FitDesert API — Detailed Markdown Documentation

Source: server.py (provided)
API prefix: /api (all documented routes are under /api unless noted)
Notes:

- Authentication is required for most routes. Where the code checks roles via helpers (e.g. get_current_user, get_current_gym_manager, get_current_head_admin, get_current_trainee), I list the *required roles* inferred from the helper name.
- Many endpoints accept Pydantic models (e.g. UserCreate, GymCreate, MemberCreate, PaymentCreate, WorkoutPlanCreate, DietPlanCreate, ProgressLogCreate, ChatRequest) — the file models.py isn't included here, so I list **inferred** fields from how each model is used. If you want fully exact field lists, paste models.py and I will update the docs.
- Times/dates returned in responses are usually timezone-aware UTC datetimes (code uses datetime.now(timezone.utc)), sometimes + IST_OFFSET when explicitly added.
- Responses are JSON unless an endpoint returns HTMLResponse.
- All endpoints exist under /api router except two root paths defined twice (/ at app root) — I document the router ones first.

Table of Contents

Auth routes

All auth routes are under /api/auth

POST /api/auth/register

Description: Register a new user and create a session.
Auth: Public (no token required)
Request body: UserCreate model — inferred fields used by code:

{

"email": "string",

"password": "string",

"name": "string",

"role": "string (expected enum - e.g. 'HEAD_ADMIN'|'GYM_MANAGER'|'TRAINEE'|'TRAINER')",

"phone": "string (optional)"

}

Behavior:

- Checks duplicate email; hashes password; inserts user document; creates a user_sessions entry with 7-day expiry.
  Success response (201/200):

{

"message": "User registered successfully",

"session_token": "session_user_...timestamp",

"user": {

"id": "user_...",

"email": "user@example.com",

"name": "Name",

"role": "TRAINEE"

}

}

Errors:

- 400 if email already registered.

POST /api/auth/login

Description: Login with email & password and create session.
Auth: Public
Request body: UserLogin model — inferred:

{

"email": "string",

"password": "string"

}

Behavior:

- Verifies credentials; returns a session token and basic user info; also stores session in DB.
  Success response:

{

"message": "Login successful",

"session_token": "session_user_...timestamp",

"user": {

"id": "user_...",

"email": "user@example.com",

"name": "User Name",

"role": "GYM_MANAGER",

"picture": "url or null"

}

}

Errors:

- 401 for invalid credentials.

GET /api/auth/session-data

Description: Get user data from Emergent OAuth session ID (external service).
Auth: Requires header X-Session-ID: <value> (the function expects this header).
Parameters (headers):

- X-Session-ID — required.
  Behavior:
- Calls https://demobackend.emergentagent.com/auth/v1/env/oauth/session-data using X-Session-ID. If user not in DB, creates a trainee user. Upserts session in user_sessions.
  Success response:

{

"id": "user_...",

"email": "email@example.com",

"name": "Name",

"picture": "url or null",

"session_token": "session-token-from-emergent"

}

Errors:

- 401 if the external response is not 200; 500 on other exceptions.

GET /api/auth/me

Description: Get current authenticated user info.
Auth: Requires active session (token) — the route calls get_current_user(request, db) which extracts user from cookie or authorization header.
Request: HTTP request with session token (cookie session_token or Authorization: Bearer <token>)
Response: returns user object (whatever get_current_user returns) — typically includes id, email, name, role, picture, etc.

POST /api/auth/logout

Description: Logout user and delete session.
Auth: Uses cookie or Authorization: Bearer <session_token>.
Behavior: deletes session doc from user_sessions.
Response:

{ "message": "Logged out successfully" }

POST /api/auth/change-password

Description: Change the currently authenticated user's password.
Auth: Authenticated user (get_current_user)
Query/body parameters: function signature uses (request: Request, old_password: str, new_password: str) — in FastAPI these are expected as **query parameters or form fields** (not JSON body). In practice call as POST form or with query: ?old_password=old&new_password=new.
Behavior: Verifies old password (except first-time users or must_change_password true), hashes new password and updates user.
Response:

{ "message": "Password changed successfully" }

Errors:

- 400 if old password incorrect.

