GenCharge

Authentication & Authorisation Documentation

Project: GenCharge (Mobile Recharge System)
Version: 1.0.0
Last Updated: 09-Nov-2025

Overview

This document explains how authentication and role-based access control (RBAC) are implemented in the GenCharge backend system.

It covers the user login flow (OTP-based), token management, and the permissions assigned to each user role. The system defines two primary roles: **User** (the standard customer) and **Admin** (the system administrator).

Authentication Flow

The system uses a passwordless, OTP-based flow for user authentication.

Step	Description
1. **Send OTP**	User enters their mobile number at POST /auth/login. The system generates an OTP, sends it via SMS, and (in development) returns it in the API response for testing.
2. **Login**	User submits their mobile number (as username) and the received OTP (as password) to POST /auth/lverify-otp-login using application/x-www-form-urlencoded format.
3. **Get Tokens**	The system validates the OTP. On success, it issues a short-lived **Access Token** and a long-lived **Refresh Token**.
4. **Access API**	The client sends the **Access Token** in the Authorization: Bearer <token> header for all protected API requests.

5. **Refresh Token**	When the Access Token expires (server returns 401), the client sends the **Refresh Token** (as the Bearer token) to POST /auth/refresh to get a new Access Token without a full login.
6. **Logout**	DELETE /auth/logout revokes the user's current token. logs the user out from all devices.

JWT Token Example (Payload)

This is an example of the data contained within a JWT Access Token. The sub (subject) is the user's mobile number, and role defines their permissions.

```
{
  "user": "9876543210",
  "role": "user",
  "jwt": 1731180000,
  "exp": 1731180900
}
```

Authentication Endpoints (Tag: "Auth")

Method	Endpoint	Auth Required	Description
POST	/auth/login	No	Send an OTP to a user's mobile number for verification.
POST	/auth/verify-otp-login	No	Authenticate with mobile and OTP to get access/refresh tokens.
POST	/auth/refresh	Yes	Generate a new access token using a valid refresh

			token.
DELETE	/auth/logout	Yes	Revoke the current user's token.
DELETE	/users/delete-account	Yes	Permanently delete the authenticated user's account.

Role-Based Access Control (RBAC)

Role	Description	Example Permissions
Admin	System administrator with full access.	Create/Update/Delete Plans, Plan types, Offers, Offer types, and Content. Can view all user profiles and bookings. Manages system backups and user roles.
User	Standard customer account.	Read-only access to Movies, Shows, Offers. Can manage their own profile (/profile/me), and create/read/cancel their own bookings and payments.

Role Permissions Matrix

Resource	Action	Admin	User
Plans	Create / Update / Delete	☑	✗
Plan	Read	☑	☑
Offers	Create / Update / Delete	☑	✗
Offer	Read	☑	☑
Recharge	Read (All)	☑	✗
Profile	Create / Read (Self)	☑	☑
Edit profile	Update(Self)	☑	☑
Offer type	REad	☑	☑
Offer type	Create, update, delete	☑	✗
Transactions	Export/ (all)	☑	✗
Transactions	Read (All)	☑	✗
transactions	Read / (Self)	☑	☑
Content (Banners, etc.)	All Actions	☑	✗
Backups	All Actions	☑	✗
Roles	All Actions	☑	✗

Middleware & Token Validation

All protected routes (Auth Required: Yes) include a FastAPI dependency that automatically validates the Authorization: Bearer token.

This dependency performs:

1. JWT decoding and signature validation.
2. Token expiration check (exp claim).
3. Token revocation check (e.g., against a blacklist in DB).
4. Role authorization check (e.g., checking if role == "admin" for Admin-only routes).

Example (simplified):

```
# auth_dependencies.py
from fastapi import Depends, HTTPException
from auth.security import oauth2_scheme, decode_access_token

async def get_current_user(token: str = Depends(oauth2_scheme)):
    payload = decode_access_token(token)
    if not payload:
        raise HTTPException(status_code=401, detail="Invalid token")

    # Logic to get user from DB using payload["user"]
    user = ...
    return user

async def get_admin_user(current_user: TokenData = Depends(get_current_user)):
    # Assumes the TokenData object has a role attribute
    if current_user.role != "admin":
        raise HTTPException(status_code=403, detail="Permission denied")
    return current_user
```

Usage Example:

```
# movies_router.py
@router.post("/", dependencies=[Depends(get_admin_user)])
async def create_movie(...):
    # This code only runs if the user is an admin
    ...

@router.get("/profile/me")
async def get_my_profile(current_user: User = Depends(get_current_user)):
    # This code runs for any authenticated user
    ...
```

Frontend Integration Notes

- **Token Storage:** Store the **Access Token** in application memory. Store the **Refresh Token** in a secure, HttpOnly cookie for maximum security.
- **API Requests:** Attach the Access Token to all protected API calls: Authorization: Bearer <ACCESS_TOKEN>
- **401 Handling:** Intercept all 401 (Unauthorized) responses. When one is received:
  1. Automatically (and silently) call POST /auth/refresh (the browser will auto-send the HttpOnly refresh token cookie).
  2. If successful, save the new Access Token in memory and re-try the original failed API request.
  3. If refresh fails (e.g., refresh token also expired), log the user out and redirect to the login page.
- **Logout:** On logout, explicitly call DELETE /auth/logout to revoke the token on the backend, then clear all tokens/user data from the client.

Token Expiry Policy

Token Type	Lifetime	Storage (Recommended)	Rotation
Access Token	15 Minutes	In-memory (client-side)	Auto-rotated via refresh flow
Refresh Token	30 Days	HttpOnly Cookie	Revoked on logout or expiry

Security Considerations

- **HTTPS:** Always use HTTPS in production to encrypt all traffic.
- **Secret Key:** Use a strong, complex, and randomly generated secret key for signing JWTs.
- **Token Revocation:** Maintain a token "blacklist" (e.g., in DB) to instantly revoke tokens on logout or password change.

End of Auth & Role Documentation