Okay, let's provide a complete, refreshed solution for your mTLS Certificate API using Spring Boot and GCP CAS, incorporating external/internal API separation and robust fault tolerance with Resilience4j.

This solution provides a "Certificate as a Service" (CaaS) API where your B2B customers can request mTLS client certificates, issued from your dedicated GCP CA B2B Pool.

## Complete Solution Refresh: mTLS Certificate API with GCP CAS & Fault Tolerance

### 1. Project Structure

mtls-gcp-cas-caas/  
├── src/  
│ ├── main/  
│ │ ├── java/  
│ │ │ └── com/  
│ │ │ └── example/  
│ │ │ └── caas/  
│ │ │ ├── CaaSApiApplication.java // Main Spring Boot App  
│ │ │ ├── api/  
│ │ │ │ ├── CertificateRequestController.java // External API Endpoint  
│ │ │ │ ├── requests/  
│ │ │ │ │ └── CertificateIssueRequest.java // External Request DTO  
│ │ │ │ └── responses/  
│ │ │ │ └── CertificateIssueResponse.java // External Response DTO  
│ │ │ ├── security/  
│ │ │ │ ├── ApiKeyAuthFilter.java // Simple API Key Filter  
│ │ │ │ └── WebConfig.java // Security Config  
│ │ │ └── service/  
│ │ │ └── CertificateIssuanceService.java // Core Logic, GCP CAS Interaction, Fault Tolerance  
│ │ └── resources/  
│ │ └── application.properties // Configuration  
│ └── test/  
│ └── java/  
│ └── com/  
│ └── example/  
│ └── caas/  
│ └── MtlsGcpCasCaasApplicationTests.java // Basic Spring Boot Test  
└── pom.xml // Maven Dependencies

### 2. pom.xml

This pom.xml includes all necessary Spring Boot starters, Google Cloud Private CA client library, BouncyCastle for CSR handling, and Resilience4j for fault tolerance.

XML

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.2.7</version> <relativePath/> </parent>  
 <groupId>com.example</groupId>  
 <artifactId>mtls-gcp-cas-caas</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>mtls-gcp-cas-caas</name>  
 <description>Certificate as a Service API with GCP CAS</description>  
  
 <properties>  
 <java.version>17</java.version>  
 <google-cloud-private-ca.version>1.5.0</google-cloud-private-ca.version> <resilience4j.version>2.2.0</resilience4j.version> </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-validation</artifactId> </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-aop</artifactId> </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId> </dependency>  
  
 <dependency>  
 <groupId>com.google.cloud</groupId>  
 <artifactId>google-cloud-privateca</artifactId>  
 <version>${google-cloud-private-ca.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>com.google.auth</groupId>  
 <artifactId>google-auth-library-oauth2-http</artifactId>  
 <version>1.23.0</version> </dependency>  
  
 <dependency>  
 <groupId>org.bouncycastle</groupId>  
 <artifactId>bcpkix-jdk15on</artifactId>  
 <version>1.70</version>  
 </dependency>  
 <dependency>  
 <groupId>org.bouncycastle</groupId>  
 <artifactId>bcprov-jdk15on</artifactId>  
 <version>1.70</version>  
 </dependency>  
  
 <dependency>  
 <groupId>io.github.resilience4j</groupId>  
 <artifactId>resilience4j-spring-boot3</artifactId>  
 <version>${resilience4j.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>io.github.resilience4j</groupId>  
 <artifactId>resilience4j-micrometer</artifactId>  
 <version>${resilience4j.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

### 3. src/main/resources/application.properties

This file configures your GCP CAS details, API security (simplified), and Resilience4j patterns.

Properties

# Spring Boot Server Port  
server.port=8080  
  
# GCP Project ID and CAS CA Pool details for the B2B API  
gcp.project-id=your-gcp-project-id  
gcp.cas.location=us-central1 # Or your region, e.g., asia-south1 for Mumbai  
gcp.cas.b2b-ca-pool-id=your-b2b-ca-pool-id  
gcp.cas.certificate-template-name=your-b2b-client-template # Recommended to use a template  
  
# API Security (Simplified: ONLY for demo. Use Spring Security + OAuth2/JWT in production)  
api.security.enabled=true  
api.security.allowed-api-keys=customer-abc-apikey,customer-xyz-apikey # Store securely (e.g., Google Secret Manager)  
  
# Certificate specific defaults/limits  
mtls.certificate.default-validity-days=365 # Default if customer doesn't provide  
mtls.certificate.max-validity-days=730 # Maximum allowed  
  
# Resilience4j Configuration for GCP CAS calls  
# Defines a circuit breaker, retry, and time limiter named 'gcpCasApi'  
  
# Circuit Breaker Defaults  
resilience4j.circuitbreaker.configs.default.slidingWindowType=COUNT\_BASED  
resilience4j.circuitbreaker.configs.default.slidingWindowSize=10  
resilience4j.circuitbreaker.configs.default.failureRateThreshold=50 # 50% failures in window to open circuit  
resilience4j.circuitbreaker.configs.default.waitDurationInOpenState=60s # Open for 60 seconds  
resilience4j.circuitbreaker.configs.default.permittedNumberOfCallsInHalfOpenState=3 # 3 calls in half-open state  
resilience4j.circuitbreaker.configs.default.recordExceptions=com.google.api.gax.rpc.UnavailableException,com.google.api.gax.rpc.DeadlineExceededException,java.io.IOException # Exceptions that mark a call as failure  
  
