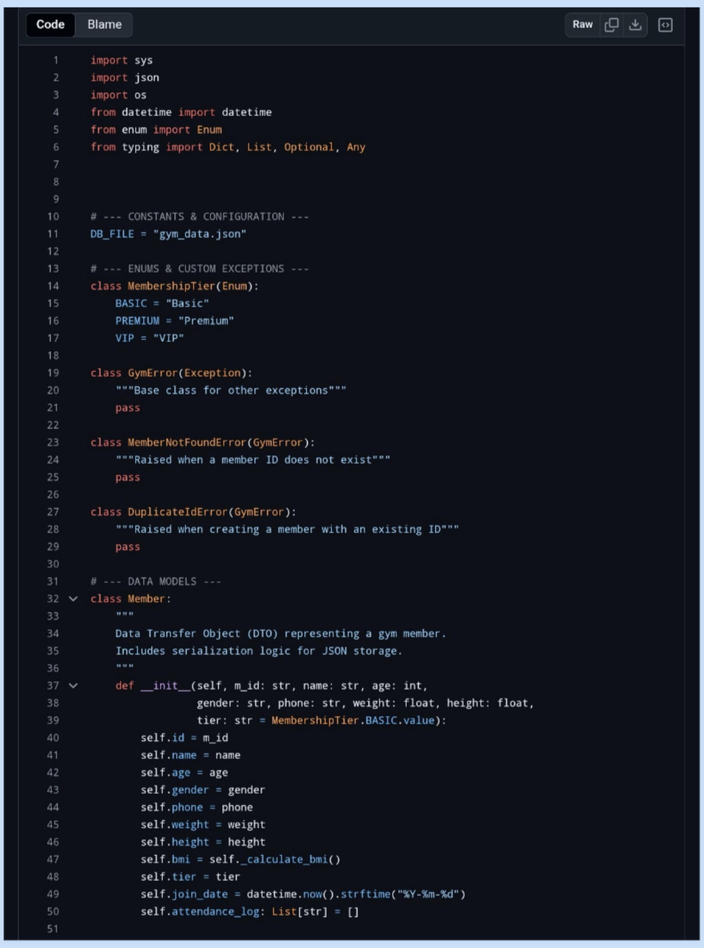
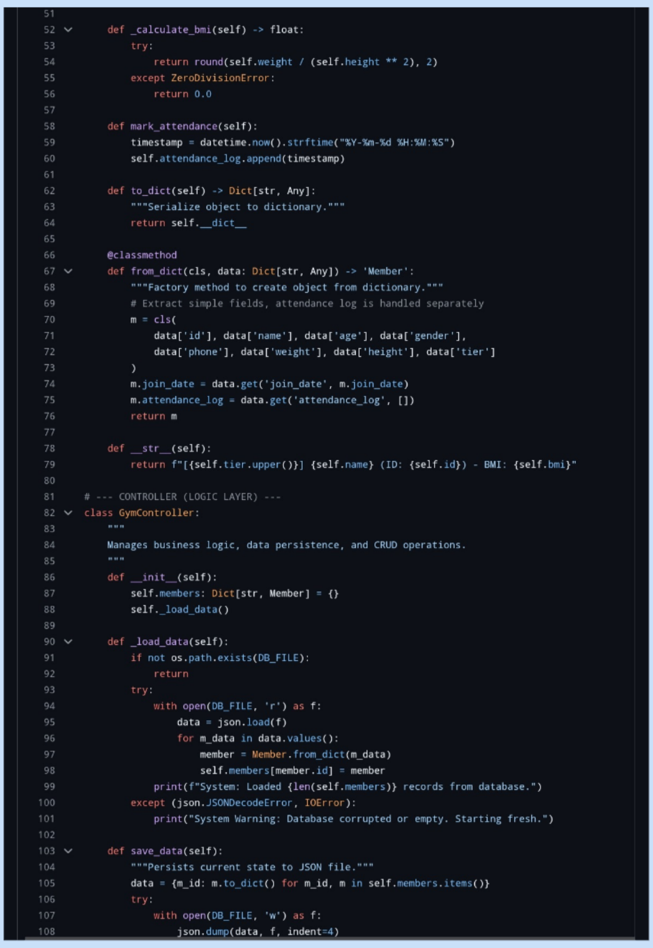
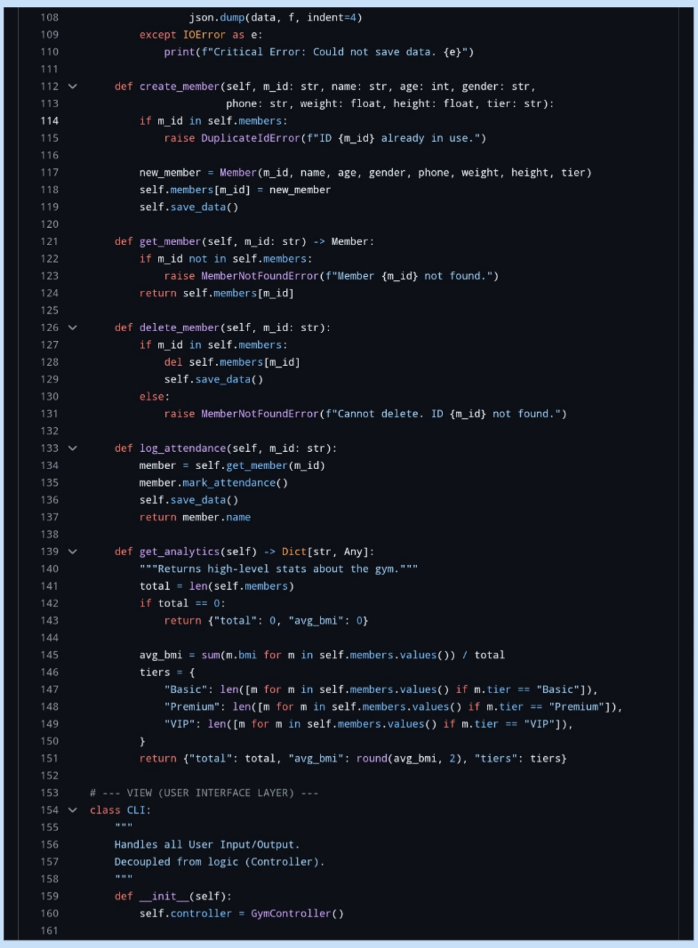


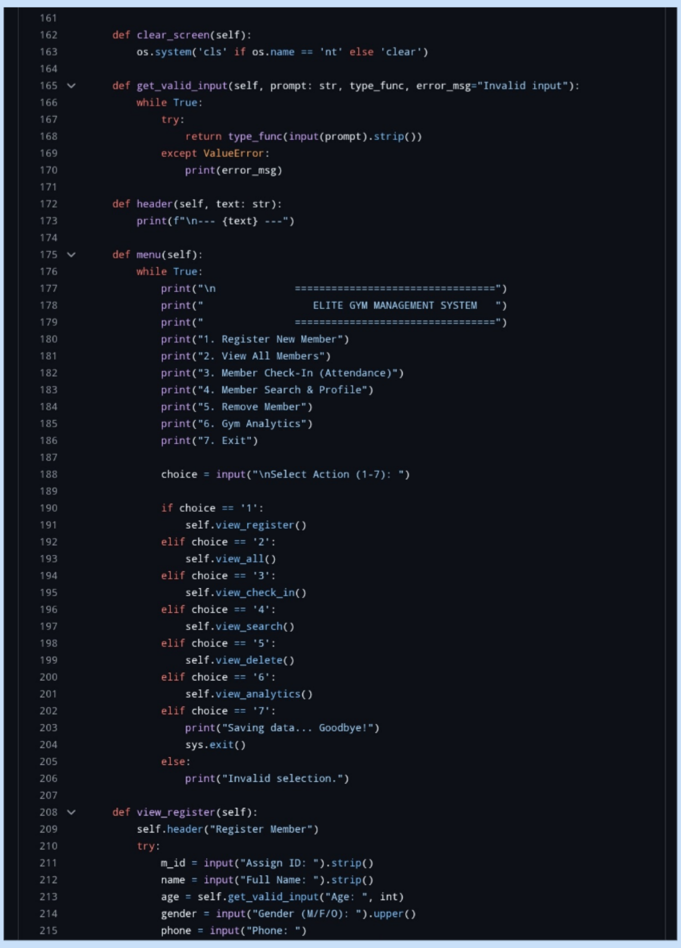
```python
import sys
import json
import os
from datetime import datetime
from enum import Enum
from typing import Dict, List, Optional, Any



# --- CONSTANTS & CONFIGURATION ---
DB_FILE = "gym_data.json"

# --- ENUMS & CUSTOM EXCEPTIONS ---
class MembershipTier(Enum):
    BASIC = "Basic"
    PREMIUM = "Premium"
    VIP = "VIP"

class GymError(Exception):
    """Base class for other exceptions"""
    pass

class MemberNotFoundError(GymError):
    """Raised when a member ID does not exist"""
    pass

class DuplicateIdError(GymError):
    """Raised when creating a member with an existing ID"""
    pass

# --- DATA MODELS ---
class Member:
    """
    Data Transfer Object (DTO) representing a gym member.
