

1. EntityMetadata

java

Copy

```
import java.util.List;

public class EntityMetadata {
    private String entityId;
    private String entityName;
    private String entityType;
    private String source;
    private String entityGroup; // e.g., "recommendable"
    private List<String> associatedRuleIds;
    private List<String> supportedActions;

    public EntityMetadata(String entityId, String entityName, String
entityType, String source,
                        String entityGroup, List<String>
associatedRuleIds, List<String> supportedActions) {
        this.entityId = entityId;
        this.entityName = entityName;
        this.entityType = entityType;
        this.source = source;
        this.entityGroup = entityGroup;
        this.associatedRuleIds = associatedRuleIds;
        this.supportedActions = supportedActions;
    }

    // Getters
    public String getEntityId() { return entityId; }
    public String getEntityName() { return entityName; }
    public String getEntityType() { return entityType; }
    public String getSource() { return source; }
    public String getEntityGroup() { return entityGroup; }
    public List<String> getAssociatedRuleIds() { return
associatedRuleIds; }
    public List<String> getSupportedActions() { return
supportedActions; }
```

```
    // Setters
    public void setEntityId(String entityId) { this.entityId =
entityId; }
    public void setEntityName(String entityName) { this.entityName =
entityName; }
    public void setEntityType(String entityType) { this.entityType =
entityType; }
    public void setSource(String source) { this.source = source; }
    public void setEntityGroup(String entityGroup) { this.entityGroup
= entityGroup; }
    public void setAssociatedRuleIds(List<String> associatedRuleIds) {
this.associatedRuleIds = associatedRuleIds; }
    public void setSupportedActions(List<String> supportedActions) {
this.supportedActions = supportedActions; }
}
```

2. Abstract Class Recommendable

```
java
Copy
import java.util.Map;

public abstract class Recommendable {
    protected EntityMetadata metadata;

    public Recommendable(EntityMetadata metadata) {
        this.metadata = metadata;
    }

    /**
     * Returns a map of key-value pairs representing the entity's
data.
     */
    public abstract Map<String, Object> getEntityData();

    public EntityMetadata getMetadata() {
        return metadata;
    }
}
```

```
}
```

3. Concrete Class ArticleRecommendable

```
java
Copy
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

public class ArticleRecommendable extends Recommendable {
    private String author;
    private String content;
    private Date publicationDate;

    public ArticleRecommendable(EntityMetadata metadata, String
author, String content, Date publicationDate) {
        super(metadata);
        this.author = author;
        this.content = content;
        this.publicationDate = publicationDate;
    }

    @Override
    public Map<String, Object> getEntityData() {
        Map<String, Object> data = new HashMap<>();
        data.put("author", author);
        data.put("content", content);
        data.put("publicationDate", publicationDate);
        return data;
    }

    // Optionally add getters/setters for article-specific fields.
    public String getAuthor() { return author; }
    public String getContent() { return content; }
    public Date getPublicationDate() { return publicationDate; }
}
```

4. Rule Class

java

Copy

```
public class Rule {
    private String ruleId;
    private String description;
    private String conditionExpression; // This could represent a DSL
    or JSON logic expression.
    private int priority;

    public Rule(String ruleId, String description, String
conditionExpression, int priority) {
        this.ruleId = ruleId;
        this.description = description;
        this.conditionExpression = conditionExpression;
        this.priority = priority;
    }

    public String getRuleId() { return ruleId; }
    public String getDescription() { return description; }
    public String getConditionExpression() { return
conditionExpression; }
    public int getPriority() { return priority; }

    public void setRuleId(String ruleId) { this.ruleId = ruleId; }
    public void setDescription(String description) { this.description
= description; }
    public void setConditionExpression(String conditionExpression) {
this.conditionExpression = conditionExpression; }
    public void setPriority(int priority) { this.priority = priority;
}
}
```

5. RuleEngine Interface

This interface represents the rule evaluation logic. You would provide an implementation that checks an entity against a list of rules given the customer attributes in the context.

