Okay, let's consolidate everything into a comprehensive solution, integrating the design patterns and explicitly detailing fault tolerance scenarios.

## Comprehensive SSL Certificate Issuance API Solution

This document outlines the architecture, design, and implementation details for an API that issues SSL certificates based on CSRs sent by B2B consumers, leveraging GCP Certificate Authority Service (CAS) with a dedicated B2B Pool. The solution emphasizes fault tolerance, modularity through design patterns, and leverages Spring Boot and Google Cloud APIs.

### 1. Introduction

The objective is to create a robust, secure, and resilient API allowing B2B consumers to submit Certificate Signing Requests (CSRs) and receive SSL certificates. The API will perform validations, interact with GCP CAS for issuance, securely store the certificates, and provide mechanisms for handling failures gracefully.

### 2. High-Level Component Diagram (Annotated with Design Patterns)

This diagram illustrates the main logical components and their interconnections, highlighting where specific design patterns are applied.

Code snippet

graph TD  
 A[API Consumer] --> B{API Gateway};  
  
 subgraph Spring Boot API Service  
 direction LR  
 C[CertificateController]  
 D[CsrValidationService (Chain of Responsibility)]  
 E[Validation Strategies (Strategy Pattern)]  
 F[CertificateService (Facade Pattern, Circuit Breaker, Retry)]  
 G[CertificateStorageRepository (Repository Pattern)]  
 H[GCP Client Config (Singleton)]  
 end  
  
 B --> C;  
 C --> D;  
 D --> E;  
 C --> F;  
 F --> G;  
 F --> H;  
 G --> H;  
  
 F --> I[GCP Certificate Authority Service - B2B Pool];  
 G --> J[GCP Secret Manager];  
 F --> K[GCP Pub/Sub (Notifications)];  
 C --> L[GCP Cloud Logging (Observability)];  
 F --> L;  
 G --> L;  
 C --> M[GCP Cloud Monitoring (Observability)];  
 F --> M;  
 G --> M;

**Component Descriptions and Pattern Usage:**

* **API Consumer:** External B2B application requesting certificates.
* **API Gateway:** (GCP API Gateway/Load Balancer) Acts as the single entry point.
  + **API Gateway Pattern:** Centralizes authentication, authorization, rate limiting, and traffic routing, offloading these from the core service.
* **Spring Boot API Service:** Core application logic.
  + **CertificateController:** Handles HTTP requests, orchestrates business logic.
  + **CsrValidationService:**
    - **Chain of Responsibility Pattern:** Orchestrates a sequence of CSR validation steps. Each validation strategy is a handler in the chain.
  + **Validation Strategies (e.g., CsrFormatValidator, DomainWhitelistValidator, KeyPropertyValidator):**
    - **Strategy Pattern:** Each concrete validator implements CsrValidationStrategy, allowing easy addition/removal of validation rules without modifying the CsrValidationService.
  + **CertificateService:**
    - **Facade Pattern:** Provides a simplified interface (issueCertificate) to the controller, hiding the complexities of GCP CAS interaction and certificate storage.
    - **Circuit Breaker Pattern:** Protects against cascading failures by monitoring calls to GCP CAS and preventing repeated attempts to a failing service.
    - **Retry Pattern:** Automatically re-attempts transiently failed calls to GCP CAS.
  + **CertificateStorageRepository:**
    - **Repository Pattern:** Abstracts the details of storing certificates in GCP Secret Manager, providing a clean interface for CertificateService.
  + **GCP Client Config:**
    - **Singleton Pattern:** Ensures only a single instance of CertificateAuthorityServiceClient and SecretManagerServiceClient is created and reused.
* **GCP Certificate Authority Service (CAS) - B2B Pool:** The managed CA service that issues certificates.
* **GCP Secret Manager:** Securely stores issued certificates and their metadata.
* **GCP Pub/Sub:** (Optional, not fully implemented in provided code) For asynchronous notifications.
* **GCP Cloud Logging & Cloud Monitoring:**
  + **Observability Patterns:** Centralized logging and metrics collection for debugging, auditing, and performance monitoring.

### 3. High-Level GCP Architecture Sample (Annotated with Fault Tolerance)

This diagram illustrates how the components map to GCP services and highlights fault tolerance mechanisms.

Code snippet

graph TD  
 User[API Consumer] --> Internet[Internet];  
 Internet --> APIGateway(GCP API Gateway / Load Balancer);  
  
 APIGateway -- Health Checks --> CloudRun(Cloud Run / GKE / Compute Engine - Spring Boot API Service);  
  
 CloudRun -- Retries, Circuit Breaker --> PrivateCAS[GCP Certificate Authority Service - B2B Pool];  
 PrivateCAS --> RootCA[Organizational Root CA];  
  
 CloudRun --> SecretManager(GCP Secret Manager - Secure Storage, Versioning);  
 CloudRun --> PubSub(GCP Pub/Sub - Asynchronous Notifications);  
 CloudRun --> CloudLogging(GCP Cloud Logging - Centralized Logs);  
 CloudRun --> CloudMonitoring(GCP Cloud Monitoring - Metrics & Alerts);  
 CloudRun --> IAM(GCP IAM - Fine-grained Permissions);  
 CloudRun --> VPCServiceControls(GCP VPC Service Controls - Security Perimeter);  
  
 subgraph "GCP Project (Fault Tolerance Mechanisms)"  
 direction LR  
 CloudRun -- Auto-scaling, Self-healing --> CloudRun  
 PrivateCAS -- Managed Service, High Availability --> PrivateCAS  
 SecretManager -- Managed Service, Replication --> SecretManager  
 PubSub -- At-least-once Delivery, Push/Pull --> PubSub  
 CloudMonitoring -- Alerts, Dashboards --> CloudMonitoring  
 APIGateway -- Load Balancing, DDoS Protection --> APIGateway  
 end

**GCP Service Mapping with Fault Tolerance:**

* **API Gateway / Load Balancer:**
  + **Fault Tolerance:** Provides load balancing across multiple instances of your API service, absorbs DDoS attacks, and offers health checks to route traffic away from unhealthy instances.
* **Spring Boot API Service (Cloud Run / GKE / Compute Engine):**
  + **Fault Tolerance:**
    - **Auto-scaling:** Scales instances up/down based on load to handle traffic spikes.
    - **Self-healing (Cloud Run/GKE):** Automatically replaces unhealthy or crashed instances.
    - **Resilience4j (Circuit Breaker/Retry):** Within the application, isolates failures to GCP CAS, preventing cascading issues and retrying transient errors.
* **GCP Certificate Authority Service (CAS):**
  + **Fault Tolerance:** A fully managed, highly available service, eliminating the need to manage CA infrastructure, backups, or high availability manually. CAS pools are regional, providing redundancy within a region.
* **GCP Secret Manager:**
  + **Fault Tolerance:** A managed service that automatically handles replication and high availability for your secrets across regions. Supports secret versioning for easy rollback.
* **GCP Pub/Sub:**
  + **Fault Tolerance:** Asynchronous messaging decouples components. Guarantees at-least-once delivery, ensuring notifications are not lost even if the subscriber is temporarily down.
* **GCP Cloud Logging & Cloud Monitoring:**
  + **Fault Tolerance:** Provides critical observability. Monitoring allows for alerting on high error rates or latency, enabling proactive issue resolution. Centralized logging aids in post-mortem analysis.
* **GCP IAM:** Controls who can access what, preventing unauthorized actions.
* **GCP VPC Service Controls:** Creates a security perimeter around sensitive services (CAS, Secret Manager) to prevent data exfiltration, adding a layer of network-level fault tolerance against misconfigurations or malicious insiders.

