API Server Application

Development Specification

Project Overview

Design and implement a production-ready, GDPR-compliant RESTful API server application using modern architectural patterns and security best practices.

Core Technical Requirements

Architecture & Language

- **Language:** Go (Golang)
- **Architecture:** Hexagonal Architecture (Ports and Adapters pattern)
    - Clear separation between domain logic, application services, and infrastructure
    - Define ports (interfaces) for external dependencies
    - Implement adapters for databases, external services, and HTTP handlers
- **API Specification:** OpenAPI Specification (OAS) v3.0 or v3.1
- **Development Approach:** API-first design (define OpenAPI spec before implementation)

Deployment & Runtime

- **Containerization:** Application must run in Docker containers
    - Provide Dockerfile with multi-stage builds
    - Include docker-compose.yml for local development with nginx load balancer
- **Architecture:** Stateless application design
    - Must support horizontal scaling
    - Compatible with external load balancer deployment
    - No server-side session storage in memory

Load Balancer Configuration

- **External Load Balancer:** nginx (for development/testing)
    - Must work with production load balancers (F5 BIG-IP, Citrix NetScaler, HAProxy)
    - Provide nginx configuration for SSL termination, health checks, load balancing
- **Health Check Endpoint:** /health or /healthz for load balancer monitoring
- **Readiness Endpoint:** /ready to indicate when service can accept traffic

Security & Session Management

- **Protocol:** HTTPS only
- **Authentication:** JWT (JSON Web Tokens)
    - Access tokens stored in HTTP-only, Secure cookies
    - Refresh tokens stored in HTTP-only, Secure cookies (separate cookie)
    - Implement proper cookie attributes (Secure, HttpOnly, SameSite=Strict)

JWT Token Management Strategy

- **Access Token:** Short-lived (15 minutes recommended)
- **Refresh Token:** Longer-lived (7-14 days recommended)
- **Token Refresh Mechanism:**
    - Access token must be refreshed after every successful API request
    - Return new access token in response header or refresh in middleware
    - If token is idle (no requests) for X configurable minutes, token expires
    - Track last activity timestamp in JWT claims
    - Implement sliding window expiration
- **Token Claims:** Include user ID, email, roles, issued_at, expires_at, last_activity
- **Token Signing:** Use RS256 (RSA) or ES256 (ECDSA) - retrieve keys from Vault

Secrets Management

- **HashiCorp Vault Integration:**
    - Database credentials
    - JWT signing keys (RSA private/public key pair)
    - OAuth client secrets (Google, Apple)
    - Encryption keys and API keys
- Implement proper Vault authentication (AppRole recommended)
- Handle secret rotation gracefully
- Implement retry logic for Vault connection failures

Data Storage

- **Database:** MariaDB
    - Use connection pooling (configure max connections)
    - Implement proper transaction management
    - Support for database migrations (suggest golang-migrate or goose)
    - Repository pattern for data access
    - Prepared statements for SQL injection prevention

Authentication & Authorization

Authentication Methods

1. Email/Password Registration & Login

- Email verification workflow (send verification email with token)
- Password requirements: minimum 12 characters, uppercase, lowercase, numbers, special chars
- Password hashing using argon2id or bcrypt (cost 12+)
- Password reset functionality with time-limited tokens
- Account lockout after 5 failed login attempts (15 minute lockout)

2. OAuth 2.0 Social Login

- Google Sign-In (OAuth 2.0)
- Apple Sign-In (OAuth 2.0)
- Handle account linking scenarios (link social account to existing email account)
- Store OAuth provider tokens securely if needed for API access

Two-Factor Authentication (2FA)

- **TOTP Support:** Time-based One-Time Password
  - Compatible with Google Authenticator and Microsoft Authenticator
  - Use standard TOTP algorithm (RFC 6238)
- **Features:**
  - QR code generation for 2FA enrollment (use PNG format)
  - Generate and securely store 10 backup codes per user (hashed)
  - 2FA recovery flow using backup codes
  - Option to disable/re-enable 2FA
  - Enforce 2FA for admin roles (configurable)
- **Security:** Store TOTP secrets encrypted in database

Authorization

- **Role-Based Access Control (RBAC):**
  - Support for roles: user, admin, super_admin (extensible)
  - Permission-based access control
  - Implement middleware for role/permission checking
  - Include roles in JWT claims

