workflow of each API endpoint in the auth service, including request/response formats and performance considerations.

1. **Login API**

Code

Endpoint: POST /api/v1/auth/login

Rate Limit: 5 attempts per IP in 15 minutes

Request:

Code

{

"email": "user@example.com",

"password": "SecurePass123!",

"institutionId": "550e8400-e29b-41d4-a716-446655440000"

}

Response (Success - 200):

Code

{

"user": {

"id": "123e4567-e89b-12d3-a456-426614174000",

"email": "user@example.com",

"firstName": "John",

"lastName": "Doe",

"roles": ["student"]

},

"tokens": {

"accessToken": "eyJhbGciOiJIUzI1NiIs...",

"refreshToken": "eyJhbGciOiJIUzI1NiIs..."

}

}

Response (Error - 401):

Code

{

"status": "error",

"message": "Invalid credentials"

}

Workflow:

1. Request validation using Joi schema
2. Rate limiting check (Redis-based)
3. User lookup via user service
4. Password verification with bcrypt
5. Token generation (JWT)
6. Store refresh token in Redis
7. Publish login event to RabbitMQ
8. Return user data and tokens

Performance Optimizations:

* Redis-based rate limiting
* Asynchronous event publishing
* Bcrypt with 12 rounds (balance between security and performance)
* Redis connection pooling
* JWT token caching

1. **Refresh Token API**

Endpoint: POST /api/v1/auth/refresh-token

Rate Limit: 100 attempts per IP per hour

Request:

{

"refreshToken": "eyJhbGciOiJIUzI1NiIs..."

}

Response (Success - 200):

{

"accessToken": "eyJhbGciOiJIUzI1NiIs...",

"refreshToken": "eyJhbGciOiJIUzI1NiIs..." // New refresh token

}

Response (Error - 401):

{

"status": "error",

"message": "Invalid refresh token"

}

Workflow:

1. Validate refresh token JWT
2. Check token type (must be 'refresh')
3. Verify token exists in Redis
4. Get user data from user service
5. Generate new tokens
6. Rotate refresh token in Redis
7. Return new tokens

Performance Optimizations:

* Redis for token storage and validation
* Parallel token rotation operations
* JWT verification without database lookup
* Token blacklisting through Redis expiry

1. **Logout API**

Endpoint: POST /api/v1/auth/logout

Authentication: Required (Bearer token)

Request:

{

"refreshToken": "eyJhbGciOiJIUzI1NiIs..."

}

Response (Success - 200):

{

"message": "Logged out successfully"

}

Workflow:

1. Delete refresh token from Redis
2. No additional validation needed
3. Token automatically invalidated

Performance Optimizations:

* Simple Redis deletion operation
* No database operations required
* Fire-and-forget token deletion

1. **Validate Token API** (Internal Microservice Route)

Code

Endpoint: POST /api/v1/auth/validate-token

Authentication: Required (Bearer token)

Request: Headers:

Code

Authorization: Bearer eyJhbGciOiJIUzI1NiIs...

Response (Success - 200):

Code

{

"valid": true,

"decoded": {

"userId": "123e4567-e89b-12d3-a456-426614174000",

"institutionId": "550e8400-e29b-41d4-a716-446655440000",

"roles": ["student"],

"type": "access",

"iat": 1672747348,

"exp": 1672748248

}

}

Response (Invalid Token - 200):

Code

{

"valid": false,

"error": "Invalid token"

}

Workflow:

1. Extract token from Authorization header
2. Verify JWT signature
3. Check token type (must be 'access')
4. Optional: Validate user status
5. Return decoded token data

Performance Optimizations:

* JWT verification without database lookup
* Optional user validation (configurable)
* No token storage required
* Stateless validation

General Performance Features:

1. **Caching**:
   * Redis for token storage
   * Connection pooling for Redis and RabbitMQ
   * In-memory caching of frequently used data
2. **Security with Performance**:
   * JWT for stateless authentication
   * Redis for token revocation
   * Rate limiting with sliding windows
   * Bcrypt with optimal rounds
3. **Scalability**:
   * Stateless JWT tokens
   * Distributed rate limiting
   * Event-driven architecture
   * Microservice communication
4. **Error Handling**:
   * Fast fail on validation
   * Proper error status codes
   * Detailed logging in development
   * Sanitized errors in production
5. **Monitoring**:
   * Health check endpoint
   * Winston logging
   * Error tracking
   * Rate limit monitoring
6. **Network Optimization**:
   * Response compression
   * Connection pooling
   * Efficient token storage
   * Minimal database queries

To monitor performance:

1. Use the Winston logger to track:
   * API response times
   * Error rates
   * Token operations
   * Service communication
2. Monitor Redis metrics:
   * Token storage size
   * Operation latency
   * Cache hit rates
3. Watch RabbitMQ:
   * Queue sizes
   * Message processing times
   * Connection status