### 4. Detailed Flowchart (Positive, Exceptional, and Fault-Tolerant Flows)

Code snippet

graph TD  
 A[API Consumer Sends CSR to API Endpoint] --> B{API Gateway Authentication & Authorization};  
  
 B -- Failed Auth/AuthZ (401/403) --> E1[Error: Unauthorized/Forbidden];  
  
 B -- Success --> C[Spring Boot API Service Receives Request];  
  
 C --> D{Validate Request DTO};  
 D -- Invalid DTO (400) --> E2[Error: Invalid Request Data];  
 D -- Valid DTO --> F[Parse CSR PEM to Object];  
 F -- Parsing Error (400) --> E3[Error: Malformed CSR PEM];  
 F -- Success --> G{CsrValidationService (Chain of Responsibility)};  
  
 G -- Validation Failed (400) --> E4[Error: CSR Policy Violation / Invalid];  
  
 G -- All Validations Pass --> H{Check for Duplicate/Revocation Requests};  
 H -- Duplicate/Conflict (409) --> E5[Error: Duplicate/Conflict];  
 H -- Unique Request --> I[Call CertificateService.issueCertificate()];  
  
 I --> J{Circuit Breaker State Check};  
 J -- Circuit Open --> K[Fallback: Return Service Unavailable (503)];  
 J -- Circuit Closed --> L{Call GCP CAS API (createCertificate)};  
  
 L -- Transient Error --> M{Retry Attempt?};  
 M -- Yes (Max Retries Not Reached) --> L;  
 M -- No (Max Retries Reached) --> N[Fallback: Return Service Unavailable (503)];  
  
 L -- Persistent Error (e.g., Auth, Quota, Invalid Request from CAS) --> O[Error: GCP CAS Service Error (500)];  
 L -- Success --> P[Receive Issued Certificate from GCP CAS];  
  
 P --> Q{CertificateStorageRepository.saveCertificate()};  
 Q -- Storage Failure (500) --> R[Error: Certificate Storage Failure];  
 Q -- Success --> S[Return Issued Certificate to API Consumer (200)];  
  
 S --> T[Send Notification (e.g., Pub/Sub - async)];  
  
 T --> U[Log All Actions (Cloud Logging)];  
 U --> V[Monitor Metrics (Cloud Monitoring)];  
  
 E1 --> W[End Flow];  
 E2 --> W;  
 E3 --> W;  
 E4 --> W;  
 E5 --> W;  
 K --> W;  
 N --> W;  
 O --> W;  
 R --> W;  
 S --> W;

### 5. Fault Tolerance Scenarios and How They Are Addressed

1. **GCP CAS Temporarily Unavailable/Slow:**
   * **Scenario:** Network glitches, temporary overload, or minor service disruption at the CAS endpoint.
   * **Solution:**
     + **Retry Pattern:** The CertificateService.issueCertificate() method is annotated with @Retry. It will automatically re-attempt the call to CAS for configured transient exceptions.
     + **Circuit Breaker Pattern:** The @CircuitBreaker annotation on issueCertificate() monitors the success/failure rate. If the failure rate exceeds a threshold, the circuit "opens," preventing further calls to CAS for a defined period, and immediately triggers a fallback.
     + **Fallback Method:** The issueCertificateFallback method provides a graceful degradation, returning a "service unavailable" message instead of endless timeouts or errors.
2. **Spring Boot API Service Overload/Crash:**
   * **Scenario:** High traffic, memory leaks, or unhandled exceptions cause API instances to become unresponsive or crash.
   * **Solution:**
     + **GCP Load Balancer/API Gateway Health Checks:** Automatically detect unhealthy instances and route traffic away.
     + **GCP Cloud Run/GKE Auto-scaling & Self-healing:** Cloud Run/GKE automatically scales instances based on traffic and replaces unhealthy containers, ensuring capacity and availability.
     + **Resource Limits:** Setting CPU and memory limits on Cloud Run/GKE prevents a single instance from consuming all resources.
3. **Invalid/Malicious CSR Submission:**
   * **Scenario:** API consumer sends a malformed CSR, requests a forbidden domain, or uses a weak key.
   * **Solution:**
     + **CSR Validation Service (Chain of Responsibility/Strategy):** Multiple validation strategies (CsrFormatValidator, DomainWhitelistValidator, KeyPropertyValidator) check the CSR before sending it to CAS.
     + **Early Exit:** Invalid CSRs are rejected early in the process (e.g., 400 Bad Request), preventing unnecessary calls to GCP CAS and protecting the CA pool from invalid requests.
4. **Certificate Storage Failure (GCP Secret Manager):**
   * **Scenario:** Temporary issues with Secret Manager, or IAM permissions problems preventing storage.
   * **Solution:**
     + **Error Handling:** The CertificateStorageRepository catches exceptions from Secret Manager. While a retry could be added here, immediate failure and logging are often preferred for critical persistence to avoid inconsistencies.
     + **Audit Logging:** Critical failures are logged to Cloud Logging for investigation.
5. **External Service Dependency Failures (e.g., Authentication Service):**
   * **Scenario:** The external system providing B2B authentication (e.g., OAuth provider) is down.
   * **Solution:**
     + **API Gateway:** Handles authentication/authorization. If the external provider fails, the API Gateway can return a 401/403 directly, preventing requests from even reaching the Spring Boot service.
6. **Observability Gaps:**
   * **Scenario:** Unable to debug issues in production or track system health.
   * **Solution:**
     + **Centralized Logging (Cloud Logging):** All requests, errors, and significant events are logged in a structured format, enabling easy searching and analysis.
     + **Metrics & Monitoring (Cloud Monitoring):** Key metrics (request rate, error rate, latency, circuit breaker state) are collected and visualized, with alerts configured for anomalies.

### 6. Spring Boot Java Implementation

This section provides the complete code.

