# Algorithm for SwiftGig Smart Contract Workflow

## Step 1: User Profiles Initialization

- Create Talent and Client profiles as on-chain objects.
- Talent metadata: full name, skill, preferred gigs, credibility score.
- Client metadata: full name, extra info, credibility score.
- Initialize credibility scores at baseline value (e.g., 50).

## Step 2: Gig Creation & Treasury Setup

- Client creates a new Gig object with metadata (name, description, reward, deadlines, number of talents needed).
- Reward is deposited into Gig Treasury at creation (escrow).
- Gig state = Active.

## Step 3: Talent Applications

- Talents apply to the gig before deadline.
- Applied talents are stored in waitlist (vector or table).

## Step 4: Talent Selection

- Client selects talent(s) from waitlist based on preference or credibility.
- Selected talents moved to 'accepted list' of the gig.

## Step 5: Work Submission

- Accepted talents submit their completed work before deadline.
- Submissions stored in gig object with metadata (URI, timestamp).
- Gig state = Submitted.

## Step 6: Client Review

- Client reviews submissions.
- If satisfied → Gig state = ReviewSatisfied.
- If unsatisfied → Gig state = ReviewUnsatisfied (reason required).

## Step 7: Reward Distribution

- If satisfied:
- - Talents claim rewards OR client distributes rewards equally/weighted.
- Treasury empties → Gig state = Closed.
- If unsatisfied and not contested:
- - Treasury refunded to client.
- If unsatisfied and contested:
- - Treasury frozen, dispute object created.

## Step 8: Dispute Resolution

- Dispute triggers a community poll among high-credibility users.
- Votes recorded on-chain (one per eligible voter).
- After voting deadline, majority determines winner:
- - Talent wins → Treasury released to talent.
- - Client wins → Treasury refunded to client.
- Credibility scores updated accordingly.
- Gig state = Closed.

## Step 9: Safeguards & Timeouts

- Refund client if no talents selected before application deadline.
- Refund client if no work submitted by completion deadline.
- Prevent double voting in disputes.
- Lock treasury during distribution or dispute resolution to prevent re-entry attacks.

# SwiftGig Smart Contract Pseudocode

```
CONTRACT SwiftGig:

    STRUCT TalentProfile:
        id              // unique ID
        full_name       // string
        skill           // string
        preferred_mode  // remote, physical, any
        credibility     // integer score

    STRUCT ClientProfile:
        id              // unique ID
        full_name       // string
        extra_info      // string
        credibility     // integer score

    STRUCT Gig:
        id              // unique gig ID
        creator         // client address
        metadata        // name, description, deadlines, reward
        treasury        // locked reward funds
        waitlist[]      // list of applicants
        accepted[]      // selected talents
        submissions[]   // {talent, URI, timestamp}
        state           // Active, Submitted, ReviewSatisfied, ReviewUnsatisfied, Dispute, Closed

    STRUCT Dispute:
        reason_client   // string
        reason_talent   // string (if contested)
        poll            // {votes_for_talent, votes_for_client, voters[]}
        resolved        // boolean
        winner          // Talent | Client

    FUNCTION create_profile(role, metadata):
        profile = new Profile
        profile.credibility = 50
        RETURN profile

    FUNCTION create_gig(client, metadata, reward):
        REQUIRE client is Client
        REQUIRE reward > 0
        gig = new Gig
        gig.treasury = reward
        gig.state = "Active"
        RETURN gig

    FUNCTION apply_to_gig(gig, talent):
        REQUIRE talent is Talent
        REQUIRE gig.state == "Active"
        ADD talent TO gig.waitlist

    FUNCTION select_talents(gig, client, selected_list):
        REQUIRE client == gig.creator
        MOVE selected_list TO gig.accepted

    FUNCTION submit_work(gig, talent, uri):
        REQUIRE talent ∈ gig.accepted
        ADD {talent, uri, now()} TO gig.submissions
        SET gig.state = "Submitted"

    FUNCTION review_submission(gig, client, satisfied, reason):
        REQUIRE client == gig.creator
        IF satisfied:
            SET gig.state = "ReviewSatisfied"
        ELSE:
            SET gig.state = "ReviewUnsatisfied"
            STORE reason IN dispute.reason_client

    FUNCTION distribute_rewards(gig):
        REQUIRE gig.state == "ReviewSatisfied"
        SPLIT gig.treasury AMONG gig.accepted
        SET gig.state = "Closed"

    FUNCTION refund_client(gig):
```

```
    REQUIRE gig.state == "ReviewUnsatisfied" AND no contest
    TRANSFER gig.treasury TO gig.creator
    SET gig.state = "Closed"

FUNCTION contest_decision(gig, talent, reason):
    REQUIRE talent ∈ gig.accepted
    SET gig.state = "Dispute"
    STORE reason IN dispute.reason_talent

FUNCTION vote_in_dispute(gig, voter, choice):
    REQUIRE voter.credibility >= threshold
    RECORD vote IN dispute.poll

FUNCTION resolve_dispute(gig):
    CALCULATE majority FROM dispute.poll
    IF majority == Talent:
        DISTRIBUTE treasury TO talents
    ELSE:
        REFUND treasury TO client
    SET gig.state = "Closed"

FUNCTION adjust_credibility(user, delta):
    UPDATE user.credibility = clamp(user.credibility + delta, 0, 100)
```