Encryptz ERP - Technical Document for Developers

Version: 1.0
Date: 23rd November 2025
Project: Encryptz Accountz ERP - Core Module
Audience: Developers, Technical Team, System Architects

Table of Contents

System Architecture

Overview

The system follows a **layered architecture** with clear separation of concerns:

```
┌──────────────────────────────────────────┐
│          Angular Frontend (UI)           │
│          Angular 17+ / TypeScript        │
└──────────────────┬───────────────────────┘
                   │ HTTP/REST
┌──────────────────▼───────────────────────┐
│       .NET 8 Web API (Controllers)       │
│       RESTful API Endpoints              │
└──────────────────┬───────────────────────┘
                   │
┌──────────────────▼───────────────────────┐
│       Business Logic Layer (Services)    │
│       Domain Logic / Validation          │
└──────────────────┬───────────────────────┘
                   │
┌──────────────────▼───────────────────────┐
│       Repository Layer (Data Access)     │
│       ADO.NET / Npgsql                   │
└──────────────────┬───────────────────────┘
                   │
┌──────────────────▼───────────────────────┐
│       PostgreSQL Database                │
│      Schema: core, admin, acct          │
└─────────────────────────────────────────┘
```

Technology Stack

Backend:

- **Framework**: .NET 8.0
- **Language**: C# 12
- **ORM**: Entity Framework Core (with ADO.NET for performance)
- **Database**: PostgreSQL 14+
- **Authentication**: JWT Bearer Tokens
- **Mapping**: AutoMapper
- **Logging**: Built-in ILogger

Frontend:

- **Framework**: Angular 17+
- **Language**: TypeScript
- **UI Library**: Angular Material / PrimeNG
- **State Management**: RxJS Observables
- **HTTP Client**: Angular HttpClient

Database:

- **RDBMS**: PostgreSQL 14+
- **Connection**: Npgsql
- **Schemas**: `core`, `admin`, `acct`

Database Schema

Schema Organization

The database is organized into three main schemas:

1. **`core`**: Core system tables (users, businesses, roles, permissions)
2. **`admin`**: Administrative tables (OTP, audit logs)
3. **`acct`**: Accounting module tables (accounts, transactions, vouchers)

Core Schema Tables

1. `core.users`

Stores user account information.

Column	Type	Description
`user_id`	UUID	Primary key, auto-generated
`user_handle`	VARCHAR(50)	Unique user identifier
`full_name`	VARCHAR(200)	User's full name
`email`	VARCHAR(256)	Email address (nullable, unique)

Column	Type	Description
`hashed_password`	TEXT	BCrypt hashed password
`mobile_country_code`	VARCHAR(10)	Country code for mobile
`mobile_number`	VARCHAR(20)	Mobile number (nullable, unique)
`pan_card_number_encrypted`	BYTEA	Encrypted PAN card number
`aadhar_number_encrypted`	BYTEA	Encrypted Aadhar number (nullable)
`pan_card_number_hash`	BYTEA	SHA256 hash for duplicate detection
`is_active`	BOOLEAN	Account active status
`created_at_utc`	TIMESTAMPTZ	Creation timestamp
`updated_at_utc`	TIMESTAMPTZ	Last update timestamp

Indexes:

- Unique index on `user_handle`
- Unique index on `email` (where not null)
- Unique index on `(mobile_country_code, mobile_number)` (where not null)
- Unique index on `pan_card_number_hash`

Relationships:

- One-to-many with `core.businesses` (created_by_user_id)
- Many-to-many with `core.businesses` via `core.user_business_roles`

2. `core.businesses`

Stores business/company information.

