Technical Design Documentation

The application consists of an TypeScript-Angular Frontend, a Java-Spring Boot Backend and a Postgres DB.

The application is intended in a cluster-manner and therefore Docker is utilized for the Container Runtimes and Docker Compose as an Ochestration Tool.

Core Components:

- **Angular Frontend** — User interface for cluster management
- **Spring Boot Backend** — Distributed application with profile-based roles
- **PostgreSQL Database** — Multi-schema data persistence

Technology Stack

Frontend

Component	Technology	Version
Framework	Angular	20.3.6
HTTP Client	Angular HttpClient	-
Routing	Angular Router	-
Production Server	Nginx	Latest

Backend

Component	Technology	Version
Framework	Spring Boot	3.x
Reactive Stack	Spring WebFlux	3.x
Database	PostgreSQL	15+
Data Access	R2DBC	-
Migrations	Flyway	-
Caching	Caffeine	-
Testing	H2 (in-memory)	-

Infrastructure

Component	Technology
Containerization	Docker
Orchestration	Docker Compose

Component	Technology
Monitoring	Prometheus + Grafana
Load Testing	k6

Architecture Patterns

Backend Design

Profile-Based Role Activation:

```
Single Application Codebase
├── shared/           # All nodes
├── cluster_manager/  # @Profile("cluster_manager")
```

Reactive Programming:

- Non-blocking I/O with `Mono<T>` and `Flux<T>`
- R2DBC for reactive database access
- WebClient for inter-node HTTP communication

Caching Strategy:

- Caffeine cache (5 min TTL, 100 entries max)
- Event-driven invalidation
- Cluster Manager polls nodes every 5 seconds for cache events

Frontend (Angular)

Core Services:

- `AuthService` — Session management
- `ApiService` — HTTP communication with Cluster Manager
- `PermissionGuard` — Route-level authorization

Feature Modules:

- **Users** — Manage users and groups
- **Nodes** — Monitor and manage SEP Sesam Server Nodes
- **Backups** — View and manage backup tasks
- **Permissions** - View group permissions
- **Clients** - View registered backup clients
- **Tasks** - Track backup task execution and status
- **Dashboard** — Embedded Grafana metrics visualization

The Frontend exclusively communicates with the Cluster Manager backend.

Backend

The Backend is a single application with modular functionality. The **shared** code runs on all backend instances, while role-specific code is activated via runtime configuration (Spring profiles).

Cluster Manager

The Cluster Manager acts as the **leader role** in the cluster. It is a Backup Node with the `cluster_manager` profile activated.

Responsibilities:

- Serves as the backbone backend to the Frontend (exclusive communication)
- Handles metadata and authentication
- Manages and orchestrates Backup Node instances
- Synchronizes users, groups, and permissions across nodes
- Proxies requests to downstream nodes
- Node health monitoring via heartbeat mechanism
- Executes backup tasks like any other Backup Node

Databases:

- `bcm` — Cluster metadata (users, groups, node registry)
- `bcm_node0` — Cluster Manager's own backup data

Spring Profile: `cluster_manager` (in addition to environment profile like `dev`)

Backup Node

Backup Nodes provide endpoints to accept delegated tasks:

All backend instances are Backup Nodes, including the Cluster Manager. Backup Nodes provide endpoints to accept delegated tasks:

Responsibilities:

- Execute backup creation tasks
- Store backup data (simulated via random data in PostgreSQL tables)
- Each Backup Node uses its own database schema (`bcm_nodeX`)
- Self-register with the Cluster Manager on startup (except CM itself)

> **Note:** Backup functionality is simulated/mocked for this MVP. Actual backup data is represented by storing random data in separate PostgreSQL tables per node.

Database:

- `bcm_nodeX` — Node-specific backup data (X = node identifier)

Spring Profile: Environment profile only (e.g., `dev`). The `cluster_manager` profile is **only** added to the Cluster Manager instance.

