SECTION 11 — Governance Layer Implementation

Platform Policy and Subscription Plan

1. What Is the Governance Layer?

The Governance Layer defines:

- Platform-wide business rules
- System-level configurations
- Commercial and operational policies

It answers questions such as:

- What is the commission percentage?
- What are the refund rules?
- What delivery charges apply?
- What subscription plans exist?

Governance is not about users or products.

It is about how the platform operates.

2. Why Governance Must Be a Separate Layer

In beginner applications, business rules are often hardcoded.

Example:

const commission = orderAmount * 0.1;

This is a mistake.

If the commission changes:

- Code must be modified.
- Application must be redeployed.
- No version history is maintained.

Enterprise systems require:

- Versioned rules
- Configurable policies
- Audit history
- No hardcoding

That is why Governance is separated.

3. Governance Module Structure

```
modules/governance/
 ├── models/
 │    ├── platformPolicy.model.js
 │    └── subscriptionPlan.model.js
 ├── controllers/
 │    ├── platformPolicy.controller.js
 │    └── subscriptionPlan.controller.js
 ├── services/
 │    ├── platformPolicy.service.js
 │    └── subscriptionPlan.service.js
 ├── governance.routes.js
```

Each governance concept is isolated into model, service, and controller.

4. Platform Policy

4.1 Purpose

Platform Policy defines system-level rules such as:

- Commission percentage or fixed value
- Refund policy
- Delivery charges

Only one policy is active at a time.

Policies are versioned and immutable.

4.2 Platform Policy Model (Conceptual Structure)

```
{
  commissionPolicy: {
    type: { type: String },
    value: Number
  },
  refundPolicy: {
    allowedDays: Number,
    partialRefundAllowed: Boolean
  },
  deliveryPolicy: {
    baseCharge: Number,
    freeDeliveryAbove: Number
  },
  version: Number,
  isActive: Boolean
}
```

Important Fields

- version: Tracks policy history.
- isActive: Only one policy can be active.
- commissionPolicy: Defines how commission is calculated.

5. Creating a New Platform Policy

Endpoint

POST /api/governance/policy

Requires permission:

MANAGE_PLATFORM_POLICY

Workflow

```
Route
  ↓
protect middleware
  ↓
authorize middleware
  ↓
Controller
  ↓
Service
  ↓
Deactivate previous active policy
  ↓
Create new version
  ↓
Return response
```

Why Deactivate Instead of Update?

Enterprise principle:

Policies should not be edited.

They should be versioned.

If policy changes:

- Old policy remains in database.
- New policy becomes active.
- Historical data remains valid.

This ensures:

- Auditability
- Data integrity

- Legal compliance

6. Getting Active Policy

Endpoint

GET /api/governance/policy/active

Returns:

- The currently active policy.

This will later be used by:

- Order calculations
- Commission logic
- Refund validation

7. Getting All Policies

Endpoint

GET /api/governance/policy

Purpose:

- View history of policy changes.
- Audit previous configurations.

This is usually restricted to administrators.

8. Subscription Plan

8.1 Purpose

Subscription Plan defines:

- Membership tiers
- Pricing
- Benefits
- Duration

Examples:

- BASIC
- PREMIUM
- ELITE

Subscription plans affect:

- Delivery benefits
- Discount percentage
- Reward multipliers

8.2 Subscription Plan Model (Conceptual Structure)

Example:

```
{
  name: String,
  price: Number,
  durationInDays: Number,
  benefits: {
    freeDelivery: Boolean,
    extraDiscountPercentage: Number,
    rewardMultiplier: Number
  },
  version: Number,
  isActive: Boolean
}
```

Important Concepts

- Plans are versioned.
- Old versions are not deleted.
- Only active plans can be assigned to users.

This follows the same governance pattern as platform policy.

9. Creating a Subscription Plan

Endpoint

POST /api/governance/subscription

Requires permission:

MANAGE_SUBSCRIPTION

Workflow

```
Route
  ↓
protect
  ↓
authorize
  ↓
Controller
  ↓
Service
  ↓
Check existing versions
  ↓
```

```
Create new version
  ↓
Return response
```

10. Why Governance Uses Services Heavily

Governance contains business rules.

Controllers only:

- Accept request
- Send response

Services handle:

- Version increment logic
- Active status updates
- Validation
- Policy consistency

This separation is critical in enterprise systems.

11. Governance Request Lifecycle Example

Example: Create Policy

```
Client
  ↓
POST /api/governance/policy
  ↓
protect middleware
  ↓
authorize middleware
  ↓
Controller
  ↓
Service (deactivate old policy, create new version)
  ↓
Database
  ↓
successResponse
```

12. Why Governance Layer Is Enterprise-Ready

Governance layer demonstrates:

- Versioned configuration
- No hardcoded business rules
- Centralized business control
- Permission-based access
- Microservice-ready modularity

In large systems:

- Business rules change frequently.
- Governance layer prevents code modification for rule changes.

13. Governance vs Identity

Identity	Governance
Handles users	Handles platform rules
Authentication	Business configuration
JWT	Policy versioning
Roles	Commercial rules

They are separate but complementary layers.

14. What Students Must Understand

1. Governance should never be hardcoded.
2. Policies must be versioned.
3. Only one active policy exists.
4. Subscription plans are business configuration.
5. Governance is protected by permission matrix.
6. Controllers remain thin.
7. Services enforce rule consistency.

15. Summary

By the end of this section, students should understand:

- What governance means in backend systems
- How platform policies are structured
- Why versioning is critical
- How subscription plans work
- Why governance must be isolated
- How authorization protects governance endpoints