Column	Type	Description
`business_id`	UUID	Primary key
`business_name`	VARCHAR(250)	Business name
`business_code`	VARCHAR(50)	Unique business code
`is_active`	BOOLEAN	Active status
`gstin`	VARCHAR(15)	GST Identification Number
`tan_number`	VARCHAR(10)	TAN Number
`address_line1`	VARCHAR(250)	Address line 1
`address_line2`	VARCHAR(250)	Address line 2
`city`	VARCHAR(100)	City
`state_id`	INTEGER	State ID
`pin_code`	VARCHAR(10)	PIN code
`country_id`	INTEGER	Country ID

Column	Type	Description
`created_by_user_id`	UUID	Creator user ID
`created_at_utc`	TIMESTAMPTZ	Creation timestamp
`updated_by_user_id`	UUID	Last updater user ID
`updated_at_utc`	TIMESTAMPTZ	Last update timestamp

Indexes:

- Unique index on `business_code`

Relationships:

- Foreign key to `core.users` (created_by_user_id, updated_by_user_id)
- One-to-many with `core.user_subscriptions`
- Many-to-many with `core.users` via `core.user_business_roles`

3. `core.subscription_plans`

Stores subscription plan definitions.

Column	Type	Description
`plan_id`	SERIAL	Primary key
`plan_name`	VARCHAR(100)	Plan name
`description`	VARCHAR(500)	Plan description
`price`	DECIMAL(18,2)	Plan price
`max_users`	INTEGER	Maximum users allowed
`max_businesses`	INTEGER	Maximum businesses allowed
`is_publicly_visible`	BOOLEAN	Public visibility
`is_active`	BOOLEAN	Active status

Relationships:

- One-to-many with `core.user_subscriptions`
- Many-to-many with `core.permissions` via `core.subscription_plan_permissions`

4. `core.user_subscriptions`

Links businesses to subscription plans.

Column	Type	Description
`subscription_id`	UUID	Primary key
`business_id`	UUID	Business ID
`plan_id`	INTEGER	Subscription plan ID
`status`	VARCHAR(50)	Status (Active, Expired, Trial, Cancelled)

Column	Type	Description
`start_date_utc`	TIMESTAMPTZ	Subscription start date
`end_date_utc`	TIMESTAMPTZ	Subscription end date
`trial_ends_at_utc`	TIMESTAMPTZ	Trial end date (nullable)
`created_at_utc`	TIMESTAMPTZ	Creation timestamp
`updated_at_utc`	TIMESTAMPTZ	Last update timestamp

Relationships:

- Foreign key to `core.businesses` (business_id)
- Foreign key to `core.subscription_plans` (plan_id)

5. `core.roles`

Stores role definitions.

Column	Type	Description
`role_id`	SERIAL	Primary key
`role_name`	VARCHAR(100)	Role name
`description`	VARCHAR(500)	Role description
`is_system_role`	BOOLEAN	System role flag

Relationships:

- Many-to-many with `core.permissions` via `core.role_permissions`
- Many-to-many with `core.users` and `core.businesses` via `core.user_business_roles`

6. `core.permissions`

Stores permission definitions.

Column	Type	Description
`permission_id`	SERIAL	Primary key
`permission_key`	VARCHAR(100)	Unique permission key
`description`	VARCHAR(500)	Permission description
`menu_item_id`	INTEGER	Associated menu item (nullable)
`module_id`	INTEGER	Module ID

Indexes:

- Unique index on `permission_key`

Relationships:

- Foreign key to `core.modules` (module_id)
- Foreign key to `core.menu_items` (menu_item_id, nullable)

- Many-to-many with `core.roles` via `core.role_permissions`
- Many-to-many with `core.subscription_plans` via `core.subscription_plan_permissions`

7. `core.modules`

Stores system module definitions.

Column	Type	Description
`module_id`	SERIAL	Primary key
`module_name`	VARCHAR(100)	Module name
`is_system_module`	BOOLEAN	System module flag
`is_active`	BOOLEAN	Active status

Relationships:

- One-to-many with `core.menu_items`
- One-to-many with `core.permissions`

8. `core.menu_items`

Stores menu item definitions for dynamic UI.

Column	Type	Description
`menu_item_id`	SERIAL	Primary key
`module_id`	INTEGER	Module ID
`parent_menu_item_id`	INTEGER	Parent menu item (nullable)
`menu_text`	VARCHAR(100)	Menu display text
`menu_url`	VARCHAR(250)	Menu URL
`icon_class`	VARCHAR(100)	Icon CSS class
`display_order`	INTEGER	Display order
`is_active`	BOOLEAN	Active status

Relationships:

- Foreign key to `core.modules` (module_id)
- Self-referencing foreign key (parent_menu_item_id)

9. `core.role_permissions`

Junction table linking roles to permissions.