#### build.gradle

Gradle

plugins {  
 id 'java'  
 id 'org.springframework.boot' version '3.3.1'  
 id 'io.spring.dependency-management' version '1.1.5'  
}  
  
group = 'com.example'  
version = '0.0.1-SNAPSHOT'  
  
java {  
 toolchain {  
 languageVersion = JavaLanguageVersion.of(17) // Or your preferred Java version  
 }  
}  
  
configurations {  
 compileOnly {  
 extendsFrom annotationProcessor  
 }  
}  
  
repositories {  
 mavenCentral()  
}  
  
dependencies {  
 implementation 'org.springframework.boot:spring-boot-starter-web'  
 implementation 'org.springframework.boot:spring-boot-starter-actuator' # For health checks and metrics  
 implementation 'com.google.cloud:google-cloud-security-private-ca:2.35.0' # Adjust version as needed  
 implementation 'com.google.cloud:google-cloud-secretmanager:2.34.0' # Adjust version as needed  
 implementation 'io.github.resilience4j:resilience4j-spring-boot3:2.2.0' # Circuit Breaker & Retry  
 implementation 'org.springframework.boot:spring-boot-starter-aop' # Required by Resilience4j  
 implementation 'org.bouncycastle:bcprov-jdk18on:1.77' # For CSR parsing  
 implementation 'org.bouncycastle:bcpkix-jdk18on:1.77' # For CSR parsing  
  
 compileOnly 'org.projectlombok:lombok:1.18.32'  
 annotationProcessor 'org.projectlombok:lombok:1.18.32'  
  
 testImplementation 'org.springframework.boot:spring-boot-starter-test'  
 testImplementation 'io.github.resilience4j:resilience4j-circuitbreaker-test:2.2.0'  
 testImplementation 'io.github.resilience4j:resilience4j-retry-test:2.2.0'  
 testRuntimeOnly 'org.junit.platform:junit-platform-launcher'  
}  
  
tasks.named('test') {  
 useJUnitPlatform()  
}

#### src/main/java/com/example/certapi/CertApiApplication.java

Java

package com.example.certapi;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.annotation.EnableAspectJAutoProxy; // Needed for Resilience4j AOP  
  
@SpringBootApplication  
@EnableAspectJAutoProxy(proxyTargetClass = true) // Enable AOP for Resilience4j annotations  
public class CertApiApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.run(CertApiApplication.class, args);  
 }  
  
}

#### src/main/java/com/example/certapi/config/GcpClientConfig.java

Java

package com.example.certapi.config;  
  
import com.google.cloud.security.privateca.v1.CertificateAuthorityServiceClient;  
import com.google.cloud.secretmanager.v1.SecretManagerServiceClient;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
import java.io.IOException;  
  
/\*\*  
 \* Configuration class for initializing Google Cloud client services.  
 \* These clients are typically singletons managed by Spring.  
 \*/  
@Configuration  
public class GcpClientConfig {  
  
 /\*\*  
 \* Provides a singleton instance of CertificateAuthorityServiceClient.  
 \* This client will use Application Default Credentials (ADC),  
 \* which automatically picks up credentials based on the environment  
 \* (e.g., service account key path, GKE Workload Identity, Cloud Run service identity).  
 \*/  
 @Bean  
 public CertificateAuthorityServiceClient certificateAuthorityServiceClient() throws IOException {  
 return CertificateAuthorityServiceClient.create();  
 }  
  
 /\*\*  
 \* Provides a singleton instance of SecretManagerServiceClient.  
 \* Uses Application Default Credentials (ADC).  
 \*/  
 @Bean  
 public SecretManagerServiceClient secretManagerServiceClient() throws IOException {  
 return SecretManagerServiceClient.create();  
 }  
}

#### src/main/java/com/example/certapi/controller/CertificateController.java

Java

package com.example.certapi.controller;  
  
import com.example.certapi.controller.dto.CertificateRequestDto;  
import com.example.certapi.controller.dto.CertificateResponse;  
import com.example.certapi.exception.CertificateIssuanceException;  
import com.example.certapi.service.CertificateService;  
import com.example.certapi.validation.CsrValidationException;  
import com.example.certapi.validation.CsrValidationService;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
/\*\*  
 \* REST Controller for handling certificate issuance requests.  
 \* Acts as the entry point for API consumers.  
 \*/  
@RestController  
@RequestMapping("/api/v1/certificates")  
public class CertificateController {  
  
 // Dependencies injected via constructor (Dependency Injection)  
 private final CertificateService certificateService;  
 private final CsrValidationService csrValidationService;  
  
 public CertificateController(CertificateService certificateService,  
 CsrValidationService csrValidationService) {  
 this.certificateService = certificateService;  
 this.csrValidationService = csrValidationService;  
 }  
  
 /\*\*  
 \* Endpoint to issue a new SSL certificate based on a provided CSR.  
 \* @param requestDto Contains the CSR PEM string and consumer ID.  
 \* @return ResponseEntity with the issued certificate or an error message.  
 \*/  
 @PostMapping("/issue")  
 public ResponseEntity<CertificateResponse> issueCertificate(@RequestBody CertificateRequestDto requestDto) {  
 try {  
 // 1. Basic input validation for the DTO  
 if (requestDto.getCsrPem() == null || requestDto.getCsrPem().isEmpty()) {  
 return ResponseEntity.badRequest().body(new CertificateResponse(null, "CSR PEM cannot be empty."));  
 }  
 if (requestDto.getConsumerId() == null || requestDto.getConsumerId().isEmpty()) {  
 return ResponseEntity.badRequest().body(new CertificateResponse(null, "Consumer ID cannot be empty."));  
 }  
  
 // 2. Validate CSR format and content using the CsrValidationService.  
 // This service applies the Chain of Responsibility/Strategy patterns.  
 csrValidationService.validateCsrAndPolicies(requestDto.getCsrPem());  
  
 // 3. Check for duplicates/revocations. This would typically involve a database lookup.  
 if (certificateService.isDuplicateRequest(requestDto.getCsrPem(), requestDto.getConsumerId())) {  
 return ResponseEntity.status(HttpStatus.CONFLICT)  
 .body(new CertificateResponse(null, "Certificate for this request already exists or is in progress."));  
 }  
  
 // 4. Issue certificate using GCP CAS via the CertificateService.  
 // The CertificateService acts as a Facade and implements Circuit Breaker/Retry.  
 String issuedCertPem = certificateService.issueCertificate(requestDto.getCsrPem(), requestDto.getConsumerId());  
  
 // 5. Return success response with the issued certificate.  
 return ResponseEntity.ok(new CertificateResponse(issuedCertPem, "Certificate issued successfully."));  
  
 } catch (CsrValidationException e) {  
 // Catches validation errors from CsrValidationService  
 System.err.println("CSR Validation Error: " + e.getMessage());  
 return ResponseEntity.badRequest().body(new CertificateResponse(null, e.getMessage()));  
 } catch (CertificateIssuanceException e) {  
 // Catches errors from CertificateService (e.g., CAS errors, storage errors, fallback responses)  
 System.err.println("Certificate Issuance Error: " + e.getMessage());  
 return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  
 .body(new CertificateResponse(null, e.getMessage()));  
 } catch (Exception e) {  
 // Generic fallback for any unhandled exceptions  
 System.err.println("An unexpected error occurred: " + e.getMessage());  
 return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  
 .body(new CertificateResponse(null, "An unexpected internal server error occurred. Please contact support."));  
 }  
 }  
}

#### src/main/java/com/example/certapi/controller/dto/CertificateRequestDto.java

Java

package com.example.certapi.controller.dto;  
  
import lombok.Data;  
  
