# Complete

# Table of Contents


1. Overview
2. Architecture
3. Environment Setup
4. Project Structure
5. Database Schema
6. Authentication System
7. API Endpoints
8. Middleware
9. Utilities
10. Socket.io Events
11. Scheduled Jobs
12. Error Handling
13. Development Guide
14. Deployment


---

# Overview

UserState API is a robust Node.js backend framework designed for managing user authentication, user profiles, payments, and referral systems. Built with Express.js, it provides a comprehensive set of features for modern web applications.

## Key Capabilities

- Multi-strategy authentication (Local, JWT, Basic)
- User registration with OTP/Email verification
- Payment processing via Xendit

- Real-time payment status updates
- Referral and point system
- Address management
- Marketing offers and voucher redemption
- Push notifications via Firebase Cloud Messaging

---

# Architecture

## Technology Stack

| Component | Technology |
|---|---|
| Runtime | Node.js |
| Web Framework | Express.js |
| Primary Database | PostgreSQL (via Sequelize ORM) |
| Secondary Database | MongoDB (via Mongoose) |
| Cache/Session Store | Redis |
| Authentication | Passport.js |
| Validation | AJV, Express Validator |
| Security | Helmet, bcrypt, argon2 |
| Real-time | Socket.io |
| Payment Gateway | Xendit |
| Email Service | Nodemailer |
| Push Notifications | Firebase Admin SDK |
| Task Scheduling | node-cron |

## Application Flow

```
Request → Express Middleware → Passport Auth → Route Handler → Controller → Model → Database
                                                                    ↓
                                                            Response/Error Handler
```

---

# Environment Setup

## Environment Variables

Create a `.env` file in the root directory with the following variables:

```
# Server Configuration
API_PORT=33010
ORIGIN_CORS=http://localhost:3000,http://example.com

# PostgreSQL Configuration
PS_HOST=localhost
PS_USER=postgres
PS_PASSWORD=your_password
PS_DB=userstate
PS_DIALECT=postgres
PS_PORT=5432
PS_POOL_MAX=50

# MongoDB Configuration
MONGO_DB=mongodb://localhost:27017/userstate

# Redis Configuration
REDIS_HOST=localhost
REDIS_PORT=6379
REDIS_PASSWORD=your_redis_password

# Authentication
SECRET_AUTH_CONFIG=your_jwt_secret_key
SANDBOX_AUTH_CONFIG=your_sandbox_key

# Xendit Configuration
XENDIT_SECRET_KEY=your_xendit_secret_key
XENDIT_PUBLIC_KEY=your_xendit_public_key
XENDIT_CALLBACK_TOKEN=your_callback_token

# External APIs
```

```
API_URL_SUNSHINE=https://sunshine-api-url
API_URL_SMSVIRO=https://smsviro-api-url
SMSVIROTOKEN=your_sms_token

# Email Configuration (Nodemailer)
EMAIL_HOST=smtp.hostinger.com
EMAIL_PORT=465
EMAIL_USER=noreply@yourdomain.com
EMAIL_PASS=your_email_password

# Firebase Configuration
# Place your Firebase service account JSON in keystore/onmapss-main-key.json
```

# Installation Steps

1. **Install Node.js Dependencies**

   ```
   npm install
   ```

2. **Setup PostgreSQL Database**

   ```
   createdb userstate
   ```

3. **Setup MongoDB** (if using payment logs)

   ```
   # Ensure MongoDB is running
   mongod --dbpath /path/to/data
   ```

4. **Setup Redis**

   ```
   # Ensure Redis is running
   redis-server
   ```

5. **Configure Firebase**
   - Download your Firebase service account key JSON
   - Place it in `keystore/onmapss-main-key.json`
6. **Run Database Migrations**
   - The application uses auto-sync for Sequelize models
   - Uncomment the sync lines in `index.js` if needed:

   ```
   await db.user_login.sync({ alter: true });
   await db.user_info.sync({ alter: true });
   // ... etc
   ```