Column	Type	Description
`role_id`	INTEGER	Role ID (composite key)
`permission_id`	INTEGER	Permission ID (composite key)

Relationships:

- Foreign key to `core.roles` (role_id)
- Foreign key to `core.permissions` (permission_id)

10. `core.user_business_roles`

Junction table linking users to roles within businesses.

Column	Type	Description
`user_id`	UUID	User ID (composite key)
`business_id`	UUID	Business ID (composite key)
`role_id`	INTEGER	Role ID (composite key)

Relationships:

- Foreign key to `core.users` (user_id)
- Foreign key to `core.businesses` (business_id)
- Foreign key to `core.roles` (role_id)

11. `core.user_businesses`

Stores user-business associations with default business flag.

Column	Type	Description
`user_id`	UUID	User ID (composite key)
`business_id`	UUID	Business ID (composite key)
`is_default`	BOOLEAN	Default business flag
`created_at_utc`	TIMESTAMPTZ	Creation timestamp

Indexes:

- Unique partial index on `(user_id, is_default)` where `is_default = true`

Relationships:

- Foreign key to `core.users` (user_id)
- Foreign key to `core.businesses` (business_id)

12. `core.audit_logs`

Stores audit trail information.

Column	Type	Description
`audit_log_id`	BIGSERIAL	Primary key
`user_id`	UUID	User who made the change
`operation`	VARCHAR(20)	Operation type (INSERT, UPDATE, DELETE)

Column	Type	Description
`table_name`	VARCHAR(100)	Table name
`record_id`	VARCHAR(100)	Record identifier
`change_description`	TEXT	Change description
`changed_at_utc`	TIMESTAMPTZ	Change timestamp

Relationships:

- Foreign key to `core.users` (user_id)

Admin Schema Tables

1. `admin.one_time_passwords`

Stores OTP for authentication.

Column	Type	Description
`otp_id`	BIGSERIAL	Primary key
`login_identifier`	VARCHAR(100)	Email or mobile
`otp`	VARCHAR(6)	OTP code
`expiry_time_utc`	TIMESTAMPTZ	Expiry timestamp
`is_used`	BOOLEAN	Used flag
`created_at_utc`	TIMESTAMPTZ	Creation timestamp

Indexes:

- Index on `login_identifier` for fast lookups

Accounting Schema Tables

1. `acct.account_types`

Stores account type definitions.

Column	Type	Description
`account_type_id`	SERIAL	Primary key
`account_type_name`	VARCHAR(50)	Account type name
`normal_balance`	VARCHAR(2)	Normal balance (Dr/Cr)

Account Types:

- Asset (Dr)
- Liability (Cr)
- Equity (Cr)
- Income (Cr)
- Expense (Dr)

2. `acct.chart_of_accounts`

Stores chart of accounts for each business.

Column	Type	Description
`account_id`	UUID	Primary key
`business_id`	UUID	Business ID
`account_type_id`	INTEGER	Account type ID
`parent_account_id`	UUID	Parent account (nullable)
`account_code`	VARCHAR(20)	Account code
`account_name`	VARCHAR(200)	Account name
`description`	VARCHAR(500)	Description
`is_active`	BOOLEAN	Active status
`is_system_account`	BOOLEAN	System account flag
`created_at_utc`	TIMESTAMPTZ	Creation timestamp
`updated_at_utc`	TIMESTAMPTZ	Last update timestamp

Indexes:

- Unique index on `(business_id, account_code)`
- Unique index on `(business_id, account_name)`

Relationships:

- Foreign key to `core.businesses` (business_id)
- Foreign key to `acct.account_types` (account_type_id)
- Self-referencing foreign key (parent_account_id)

3. `acct.transaction_headers`

Stores transaction header information (journal entries).

