Validación de email (java)

1) Estructura del proyecto + Docker

PROMPT (pégalo tal cual)

Crea un proyecto Spring Boot 3 (Java 17) llamado "auth-service" con paquete base com.adoptafacil.auth. Agrega dependencias Maven:
- spring-boot-starter-web, spring-boot-starter-validation, spring-boot-starter-security, spring-boot-starter-mail, spring-boot-starter-actuator, spring-boot-starter-data-jpa, mysql-connector-j, lombok, springdoc-openapi-starter-webmvc-ui, caffeine, bucket4j-core, dnsjava, commons-validator, logstash-logback-encoder
- testing: spring-boot-starter-test, junit-jupiter, wiremock-jre8-standalone

Genera:
- controller GET /api/ping que devuelva {"pong":true, "ts": ISO8601}
- Actuator habilitado con /actuator/health
- OpenAPI en /swagger-ui y /v3/api-docs
- application.yml con placeholders EXACTOS:

```
spring:
  datasource:
    url: jdbc:mysql://db:3306/authdb?useSSL=false&serverTimezone=UTC
    username: ${DB_RO_USER:auth}
    password: ${DB_RO_PASS:secret}
  jpa:
    hibernate:
      ddl-auto: update
  mail:
    host: ${SMTP_HOST:mailhog}
    port: ${SMTP_PORT:1025}
    username: ${SMTP_USER:}
    password: ${SMTP_PASS:}
```

```
server:
  port: 8080
logging:
  level:
    root: INFO
  pattern:
    console: "%msg%n"
  logstash:
    enabled: true

email:
  provider: ${EMAIL_PROVIDER:smtp}   # smtp|resend|brevo|postmark
  from: ${EMAIL_FROM:no-reply@midominio.com}
  resend.apiKey: ${RESEND_API_KEY:}
  brevo.apiKey: ${BREVO_API_KEY:}
  postmark.apiKey: ${POSTMARK_API_KEY:}

emailVerifier:
  provider: ${EMAIL_VERIFIER_PROVIDER:inhouse} # inhouse|mailboxlayer|abs
tract
  disposableLists:
    - https://raw.githubusercontent.com/disposable-email-domains/disposable
-email-domains/master/disposable_email_domains.txt
    - https://raw.githubusercontent.com/amieiro/disposable-email-domains/mai
n/disposable_email_blocklist.json
  mailboxlayer.apiKey: ${MAILBOXLAYER_API_KEY:}
  abstract.apiKey: ${ABSTRACT_API_KEY:}

security:
  webhook:
    secret: ${WEBHOOK_SECRET:changeme}
jwt:
  publicKey: ${JWT_PUBLIC_KEY:}

- Dockerfile basado en eclipse-temurin:17-jdk con objetivo dev que lance 'mv
n spring-boot:run'
```

```
- docker-compose.yml con servicios:
  - db: mysql:8 (MYSQL_DATABASE=authdb, MYSQL_USER=auth, MYSQL_PAS
SWORD=secret, MYSQL_ROOT_PASSWORD=root)
  - mailhog: ports 1025,8025
  - app: build ., depends_on db y mailhog, envs mapeados de application.yml
- Configura Logback con logstash-logback-encoder para logs JSON en conso
la
```

Qué ejecutar para probar

```
mkdir auth-service && cd auth-service
git init
# si no generaste con Initializr, usa el pom creado por Copilot
docker compose up -d --build
curl -s http://localhost:8080/api/ping
curl -s http://localhost:8080/actuator/health
```

Criterios de aceptación

- `GET /api/ping` y `/actuator/health` → **200**.
- Swagger UI en `/swagger-ui/index.html`.

Rollback rápido

- `docker compose logs app` para detectar fallos.
- Si DB falla: comentar `ddl-auto` o usar H2 temporalmente.
- Levanta solo `mailhog` + `app` con H2 para aislar.

---

2) Seguridad básica: HMAC, rate limit y PIN

PROMPT

```
Implementa seguridad y antiabuso base:
```

1) Crea util HmacUtil con:
- sign(payload, secret): HMAC-SHA256 hex
- verify(signatureHex, payload, secret): boolean

2) Configura Bucket4j con filtros por endpoint:
- send-code: 5/min por IP y 3/min por email
- verify-code: 10/min por IP y 5/min por email
- reset: 3/min por IP
Responde 429 JSON: {"error":"rate_limited"} y expone métricas Micrometer (contadores por nombre).

3) Entidad JPA EmailCode:
- id (UUID), email, purpose (VERIFY|RESET), code_hash, expires_at (Instant), attempts (int), locked_until (Instant nullable), meta (TEXT JSON), created_at (Instant)
Repo EmailCodeRepository.