/\*\*  
 \* Data Transfer Object (DTO) for incoming certificate issuance requests.  
 \* Uses Lombok for boilerplate code (getters, setters, equals, hashCode, toString).  
 \*/  
@Data  
public class CertificateRequestDto {  
 private String csrPem; // The Certificate Signing Request in PEM format  
 private String consumerId; // Identifier for the B2B API consumer  
}

#### src/main/java/com/example/certapi/controller/dto/CertificateResponse.java

Java

package com.example.certapi.controller.dto;  
  
import lombok.Data;  
  
/\*\*  
 \* Data Transfer Object (DTO) for outgoing certificate issuance responses.  
 \* Contains the issued certificate PEM and a descriptive message.  
 \*/  
@Data  
public class CertificateResponse {  
 private String issuedCertPem; // The issued certificate in PEM format  
 private String message; // A message describing the outcome of the request  
  
 public CertificateResponse(String issuedCertPem, String message) {  
 this.issuedCertPem = issuedCertPem;  
 this.message = message;  
 }  
}

#### src/main/java/com/example/certapi/exception/CertificateIssuanceException.java

Java

package com.example.certapi.exception;  
  
/\*\*  
 \* Custom exception for errors occurring during the certificate issuance process.  
 \* This helps to categorize and handle specific types of errors.  
 \*/  
public class CertificateIssuanceException extends RuntimeException {  
 public CertificateIssuanceException(String message) {  
 super(message);  
 }  
 public CertificateIssuanceException(String message, Throwable cause) {  
 super(message, cause);  
 }  
}

#### src/main/java/com/example/certapi/repository/CertificateStorageRepository.java

Java

package com.example.certapi.repository;  
  
import com.google.api.gax.rpc.NotFoundException;  
import com.google.cloud.secretmanager.v1.AddSecretVersionRequest;  
import com.google.cloud.secretmanager.v1.CreateSecretRequest;  
import com.google.cloud.secretmanager.v1.Replication;  
import com.google.cloud.secretmanager.v1.Secret;  
import com.google.cloud.secretmanager.v1.SecretManagerServiceClient;  
import com.google.cloud.secretmanager.v1.SecretName;  
import com.google.protobuf.ByteString;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Repository;  
  
import java.util.HashMap;  
import java.util.Map;  
  
/\*\*  
 \* Repository class for securely storing issued certificates in GCP Secret Manager.  
 \* Implements the Repository Pattern to abstract storage details.  
 \*/  
@Repository  
public class CertificateStorageRepository {  
  
 @Value("${gcp.project.id}")  
 private String projectId;  
  
 private final SecretManagerServiceClient secretManagerClient;  
  
 public CertificateStorageRepository(SecretManagerServiceClient secretManagerClient) {  
 this.secretManagerClient = secretManagerClient;  
 }  
  
 /\*\*  
 \* Saves an issued certificate and its metadata to GCP Secret Manager.  
 \* Each certificate is stored as a new version of a secret.  
 \*  
 \* @param certificateResourceName The full GCP CAS resource name of the issued certificate.  
 \* @param issuedCertPem The issued certificate content in PEM format.  
 \* @param consumerId The identifier of the B2B consumer who requested the certificate.  
 \*/  
 public void saveCertificate(String certificateResourceName, String issuedCertPem, String consumerId) {  
 try {  
 // Extract a unique identifier from the GCP CAS resource name for the Secret ID  
 String[] parts = certificateResourceName.split("/");  
 String certIdFromCas = parts[parts.length - 1]; // e.g., "cert-consumer-1234abcd-12345678"  
 String secretId = "issued-cert-" + certIdFromCas; // Prefix for clarity  
 String parent = String.format("projects/%s", projectId);  
  
 // Labels for better organization and searchability in Secret Manager  
 Map<String, String> labels = new HashMap<>();  
 labels.put("consumer\_id", consumerId);  
 labels.put("gcp\_cas\_name", certificateResourceName);  
 labels.put("type", "issued-ssl-certificate"); // Custom label  
  
 // Check if the secret already exists. If not, create it.  
 try {  
 secretManagerClient.getSecret(SecretName.of(projectId, secretId));  
 System.out.println("Secret '" + secretId + "' already exists. Adding new version.");  
 } catch (NotFoundException e) {  
 // Secret does not exist, create a new one  
 Secret secret = Secret.newBuilder()  
 .setReplication(Replication.newBuilder().setAutomatic(Replication.Automatic.newBuilder().build()))  
 .putAllLabels(labels)  
 .build();  
  
 CreateSecretRequest createSecretRequest = CreateSecretRequest.newBuilder()  
 .setParent(parent)  
 .setSecretId(secretId)  
 .setSecret(secret)  
 .build();  
 secretManagerClient.createSecret(createSecretRequest);  
 System.out.println("Created new secret: " + secretId);  
 }  
  
 // Add a new version to the secret with the certificate content  
 AddSecretVersionRequest addVersionRequest = AddSecretVersionRequest.newBuilder()  
 .setParent(SecretName.of(projectId, secretId).toString())  
 .setPayload(com.google.cloud.secretmanager.v1.SecretPayload.newBuilder()  
 .setData(ByteString.copyFromUtf8(issuedCertPem))  
 .build())  
 .build();  
 secretManagerClient.addSecretVersion(addVersionRequest);  
 System.out.println("Added new version to secret: " + secretId);  
  
 } catch (Exception e) {  
 System.err.println("Error storing certificate in Secret Manager: " + e.getMessage());  
 // It's critical to know if storage failed. Re-throw as a runtime exception.  
 throw new RuntimeException("Failed to store issued certificate securely in Secret Manager.", e);  
 }  
 }  
 // Additional methods (e.g., getCertificate, listCertificates, deleteCertificate) could be added here.  
}

#### src/main/java/com/example/certapi/service/CertificateService.java

Java

package com.example.certapi.service;  
  
import com.example.certapi.exception.CertificateIssuanceException;  
import com.example.certapi.repository.CertificateStorageRepository;  
import com.google.api.gax.rpc.ApiException;  
import com.google.cloud.security.privateca.v1.Certificate;  
import com.google.cloud.security.privateca.v1.CertificateAuthorityServiceClient;  
import com.google.cloud.security.privateca.v1.CreateCertificateRequest;  
import com.google.protobuf.Duration;  
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;  
import io.github.resilience4j.retry.annotation.Retry;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Service;  
  
import java.util.UUID;  
  
/\*\*  
 \* Service class responsible for orchestrating certificate issuance with GCP CAS  
 \* and storing the issued certificates.  
 \* This class acts as a Facade to the underlying GCP client APIs and storage.  
 \* It also implements Fault Tolerance patterns (Circuit Breaker, Retry).  
 \*/  
@Service  
public class CertificateService {  
  
 // Externalized configuration for GCP CAS details  
 @Value("${gcp.project.id}")  
 private String projectId;  
 @Value("${gcp.ca.pool.location}")  
 private String caPoolLocation;  
 @Value("${gcp.ca.pool.name}")  
 private String caPoolName;  
 @Value("${gcp.ca.certificate.template.name}")  
 private String certificateTemplateName;  
 @Value("${app.certificate.lifetime-days:365}")  
 private int certificateLifetimeDays;  
  