Column	Type	Description
`transaction_header_id`	UUID	Primary key
`business_id`	UUID	Business ID
`transaction_date`	DATE	Transaction date
`reference_number`	VARCHAR(100)	Reference number
`description`	VARCHAR(500)	Description
`created_by_user_id`	UUID	Creator user ID
`created_at_utc`	TIMESTAMPTZ	Creation timestamp

Relationships:

- Foreign key to `core.businesses` (business_id)
- Foreign key to `core.users` (created_by_user_id)
- One-to-many with `acct.transaction_details`

4. `acct.transaction_details`

Stores transaction detail lines (debit/credit entries).

Column	Type	Description
`transaction_detail_id`	BIGSERIAL	Primary key
`transaction_header_id`	UUID	Transaction header ID
`account_id`	UUID	Account ID
`debit_amount`	DECIMAL(18,2)	Debit amount
`credit_amount`	DECIMAL(18,2)	Credit amount

Constraints:

- Check constraint: Either debit_amount > 0 OR credit_amount > 0 (not both)

Relationships:

- Foreign key to `acct.transaction_headers` (transaction_header_id)
- Foreign key to `acct.chart_of_accounts` (account_id)

5. `acct.vouchers`

Stores voucher information.

Column	Type	Description
`voucher_id`	UUID	Primary key
`business_id`	UUID	Business ID
`voucher_type`	VARCHAR(50)	Voucher type
`voucher_number`	VARCHAR(50)	Voucher number
`voucher_date`	DATE	Voucher date
`reference_number`	VARCHAR(100)	Reference number
`description`	VARCHAR(500)	Description
`status`	VARCHAR(20)	Status (Draft, Posted)
`created_by_user_id`	UUID	Creator user ID
`posted_by_user_id`	UUID	Posted by user ID (nullable)
`created_at_utc`	TIMESTAMPTZ	Creation timestamp
`posted_at_utc`	TIMESTAMPTZ	Posted timestamp (nullable)

Relationships:

- Foreign key to `core.businesses` (business_id)
- Foreign key to `core.users` (created_by_user_id, posted_by_user_id)
- One-to-many with `acct.voucher_details`

6. `acct.voucher_details`

Stores voucher detail lines.

Column	Type	Description
`voucher_detail_id`	BIGSERIAL	Primary key
`voucher_id`	UUID	Voucher ID
`account_id`	UUID	Account ID
`debit_amount`	DECIMAL(18,2)	Debit amount
`credit_amount`	DECIMAL(18,2)	Credit amount
`description`	VARCHAR(500)	Line description

Relationships:

- Foreign key to `acct.vouchers` (voucher_id)
- Foreign key to `acct.chart_of_accounts` (account_id)

API Architecture

API Structure

The API follows RESTful conventions with versioning:

```
/api/v1/{module}/{resource}
```

Example Endpoints:

- `/api/v1/auth/login`
- `/api/v1/businesses/{id}/dashboard`
- `/api/v1/businesses/{businessId}/vouchers`
- `/api/ChartOfAccounts/business/{businessId}`

Controller Organization

Controllers are organized by module:

```
Controllers/
├── Core/
│   ├── AuthController.cs
│   ├── BusinessController.cs
│   ├── DashboardController.cs
│   ├── UsersController.cs
│   ├── RolesController.cs
│   ├── PermissionsController.cs
│   └── ...
├── Admin/
│   ├── AdminDashboardController.cs
│   └── UserController.cs
└── Accounts/
    ├── ChartOfAccountsController.cs
    ├── TransactionsController.cs
    ├── VouchersController.cs
    ├── LedgerController.cs
    └── ReportsController.cs
```

Dependency Injection

Services are registered in `Program.cs`:

```
// Repository Layer
builder.Services.AddScoped<IUserRepository, UserRepository>();
builder.Services.AddScoped<IChartOfAccountRepository, ChartOfAccountRepository>
();

// Business Logic Layer
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IChartOfAccountService, ChartOfAccountService>();

// Infrastructure
builder.Services.AddScoped<IDbConnectionFactory, NpgsqlConnectionFactory>();
builder.Services.AddScoped<ExceptionHandler>();
```