    Includes serialization logic for JSON storage.
    """
    def __init__(self, m_id: str, name: str, age: int,
                 gender: str, phone: str, weight: float, height: float,
                 tier: str = MembershipTier.BASIC.value):
        self.id = m_id
        self.name = name
        self.age = age
        self.gender = gender
        self.phone = phone
        self.weight = weight
        self.height = height
        self.bmi = self._calculate_bmi()
        self.tier = tier
        self.join_date = datetime.now().strftime("%Y-%m-%d")
        self.attendance_log: List[str] = []

```



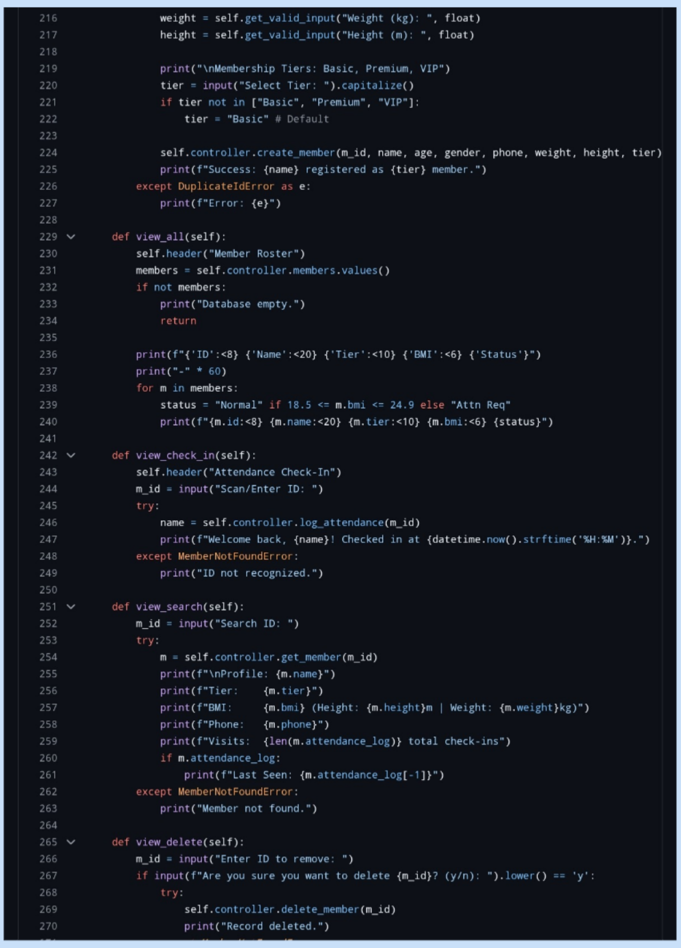
```python
    def _calculate_bmi(self) -> float:
        try:
            return round(self.weight / (self.height ** 2), 2)
        except ZeroDivisionError:
            return 0.0

    def mark_attendance(self):
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        self.attendance_log.append(timestamp)

    def to_dict(self) -> Dict[str, Any]:
        """Serialize object to dictionary."""
        return self.__dict__

    @classmethod
    def from_dict(cls, data: Dict[str, Any]) -> 'Member':
        """Factory method to create object from dictionary."""
        # Extract simple fields, attendance log is handled separately
        m = cls(
            data['id'], data['name'], data['age'], data['gender'],
            data['phone'], data['weight'], data['height'], data['tier']
        )
        m.join_date = data.get('join_date', m.join_date)
        m.attendance_log = data.get('attendance_log', [])
        return m

    def __str__(self):
        return f"[{self.tier.upper()}] {self.name} (ID: {self.id}) - BMI: {self.bmi}"

# --- CONTROLLER (LOGIC LAYER) ---
class GymController:
    """
    Manages business logic, data persistence, and CRUD operations.