# Specific Circuit Breaker for GCP CAS API  
resilience4j.circuitbreaker.instances.gcpCasApi.baseConfig=default  
  
# Retry Defaults  
resilience4j.retry.configs.default.maxAttempts=5 # Max 5 retries (initial attempt + 4 retries)  
resilience4j.retry.configs.default.waitDuration=1s # Initial wait of 1 second before first retry  
resilience4j.retry.configs.default.exponentialBackoffMultiplier=2 # Exponential backoff: 1s, 2s, 4s, 8s...  
resilience4j.retry.configs.default.retryExceptions=com.google.api.gax.rpc.UnavailableException,com.google.api.gax.rpc.DeadlineExceededException,com.google.api.gax.rpc.InternalException,com.google.api.gax.rpc.ResourceExhaustedException,java.io.IOException # Specific transient exceptions to retry  
  
# Specific Retry for GCP CAS API  
resilience4j.retry.instances.gcpCasApi.baseConfig=default  
  
# Time Limiter Defaults (for setting explicit timeouts on decorated methods)  
# This timeout applies to a single execution attempt, including potential retries within this limit.  
# The overall execution time can exceed this if multiple retries happen.  
# For overall operation timeout, you might manage it at the CompletableFuture level or higher.  
resilience4j.timelimiter.configs.default.timeoutDuration=30s # 30-second timeout for each attempt  
resilience4j.timelimiter.instances.gcpCasApi.baseConfig=default  
  
# Actuator Endpoints (for monitoring Resilience4j metrics, health)  
management.endpoints.web.exposure.include=health,prometheus,circuitbreakers,retries,timelimiters  
management.endpoint.health.show-details=always

### 4. Java Code

#### 4.1. src/main/java/com/example/caas/CaaSApiApplication.java

Java

package com.example.caas;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class CaaSApiApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.run(CaaSApiApplication.class, args);  
 }  
  
}

#### 4.2. src/main/java/com/example/caas/security/ApiKeyAuthFilter.java

(Simplified for demo - **DO NOT USE IN PRODUCTION AS-IS**)

Java

package com.example.caas.security;  
  
import jakarta.servlet.FilterChain;  
import jakarta.servlet.ServletException;  
import jakarta.servlet.ServletRequest;  
import jakarta.servlet.ServletResponse;  
import jakarta.servlet.http.HttpServletRequest;  
import jakarta.servlet.http.HttpServletResponse;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
import org.springframework.web.filter.GenericFilterBean;  
  
import java.io.IOException;  
import java.util.Arrays;  
import java.util.HashSet;  
import java.util.Set;  
  
@Component  
public class ApiKeyAuthFilter extends GenericFilterBean {  
  
 @Value("${api.security.enabled:false}")  
 private boolean securityEnabled;  
  
 @Value("${api.security.allowed-api-keys:}")  
 private String allowedApiKeysCsv;  
  
 private Set<String> allowedApiKeys;  
  
 @Override  
 public void initFilterBean() throws ServletException {  
 if (securityEnabled && allowedApiKeysCsv != null && !allowedApiKeysCsv.isEmpty()) {  
 this.allowedApiKeys = new HashSet<>(Arrays.asList(allowedApiKeysCsv.split(",")));  
 } else {  
 this.allowedApiKeys = new HashSet<>();  
 }  
 }  
  
 @Override  
 public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  
 throws IOException, ServletException {  
  
 HttpServletRequest httpRequest = (HttpServletRequest) request;  
 HttpServletResponse httpResponse = (HttpServletResponse) response;  
  
 // Bypass security for actuator endpoints (e.g., /actuator/health)  
 if (httpRequest.getRequestURI().startsWith("/actuator") && securityEnabled) {  
 chain.doFilter(request, response);  
 return;  
 }  
  
 if (!securityEnabled) {  
 chain.doFilter(request, response); // Bypass security if disabled  
 return;  
 }  
  
 String apiKey = httpRequest.getHeader("X-API-Key"); // Or Authorization: Bearer <key>  
  
 if (apiKey != null && allowedApiKeys.contains(apiKey)) {  
 // In a real app, map API key to customer ID or roles (e.g., using Spring Security Context)  
 chain.doFilter(request, response);  
 } else {  
 httpResponse.sendError(HttpServletResponse.SC\_UNAUTHORIZED, "Unauthorized: Invalid or missing API Key");  
 }  
 }  
}