Gym routes

All gym routes are under /api/gyms

POST /api/gyms/create

Description: Create a gym for a user (Head Admin only). Creates owner user if not present, creates gym document and generates inline QR code (base64 PNG).
Auth: Head Admin (get_current_head_admin)
Parameters: signature: (request, gym_data: GymCreate, owner_email: str, password: str) — so:

- gym_data in JSON body (GymCreate) — inferred fields used: name, address, city, state, phone, email
- owner_email and password as query params or form-params (function signature indicates they are expected as parameters alongside body). In FastAPI when you have a body model plus other parameters, those other parameters are treated as query parameters by default. So call like: POST /api/gyms/create?owner_email=owner@example.com&password=... with JSON body gym_data.
  Behavior: creates owner account if not exists, generates gym_id, creates QR base64, inserts gym doc (kyc_verified true, is_active false). Also returns QR.
  Success response:

{

"message": "Gym created successfully",

"gym_id": "gym_...",

"owner_email": "owner@example.com",

"qr_code": "<base64 PNG string>"

}

POST /api/gyms/register

Description: Register a new gym for the current Gym Manager (self-registration). Generates QR code and creates gym doc with kyc_verified=False.
Auth: Gym Manager (get_current_gym_manager)
Request body: GymCreate model — inferred fields: name, address, city, state, phone, email
Success response:

{

"message": "Gym registered successfully",

"gym_id": "gym_...",

"qr_code": "<base64 PNG>"

}

GET /api/gyms/my-gym

Description: Get gym of the current gym manager, including stats (total members, active members, today's attendance).
Auth: Gym Manager (get_current_gym_manager)
Response: gym document with an added stats field:

```
{
  "_id": "gym_...",
  "name": "...",
  "address": "...",
  ...
  "stats": {
    "total_members": 10,
    "active_members": 8,
    "today_attendance": 12
  }
}
```

Note: the code replaces _id to id for some responses — this endpoint mutates _id then returns gym object.

GET /api/gyms/all

Description: Get all gyms (Head Admin only). Enriches each gym with stats.
Auth: Head Admin (get_current_head_admin)
Response: array of gyms (up to 1000) each with id and stats:

```
[
  {
    "id": "gym_...",
    "name": "...",
    "stats": { "total_members": 10, "active_members": 8 }
  },
  ...
]
```

GET /api/gyms/{gym_id}

Description: Get gym details by id.
Auth: Any authenticated user (get_current_user)
Path params:

- gym_id - string (e.g., gym_12345)
  Response: gym document. 404 if not found.

PUT /api/gyms/{gym_id}

Description: Update gym details (Head Admin only).
Auth: Head Admin (get_current_head_admin)
Path params: gym_id
Request body: GymCreate — expected fields updated: name, address, city, state, phone, email
Response:

{ "message": "Gym updated successfully" }

Errors: 404 if gym not found.

PUT /api/gyms/{gym_id}/subscription

Description: Update gym subscription plan and expiry (Head Admin only).
Auth: Head Admin (get_current_head_admin)
Path params: gym_id
Query parameters: plan (string: "basic" | "pro" | "premium"), duration_days (int, default 30)
Behavior: Sets subscription_plan and subscription_expiry to now + duration_days.
Response:

{ "message": "Subscription updated successfully", "new_expiry": "2025-11-..." }

Errors: 400 for invalid plan; 404 if gym not found.

PUT /api/gyms/{gym_id}/status

Description: Activate or suspend gym (Head Admin only).
Auth: Head Admin (get_current_head_admin)
Path params: gym_id
Query/body param: is_active: bool (default not provided) — in function signature it's a parameter, thus expected as query param.
Response: { "message": "Gym activated successfully" } or suspended accordingly.

DELETE /api/gyms/{gym_id}

Description: Permanently delete gym and all related data (members, attendance, plans, payments, progress logs).
Auth: Head Admin (get_current_head_admin)
Path params: gym_id
Response: { "message": "Gym and all related data deleted successfully" }

POST /api/gyms/start-free-trial

Description: Activate 7-day free trial for the gym of the current manager.
Auth: Gym Manager (get_current_gym_manager)
Response:

{ "message": "Free trial started successfully", "trial_expires": "2025-11-11T..." }

