(new Matricula(String.valueOf(matriculaCounter.getAndIncrement())), cpf, nome, dataNascimento);
36    }
37 }
```

```
1 public class AcademiaFuncionalidade implements EventoBarramento {
2
3     public final Repositorio repositorio;
4     public final GuildaService guildaService;
5     public final CheckinService checkinService;
6     public final TorneioService torneioService;
7     public final AulaService aulaService;
8     public final AlunoService alunoService;
9     public final ReservaService reservaService;
10    public final RankingService rankingService;
11    public final AvaliacaoService avaliacaoService;
12    public final ReembolsoService reembolsoService;
13    public final TreinoService treinoService;
14    public final FrequenciaService frequenciaService;
15    public final br.com.forgefit.dominio.aluno.AvaliacaoFisicaService avaliacaoFisicaService;
16
17    public Aluno alunoAtual;
18    public Aula aulaAtual;
19
20    public List<Object> eventos;
21    public Exception excecao;
22
23    public AcademiaFuncionalidade() {
24        this.repositorio = new Repositorio();
25        this.eventos = new ArrayList<>();
26    }
27 }
```



# PERSONAS – SISTEMA

```
matriculaAluno = new Matricula(matriculaStr);
Cpf cpf = new Cpf(matriculaStr.replaceAll("[^0-9]", "").substring(0, 11));
Aluno aluno = new Aluno(matriculaAluno, cpf, "Aluno Teste", LocalDate.of(1990, 1, 1));
contexto.repositorio.salvar(aluno);
```

- **Funcionalidade:** Cancelamento de Reserva com política de reembolso
- **Código necessário para os testes serem bem sucedidos:**
  - Repositorio, AlunoRepositorio

```
● ● ●  
1 public class Repositorio implements AlunoRepositorio,
2                                     GuildaRepositorio, CheckinRepositorio, TorneioRepositorio,
3                                     AulaRepositorio, RankingRepositorio, AvaliacaoRepositorio,
4                                     TreinoRepositorio, FrequenciaRepositorio, ProfessorRepository {
5
6     private Map<Matricula, Aluno> alunos = new HashMap<>();
7     private Map<Cpf, Aluno> alunosPorCpf = new HashMap<>();
8
9     @Override
10    public void salvar(Aluno aluno) {
11        notNull(aluno, "O aluno não pode ser nulo");
12        alunos.put(aluno.getMatricula(), aluno);
13        alunosPorCpf.put(aluno.getCpf(), aluno);
14    }
15
16    @Override
17    public Optional<Aluno> obterPorMatricula(Matricula matricula) {
18        notNull(matricula, "A matrícula não pode ser nula");
19        return Optional.ofNullable(alunos.get(matricula));
20    }
21
22    @Override
23    public Optional<Aluno> obterAlunoPorCpf(Cpf cpf) {
24        notNull(cpf, "O CPF não pode ser nulo");
25        return Optional.ofNullable(alunosPorCpf.get(cpf));
26    }
27
28    private void notNull(Object value, String errorMessage) {
29        if (value == null) {
30            throw new IllegalArgumentException(errorMessage);
31        }
32    }
33 }
```



```
● ● ●  
1 package br.com.forgefit.dominio.aluno;
2
3 import java.util.Optional;
4
5 public interface AlunoRepositorio {
6     void salvar(Aluno aluno);
7     Optional<Aluno> obterPorMatricula(Matricula matricula);
8     Optional<Aluno> obterAlunoPorCpf(Cpf cpf);
9 }
```

# PERSONAS - SISTEMA

- **Funcionalidade:** Cancelamento de Reserva com política de reembolso
- **Código necessário para os testes serem bem sucedidos:**
  - AulaService, AulaRepositorio, criarAulaUnica(), verificarConflitoHorario(), temConflito()