#### 4.3. src/main/java/com/example/caas/security/WebConfig.java

Java

package com.example.caas.security;  
  
import org.springframework.boot.web.servlet.FilterRegistrationBean;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class WebConfig {  
  
 private final ApiKeyAuthFilter apiKeyAuthFilter;  
  
 public WebConfig(ApiKeyAuthFilter apiKeyAuthFilter) {  
 this.apiKeyAuthFilter = apiKeyAuthFilter;  
 }  
  
 @Bean  
 public FilterRegistrationBean<ApiKeyAuthFilter> apiKeyFilterRegistration() {  
 FilterRegistrationBean<ApiKeyAuthFilter> registration = new FilterRegistrationBean<>();  
 registration.setFilter(apiKeyAuthFilter);  
 registration.addUrlPatterns("/api/v1/certificates/\*"); // Apply to specific API paths  
 // Exclude actuator paths from the API key filter, as they are handled above  
 // For production, actuator endpoints should also be secured separately.  
 registration.setOrder(1); // Set order if multiple filters  
 return registration;  
 }  
}

#### 4.4. src/main/java/com/example/caas/api/requests/CertificateIssueRequest.java (External DTO)

Java

package com.example.caas.api.requests;  
  
import jakarta.validation.constraints.NotBlank;  
import jakarta.validation.constraints.Pattern;  
import jakarta.validation.constraints.Size;  
  
import java.util.List;  
import java.util.Map;  
  
public class CertificateIssueRequest {  
  
 @NotBlank(message = "CSR PEM cannot be empty.")  
 @Pattern(regexp = "^-----BEGIN CERTIFICATE REQUEST-----[\\s\\S]\*-----END CERTIFICATE REQUEST-----\\s\*$",  
 message = "Invalid CSR PEM format.")  
 private String csrPem;  
  
 @NotBlank(message = "Customer ID cannot be empty.")  
 @Size(min = 1, max = 255, message = "Customer ID must be between 1 and 255 characters.")  
 private String customerId; // Used for internal tracking and potentially for policy enforcement  
  
 private Integer requestedValidityDays; // Optional: Customer can request, but API will enforce max  
  
 private List<String> dnsNames; // Optional Subject Alternative Names (DNS)  
  
 private List<String> ipAddresses; // Optional Subject Alternative Names (IP)  
  
 private Map<String, String> metadata; // Optional: For customer-specific metadata to store with cert  
  
 // Getters and Setters  
 public String getCsrPem() {  
 return csrPem;  
 }  
  
 public void setCsrPem(String csrPem) {  
 this.csrPem = csrPem;  
 }  
  
 public String getCustomerId() {  
 return customerId;  
 }  
  
 public void setCustomerId(String customerId) {  
 this.customerId = customerId;  
 }  
  
 public Integer getRequestedValidityDays() {  
 return requestedValidityDays;  
 }  
  
 public void setRequestedValidityDays(Integer requestedValidityDays) {  
 this.requestedValidityDays = requestedValidityDays;  
 }  
  
 public List<String> getDnsNames() {  
 return dnsNames;  
 }  
  
 public void setDnsNames(List<String> dnsNames) {  
 this.dnsNames = dnsNames;  
 }  
  
 public List<String> getIpAddresses() {  
 return ipAddresses;  
 }  
  
 public void setIpAddresses(List<String> ipAddresses) {  
 this.ipAddresses = ipAddresses;  
 }  
  
 public Map<String, String> getMetadata() {  
 return metadata;  
 }  
  
 public void setMetadata(Map<String, String> metadata) {  
 this.metadata = metadata;  
 }  
}

#### 4.5. src/main/java/com/example/caas/api/responses/CertificateIssueResponse.java (External DTO)

Java

package com.example.caas.api.responses;  
  
public class CertificateIssueResponse {  
 private String status; // "success" or "error"  
 private String certificateId; // GCP CAS Certificate resource name  
 private String certificatePem; // The issued certificate + full chain in PEM format  
 private String message; // Human-readable message  
  
 // Constructors  
 public CertificateIssueResponse(String status, String certificateId, String certificatePem, String message) {  
 this.status = status;  
 this.certificateId = certificateId;  
 this.certificatePem = certificatePem;  
 this.message = message;  
 }  
  
 public CertificateIssueResponse(String status, String message) {  
 this.status = status;  
 this.message = message;  
 }  
  
 // Getters and Setters  
 public String getStatus() {  
 return status;  
 }  
  
 public void setStatus(String status) {  
 this.status = status;  
 }  
  
 public String getCertificateId() {  
 return certificateId;  
 }  
  
 public void setCertificateId(String certificateId) {  
 this.certificateId = certificateId;  
 }  
  
 public String getCertificatePem() {  
 return certificatePem;  
 }  
  
 public void setCertificatePem(String certificatePem) {  
 this.certificatePem = certificatePem;  
 }  
  
 public String getMessage() {  
 return message;  
 }  
  
 public void setMessage(String message) {  
 this.message = message;  
 }  
}

#### 4.6. src/main/java/com/example/caas/service/CertificateIssuanceService.java (Core Logic with Fault Tolerance)

This service manages the mapping from external request to internal GCP CAS calls and applies Resilience4j.