    """
    def __init__(self):
        self.members: Dict[str, Member] = {}
        self._load_data()

    def _load_data(self):
        if not os.path.exists(DB_FILE):
            return
        try:
            with open(DB_FILE, 'r') as f:
                data = json.load(f)
                for m_data in data.values():
                    member = Member.from_dict(m_data)
                    self.members[member.id] = member
            print(f"System: Loaded {len(self.members)} records from database.")
        except (json.JSONDecodeError, IOError):
            print("System Warning: Database corrupted or empty. Starting fresh.")

    def save_data(self):
        """Persists current state to JSON file."""
        data = {m_id: m.to_dict() for m_id, m in self.members.items()}
        try:
            with open(DB_FILE, 'w') as f:
                json.dump(data, f, indent=4)
```



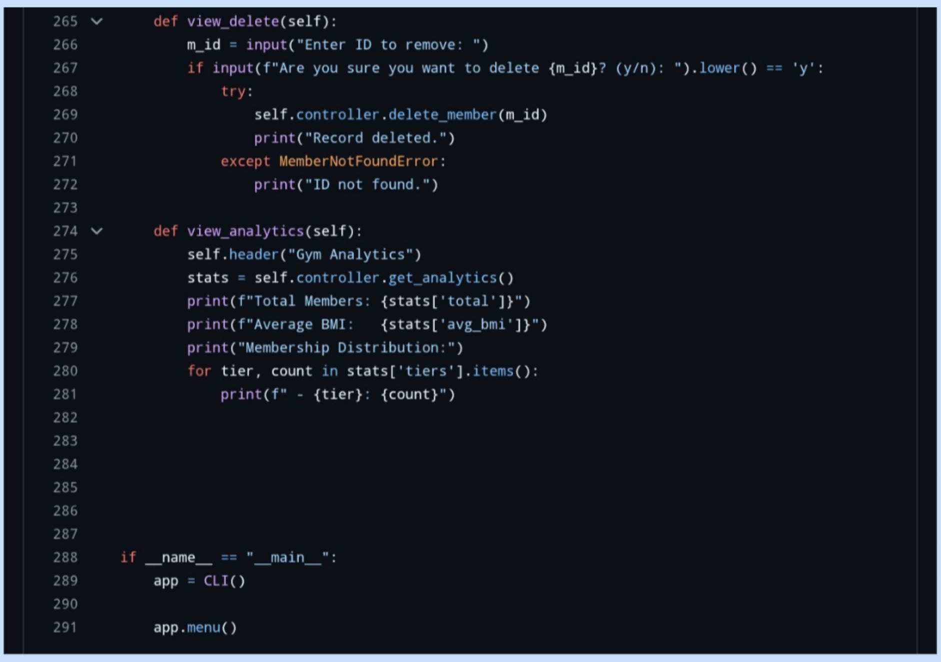
```python
        except IOError as e:
            print(f"Critical Error: Could not save data. {e}")

    def create_member(self, m_id: str, name: str, age: int, gender: str,
                      phone: str, weight: float, height: float, tier: str):
        if m_id in self.members:
            raise DuplicateIdError(f"ID {m_id} already in use.")

        new_member = Member(m_id, name, age, gender, phone, weight, height, tier)
        self.members[m_id] = new_member
        self.save_data()

    def get_member(self, m_id: str) -> Member:
        if m_id not in self.members:
            raise MemberNotFoundError(f"Member {m_id} not found.")
        return self.members[m_id]

    def delete_member(self, m_id: str):
        if m_id in self.members:
            del self.members[m_id]
            self.save_data()
        else:
            raise MemberNotFoundError(f"Cannot delete. ID {m_id} not found.")

    def log_attendance(self, m_id: str):
        member = self.get_member(m_id)
        member.mark_attendance()
        self.save_data()
        return member.name

    def get_analytics(self) -> Dict[str, Any]:
        """Returns high-level stats about the gym."""
        total = len(self.members)
        if total == 0:
            return {"total": 0, "avg_bmi": 0}

        avg_bmi = sum(m.bmi for m in self.members.values()) / total
        tiers = {
            "Basic": len([m for m in self.members.values() if m.tier == "Basic"]),
            "Premium": len([m for m in self.members.values() if m.tier == "Premium"]),
            "VIP": len([m for m in self.members.values() if m.tier == "VIP"]),
        }
        return {"total": total, "avg_bmi": round(avg_bmi, 2), "tiers": tiers}

# --- VIEW (USER INTERFACE LAYER) ---
class CLI:
    """
    Handles all User Input/Output.
    Decoupled from logic (Controller).
