Tutorial

Supply Chain Collaboration End-to-End

Version 1.2.2-SNAPSHOT, 2025-09-13T10:41:36Z

Table of Contents

In this tutorial, we will model a simplified supply chain collaboration system, create repositories and REST resources, and see how Quantum provides a robust set of APIs that a UI can utilize—all with multi-tenancy baked in.

Chapter 1. Domain Overview

Functional Area: Collaboration

Functional Domains:

- Partner: Companies participating in the supply chain (manufacturers, carriers, retailers)
- Shipment: Units of transport moving goods between locations
- Task: Collaborative tasks associated with shipments (e.g., confirm pickup, update ETA)

Chapter 2. Models

```java
import dev.morphia.annotations.Entity;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
import com.e2eq.framework.model.persistent.base.BaseModel;

@Entity
@Data
@NoArgsConstructor
@SuperBuilder
@EqualsAndHashCode(callSuper = true)
public class Partner extends BaseModel {
    private String refCode;
    private String name;
    @Override public String bmFunctionalArea() { return "Collaboration"; }
    @Override public String bmFunctionalDomain() { return "Partner"; }
}

@Entity
@Data
@NoArgsConstructor
@SuperBuilder
@EqualsAndHashCode(callSuper = true)
public class Shipment extends BaseModel {
    private String trackingNumber;
    private String origin;
    private String destination;
    @Override public String bmFunctionalArea() { return "Collaboration"; }
    @Override public String bmFunctionalDomain() { return "Shipment"; }
}
```

DataDomain on BaseModel ensures tenant context (tenantId/orgRefName/etc.) is carried and indexed as needed (see idx_refIdUniqueInDomain).

Chapter 3. Repositories

```java
import com.e2eq.framework.model.persistent.morphia.MorphiaRepo;

public interface PartnerRepo extends MorphiaRepo<Partner> {}
public interface ShipmentRepo extends MorphiaRepo<Shipment> {}
```

Chapter 4. REST Resources

```java
import com.e2eq.framework.rest.resources.BaseResource;
import jakarta.ws.rs.Path;

@Path("/partners")
public class PartnerResource extends BaseResource<Partner, PartnerRepo> {}

@Path("/shipments")
public class ShipmentResource extends BaseResource<Shipment, ShipmentRepo> {}
```

These resources automatically expose consistent find/get/list/save/update/delete endpoints with DataDomain-aware filtering.

Chapter 5. Multi-Tenant Sharing Scenarios

- Shared Partner Directory: Allow cross-tenant read of Partner records while keeping Shipments strictly tenant-scoped. Implement in RuleContext by permitting VIEW on Partner across tenants when a sharing flag is set in DataDomain (e.g., orgRefName == "PUBLIC").
- Strict Shipment Isolation: Enforce that Shipment queries always constrain by tenantId and (optionally) orgRefName.

Chapter 6. RuleContext Sketch

```
public class CollaborationRuleContext /* implements your RuleContext SPI */ {
    public boolean canView(Object model, DomainContext ctx) {
        // Allow partner directory reads across tenants if shared
        if (model instanceof Partner p) {
            return p.getDataDomain() != null && "PUBLIC".equals(p.getDataDomain()
.getOrgRefName());
        }
        // Default: require same tenant
        return ctx.getTenantId().equals(extractTenant(model));
    }
}
```

Chapter 7. Authentication

Use quantum-jwt-provider to accept JWTs from your IdP. Map claims to tenantId/orgRefName/user/roles; BaseResource builds DomainContext for RuleContext.

Chapter 8. Separating Models and Repos for Workflows

Keep models and repositories in their own modules (as this repository does for quantum-models and quantum-morphia-repos). This allows:

- Integration into workflows like Temporal (activities operating directly on repos/models)
- Orchestration by tools like Windmill, while REST APIs remain in another service/module

Chapter 9. Result

With these building blocks, a UI can:

- Search the Partner directory (cross-tenant where allowed)
- Create/manage Shipments scoped to the tenant
- Update and track tasks with action-driven UIActionList feedback

All while maintaining consistent policy enforcement via RuleContext and DataDomain.