Java

package com.example.caas.service;  
  
import com.google.api.gax.core.FixedCredentialsProvider;  
import com.google.auth.oauth2.GoogleCredentials;  
import com.google.cloud.security.privateca.v1.\*;  
import com.example.caas.api.requests.CertificateIssueRequest;  
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;  
import io.github.resilience4j.retry.annotation.Retry;  
import io.github.resilience4j.timelimiter.annotation.TimeLimiter;  
import org.bouncycastle.util.io.pem.PemObject;  
import org.bouncycastle.util.io.pem.PemReader;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Service;  
  
import java.io.IOException;  
import java.io.StringReader;  
import java.security.Security;  
import java.time.Duration;  
import java.util.concurrent.CompletableFuture;  
import java.util.stream.Collectors;  
  
import org.bouncycastle.jce.provider.BouncyCastleProvider;  
import org.bouncycastle.pkcs.PKCS10CertificationRequest;  
  
@Service  
public class CertificateIssuanceService {  
  
 private static final Logger log = LoggerFactory.getLogger(CertificateIssuanceService.class);  
  
 @Value("${gcp.project-id}")  
 private String projectId;  
  
 @Value("${gcp.cas.location}")  
 private String casLocation;  
  
 @Value("${gcp.cas.b2b-ca-pool-id}")  
 private String b2bCaPoolId;  
  
 @Value("${gcp.cas.certificate-template-name:default}")  
 private String certificateTemplateName;  
  
 @Value("${mtls.certificate.default-validity-days:365}")  
 private int defaultValidityDays;  
  
 @Value("${mtls.certificate.max-validity-days:730}")  
 private int maxValidityDays;  
  
 static {  
 Security.addProvider(new BouncyCastleProvider());  
 }  
  
 /\*\*  
 \* Issues a client certificate using GCP CAS based on an external API request.  
 \* This method performs the mapping from the external DTO to the internal GCP CAS API objects.  
 \*  
 \* Fault Tolerance:  
 \* - @CircuitBreaker("gcpCasApi"): Prevents repeated calls to an unhealthy GCP CAS.  
 \* - @Retry("gcpCasApi"): Retries transient failures for GCP CAS API calls.  
 \* - @TimeLimiter("gcpCasApi"): Ensures the operation completes within a defined timeout for each attempt.  
 \*  
 \* @param externalRequest The external CertificateIssueRequest DTO from the customer.  
 \* @return A CompletableFuture that completes with the issued GCP CAS Certificate object.  
 \* @throws IllegalArgumentException if the request is invalid.  
 \* @throws RuntimeException if an internal error occurs after retries/circuit breaker limits are exhausted.  
 \*/  
 @CircuitBreaker(name = "gcpCasApi", fallbackMethod = "issueClientCertificateFallback")  
 @Retry(name = "gcpCasApi", fallbackMethod = "issueClientCertificateFallback")  
 @TimeLimiter(name = "gcpCasApi")  
 public CompletableFuture<com.google.cloud.security.privateca.v1.Certificate> issueClientCertificate(  
 CertificateIssueRequest externalRequest) {  
  
 // Wrap the synchronous execution in a CompletableFuture.supplyAsync  
 // to integrate with @TimeLimiter and other async features of Resilience4j  
 return CompletableFuture.supplyAsync(() -> {  
 // --- 1. Validate and Map External Request to Internal Logic/Policy ---  
 // These validations happen BEFORE the Resilience4j mechanisms,  
 // as they are client-side errors and should fail fast.  
 if (externalRequest.getCsrPem() == null || externalRequest.getCsrPem().isEmpty()) {  
 throw new IllegalArgumentException("CSR PEM is required.");  
 }  
 if (externalRequest.getCustomerId() == null || externalRequest.getCustomerId().isEmpty()) {  
 throw new IllegalArgumentException("Customer ID is required.");  
 }  
  
 PKCS10CertificationRequest csr;  
 try {  
 csr = parseCsrFromPem(externalRequest.getCsrPem());  
 if (csr == null) {  
 throw new IllegalArgumentException("Invalid CSR PEM provided or not a valid CSR.");  
 }  
 } catch (IOException e) {  
 throw new IllegalArgumentException("Failed to parse CSR PEM: " + e.getMessage(), e);  
 }  
  
 int requestedDays = externalRequest.getRequestedValidityDays() != null  
 ? externalRequest.getRequestedValidityDays()  
 : defaultValidityDays;  
  
 if (requestedDays <= 0 || requestedDays > maxValidityDays) {  
 throw new IllegalArgumentException(  
 String.format("Requested validity days must be between 1 and %d.", maxValidityDays));  
 }  
  