Authentication Flow

Registration Flow

```
1. Client → POST /api/v1/auth/register
   {
     "userHandle": "john_doe",
     "email": "john@example.com",
     "password": "SecurePass123!",
     "fullName": "John Doe",
     "panCardNumber": "ABCDE1234F"
   }

2. AuthService.RegisterAsync()
   - Validate input
   - Check duplicate user handle/email/PAN
   - Hash password (BCrypt)
   - Encrypt PAN card
   - Create user record
   - Generate JWT tokens

3. Response:
   {
     "accessToken": "eyJhbGciOiJIUzI1NiIs...",
     "refreshToken": "refresh_token_here",
     "expiresAt": "2025-01-20T10:00:00Z",
     "userId": "uuid",
     "userHandle": "john_doe"
   }
```

Login Flow

```
1. Client → POST /api/v1/auth/login
   {
     "loginIdentifier": "john@example.com",
     "password": "SecurePass123!"
   }

2. AuthService.LoginAsync()
   - Find user by email/user handle
   - Verify password (BCrypt)
   - Check if user is active
   - Generate JWT tokens
   - Store refresh token in database

3. Response:
   {
     "accessToken": "eyJhbGciOiJIUzI1NiIs...",
     "refreshToken": "refresh_token_here",
     "expiresAt": "2025-01-20T10:00:00Z"
   }
```

Token Refresh Flow

```
1. Client → POST /api/v1/auth/refresh
   {
     "refreshToken": "refresh_token_here"
   }
   OR
   Cookie: refreshToken=refresh_token_here

2. AuthService.RefreshAsync()
   - Validate refresh token
   - Check if token is revoked
   - Check if token is expired
   - Revoke old refresh token
   - Generate new access token
   - Generate new refresh token (rotation)

3. Response:
   {
     "accessToken": "new_access_token",
     "refreshToken": "new_refresh_token",
     "expiresAt": "2025-01-20T11:00:00Z"
   }
```

Protected Endpoint Access

```
1. Client → GET /api/v1/businesses/{id}/dashboard
   Header: Authorization: Bearer {accessToken}

2. JWT Middleware:
   - Validate token signature
   - Check expiration
   - Extract claims (userId, permissions)

3. Authorization:
   - Check user has access to business
   - Check user has required permissions
   - Execute controller action

4. Response:
   {
     "kpis": {...},
     "recentActivities": [...],
     "subscriptionStatus": {...}
   }
```

API Endpoints

Authentication Endpoints

Method	Endpoint	Description	Auth Required
POST	`/api/v1/auth/register`	Register new user	No
POST	`/api/v1/auth/login`	Login user	No
POST	`/api/v1/auth/refresh`	Refresh access token	No
POST	`/api/v1/auth/logout`	Logout user	Yes
POST	`/api/v1/auth/revoke`	Revoke refresh token	Yes
GET	`/api/v1/auth/users/me`	Get current user	Yes

Business Endpoints

Method	Endpoint	Description	Auth Required
GET	`/api/v1/businesses`	Get user's businesses	Yes
GET	`/api/v1/businesses/{id}`	Get business by ID	Yes
POST	`/api/v1/businesses`	Create business	Yes
PUT	`/api/v1/businesses/{id}`	Update business	Yes
DELETE	`/api/v1/businesses/{id}`	Delete business	Yes
GET	`/api/v1/businesses/{id}/dashboard`	Get dashboard data	Yes

User Management Endpoints

Method	Endpoint	Description	Auth Required
GET	`/api/v1/users`	Get all users	Yes (Admin)
GET	`/api/v1/users/{id}`	Get user by ID	Yes
PUT	`/api/v1/users/{id}`	Update user	Yes
POST	`/api/v1/users/{id}/businesses`	Add user to business	Yes
PUT	`/api/v1/users/{id}/default-business`	Set default business	Yes

