Instagram OAuth (Basic Display) for Social Casino — Setup & Code

This guide wires **“Sign in with Instagram”** (Basic Display) into your existing FastAPI + MySQL project, designed for a **social casino** (no real money). It:

- Keeps your current DB schema.
- Links Instagram accounts to your players.
- Creates local sessions and USD/VND wallets on first signup.
- Uses **ngrok** in development.

---

0) Summary of the Flow

1) User clicks **Continue with Instagram** → `/oauth/instagram/start`. 2) Instagram login & consent (scope: `user_profile`). 3) Instagram redirects to `/oauth/instagram/callback?code=...`. 4) We exchange the `code` for a short-lived token, fetch `{id, username}` from `graph.instagram.com/me`. 5) If the IG id is already linked → we create a local **session token** and log the user in. 6) If not linked → we show a **Finish Signup** page that collects email/password, creates the player, makes **USD & VND wallets**, links IG, and starts a session.

> We keep Instagram strictly as an **identity provider**; your system owns sessions.

---

1) Facebook Developer Setup (Instagram Basic Display)

> Do this with a personal Facebook account that will own the app.

1. **Create a Facebook App**
2. Go to developers.facebook.com → *My Apps* → **Create App**.
3. Choose **Consumer** app type, name it (e.g., `IGW Social Casino`), and finish creation.
4. **Add Product: Instagram Basic Display**
5. In your App dashboard: *Add products to your app* → **Instagram Basic Display** → **Set Up**.
6. **Configure Instagram Basic Display**
7. **Valid OAuth Redirect URIs** → add your dev URL:
   `https://<your-ngrok>.ngrok.io/oauth/instagram/callback`
   (Later add production domain.)
8. **Deauthorize Callback URL** → temporary placeholder:
   `https://<your-ngrok>.ngrok.io/fb/deauth`
9. **Data Deletion Request URL** → temporary placeholder:
   `https://<your-ngrok>.ngrok.io/fb/delete`

10. Save changes.

11. **Create Instagram Test Users**

12. *Roles* → **Instagram Testers** → **Add Instagram Testers** → enter an IG account → send invite.
13. The invited IG account must accept the invite in the Instagram app (Settings → Website permissions → Apps & Websites → Tester Invites).

14. **Get Credentials**

15. *Instagram Basic Display* → **Basic Display** → **App ID** and **App Secret**.

    While in development, only **testers** can log in. For production, Meta review may be restrictive for RMG—fine for a **social casino**, but keep Instagram optional.

---

2) Local Dev Setup (ngrok + .env)

1) Start your API on **8001**:

```
uvicorn app.main:app --reload --port 8001
```

2) Expose it via **ngrok**:

```
ngrok http 8001
```

Copy the **https** forwarding URL for use below.

3) Create/update your `.env` (in `igw/`):

```
APP_ENV=dev

DB_URL=mysql+pymysql://root:YOURPASS@127.0.0.1:3306/apidb
REDIS_URL=redis://127.0.0.1:6379/0

JWT_SIGNING_KEY=change_me
PASSWORD_BCRYPT_ROUNDS=12

DEFAULT_WALLET_CURRENCIES=USD,VND
DEFAULT_WALLET_TYPE=CASH

# Instagram Basic Display
IG_CLIENT_ID=xxxxxxxxxxxxxxxx
IG_CLIENT_SECRET=xxxxxxxxxxxxxxxx
IG_REDIRECT_URI=https://<your-ngrok>.ngrok.io/oauth/instagram/callback
```

```
# Optional SQL echo for debugging (0/1)
SQL_ECHO=0
```

Restart Uvicorn after editing.

---

3) Database Changes

Add a mapping table for external IDs (future-proof: you may link other providers later):

```
CREATE TABLE IF NOT EXISTS player_external_ids (
  id INT PRIMARY KEY AUTO_INCREMENT,
  userId INT NOT NULL,
  provider VARCHAR(50) NOT NULL,         -- 'instagram'
  external_id VARCHAR(255) NOT NULL,     -- IG numeric id
  username VARCHAR(255) NULL,            -- IG handle
  verified TINYINT(1) NOT NULL DEFAULT 1,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE KEY u_provider_ext (provider, external_id),
  INDEX idx_user (userId),
  CONSTRAINT fk_player_ext_ids_user FOREIGN KEY (userId) REFERENCES
players(userId)
);
```

> Your existing `players` table already has `ext_user_id` and `user_name`; we'll keep mapping cleanly in `player_external_ids` and optionally copy IG id into `players.ext_user_id`.