 // --- 2. Authenticate with GCP and prepare GCP CAS client ---  
 GoogleCredentials credentials;  
 try {  
 credentials = GoogleCredentials.getApplicationDefault();  
 } catch (IOException e) {  
 // This is a critical setup error, usually not transient. Fail fast.  
 throw new RuntimeException("Failed to obtain GCP credentials. Check application environment.", e);  
 }  
  
 try (CertificateAuthorityServiceClient client = CertificateAuthorityServiceClient.newBuilder()  
 .setCredentialsProvider(FixedCredentialsProvider.create(credentials))  
 // For more granular control over timeouts per RPC, you'd configure  
 // ClientContext.Builder and Endpoint/TransportChannelProvider here.  
 // Resilience4j's @TimeLimiter acts as an overall execution timeout.  
 .build()) {  
  
 String caPoolName = String.format("projects/%s/locations/%s/caPools/%s", projectId, casLocation, b2bCaPoolId);  
  
 // --- 3. Construct Internal GCP CAS Request (IssueCertificateRequest) from external DTO ---  
 CertificateConfig.Builder certConfigBuilder = CertificateConfig.newBuilder()  
 .setPublicKey(PublicKey.newBuilder()  
 .setFormat(PublicKey.KeyFormat.PEM\_CSR)  
 .setKey(com.google.protobuf.ByteString.copyFromUtf8(externalRequest.getCsrPem()))  
 .build());  
  
 // Add Subject Alt Names from external request if provided  
 if ((externalRequest.getDnsNames() != null && !externalRequest.getDnsNames().isEmpty()) ||  
 (externalRequest.getIpAddresses() != null && !externalRequest.getIpAddresses().isEmpty())) {  
  
 SubjectAltNames.Builder sanBuilder = SubjectAltNames.newBuilder();  
 if (externalRequest.getDnsNames() != null) {  
 sanBuilder.addAllDnsNames(externalRequest.getDnsNames());  
 }  
 if (externalRequest.getIpAddresses() != null) {  
 sanBuilder.addAllIpAddresses(externalRequest.getIpAddresses());  
 }  
  
 certConfigBuilder.setSubjectConfig(CertificateConfig.SubjectConfig.newBuilder()  
 .setSubjectAltName(sanBuilder.build())  
 .build());  
 }  
  
 IssueCertificateRequest.Builder casIssueRequestBuilder = IssueCertificateRequest.newBuilder()  
 .setParent(caPoolName)  
 .setCertificateConfig(certConfigBuilder.build())  
 .setLifetime(com.google.protobuf.Duration.newBuilder().setSeconds(Duration.ofDays(requestedDays).getSeconds()));  
  
 // Use a specific certificate template if configured (highly recommended for B2B policies)  
 if (certificateTemplateName != null && !certificateTemplateName.equalsIgnoreCase("default")) {  
 String templatePath = String.format("projects/%s/locations/%s/certificateTemplates/%s", projectId, casLocation, certificateTemplateName);  
 casIssueRequestBuilder.setCertificateTemplate(templatePath);  
 }  
  
 // --- Idempotency Key (crucial for retries) ---  
 // If a request times out or fails after the CAS call is received,  
 // retrying with the same request ID ensures CAS doesn't issue a duplicate.  
 // It should be unique per \*logical\* request by the customer.  
 // A hash of the CSR + customer ID + validity is a reasonable approach  
 // if the customer doesn't provide their own unique request ID.  
 String uniqueRequestId = "customer-" + externalRequest.getCustomerId() +  
 "-csrhash-" + externalRequest.getCsrPem().hashCode() +  
 "-validity-" + requestedDays; // Unique per customer, CSR content, and validity  
 casIssueRequestBuilder.setRequestId(uniqueRequestId);  
  
 // Add labels for better tracking in GCP CAS  
 if (externalRequest.getMetadata() != null) {  
 casIssueRequestBuilder.putAllLabels(externalRequest.getMetadata());  
 }  
 casIssueRequestBuilder.putLabels("customer\_id", externalRequest.getCustomerId());  
 casIssueRequestBuilder.putLabels("api\_request\_hash", String.valueOf(externalRequest.hashCode())); // Track content of the external API request  
  
 log.info("Attempting to issue certificate for customer '{}' from CA Pool '{}'. Unique Request ID: {}",  
 externalRequest.getCustomerId(), caPoolName, uniqueRequestId);  
  
 // --- 4. Execute Internal GCP CAS API Call ---  
 com.google.cloud.security.privateca.v1.Certificate issuedCertificate = client.issueCertificate(casIssueRequestBuilder.build());  
  
 log.info("Certificate issued successfully from CAS. Internal GCP CAS Certificate Name: {}", issuedCertificate.getName());  
  
 // --- 5. Return Internal GCP CAS Certificate object ---  
 // The controller will then map this internal object to the external response DTO.  
 return issuedCertificate;  
  
 } catch (Exception e) {  
 // Resilience4j will intercept this. If retries are available, it will retry.  
 // If the circuit is open or retries are exhausted, the fallback will be triggered.  
 log.error("Error during GCP CAS certificate issuance for customer {}. Error: {}", externalRequest.getCustomerId(), e.getMessage(), e);  
 // Re-throw to allow Resilience4j to handle  
 throw new RuntimeException("GCP CAS communication failed: " + e.getMessage(), e);  
 }  
 });  
 }  
  