Role & Permission Endpoints

Method	Endpoint	Description	Auth Required
GET	`/api/v1/roles`	Get all roles	Yes
POST	`/api/v1/roles`	Create role	Yes (Admin)
PUT	`/api/v1/roles/{id}`	Update role	Yes (Admin)
DELETE	`/api/v1/roles/{id}`	Delete role	Yes (Admin)
GET	`/api/v1/permissions`	Get all permissions	Yes
GET	`/api/v1/role-permissions/{roleId}`	Get role permissions	Yes
POST	`/api/v1/role-permissions`	Assign permissions to role	Yes (Admin)

Accounting Endpoints

Chart of Accounts

Method	Endpoint	Description	Auth Required
GET	`/api/ChartOfAccounts/business/{businessId}`	Get all accounts	Yes
GET	`/api/ChartOfAccounts/{id}`	Get account by ID	Yes
POST	`/api/ChartOfAccounts`	Create account	Yes
PUT	`/api/ChartOfAccounts/{id}`	Update account	Yes
DELETE	`/api/ChartOfAccounts/{id}`	Delete account	Yes

Transactions

Method	Endpoint	Description	Auth Required
GET	`/api/Transactions/business/{businessId}`	Get all transactions	Yes
GET	`/api/Transactions/{id}`	Get transaction by ID	Yes
POST	`/api/Transactions`	Create transaction	Yes
DELETE	`/api/Transactions/{id}`	Delete transaction	Yes

Vouchers

Method	Endpoint	Description	Auth Required
GET	`/api/v1/businesses/{businessId}/vouchers`	Get all vouchers	Yes
GET	`/api/v1/businesses/{businessId}/vouchers/{voucherId}`	Get voucher by ID	Yes
POST	`/api/v1/businesses/{businessId}/vouchers`	Create voucher	Yes
PUT	`/api/v1/businesses/{businessId}/vouchers/{voucherId}`	Update voucher	Yes
DELETE	`/api/v1/businesses/{businessId}/vouchers/{voucherId}`	Delete voucher	Yes
POST	`/api/v1/businesses/{businessId}/vouchers/{voucherId}/post`	Post voucher	Yes

Ledger

Method	Endpoint	Description	Auth Required
GET	`/api/Ledger/business/{businessId}/account/{accountId}`	Get ledger for account	Yes
GET	`/api/Ledger/business/{businessId}`	Get all ledgers	Yes

Data Flow Diagrams

User Registration Flow

```
┌─────────┐
│ Client  │
└─────────┘
     │ POST /api/v1/auth/register
     ▼
┌─────────────────┐
│ AuthController  │
└─────────────────┘
     │
     ▼
┌─────────────────┐
│  AuthService    │
│  RegisterAsync()│
└─────────────────┘
     │
     ├──► Validate Input
     ├──► Check Duplicates
     ├──► Hash Password
```

```
         ├─► Encrypt PAN
         │
         ▼
┌─────────────────┐
│ UserRepository  │
│  AddAsync()     │
└────────┬────────┘
         │
         ▼
┌─────────────────┐
│  PostgreSQL     │
│  core.users     │
└─────────────────┘
```

Voucher Posting Flow

```
┌─────────┐
│ Client  │
└────┬────┘
     │ POST /api/v1/businesses/{id}/vouchers/{id}/post
     ▼
┌─────────────────┐
│VouchersController│
└────────┬────────┘
         │
         ▼
┌─────────────────┐
│  VoucherService  │
│  PostVoucherAsync│
└────────┬────────┘
         │
         ├─► Validate Voucher
         ├─► Check Double Entry Balance
         ├─► Create Transaction Header
         ├─► Create Transaction Details
         │
         ▼
┌─────────────────┐
│TransactionRepo  │
│CreateTransaction│
└────────┬────────┘
         │
         ▼
┌─────────────────┐
│  PostgreSQL     │
│  acct.transaction_headers
│  acct.transaction_details
└─────────────────┘
```

Repository Pattern

Repository Interface

```
public interface IChartOfAccountRepository
{
    Task<IEnumerable<ChartOfAccount>> GetAllChartOfAccountsAsync(Guid
businessId);
    Task<ChartOfAccount> GetChartOfAccountByIdAsync(Guid id);
    Task<ChartOfAccount> AddAsync(ChartOfAccount account);
    Task<bool> UpdateAsync(ChartOfAccount account);
    Task<bool> DeleteAsync(Guid id);
}
```