---

4) Code Changes

A) `app/config.py` — add Instagram settings & SQL echo

```
from pydantic_settings import BaseSettings, SettingsConfigDict

class Settings(BaseSettings):
    app_env: str = "dev"

    db_url: str
    redis_url: str | None = None

    jwt_signing_key: str
    password_bcrypt_rounds: int = 12

    default_wallet_currencies: str = "USD,VND"
    default_wallet_type: str = "CASH"
    # Instagram Basic Display
    ig_client_id: str | None = None
    ig_client_secret: str | None = None
    ig_redirect_uri: str | None = None

    # Debug
    sql_echo: int = 0

    model_config = SettingsConfigDict(env_file=".env",
env_file_encoding="utf-8", case_sensitive=False)

settings = Settings()
```

B) `app/db.py` — enable optional SQL echo

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, DeclarativeBase
from .config import settings

class Base(DeclarativeBase):
    pass

engine = create_engine(
    settings.db_url,
    pool_pre_ping=True,
    pool_recycle=3600,
    future=True,
    echo=bool(settings.sql_echo),
)
SessionLocal = sessionmaker(bind=engine, autoflush=False, autocommit=False,
future=True)

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

C) `app/models.py` — align to your column names and add `PlayerExternalId`

```
from sqlalchemy.orm import Mapped, mapped_column, relationship
from sqlalchemy import String, Integer, Date, DateTime, Numeric, ForeignKey,
CHAR, UniqueConstraint
from .db import Base

class Player(Base):
    __tablename__ = "players"
    user_id: Mapped[int] = mapped_column("userId", Integer, primary_key=True,
autoincrement=True)
    user_name: Mapped[str | None] = mapped_column("user_name", String(255))
    ext_user_id: Mapped[str | None] = mapped_column("ext_user_id",
String(1000))
    first_name: Mapped[str | None] = mapped_column(String(100))
    last_name: Mapped[str | None] = mapped_column(String(100))
    email: Mapped[str] = mapped_column(String(255), unique=True,
nullable=False)
    password_hash: Mapped[str] = mapped_column(String(255), nullable=False)
    language_code: Mapped[str] = mapped_column(CHAR(2), nullable=False)
    registration_date: Mapped[DateTime | None] = mapped_column(DateTime)
    status: Mapped[str | None] = mapped_column(String(20), default="active")
    date_of_birth: Mapped[Date | None] = mapped_column(Date)
    phone_number: Mapped[str | None] = mapped_column(String(20))
    country: Mapped[str | None] = mapped_column(String(100))

class Wallet(Base):
    __tablename__ = "wallets"

    wallet_id: Mapped[int] = mapped_column(Integer, primary_key=True,
autoincrement=True)
    user_id: Mapped[int] = mapped_column("userId",
ForeignKey("players.userId"), nullable=False, index=True)
    wallet_type: Mapped[str] = mapped_column(String(10), nullable=False)
    balance: Mapped[float] = mapped_column(Numeric(15, 2), default=0)
    currency_code: Mapped[str] = mapped_column(CHAR(3), nullable=False,
default="USD")
    last_updated: Mapped[DateTime | None] = mapped_column(DateTime)

class Session(Base):
    __tablename__ = "sessions"

    session_id: Mapped[int] = mapped_column(Integer, primary_key=True,
autoincrement=True)
    user_id: Mapped[int] = mapped_column("userId",
ForeignKey("players.userId"), nullable=False, index=True)
    token: Mapped[str] = mapped_column(String(255), unique=True,
nullable=False)
    login_time: Mapped[DateTime | None] = mapped_column(DateTime)
    logout_time: Mapped[DateTime | None] = mapped_column(DateTime)
    status: Mapped[str | None] = mapped_column(String(10), default="active")
    login_ip: Mapped[str] = mapped_column("Login_IP", String(255))  # NOTE:
your schema has UNIQUE on this

class PlayerExternalId(Base):
    __tablename__ = "player_external_ids"
    __table_args__ = (
        UniqueConstraint("provider", "external_id", name="u_provider_ext"),
    )
    id: Mapped[int] = mapped_column(Integer, primary_key=True,
autoincrement=True)
    user_id: Mapped[int] = mapped_column("userId",
ForeignKey("players.userId"), index=True)
    provider: Mapped[str] = mapped_column(String(50))
    external_id: Mapped[str] = mapped_column(String(255))
    username: Mapped[str | None] = mapped_column(String(255))
    verified: Mapped[int] = mapped_column(Integer, default=1)
    created_at: Mapped[DateTime | None] = mapped_column(DateTime)
```