 /\*\*  
 \* Fallback method for issueClientCertificate in case of Circuit Breaker open, Retries exhausted, or Timeout.  
 \* This provides a graceful degradation for the API.  
 \*  
 \* @param externalRequest The original request that caused the failure.  
 \* @param throwable The exception that triggered the fallback.  
 \* @return A CompletableFuture that completes exceptionally with a more descriptive error.  
 \*/  
 private CompletableFuture<com.google.cloud.security.privateca.v1.Certificate> issueClientCertificateFallback(  
 CertificateIssueRequest externalRequest, Throwable throwable) {  
 log.error("Fallback triggered for certificate issuance for customer {}. Reason: {}", externalRequest.getCustomerId(), throwable.getMessage(), throwable);  
  
 // Customize the exception based on the type of throwable  
 if (io.github.resilience4j.circuitbreaker.CallNotPermittedException.class.isInstance(throwable)) {  
 // Circuit breaker is open - fail fast and inform client  
 return CompletableFuture.failedFuture(new RuntimeException("Certificate issuance service is currently overloaded or unavailable. Please try again later."));  
 } else if (java.util.concurrent.TimeoutException.class.isInstance(throwable)) {  
 // Timeout occurred - indicates long-running operation or external service lag  
 return CompletableFuture.failedFuture(new RuntimeException("Certificate issuance request timed out. The operation might still be in progress on the server. Please try again or check status later."));  
 } else {  
 // Other errors after retries exhausted (e.g., persistent connectivity issues, unhandled GCP errors)  
 // Log the original exception for debugging, but provide a generic message to the customer.  
 return CompletableFuture.failedFuture(new RuntimeException("Failed to issue certificate after multiple attempts due to an internal error. Please contact support."));  
 }  
 }  
  
 /\*\*  
 \* Helper method to parse a PEM-encoded CSR into a BouncyCastle PKCS10CertificationRequest object.  
 \* @param pemCsr The PEM encoded CSR string.  
 \* @return PKCS10CertificationRequest object.  
 \* @throws IOException if parsing fails.  
 \*/  
 private PKCS10CertificationRequest parseCsrFromPem(String pemCsr) throws IOException {  
 try (PemReader reader = new PemReader(new StringReader(pemCsr))) {  
 PemObject pemObject = reader.readPemObject();  
 if (pemObject == null || !"CERTIFICATE REQUEST".equals(pemObject.getType())) {  
 log.error("Invalid PEM object type: Expected 'CERTIFICATE REQUEST', got '{}'", pemObject != null ? pemObject.getType() : "null");  
 return null;  
 }  
 return new PKCS10CertificationRequest(pemObject.getContent());  
 }  
 }  
}

#### 4.7. src/main/java/com/example/caas/api/CertificateRequestController.java (External API Endpoint)

This controller exposes the API and handles the mapping of external DTOs to/from the service layer.

Java

package com.example.caas.api;  
  
import com.example.caas.api.requests.CertificateIssueRequest;  
import com.example.caas.api.responses.CertificateIssueResponse;  
import com.example.caas.service.CertificateIssuanceService;  
import jakarta.validation.Valid;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.concurrent.CompletionException; // For unwrapping CompletableFuture exceptions  
import java.util.stream.Collectors;  
  
@RestController  
@RequestMapping("/api/v1/certificates")  
public class CertificateRequestController {  
  
 private static final Logger log = LoggerFactory.getLogger(CertificateRequestController.class);  
  
 private final CertificateIssuanceService issuanceService;  
  
 public CertificateRequestController(CertificateIssuanceService issuanceService) {  
 this.issuanceService = issuanceService;  
 }  
  
 /\*\*  
 \* API endpoint for customers to request a new mTLS client certificate.  
 \*  
 \* @param request The external CertificateIssueRequest DTO containing CSR and other details.  
 \* @return ResponseEntity with CertificateIssueResponse (success or error).  
 \*/  
 @PostMapping("/issue")  
 public ResponseEntity<CertificateIssueResponse> issueCertificate(@Valid @RequestBody CertificateIssueRequest request) {  
 try {  
 log.info("Received external API request for certificate issuance for customer: {}", request.getCustomerId());  
  
 // Delegate to the service layer. The service will handle GCP CAS interaction  
 // and fault tolerance (retries, circuit breaker, timeout).  
 com.google.cloud.security.privateca.v1.Certificate issuedInternalCert =  
 issuanceService.issueClientCertificate(request).join(); // .join() blocks and propagates exceptions  
  
 // --- Map Internal GCP CAS Certificate object to External API Response DTO ---  
 // Combine the issued certificate and its chain for the customer response.  
 // The customer needs the full chain to establish trust with your B2B server.  
 String fullCertificateChainPem = issuedInternalCert.getPemCertificate() + "\n" +  
 issuedInternalCert.getPemCertificateChainList().stream()  
 .collect(Collectors.joining("\n"));  
  