# Project Structure

## Directory Overview

```
userstate-api/
│
├── config/                 # Configuration files
│   ├── api.js              # External API configurations
│   ├── auth.config.js      # JWT and auth secrets
│   ├── db.js               # Database connection settings
│   ├── multer.js           # File upload configuration
│   ├── pg.js               # PostgreSQL initialization
│   ├── redis.js            # Redis client setup
│   └── voucher-ext.json    # Voucher extensions data
│
├── controllers/            # Business logic handlers
│   ├── address.controller.js      # Address CRUD operations
│   ├── auth.controller.js         # Authentication logic
│   ├── faq.controller.js          # FAQ management
│   ├── fcm.controller.js          # Firebase Cloud Messaging
│   ├── marketing_offer.controller.js  # Marketing offers
│   ├── passport.controller.js     # Passport strategy handlers
│   ├── payment.controller.js      # Payment processing
│   ├── payout.controller.js       # Payout operations
│   ├── point_mutation.controller.js   # Points system
│   ├── referral.controller.js     # Referral system
│   └── user_info.controller.js    # User profile management
│
├── middleware/             # Custom middleware
│   ├── auth.js             # Authentication middleware
│   ├── authJwt.js          # JWT verification
│   ├── error_param_handler.js # Error handling
│   ├── express_validator.js  # Express validator wrapper
│   ├── json_validator.js     # AJV JSON schema validator
│   ├── other_validator.js    # Custom validators
│   ├── passport.js           # Passport strategies
```

```
│   ├── upload_update_profile.js # File upload middleware
│   └── verify.js              # Token verification
│
├── models/                    # Database models
│   ├── index.js               # Model loader and relationships
│   ├── user_login.model.js    # User credentials
│   ├── user_info.model.js     # User profile information
│   ├── user_balance.model.js # User wallet balance
│   ├── address_list.model.js # User addresses
│   ├── point_mutation.model.js # Point transactions
│   ├── referral_history.model.js # Referral tracking
│   ├── marketing_offer.model.js # Offers
│   ├── fcm.model.js           # FCM tokens
│   ├── api_key.model.js       # API keys for services
│   ├── otp_tracker.model.js  # OTP tracking
│   ├── bank_account.model.js # Bank account info
│   ├── bank_channel.model.js # Payment channels
│   ├── payment_type.model.js # Payment methods
│   ├── user_transaction.model.js # Transaction history
│   ├── voucher_redemption.model.js # Voucher usage
│   ├── deactivate_reason.model.js # Account deactivation
│   ├── xendit_payout_logs.model.js # Payout logs (MongoDB)
│   └── xendit_checkout_logs.model.js # Checkout logs (MongoDB)
│
├── routers/                   # Route definitions
│   ├── address.route.js       # Address endpoints
│   ├── auth.route.js          # Authentication endpoints
│   ├── faq.route.js           # FAQ endpoints
│   ├── fcm.route.js           # FCM endpoints
│   ├── marketing_offer.route.js # Marketing endpoints
│   ├── payment.route.js       # Payment endpoints
│   ├── point_mutation.route.js # Points endpoints
│   ├── referral.route.js      # Referral endpoints
│   └── user_info.route.js     # User profile endpoints
│
├── schemas/                   # Request validation schemas
│   ├── index.js               # Schema exporter
│   ├── address.schema.js      # Address validation
│   ├── auth.schema.js         # Auth validation
│   ├── fcm.schema.js          # FCM validation
```

```
│   ├── marketing_offer.schema.js # Marketing validation
│   ├── payment.schema.js     # Payment validation
│   ├── point_mutation.schema.js # Points validation
│   ├── referral.schema.js    # Referral validation
│   └── user_info.schema.js   # User profile validation
│
├── socket/                   # Socket.io implementations
│   └── payment_status.socket.js # Real-time payment updates
│
├── scheduler/                # Cron job definitions
│   └── xendit.scheduler.js   # Payment status sync
│
├── utilities/                # Utility functions
│   ├── encrypt.util.js       # Encryption helpers
│   ├── general.util.js       # General utilities
│   ├── hashing_center.util.js # Hashing functions
│   └── qrcode.util.js        # QR code generation
│
├── libraries/                # Static data files
│   ├── dir_day.json          # Day mappings
│   ├── dir_digits.json       # Digit mappings
│   ├── dir_month.json        # Month mappings
│   └── dir_year.json         # Year mappings
│
├── keystore/                 # Private keys and certificates
│   └── onmapss-main-key.json # Firebase service account key
│
├── resources/                # Generated files (gitignored)
│   └── user_profile_img/     # User profile images
│
├── index.js                  # Application entry point
├── utility.js                # Shared utility functions
├── package.json              # Dependencies and scripts
└── .gitignore                # Git ignore rules
```

---

# Database Schema

# PostgreSQL Tables (via Sequelize)

## 1. user_login

Stores user authentication credentials.

| Column | Type | Description |
|---|---|---|
| username | STRING (PK) | Unique username |
| pwd | STRING | Hashed password |
| disabled | BOOLEAN | Account status |
| created_at | DATE | Registration date |
| updated_at | DATE | Last update |

## 2. user_info

Stores user profile information.

| Column | Type | Description |
|---|---|---|
| id | INTEGER (PK, Auto) | Primary key |
| username | STRING (FK) | Reference to user_login |
| full_name | STRING | User's full name |
| email | STRING (Unique) | Email address |
| phone_number | STRING | Phone number |
| profile_img | STRING | Profile image path |
| agent | BOOLEAN | Agent status |
| referral_code | STRING | Unique referral code |
| created_at | DATE | Creation date |
| updated_at | DATE | Last update |

## 3. user_balance

Tracks user wallet balance.

| Column | Type | Description |
|---|---|---|
| username | STRING (PK, FK) | Reference to user_login |
| balance | DECIMAL | Current balance |

| Column | Type | Description |
|---|---|---|
| points | INTEGER | Reward points |
| created_at | DATE | Creation date |
| updated_at | DATE | Last update |

# 4. address_list

Stores user addresses.

