Algorithm: Insurance Claim Voting System

1. Initiate Claim Request

- The user (claimant) submits a claim by providing:
    - **Purpose** of the claim.
    - **Amount** requested.
    - **Uploaded documents** (e.g., bills, reports).
- The system associates the claim with the user's:
    - **Profile details** (e.g., name, age, occupation, etc.).
    - **Risk score** (calculated earlier).
    - **Claim history** (past claims and their statuses).

Store the claim in the database with a unique claim ID and initial status: `Pending`.

---

2. Notify Community Members

- Fetch the list of **community members** excluding the claimant.
- Notify all members in the community about the new claim via:
    - A **dashboard notification** or **email**.
- Provide them with:
    - The claimant's profile details.
    - Risk score.
    - History of claims and decisions.
    - Current claim details (purpose, amount, documents).

---

3. Voting Phase

- Each member of the community can cast their vote:
    - **Yes** (approve the claim).
    - **No** (reject the claim).
- Members must submit their votes within a **defined time limit** (e.g., 48 hours).

Votes are stored in the database as:

```
 {
  "claim_id": "<claim_id>",
  "user_id": "<voter_id>",
  "vote": "Yes" or "No"
}
```

---

4. Calculate Results

- After the voting period ends, the system tallies the votes:
    - Count total **Yes** and **No** votes.
    - Calculate the **majority threshold**:
        - If more than **50%** of total votes are "Yes," the claim is approved.
        - Otherwise, the claim is rejected.

---

5. Update Claim Status

- If **approved**:
    - Update the claim status to `Approved`.
    - Notify the claimant and initiate fund transfer for the requested amount.
- If **rejected**:
    - Update the claim status to `Rejected`.
    - Notify the claimant with the rejection reason (e.g., lack of majority).

---

6. Handle Edge Cases

- If no votes are cast within the time limit, the claim status defaults to `Rejected`.
- If there's a **tie** (equal "Yes" and "No" votes):
    - The claim is sent to the **community admin** for the final decision.
- If users repeatedly fail to vote:
    - Consider adding a penalty or reminder system.

Flow Diagram

1. Claimant submits the claim.
2. Notify members → Members vote → Votes tallied.
3. Decision:
    - Majority "Yes" → Approve and release funds.
    - Majority "No" → Reject with reason.
    - Tie → Admin decides.

SQL Database Structure

1. **Claims Table**:
   - `claim_id`, `user_id`, `purpose`, `amount`, `documents`, `status`, `created_at`
2. **Votes Table**:
   - `vote_id`, `claim_id`, `user_id`, `vote`, `created_at`
3. **Users Table**:
   - `user_id`, `name`, `risk_score`, `claim_history`, `profile_details`

Required Tools

1. **Frontend**
   - React.js for user interfaces
   - TypeScript for better code reliability
   - Tailwind CSS or Bootstrap for styling
   - Axios for API communication
   - Socket.IO for real-time updates
2. **Backend**
   - Flask or Node.js with Express for API and server logic
   - PostgreSQL or MySQL for structured data storage
   - Python for integrating the ML model
   - Flask-SocketIO or Socket.IO for real-time voting notifications
3. **Authentication and Security**
   - JSON Web Tokens (JWT) for session management
   - HTTPS with SSL/TLS for secure data transmission
   - AWS S3 or Google Cloud Storage for document uploads
4. **Deployment and Monitoring**
   - Docker for containerization
   - AWS EC2 or Heroku for hosting
   - Prometheus and Grafana for performance monitoring
5. **Machine Learning Integration**
   - scikit-learn for prediction handling in Python
   - Flask endpoints to serve the ML model predictions

Implementation Notes

- Use a real-time **websocket** or **polling** mechanism to update votes in the UI.
- Ensure **data security** for uploaded documents and claims.
- Create a dashboard to display voting progress and decisions transparently.