Shared Functionality

Code and endpoints present on all backend instances:

- **Node Registration** — Nodes self-register with the Cluster Manager on startup
- **Heartbeat** — Health check mechanism for node monitoring
- **Common Models** — Shared domain models and DTOs
- **Base Configuration** — Security, CORS, web configuration

Database

PostgreSQL is used with the following schema structure:

Node Type	Database	Purpose
Cluster Manager	`bcm`	Cluster metadata, users, groups, node registry
All Nodes (including CM)	`bcm_nodeX`	Each node's backup data (node0 for CM, node1-6 for others)

All databases run in the same PostgreSQL instance within the Docker Compose setup.

Note: The Cluster Manager uses **both** `bcm` (for cluster metadata) and `bcm_node0` (for its own backup data), demonstrating that it participates as both leader and worker.

Core Domain Models

Cluster Metadata (`bcm`)

- **User**: id, username, password_hash, created_at
- **Group**: id, name, permissions[]
- **Node**: id, hostname, port, status, last_heartbeat
- **UserGroup**: user_id, group_id (many-to-many)

Backup Data (`bcm_nodeX`)

- **Backup**: id, client_id, task_id, data_blob, created_at, size
- **Task**: id, type, status, scheduled_at, completed_at
- **Client**: id, name, registered_at, node_id

REST API Structure

Frontend-Facing (Cluster Manager Only)

Note: Representative endpoints shown. See controller classes for complete API.

Authentication

Method	Endpoint	Description
POST	`/api/v1/cm/auth/login`	User login
GET	`/api/v1/cm/auth/validate`	Validate session

Node Management

Method	Endpoint	Description	Permission
GET	`/api/v1/cm/nodes`	List nodes	NODE_READ
DELETE	`/api/v1/cm/node/{id}`	Delete node	NODE_DELETE

User Management

Method	Endpoint	Description
GET	`/api/v1/cm/users`	List users
POST	`/api/v1/cm/users`	Create user

Backup Management

Method	Endpoint	Description
GET	`/api/v1/cm/backups`	Get backups

Internal (All Nodes)

Method	Endpoint	Description
POST	`/api/v1/cm/register`	Node registration
GET	`/api/v1/ping`	Heartbeat check
POST	`/api/v1/bn/backups/{id}/execute`	Execute backup

Communication Flow

Node Registration

```
Backup Node Startup
  ↓
POST /api/v1/cm/register
  ↓
Cluster Manager validates
  ↓
Stores in bcm.nodes
  ↓
Returns 200 OK
```

Heartbeat Monitoring

```
Scheduled Task (Cluster Manager)
  ↓
GET /api/v1/ping (all nodes)
```

```
  ↓
Success → Mark ACTIVE
Timeout (5s) → Mark INACTIVE
  ↓
Sync node status
```

Caching & Performance

Caffeine Cache Configuration

- **TTL:** 5 minutes
- **Max Entries:** 100
- **Cached Data:** Backup pages, client pages, task pages

Cache Invalidation

```
Backup Node modifies data
  ↓
Stores event in event_store table
  ↓
Cluster Manager polls every 5s
  ↓
Acknowledges events
  ↓
Clears relevant caches
```

Endpoint: `GET /api/v1/cm/cache/inspect` (debug)

Security Design

Authentication

- Session-based (Spring Security WebSession)

Authorization

- **Role-based**: ADMIN, OPERATOR, VIEWER (hierarchical ranking)
- **Permission-based enforcement**: Roles grant permissions (NODE_READ, NODE_UPDATE, NODE_DELETE, NODE_CONTROL, etc.)
- **Enforcement**: `@PreAuthorize` annotations on endpoints

Observability

Logging

- **Framework**: SLF4J + Logback
- **Levels**: DEBUG (dev), INFO (prod)
- **Correlation IDs**: Track requests across nodes

Metrics (Grafana)

- Node health status
- Backup task success rate
- Database connection pool usage
- API response times

Heartbeat Mechanism:

- Cluster Manager pings all nodes to check availability
- Ping timeout: 5 seconds per node
- Recovery: Automatic reactivation on next successful ping