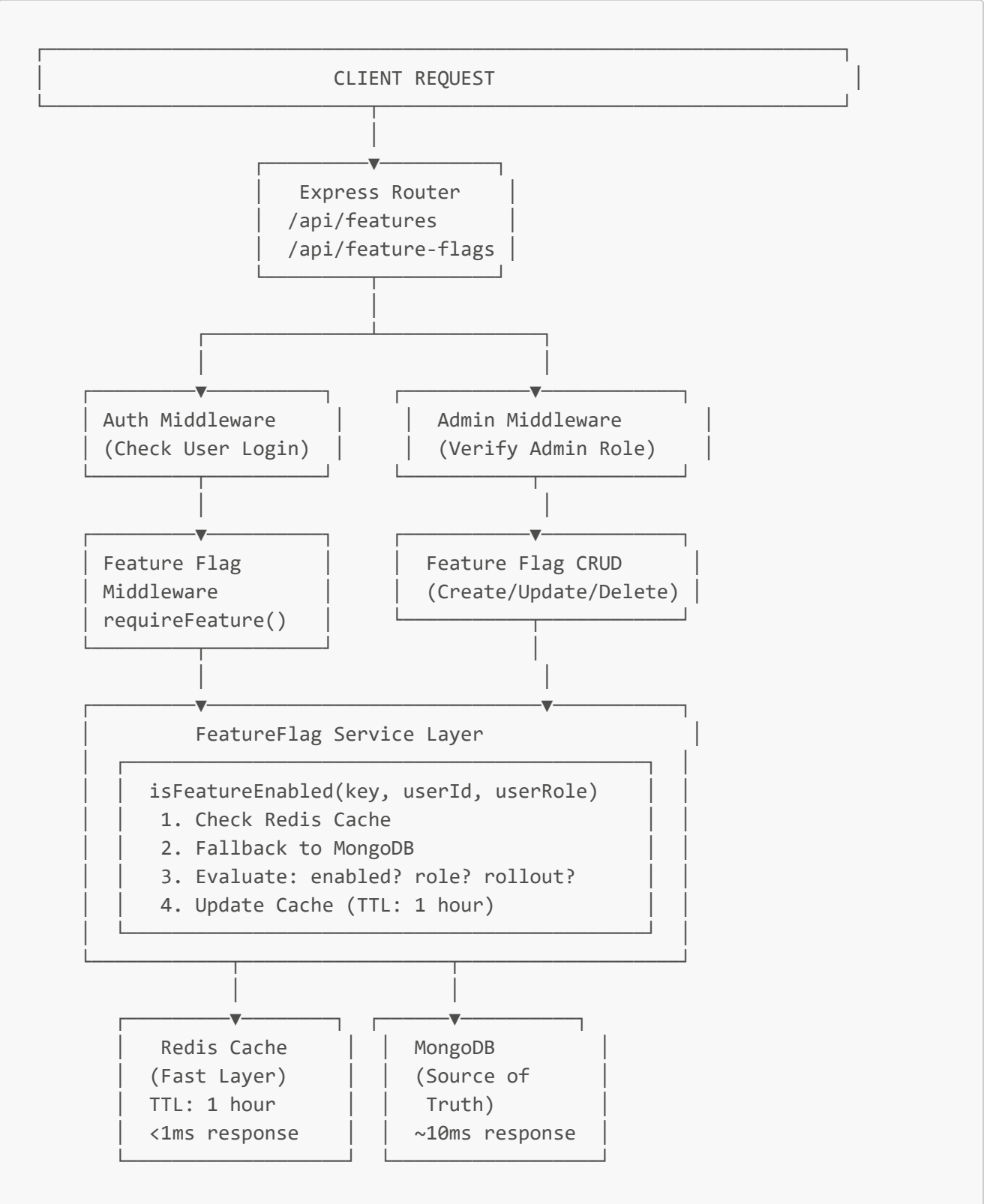
Feature Flag System - Architecture Overview

System Architecture



Data Flow Diagrams

1. **Feature Check Flow (Cached)**

```
User Request
    ↓
Auth Middleware (extract user info)
    ↓
requireFeature("ONLINE_PAYMENT")
    ↓
FeatureFlagService.isFeatureEnabled()
    ↓
Check Redis Cache → ☑ HIT
    ↓
Evaluate feature logic
    ↓
Return true/false → Proceed or 403
```