```
● ● ●
1 public class AulaService {
2     private final AulaRepositorio aulaRepositorio;
3     private final AtomicInteger aulaIdCounter = new AtomicInteger(1);
4     private final AtomicInteger excecaoIdCounter = new AtomicInteger(1);
5
6     public AulaService(AulaRepositorio aulaRepositorio) {
7         notNull(aulaRepositorio, "O repositório de aulas não pode ser nulo");
8         this.aulaRepositorio = aulaRepositorio;
9     }
10
11    public Aula criarAulaUnica(ProfessorId professorId, Modalidade modalidade, Espaco espaco, int capacidade,
12                                LocalDateTime inicio, LocalDateTime fim) {
13        verificarConflitoHorario(espaco, professorId, inicio, fim, null);
14
15        AulaId id = new AulaId(aulaIdCounter.getAndIncrement());
16        Aula aula = new Aula(id, professorId, modalidade, espaco, capacidade, inicio, fim);
17        aulaRepositorio.salvar(aula);
18        return aula;
19    }
}
```

```
● ● ●
1 public interface AulaRepositorio {
2     void salvar(Aula aula);
3     Optional<Aula> obterPorId(AulaId aulaId);
4     List<Aula> listarTodas();
5     List<Aula> buscarPorEspacoEPeriodo(Espaco espaco, LocalDateTime inicio, LocalDateTime fim);
6     List<Aula> buscarPorProfessorEPeriodo(ProfessorId professorId, LocalDateTime inicio, LocalDateTime fim);
7 }
```

```
aulaCriada = contexto.aulaService.criarAulaUnica(
    new ProfessorId(1),
    Modalidade.YOGA,
    Espaco.ESTUDIO_PILATES,
    20,
    inicio,
    fim
);

contexto.reservaService.reservarVaga(matriculaAluno, aulaCriada.getId());
```

```
● ● ●
1 public Aula criarAulaRecorrente(ProfessorId professorId, Modalidade modalidade, Espaco espaco, int capacidade,
2                                 LocalDateTime inicio, LocalDateTime fim, TipoRecorrenca tipoRecorrenca,
3                                 List<DiaDaSemana> diasDaSemana, LocalDate dataFimRecorrenca) {
4     notNull(tipoRecorrenca, "O tipo de recorrência não pode ser nulo");
5     notNull(diasDaSemana, "Os dias da semana não podem ser nulos");
6     notNull(dataFimRecorrenca, "A data fim da recorrência não pode ser nula");
7
8     Recorrenca recorrenca = new Recorrenca(tipoRecorrenca, diasDaSemana, dataFimRecorrenca);
9
10    verificarConflitoHorario(espaco, professorId, inicio, fim, null);
11
12    LocalDate dataAtual = inicio.toLocalDate();
13    LocalTime horaInicio = inicio.toLocalTime();
14    LocalTime horaFim = fim.toLocalTime();
15
16    while (dataAtual.isBefore(dataFimRecorrenca) || dataAtual.isEqual(dataFimRecorrenca)) {
17        dataAtual = dataAtual.plusDays(1);
18
19        DiaDaSemana diaAtual = DiaDaSemana.fromDayOfWeek(dataAtual.getDayOfWeek());
20        if (diasDaSemana.contains(diaAtual) && dataAtual.isAfter(inicio.toLocalDate())) {
21            LocalDateTime inicioOcorrencia = LocalDateTime.of(dataAtual, horaInicio);
22            LocalDateTime fimOcorrencia = LocalDateTime.of(dataAtual, horaFim);
23
24            try {
25                verificarConflitoHorario(espaco, professorId, inicioOcorrencia, fimOcorrencia, null);
26            } catch (IllegalStateException e) {
27                throw new IllegalStateException("Conflito em ocorrência futura");
28            }
29        }
30    }
31
32    AulaId id = new AulaId(aulaIdCounter.getAndIncrement());
33    Aula aula = new Aula(id, professorId, modalidade, espaco, capacidade, inicio, fim, recorrenca);
34
35    aulaRepositorio.salvar(aula);
36
37    return aula;
}
```

# PERSONAS – SISTEMA

- Funcionalidade: Cancelamento de Reserva com política de reembolso
- Código necessário para os testes serem bem sucedidos:
  - ReservaService, Reserva, PosicaoListaDeEspera, reservarVaga(), entrarNaListaDeEspera()

