**📘 Peer Tutor — Peer Tutoring Matching System**

**1) Project Title / Problem Statement**

**Project Title:**  
**Peer Tutor — Peer Tutoring Matching System**

**Problem Statement:**  
The project aims to develop a **peer-tutoring matching system** that connects learners with tutors by matching preferences such as subjects, availability, skill level. Using **Firebase** as the backend, the system applies a **compatibility algorithm** to recommend the most suitable tutors for each learner, simplifying the tutor discovery and session booking process.

**2) Brief Description**

**Peer Tutor** is a **platform** that enables students to find and connect with peer tutors. The system allows both **learners** and **tutors** to register, create profiles, and manage tutoring sessions.

Built entirely with **Firebase (Auth + Firestore)** and **Vanilla JavaScript**, it implements a **compatibility-based tutor matching algorithm** to rank tutors based on learner preferences.

**🔑 Core Features**

* **User Authentication:** Firebase Auth (Email/Password + Google Sign-In)
* **User Roles:** Tutor / Learner
* **Tutor Matching:** Rule-based scoring algorithm + Priority Queue
* **Session Management:** Book, view, cancel tutoring sessions
* **Demo Data Generator:** Auto-seeding for testing and demo purposes
* **Analytics Dashboard:** Simple insights using Chart.js

**3) Industry / Domain**

**Primary Domain:** Education — *EdTech (Peer-to-Peer Learning Platform)*  
**Secondary Domain:** SaaS / Marketplace Platform

**4) Frontend Technology Used**

| **Component** | **Description** |
| --- | --- |
| **HTML** | index.html, login.html, dashboard.html, profile.html, lead.html, analytics.html, profile.html, style.css. |
| **CSS** | UI layout and responsive design. |
| **JavaScript** | Client-side logic, Firebase interaction, and utilities |
| **Firebase JS SDK (CDN)** | firebase-app, auth, firestore, analytics |
| **Chart.js** | Visualization for analytics dashboard in analtics.html |
| **Bootstrap (CSS)** | UI layout and responsive design in index.html |

**5) Backend / Database Technology Used**

| **Service** | **Purpose** |
| --- | --- |
| **Firebase Authentication** | User sign-up/login via Email/Password & Google |
| **Cloud Firestore** | Stores user profiles, sessions, and analytics data |
| **Firebase Hosting** | Deployment and serving of the web app |
| **Firestore Rules & Indexes** | Security and query optimization |

**6) Data Structures & Algorithms (DSA) Used**

**🧠 Algorithms**

1. **Priority Queue (Max Heap) for Tutor Matching**
2. export class PriorityQueue {
3. constructor() {
4. this.heap = [];
5. }
6. enqueue(item, priority) {
7. const node = { item, priority };
8. this.heap.push(node);
9. this.bubbleUp(this.heap.length - 1);
10. }
11. dequeue() {
12. if (this.heap.length === 0) return null;
13. if (this.heap.length === 1) return this.heap.pop().item;
14. const max = this.heap[0];
15. this.heap[0] = this.heap.pop();
16. this.bubbleDown(0);
17. return max.item;
18. }
19. bubbleUp(index) {
20. while (index > 0) {
21. const parentIndex = Math.floor((index - 1) / 2);
22. if (this.heap[index].priority <= this.heap[parentIndex].priority) break;
23. [this.heap[index], this.heap[parentIndex]] = [this.heap[parentIndex], this.heap[index]];
24. index = parentIndex;
25. }
26. }
27. bubbleDown(index) {
28. while (true) {
29. let largest = index;
30. const leftChild = 2 \* index + 1;
31. const rightChild = 2 \* index + 2;
32. if (leftChild < this.heap.length &&
33. this.heap[leftChild].priority > this.heap[largest].priority) {
34. largest = leftChild;
35. }
36. if (rightChild < this.heap.length &&
37. this.heap[rightChild].priority > this.heap[largest].priority) {
38. largest = rightChild;
39. }
40. if (largest === index) break;
41. [this.heap[index], this.heap[largest]] = [this.heap[largest], this.heap[index]];
42. index = largest;
43. }
44. }
45. size() {
46. return this.heap.length;
47. }
48. peek() {
49. return this.heap.length > 0 ? this.heap[0].item : null;
50. }
51. }