 CertificateIssueResponse response = new CertificateIssueResponse(  
 "success",  
 issuedInternalCert.getName(), // Return the GCP CAS resource name as an ID for future reference (e.g., revocation)  
 fullCertificateChainPem,  
 "Certificate issued successfully."  
 );  
 return ResponseEntity.status(HttpStatus.CREATED).body(response);  
  
 } catch (CompletionException ce) {  
 // Unwrap the original exception from CompletableFuture  
 Throwable actualCause = ce.getCause();  
  
 if (actualCause instanceof IllegalArgumentException) {  
 // Client-side validation errors  
 log.warn("Bad request from customer {}: {}", request.getCustomerId(), actualCause.getMessage());  
 return ResponseEntity.badRequest().body(new CertificateIssueResponse("error", actualCause.getMessage()));  
 } else if (actualCause instanceof RuntimeException) {  
 // Errors propagated from the service layer, often due to Resilience4j fallback  
 log.error("Service layer error during certificate request for customer {}: {}", request.getCustomerId(), actualCause.getMessage(), actualCause);  
 // For a 5xx error, provide a generic message to the client.  
 return ResponseEntity.status(HttpStatus.SERVICE\_UNAVAILABLE) // Appropriate for circuit breaker open/retry exhaustion  
 .body(new CertificateIssueResponse("error", actualCause.getMessage())); // Fallback messages are customer-friendly  
 } else {  
 // Catch any other unexpected exceptions from the async execution  
 log.error("Unexpected error during certificate request for customer {}: {}", request.getCustomerId(), actualCause.getMessage(), actualCause);  
 return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  
 .body(new CertificateIssueResponse("error", "An unexpected internal error occurred."));  
 }  
 } catch (IllegalArgumentException e) {  
 // This catches validation errors that might happen before async execution or if @Valid is configured differently.  
 log.warn("Bad request (pre-processing) from customer {}: {}", request.getCustomerId(), e.getMessage());  
 return ResponseEntity.badRequest().body(new CertificateIssueResponse("error", e.getMessage()));  
 } catch (Exception e) {  
 // Catch any other unexpected exceptions that might bypass the above.  
 log.error("Unhandled exception during certificate request for customer {}: {}", request.getCustomerId(), e.getMessage(), e);  
 return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  
 .body(new CertificateIssueResponse("error", "An unhandled internal error occurred."));  
 }  
 }  
  
 // TODO: Implement /revoke, /status and other lifecycle management endpoints  
 // These would follow a similar pattern: external DTO -> service call -> internal GCP CAS call ->  
 // internal GCP CAS response -> external DTO response.  
}

## Setup and Testing Instructions

### 1. GCP Setup

1. **Enable GCP CAS API:**  
   Bash  
   gcloud services enable privateca.googleapis.com
2. **Create a CA Pool for B2B:**  
   Bash  
   gcloud privateca pools create your-b2b-ca-pool-id \  
    --project your-gcp-project-id \  
    --location us-central1 # Or your chosen region (e.g., asia-south1)
3. **Create a Root or Subordinate CA within the Pool:** For a production B2B system, you'd typically have a robust CA hierarchy. For this example, a self-signed root within the pool is simplest.  
   Bash  
   gcloud privateca certificates create your-b2b-sub-ca \  
    --pool your-b2b-ca-pool-id \  
    --location us-central1 \  
    --subject "CN=My B2B Client CA, O=YourCompany" \  
    --use-explicit-key-request \  
    --key-algorithm "RSA\_PKCS1\_2048" \  
    --is-ca \  
    --max-issuer-path-length 0 \  
    --lifetime 3650d # 10 years  
   Follow the prompts to generate the key and activate it.
4. **Define a Certificate Template (Highly Recommended):** This enforces policies for certificates issued through your API. Create template\_config.yaml:  
   YAML  
   identityConstraints:  
    # Allow subject and SANs from CSR to be passed through, controlled by validation in your API  
    allowSubjectPassthrough: true  
    allowSubjectAltNamesPassthrough: true  
   keyUsage:  
    baseKeyUsage:  
    digitalSignature: true  
    nonRepudiation: true # For proving non-repudiation of actions with the cert  
    keyEncipherment: true # For key agreement/exchange (e.g., TLS handshake)  
    extendedKeyUsage:  
    clientAuth: true # ESSENTIAL for mTLS client certificates  
   # Optional: policy for allowed extensions or specific values  
   # policy:  
   # ...  
   Then create the template:  
   Bash  
   gcloud privateca templates create your-b2b-client-template \  
    --project your-gcp-project-id \  
    --location us-central1 \  
    --description "Template for B2B client certificates" \  
    --passthrough-extensions "2.5.29.17" # Allow Subject Alternative Names (SANs) from CSR  
    --web-server-tls-allowed=false \  
    --client-tls-allowed=true \  
    --data-encipherment-allowed=true \  
    --template-file template\_config.yaml  
   Update application.properties with gcp.cas.certificate-template-name=your-b2b-client-template.
5. **Grant IAM Permissions:** The Service Account that your Spring Boot CaaS API will run as *must* have the roles/privateca.certificateRequester role on your your-b2b-ca-pool-id.  
   Bash  
   gcloud privateca pools add-iam-policy-binding your-b2b-ca-pool-id \  
    --location us-central1 \  
    --member serviceAccount:your-service-account@your-gcp-project-id.iam.gserviceaccount.com \  
    --role roles/privateca.certificateRequester  
   For local development, ensure your gcloud auth application-default login uses an account with this permission. For deployment (e.g., GKE, Cloud Run), configure Workload Identity.