2. **Feature Check Flow (Uncached)**

```
User Request
    ↓
Auth Middleware
    ↓
requireFeature("ONLINE_PAYMENT")
    ↓
FeatureFlagService.isFeatureEnabled()
    ↓
Check Redis Cache → ❌ MISS
    ↓
Query MongoDB → Fetch flag data
    ↓
Store in Redis (TTL: 1 hour)
    ↓
Evaluate feature logic
    ↓
Return true/false → Proceed or 403
```

3. **Admin Update Flow**

```
Admin Request (PUT /api/feature-flags/ONLINE_PAYMENT)
    ↓
adminMiddleware (verify admin role)
    ↓
FeatureFlagService.updateFeatureFlag()
    ↓
Update MongoDB
    ↓
Invalidate Redis Cache (DEL key)
```

```
    ↓
Return success response
    ↓
Next request → Cache MISS → Fresh read from MongoDB
```

🎯 Feature Evaluation Logic

```
START: Check if feature X is enabled for user Y

1. Is feature globally enabled?
   ├─ NO  → ❌ DENY ACCESS (return 403)
   └─ YES → Continue to step 2

2. Is user in whitelist (allowedUserIds)?
   ├─ YES → ☑ ALLOW ACCESS (bypass all other checks)
   └─ NO  → Continue to step 3

3. Is user role in allowedRoles?
   ├─ NO  → ❌ DENY ACCESS (return 403)
   └─ YES → Continue to step 4

4. Calculate user's rollout percentile
   hash(userId) % 100 = userPercentile

5. Is userPercentile < rolloutPercentage?
   ├─ NO  → ❌ DENY ACCESS (not in rollout group)
   └─ YES → ☑ ALLOW ACCESS

END: Feature access granted
```

🔄 Cache Strategy

Cache Key Format

```
feature_flag:<FEATURE_KEY>

Examples:
- feature_flag:ONLINE_PAYMENT
- feature_flag:AI_CHATBOT
- feature_flag:FEATURED_MEDICINES
```

Cache Lifecycle

```
1. WRITE (on first read)
   MongoDB → Fetch data → Redis.setEx(key, 3600, data)

2. READ (subsequent requests)
   Redis.get(key) → Return cached data (instant)

3. INVALIDATE (on update/delete)
   Redis.del(key) → Force fresh read on next request

4. EXPIRY (automatic)
   After 1 hour → Redis auto-deletes → Next read refreshes cache
```

Cache Hit Ratio

```
Expected Performance:
- First request:  Cache MISS → ~10ms (MongoDB)
- Next requests:  Cache HIT  → <1ms (Redis)
- Hit ratio:      >95% (production workload)
```

Database Schema

MongoDB Collection: `featureflags`

```
{
  _id: ObjectId("..."),
  key: "ONLINE_PAYMENT",              // Unique, uppercase, indexed
  name: "Online Payment Gateway",
  description: "Enables payment processing",
  enabled: true,                      // Master switch
  allowedRoles: ["ADMIN", "CUSTOMER"], // Role-based access
  allowedUserIds: [ObjectId("...")],  // User whitelist
  rolloutPercentage: 100,             // 0-100
  createdAt: ISODate("2026-01-15"),
  updatedAt: ISODate("2026-01-20")
}
```

Indexes

```
{ key: 1 }                    // Unique index
{ enabled: 1, key: 1 }        // Compound index for queries
{ allowedRoles: 1 }           // Role-based queries
```

🌐 API Request/Response Examples

Example 1: Create Feature Flag (Admin)

```
POST /api/feature-flags
Authorization: Bearer <admin_token>
Content-Type: application/json

{
  "key": "FEATURED_MEDICINES",
  "name": "Featured Medicines",
  "enabled": true,
  "allowedRoles": ["ADMIN"],
  "rolloutPercentage": 100
}

→ Response:
{
  "success": true,
  "message": "Feature flag created successfully",
  "data": {
    "_id": "...",
    "key": "FEATURED_MEDICINES",
    "enabled": true,
    ...
  }
}
```

Example 2: Get User Features (Public)

```
GET /api/features
Authorization: Bearer <user_token>

→ Response:
{
  "success": true,
  "message": "User features retrieved successfully",
  "data": {
    "ONLINE_PAYMENT": true,
    "FEATURED_MEDICINES": false,
    "AI_CHATBOT": true
  }
}
```

Example 3: Protected Route

```
GET /api/featured-medicines
Authorization: Bearer <customer_token>

If FEATURED_MEDICINES is disabled for user:
→ Response:
{
  "success": false,
  "message": "Feature 'FEATURED_MEDICINES' is not available for your account",
  "statusCode": 403
}
```