GDPR Compliance Requirements

User Rights Implementation

- **Right to Access:**
  - Endpoint to export all user data in JSON/XML format
  - Include all personal data, activity logs, authentication history
- **Right to Rectification:**
  - Endpoints for users to update their personal information
  - Audit log of data changes
- **Right to Erasure (Right to be Forgotten):**
  - /users/{id}/delete endpoint with hard/soft delete options
  - Anonymize user data instead of deletion (configurable)
  - Cascade deletion handling for related records
  - Retain minimal data for legal/compliance purposes
- **Right to Data Portability:**
  - Export user data in machine-readable format (JSON)
  - Include all user-generated content
- **Right to Restrict Processing:**
  - Account suspension/freeze functionality
  - Temporary processing restrictions
- **Consent Management:**
  - Store consent records (timestamp, version, IP address)
  - Allow users to withdraw consent
  - Granular consent options (marketing, analytics, etc.)
  - Consent versioning

GDPR Technical Requirements

- **Data Minimization:** Only collect necessary data
- **Encryption:**
  - Data at rest: Encrypt PII in database
  - Data in transit: HTTPS only
- **Audit Logging:** Log all data access and modifications
  - Timestamp, User ID, Action performed, IP address, Changed fields
- **Data Retention:**
  - Configurable retention policies
  - Automatic data purging after retention period
  - Anonymization of old data

API Endpoints (Required)

Authentication & User Management

- **POST /api/v1/auth/register** - Email registration
- **POST /api/v1/auth/login** - Login with credentials
- **POST /api/v1/auth/logout** - Logout (invalidate tokens)
- **POST /api/v1/auth/refresh** - Refresh access token
- **POST /api/v1/auth/forgot-password** - Request password reset
- **POST /api/v1/auth/reset-password** - Reset password with token
- **POST /api/v1/auth/verify-email** - Verify email address
- **POST /api/v1/auth/resend-verification** - Resend verification email

OAuth

- **GET /api/v1/auth/google** - Initiate Google OAuth
- **GET /api/v1/auth/google/callback** - Google OAuth callback
- **GET /api/v1/auth/apple** - Initiate Apple OAuth
- **GET /api/v1/auth/apple/callback** - Apple OAuth callback

Two-Factor Authentication

- **POST /api/v1/auth/2fa/enable** - Enable 2FA (returns QR code)
- **POST /api/v1/auth/2fa/verify** - Verify 2FA code during setup
- **POST /api/v1/auth/2fa/disable** - Disable 2FA
- **GET /api/v1/auth/2fa/backup-codes** - Generate new backup codes
- **POST /api/v1/auth/2fa/validate** - Validate 2FA code during login

GDPR Endpoints

- **GET /api/v1/users/me/data** - Export all user data
- **GET /api/v1/users/me/consent** - Get consent history
- **POST /api/v1/users/me/consent** - Update consent preferences
- **POST /api/v1/users/me/anonymize** - Anonymize account data

Testing Requirements

Unit Tests

- **Coverage:** 80%+ code coverage
- **Test all layers:**
    - Domain logic (business rules)
    - Use cases/application services
    - Adapters (repositories, HTTP handlers)
- **Mock external dependencies:**
    - Database (use sqlmock or testcontainers)
    - Vault client, OAuth providers, Email service
- **Test scenarios:**
    - Authentication flows, JWT generation/validation, Token refresh logic
    - 2FA enrollment, Password reset, GDPR data export/deletion

Monitoring & Observability

Prometheus Metrics

Expose metrics on /metrics endpoint:

- **HTTP Metrics:** Request duration, count by endpoint/status, active requests
- **Application Metrics:** Active users, login attempts, token refresh count
- **Database Metrics:** Connection pool stats, query duration
- **Business Metrics:** User registrations, GDPR requests

Grafana Dashboards

Provide JSON dashboard definitions:

- **API Overview Dashboard:** Request rate, error rate, response times
- **Authentication Dashboard:** Login success/failure rate, active sessions
- **System Health Dashboard:** CPU/memory, database connection pool
- **GDPR Compliance Dashboard:** Data export/deletion requests

Key Go Libraries

- **HTTP Router:** chi (recommended) or gorilla/mux
- **OpenAPI:** oapi-codegen (generates server stubs from OpenAPI spec)
- **JWT:** golang-jwt/jwt/v5
- **OAuth:** golang.org/x/oauth2
- **2FA/TOTP:** pquerna/otp
- **Database:** go-sql-driver/mysql, jmoiron/sqlx
- **Migrations:** golang-migrate/migrate/v4
- **Vault:** hashicorp/vault/api
- **Metrics:** prometheus/client_golang
- **Logging:** uber-go/zap or sirupsen/logrus
- **Testing:** stretchr/testify

Non-Functional Requirements

- **Performance:** Handle 1000+ concurrent requests per instance
- **Reliability:** Graceful shutdown, retry logic, circuit breaker pattern
- **Scalability:** Horizontal scaling, stateless design, connection pooling
- **Maintainability:** Clean code, comprehensive documentation, structured logging