### 2. Run the Spring Boot API

1. **Update application.properties**: Fill in your-gcp-project-id, your-b2b-ca-pool-id, and define api.security.allowed-api-keys.
2. **Build and Run:**  
   Bash  
   mvn clean install  
   java -jar target/mtls-gcp-cas-caas-0.0.1-SNAPSHOT.jar  
   Or simply:  
   Bash  
   mvn spring-boot:run

### 3. Test the API (Using curl)

1. Generate a Sample CSR (Simulating Customer Action):  
   A customer would generate their private key and a Certificate Signing Request (CSR) on their system.  
   Bash  
   # Generate a private key  
   openssl genrsa -out client.key 2048  
     
   # Generate a CSR with Subject and Subject Alternative Names (SANs)  
   openssl req -new -key client.key -out client.csr -subj "/CN=customer-service-alpha.example.com/O=CustomerX/OU=Production" -addext "subjectAltName = DNS:api-gateway.customerx.com,IP:10.0.0.1"  
     
   # Display the CSR in PEM format (copy this for the API request)  
   openssl req -in client.csr -text -noout -inform PEM  
     
   Copy the entire PEM content of client.csr (including -----BEGIN CERTIFICATE REQUEST----- and -----END CERTIFICATE REQUEST-----).
2. Call Your API:  
   Replace <YOUR\_CSR\_PEM\_CONTENT> with the actual CSR content copied above.  
   Ensure X-API-Key matches one from application.properties.  
   Bash  
   # Store CSR content in a variable (replace newlines)  
   CSR\_CONTENT=$(cat client.csr | tr -d '\n')  
     
   curl -X POST \  
    http://localhost:8080/api/v1/certificates/issue \  
    -H "Content-Type: application/json" \  
    -H "X-API-Key: customer-abc-apikey" \  
    -d "{  
    \"customerId\": \"customer-alpha-123\",  
    \"csrPem\": \"${CSR\_CONTENT}\",  
    \"requestedValidityDays\": 365,  
    \"dnsNames\": [\"api-gateway.customerx.com\"],  
    \"ipAddresses\": [\"10.0.0.1\"],  
    \"metadata\": {\"system\": \"inventory-service\", \"env\": \"prod\"}  
    }" | jq .  
     
   Expected Successful Response:  
   You should receive a JSON response similar to this:  
   JSON  
   {  
    "status": "success",  
    "certificateId": "projects/your-gcp-project-id/locations/us-central1/caPools/your-b2b-ca-pool-id/certificates/a-generated-cert-id",  
    "certificatePem": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----\n-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----\n...",  
    "message": "Certificate issued successfully."  
   }  
     
   Save the certificatePem content to a file (e.g., issued\_client\_cert\_chain.pem).
3. Verify the Issued Certificate (Customer Side):  
   The customer would take the certificatePem (which includes their issued certificate and the CA chain) and combine it with their client.key.  
   Bash  
   # Inspect the issued certificate (first part of certificatePem)  
   openssl x509 -in issued\_client\_cert\_chain.pem -text -noout  
     
   # Verify the chain (optional, but good for customer to check)  
   # This assumes you have the root CA of your B2B pool downloaded.  
   # openssl verify -CAfile /path/to/your-b2b-root-ca.pem issued\_client\_cert\_chain.pem

### 4. Test Fault Tolerance

To test fault tolerance, you'd need to simulate failures from GCP CAS. This is harder to do manually, but you can conceptually understand how it works:

* **Retry:** If a DeadlineExceededException (timeout from GCP side) occurs, Resilience4j will automatically retry the issueClientCertificate method up to 5 times with increasing delays.
* **Circuit Breaker:** If a significant percentage of calls fail (e.g., 50% in 10 calls), the circuit will "open." Subsequent calls will immediately trigger the issueClientCertificateFallback method without even attempting to call GCP CAS, preventing further load on the potentially unhealthy service. After 60 seconds, it enters "half-open" state, allowing a few test calls to see if the service has recovered.
* **Time Limiter:** If any single call attempt (including within retries) takes longer than 30 seconds, a TimeoutException is triggered, leading to the fallback.

Monitoring Resilience4j State:

While your application is running, you can check its health and Resilience4j metrics via Actuator:

* **Health:** http://localhost:8080/actuator/health
* **Prometheus Metrics:** http://localhost:8080/actuator/prometheus (Look for metrics like resilience4j\_circuitbreaker\_state, resilience4j\_retry\_calls\_succeeded\_with\_retry, etc.)

This complete solution provides a robust and fault-tolerant API for B2B mTLS certificate issuance using GCP CAS and Spring Boot.