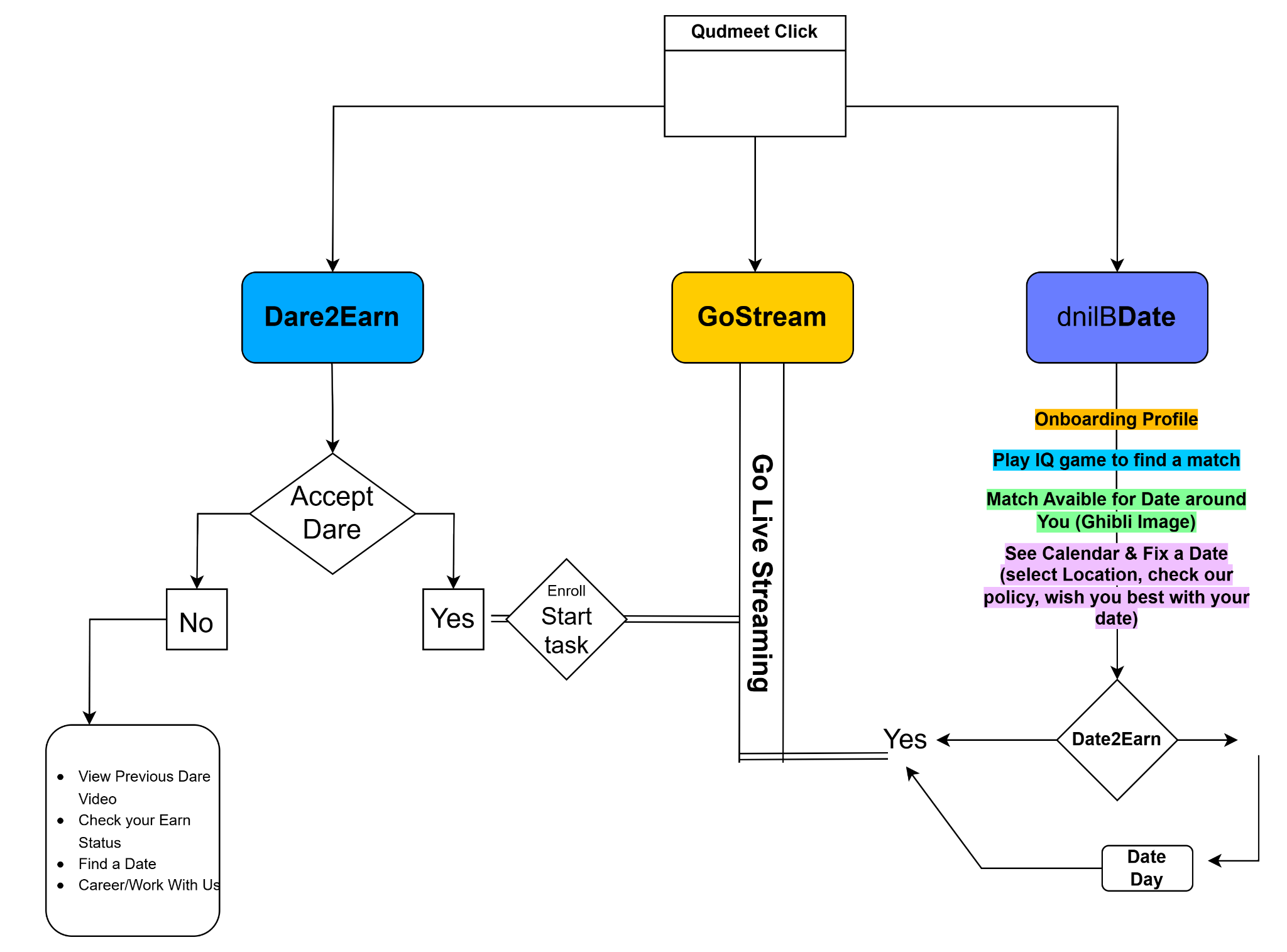
I want to create a application call Qudmeet click. where users can meet each other, request for a blind date meet. here people don't chat with each other, we only automatically match them based on there profile and also considering there performance in the blind date game every users are required to play before they get match. although I have not yet think of the game yet but am considering implementing Generative AI for the game automation. while the game will consist of image based, IQ based, dating base, text base, and fun. Now the main features of the application are: **dnildDate(blindDate), GoStream, Dare2Earn**

1. Match making and blind date fixer
2. Live streaming during date to earn money through monetize feature (user can apply for monetize feature which is a paid plan monthly fee, and the user get a digital score point for every stream of user view on their live blind date video with their match recorded during the date session”
3. Dare to do live video task (we pay users for every dare they complete like we can ask them to make a live video of themself proposing to a stranger on the road, make a particular dance move, do a push ups etc) the dare to do task will be make available general on the platform and anyone can accept the dare. The first person to complete the dare within the set timing deadline will be rewarded. And they can see the reward on their profile and request for catch out.
4. The live streaming and view and video uploaded on the platform should be automatic reflect in our youtube channel page (optional)

Below is the flow diagram

# Qudmeet Click - Complete Project Documentation

## Table of Contents

1. Project Overview
2. System Architecture
3. Core Features
4. Workflows
5. Technical Implementation
6. AI Integration
7. Database Schema
8. API Documentation
9. Deployment Guide
10. Future Enhancements

## Project Overview <a name="project-overview"></a>

Qudmeet Click is an innovative dating platform that combines AI-powered matchmaking with gamification and monetization features. The platform eliminates traditional chatting and focuses on automated matching based on profile data and performance in blind date games.