4) PinService:
- generate(email, purpose): genera PIN 6 dígitos, hashea con BCrypt o Argon2, TTL 10m configurable; persiste y emite log JSON event=pin_generated
- verify(email, purpose, code): valida lockout; compara hash; incrementa attempts; tras 5 fallos bloquea 15m (locked_until); logs event=pin_verified/pin_failed/locked
- cleanup job @Scheduled para expirados

5) SecurityConfig:
- Permitir públicos: /api/ping, /actuator/**, /v3/api-docs/**, /swagger-ui/**, /api/auth/**, /api/internal/**
- Añade filtro HMAC opcional para /api/internal/** con header X-Signature (usa HmacUtil.verify)
- Logs estructurados (ts, level, svc=auth-service, event, email, ip)
- Tests unitarios: expiración, intentos, lockout; verificación HMAC y 429

Qué ejecutar para probar

```
mvn -q -DskipTests=false test
```

Criterios de aceptación

- 5° intento fallido → bloqueo 15m.
- 429 en exceso de cuota con cuerpo `{"error":"rate_limited"}`.
- Logs JSON con eventos de PIN.

Rollback

- `rateLimit.enabled=false` por config para liberar tráfico temporalmente.

---

3) Endpoints + contratos OpenAPI

PROMPT

```
Crea controladores y DTOs con validaciones y ejemplos OpenAPI:

Endpoints:
- POST /api/auth/email/send-code
  body: { "email": "user@mail.com", "purpose": "verify|reset" }
  UX:
    purpose=reset:
      - si email existe → {"status":"email_enviado"}
      - si NO existe → {"status":"email_no_registrado"}
    purpose=verify:
      - por defecto → {"status":"email_enviado"}
      - si verify.strictExists=true:
        - si existe → {"status":"email_ya_registrado"}
        - si no → {"status":"email_enviado"}
  incluir meta: { "email_provider": "${email.provider}", "verifier_provider": "${e
mailVerifier.provider}" }

- POST /api/auth/email/verify-code
```

```
  body: { "email":"...", "purpose":"verify|reset", "code":"123456" }
  resp:
    - { "valid": true, "reset_token": "<si purpose=reset y válido>" }
    - { "valid": false }
  reset_token: JWT firmado o UUID con TTL 15m, one-time.

- POST /api/auth/password/reset
  body: { "email":"...", "reset_token":"...", "new_password":"..." }
  resp: { "ok": true }

- POST /api/internal/email/check
  body: { "email":"..." }
  resp: { "deliverable": true|false, "reason": "OK|INVALID_SYNTAX|NO_MX|DIS
POSABLE|PROVIDER_UNAVAILABLE" }

- POST /api/internal/users/exists
  body: { "email":"..." }
  resp: { "exists": true|false }

Añade @Operation y ejemplos OpenAPI; define schemas y ejemplos EXACTO
S.
Maneja errores con Problem+JSON estándar (application/problem+json).
Incluye tests de contrato mínimos para respuestas de ejemplo.
```

Qué ejecutar para probar

```
curl -s -X POST http://localhost:8080/api/auth/email/send-code \
 -H "Content-Type: application/json" \
 -d '{"email":"user@test.com","purpose":"verify"}'
```

Criterios de aceptación

- Respuestas coinciden exactamente con los ejemplos.
- /swagger-ui muestra schemas + examples.

Rollback

- `verify.strictExists=false` para modo “no enumeración”.

---

4) Modelado y persistencia (Providers y eventos)

PROMPT

Modela entidades y capas de email:

Entidades JPA:
- EmailCode(id, email, purpose, code_hash, expires_at, attempts, locked_until, meta TEXT, created_at)
- EmailEvent(id, email, event_type, payload TEXT JSON, created_at, provider, success BOOLEAN, latency_ms BIGINT)

Repos: EmailCodeRepository, EmailEventRepository.

Servicios:
- EmailProvider (Strategy) con implementaciones:
  - SmtpProvider (MailHog/dev via JavaMailSender)
  - ResendProvider, BrevoProvider, PostmarkProvider (usa WebClient; inyecta API keys desde config)
- EmailTemplateFactory: retorna plantilla por propósito (VERIFY/RESET)
- EmailContentBuilder: render HTML y text/plain con variables {code, expires_at, appName, supportEmail}

Registra EmailEvent en envíos (success/failure y latency).
Incluye @DataJpaTest para repos y tests de servicios con mocks de providers.

Qué ejecutar para probar

```
mvn -q -DskipTests=false test
```

Criterios de aceptación

- CRUD funciona y se persisten EmailEvent en envíos.
- Cambiar `email.provider` conmuta implementaciones.

Rollback

- Forzar `email.provider=smtp` (MailHog).