Security Model

Authentication Flow

```
1. Request arrives with JWT token
2. CheckLoginMiddleware extracts:
   - req.user._id
   - req.user.role
   - req.user.email
3. Feature flag middleware uses this context
4. Service evaluates feature access
5. Return 401 (no auth) or 403 (no access)
```

Authorization Levels

```
Admin Routes:
- Create/Update/Delete flags
- View all flags
- Clear cache

Public Routes:
- Get own enabled features (authenticated users only)

Protected Routes:
- requireFeature() middleware checks per request
```

Performance Metrics

Latency (Expected)

```
┌──────────────────────────┬──────────────┐
│ Operation                │ Latency      │
├──────────────────────────┼──────────────┤
│ Redis cache hit          │ <1ms         │
│ Redis cache miss         │ ~10ms        │
│ MongoDB read             │ ~8-12ms      │
│ Feature evaluation       │ <5ms         │
│ Total (cached)           │ ~6ms         │
│ Total (uncached)         │ ~15-20ms     │
└──────────────────────────┴──────────────┘
```

Throughput (Estimated)

```
- Cached requests:   ~10,000 req/s
- Uncached requests: ~500 req/s
- CPU usage:         <5% (with caching)
- Memory:            ~50MB (Redis overhead)
```

🚀 Deployment Checklist

```
☑ MongoDB schema created
☑ Redis connection configured
☑ Service layer implemented
☑ Middleware created
☑ Admin routes added
☑ Public routes added
☑ Routes registered in main router
☑ TypeScript types defined
☑ Error handling implemented
☑ Cache strategy in place
☑ Seed script created
☑ Documentation written
☑ Postman collection provided
☑ Example routes created
```

📦 File Dependencies

```
FeatureFlag System Files:
│
├── Databases/
│   ├── Entities/featureFlag.Interface.ts  ← TypeScript types
│   ├── Schema/featureFlag.Schema.ts       ← Mongoose schema
```

```
│   └── Models/featureFlag.Models.ts       ← Mongoose model
│
├── Services/
│   └── featureFlag.Service.ts             ← Core business logic
│
├── Middlewares/
│   └── featureFlagMiddleware.ts           ← Route protection
│
├── Routers/Routers/
│   ├── featureFlag.Routes.ts              ← Admin APIs
│   └── features.Routes.ts                 ← Public API
│
├── Utils/
│   └── Roles.enum.ts                      ← Updated with PHARMACIST
│
├── config/
│   └── redis.ts                           ← Redis connection (existing)
│
├── scripts/
│   └── seedFeatureFlags.ts                ← Database seeding
│
└── examples/
    └── featureFlag.examples.ts            ← Usage examples
```

🎓 Usage Patterns

Pattern 1: Simple Route Protection

```
router.get('/feature', requireFeature('FEATURE_KEY'), controller);
```

Pattern 2: Multiple Middleware

```
router.post(
  '/payment',
  customersMiddleware,
  requireFeature('ONLINE_PAYMENT'),
  rateLimit,
  processPayment
);
```

Pattern 3: Conditional Logic

```
const hasFeature = await FeatureFlagService.isFeatureEnabled(
  'FEATURE_KEY', userId, userRole
```

```
);

if (hasFeature) {
  // Show premium features
}
```

Pattern 4: Frontend Integration

```
// Fetch on app load
const features = await api.get('/api/features');

// Use throughout app
{features.ONLINE_PAYMENT && <PayButton />}
```

🎉 Summary

You now have a **complete, production-ready feature flag system** with:

1. ☑ **Database-driven** configuration (MongoDB)
2. ☑ **High-performance** caching (Redis)
3. ☑ **Flexible access control** (roles + whitelist + rollout)
4. ☑ **Zero-downtime** feature deployment
5. ☑ **Comprehensive APIs** (Admin + Public)
6. ☑ **Route protection** middleware
7. ☑ **Seed script** for initialization
8. ☑ **Full documentation** and examples

Start using it:

```
# 1. Seed initial flags
bun run scripts/seedFeatureFlags.ts

# 2. Start server
bun run dev

# 3. Test with Postman
Import: FeatureFlag_API_Tests.postman_collection.json

# 4. Protect your routes
import { requireFeature } from './Middlewares/featureFlagMiddleware';
router.get('/my-feature', requireFeature('MY_FEATURE'), controller);
```

Questions? See `FEATURE_FLAG_SYSTEM.md` or `FEATURE_FLAG_QUICK_REFERENCE.md` 🚀