D) `app/routes/oauth_instagram.py` — Instagram OAuth (start, callback) + minimal HTML pages

```
from fastapi import APIRouter, Depends, HTTPException, Request
from fastapi.responses import RedirectResponse, HTMLResponse
from sqlalchemy.orm import Session
from sqlalchemy import select
import httpx, secrets, urllib.parse

from ..config import settings
from ..db import get_db
from ..models import PlayerExternalId, Session as DbSession
from ..utils.security import new_session_token, now
from ..utils.ip import get_client_ip

router = APIRouter(prefix="/oauth/instagram", tags=["oauth-instagram"])

OAUTH_AUTHORIZE = "https://api.instagram.com/oauth/authorize"
OAUTH_TOKEN     = "https://api.instagram.com/oauth/access_token"
ME_ENDPOINT     = "https://graph.instagram.com/me?fields=id,username"

# --- Helpers ---

def _require_cfg():
    if not (settings.ig_client_id and settings.ig_client_secret and
settings.ig_redirect_uri):
        raise HTTPException(500, "Instagram OAuth not configured")

# --- Pages ---

@router.get("/start")
def start(next: str | None = None):
    _require_cfg()
    state = secrets.token_urlsafe(16)
    params = {
        "client_id": settings.ig_client_id,
        "redirect_uri": settings.ig_redirect_uri,
        "scope": "user_profile",
        "response_type": "code",
        "state": state,
    }
    url = f"{OAUTH_AUTHORIZE}?{urllib.parse.urlencode(params)}"
    return RedirectResponse(url)

@router.get("/callback")
async def callback(request: Request, code: str | None = None, state: str |
None = None, db: Session = Depends(get_db)):
    _require_cfg()
    if not code:
        raise HTTPException(400, "Missing code")

    # 1) Exchange code -> token
    data = {
        "client_id": settings.ig_client_id,
        "client_secret": settings.ig_client_secret,
        "grant_type": "authorization_code",
        "redirect_uri": settings.ig_redirect_uri,
        "code": code,
    }
    async with httpx.AsyncClient(timeout=20) as client:
        tres = await client.post(OAUTH_TOKEN, data=data)
        if tres.status_code != 200:
            raise HTTPException(400, f"Token exchange failed: {tres.text}")
        payload = tres.json()
        access_token = payload.get("access_token")
        # user_id = payload.get("user_id")  # not needed if we call /me

        mres = await client.get(ME_ENDPOINT, params={"access_token":
access_token})
        if mres.status_code != 200:
            raise HTTPException(400, f"Profile fetch failed: {mres.text}")
        me = mres.json()
        ig_id = str(me.get("id"))
        ig_username = me.get("username") or ""

    # 2) If IG id already linked → sign in locally
    existing = db.scalar(select(PlayerExternalId).where(
        PlayerExternalId.provider == "instagram",
        PlayerExternalId.external_id == ig_id,
    ))
    if existing:
        token = new_session_token()
        ip = get_client_ip(request)
        db.add(DbSession(user_id=existing.user_id, token=token, login_ip=ip,
status="active", login_time=now()))
        db.commit()
        return HTMLResponse(f"""
            <h2>Login successful</h2>
            <p>Welcome back, IG: {ig_username} (id {ig_id}).</p>
            <p>Your session token: <code>{token}</code></p>
        """)

    # 3) If IG not linked → show Finish Signup page
    q = urllib.parse.urlencode({"ig_id": ig_id, "ig_username": ig_username})
    return RedirectResponse(url=f"/signup/complete?{q}")
```

E) Extend `app/routes/auth.py` — finish signup + (already existing) register/login

```
from fastapi import APIRouter, Depends, HTTPException, Request
from pydantic import BaseModel, EmailStr
from sqlalchemy import select
from sqlalchemy.orm import Session

from ..db import get_db
from ..models import Player, Wallet, Session as DbSession, PlayerExternalId
from ..utils.security import hash_password, verify_password,
new_session_token, now
from ..utils.ip import get_client_ip
from ..config import settings

router = APIRouter(prefix="/auth", tags=["auth"])

class CompleteSignupReq(BaseModel):
    email: EmailStr
    password: str
    language_code: str = "en"
    country: str | None = None
    ig_id: str
    ig_username: str | None = None

@router.post("/signup/complete")
def complete_signup(req: CompleteSignupReq, request: Request, db: Session =
Depends(get_db)):
    existing = db.scalar(select(Player).where(Player.email == req.email))
    if existing:
        raise HTTPException(409, "Email already registered")

    p = Player(
        email=req.email,
        password_hash=hash_password(req.password),
        language_code=req.language_code[:2],
        country=req.country,
        status="active",
        ext_user_id=req.ig_id,  # optional copy
    )
    db.add(p); db.flush()

    # default wallets
    for code in (c.strip().upper() for c in
settings.default_wallet_currencies.split(",")):
        db.add(Wallet(user_id=p.user_id,
wallet_type=settings.default_wallet_type, balance=0, currency_code=code))

    # link IG
    db.add(PlayerExternalId(user_id=p.user_id, provider="instagram",
external_id=req.ig_id, username=req.ig_username or "", verified=1))

    # local session
    token = new_session_token()
    ip = get_client_ip(request)
    db.add(DbSession(user_id=p.user_id, token=token, login_ip=ip,
status="active", login_time=now()))
    db.commit()

    return {"userId": p.user_id, "session_token": token}

# (Your existing /register, /login endpoints remain; omitted here for
brevity)
```

