Authentication API Documentation

Overview

The backend uses Laravel Sanctum for authentication with UUID tokens. After login, the client receives a token that must be included in all protected API requests.

Base URL

```
http://localhost:8001/api
```

Authentication Endpoints

1. Register (Create Account)

Endpoint: `POST /auth/register`

Description: Create a new user account

Request:

```
curl -X POST "http://localhost:8001/api/auth/register" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "John Doe",
    "email": "john@example.com",
    "password": "password123",
    "password_confirmation": "password123"
  }'
```

Request Body:

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "password123",
  "password_confirmation": "password123"
}
```

Validation Rules:

- `name` - Required, string, max 255 characters
- `email` - Required, email format, unique in database
- `password` - Required, min 8 characters, must match password_confirmation

Success Response (200 OK):

```
{
  "message": "User registered successfully",
  "data": {
    "id": "019b39d5-ff0e-7288-9503-208e353dfda3",
    "name": "John Doe",
    "email": "john@example.com",
    "role": "user",
    "is_active": true,
    "created_at": "2025-12-20T12:34:56.000000Z",
    "updated_at": "2025-12-20T12:34:56.000000Z"
  },
  "token": "019b39d5-ff0e-7288-9503-
208e353dfda3|gM5xK2pL9qR8wE1tY3uI4oP5sA6dF7gH8jK9lM0nB1cV2xZ3y",
  "token_type": "Bearer"
}
```

Error Response (422 Unprocessable Entity):

```
{
  "message": "The email has already been taken.",
  "errors": {
    "email": [
      "The email has already been taken."
    ]
  }
}
```

PROF

---

2. Login (Authenticate User)

Endpoint: `POST /auth/login`

Description: Authenticate with email and password, receive token

Request:

```
curl -X POST "http://localhost:8001/api/auth/login" \
  -H "Content-Type: application/json" \
  -d '{
    "email": "john@example.com",
```

```
    "password": "password123"
  }'
```

Request Body:

```
{
  "email": "john@example.com",
  "password": "password123"
}
```

Success Response (200 OK):

```
{
  "message": "Login successful",
  "data": {
    "id": "019b39d5-ff0e-7288-9503-208e353dfda3",
    "name": "John Doe",
    "email": "john@example.com",
    "role": "user",
    "is_active": true,
    "created_at": "2025-12-20T12:34:56.000000Z",
    "updated_at": "2025-12-20T12:34:56.000000Z"
  },
  "token": "019b39d5-ff0e-7288-9503-
208e353dfda3|gM5xK2pL9qR8wE1tY3uI4oP5sA6dF7gH8jK9lM0nB1cV2xZ3y",
  "token_type": "Bearer"
}
```

Error Response (401 Unauthorized):

PROF

```
{
  "message": "Invalid credentials"
}
```

---

3. Get Current User Info

Endpoint: `GET /auth/me`

Authentication: ✅ Required

Description: Get details of currently authenticated user

Request:

```
curl -X GET "http://localhost:8001/api/auth/me" \
  -H "Authorization: Bearer 019b39d5-ff0e-7288-9503-
208e353dfda3|gM5xK2pL9qR8wE1tY3uI4oP5sA6dF7gH8jK9lM0nB1cV2xZ3y"
```

Success Response (200 OK):

```
{
  "data": {
    "id": "019b39d5-ff0e-7288-9503-208e353dfda3",
    "name": "John Doe",
    "email": "john@example.com",
    "role": "user",
    "is_active": true,
    "created_at": "2025-12-20T12:34:56.000000Z",
    "updated_at": "2025-12-20T12:34:56.000000Z"
  }
}
```

Error Response (401 Unauthorized):

```
{
  "message": "Unauthenticated"
}
```

---

4. Refresh Token

Endpoint: `POST /auth/refresh`

Authentication: ✅ Required

Description: Generate a new token (extends session)

Request:

```
curl -X POST "http://localhost:8001/api/auth/refresh" \
  -H "Authorization: Bearer 019b39d5-ff0e-7288-9503-
208e353dfda3|gM5xK2pL9qR8wE1tY3uI4oP5sA6dF7gH8jK9lM0nB1cV2xZ3y"
```

Success Response (200 OK):

```
{
  "message": "Token refreshed successfully",
```

```
  "token": "019b39d5-ff0e-7288-9503-
208e353dfda3|nE7qW2pL9qR8wE1tY3uI4oP5sA6dF7gH8jK9lM0nB1cV2xZ3y",
  "token_type": "Bearer"
}
```

---

5. Logout

Endpoint: `POST /auth/logout`

Authentication: ✅ Required

Description: Invalidate current token and logout

Request:

```
curl -X POST "http://localhost:8001/api/auth/logout" \
  -H "Authorization: Bearer 019b39d5-ff0e-7288-9503-
208e353dfda3|gM5xK2pL9qR8wE1tY3uI4oP5sA6dF7gH8jK9lM0nB1cV2xZ3y"
```

Success Response (200 OK):

```
{
  "message": "Logged out successfully"
}
```

Token Format

Token Structure:

PROF

```
{user_id}|{random_base64_string}
```

Example:

```
019b39d5-ff0e-7288-9503-
208e353dfda3|gM5xK2pL9qR8wE1tY3uI4oP5sA6dF7gH8jK9lM0nB1cV2xZ3y
```

- First part: User UUID
- Second part: Random token string (68 characters)

Using Tokens in Requests

Header Format

```
Authorization: Bearer {token}
```

Example Authenticated Request

```
curl -X GET "http://localhost:8001/api/stations" \
  -H "Authorization: Bearer 019b39d5-ff0e-7288-9503-
208e353dfda3|gM5xK2pL9qR8wE1tY3uI4oP5sA6dF7gH8jK9lM0nB1cV2xZ3y"
```

User Roles

Users can have different roles that determine what they can access:

Role	Description	Can Manage	Can View
`user`	Regular user	Nothing special	Public endpoints
`admin`	Administrator	All stations, managers, users	Everything
`station_manager`	Station manager	Only assigned station	Their assigned station

---

Common HTTP Status Codes

Status	Meaning	Example
200	Success	Login successful
201	Created	User registered
400	Bad Request	Missing required fields
401	Unauthorized	Invalid token or not authenticated
403	Forbidden	No permission for this action
404	Not Found	Resource doesn't exist
422	Validation Error	Email already exists
500	Server Error	Database error

PROF

Frontend Integration Examples

JavaScript/TypeScript (Fetch API)

Register:

```
async function register(name, email, password) {
  const response = await
fetch('http://localhost:8001/api/auth/register', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      name,
      email,
      password,
      password_confirmation: password
    })
  });

  const data = await response.json();
  if (response.ok) {
    localStorage.setItem('token', data.token);
    return data.data; // User object
  } else {
    throw new Error(data.message);
  }
}
```

Login:

```
async function login(email, password) {
  const response = await fetch('http://localhost:8001/api/auth/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ email, password })
  });

  const data = await response.json();
  if (response.ok) {
    localStorage.setItem('token', data.token);
    return data.data; // User object
  } else {
    throw new Error(data.message);
  }
}
```

PROF

Get Current User:

```
async function getCurrentUser(token) {
  const response = await fetch('http://localhost:8001/api/auth/me', {
    headers: { 'Authorization': `Bearer ${token}` }
  });
```

```
  if (!response.ok) throw new Error('Unauthorized');
  return (await response.json()).data;
}
```

Authenticated Request:

```
async function fetchStations(token) {
  const response = await fetch('http://localhost:8001/api/stations', {
    headers: { 'Authorization': `Bearer ${token}` }
  });

  return response.json();
}
```

Logout:

```
async function logout(token) {
  await fetch('http://localhost:8001/api/auth/logout', {
    method: 'POST',
    headers: { 'Authorization': `Bearer ${token}` }
  });

  localStorage.removeItem('token');
}
```

React Hook Pattern

PROF

```
// useAuth.js
import { useState, useEffect } from 'react';

export function useAuth() {
  const [user, setUser] = useState(null);
  const [token, setToken] = useState(localStorage.getItem('token'));
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const login = async (email, password) => {
    setLoading(true);
    try {
      const res = await fetch('http://localhost:8001/api/auth/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, password })
      });
      const data = await res.json();
```

```
      if (!res.ok) throw new Error(data.message);

      setToken(data.token);
      setUser(data.data);
      localStorage.setItem('token', data.token);
      return data.data;
    } catch (err) {
      setError(err.message);
      throw err;
    } finally {
      setLoading(false);
    }
  };

  const logout = async () => {
    setLoading(true);
    try {
      await fetch('http://localhost:8001/api/auth/logout', {
        method: 'POST',
        headers: { 'Authorization': `Bearer ${token}` }
      });
      setUser(null);
      setToken(null);
      localStorage.removeItem('token');
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    if (token && !user) {
      fetch('http://localhost:8001/api/auth/me', {
        headers: { 'Authorization': `Bearer ${token}` }
      })
      .then(res => res.json())
      .then(data => setUser(data.data))
      .catch(() => setToken(null));
    }
  }, [token]);

  return { user, token, login, logout, loading, error };
}
```

PROF

Axios Configuration

```
// Create axios instance with token
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:8001/api'
});

// Attach token to all requests
export function setAuthToken(token) {
  if (token) {
    api.defaults.headers.common['Authorization'] = `Bearer ${token}`;
  } else {
    delete api.defaults.headers.common['Authorization'];
  }
}

// Usage
export async function login(email, password) {
  const { data } = await api.post('/auth/login', { email, password });
  setAuthToken(data.token);
  localStorage.setItem('token', data.token);
  return data;
}

export default api;
```

Testing Credentials

Use these credentials to test authentication:

Admin Account:

PROF

```
Email: admin@example.com
Password: password
```

Manager Account:

```
Email: infin@lpgtamale.com
Password: password
```

Security Notes

1. ✅ **Always use HTTPS in production** (not HTTP)
2. ✅ **Store tokens securely** - Use HttpOnly cookies or secure localStorage

3. ✅ **Include token in all authenticated requests** - Via Authorization header
4. ✅ **Handle token expiration** - Implement token refresh mechanism
5. ✅ **Clear token on logout** - Remove from storage immediately
6. ✅ **Validate user on app load** - Call `/auth/me` to verify token validity

---

Common Issues & Solutions

Issue: "Unauthenticated" error

- **Solution:** Check token is included in Authorization header
- **Format:** `Authorization: Bearer {token}` (note the space)

Issue: "Invalid credentials" on login

- **Solution:** Verify email and password are correct
- **Check:** User account exists and is not inactive

Issue: Token keeps expiring

- **Solution:** Implement token refresh before expiration
- **Call:** `POST /auth/refresh` regularly or on 401 errors

Issue: CORS errors

- **Solution:** Backend is configured to allow requests from localhost
- **In production:** Update CORS settings in `.env`

Next Steps

1. ✅ Implement login/register form in Next.js
2. ✅ Store token in localStorage/cookies
3. ✅ Add Authorization header to all API requests
4. ✅ Handle 401 responses with automatic logout
5. ✅ Redirect to login if not authenticated
6. ✅ Display user info in navbar/header

PROF

API Rate Limiting

Currently no rate limiting is enforced. In production, consider:

- Max 5 login attempts per IP per 15 minutes
- Max 20 API requests per minute per token
- Implement exponential backoff for retries

Support

For issues or questions:

1. Check error response message
2. Verify token format (user_id|random_string)
3. Ensure all required fields are provided
4. Check that user is not inactive