```
● ● ●
1 public class ReservaService {
2     private final AulaRepositorio aulaRepositorio;
3     private final AlunoService alunoService;
4     private final ReembolsoService reembolsoService;
5
6     public ReservaService(AulaRepositorio aulaRepositorio, AlunoService alunoService,
7             ReembolsoService reembolsoService) {
8         notNull(aulaRepositorio, "O repositório de aulas não pode ser nulo");
9         notNull(reembolsoService, "O serviço de reembolso não pode ser nulo");
10        this.aulaRepositorio = aulaRepositorio;
11        this.alunoService = alunoService;
12        this.reembolsoService = reembolsoService;
13    }
14
15    public Reserva reservarVaga(Matricula alunoMatricula, AulaId aulaId) {
16        notNull(alunoMatricula, "A matrícula do aluno não pode ser nula");
17        notNull(aulaId, "O id da aula não pode ser nulo");
18
19        Aula aula = obterAula(aulaId);
20
21        if (aula.temVagaDisponivel()) {
22            Reserva reserva = new Reserva(alunoMatricula, LocalDateTime.now());
23            aula.adicionarReserva(reserva);
24            aulaRepositorio.salvar(aula);
25            return reserva;
26        }
27
28        entrarNaListaDeEspera(alunoMatricula, aula);
29        return null;
30    }
}
```

```
● ● ●
1 public String tentarReservarVaga(Matricula alunoMatricula, AulaId aulaId) {
2     Reserva reserva = reservarVaga(alunoMatricula, aulaId);
3     if (reserva != null) {
4         return "Reserva confirmada.";
5     }
6     return "Vaga indisponível. Aluno adicionado à lista de espera.";
7 }
8
9 private void entrarNaListaDeEspera(Matricula alunoMatricula, Aula aula) {
10    PosicaoListaDeEspera posicao = new PosicaoListaDeEspera(alunoMatricula, LocalDateTime.now());
11    aula.adicionarNaListaDeEspera(posicao);
12    aulaRepositorio.salvar(aula);
13 }
14
15 public class PosicaoListaDeEspera {
16     private final Matricula alunoMatricula;
17     private final LocalDateTime timestampDeEntrada;
18
19     public PosicaoListaDeEspera(Matricula alunoMatricula, LocalDateTime timestampDeEntrada) {
20         notNull(alunoMatricula, "A matrícula do aluno não pode ser nula");
21         this.alunoMatricula = alunoMatricula;
22
23         notNull(timestampDeEntrada, "O timestamp de entrada não pode ser nulo");
24         this.timestampDeEntrada = timestampDeEntrada;
25     }
26
27     public Matricula getAlunoMatricula() {
28         return alunoMatricula;
29     }
30
31     public LocalDateTime getTimestampDeEntrada() {
32         return timestampDeEntrada;
33     }
34 }
35
36 public class Reserva {
37     private final Matricula alunoMatricula;
38     private final LocalDateTime dataDaReserva;
39     private StatusReserva status;
40
41     public Reserva(Matricula alunoMatricula, LocalDateTime dataDaReserva) {
42         notNull(alunoMatricula, "A matrícula do aluno não pode ser nula");
43         notNull(dataDaReserva, "A data da reserva não pode ser nula");
44
45         this.alunoMatricula = alunoMatricula;
46         this.dataDaReserva = dataDaReserva;
47         this.status = StatusReserva.CONFIRMADA;
48     }
49
50     public Matricula getAlunoMatricula() {
51         return alunoMatricula;
52     }
53
54     public LocalDateTime getDataDaReserva() {
55         return dataDaReserva;
56     }
57
58     public StatusReserva getStatus() {
59         return status;
60     }
61 }
```

```
aulaCriada = contexto.aulaService.criarAulaUnica(
    new ProfessorId(1),
    Modalidade.YOGA,
    Espaco.ESTUDIO_PILATES,
    20,
    inicio,
    fim
);

contexto.reservaService.reservarVaga(matriculaAluno, aulaCriada.getId());
```

# PERSONAS – SISTEMA

- Funcionalidade: Cancelamento de Reserva com política de reembolso
- Código necessário para os testes serem bem sucedidos:
  - ReembolsoService, cancelarReserva(), promoverPrimeiroDaListaSeHouver()

