File: server/storage.ts

Type: ts | Size: 5732 bytes

```
import {
  users,
  readings,
  dailyCards,
  type User,
  type InsertUser,
  type Reading,
  type InsertReading,
  type DailyCard,
  type InsertDailyCard
} from "@shared/schema";

export interface IStorage {
  getUser(id: number): Promise<User | undefined>;
  getUserByUsername(username: string): Promise<User | undefined>;
  getUsers(): Map<number, User>;
  createUser(user: InsertUser): Promise<User>;
  updateUser(id: number, userData: Partial<User>): Promise<User>;
  getReadingsByUserId(userId: number): Promise<Reading[]>;
  createReading(reading: InsertReading): Promise<Reading>;
  getDailyCardsByUserId(userId: number): Promise<DailyCard[]>;
  getDailyCardForToday(userId: number): Promise<DailyCard | undefined>;
  createDailyCard(dailyCard: InsertDailyCard): Promise<DailyCard>;
  updateStripeCustomerId(userId: number, customerId: string): Promise<User>;
  updateSubscriptionStatus(userId: number, status: string, expiryDate?: Date | null): Promise<User>;
  updateUserStripeInfo(userId: number, stripeInfo: { customerId: string, subscriptionId: string }): Promise<Us...
}

export class MemStorage implements IStorage {
  private users: Map<number, User>;
  private readings: Map<number, Reading>;
  private dailyCards: Map<number, DailyCard>;
  private userId: number;
  private readingId: number;
  private dailyCardId: number;

  constructor() {
    this.users = new Map();
    this.readings = new Map();
    this.dailyCards = new Map();
    this.userId = 1;
    this.readingId = 1;
    this.dailyCardId = 1;

    // Add a sample user for development
    this.createUser({
      username: "stella",
      password: "password123",
      displayName: "Stella",
      intention: "Find clarity in my relationships and attract authentic love"
    });
  }

  async getUser(id: number): Promise<User | undefined> {
    return this.users.get(id);
  }

  async getUserByUsername(username: string): Promise<User | undefined> {
    return Array.from(this.users.values()).find(
      (user) => user.username === username,
    );
  }

  getUsers(): Map<number, User> {
    return this.users;
  }

  async createUser(insertUser: InsertUser): Promise<User> {
    const id = this.userId++;
    const now = new Date();
```

```typescript
    const user: User = {
      ...insertUser,
      id,
      joinedAt: now,
      level: 1,
      levelProgress: 0,
      // Default subscription values
      isPremium: false,
      subscriptionStatus: 'free',
      subscriptionExpiry: null,
      stripeCustomerId: null,
      stripeSubscriptionId: null
    };
    this.users.set(id, user);
    return user;
  }

  async getReadingsByUserId(userId: number): Promise<Reading[]> {
    return Array.from(this.readings.values()).filter(
      (reading) => reading.userId === userId
    );
  }

  async createReading(insertReading: InsertReading): Promise<Reading> {
    const id = this.readingId++;
    const reading: Reading = {
      ...insertReading,
      id,
      date: new Date()
    };
    this.readings.set(id, reading);
    return reading;
  }

  async getDailyCardsByUserId(userId: number): Promise<DailyCard[]> {
    return Array.from(this.dailyCards.values())
      .filter((card) => card.userId === userId)
      .sort((a, b) => b.date.getTime() - a.date.getTime());
  }

  async getDailyCardForToday(userId: number): Promise<DailyCard | undefined> {
    const today = new Date();
    today.setHours(0, 0, 0, 0);

    return Array.from(this.dailyCards.values()).find(
      (card) => {
        const cardDate = new Date(card.date);
        cardDate.setHours(0, 0, 0, 0);
        return card.userId === userId && cardDate.getTime() === today.getTime();
      }
    );
  }

  async createDailyCard(insertDailyCard: InsertDailyCard): Promise<DailyCard> {
    const id = this.dailyCardId++;
    const dailyCard: DailyCard = {
      ...insertDailyCard,
      id,
      date: new Date()
    };
    this.dailyCards.set(id, dailyCard);
    return dailyCard;
  }

  async updateUser(id: number, userData: Partial<User>): Promise<User> {
    const user = await this.getUser(id);
    if (!user) {
      throw new Error(`User with id ${id} not found`);
    }

    const updatedUser = {
      ...user,
      ...userData
    };

    this.users.set(id, updatedUser);
    return updatedUser;
  }

  async updateStripeCustomerId(userId: number, customerId: string): Promise<User> {
    const user = await this.getUser(userId);
    if (!user) {
      throw new Error(`User with id ${userId} not found`);
    }

    const updatedUser = {
      ...user,
      stripeCustomerId: customerId
    };

    this.users.set(userId, updatedUser);
    return updatedUser;
  }

  async updateSubscriptionStatus(userId: number, status: string, expiryDate?: Date | null): Promise<User> {
    const user = await this.getUser(userId);
    if (!user) {
      throw new Error(`User with id ${userId} not found`);
    }

    const updatedUser = {
      ...user,
      subscriptionStatus: status,
      isPremium: status === 'active',
      subscriptionExpiry: expiryDate || null
    };

    this.users.set(userId, updatedUser);
    return updatedUser;
  }

  async updateUserStripeInfo(userId: number, stripeInfo: { customerId: string, subscriptionId: string }): Prom...
    const user = await this.getUser(userId);
    if (!user) {
      throw new Error(`User with id ${userId} not found`);
    }

    const updatedUser = {
      ...user,
      stripeCustomerId: stripeInfo.customerId,
      stripeSubscriptionId: stripeInfo.subscriptionId,
      isPremium: true,
      subscriptionStatus: 'active'
    };

    this.users.set(userId, updatedUser);
    return updatedUser;
  }
}

export const storage = new MemStorage();
```