| Column | Type | Description |
|---|---|---|
| id | INTEGER (PK, Auto) | Primary key |
| username | STRING (FK) | Reference to user_login |
| label | STRING | Address label |
| recipient_name | STRING | Recipient name |
| phone_number | STRING | Contact number |
| address | TEXT | Full address |
| is_default | BOOLEAN | Default address flag |
| created_at | DATE | Creation date |
| updated_at | DATE | Last update |

# 5. point_mutation

Tracks point transactions.

| Column | Type | Description |
|---|---|---|
| id | INTEGER (PK, Auto) | Primary key |
| username | STRING (FK) | Reference to user_login |
| type | STRING | CREDIT/DEBIT |
| amount | INTEGER | Point amount |
| description | STRING | Transaction description |
| reference_id | STRING | External reference |
| created_at | DATE | Transaction date |

# 6. referral_history

Tracks referral relationships.

| Column | Type | Description |
|---|---|---|
| id | INTEGER (PK, Auto) | Primary key |
| username | STRING (FK) | Referred user |
| submitted_by | STRING (FK) | Referrer username |
| referral_code | STRING | Used referral code |
| status | STRING | Referral status |
| created_at | DATE | Creation date |

# 7. marketing_offer

Stores promotional offers.

| Column | Type | Description |
|---|---|---|
| id | INTEGER (PK, Auto) | Primary key |
| username | STRING (FK) | Target user (nullable) |
| title | STRING | Offer title |
| description | TEXT | Offer details |
| image_url | STRING | Offer image |
| valid_until | DATE | Expiration date |
| is_active | BOOLEAN | Active status |
| created_at | DATE | Creation date |

# 8. fcm

Stores Firebase Cloud Messaging tokens.

| Column | Type | Description |
|---|---|---|
| id | INTEGER (PK, Auto) | Primary key |
| username | STRING (FK) | Reference to user_login |
| token | TEXT | FCM device token |
| device_type | STRING | iOS/Android |
| created_at | DATE | Registration date |

# 9. user_transaction

Tracks payment transactions.

| Column | Type | Description |
|---|---|---|
| id | STRING (PK) | Transaction ID |
| username | STRING (FK) | Reference to user_login |
| type | STRING | Transaction type |
| amount | DECIMAL | Transaction amount |
| status | STRING | Transaction status |
| payment_method | STRING | Payment channel |
| reference_id | STRING | External reference |
| created_at | DATE | Transaction date |

# 10. voucher_redemption

Tracks voucher usage.

| Column | Type | Description |
|---|---|---|
| id | INTEGER (PK, Auto) | Primary key |
| username | STRING (FK) | Reference to user_login |
| voucher_code | STRING | Voucher code |
| discount_amount | DECIMAL | Discount value |
| redeemed_at | DATE | Redemption date |

# 11. api_key

Stores API keys for service authentication.

| Column | Type | Description |
|---|---|---|
| holder | STRING (PK) | Service identifier |
| key | STRING | Hashed API key |
| description | STRING | Key description |
| created_at | DATE | Creation date |

# 12. otp_tracker

Tracks OTP requests for rate limiting.

| Column | Type | Description |
|---|---|---|
| id | INTEGER (PK, Auto) | Primary key |

| Column | Type | Description |
| --- | --- | --- |
| identifier | STRING | Email/Phone |
| otp_type | STRING | OTP purpose |
| count | INTEGER | Request count |
| last_request | DATE | Last request time |

# MongoDB Collections (via Mongoose)

## xendit_payout_logs

Stores Xendit payout transaction logs.

```
{
  payout_id: String,
  username: String,
  amount: Number,
  status: String,
  bank_code: String,
  account_number: String,
  account_holder_name: String,
  description: String,
  reference_id: String,
  created_at: Date,
  updated_at: Date
}
```

## xendit_checkout_logs

Stores Xendit checkout/invoice logs.

```
{
  invoice_id: String,
  external_id: String,
  username: String,
  amount: Number,
  status: String,
  payment_method: String,
  payment_channel: String,
```

```
  description: String,
  invoice_url: String,
  expiry_date: Date,
  created_at: Date,
  updated_at: Date
}
```

# Model Relationships

```
user_login
  ├── hasOne → user_info
  ├── hasOne → user_balance
  ├── hasMany → address_list
  ├── hasMany → marketing_offer
  ├── hasMany → fcm
  ├── hasMany → user_transaction
  └── hasMany → voucher_redemption

user_info
  ├── belongsTo → user_login
  ├── belongsTo → user_balance
  └── hasMany → referral_history (as referrer)

referral_history
  ├── belongsTo → user_info (username)
  └── belongsTo → user_info (submitted_by as referrer)
```

---

# Authentication System

The application uses **Passport.js** with three authentication strategies:

# 1. Local Strategy (Username/Password)

**Used for**: User login with username/email and password

**Endpoint**: `POST /auth`

**Flow**:

1. User submits credentials
2. System checks database for username or email
3. If user not confirmed, checks Redis for temporary data
4. Password is verified using bcrypt
5. JWT token generated on success