java

Copy

```
import java.util.List;
import java.util.Map;

public interface RuleEngine {
    /**
     * Evaluates the provided rules for the given entity and context.
     *
     * @param entity The recommendable entity.
     * @param context A map of customer attributes and other context
     variables.
     * @param rules A list of rules to evaluate.
     * @return true if the entity passes all rule evaluations; false
     otherwise.
     */
    boolean evaluate(Recommendable entity, Map<String, Object>
context, List<Rule> rules);
}
```

6. RecommendationEngine Class

The recommendation engine uses the rule engine to determine which recommendable entities should be recommended based on the customer context and associated rules.

java

Copy

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class RecommendationEngine {
    private RuleEngine ruleEngine;

    public RecommendationEngine(RuleEngine ruleEngine) {
```

```

        this.ruleEngine = ruleEngine;
    }

    /**
     * Filters and returns the list of entities that pass the rule
    evaluations.
     *
     * @param entities A list of recommendable entities.
     * @param context A map of customer attributes and other
    contextual data.
     * @param allRules A list of all possible rules.
     * @return A list of recommended entities.
     */
    public List<Recommendable> recommend(List<Recommendable> entities,
    Map<String, Object> context, List<Rule> allRules) {
        List<Recommendable> recommendations = new ArrayList<>();

        for (Recommendable entity : entities) {
            // Filter rules associated with the current entity.
            List<Rule> entityRules = allRules.stream()
                .filter(rule ->
    entity.getMetadata().getAssociatedRuleIds().contains(rule.getRuleId())
    )
                .collect(Collectors.toList());

            // Evaluate the rules using the rule engine.
            boolean rulePassed = ruleEngine.evaluate(entity, context,
    entityRules);
            if (rulePassed) {
                recommendations.add(entity);
            }
        }

        return recommendations;
    }
}

```

Final Notes

- **EntityMetadata vs. EntityGroup:**

Here, the `entityGroup` field in the `EntityMetadata` class helps you group entities (e.g., "recommendable") independently from their specific `entityType` (e.g., "Article").

- **Enforcing Rules:**

The rule evaluation is decoupled via the `RuleEngine` interface. In a production system, you might implement this using AOP, a service layer, or the proxy pattern.

This complete model allows different teams to implement concrete recommendable objects (like `ArticleRecommendable`) while leveraging a centralized recommendation engine that enforces customer-specific rules.

o3-mini

You said:

implement this using AOP

ChatGPT said:

Reasoned for 5 seconds

Below is an example of how you might implement the above model with Aspect-Oriented Programming (AOP) in a Spring Boot application. In this example, we'll intercept calls to the method that returns the entity's data (i.e. `getEntityData()`) and enforce rule evaluation before the data is returned.

For simplicity, assume you already have these classes implemented:

- **EntityMetadata**
- **Recommendable** (abstract class) and its concrete subclass (e.g., `ArticleRecommendable`)
- **Rule**
- **RuleEngine** (an interface with an implementation that evaluates the rules)

We'll add an AOP aspect that:

- Retrieves the current customer context (e.g., via a helper like `CustomerContextHolder`),

- Fetches the associated rules (using, for instance, a [RuleRepository](#)), and
 - Invokes the rule engine to validate access before the entity's data is returned.
-

1. The AOP Aspect

java

Copy

```
package com.example.recommendation.aop;

import com.example.recommendation.Rule;
import com.example.recommendation.Recommendable;
import com.example.recommendation.RuleEngine;
import com.example.recommendation.repository.RuleRepository;
import com.example.recommendation.security.CustomerContext;
import com.example.recommendation.security.CustomerContextHolder;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.security.access.AccessDeniedException;

import java.util.List;
import java.util.Map;

@Aspect
@Component
public class RuleEnforcementAspect {

    @Autowired
    private RuleEngine ruleEngine;

    @Autowired
    private RuleRepository ruleRepository;

    /**
```



```

        * This advice intercepts calls to getEntityData() on any
Recommendable.
        */
        @Before("execution(*
com.example.recommendation.Recommendable.getEntityData(..)) &&
target(entity)")
        public void enforceRules(JoinPoint joinPoint, Recommendable
entity) throws Throwable {
            // Retrieve customer attributes from a custom context holder.
            CustomerContext customerContext =
CustomerContextHolder.getContext();
            Map<String, Object> context = customerContext.getAttributes();