Repository Implementation

```
public class ChartOfAccountRepository : IChartOfAccountRepository
{
    private readonly IDbConnectionFactory _connectionFactory;

    public ChartOfAccountRepository(IDbConnectionFactory connectionFactory)
    {
        _connectionFactory = connectionFactory;
    }

    public async Task<IEnumerable<ChartOfAccount>>
GetAllChartOfAccountsAsync(Guid businessId)
    {
        using var connection = _connectionFactory.CreateConnection();
        var query = @"
            SELECT account_id, business_id, account_type_id, parent_account_id,
                   account_code, account_name, description, is_active
            FROM acct.chart_of_accounts
            WHERE business_id = @businessId
            ORDER BY account_code";

        return await connection.QueryAsync<ChartOfAccount>(query, new {
businessId });
    }
}
```

Connection Factory

```
public interface IDbConnectionFactory
{
    IDbConnection CreateConnection();
}

public class NpgsqlConnectionFactory : IDbConnectionFactory
{
    private readonly string _connectionString;

    public NpgsqlConnectionFactory(IConfiguration configuration)
    {
        _connectionString =
configuration.GetConnectionString("DefaultConnection");
```

```
    }

    public IDbConnection CreateConnection()
    {
        return new NpgsqlConnection(_connectionString);
    }
}
```

Service Layer

Service Interface

```
public interface IChartOfAccountService
{
    Task<IEnumerable<ChartOfAccountDto>> GetAllChartOfAccountsAsync(Guid
businessId);
    Task<ChartOfAccountDto> GetChartOfAccountByIdAsync(Guid id);
    Task<ChartOfAccountDto> CreateChartOfAccountAsync(CreateChartOfAccountDto
dto);
    Task UpdateChartOfAccountAsync(Guid id, UpdateChartOfAccountDto dto);
    Task DeleteChartOfAccountAsync(Guid id);
}
```

Service Implementation

```
public class ChartOfAccountService : IChartOfAccountService
{
    private readonly IChartOfAccountRepository _repository;
    private readonly IMapper _mapper;

    public ChartOfAccountService(
        IChartOfAccountRepository repository,
        IMapper mapper)
    {
        _repository = repository;
        _mapper = mapper;
    }

    public async Task<ChartOfAccountDto>
CreateChartOfAccountAsync(CreateChartOfAccountDto dto)
    {
        // Business logic validation
        if (string.IsNullOrWhiteSpace(dto.AccountCode))
            throw new ArgumentException("Account code is required");

        // Map DTO to Entity
        var account = _mapper.Map<ChartOfAccount>(dto);
        account.AccountID = Guid.NewGuid();
        account.CreatedAtUTC = DateTime.UtcNow;

        // Save to repository
        var created = await _repository.AddAsync(account);
        // Map Entity to DTO
        return _mapper.Map<ChartOfAccountDto>(created);
    }
}
```

DTOs and Mapping

DTO Structure

```
// Request DTO
public class CreateChartOfAccountDto
{
    public Guid BusinessID { get; set; }
    public int AccountTypeID { get; set; }
    public Guid? ParentAccountID { get; set; }
    public string AccountCode { get; set; }
    public string AccountName { get; set; }
    public string Description { get; set; }
}

// Response DTO
public class ChartOfAccountDto
{
    public Guid AccountID { get; set; }
    public Guid BusinessID { get; set; }
    public int AccountTypeID { get; set; }
    public string AccountCode { get; set; }
    public string AccountName { get; set; }
    // ... other properties
}
```

AutoMapper Profile

```
public class ChartOfAccountMappingProfile : Profile
{
    public ChartOfAccountMappingProfile()
    {
        CreateMap<CreateChartOfAccountDto, ChartOfAccount>();
        CreateMap<UpdateChartOfAccountDto, ChartOfAccount>();
        CreateMap<ChartOfAccount, ChartOfAccountDto>();
    }
}
```