 // Dependencies injected via constructor (Dependency Injection)  
 private final CertificateAuthorityServiceClient caClient;  
 private final CertificateStorageRepository certificateStorageRepository;  
  
 public CertificateService(CertificateAuthorityServiceClient caClient,  
 CertificateStorageRepository certificateStorageRepository) {  
 this.caClient = caClient;  
 this.certificateStorageRepository = certificateStorageRepository;  
 }  
  
 /\*\*  
 \* Issues an SSL certificate using GCP Certificate Authority Service.  
 \* Applies Circuit Breaker and Retry patterns for fault tolerance.  
 \*  
 \* @param csrPem The Certificate Signing Request in PEM format.  
 \* @param consumerId The ID of the B2B consumer.  
 \* @return The issued certificate in PEM format.  
 \* @throws CertificateIssuanceException if any error occurs during issuance or storage.  
 \*/  
 @CircuitBreaker(name = "gcpCasService", fallbackMethod = "issueCertificateFallback")  
 @Retry(name = "gcpCasService")  
 public String issueCertificate(String csrPem, String consumerId) throws CertificateIssuanceException {  
 System.out.println("Attempting to issue certificate for consumer: " + consumerId);  
 try {  
 // Construct the full resource paths for CA Pool and Certificate Template  
 String caPoolPath = String.format("projects/%s/locations/%s/caPools/%s", projectId, caPoolLocation, caPoolName);  
 String certificateTemplatePath = String.format("projects/%s/locations/%s/certificateTemplates/%s", projectId, caPoolLocation, certificateTemplateName);  
  
 // Build the request to create a certificate in GCP CAS  
 CreateCertificateRequest request = CreateCertificateRequest.newBuilder()  
 .setParent(caPoolPath)  
 .setCertificateId(generateUniqueCertificateId(consumerId)) // Unique ID for the certificate resource  
 .setCertificate(Certificate.newBuilder()  
 .setPemCsr(csrPem) // Set the CSR from the consumer  
 .setCertificateTemplate(certificateTemplatePath) // Apply the configured template  
 .setLifetime(Duration.newBuilder().setSeconds((long) certificateLifetimeDays \* 24 \* 60 \* 60).build()) // Set certificate lifetime  
 .build())  
 .build();  
  
 // Call the GCP CAS API to create the certificate  
 Certificate certificate = caClient.createCertificate(request);  
  
 String issuedCertPem = certificate.getPemCertificate();  
 String certificateName = certificate.getName(); // GCP CAS assigned full resource name of the certificate  
  
 // Store the issued certificate securely using the Repository Pattern  
 certificateStorageRepository.saveCertificate(certificateName, issuedCertPem, consumerId);  
  
 System.out.println("Certificate issued and stored successfully. Resource: " + certificateName);  
 return issuedCertPem;  
  
 } catch (ApiException e) {  
 // Catches exceptions specific to Google Cloud APIs  
 System.err.println("GCP CAS API Error: " + e.getStatusCode().getCode() + " - " + e.getMessage());  
 throw new CertificateIssuanceException("Error interacting with GCP Certificate Authority Service: " + e.getMessage(), e);  
 } catch (RuntimeException e) {  
 // Catches RuntimeExceptions, including those thrown by CertificateStorageRepository  
 System.err.println("A runtime error occurred during certificate issuance: " + e.getMessage());  
 throw new CertificateIssuanceException("A critical error occurred during certificate issuance or storage.", e);  
 } catch (Exception e) {  
 // Catch any other unexpected exceptions  
 System.err.println("An unexpected error occurred during certificate issuance: " + e.getMessage());  
 throw new CertificateIssuanceException("An unexpected error occurred during certificate issuance.", e);  
 }  
 }  
  
 /\*\*  
 \* Fallback method for the Circuit Breaker.  
 \* This method is invoked when the circuit breaker is open or calls exceed retry limits.  
 \*  
 \* @param csrPem The original CSR PEM.  
 \* @param consumerId The original consumer ID.  
 \* @param t The throwable that triggered the fallback.  
 \* @return A message indicating service unavailability.  
 \* @throws CertificateIssuanceException Always throws, to propagate the error gracefully.  
 \*/  
 public String issueCertificateFallback(String csrPem, String consumerId, Throwable t) throws CertificateIssuanceException {  
 System.err.println("Fallback triggered for issueCertificate operation. Error: " + t.getMessage());  
 // In a real-world scenario, you might log more details, send an alert,  
 // or attempt to return a cached certificate for renewals.  
 throw new CertificateIssuanceException("Certificate issuance service currently unavailable due to external system issues. Please try again later.", t);  
 }  
  
 /\*\*  
 \* Placeholder for checking if a similar certificate request is a duplicate.  
 \* In a production system, this would query a database or cache.  
 \*  
 \* @param csrPem The CSR PEM.  
 \* @param consumerId The consumer ID.  
 \* @return True if a duplicate request is found, false otherwise.  
 \*/  
 public boolean isDuplicateRequest(String csrPem, String consumerId) {  
 // TODO: Implement actual logic to check for duplicates.  
 // This could involve hashing the CSR subject/public key and checking against stored certificates.  
 // Or querying a database that tracks active certificate requests.  
 return false;  
 }  
  
 /\*\*  
 \* Generates a unique, GCP CAS-compliant certificate ID.  
 \* GCP CAS certificate IDs must be 1-63 characters, lowercase letters, numbers, hyphens.  
 \*  
 \* @param consumerId The consumer's identifier.  
 \* @return A unique certificate ID string.  
 \*/  
 private String generateUniqueCertificateId(String consumerId) {  
 // Sanitize consumerId to be safe for GCP resource names  
 String sanitizedConsumerId = consumerId.replaceAll("[^a-z0-9-]", "").toLowerCase();  
 // Limit length to avoid issues and combine with a short UUID for uniqueness  
 if (sanitizedConsumerId.length() > 30) {  
 sanitizedConsumerId = sanitizedConsumerId.substring(0, 30);  
 }  
 return "cert-" + sanitizedConsumerId + "-" + UUID.randomUUID().toString().substring(0, 8);  
 }  
}

#### src/main/java/com/example/certapi/validation/CsrValidationException.java

Java

package com.example.certapi.validation;  
  
/\*\*  
 \* Custom exception for errors encountered during CSR validation.  
 \* Helps distinguish validation errors from other system errors.  
 \*/  
public class CsrValidationException extends RuntimeException {  
 public CsrValidationException(String message) {  
 super(message);  
 }  
 public CsrValidationException(String message, Throwable cause) {  
 super(message, cause);  
 }  
}

#### src/main/java/com/example/certapi/validation/CsrValidationService.java

Java

package com.example.certapi.validation;  
  
import org.bouncycastle.jce.provider.BouncyCastleProvider;  
import org.bouncycastle.pkcs.PKCS10CertificationRequest;  
import org.bouncycastle.util.io.pem.PemObject;  
import org.bouncycastle.util.io.pem.PemReader;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import javax.annotation.PostConstruct;  
import java.io.StringReader;  
import java.security.Security;  
import java.util.List;  
  