```
try {
    mensagemResposta = contexto.reservaService.cancelarReserva(matriculaAluno, aulaCriada.getId(), momentoCancelamento);
} catch (IllegalArgumentException e) {
    contexto.excecao = e;
    mensagemResposta = "nenhuma reserva encontrada para cancelamento";
} catch (Exception e) {
    contexto.excecao = e;
    mensagemResposta = e.getMessage();
}
```

```
● ● ●
1 public class ReembolsoService {
2
3     private static final long HORAS_MINIMAS_REEMBOLSO_TOTAL = 15 * 24;
4     private static final double PERCENTUAL_REEMBOLSO_TOTAL = 1.0;
5     private static final double PERCENTUAL_REEMBOLSO_PARCIAL = 0.5;
6     private static final long HORAS_MINIMAS_REEMBOLSO_PARCIAL = 5 * 24;
7
8     public double calcularCreditoDeReembolso(AulaId aulaId, Aula aula, LocalDateTime momentoCancelamento) {
9         notNull(aulaId, "O ID da aula não pode ser nulo");
10        notNull(aula, "A aula não pode ser nula");
11        notNull(momentoCancelamento, "O momento do cancelamento não pode ser nulo");
12
13        LocalDateTime inicioAula = aula.getInicio();
14        Duration antecedencia = Duration.between(momentoCancelamento, inicioAula);
15        long horasAntecedencia = antecedencia.toHours();
16
17        if (horasAntecedencia >= HORAS_MINIMAS_REEMBOLSO_TOTAL) {
18            return calcularValorBase() * PERCENTUAL_REEMBOLSO_TOTAL;
19        }
20
21        if (horasAntecedencia >= HORAS_MINIMAS_REEMBOLSO_PARCIAL) {
22            return calcularValorBase() * PERCENTUAL_REEMBOLSO_PARCIAL;
23        }
24
25        return 0.0;
26    }
27
28    private double calcularValorBase() {
29        return 20.0;
30    }
31
32    public boolean temDireitoAReembolso(Aula aula, LocalDateTime momentoCancelamento) {
33        return calcularCreditoDeReembolso(aula.getId(), aula, momentoCancelamento) > 0;
34    }
}
```

```
● ● ●
1     public void cancelarReserva(Matricula alunoMatricula, AulaId aulaId) {
2         cancelarReserva(alunoMatricula, aulaId, LocalDateTime.now());
3     }
4
5     public String cancelarReserva(Matricula alunoMatricula, AulaId aulaId, LocalDateTime momentoCancelamento) {
6         notNull(alunoMatricula, "A matrícula do aluno não pode ser nula");
7         notNull(aulaId, "O id da aula não pode ser nulo");
8         notNull(momentoCancelamento, "O momento do cancelamento não pode ser nulo");
9
10        Aula aula = obterAula(aulaId);
11
12        double credito = reembolsoService.calcularCreditoDeReembolso(aulaId, aula, momentoCancelamento);
13
14        alunoService.adicionarCreditos(alunoMatricula, credito);
15
16        aula.cancelarReserva(alunoMatricula);
17        aulaRepositorio.salvar(aula);
18
19        promoverPrimeiroDaListaSeHouver(aula);
20
21        return reembolsoService.obterMensagemDeReembolso(credito);
22    }
23
24    private void promoverPrimeiroDaListaSeHouver(Aula aula) {
25        aula.removerPrimeiroDaListaDeEspera().ifPresent(posicao -> {
26
27            Matricula proximoAlunoMatricula = posicao.getAlunoMatricula();
28
29            Reserva novaReserva = new Reserva(proximoAlunoMatricula, LocalDateTime.now());
30            aula.adicionarReserva(novaReserva);
31            aulaRepositorio.salvar(aula);
32        });
33    }
34
35    private Aula obterAula(AulaId aulaId) {
36        return aulaRepositorio.obterPorId(aulaId)
37            .orElseThrow(() -> new IllegalArgumentException("Aula não encontrada"));
38    }
39 }
```

# GUILDA FORGEFIT



GUSTAVO  
MOURATO



LEONARDO  
MATOS



PAULO  
ROSADO



THOMAZ  
LIMA



VINÍCIUS  
DE ANDRADE



NA FORGEFIT NÓS NÃO APENAS LEVANTAMOS FERRO,  
NÓS FORJAMOS FORÇA!