Server API Documentation

Project Overview

This is a Node.js Express server application that provides user authentication and management services for a mobile/desktop application. The system uses device-based authentication with sold token validation for user registration. Key features include:

- Device-based user authentication using JWT tokens
- Sold token validation system for new user registration
- User profile management with CRUD operations
- Account deletion and restoration functionality
- Contract management with expiration dates
- SSL/HTTPS support with Let's Encrypt certificates

The application appears to be designed for an educational platform ("madaure-education.com") that manages user accounts through device identification and purchased access tokens.

System Architecture

Technology Stack

- **Runtime**: Node.js
- **Framework**: Express.js
- **Database**: MongoDB with Mongoose ODM
- **Authentication**: JWT (JSON Web Tokens)
- **Security**: SHA-256 hashing with salt for device verification
- **SSL/TLS**: HTTPS with Let's Encrypt certificates
- **Body Parsing**: body-parser for URL-encoded data

Security Model

- Device-based authentication using hashed device IDs
- JWT tokens with configurable expiration (5-7 days)
- Sold token validation to prevent unauthorized registrations
- Salted SHA-256 hashing for device ID verification

Database Design

The system uses MongoDB with the following collections:

- Users (active user accounts)
- ArchivedUsers (deleted accounts within 30-day recovery period)
- Contracts (user access contracts with expiration dates)
- SoldTokens (purchasable access tokens)
- AuthTokens (JWT token management)

Project Structure

```
project/
├── server.js                # Main server entry point
├── config/
│   └── db.js               # Database connection configuration
├── controllers/
│   └── appController.js    # Main application logic
├── models/
│   ├── User.js             # User data model
│   ├── ArchivedUser.js     # Archived user model
│   ├── Contract.js         # User contract model
│   ├── SoldToken.js        # Sold token model
│   └── AuthToken.js        # Authentication token model
├── routes/
│   └── appRoutes.js        # API route definitions
├── services/
│   └── keyService.js       # Token validation and utility services
├── middlewares/
│   └── auth.js             # Authentication middleware
└── SSL certificates/
    ├── fullchain.pem       # SSL certificate chain
    └── privkey.pem         # SSL private key
```

Protocol Specification

Authentication Flow

1. **Device Verification**: Client sends `deviceId` and `hDeviceId` (SHA-256 hash)
2. **Token Generation**: Server validates device and returns JWT auth token
3. **API Access**: Client includes auth token in subsequent requests
4. **Token Refresh**: Client can request new auth tokens using device verification

Device ID Hashing

javascript

```javascript
// Client-side hashing (for reference)
const salt = "30BDR3rErEalO1D5";
const hash = crypto.createHash("sha256");
hash.update(deviceId + salt);
const hashedDeviceId = hash.digest("hex");
```

User Registration Process

1. Client provides `deviceId` and `soldToken`
2. Server validates sold token availability and authenticity
3. Server creates user account and contract record
4. Server marks sold token as consumed
5. Server returns user information string

Data Format Standards

- **User Info String**: Pipe-delimited format `firstName#lastName#birthDate#placeOfBirth#email#phone#school#address#userType`
- **Date Format**: Handled by `keyService.formatDate()` utility
- **User Types**: Numeric (1 for 10-char tokens, 2 for 8-char tokens)

API Reference

Authentication Endpoints

Check Registration Status

http

```http
GET /isRegisteredBefore/:deviceId/:hDeviceId
```

Parameters:

- `deviceId` (string): Unique device identifier
- `hDeviceId` (string): SHA-256 hash of deviceId with salt

Responses:

- `{ status: "ACCESS DENIED" }` - Invalid device hash
- `{ status: "NEW USER", authToken: "..." }` - Device not registered
- `{ status: "DELETED", authToken: "..." }` - Account was deleted

- `{ status: "REGISTERED", authToken: "...", userInfo: "..." }` - Active account

Generate New Auth Token

```http
GET /sendNewAuthToken/:deviceId/:hDeviceId
```

Parameters:

- `deviceId` (string): Unique device identifier
- `hDeviceId` (string): SHA-256 hash of deviceId with salt

Response:

```json
{
  "authToken": "jwt_token_string"
}
```

Validate Auth Token

```http
POST /validateAuthToken
Content-Type: application/x-www-form-urlencoded
```

Body:

- `deviceId` (string): Device identifier
- `authToken` (string): JWT token to validate

Responses:

- `{ status: "VALID", userInfo: "..." }` - Token is valid
- `{ status: "NOT VALID" }` - Invalid token
- `{ status: "USER NOT FOUND" }` - User doesn't exist

User Management Endpoints

Register New User

```http
POST /registerNewUser
Content-Type: application/x-www-form-urlencoded
Authorization: Required (auth middleware)
```

Body:

- `deviceId` (string): Device identifier
- `soldToken` (string): Purchased access token

Responses:

- `{ status: "CREATED", userInfo: "..." }` - Account created successfully
- `{ status: "INVALID SOLD TOKEN" }` - Token invalid or consumed
- `{ status: "ALREADY HAVE AN ACCOUNT" }` - Token used by different device
- `{ status: "EXPIRED CONTRACT" }` - Associated contract expired

Save User Information

```http
POST /saveUserInformation
Content-Type: application/x-www-form-urlencoded
Authorization: Required (auth middleware)
```

Body:

- `deviceId` (string): Device identifier
- `userInfo` (string): Pipe-delimited user data

Response:

```json
{
  "status": "SAVED",
  "userInfo": "updated_user_info_string"
}
```

Delete Account

```http
POST /deleteAccount
Content-Type: application/x-www-form-urlencoded
Authorization: Required (auth middleware)
```

Body:

- `deviceId` (string): Device identifier

Response:

```json
{
  "status": "DELETED"
}
```

Restore Account

```http
POST /restoreAccount
Content-Type: application/x-www-form-urlencoded
Authorization: Required (auth middleware)
```

Body:

- `deviceId` (string): Device identifier

Responses:

- `{ status: "RESTORED" }` - Account restored successfully
- `{ status: "EXPIRED" }` - Account archived over 30 days ago

Data Models

User Model

javascript

```
{
  deviceId: String,          // Unique device identifier
  firstName: String,         // User's first name
  lastName: String,          // User's last name
  birthDate: String,         // Birth date (formatted string)
  placeOfBirth: String,      // Place of birth
  email: String,             // Email address
  phone: String,             // Phone number
  school: String,            // School name
  address: String,           // Address/commune
  userType: Number           // User type (1 or 2)
}
```

ArchivedUser Model

javascript

```
{
  // All User model fields plus:
  archivedAt: Date         // Timestamp when user was archived
}
```

Contract Model

javascript

```
{
  deviceId: String,        // Associated device ID
  soldToken: String,       // Associated sold token
  startDate: Date,         // Contract start date
  expiringDate: Date       // Contract expiration date (1 year from start)
}
```

SoldToken Model

javascript

```
{
  key: String,             // Token key/code
  consumed: Boolean        // Whether token has been used
}
```

Response Data Formats

User Info String Format

The user information is returned as a pipe-delimited string:

```
firstName#lastName#birthDate#placeOfBirth#email#phone#school#address#userType
```

Authentication Token

JWT tokens contain:

```javascript
{
  deviceId: "device_identifier",
  exp: timestamp,           // Expiration time
  iat: timestamp            // Issued at time
}
```

Error Handling

All endpoints return appropriate HTTP status codes:

- `200`: Success
- `400`: Bad request (invalid input)
- `401`: Unauthorized (invalid token)
- `500`: Internal server error

Error responses follow the format:

```json
{
  "error": "error_message_string"
}
```