**Middleware**: `passport.PassportLocal`

# 2. JWT Strategy (Bearer Token)

**Used for**: Protected route access

**Endpoints**: Most authenticated routes

**Flow**:

1. Client sends JWT in Authorization header: `Bearer <token>`
2. Token is decoded and validated
3. User's password hash is used as secret key (dynamic per user)
4. User information extracted from token payload

**Middleware**: `passport.PassportJwt`

**Token Structure**:

```
{
  user: "username",
  agent: true/false,
  iat: 1234567890,
  exp: 1234567890
}
```

# 3. Basic Strategy (API Key)

**Used for**: Service-to-service authentication

**Endpoints**: Internal/system endpoints (`/internal/*`, `/sys/*`)

**Flow**:

1. Client sends Basic Auth header: `Basic base64(holder:apiKey)`

2. System looks up API key in database
3. API key verified using bcrypt

**Middleware**: `passport.PassportBasic`

# Token Management

**JWT Secret Key**: Dynamically generated from user's password hash

- Formula: `pwd.slice(32, 48) + pwd.slice(14, 30)`
- Ensures tokens invalidate on password change

**Token Expiration**: Configurable via environment

- Session tokens can be extended via `/extend` endpoint

**Redis Storage**: Used for temporary user data during registration

---

# API Endpoints

# Authentication Endpoints

## POST /auth

**Description**: User login
**Auth**: Local Strategy
**Body**:

```
{
  "username": "john_doe",
  "password": "securePassword123"
}
```

**Response**:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "agent": false
}
```

# POST /signup

**Description**: User registration with email/OTP verification
**Auth**: None
**Body**:

```
{
  "username": "john_doe",
  "password": "securePassword123",
  "email": "john@example.com",
  "full_name": "John Doe",
  "phone_number": "628123456789",
  "referral_code": "REF123",
  "mode": "LINK"
}
```

**Modes**:

- `LINK`: Email verification link
- `OTP`: SMS OTP verification

# POST /check/email

**Description**: Check if email is already registered
**Auth**: None
**Body**:

```
{
  "email": "john@example.com"
}
```

# POST /verify-sign-up

**Description**: Verify OTP or email verification
**Auth**: JWT Token
**Body**:

```
{
  "otp": "123456"
}
```

# POST /request-reset-pwd

**Description**: Request password reset
**Auth**: None
**Body**:

```
{
  "email": "john@example.com"
}
```

## POST /reset-pwd

**Description**: Reset password with token
**Auth**: Token from email
**Body**:

```
{
  "password": "newSecurePassword123"
}
```

## POST /update-pwd

**Description**: Update password (logged in)
**Auth**: JWT
**Body**:

```
{
  "old_password": "oldPassword123",
  "new_password": "newPassword123"
}
```

## GET /extend

**Description**: Extend JWT token expiration
**Auth**: JWT
**Response**:

```
{
  "token": "new_jwt_token..."
}
```

## GET /verify

**Description**: Verify if JWT token is valid
**Auth**: JWT
**Response**: `200 OK` or `401 Unauthorized`

## GET /logoff

**Description**: Logout user
**Auth**: JWT
**Response**: `200 OK`

## POST /deactivate

**Description**: Deactivate user account
**Auth**: JWT
**Body**:

```
{
  "reason": "No longer needed"
}
```

# User Profile Endpoints

## GET /user/profile

**Description**: Get user profile information
**Auth**: JWT
**Response**:

```
{
  "username": "john_doe",
  "full_name": "John Doe",
  "email": "john@example.com",
  "phone_number": "628123456789",
  "profile_img": "/resources/user_profile_img/john_doe.jpg",
  "referral_code": "JOHN123",
  "balance": 50000,
  "points": 150
}
```

## PUT /user/profile

**Description**: Update user profile
**Auth**: JWT
**Body**:

```
{
  "full_name": "John Updated",
  "phone_number": "628987654321"
}
```

## POST /user/update-profile-img

**Description**: Upload profile image
**Auth**: JWT
**Content-Type**: `multipart/form-data`
**Body**: Form data with `profile_img` file

## GET /user/referral-code

**Description**: Get user's referral code
**Auth**: JWT
**Response**:

```
{
  "referral_code": "JOHN123"
}
```

# Address Endpoints

## GET /address/list

**Description**: Get all user addresses
**Auth**: JWT
**Response**:

```
[
  {
    "id": 1,
    "label": "Home",
    "recipient_name": "John Doe",
    "phone_number": "628123456789",
    "address": "Jl. Example No. 123",
```

```
    "is_default": true
  }
]
```


## POST /address

**Description**: Create new address
**Auth**: JWT
**Body**:


```
{
  "label": "Office",
  "recipient_name": "John Doe",
  "phone_number": "628123456789",
  "address": "Jl. Office No. 456",
  "is_default": false
}
```


## PUT /address/:id

**Description**: Update address
**Auth**: JWT
**Body**: Same as create

## DELETE /address/:id

**Description**: Delete address
**Auth**: JWT

## POST /address/set-default

**Description**: Set default address
**Auth**: JWT
**Body**:


```
{
  "id": 1
}
```


# Payment Endpoints

# POST /payment-checkout

**Description**: Create payment checkout/invoice
**Auth**: JWT
**Body**:

```
{
  "amount": 100000,
  "description": "Order #12345",
  "payment_method": "EWALLET",
  "ewallet_type": "OVO"
}
```

# POST /sys/payment-checkout

**Description**: System checkout (for internal services)
**Auth**: Basic
**Body**:

```
{
  "username": "john_doe",
  "amount": 100000,
  "description": "Order #12345"
}
```

# POST /payment-callback

**Description**: Xendit payment callback
**Auth**: Callback Token
**Body**: Xendit callback payload

# GET /balance

**Description**: Get user balance
**Auth**: JWT
**Response**:

```
{
  "balance": 50000,
  "points": 150
}
```

## POST /balance/decrease

**Description**: Decrease user balance
**Auth**: JWT
**Body**:

```
{
  "amount": 10000,
  "description": "Purchase payment"
}
```

## GET /invoice/detail

**Description**: Get invoice details
**Auth**: None (query param based)
**Query**: `?invoice_id=inv_123`

## GET /invoice/detail/xendit

**Description**: Get Xendit invoice details
**Auth**: JWT
**Query**: `?invoice_id=inv_123`

## GET /invoice/list/xendit

**Description**: List user's Xendit invoices
**Auth**: JWT

## POST /invoice/cancel/xendit

**Description**: Cancel Xendit invoice
**Auth**: JWT
**Body**:

```
{
  "invoice_id": "inv_123"
}
```

## POST /redeem-voucher

**Description**: Redeem voucher code
**Auth**: Basic
**Body**:

```
{
  "username": "john_doe",
  "voucher_code": "PROMO123"
}
```

## GET /check-voucher-redemption

**Description**: Check if user has redeemed voucher
**Auth**: JWT
**Query**: `?voucher_code=PROMO123`

## POST /bank/list

**Description**: Get list of available bank codes
**Auth**: None
**Body**:

```
{
  "type": "DISBURSEMENT"
}
```

## GET /payment/list

**Description**: Get available payment types
**Auth**: None

# Point & Referral Endpoints

## GET /points/history

**Description**: Get point transaction history
**Auth**: JWT
**Response**:

```
[
  {
    "id": 1,
    "type": "CREDIT",
    "amount": 50,
    "description": "Referral bonus",
    "created_at": "2024-01-01T00:00:00.000Z"
```

```
  }
]
```

## POST /points/redeem

**Description**: Redeem points for balance
**Auth**: JWT
**Body**:

```
{
  "points": 100
}
```

## GET /referral/history

**Description**: Get referral history
**Auth**: JWT
**Response**:

```
[
  {
    "username": "referred_user",
    "referral_code": "JOHN123",
    "status": "completed",
    "created_at": "2024-01-01T00:00:00.000Z"
  }
]
```

## GET /referral/stats

**Description**: Get referral statistics
**Auth**: JWT
**Response**:

```
{
  "total_referrals": 10,
  "completed_referrals": 8,
  "pending_referrals": 2,
  "total_earned_points": 400
}
```

# FCM Endpoints

## POST /fcm/register

**Description**: Register FCM device token
**Auth**: JWT
**Body**:

```
{
  "token": "fcm_device_token...",
  "device_type": "android"
}
```

## DELETE /fcm/token

**Description**: Remove FCM token
**Auth**: JWT
**Query**: `?token=fcm_device_token...`

## POST /fcm/send

**Description**: Send push notification (admin)
**Auth**: Basic
**Body**:

```
{
  "username": "john_doe",
  "title": "New Offer",
  "body": "Check out our latest promotion!",
  "data": {
    "offer_id": "123"
  }
}
```

# Marketing Offer Endpoints

## GET /marketing-offer/list

**Description**: Get available marketing offers
**Auth**: JWT

**Response**:

```
[
  {
    "id": 1,
    "title": "Summer Sale",
    "description": "Get 50% off!",
    "image_url": "/images/offer1.jpg",
    "valid_until": "2024-12-31T23:59:59.000Z",
    "is_active": true
  }
]
```

## POST /marketing-offer

**Description**: Create marketing offer (admin)
**Auth**: Basic
**Body**:

```
{
  "title": "New Year Sale",
  "description": "Special discount",
  "image_url": "/images/offer.jpg",
  "valid_until": "2024-12-31"
}
```

---

# Middleware

# Authentication Middleware

## passport.PassportLocal

- Validates username and password
- Returns JWT token on success
- Used in: `/auth`

## passport.PassportJwt

- Validates Bearer token
- Extracts user from token
- Dynamic secret key per user
- Used in: Most protected routes

### passport.PassportBasic

- Validates HTTP Basic Auth
- Checks API key against database
- Used in: `/internal/*`, `/sys/*` routes

### authJwt.verifyToken

- Custom JWT verification
- Used for specific token validation scenarios
- Checks token from Authorization header or query params

## Validation Middleware

### validator.validate({ body: schema })

- Uses AJV to validate request body against JSON schema
- Automatically returns 400 with validation errors
- Schemas defined in `schemas/` directory

### contentTypeValid(type)

- Ensures request Content-Type matches expected type
- Example: `contentTypeValid("application/json")`

### verifyToken(type)

- Validates special tokens (REGISTER, RESET_PWD)
- Checks token from query params or body
- Validates against Redis storage

### verifyCallbackToken

- Validates Xendit callback tokens
- Ensures callbacks are from legitimate source

## Error Handling Middleware

### errorHandlerParam.JsonHandler

- Catches JSON parsing errors
- Returns proper error response

### errorHandlerParam.otherHandler

- Global error handler
- Formats error responses
- Logs errors for debugging

## Upload Middleware

### upload_update_profile

- Handles profile image uploads
- Uses multer for multipart/form-data
- Validates file type and size
- Stores in `/resources/user_profile_img/`

---

# Utilities

## utility.js

### provideKey(req, rawJwtToken, done)

- Provides secret key for JWT verification
- Extracts key from user's password hash
- Ensures tokens invalidate on password change

### admin()

- Initializes Firebase Admin SDK
- Returns admin instance for FCM

### sendMail(to, subject, text, html)

- Sends email via Nodemailer
- Uses configured SMTP settings

- Supports HTML templates

### generateRandomString(length, numberOnly)

- Generates random strings or numbers
- Used for OTP generation

### syncInitBalance()

- Initializes user balance records
- Utility function for data migration

## Encryption Utilities (utilities/encrypt.util.js)

- Encryption and decryption helpers
- Used for sensitive data

## Hashing Utilities (utilities/hashing_center.util.js)

- Centralized hashing functions
- bcrypt and argon2 wrappers

## QR Code Utilities (utilities/qrcode.util.js)

- QR code generation
- Used for payment or referral codes

## General Utilities (utilities/general.util.js)

- Common helper functions
- Data formatting and transformation

---

# Socket.io Events

# Payment Status Socket

**Namespace**: Default namespace
**File**: `socket/payment_status.socket.js`

## Server Events

### payment:status:update

**Description**: Broadcast payment status update to specific user
**Payload**:

```
{
  invoice_id: "inv_123",
  status: "PAID",
  amount: 100000,
  paid_at: "2024-01-01T00:00:00.000Z"
}
```

## Client Events

### subscribe:payment

**Description**: Subscribe to payment updates for user
**Payload**:

```
{
  username: "john_doe"
}
```

### unsubscribe:payment

**Description**: Unsubscribe from payment updates
**Payload**:

```
{
  username: "john_doe"
}
```

# Usage Example

**Client-side (JavaScript)**:

```
const socket = io('http://localhost:33010');

// Subscribe to payment updates
socket.emit('subscribe:payment', { username: 'john_doe' });

// Listen for payment status updates
socket.on('payment:status:update', (data) => {
  console.log('Payment updated:', data);
  // Update UI accordingly
});

// Unsubscribe when done
socket.emit('unsubscribe:payment', { username: 'john_doe' });
```

---

# Scheduled Jobs

## Xendit Payment Status Sync

**File**: `scheduler/xendit.scheduler.js`
**Schedule**: Every 2 hours (`0 */2 * * *`)
**Function**: `expireStatusUpdater`

**Purpose**:

- Synchronizes payment status with Xendit
- Updates expired invoices
- Notifies users of status changes via Socket.io

**Manual Trigger**: Uncomment endpoint in `index.js`

```
app.get("/scheduler", xenditScheduler.expireStatusUpdater);
```

**Implementation Details**:

1. Fetches pending/processing invoices from database
2. Queries Xendit API for current status
3. Updates local database with latest status

4. Emits Socket.io events for status changes
5. Logs results to MongoDB

---

# Error Handling

## Error Response Format

All errors follow a consistent format:

```
{
  "status": false,
  "message": "Error description",
  "errorCode": 400,
  "errors": [] // Optional validation errors
}
```

## HTTP Status Codes

| Code | Description | Usage |
|---|---|---|
| 200 | OK | Successful request |
| 201 | Created | Resource created successfully |
| 400 | Bad Request | Invalid request data |
| 401 | Unauthorized | Missing or invalid authentication |
| 403 | Forbidden | Authenticated but not permitted |
| 404 | Not Found | Resource not found |
| 409 | Conflict | Duplicate resource (e.g., email exists) |
| 422 | Unprocessable Entity | Invalid credentials |
| 500 | Internal Server Error | Server error |

## Common Error Messages

### Authentication Errors

- `"Invalid Credential"` (422): Wrong username/password
- `"Account does not exist"` (500): User not found
- `"Account is Disabled"` (403): Account deactivated
- `"OTP Not Confirmed"` (403): Registration pending OTP
- `"Account is Not Confirmed"` (403): Registration pending email

## Validation Errors

- `"Email is already registered"` (409): Duplicate email
- `"PhoneRequired"`: Phone number required for OTP mode
- `"PhoneFormatErr"`: Invalid phone number format

## Payment Errors

- `"Insufficient Balance"`: Not enough funds
- `"Invoice Not Found"`: Invalid invoice ID
- `"Invoice Already Paid"`: Cannot modify paid invoice

# Error Handler Middleware

**JsonHandler**: Catches JSON parsing errors