---

5) Plantillas de correo (verify/reset)

PROMPT

```
Crea plantillas:

Archivos:
- src/main/resources/templates/verify_email.html y verify_email.txt
- src/main/resources/templates/reset_password.html y reset_password.txt

Requisitos:
- Diseño responsive minimalista; CTA claro; muestra PIN grande; “expira en X min”.
- Versiones texto plano con PIN y expiración.
- EmailContentBuilder debe aceptar variables {code, expires_at, appName, supportEmail} y renderizar ambas variantes.

Tests:
- Unit tests del builder: verifica reemplazo correcto de variables (HTML y TXT).
```

Qué ejecutar para probar

```
curl -s -X POST http://localhost:8080/api/auth/email/send-code \
 -H "Content-Type: application/json" \
 -d '{"email":"dev@local.test","purpose":"verify"}'
# Revisa MailHog: http://localhost:8025
```

Criterios de aceptación

- Email visible en MailHog con HTML correcto y versión TXT.

Rollback

- Si Thymeleaf falla, usa StringSubstitutor/format temporal.

---

6) Integración con Laravel (exists + webhooks)

PROMPT

```
Integra verificación de existencia y webhooks:

UsersExistsService con 2 estrategias:
- HttpLaravelUsersExists: POST ${LARAVEL_URL}/internal/users/exists {emai
l} → boolean
- ReadOnlyDbUsersExists: SELECT EXISTS(SELECT 1 FROM users WHERE em
ail = ?) usando datasource RO
Config: users.exists.strategy = http|db

WebhookClient:
- postSigned(url, body, secret): firma body (HMAC-SHA256) y envía header X
-Signature
Eventos:
- email-verified: { email:"...", verified_at: ISO8601 }
- password-reset: { email:"...", reset_at: ISO8601 }

Reintentos con backoff exponencial; registra EmailEvent (event_type=WEBHO
```

OK_SENT, success, latency_ms).
Anota en OpenAPI el header X-Signature en descripción de webhooks.
Incluye tests de integración con WireMock para validar firma.

Qué ejecutar para probar (simulación de firma)

```
body='{"email":"user@test.com","verified_at":"2025-10-16T12:00:00Z"}'
sig=$(echo -n "$body" | openssl dgst -sha256 -hmac "changeme" -hex | cut -d" " -f2)
curl -X POST http://laravel.local/webhooks/email-verified \
 -H "Content-Type: application/json" -H "X-Signature: $sig" -d "$body"
```

Criterios de aceptación

- /internal/users/exists operativo según estrategia.
- Webhooks enviados y firmados; WireMock verifica header.

Rollback

- Cambia a estrategia `db` si Laravel no está disponible.
- Desactiva envío y deja reintentos en cola.

---

7) Tests (unitarios, integración, contrato)

PROMPT

Crea suite de pruebas:

Unit:
- PinServiceTest: genera/verifica PIN; simula expiración; cuenta intentos; lock out a los 5 fallos.
- HmacUtilTest: sign/verify con casos válidos y firma incorrecta.
- EmailContentBuilderTest: placeholders en HTML/TXT.

```
Integración (SpringBootTest + Testcontainers MySQL):
- AuthFlowIT:
  1) send-code (purpose=reset con usuario existente) → {"status":"email_enviado"}
  2) verify-code correcto → { "valid": true, "reset_token": "..." }
  3) password/reset con token → { "ok": true }

Contrato (WireMock):
- WebhookContractTest: intercepta email-verified/password-reset y valida X-Signature.

RateLimit:
- RateLimitTest: dispara 10x send-code y espera 429 con {"error":"rate_limited"}.

Objetivo: cobertura >= 80% en PinService, EmailVerificationService, WebhookClient. Genera reportes surefire y jacoco.
```

Qué ejecutar para probar

```
mvn -q -DskipTests=false test
```

Criterios de aceptación

- Tests verdes; cobertura ≥ 80% en clases clave.

Rollback

- Perfil `Pno-testcontainers` y H2 si hay problemas con MySQL en CI.

---

8) Observabilidad (Actuator, métricas, logs JSON, auditoría)

PROMPT

```
Añade observabilidad:

- Micrometer: counters y timers en puntos clave:
  emails.sent, emails.failed, pins.generated, pins.verified, rate_limit.hits, smtp.failures
- Logback JSON: ts, level, svc="auth-service", event, email, provider, latency_ms, ip, traceId
- GET /api/audits/findings?from=&to=:
  - Devuelve agregados simples desde EmailEvent: conteo por event_type, por provider y tasa de éxito.
  - Validar parámetros y retornar 200 con payload JSON.

Tests:
- Métricas: incrementar y consultar vía MeterRegistry mock/Spy.
- Logs: appender de test que verifique campo "event".
```

Qué ejecutar para probar

```
curl -s http://localhost:8080/actuator/metrics | jq .
curl -s "http://localhost:8080/api/audits/findings?from=2025-10-01&to=2025-10-31"
```

Criterios de aceptación

- Métricas visibles; logs JSON estructurados.
- /api/audits/findings responde 200 con agregados.