/\*\*  
 \* Service for validating Certificate Signing Requests (CSRs).  
 \* Implements the Chain of Responsibility pattern by iterating through a list of  
 \* CsrValidationStrategy implementations.  
 \*/  
@Service  
public class CsrValidationService {  
  
 // Spring will auto-inject all beans that implement CsrValidationStrategy (Chain of Responsibility)  
 private final List<CsrValidationStrategy> validationStrategies;  
  
 /\*\*  
 \* Initializes Bouncy Castle as a security provider.  
 \* Bouncy Castle is necessary for parsing and inspecting CSRs in Java.  
 \*/  
 @PostConstruct  
 public void init() {  
 if (Security.getProvider(BouncyCastleProvider.PROVIDER\_NAME) == null) {  
 Security.addProvider(new BouncyCastleProvider());  
 System.out.println("Bouncy Castle Security Provider added.");  
 }  
 }  
  
 public CsrValidationService(@Autowired List<CsrValidationStrategy> validationStrategies) {  
 this.validationStrategies = validationStrategies;  
 System.out.println("Loaded " + validationStrategies.size() + " CSR validation strategies.");  
 }  
  
 /\*\*  
 \* Parses a CSR from PEM format into a Bouncy Castle PKCS10CertificationRequest object.  
 \* @param csrPem The CSR in PEM string format.  
 \* @return The parsed PKCS10CertificationRequest object.  
 \* @throws CsrValidationException if the PEM is malformed or not a valid CSR.  
 \*/  
 public PKCS10CertificationRequest parseCsrPem(String csrPem) throws CsrValidationException {  
 try (PemReader pemReader = new PemReader(new StringReader(csrPem))) {  
 PemObject pemObject = pemReader.readPemObject();  
 if (pemObject == null || !pemObject.getType().equals("CERTIFICATE REQUEST")) {  
 throw new CsrValidationException("Invalid PEM object type. Expected 'CERTIFICATE REQUEST'.");  
 }  
 return new PKCS10CertificationRequest(pemObject.getContent());  
 } catch (Exception e) {  
 throw new CsrValidationException("Failed to parse CSR PEM: " + e.getMessage(), e);  
 }  
 }  
  
 /\*\*  
 \* Validates the CSR against all configured validation strategies.  
 \* This method implements the Chain of Responsibility: each strategy in the list  
 \* is executed sequentially. If any strategy throws a CsrValidationException,  
 \* the chain is broken, and the exception is propagated.  
 \*  
 \* @param csrPem The CSR in PEM string format to be validated.  
 \* @throws CsrValidationException if any validation rule is violated.  
 \*/  
 public void validateCsrAndPolicies(String csrPem) throws CsrValidationException {  
 // First, parse the CSR to an object that all strategies can work with.  
 PKCS10CertificationRequest csr = parseCsrPem(csrPem);  
  
 // Iterate through all injected validation strategies (Chain of Responsibility)  
 for (CsrValidationStrategy strategy : validationStrategies) {  
 strategy.validate(csr); // Each strategy performs its specific validation.  
 // If validate() throws an exception, it breaks the loop and the exception is caught higher up.  
 }  
 System.out.println("All CSR validations and policies passed successfully.");  
 }  
}

#### src/main/java/com/example/certapi/validation/CsrValidationStrategy.java

Java

package com.example.certapi.validation;  
  
import org.bouncycastle.pkcs.PKCS10CertificationRequest;  
  
/\*\*  
 \* Interface for CSR validation strategies.  
 \* Implements the Strategy Pattern. Each concrete implementation will define  
 \* a specific validation rule for the CSR.  
 \*/  
public interface CsrValidationStrategy {  
 /\*\*  
 \* Validates a given CSR based on a specific rule.  
 \* @param csr The parsed Bouncy Castle PKCS10CertificationRequest object.  
 \* @throws CsrValidationException if the CSR fails this specific validation rule.  
 \*/  
 void validate(PKCS10CertificationRequest csr) throws CsrValidationException;  
}

#### src/main/java/com/example/certapi/validation/impl/CsrFormatValidator.java

Java

package com.example.certapi.validation.impl;  
  
import com.example.certapi.validation.CsrValidationException;  
import com.example.certapi.validation.CsrValidationStrategy;  
import org.bouncycastle.pkcs.PKCS10CertificationRequest;  
import org.springframework.core.annotation.Order;  
import org.springframework.stereotype.Component;  
  
/\*\*  
 \* Validates the basic format and structure of the CSR.  
 \* This should ideally be the first validator in the chain.  
 \*/  
@Component  
@Order(1) // Ensures this runs first in the Chain of Responsibility  
public class CsrFormatValidator implements CsrValidationStrategy {  
  
 @Override  
 public void validate(PKCS10CertificationRequest csr) throws CsrValidationException {  
 if (csr == null) {  
 throw new CsrValidationException("CSR object is null after parsing. This indicates a severe parsing issue.");  
 }  
 // Additional low-level format checks could be performed here if Bouncy Castle's parser  
 // doesn't cover all desired malformation checks (e.g., checking for specific  
 // ASN.1 tag values or unexpected optional fields if strictly required).  
 // For most cases, successful parsing by Bouncy Castle implies basic format correctness.  
  
 if (!csr.isSignatureValid()) {  
 throw new CsrValidationException("CSR signature is invalid. CSR may be tampered or corrupted.");  
 }  
  
 System.out.println("Validation: CSR Format and Signature are valid.");  
 }  
}

#### src/main/java/com/example/certapi/validation/impl/DomainWhitelistValidator.java

Java

package com.example.certapi.validation.impl;  
  
import com.example.certapi.validation.CsrValidationException;  
import com.example.certapi.validation.CsrValidationStrategy;  
import org.bouncycastle.asn1.ASN1ObjectIdentifier;  
import org.bouncycastle.asn1.x509.Extension;  
import org.bouncycastle.asn1.x509.GeneralName;  
import org.bouncycastle.asn1.x509.GeneralNames;  
import org.bouncycastle.pkcs.PKCS10CertificationRequest;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.core.annotation.Order;  
import org.springframework.stereotype.Component;  
  
import java.util.Arrays;  
import java.util.HashSet;  
import java.util.Set;  
import java.util.stream.Collectors;  
  
/\*\*  
 \* Validates that all requested domains (CN and SANs) are part of an allowed whitelist.  
 \* This is a crucial policy enforcement validator.  
 \*/  
@Component  
@Order(2) // Runs after format validation  
public class DomainWhitelistValidator implements CsrValidationStrategy {  
  
 // Allowed domains configured in application.properties as a comma-separated list  
 @Value("${app.allowed-domains}")  
 private Set<String> allowedDomains;  
  