| **Operation** | **Time Complexity** | **Explanation** |
| --- | --- | --- |
| enqueue | O(log n) | Worst-case bubbling up from bottom to root |
| dequeue | O(log n) | Worst-case bubbling down from root to leaf |
| peek | O(1) | Directly access heap[0] |
| size | O(1) | Directly access array length |

* **Overall Time Complexity:** O(n log n)
* **Space Complexity:** **Space Complexity:** O(n) → storing n tutors in the heap.

1. **Sliding Window for Active Users Calculation**
2. export function countActiveUsersLastNDays(users, days = 7) {
3. const now = new Date();
4. const windowStart = new Date(now.getTime() - days \* 24 \* 60 \* 60 \* 1000);
5. let activeCount = 0;
6. for (const user of users) {
7. if (user.lastActive && user.lastActive.toDate) {
8. const lastActive = user.lastActive.toDate();
9. if (lastActive >= windowStart && lastActive <= now) {
10. activeCount++;
11. }
12. }
13. }
14. return activeCount;
15. }

| **Operation** | **Time Complexity** | **Explanation** |
| --- | --- | --- |
| Loop through users | O(n) | Need to check all n users once |
| Inside loop | O(1) | Date comparison is constant time |

* **Overall Time Complexity:** O(n)
* **Space Complexity:** O(1)

1. **BFS for Referral Network Analysis**
2. export function analyzeReferralNetworkBFS(referrals, startUserId) {
3. const graph = new Map();
4. for (const ref of referrals) {
5. if (!graph.has(ref.userId)) {
6. graph.set(ref.userId, []);
7. }
8. graph.get(ref.userId).push(ref.referredUserId);
9. }
10. const queue = [{ userId: startUserId, level: 0 }];
11. const visited = new Set([startUserId]);
12. const levels = new Map();
13. let maxDepth = 0;
14. while (queue.length > 0) {
15. const { userId, level } = queue.shift();
16. maxDepth = Math.max(maxDepth, level);
17. if (!levels.has(level)) {
18. levels.set(level, 0);
19. }
20. levels.set(level, levels.get(level) + 1);
21. if (graph.has(userId)) {
22. for (const referredUser of graph.get(userId)) {
23. if (!visited.has(referredUser)) {
24. visited.add(referredUser);
25. queue.push({ userId: referredUser, level: level + 1 });
26. }
27. }
28. }
29. }
30. return {
31. totalReferrals: visited.size - 1,
32. maxDepth,
33. levelDistribution: Object.fromEntries(levels)
34. };
35. }

| **Operation** | **Time Complexity** | **Explanation** |
| --- | --- | --- |
| Build graph | O(E) | E = number of referral edges |
| BFS traversal | O(V + E) | V = number of users (nodes), E = edges |
| Queue operations | O(V) | Each node enqueued/dequeued once |
| Map/Set operations | O(1) | Checking visited or updating levels |

* **Overall Time Complexity:** O(V + E)
* **Space Complexity:** O(V + E) → graph + queue + visited + levels

1. **Scoring / Matching Algorithm**
2. export function calculateCompatibilityScore(tutor, learner) {
3. let score = 0;
4. const tutorSubjects = Array.isArray(tutor.subjects) ? tutor.subjects : [];
5. const learnerSubjects = Array.isArray(learner.subjects) ? learner.subjects : [];
6. const subjectMatch = tutorSubjects.some(subject =>
7. learnerSubjects.includes(subject)
8. );
9. if (subjectMatch) {
10. score += 50;
11. }
12. const tutorAvailability = Array.isArray(tutor.availability) ? tutor.availability : [];
13. const learnerAvailability = Array.isArray(learner.availability) ? learner.availability : [];
14. const availabilityMatch = tutorAvailability.some(slot =>
15. learnerAvailability.includes(slot)
16. );
17. if (availabilityMatch) {
18. score += 30;
19. }
20. const rating = tutor.rating || 0;
21. score += Math.min(rating \* 4, 20);
22. return score;
23. }