    """
    def __init__(self):
        self.controller = GymController()

```



```python

    def clear_screen(self):
        os.system('cls' if os.name == 'nt' else 'clear')

    def get_valid_input(self, prompt: str, type_func, error_msg="Invalid input"):
        while True:
            try:
                return type_func(input(prompt).strip())
            except ValueError:
                print(error_msg)

    def header(self, text: str):
        print(f"\n--- {text} ---")

    def menu(self):
        while True:
            print("\n             ==================================")
            print("                  ELITE GYM MANAGEMENT SYSTEM   ")
            print("               ==================================")
            print("1. Register New Member")
            print("2. View All Members")
            print("3. Member Check-In (Attendance)")
            print("4. Member Search & Profile")
            print("5. Remove Member")
            print("6. Gym Analytics")
            print("7. Exit")

            choice = input("\nSelect Action (1-7): ")

            if choice == '1':
                self.view_register()
            elif choice == '2':
                self.view_all()
            elif choice == '3':
                self.view_check_in()
            elif choice == '4':
                self.view_search()
            elif choice == '5':
                self.view_delete()
            elif choice == '6':
                self.view_analytics()
            elif choice == '7':
                print("Saving data... Goodbye!")
                sys.exit()
            else:
                print("Invalid selection.")

    def view_register(self):
        self.header("Register Member")
        try:
            m_id = input("Assign ID: ").strip()
            name = input("Full Name: ").strip()
            age = self.get_valid_input("Age: ", int)
            gender = input("Gender (M/F/O): ").upper()
            phone = input("Phone: ")
```



```python
            weight = self.get_valid_input("Weight (kg): ", float)
            height = self.get_valid_input("Height (m): ", float)

            print("\nMembership Tiers: Basic, Premium, VIP")
            tier = input("Select Tier: ").capitalize()
            if tier not in ["Basic", "Premium", "VIP"]:
                tier = "Basic" # Default

            self.controller.create_member(m_id, name, age, gender, phone, weight, height, tier)
            print(f"Success: {name} registered as {tier} member.")
        except DuplicateIdError as e:
            print(f"Error: {e}")

    def view_all(self):
        self.header("Member Roster")
        members = self.controller.members.values()
        if not members:
            print("Database empty.")
            return

        print(f"{'ID':<8} {'Name':<20} {'Tier':<10} {'BMI':<6} {'Status'}")
        print("-" * 60)
        for m in members:
            status = "Normal" if 18.5 <= m.bmi <= 24.9 else "Attn Req"
            print(f"{m.id:<8} {m.name:<20} {m.tier:<10} {m.bmi:<6} {status}")

    def view_check_in(self):
        self.header("Attendance Check-In")
        m_id = input("Scan/Enter ID: ")
        try:
            name = self.controller.log_attendance(m_id)
            print(f"Welcome back, {name}! Checked in at {datetime.now().strftime('%H:%M')}.")
        except MemberNotFoundError:
            print("ID not recognized.")

    def view_search(self):
        m_id = input("Search ID: ")
        try:
            m = self.controller.get_member(m_id)
            print(f"\nProfile: {m.name}")
            print(f"Tier:    {m.tier}")
            print(f"BMI:     {m.bmi} (Height: {m.height}m | Weight: {m.weight}kg)")
            print(f"Phone:   {m.phone}")
            print(f"Visits:  {len(m.attendance_log)} total check-ins")
            if m.attendance_log:
                print(f"Last Seen: {m.attendance_log[-1]}")
        except MemberNotFoundError:
            print("Member not found.")

    def view_delete(self):
        m_id = input("Enter ID to remove: ")
        if input(f"Are you sure you want to delete {m_id}? (y/n): ").lower() == 'y':
            try:
                self.controller.delete_member(m_id)
                print("Record deleted.")
```



```python
    def view_delete(self):
        m_id = input("Enter ID to remove: ")
        if input(f"Are you sure you want to delete {m_id}? (y/n): ").lower() == 'y':
            try:
                self.controller.delete_member(m_id)
                print("Record deleted.")
            except MemberNotFoundError:
                print("ID not found.")

    def view_analytics(self):
        self.header("Gym Analytics")
        stats = self.controller.get_analytics()
        print(f"Total Members: {stats['total']}")
        print(f"Average BMI:   {stats['avg_bmi']}")
        print("Membership Distribution:")
        for tier, count in stats['tiers'].items():
            print(f" - {tier}: {count}")






if __name__ == "__main__":
    app = CLI()

    app.menu()
```