 @Override  
 public void validate(PKCS10CertificationRequest csr) throws CsrValidationException {  
 Set<String> requestedDomains = extractDomainsFromCsr(csr);  
  
 if (requestedDomains.isEmpty()) {  
 throw new CsrValidationException("CSR must contain at least one domain (Common Name or Subject Alternative Name).");  
 }  
  
 // Check if all requested domains are present in the allowed list  
 if (!allowedDomains.containsAll(requestedDomains)) {  
 // Identify which specific domains are not allowed  
 Set<String> disallowed = new HashSet<>(requestedDomains);  
 disallowed.removeAll(allowedDomains);  
 throw new CsrValidationException("One or more requested domains are not allowed by policy: " + disallowed);  
 }  
 System.out.println("Validation: All requested domains (" + requestedDomains + ") are whitelisted.");  
 }  
  
 /\*\*  
 \* Helper method to extract Common Name (CN) and Subject Alternative Names (SANs) from the CSR.  
 \* @param csr The parsed PKCS10CertificationRequest.  
 \* @return A set of extracted domain names (lowercase).  
 \*/  
 private Set<String> extractDomainsFromCsr(PKCS10CertificationRequest csr) {  
 Set<String> domains = new HashSet<>();  
  
 // Extract Common Name (CN) from the Subject  
 // OID for CN is 2.5.4.3  
 if (csr.getSubject().getRDNs(new ASN1ObjectIdentifier("2.5.4.3")) != null) {  
 Arrays.stream(csr.getSubject().getRDNs(new ASN1ObjectIdentifier("2.5.4.3")))  
 .forEach(rdn -> Arrays.stream(rdn.getTypesAndValues())  
 .filter(typeAndValue -> typeAndValue.getType().equals(new ASN1ObjectIdentifier("2.5.4.3")))  
 .forEach(typeAndValue -> domains.add(typeAndValue.getValue().toString())));  
 }  
  
 // Extract Subject Alternative Names (SANs) from extensions  
 // SAN extension OID is 2.5.29.17  
 try {  
 Extension sanExtension = csr.getAttributes()[0].getAttributes()[0].getAttrValues().getObjects()  
 .stream()  
 .filter(o -> o instanceof Extension)  
 .map(o -> (Extension) o)  
 .filter(ext -> ext.getExtnId().equals(Extension.subjectAlternativeName))  
 .findFirst()  
 .orElse(null);  
  
 if (sanExtension != null) {  
 GeneralNames names = GeneralNames.getInstance(sanExtension.getParsedValue());  
 for (GeneralName name : names.getNames()) {  
 if (name.getTagNo() == GeneralName.dNSName) { // DNS Name tag  
 domains.add(name.getName().toString());  
 }  
 // Could also handle IP addresses (GeneralName.iPAddress) if needed  
 }  
 }  
 } catch (Exception e) {  
 // Log and throw if SAN parsing fails, as it indicates a malformed CSR  
 throw new CsrValidationException("Failed to parse Subject Alternative Names from CSR: " + e.getMessage(), e);  
 }  
 return domains.stream().map(String::toLowerCase).collect(Collectors.toSet());  
 }  
}

#### src/main/java/com/example/certapi/validation/impl/KeyPropertyValidator.java

Java

package com.example.certapi.validation.impl;  
  
import com.example.certapi.validation.CsrValidationException;  
import com.example.certapi.validation.CsrValidationStrategy;  
import org.bouncycastle.asn1.pkcs.RSAPublicKey;  
import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;  
import org.bouncycastle.pkcs.PKCS10CertificationRequest;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.core.annotation.Order;  
import org.springframework.stereotype.Component;  
  
import java.util.Arrays;  
import java.util.Set;  
import java.util.stream.Collectors;  
  
/\*\*  
 \* Validates the public key properties within the CSR, such as algorithm and key size.  
 \* Enforces security policies for cryptographic key strength.  
 \*/  
@Component  
@Order(3) // Runs after format and domain validation  
public class KeyPropertyValidator implements CsrValidationStrategy {  
  
 @Value("${app.min-key-size.rsa:2048}")  
 private int minRsaKeySize; // Minimum allowed RSA key size in bits  
  
 @Value("${app.allowed-key-algorithms:RSA,EC}")  
 private Set<String> allowedKeyAlgorithms; // Comma-separated list of allowed algorithms (e.g., RSA, EC)  
  
 @Override  
 public void validate(PKCS10CertificationRequest csr) throws CsrValidationException {  
 SubjectPublicKeyInfo pkInfo = csr.getSubjectPublicKeyInfo();  
 String keyAlgorithmOid = pkInfo.getAlgorithm().getAlgorithm().getId();  
 String keyAlgorithmName = getKeyAlgorithmName(keyAlgorithmOid);  
  
 // Validate Allowed Key Algorithms  
 if (!allowedKeyAlgorithms.contains(keyAlgorithmName.toUpperCase())) {  
 throw new CsrValidationException("Key algorithm '" + keyAlgorithmName + "' (OID: " + keyAlgorithmOid + ") is not allowed by policy. Allowed: " + allowedKeyAlgorithms);  
 }  
  
 // Validate Key Size based on algorithm  
 if ("RSA".equalsIgnoreCase(keyAlgorithmName)) {  
 try {  
 RSAPublicKey rsaPublicKey = RSAPublicKey.getInstance(pkInfo.parsePublicKey());  
 int keySize = rsaPublicKey.getModulus().bitLength();  
 if (keySize < minRsaKeySize) {  
 throw new CsrValidationException("RSA key size (" + keySize + " bits) is below minimum allowed size (" + minRsaKeySize + " bits).");  
 }  
 System.out.println("Validation: RSA Key size (" + keySize + " bits) is valid.");  
 } catch (Exception e) {  
 throw new CsrValidationException("Failed to extract RSA key properties from CSR: " + e.getMessage(), e);  
 }  
 } else if ("EC".equalsIgnoreCase(keyAlgorithmName)) {  
 // For EC keys, validating curve parameters is more complex and depends on policy.  
 // You might parse OIDs for specific named curves.  
 System.out.println("Validation: EC Key algorithm is allowed. Curve validation can be added if needed.");  
 }  
 // Add validation for other key algorithms as needed  
  
 System.out.println("Validation: Key algorithm '" + keyAlgorithmName + "' is allowed.");  
 }  
  
 /\*\*  
 \* Simple helper to map OID to common algorithm name.  
 \* Extend as needed for more OIDs.  
 \*/  
 private String getKeyAlgorithmName(String oid) {  
 if ("1.2.840.113549.1.1.1".equals(oid)) {  
 return "RSA";  
 } else if ("1.2.840.10045.2.1".equals(oid)) {  
 return "EC";  
 }  
 return oid; // Return OID if unknown  
 }  
}

### 7. Application Properties

Properties

# src/main/resources/application.properties  
  
# Spring Boot Application Settings  
spring.application.name=ssl-cert-api  
server.port=8080  
  
# GCP Project Configuration (EXTERNALIZED)  
gcp.project.id=your-gcp-project-id-here # REPLACE WITH YOUR GCP PROJECT ID  
gcp.ca.pool.location=your-gcp-region # e.g., us-central1, asia-south1 # REPLACE WITH YOUR GCP REGION  
gcp.ca.pool.name=your-b2b-ca-pool # REPLACE WITH YOUR GCP CAS B2B POOL NAME  
gcp.ca.certificate.template.name=your-default-cert-template # REPLACE WITH YOUR GCP CAS CERTIFICATE TEMPLATE NAME  
  