| **Operation** | **Time Complexity** | **Explanation** |
| --- | --- | --- |
| Subject match | O(m \* n) | m = tutor subjects, n = learner subjects (some + includes) |
| Availability match | O(p \* q) | p = tutor availability slots, q = learner availability slots |
| Rating calculation | O(1) | Constant time |
| Overall | O(m \* n + p \* q) | Dominated by array comparisons |
| Space Complexity | O(m + n + p + q) | Temporary arrays for subjects and availability |

* **Overall Time Complexity:** O(m \* n + p \* q) → Dominated by array comparisons
* **Space Complexity:** O(m + n + p + q) → Temporary arrays for subjects and availability

**7) Workflow / Integration (End-to-End)**

**Step 1: User Authentication**

* User signs up or logs in using Email or Google via Firebase Auth.
* Redirect based on role (Tutor or Learner).

**Step 2: Profile Setup**

* User fills in subjects, role, availability, etc.
* Data saved to Firestore in “users” collection.

**Step 3: Tutor Matching**

* Learner selects a subject and time slot.
* System queries all tutors and calculates compatibility score.
* Tutors ranked using Priority Queue → top matches are shown to learner.

**Step 4: Session Booking**

* Learner selects a tutor → creates a new “session” collection in Firestore.
* Both users can view, update, or cancel sessions.

**Step 5: Analytics & Insights**

* Analytics dashboard uses Chart.js for statistics like:
  + Realtime Dashboard:- Includes Total users, active users, total sessions.
  + Business Metrics:- Calculates RFM, CLV and NPS Score using demo data
  + Referral analysis (planned)

**Step 6: Hosting & Access**

* Entire app deployed via Firebase Hosting.

**8) Business Concept / Strategy**

**Business Model Options:**

1. **Freemium Model:** Free for basic use; premium for verified tutors or unlimited sessions.
2. **Commission Model:** Platform takes a small cut per booked session.
3. **Subscription Model:** Monthly/annual plans for learners and tutors.
4. **Advertising Model:** Non-intrusive targeted ads for educational resources.

**9) Target Audience & Value Proposition**

| **Audience** | **Needs** | **Solution Provided** |
| --- | --- | --- |
| **Students (1–12 / College)** | Need quick, reliable help in subjects | Fast matching with qualified peers |
| **Tutors (Students / Graduates)** | Want to earn income or gain experience | Platform to find learners easily |
| **Schools / Institutions** | Need scalable tutoring system | White-label or enterprise version |

**Value Proposition:**  
Peer Tutor offers a lightweight, scalable, and intelligent matching system for peer-to-peer learning — no complex setup, instant deployment, and intuitive UX.

**10) Pricing Strategy / Product Flow**

**🧾 Pricing Strategy**

The **Peer Tutor** platform can adopt a **multi-tiered pricing model** combining freemium access with paid upgrades, depending on user needs.

| **Plan** | **Description / Features** | **Target User** | **Price (Example)** |
| --- | --- | --- | --- |
| **Free Plan** | • Access to basic tutor search & 3 sessions per month • View tutor ratings and profiles • Limited analytics and matching options | Students / Learners | **₹0** |
| **Tutor Pro Plan** | • Verified tutor badge • Priority listing in search results • Access to session performance analytics | Peer Tutors | ₹**499/month ₹5,499/year** |
| **Learner Plus Plan** | • Unlimited session bookings • Request preferred time slots • Ad-free experience | Learners / Parents | **₹349/month ₹3999/year** |

**💡 Add-Ons**

* **Commission-based earnings** — 10-15% fee per session booked through the platform.
* **Referral rewards** — bonus credits for users who bring in new learners/tutors.
* **Certification & verification fees** — optional identity or qualification verification for tutors.

