1) Núcleo de autenticação com sessão opaca (sem e-mail)

Objetivo: permitir registro, login, logout e identificação do usuário autenticado com o mínimo de moving parts.

Decisões conservadoras

- **Sessão opaca em cookie httpOnly** (token aleatório + hash no banco) — nada de JWT ainda.
- **BCrypt** (via `PasswordEncoder`) para senhas.
- **CSRF ON** (padrão do Spring Security) e **CORS** restrito ao seu domínio.
- **Spring Security** com filtro que lê a sessão pelo cookie.

Endpoints

- `POST /auth/register` — cria usuário.
- `POST /auth/login` — valida credenciais; cria **session** e seta cookie `SESSIONID` (`HttpOnly; Secure; SameSite=Strict`).
- `POST /auth/logout` — revoga sessão atual (apaga do banco).
- `GET /auth/me` — retorna usuário autenticado (id, email).

Esquema (Flyway)

- `users(id UUID, email UNIQUE, password_hash, created_at, updated_at)`
- `sessions(id UUID, user_id, token_hash, user_agent, ip, expires_at, revoked_at)`

Aceite mínimo

- Login cria sessão válida por 7 dias.
- `me` retorna 401 sem sessão e 200 com sessão.
- Logout invalida sessão e limpa cookie.

Extensibilidade preparada

- Colunas `ip`, `user_agent`, `expires_at`, `revoked_at` já habilitam auditoria, lista de sessões e rotação futura.
- Trocar depois para JWT é opcional e não quebra o contrato atual.

2) Recuperação de senha por e-mail (fluxo completo)

Objetivo: permitir reset seguro sem expor nada do usuário.

Infra

- `spring-boot-starter-mail` (ou provider SMTP/Transactional).
- Token **aleatório** de uso único, **hash** no banco, expira em 30–60 min.

Endpoints

- `POST /auth/forgot-password` — sempre responde 200 (mesmo se e-mail não existe). Grava `password_resets(token_hash, user_id, expires_at, used_at)` e envia e-mail com link.
- `POST /auth/reset-password` — recebe `token` e `new_password`, valida janela de expiração, marca `used_at`, atualiza `password_hash` e **revoga todas as sessões ativas** do usuário (logout global).

Esquema

- `password_resets(id UUID, user_id, token_hash, expires_at, used_at)`

Aceite mínimo

- Tokens expiram corretamente e não podem ser reutilizados.
- Ao resetar, todas as sessões do usuário são revogadas.

Extensibilidade preparada

- Reuso do mesmo padrão (token aleatório + hash) para verificação de e-mail e magic links depois.

3) Gestão de sessões & renovação conservadora

Objetivo: dar visibilidade e controle de dispositivos sem introduzir JWT ainda.

Melhorias

- **Rotação de sessão no login** (opcional) ou **renovação explícita**:
  - `POST /auth/refresh` — emite **novo token de sessão** e invalida o anterior (token swapping), estendendo `expires_at` de modo seguro.
- **Lista e revogação de sessões**:
  - `GET /auth/sessions` — lista sessões (id, created/expires, ip, UA, last_used).
  - `DELETE /auth/sessions/{id}` — revoga sessão específica (logout remoto).

Esquema

- Acrescente `created_at`, `last_used_at` em `sessions`.

Aceite mínimo

- Refresh troca o cookie; sessão antiga é inválida.
- Usuário consegue encerrar outras sessões com 200/404/403 apropriados.

Extensibilidade preparada

- Trocar a camada de sessão por **JWT access (curto)** + **refresh opaco** depois é trivial: mantenha o contrato `/auth/refresh` e a tabela `sessions`.

4) Verificação de e-mail + hardening básico

Objetivo: reduzir fraude/abuso e fechar o “mínimo robusto”.

Funcionalidades

- **Verificação de e-mail**:

    - `POST /auth/verify-email/request` — (re)envia link com token.
    - `POST /auth/verify-email/confirm` — consome token, seta `email_verified_at`.
    - Bloqueie funcionalidades sensíveis para `email_verified_at IS NULL` (mas permita login).
- **Rate limiting** (conservador) no login/forgot:
    - Bucket por IP+email (ex.: 5/min) — usar filtro simples (Guava RateLimiter, Bucket4j) ou gateway.
- **Security headers** e pequenos reforços:
    - HSTS, `X-Content-Type-Options`, `X-Frame-Options`, `Content-Security-Policy` (mínima).
    - Mensagens de erro padronizadas; nunca diferenciar "e-mail não existe".
    - Logging de eventos de segurança: login ok/falho, reset, refresh, revoke.

Esquema

- `users` ganha `email_verified_at TIMESTAMP NULL`.
- `email_verifications(id, user_id, token_hash, expires_at, used_at)`

Aceite mínimo

- Usuário não verificado recebe 403 para rotas marcadas como "requiresVerified".
- Tentativas acima do limite retornam 429 com `Retry-After`.

Extensibilidade preparada

- Com verificação + rate limit implementados, você pode adicionar **MFA TOTP** como próximo passo sem alterar rotas existentes.

Stack e organização (sugestão)

- **Deps:** `spring-boot-starter-web`, `spring-boot-starter-security`, `spring-boot-starter-validation`, `spring-boot-starter-data-jpa`, `spring-boot-starter-mail`, `flyway-core`.
- **Pacotes:**
    - `domain` (entities, repos)
    - `security` (SecurityConfig, filters)
    - `auth` (controllers, dtos, service)

    - `mail` (templating/adapters)
    - `support` (rate-limit, error handling, audit)
- **Config sensível** via env/Secrets (SMTP, cookies, domain, HSTS, etc.).

Critérios gerais de pronto (para todas as tarefas)

- Testes de integração para **/register, /login, /me, /logout** e fluxos de **reset** e **refresh**.
- Cookies sempre `HttpOnly; Secure; SameSite=Strict`.
- `PasswordEncoder` configurado (BCrypt com custo atual).
- CSRF ativo (exija token em mutações quando for SPA com cookie).
- Respostas de erro com contrato uniforme (`code`, `message`, `details?`).