Key Differentiators:

* Mandatory AI-powered blind date games before matching
* Live streaming of dates with monetization
* Dare-based reward system (Dare2Earn)
* Automated YouTube content syndication

## System Architecture <a name="system-architecture"></a>

[https://diagram.png](https://diagram.png/)

## Core Features <a name="core-features"></a>

### 1. dnildDate (Blind Date Matchmaking)

Feature Flow:

1. User onboarding with profile creation
2. Mandatory completion of blind date games:
   * Image-based interpretation
   * IQ challenges
   * Dating scenario responses
   * Text analysis tasks
   * Fun activities
3. AI-powered compatibility scoring
4. Automated match suggestions
5. Date scheduling interface

AI Components:

* GPT-4 for question generation
* Gemini for response analysis
* Custom matching algorithm

### 2. GoStream (Live Streaming)

Monetization Model:

* Monthly subscription for streamers ($9.99/month)
* Digital score points based on:
  + View count (1 point per 100 views)
  + Engagement (likes, shares)
  + Duration (1 point per 10 minutes)

Technical Implementation:

* Livepeer integration for streaming
* Real-time viewership analytics
* Automated highlight clipping

### 3. Dare2Earn

Workflow:

1. System generates dares (daily/weekly)
2. Users accept challenges:
   * Public proposals
   * Dance challenges
   * Fitness tests
   * Social experiments
3. Video submission and AI verification
4. Reward distribution to first completer

Reward Structure:

* Small dares: $1-$5
* Medium dares: $5-$20
* Viral dares: $20-$100

## Workflows <a name="workflows"></a>

### User Registration Flow

Diagram

Code

Download

Sign Up

Profile Creation

Game Completion

AI Analysis

Match Suggestions

Date Scheduling

### Dare2Earn Flow

Diagram

Code

Download

Dare Published

User Accepts

Video Submission

AI Verification

Reward Distribution

YouTube Sync

## Technical Implementation <a name="technical-implementation"></a>

### Tech Stack

* Frontend: Next.js 14 (App Router)
* Backend: Next.js API Routes
* Database: Neon PostgreSQL
* Authentication: Clerk
* AI Services: OpenAI, Gemini
* Streaming: Livepeer
* Automation: n8n

### Key Dependencies

json

Copy

Download

{

"dependencies": {

"@clerk/nextjs": "^4.25.0",

"livepeer": "^2.15.0",

"@google/generative-ai": "^0.8.0",

"openai": "^4.28.0",

"@neondatabase/serverless": "^0.5.0"

}

}

## AI Integration <a name="ai-integration"></a>

### Matchmaking Algorithm

typescript

Copy

Download

async function calculateCompatibility(user1: User, user2: User) {

const prompt = `

Analyze compatibility between:

User 1: ${JSON.stringify(user1)}

User 2: ${JSON.stringify(user2)}

Consider:

- Game performance

- Personality traits

- Dating preferences

Return { score: number, explanation: string }

`;

const response = await openai.chat.completions.create({

model: "gpt-4",

messages: [{ role: "user", content: prompt }],

response\_format: { type: "json\_object" }

});

return JSON.parse(response.choices[0].message.content);

}

### Dare Verification

typescript

Copy

Download

async function verifyDareCompletion(videoUrl: string, dareText: string) {

const model = genAI.getGenerativeModel({ model: "gemini-pro-vision" });

const result = await model.generateContent([

`Verify if this video shows completion of: "${dareText}"`,

{ mimeType: "video/mp4", url: videoUrl }

]);

return result.response.text().includes("YES");

}

## Database Schema <a name="database-schema"></a>

### Tables Structure

sql

Copy

Download

*-- Users Table*

CREATE TABLE users (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

clerk\_id VARCHAR(255) UNIQUE NOT NULL,

name VARCHAR(100) NOT NULL,

email VARCHAR(255) UNIQUE,

preferences JSONB DEFAULT '{}',

game\_scores JSONB DEFAULT '{}',

streaming\_enabled BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP DEFAULT NOW()

);

*-- Matches Table*

CREATE TABLE matches (

id SERIAL PRIMARY KEY,

user1\_id UUID REFERENCES users(id),

user2\_id UUID REFERENCES users(id),

compatibility\_score FLOAT,

scheduled\_time TIMESTAMP,

location VARCHAR(255),

status VARCHAR(50) DEFAULT 'pending'

);

*-- Dares Table*

CREATE TABLE dares (

id SERIAL PRIMARY KEY,

title VARCHAR(255) NOT NULL,

description TEXT,

reward DECIMAL(10,2) NOT NULL,

deadline TIMESTAMP,

completed\_by UUID REFERENCES users(id),

verification\_video\_url VARCHAR(512)

);

## API Documentation <a name="api-documentation"></a>

### Key Endpoints

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/games/start | POST | Initiate blind date games |
| /api/matches/suggest | GET | Get match suggestions |
| /api/stream/start | POST | Begin live stream |
| /api/dares/list | GET | List available dares |
| /api/dares/submit | POST | Submit dare completion |

## Deployment Guide <a name="deployment-guide"></a>

### Vercel Deployment Steps

1. Connect your GitHub repository
2. Set environment variables:
3. text
4. Copy
5. Download

CLERK\_SECRET\_KEY=your\_key

OPENAI\_API\_KEY=your\_key

LIVEPEER\_API\_KEY=your\_key

1. DATABASE\_URL=neon\_connection\_string
2. Configure build settings:
3. text
4. Copy
5. Download

Framework: Next.js

Build Command: npm run build

1. Output Directory: .next
2. Deploy!

……………………………………………………………….