**11) Customer Journey (Acquisition → Engagement → Retention → Loyalty)**

* **Acquisition:** Users discover Peer Tutor through social media, referrals, and college outreach, then onboard via quick Firebase signup and profile setup.
* **Engagement:** Learners book sessions with matched tutors, interact through scheduling and chat features, and tutors stay active via booking alerts and earnings summaries.
* **Retention:** Regular session bookings, reliable tutor performance, and positive learning outcomes encourage repeat usage and consistent platform engagement.
* **Loyalty:** Referral incentives, user satisfaction (NPS), and trust built through transparent tutor–learner interactions convert active users into long-term advocates.

**12) Customer Loyalty Strategy**

* Enhance retention via **personalized tutor recommendations**, and **loyalty discounts** for high-frequency learners.
* Strengthen tutor loyalty through transparent earnings, consistent scheduling, and recognition badges.

**13) Business Metrics**

**1. RFM Analysis — *Recency, Frequency, Monetary***  
I helps identify top learners and high-value tutors.

| **Metric** | **Meaning** | **Data Source** | **Calculation / Logic** | **Example Interpretation** |
| --- | --- | --- | --- | --- |
| **R – Recency** | How recently a learner booked or a tutor taught a session | sessions collection | R = today - lastSessionDate | Lower R = more active |
| **F – Frequency** | Number of sessions completed in a time period | sessions collection filtered by user ID | F = count(sessions) over last 90 days | Higher F = more loyal |
| **M – Monetary** | Total spend (learners) or total earnings (tutors) | Derived from session.price | M = sum(price) | Higher M = more valuable |

**RFM Scoring (1–5 scale)**

Each user can be assigned scores:

* **R:** Recent = 5 → active this week, 1 → inactive > 3 months
* **F:** Frequency = 5 → 10+ sessions, 1 → 1 session
* **M:** Monetary = 5 → top 20% earners/spenders, 1 → bottom 20%

**RFM Score = (R × 0.4) + (F × 0.3) + (M × 0.3)**  
→ You can rank users by RFM to identify **VIP tutors/learners**, **churn risks**, and **re-engagement targets**.

**2. CLV — *Customer Lifetime Value***

| **Metric** | **Description** | **Example Value (Learner)** |
| --- | --- | --- |
| Average Revenue per Session | How much the platform earns per booking | ₹150 |
| Sessions per Month | Avg. number of sessions booked per user | 3 |
| Average Retention | How long the learner stays active | 8 months |
| CAC | Average cost to acquire a learner | ₹200 |

CLV = (₹150×3×8)−₹200=₹3,400 per learner

**Tutor CLV (platform perspective)**

If tutors pay a small subscription (₹499/month) or platform takes 10% commission:

Example: ₹499× 12 = ₹5988 per tutor per year.

**3. NPS — *Net Promoter Score***

| **Response (1–10 scale)** | **Category** | **Action** |
| --- | --- | --- |
| 9–10 | Promoters | Highly satisfied users |
| 7–8 | Passives | Neutral users |
| 0–6 | Detractors | Unsatisfied users |

**Example:**

* 100 responses → 60 Promoters, 25 Passives, 15 Detractors
* NPS = (60% − 15%) = **+45 (Good)**

**In *PeerTutor* context (in future):**

* Collect via feedback modal or follow-up email after 3 completed sessions.
* High NPS among learners → good matching & session experience.
* High NPS among tutors → fair pay & transparent scheduling.

**14) Project Link / GitHub / Deployment**

| **Type** | **Link** |
| --- | --- |
| **GitHub Repository** | *https://github.com/Vrutti88/peer-tutor* |
| **Live Deployment (Firebase Hosting)** | *https://peer-tutor-c144b.web.app* |
| **Github Hosting** | *https://vrutti88.github.io/peer-tutor/public/index.html* |