```
app.use(errorHandlerParam.JsonHandler);
```

**otherHandler**: Global error handler

```
app.use(errorHandlerParam.otherHandler);
```

# Logging

The application uses **morgan** with custom colored format for request logging:

- Method: Blue
- URL: Green
- Status: Yellow
- Response Time: Cyan

Errors are logged to console with stack traces in development.

---

# Development Guide

# Getting Started

1. **Clone and Install**

   ```
   git clone <repo-url>
   cd userstate-api
   npm install
   ```

2. **Configure Environment**

   ```
   cp .env.example .env
   # Edit .env with your configuration
   ```

3. **Start Development Server**

   ```
   npm start
   ```

   Uses nodemon for auto-reload on file changes.

# Project Conventions

## Code Style

- Use ES6+ features
- Async/await preferred over callbacks
- Use destructuring for cleaner code
- Follow existing file naming patterns

## File Naming

- Models: `*.model.js`
- Controllers: `*.controller.js`
- Routes: `*.route.js`
- Schemas: `*.schema.js`
- Middleware: descriptive names with context

## Database Conventions

- Use Sequelize for PostgreSQL operations
- Use Mongoose for MongoDB operations

- Always use transactions for critical operations
- Define model relationships in `models/index.js`

## Security Best Practices

- Always hash passwords with bcrypt/argon2
- Validate all user input
- Use parameterized queries (ORM handles this)
- Sanitize file uploads
- Implement rate limiting for sensitive endpoints
- Never log sensitive data (passwords, tokens, keys)

# Adding New Features

## 1. Adding a New Endpoint

**Step 1**: Create Controller (`controllers/feature.controller.js`)

```
const db = require("../models");


exports.myEndpoint = async (req, res) => {
  try {
    // Your logic here
    res.status(200).json({ success: true });
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

**Step 2**: Create Schema (`schemas/feature.schema.js`)

```
exports.mySchema = {
  type: "object",
  properties: {
    field1: { type: "string" },
    field2: { type: "number" }
  },
  required: ["field1"]
};
```

**Step 3**: Create Route (`routers/feature.route.js`)

```
const controller = require("../controllers/feature.controller");
const passport = require("../controllers/passport.controller");
const { validator, feature } = require("../schemas");

module.exports = (app) => {
  app.post(
    "/feature/endpoint",
    [
      passport.PassportJwt,
      validator.validate({ body: feature.mySchema })
    ],
    controller.myEndpoint
  );
};
```

**Step 4**: Load Route in `index.js`

```
require("./routers/feature.route")(app);
```

# 2. Adding a New Model

**Step 1**: Create Model File (`models/my_table.model.js`)

```
module.exports = (sequelize, DataTypes) => {
  const MyTable = sequelize.define("my_table", {
    id: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    field1: {
      type: DataTypes.STRING,
      allowNull: false
    },
    field2: {
      type: DataTypes.TEXT,
      allowNull: true
    }
  }, {
    timestamps: true,
```

```
    underscored: true
  });

  return MyTable;
};
```

**Step 2**: Load Model in `models/index.js`

```
db.my_table = require("./my_table.model")(
  sequalize_config,
  sequelize.DataTypes
);
```

**Step 3**: Define Relationships (if any)

```
db.user_login.hasMany(db.my_table, {
  foreignKey: "username",
  sourceKey: "username"
});
```

# Testing

## Manual Testing

Use tools like:

- **Postman**: For API testing
- **curl**: For command-line testing
- **Socket.io Client**: For WebSocket testing

## Example curl Commands

**Login**:

```
curl -X POST http://localhost:33010/auth \
  -H "Content-Type: application/json" \
  -d '{"username":"test","password":"password123"}'
```

**Get Profile** (with JWT):

```
curl -X GET http://localhost:33010/user/profile \
  -H "Authorization: Bearer YOUR_JWT_TOKEN"
```

**Basic Auth** (Internal endpoints):

```
curl -X POST http://localhost:33010/sys/payment-checkout \
  -H "Content-Type: application/json" \
  -H "Authorization: Basic $(echo -n 'holder:apikey' | base64)" \
  -d '{"username":"test","amount":10000}'
```

# Debugging

## Enable Sequelize Query Logging

In `config/db.js`:

```
const sequalize_config = new sequelize(
  userstate.DB,
  userstate.USER,
  userstate.PASSWORD,
  {
    host: userstate.HOST,
    dialect: userstate.DIALECT,
    logging: console.log, // Add this line
    // ...
  }
);
```

## Database Connection Issues

Check:

1. PostgreSQL is running: `pg_isready`
2. Credentials in `.env` are correct
3. Database exists: `psql -l`
4. Firewall allows connection

## Redis Connection Issues

Check:

1. Redis is running: `redis-cli ping`
2. Redis configuration in `config/redis.js`
3. Password authentication if enabled

## Common Issues

**"Cannot find module"**: Run `npm install`

**"Port already in use"**: Change `API_PORT` in `.env` or kill process:

```
lsof -ti:33010 | xargs kill -9
```

**"Connection refused"**: Database or Redis not running

---

# Deployment

# Production Checklist

- ☐ Set `NODE_ENV=production` in environment
- ☐ Use strong, unique values for `SECRET_AUTH_CONFIG`
- ☐ Configure production database credentials
- ☐ Set up proper CORS origins
- ☐ Enable HTTPS/SSL
- ☐ Configure firewall rules
- ☐ Set up database backups
- ☐ Configure monitoring and logging
- ☐ Set up PM2 or similar process manager
- ☐ Remove or protect debug endpoints

# Using PM2

**Install PM2**:

```
npm install -g pm2
```

**Start Application**:

```
pm2 start index.js --name userstate-api
```

**PM2 Configuration** (`ecosystem.config.js`):

```
module.exports = {
  apps: [{
    name: 'userstate-api',
    script: './index.js',
    instances: 2,
    exec_mode: 'cluster',
    env: {
      NODE_ENV: 'production',
    },
    error_file: './logs/err.log',
    out_file: './logs/out.log',
    log_date_format: 'YYYY-MM-DD HH:mm:ss Z'
  }]
};
```

**Start with config**:

```
pm2 start ecosystem.config.js
```

**Useful PM2 Commands**:

```
pm2 list              # List all processes
pm2 logs userstate-api # View logs
pm2 restart userstate-api # Restart app
pm2 stop userstate-api    # Stop app
pm2 delete userstate-api  # Remove from PM2
pm2 monit             # Monitor resources
```

# Nginx Reverse Proxy

**Example Nginx Configuration**:

```
server {
    listen 80;
    server_name api.yourdomain.com;
```

```
    location / {
        proxy_pass http://localhost:33010;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;
    }

    # WebSocket support for Socket.io
    location /socket.io/ {
        proxy_pass http://localhost:33010;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
    }
}
```

# Docker Deployment

**Dockerfile**:

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./
RUN npm ci --only=production

COPY . .

EXPOSE 33010

CMD ["node", "index.js"]
```

**docker-compose.yml**:

```
version: '3.8'

services:
  api:
    build: .
    ports:
      - "33010:33010"
    environment:
      - NODE_ENV=production
    env_file:
      - .env
    depends_on:
      - postgres
      - redis
      - mongo
    restart: unless-stopped

  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: userstate
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: your_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    restart: unless-stopped

  redis:
    image: redis:7-alpine
    command: redis-server --requirepass your_redis_password
    volumes:
      - redis_data:/data
    restart: unless-stopped

  mongo:
    image: mongo:6
    volumes:
      - mongo_data:/data/db
```

```
    restart: unless-stopped

volumes:
  postgres_data:
  redis_data:
  mongo_data:
```

# Environment Variables for Production

```
# Production Settings
NODE_ENV=production
API_PORT=33010

# Use production database
PS_HOST=your-prod-db-host
PS_USER=prod_user
PS_PASSWORD=strong_password
PS_DB=userstate_prod
PS_DIALECT=postgres
PS_PORT=5432
PS_POOL_MAX=50

# Strong JWT secret
SECRET_AUTH_CONFIG=long_random_string_min_32_chars

# Production CORS
ORIGIN_CORS=https://app.yourdomain.com,https://www.yourdomain.com

# External services
XENDIT_SECRET_KEY=xnd_production_...
```

# Monitoring

**Health Check Endpoint**: Add to `index.js`:

```
app.get("/health", (req, res) => {
  res.json({
    status: "OK",
```

```
    timestamp: new Date().toISOString(),
    uptime: process.uptime()
  });
});
```

**PM2 Monitoring**:

```
pm2 install pm2-logrotate  # Log rotation
pm2 set pm2-logrotate:max_size 10M
pm2 set pm2-logrotate:retain 7
```

# Backup Strategy

**Database Backup** (PostgreSQL):

```
# Daily backup script
pg_dump -U postgres -h localhost userstate_prod > backup_$(date +%Y%m%d).sql

# Automated daily backup with cron
0 2 * * * /path/to/backup_script.sh
```

**Redis Backup**: Redis automatically saves RDB snapshots. Configure in `redis.conf`:

```
save 900 1
save 300 10
save 60 10000
```

---

# Conclusion

This documentation provides a comprehensive guide to the UserState API. For questions or issues, please contact the development team or create an issue in the repository.

# Quick Links

- **Repository**: [Your repository URL]
- **API Base URL**: `http://localhost:33010` (development)
- **Socket.io**: Same as API base URL

# Support

For support and questions:

- Email: support@yourdomain.com
- Documentation updates: Create a pull request
- Bug reports: Open an issue

---

**Last Updated**: February 2024
**Version**: 1.0.0
**Author**: Development Team

---

Revision #1
Created 10 February 2026 03:27:03 by ondeliveloper
Updated 10 February 2026 03:35:39 by ondeliveloper