Error Handling

Exception Handler

```
public class ExceptionHandler
{
    private readonly ILogger<ExceptionHandler> _logger;

    public ExceptionHandler(ILogger<ExceptionHandler> logger)
    {
        _logger = logger;
    }

    public void LogError(Exception ex)
    {
        _logger.LogError(ex, "An error occurred: {Message}", ex.Message);
    }
}
```

Controller Error Handling

```
[HttpPost]
public async Task<IActionResult> Create(CreateChartOfAccountDto dto)
{
    try
    {
        var account = await
_chartOfAccountService.CreateChartOfAccountAsync(dto);
        return CreatedAtAction(nameof(GetById), new { id = account.AccountID },
account);
    }
    catch (ArgumentException ex)
    {
        return BadRequest(new { message = ex.Message });
    }
    catch (Exception ex)
    {
        _exceptionHandler.LogError(ex);
        return StatusCode(500, "An internal error occurred.");
    }
}
```

Database Migrations

Migration Files

Migration files are stored in `/API/DB-Backup/`:

- `Complete_Schema_PostgreSQL.sql` - Complete schema creation
- `0002_core_api_support.sql` - Core API support migration
- `0002_core_schema_improvements.sql` - Schema improvements

Running Migrations

```
# Connect to PostgreSQL
psql -h localhost -U your_username -d encryptzERPCore

# Run migration
\i /path/to/Complete_Schema_PostgreSQL.sql
```

Deployment Architecture

Production Setup

```
┌─────────────────┐
│   Nginx (443)   │  ← SSL Termination
└────────┬────────┘
         │ Proxy
┌────────▼────────┐
│  .NET API (5000)│
└────────┬────────┘
         │
┌────────▼────────┐
│  PostgreSQL     │
│  (5432)         │
└─────────────────┘
```

Configuration

Nginx Configuration:

- SSL certificate
- Reverse proxy to API
- Static file serving
- CORS headers

API Configuration:

- Connection string
- JWT settings
- CORS origins
- Logging configuration

Development Guidelines

Code Structure

1. **Controllers**: Handle HTTP requests/responses
2. **Services**: Business logic and validation
3. **Repositories**: Data access
4. **DTOs**: Data transfer objects
5. **Entities**: Database entities

Best Practices

1. Use dependency injection
2. Follow repository pattern
3. Use DTOs for API communication
4. Implement proper error handling
5. Log all errors
6. Validate input at service layer
7. Use transactions for multi-step operations
8. Follow RESTful conventions

Testing

Unit Tests

Located in `/API/Tests/`:

- `AuthTests/` - Authentication tests
- `BusinessLogic.Tests/` - Service layer tests

Integration Tests

- API endpoint tests
- Database integration tests
- Authentication flow tests

API Documentation

Swagger

Swagger UI is available at:

- Development: `https://localhost:7037/swagger`
- Production: `https://your-domain.com/swagger`

Postman Collection

Postman collection available at:

- `/postman/Auth.postman_collection.json`

Security Considerations

1. **Password Hashing**: BCrypt with salt
2. **PAN/Aadhar Encryption**: Encrypted at rest
3. **JWT Tokens**: Short-lived access tokens
4. **Refresh Tokens**: Rotated on each refresh
5. **HTTPS**: Enforced in production
6. **CORS**: Configured for specific origins
7. **SQL Injection**: Parameterized queries
8. **XSS**: Input validation and sanitization

Performance Optimization

1. **Database Indexes**: On frequently queried columns
2. **Connection Pooling**: Npgsql connection pooling
3. **Caching**: Consider Redis for frequently accessed data
4. **Pagination**: Implement for large datasets
5. **Async/Await**: All I/O operations are async

Troubleshooting

Common Issues

1. **Connection String**: Verify PostgreSQL connection
2. **JWT Secret**: Ensure secret key is configured
3. **CORS**: Check allowed origins
4. **Permissions**: Verify user has required permissions
5. **Database Schema**: Ensure migrations are applied

Last Updated: 23rd November 2025
Maintained By: Development Team