F) Minimal HTML page for Finish Signup Add to `app/main.py` (or make a small `pages.py` router):

```
from fastapi import FastAPI, Request
from fastapi.responses import HTMLResponse
from .routes.auth import router as auth_router
from .routes.oauth_instagram import router as ig_oauth_router

app = FastAPI(title="IGW Local")
app.include_router(auth_router)
app.include_router(ig_oauth_router)

@app.get("/signup/complete", response_class=HTMLResponse)
async def signup_complete_page(request: Request, ig_id: str, ig_username: str
= ""):
    return f"""
    <html><body>
      <h2>Finish Signup</h2>
      <p>Instagram: {ig_username} (ID {ig_id})</p>
      <form method="post" action="/auth/signup/complete">
        <input type="hidden" name="ig_id" value="{ig_id}">
        <input type="hidden" name="ig_username" value="{ig_username}">
        <label>Email <input name="email" type="email" required></label><br>
        <label>Password <input name="password" type="password" required></
label><br>
        <label>Language <input name="language_code" value="en"></label><br>
        <label>Country <input name="country" value="GB"></label><br>
        <button type="submit">Create account</button>
      </form>
    </body></html>
    """

@app.get("/health")
def health():
    return {"status": "ok"}
```

The form posts as `application/x-www-form-urlencoded`. FastAPI will parse it into `CompleteSignupReq` automatically because field names match.

---

5) Test Plan (Dev)

1. **Run**: `uvicorn app.main:app --reload --port 8001`
2. **Expose**: `ngrok http 8001` → copy the HTTPS URL.
3. Put the ngrok URL in Facebook App **Valid OAuth Redirect URIs** & `.env` → `IG_REDIRECT_URI`.
4. In a browser, open: `https://<ngrok>.ngrok.io/oauth/instagram/start`
5. Log in as an **Instagram Test User** and accept permissions.
6. You should be redirected to `/signup/complete?ig_id=...&ig_username=...`.
7. Fill the **Finish Signup** form → submit. You get JSON with `{userId, session_token}`.
8. Confirm in DB: new row in `players`, wallets for **USD** and **VND**, and a mapping row in `player_external_ids`.
9. Re-run step 4 with the same IG account → you should see **Login successful** page (no need to re-signup).

---

6) Production Notes (when you're ready)

- Prefer **HttpOnly cookies** for session token storage; avoid query strings.
- Implement **/fb/deauth** and **/fb/delete** endpoints to satisfy policy (they can be stubs initially).
- Consider exchanging for a **long-lived token** (optional for just id/username): `GET https://graph.instagram.com/access_token?grant_type=ig_exchange_token&client_secret=<APP_SECRET>&access_token=<SHORT_TOKEN>`
- Keep Instagram login **optional**; offer email + magic link/passkeys as a fallback.
- Ensure **rate limits** and **bot protection** on signup.

---

7) Troubleshooting

- **Redirect URI mismatch**: Make sure `.env` `IG_REDIRECT_URI` exactly matches FB App settings (including scheme and path).
- **Only testers can login**: Add your IG account as **Instagram Tester** and accept invite.
- **500 during register**: Check column names in `models.py` match your DB (see `SHOW COLUMNS FROM players;`).
- **Duplicate Login_IP**: Your `sessions.Login_IP` is unique; remove that index if you want multiple sessions from same IP: `ALTER TABLE sessions DROP INDEX Login_IP;`

8) Next Steps

- Swap the Finish Signup HTML for a proper React/Next.js page later.
- After login, show a **wallet selector** (USD/VND).
- Then add a **game lobby** → when you integrate providers, you’ll have everything ready to use the local session.