            // Retrieve rules associated with the entity.
            List<Rule> rules =
ruleRepository.getRules(entity.getMetadata().getAssociatedRuleIds());

            // Evaluate the rules.
            boolean rulePassed = ruleEngine.evaluate(entity, context,
rules);

            if (!rulePassed) {
                throw new AccessDeniedException("Access to this entity is
denied based on customer attributes.");
            }
        }
    }
}

```

2. Example Implementation of a Customer Context Holder

This helper class simulates a context that would typically be built from the authenticated user's session or security context.

```

java
Copy
package com.example.recommendation.security;

import java.util.HashMap;

```

```

import java.util.Map;

public class CustomerContext {
    private Map<String, Object> attributes = new HashMap<>();

    public CustomerContext(Map<String, Object> attributes) {
        this.attributes = attributes;
    }

    public Map<String, Object> getAttributes() {
        return attributes;
    }
}

java
Copy
package com.example.recommendation.security;

public class CustomerContextHolder {
    // For simplicity, this returns a dummy context. In a real
    // application, you might extract details from the security context.
    public static CustomerContext getContext() {
        // Simulate customer attributes.
        return new CustomerContext(
            java.util.Map.of(
                "age", 30,
                "location", "US",
                "preferences", java.util.List.of("technology",
"science")
            )
        );
    }
}

```

3. Sample RuleRepository

A simple repository interface and its dummy implementation to fetch rules by their IDs.

```

java
Copy
package com.example.recommendation.repository;

import com.example.recommendation.Rule;
import org.springframework.stereotype.Repository;

import java.util.ArrayList;
import java.util.List;

@Repository
public class RuleRepository {

    // Dummy implementation for demo purposes.
    public List<Rule> getRules(List<String> ruleIds) {
        List<Rule> rules = new ArrayList<>();
        // For each ruleId, create a dummy rule (in practice, you'd
query a database or other data source)
        for (String id : ruleIds) {
            rules.add(new Rule(id, "Dummy rule for " + id,
"dummyCondition", 1));
        }
        return rules;
    }
}

```

4. Sample RuleEngine Implementation

A basic implementation that always approves or denies access based on a dummy condition. In a real-world scenario, your implementation might parse a DSL or JSON-based condition expression.

```

java
Copy
package com.example.recommendation;

import com.example.recommendation.Recommendable;
import org.springframework.stereotype.Component;

```

```

import java.util.List;
import java.util.Map;

@Component
public class SimpleRuleEngine implements RuleEngine {

    @Override
    public boolean evaluate(Recommendable entity, Map<String, Object>
context, List<Rule> rules) {
        // Dummy implementation: if any rule has a conditionExpression
        "deny", then deny access.
        for (Rule rule : rules) {
            if
("deny".equalsIgnoreCase(rule.getConditionExpression())) {
                return false;
            }
        }
        return true; // Otherwise, allow access.
    }
}

```

5. How It All Works

- Entity Access:**
 When a client calls `getEntityData()` on an instance of a subclass of `Recommendable` (e.g., `ArticleRecommendable`), the aspect intercepts the call.
- Customer Context:**
 The aspect retrieves the customer's attributes from the `CustomerContextHolder`.
- Rule Retrieval & Evaluation:**
 It then fetches the associated rules for the entity from the `RuleRepository` and passes the entity, context, and rules to the `RuleEngine` for evaluation.
- Access Decision:**
 If the rule evaluation fails, an `AccessDeniedException` is thrown. Otherwise, the

method proceeds to return the entity's data.

Final Note

This implementation uses Spring AOP to cleanly separate rule enforcement from business logic. It ensures that every access to the entity's data goes through the rule evaluation process, making it a modern and maintainable solution for enforcing customer-based access rules.

You can integrate and test this setup within your Spring Boot application to verify that the aspect correctly intercepts method calls and enforces the rule logic as intended.

o3-mini

