Mathletics Architecture

1. List of Features

S. No.	Feature Name	Short Description	Phase	Priority
1	START SCREEN	Starting Screen with Game Logo / Company Logo	PHASE 1	1
2	SIGN IN PAGE	Sign in to the Game Using Google/ FB/ etc		2
3	HOME PAGE	Home Page - Post Signing In		3
4	SIGN UP FLOW	Sign Up onto the Game using Email		4
5	GAME PLAY	Section where you select difficulties		5
6	USER PROFILE	Login/ Age/ Gender/ Location/ Others		6
7	GAMIFICATION	Point System for each Game	PHASE 2	7
8	COMPUTER MODE	Add a Computer Mode to difficulty		8
9	MULTIP. MODE	Person to Person Game Online		9
10	IN GAME REACTION	Share reactions with other players in Multiplayer mode		10
11	SOUND	Add Sound to Click, Completion etc		11
12	ADD FRIENDS	Add users to play with as friends to start a match with them		12
13	NOTIFICATIONS	In-App Notifications/ Pop-Ups		13
14	THEME	Change Background, Font which can be bought from Store		14
15	HISTORY & STATS	Stats of basic KPIs for each user, and previously played games		15
16	LEADERBOARD	Leaderboard to show, rankings globally, country wise & with friends		16
17	ACHIEVEMENTS	Badges that User get on completing some task		17
18	DEV. DASHBOARD	To analyse app performance & user journey		18
19	PUSH NOTIFICATION	To send to users regarding game updates etc.		19
20	ADVERTISEMENT	Integrate Ads to generate ad based revenue		20
21	LANGUAGE	Add Languages other than English		21
22	DEMO	Demo Screen (to show people how to start)		22
23	Analysis	Analysis Post Each Game		23
24	ADD. GAME PL OPTION	Add difficulties in Game Play for multiplayer mode		24
25	ADMIN PANEL	Part of the Dashboard	PHASE 3	25
26	TOURNAMENTS	Daily Tournaments for Players (Based on Rating) to register/ play and get rewards		26
27	CHALLENGE	A Section which opens Challenge/ Puzzle Section		27
28	CHALLENGE GAMIFICATION	Will allow users to answer difficult questions like Normal Game and also allow users to take hint/ life etc.		28
29	DAILY REWARDS	Daily coin system to bring back users		29
30	LEARN	A Section which opens Learn Section		30
31	LEARN DETAILS	Page where tips & Tricks open within the learn section. Could be small video, infographic or link with text		31
32	STORE	Store to buy using themes, special items using Coins, Cash		32
33	IN GAME MSG	Send messages to people		33

Technical Architecture

Comprehensive architecture for the mobile-first math game described. Covers high-level components, data model, API surface, realtime & multiplayer design, infrastructure, security, observability, and scaling guidance.

High-Level Component Diagram (textual)

1. **Client Apps**
    - Mobile: React Native (iOS + Android), native modules for IAP and push.
    - Web: React js (Admin / Leaderboards / Store).
2. **API Layer**
    - Gateway / Edge: CDN + API Gateway (Auth, rate limit, TLS termination).
    - Public REST APIs (stateless): Node.js (Express)
3. **Realtime Layer**
    - WebSocket servers (Socket.IO / uWebSockets / Phoenix Channels) behind a Load Balancer.
    - Matchmaking service (stateless microservice) + Redis for ephemeral state & pub/sub.
4. **Core Services / Microservices**
    - Auth & Identity (OAuth + social logins + JWT).
    - Game Engine (question generation, scoring, timer logic).
    - Matchmaking & Multiplayer.
    - Achievements & Gamification.
    - Notifications (push & in-app messages).
    - Analytics / Events pipeline.
    - Admin & Developer Dashboard.
5. **Data Stores**
    - Primary DB: MongoDB (relational data & schema).
    - Cache / Real-time store: Redis (leaderboards, locks, session store, pub/sub).
    - Time-series / Analytics: ClickHouse or BigQuery (events & metrics).
    - Object Storage: S3 / GCS (avatars, themes, media).
6. **Integration & 3rd Parties**
    - FCM / APNs for push notifications.
    - OAuth providers (Google, Apple, Facebook).
    - Ad Networks (AdMob) and analytics (Firebase Analytics / Amplitude / Mixpanel).

