1. **Core Authentication**:
   * Login with email/password
   * JWT token generation (access and refresh tokens)
   * Token refresh mechanism
   * Logout functionality
   * Token validation
2. **Security Features**:
   * Password hashing with bcrypt
   * Rate limiting for login and token refresh
   * JWT token rotation
   * Redis-based token storage
   * CORS protection
   * Helmet security headers
   * Request compression
3. **Error Handling**:
   * Custom error classes
   * Centralized error handling
   * Production-safe error messages
   * Structured logging with Winston
4. **Microservice Integration**:
   * User service client for user operations
   * RabbitMQ integration for event publishing
   * Redis for token and rate limit storage
5. **Middleware**:
   * Rate limiting
   * Error handling
   * Request parsing
   * Security headers
   * CORS configuration
6. **API Routes**:
   * **/api/v1/auth/login**
   * **/api/v1/auth/refresh-token**
   * **/api/v1/auth/logout**
   * **/api/v1/auth/validate-token**
   * **/health** endpoint for service health checks

The service is well-structured, follows best practices, and includes all necessary security measures. To run the service:

1. Set up environment variables:
   * **JWT\_SECRET**
   * **JWT\_REFRESH\_SECRET**
   * **REDIS\_HOST**
   * **REDIS\_PORT**
   * **REDIS\_PASSWORD**
   * **RABBITMQ\_URL**
   * **USER\_SERVICE\_URL**
   * **ALLOWED\_ORIGINS** (optional)
   * **NODE\_ENV**
   * **LOG\_LEVEL** (optional)

npm install

# Development

npm run dev

# Production

npm start

1. **Auth Routes (auth.routes.js)**:
   * POST **/login** - User login with email/password
   * POST **/refresh-token** - Get new access token using refresh token
   * POST **/logout** - User logout
   * POST **/validate-token** - Internal route for token validation
2. **Auth Controller (auth.controller.js)**:
   * Handles all route logic
   * Validates input
   * Calls auth service methods
   * Handles errors appropriately
3. **Auth Service (auth.service.js)**:
   * Core authentication logic
   * JWT token generation and validation
   * Password hashing and verification
   * Integration with Redis for token management
   * Integration with message queue for events
   * Communication with user service
4. **User Service Client (user.service.js)**:
   * Handles communication with user service
   * Methods for user lookup and updates
   * Error handling for service communication
5. **Error Handling (errors.js)**:
   * Comprehensive error classes
   * Proper HTTP status codes
   * Operational vs Programming errors
6. **Validators (auth.validator.js)**:
   * Input validation schemas
   * Custom error messages
   * Required fields validation
7. **Configuration (app.config.js)**:
   * Service configuration
   * JWT settings
   * Redis settings
   * Security settings
   * Message queue configuration
   * Service URLs

The auth service is now ready to:

1. Handle user authentication
2. Manage JWT tokens
3. Integrate with the user service
4. Handle token refresh and validation
5. Manage user sessions
6. Emit authentication events
7. Validate and secure routes

To use the service:

1. Configure environment variables (JWT secrets, Redis connection, etc.)
2. Ensure Redis is running for token management
3. Ensure RabbitMQ is running for event handling
4. Make sure the user service is accessible
5. Start the service using the app entry point