Application Code Structure

```
/
├── cmd/api/                # Application entry point
├── internal/
│   ├── domain/             # Core business logic
│   ├── ports/              # Interfaces (ports)
│   ├── application/        # Use cases
│   ├── adapters/           # Implementations
│   │   ├── http/           # HTTP handlers
│   │   ├── repository/     # Database
│   │   ├── vault/          # Vault client
│   │   └── oauth/          # OAuth providers
│   └── config/             # Configuration
├── migrations/             # Database migrations
├── api/openapi.yaml        # OpenAPI spec
├── docker/
│   ├── Dockerfile
│   ├── docker-compose.yml
│   └── nginx/nginx.conf
└── tests/                  # Integration tests
```

Additional Functionality

This section is reserved for specifying additional features and functionality beyond the core requirements. Use this space to document:

Custom Business Logic

Define any domain-specific business rules, workflows, or processes that are unique to your application:

- Example: Multi-tenancy support with organization management
- Example: Subscription tiers and feature gating
- Example: Approval workflows for specific operations
- Example: Custom notification rules and alert systems

Additional API Endpoints

List any additional API endpoints specific to your application domain:

- Example: POST /api/v1/organizations - Create organization
- Example: GET /api/v1/teams/{id}/members - List team members
- Example: POST /api/v1/webhooks - Configure webhooks
- Example: GET /api/v1/analytics/dashboard - Get analytics data

Third-Party Integrations

Specify any external services or APIs to integrate:

- Example: Payment processing (Stripe, PayPal)
- Example: Email service provider (SendGrid, AWS SES)
- Example: SMS notifications (Twilio)
- Example: File storage (AWS S3, Google Cloud Storage)
- Example: Analytics platforms (Google Analytics, Mixpanel)

Advanced Features

Document any advanced features or capabilities:

- Example: Real-time notifications via WebSockets
- Example: Background job processing with queues
- Example: GraphQL API alongside REST
- Example: Full-text search capabilities (Elasticsearch)
- Example: Caching layer (Redis) for performance
- Example: API versioning strategy

Compliance & Security Add-ons

Additional compliance requirements or security measures:

- Example: SOC 2 compliance requirements
- Example: HIPAA compliance for health data
- Example: PCI DSS for payment card data
- Example: IP whitelisting/blacklisting
- Example: Advanced threat detection and prevention

Reporting & Analytics

Reporting capabilities and analytics requirements:

- Example: Automated report generation (PDF/Excel)
- Example: Custom dashboard creation for users

- Example: Data export capabilities (CSV, JSON, XML)
- Example: Usage analytics and metrics tracking

Administration & Management

Administrative features and management tools:

- Example: Admin panel for user management
- Example: Feature flags and A/B testing support
- Example: System configuration management API
- Example: Audit trail viewer and management

Notes & Special Requirements

Use this section to document any special considerations, constraints, or notes:

- Development timeline and milestones
- Budget constraints or resource limitations
- Specific technology preferences or restrictions
- Known challenges or technical debt
- Future expansion plans

Implementation Phases

Phase 1: Foundation (Week 1)

- OpenAPI specification
- Project structure setup
- Docker environment
- Database schema and migrations

Phase 2: Core Authentication (Week 2)

- Email/password registration and login
- JWT generation and validation
- Token refresh mechanism with idle timeout

Phase 3: OAuth & 2FA (Week 3)

- Google and Apple OAuth integration
- 2FA enrollment and verification

Phase 4: GDPR & Compliance (Week 4)

- Data export and deletion endpoints
- Consent management and audit logging

Phase 5: Monitoring & Production (Week 5)

- Prometheus metrics and Grafana dashboards
- nginx load balancer setup and security hardening

Phase 6: Testing & QA (Week 6)

- Comprehensive unit and integration tests
- Load testing and security testing

Success Criteria

- All endpoints defined in OpenAPI spec implemented
- 80%+ test coverage achieved
- All security requirements met
- GDPR compliance verified
- Load balancer integration working
- Monitoring dashboards operational
- Application runs in containers
- Horizontal scaling verified

Getting Started

Begin by:

- Creating the detailed OpenAPI 3.x specification with all endpoints
- Setting up the project structure following hexagonal architecture
- Implementing the JWT middleware with token refresh logic
- Building the authentication flows (email, OAuth, 2FA)
- Implementing GDPR compliance features
- Setting up monitoring and observability
- Writing comprehensive tests
- Creating deployment documentation

Remember: API-first approach means the OpenAPI specification drives the implementation.