Data Model (summary)

Tables

- `users` (id, email,password, first_name, last_name, year_of_birth,phone, social_ids, language, country, created_at, last_login,last_updated,status, is_verified)

- `profiles` (user_id, name, gender, age, location, avatar_url, coins, gems, country,player_rating,photo,keyboard/theme)
- sound_setting(id,music)
- `games` (game_id, mode, difficulty_settings_json, created_by, created_at)
- `game_sessions` (session_id, user_id, game_id, start_ts, end_ts, score, result_json)
- `questions` (question_id, question_text, input_range, result_range, symbols, feedback, correct_answer, metadata)
- `challenges` (challenge_id, title, description, difficulty, best_time)
- `achievements` (achievement_id, name, criteria_json)
- `user_achievements` (user_id, achievement_id, awarded_at)
- `friends` (user_id, friend_user_id, status, created_at)
- friends_profile(user_id,games_id,games_played,win_count, loss_count,draw_count)
- `inbox_messages` (message_id, from_user, to_user, session_id, message_text, created_at)
- `leaderboard_daily` (date, scope, user_id, score) — can be derived / or maintained in Redis sorted sets
- `notifications` (notif_id, user_id, payload_json, sent_at, read_at)
- feedback(question_id, correct_answer, question_symbol, correctly_answered, incorrectly_answered,skipped_answer, time_taken)
- game_header(game_id, player_ids, game_type, correct_questions_count (each), incorrect_questions_count, skipped_questions_count (each player), scores (each), game time_type, game_difficulty_type, game_symbol_type, winner, opponent_type, game_datetime, player_rating _start, player_rating_end)
- game_details(game_id, question_number, question_id, answer (each player), answer_status (each), points (each),bonus_points (each), other_points (each), correct_answer, time_taken (each), question_meter)

Redis usage

- Active WebSocket sessions mapping: ws:session:{sessionId} -> {userId, socketId, ttl}
- Leaderboards: Redis Sorted Sets (zadd, zrangebyscore) per scope (global, country, friends)
- Matchmaking queues by difficulty/score bucket: Lists or Sorted Sets
- Short-lived locks & counters

API Surface

Auth

- POST `/auth/register` — create account (email / phone)
- POST `/auth/social` — social login (Google/Apple/Facebook) -> returns JWT

- POST `/auth/login` — login (email/phone)
- POST `/auth/refresh` — refresh token

Profile

- GET `/me` — user profile
- PUT `/me` — update profile (name, gender, age, location, avatar)
- GET `/users/:id/friends`
- POST `/friends` — add friend

Game & Questions

- POST `/game/start` — request new game (mode + difficulty) -> returns session id & first question
- POST `/game/answer` — submit an answer (session_id, question_id, answer) -> returns result, next question, updated timer/score
- POST `/game/finish` — finish game session -> compute final score
- GET `/user/games?limit=X` — fetch last X games
- GET `/challenges` — list active challenges
- POST `/challenge/:id/submit` — submit completion time