Rollback

- Deshabilitar endpoint si no se usa en prod (feature flag).

---

9) Seguridad y anti-abuso (listas desechables, no-enumeración)

PROMPT

```
Completa antiabuso:

Config:
rateLimit:
  sendCode:
    perIpPerMin: 5
    perEmailPerMin: 3
  verifyCode:
    perIpPerMin: 10
    perEmailPerMin: 5
  reset:
    perIpPerMin: 3
verify:
  strictExists: ${VERIFY_STRICT:false}

EmailVerifier in-house:
- Valida sintaxis (commons-validator), consulta MX (dnsjava), y verifica domin
io en listas desechables.
- Scheduled diario: descarga listas de disposable (URLs de application.yml); c
achea en Caffeine y persiste evento en EmailEvent (event_type=DISPOSABLE_
SYNC). Si fallan fuentes, conserva cache anterior.

Endpoint /api/internal/email/check debe retornar:
- deliverable=false y reason=DISPOSABLE si el dominio está en lista.

Documenta recomendaciones: CAPTCHA (hCaptcha/Turnstile) y WAF (Cloudfl
are) para prod.

Tests:
- Caso dominio desechable: reason=DISPOSABLE
- Caso sin MX: reason=NO_MX
```

Qué ejecutar para probar

```
curl -s -X POST http://localhost:8080/api/internal/email/check \
 -H "Content-Type: application/json" \
 -d '{"email":"temp@dispostable.com"}'
```

Criterios de aceptación

- 429 cuando se exceden límites configurados.
- Dominios desechables → deliverable=false.

Rollback

- Dejar `emailVerifier.provider=inhouse` con `disposableLists=[]` temporalmente.

---

10) Entrega final: README, envs, cURLs, Postman, despliegue

PROMPT

```
Genera artefactos de entrega:

1) README.md:
- Visión general y arquitectura (diagrama simple ASCII o mermaid opcional)
- Cómo correr en dev: docker compose up -d --build
- Endpoints con ejemplos de request/response EXACTOS
- cURLs de verificación de todos los flujos
- Variables de entorno y .env.example
- Seguridad: HMAC, rate limits, no-enumeración, recomendaciones CAPTCH
A/WAF
- Observabilidad: Actuator, métricas, logs JSON
- Troubleshooting y rollback rápidos
- Guía de despliegue docker (staging/prod): variables mínimas, healthchecks,
estrategias

2) .env.example con:
```

```
DB_RO_USER, DB_RO_PASS, SMTP_HOST, SMTP_PORT, SMTP_USER, SMTP_
PASS, EMAIL_PROVIDER, EMAIL_FROM, RESEND_API_KEY, BREVO_API_KEY,
POSTMARK_API_KEY, EMAIL_VERIFIER_PROVIDER, MAILBOXLAYER_API_KEY,
ABSTRACT_API_KEY, WEBHOOK_SECRET, JWT_PUBLIC_KEY, LARAVEL_URL,
USERS_EXISTS_STRATEGY, VERIFY_STRICT

3) Makefile:
- build, test, up, down, logs

4) Postman collection JSON en docs/postman/auth-service.postman_collectio
n.json
- Folders: Auth (send-code, verify-code, reset), Internal (email/check, users/e
xists), Webhooks
- Variables de entorno para host y WEBHOOK_SECRET

5) docs/checklist.md:
- Criterios de aceptación por endpoint
- Pasos de rollback rápido por bloque (1-10)
```

Qué ejecutar para probar

```
make build
make test
make up
# cURLs de README (send-code, verify-code, reset, internal/*)
```

Criterios de aceptación

- README y .env.example completos.
- Postman importable y funcional.
- Makefile operativo.

Rollback

- Forzar EMAIL_PROVIDER=smtp y EMAIL_VERIFIER_PROVIDER=inhouse.

- verify.strictExists=false y límites reducidos.
- Pausar webhooks con cola de reintentos.

---

cURLs de smoke test (rápidos)

```
curl -s http://localhost:8080/api/ping
curl -s http://localhost:8080/actuator/health
curl -s -X POST http://localhost:8080/api/auth/email/send-code -H "Content-Type: application/json" -d '{"email":"user@test.com","purpose":"verify"}'
curl -s -X POST http://localhost:8080/api/internal/email/check -H "Content-Type: application
```