# Application Specific Policies (EXTERNALIZED)  
app.certificate.lifetime-days=365 # Default certificate validity in days  
  
# CSR Validation Policies  
# Comma-separated list of allowed domains (e.g., example.com, \*.example.com)  
app.allowed-domains=example.com,sub.example.com,mycorp.org  
# Minimum allowed RSA key size in bits  
app.min-key-size.rsa=2048  
# Comma-separated list of allowed key algorithms (e.g., RSA, EC)  
app.allowed-key-algorithms=RSA,EC  
  
# Resilience4j Circuit Breaker Configuration  
# Instance name 'gcpCasService' refers to the one used in @CircuitBreaker/@Retry annotations  
resilience4j.circuitbreaker.instances.gcpCasService.registerHealthIndicator=true # Exposes /actuator/health endpoint for CB state  
resilience4j.circuitbreaker.instances.gcpCasService.failureRateThreshold=50 # Percentage of failed calls to trip the circuit  
resilience4j.circuitbreaker.instances.gcpCasService.slowCallRateThreshold=100 # Percentage of slow calls to trip the circuit (100% means all calls are slow)  
resilience4j.circuitbreaker.instances.gcpCasService.slowCallDurationThreshold=5s # Calls slower than this are considered slow  
resilience4j.circuitbreaker.instances.gcpCasService.waitDurationInOpenState=60s # Time circuit remains open before trying half-open  
resilience4j.circuitbreaker.instances.gcpCasService.permittedNumberOfCallsInHalfOpenState=10 # Number of calls allowed in half-open state  
resilience4j.circuitbreaker.instances.gcpCasService.slidingWindowType=COUNT\_BASED # Use count-based sliding window  
resilience4j.circuitbreaker.instances.gcpCasService.slidingWindowSize=100 # Size of the sliding window for metrics (last 100 calls)  
resilience4j.circuitbreaker.instances.gcpCasService.minimumNumberOfCalls=10 # Minimum calls required before calculating failure rate  
  
# Resilience4j Retry Configuration  
resilience4j.retry.instances.gcpCasService.maxAttempts=3 # Max number of retry attempts  
resilience4j.retry.instances.gcpCasService.waitDuration=2s # Initial wait duration before first retry  
resilience4j.retry.instances.gcpCasService.enableExponentialBackoff=true # Enable exponential backoff  
resilience4j.retry.instances.gcpCasService.exponentialBackoffMultiplier=2 # Multiplier for exponential backoff (2s, 4s, 8s...)  
# Configure which exceptions should trigger a retry (Google API runtime exceptions are often transient)  
resilience4j.retry.instances.gcpCasService.retryExceptions=com.google.api.gax.rpc.ApiException  
  
# Logging Configuration (Example - adapt for production)  
logging.level.root=INFO  
logging.level.com.example.certapi=DEBUG # Adjust for more verbose logging of your application  
logging.level.com.google.cloud=WARN # Suppress verbose GCP client logging

### 8. Deployment Considerations

* **GCP Cloud Run (Recommended for Serverless Containers):**
  + Deploy your Spring Boot JAR as a container.
  + Leverage Cloud Run's auto-scaling, built-in load balancing, and fully managed environment.
  + Ensure your Cloud Run service identity (service account) has the necessary IAM roles:
    - roles/privateca.certificateRequester (on your CAS CA Pool)
    - roles/secretmanager.secretAccessor (on secrets where certificates are stored)
    - roles/logging.logWriter (for Cloud Logging)
    - roles/monitoring.metricWriter (for Cloud Monitoring)
* **Google Kubernetes Engine (GKE):**
  + For more complex deployments, fine-grained control, or existing Kubernetes ecosystems.
  + Deploy as a Kubernetes Deployment and Service.
  + Use Workload Identity for secure IAM integration.
* **GCP Compute Engine (VMs):**
  + If you require specific OS configurations or long-running processes that don't fit containerized environments. Less managed overhead than Cloud Run/GKE.
* **IAM (Identity and Access Management):** Critical for security. Ensure your service accounts have the *least privilege* required to perform their tasks.
* **VPC Service Controls:** Implement a security perimeter around your CAS CA Pool and Secret Manager to prevent unauthorized data exfiltration, adding a vital layer of defense.
* **Network Security:** Configure firewall rules appropriately to limit inbound traffic to your API Gateway and outbound traffic from your Spring Boot service.

This comprehensive solution provides a robust, fault-tolerant, and well-structured API for issuing SSL certificates on GCP, built with Spring Boot and standard design patterns.

**Sources**

1. <https://github.com/20HyeonsuLee/INFRA_BACK>

2. <https://github.com/dlehddn/slack-error-sender>

3. <https://github.com/gyuwon/SpringCamp2024>

4. <https://github.com/luismarcilio/textSearch>

5. <https://github.com/StefanUAIC/addresscorrection>

6. <https://github.com/PavelKuchkovskii/MiscalculationWEB>

7. <https://github.com/AmielDeLosReyes/D-Baesics-Ecommerce>

8. <https://github.com/swedenconnect/ca-engine> subject to license (Apache - 2.0)

9. <https://github.com/GoogleCloudPlatform/spring-cloud-gcp> subject to license (Apache - 2.0)

10. <https://github.com/googleapis/google-cloud-java> subject to license (Apache - 2.0)

11. <https://cloud.google.com/secret-manager/docs/samples/secretmanager-delete-secret-with-etag?hl=ko>

12. <https://stackoverflow.com/questions/77386802/nested-exception-is-java-lang-noclassdeffounderror-could-not-initialize-class-c>

13. <https://github.com/2JeanRojas2/BackEnd2>

14. <https://cloud.google.com/certificate-authority-service/docs/requesting-certificates?hl=id>

15. <https://cloud.google.com/certificate-authority-service/docs/samples/privateca-create-certificate-csr?hl=ko>

16. <https://stackoverflow.com/questions/76449684/localstack-sqs-cant-be-reached>

17. <https://github.com/wso2-attic/product-emm> subject to license (Apache - 2.0)

18. <https://stackoverflow.com/questions/44824897/getting-subject-alternate-names-with-pkcs10certificationrequest?rq=1>

19. <https://github.com/Burence1/covid>

20. <https://github.com/mhus-info/mhus-lib> subject to license (Apache - 2.0)

21. <https://github.com/babylonhealth/certificate-transparency-android> subject to license (Apache - 2.0)

22. <https://github.com/hortonworks/cloudbreak> subject to license (Apache - 2.0)

23. <https://github.com/genome-nexus/genome-nexus> subject to license (MIT)

24. <https://github.com/verygoodsecurity/starlarky> subject to license (Apache - 2.0)

25. <https://github.com/courier-mgmt/auth-nt>