Multiplayer

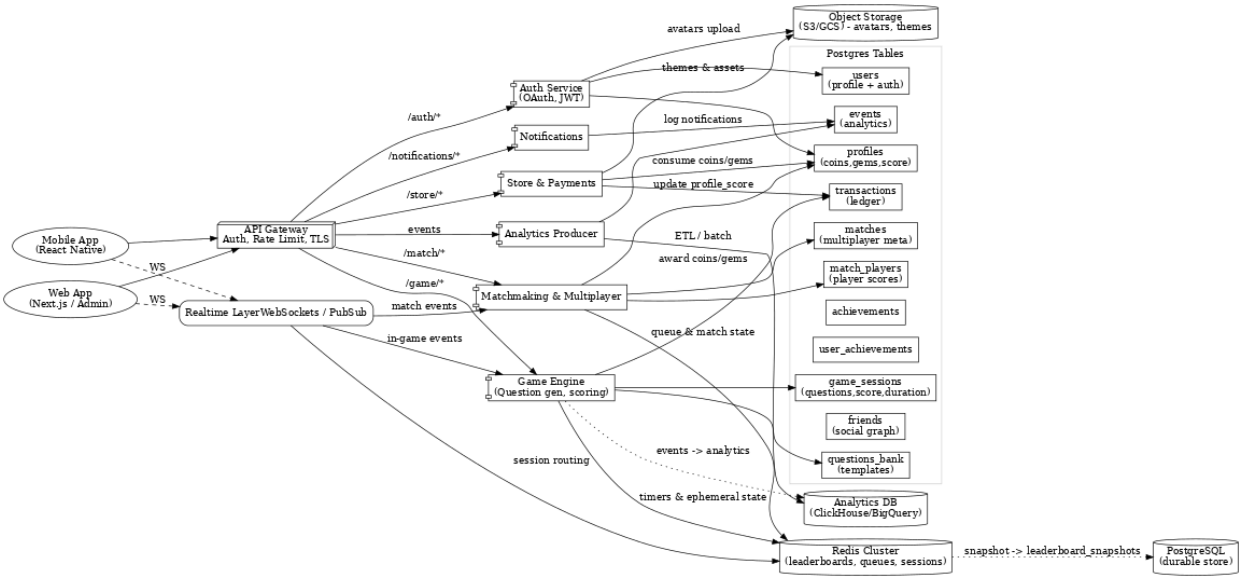
- POST `/match/join` — join matchmaking (user_id, mode, difficulty)
- POST `/match/leave`
- WebSocket namespace `/ws/match` for realtime events: `match.started`, `player.answer`, `match.result`

Notifications

- POST `/notifications/send` — internal API to queue in-app / push

Admin

- GET `/admin/stats` — app analytics
- POST `/admin/push` — targeted push



Database Architecture

This document provides a concrete database architecture for the MathGame app: relational schema (MongoDB) for durable data, Redis patterns for real-time state & leaderboards, and guidance on indexing, partitioning, scaling, backups and sample SQL snippets.

Design principles

- Authoritative & durable state (users, payments, achievements, audit logs, historical snapshots).
- High-speed runtime state in Redis (active sessions, leaderboards, matchmaking queues, ephemeral timers).
- Event-driven analytics: produce immutable events into an events table or Kafka for downstream analytics (ClickHouse/BigQuery).
- Partitioning by time for high-volume tables (game_sessions, events) to improve query performance and retention.
- Use JSONB selectively for flexible/dynamic fields (difficulty settings, question payloads), but keep frequently queried fields as typed columns.

Key Entities & Relationships (Summary)

- users (1) — (1) profiles
- users (1) — (N) game_sessions
- game_sessions (1) — (N) questions_attempted (or questions stored in JSONB inside game_sessions)
- users (1) — (N) user_achievements
- users (1) — (N) transactions
- users (1) — (N) friends (self-referential)
- matches (1) — (N) match_players — stores multiplayer matches
- Leaderboards implemented in Redis and periodically snapshotted to mongoDB leaderboard_snapshots

---

Schema (recommended types & indexes)

Below are core tables with example CREATE TABLE snippets. Use UUID primary keys for easier sharding and security.

1. Users

```
CREATE TABLE users (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
email TEXT UNIQUE,
phone TEXT UNIQUE,
password_hash TEXT,
created_at timestamptz DEFAULT now(),
last_login timestamptz,
auth_providers JSONB, -- {google: id, apple: id}
is_verified BOOLEAN DEFAULT false
);
CREATE INDEX idx_users_last_login ON users(last_login);
```

2. Profiles

```
CREATE TABLE profiles (
user_id UUID PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE,
display_name TEXT,
gender TEXT,
age SMALLINT,
country_code CHAR(2),
city TEXT,
```

```
avatar_url TEXT,
coins BIGINT DEFAULT 0,
gems BIGINT DEFAULT 0,
profile_score DOUBLE PRECISION DEFAULT 1000.0, -- ELO-like score
created_at timestamptz DEFAULT now(),
updated_at timestamptz DEFAULT now()
);
CREATE INDEX idx_profiles_country ON profiles(country_code);
```

3. game_sessions

High-volume table: store summary per game session.

```
CREATE TABLE game_sessions (

session_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

user_id UUID REFERENCES users(id) NOT NULL,

mode TEXT NOT NULL, -- 'questionnaire' | 'timer' | 'multiplayer'

difficulty JSONB, -- {input_max, result_max, symbols, allow_negative}

questions JSONB,  -- list of questions + answers (or store minimal payload)

score INT,

start_ts timestamptz DEFAULT now(),

end_ts timestamptz,

duration_ms BIGINT,

result JSONB, -- aggregate: correctness, streaks, etc.

);

-- Partition by range on start_ts by month
```

Partitioning strategy: create monthly partitions game_sessions_2025_09, etc. Indices: CREATE INDEX ON game_sessions (user_id, start_ts DESC);

4. Questions_bank

Store question templates (seed-based) and metadata to reproduce questions deterministically.

```
CREATE TABLE questions_bank (

id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

template TEXT, -- e.g. "{a} + {b}"

input_range JSONB,

result_range JSONB,

symbols TEXT[],

difficulty SMALLINT,

created_at timestamptz DEFAULT now()

);

CREATE INDEX idx_questions_bank_difficulty ON questions_bank(difficulty)
```

5. Matches & Match_players (multiplayer)

```
CREATE TABLE matches (
match_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
mode TEXT,
difficulty JSONB,
created_at timestamptz DEFAULT now(),
finished_at timestamptz,
result JSONB
);
CREATE TABLE match_players (
match_id UUID REFERENCES matches(match_id) ON DELETE CASCADE,
user_id UUID REFERENCES users(id),
score INT,
finished BOOLEAN DEFAULT false,
PRIMARY KEY (match_id, user_id)
);
CREATE INDEX idx_match_players_user ON match_players(user_id);
```

6. Achievements and User_achievements

```
CREATE TABLE achievements (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
key TEXT UNIQUE,
title TEXT,
description TEXT,
criteria JSONB
);
CREATE TABLE user_achievements (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
user_id UUID REFERENCES users(id),
achievement_id UUID REFERENCES achievements(id),
awarded_at timestamptz DEFAULT now()
);
CREATE INDEX idx_user_achievements_user ON user_achievements(user_id);
```

7. Friends

```
CREATE TABLE friends (
user_id UUID REFERENCES users(id),
friend_id UUID REFERENCES users(id),
status TEXT, -- 'requested'|'accepted'|'blocked'
created_at timestamptz DEFAULT now(),
PRIMARY KEY (user_id, friend_id)
);
CREATE INDEX idx_friends_user ON friends(user_id);
```

8. Notifications, Inbox_messages

Keep minimal content for persistence; large payloads in JSONB.

9. Events (immutable analytics events)

```
CREATE TABLE events (
id BIGSERIAL PRIMARY KEY,
user_id UUID,
event_type TEXT,
payload JSONB,
created_at timestamptz DEFAULT now()
) PARTITION BY RANGE (created_at);
```

Redis — runtime & patterns

Use Redis Cluster. Keep short-lived, high-throughput info here.

Key patterns

- ws:session:{sessionId} -> hash { userId, socketId, expires }
- leaderboard:global -> ZSET (score -> userId)
- leaderboard:country:{CC} -> ZSET
- match:queue:{difficulty_bucket} -> LIST or ZSET (score -> userId)
- timer:session:{sessionId} -> integer (remaining ms)
- match:state:{matchId} -> HASH or JSON string (current scores, question index)
- streak:{userId} -> INT (consecutive correct answers)

Indexing & Query patterns

- Index game_sessions(user_id, start_ts DESC) for fetching last X games efficiently.
- Index match_players(user_id) for a user's multiplayer history.
- Partial indexes for active users: CREATE INDEX ON users (id) WHERE last_login > now() - interval '30 days'.
- Use materialized views for aggregated daily stats (DAU, avg session length) refreshed hourly.

Delivery Phase

Delivery 1:

1. Sound
2. Themes
3. Notifications
4. Push Notifications
5. Add Friends
6. Admin Panel

Delivery 2:

1. Multiplayer Mode
2. In Game Reaction
3. Leaderboard
4. Language

Delivery 3:

1. Computer Mode
2. History and Stats
3. Dev Dashboard
4. Advertisement

Delivery 4:

1. Analysis
2. Add GAme pl Options
3. Demo
4. Achievements