Member & Trainer routes

All under /api/members and /api/trainers

POST /api/members

Description: Add a new member to the current manager's gym. May create a new user in users collection if email not found.
Auth: Gym Manager (get_current_gym_manager) — also enforces an active gym subscription via ensure_active_subscription(db, user.id).
Request body: MemberCreate — inferred fields used in code:

```
{
  "email": "member@example.com",
  "password": "string (for new user creation)",
  "name": "Full Name",
  "phone": "string",
  "is_trainer": false,          // boolean
  "photo": "url/base64/null",
  "membership_plan": "string",
  "plan_duration_months": 1,
  "goal": "string",
  "assigned_trainer_id": "trainer_member_id or null",
  "height": 170,
  "weight": 65,
  "age": 25
}
```

Behavior:

- If user with email exists, uses existing user id and verifies not already a member.
- If not exists, creates user (role = TRAINER or TRAINEE depending on is_trainer).
- Creates members document with membership expiry calculated from plan_duration_months (default 1 month).

Success response:

{ "message": "Member added successfully", "member_id": "member_...", "user_id": "user_..." }

Errors: 404 if no gym for manager; 400 if user is already a member.

GET /api/trainers

Description: Get all trainers for the current gym manager. Enriches each with user name & email.
Auth: Gym Manager (get_current_gym_manager)
Response: list of member documents (trainers) with user_name, user_email, and id fields.

GET /api/members

Description: Get all members for the current gym manager.
Auth: Gym Manager (get_current_gym_manager) and ensures active subscription.
Response: list of members (up to 1000) with user_name, user_email, and id.

GET /api/members/my-profile

Description: Get member profile for an authenticated trainee (current user).
Auth: Trainee (get_current_trainee)
Response: member object with fields and gym name & qr code:

```
{
  "id": "member_...",
  "user_id": "user_...",
  "gym_id": "gym_...",
  "gym_name": "Gym Name",
  "gym_qr": "<base64>",
  ...
}
```

Returns: { "message": "No membership found", "member": null } if no membership.

GET /api/members/{member_id}

Description: Get member details (accessible by gym managers, trainers, head admin).
Auth: get_current_user then role check: allowed roles are GYM_MANAGER, TRAINER, HEAD_ADMIN.

Path params: member_id
Response: member doc enriched with user_name, user_email, id. 404 if not found.

PUT /api/members/{member_id}

Description: Update member details (Gym Manager only).
Auth: Gym Manager (get_current_gym_manager) and ensure active subscription.
Path params: member_id
Request body: MemberCreate (same fields as create but used to update membership_plan, plan_duration_months, goal, assigned_trainer_id, height, weight, age).
Response: { "message": "Member updated successfully" } or 404 if not found.

PUT /api/members/{member_id}/assign-trainer

Description: Assign a trainer to a member (Gym Manager only).
Auth: Gym Manager (get_current_gym_manager) and ensure active subscription.
Path params: member_id
Query / body param: trainer_id (string) — function signature suggests query param.
Response: { "message": "Trainer assigned successfully" }

DELETE /api/members/{member_id}

Description: Delete a member from a gym and related data (attendance, plans, progress, payments) and delete user from users collection if they have no other memberships.
Auth: Gym Manager (get_current_gym_manager) and ensure active subscription.
Path params: member_id
Response: { "message": "Member deleted successfully" }
Notes: code deletes user if other_memberships == 0.

PUT /api/members/{member_id}/extend

Description: Extend a member's membership_expiry by extra_days.
Auth: Gym Manager (get_current_gym_manager) and ensure active subscription.
Path params: member_id
Query param: extra_days (int, default 30)
Response:

{ "message": "Membership extended", "new_expiry": "2025-11-..." }

Attendance routes

All under /api/attendance

POST /api/attendance/scan

Description: Mark attendance using QR scan data. This endpoint accepts a JSON body containing qr_code string extracted from QR. It automatically detects check-in vs check-out.

Auth: Authenticated user (get_current_user) — trainees only; trainers are blocked.
Request body JSON:

{ "qr_code": "fitdesert://gym/gym_12345/attendance" }

Behavior:

- Extracts gym_id from QR code via regex (expects gym_... in code).
- Verifies gym exists and member belongs to gym.
- If no attendance record for today: inserts check-in with check_in_time (UTC + IST_OFFSET in this code path).
- If existing record has check_out_time null: updates with check_out_time.
- If both present: returns 400 "Already checked in and out for today".

Success responses:

{ "message": "Checked in successfully", "type": "check_in" }

{ "message": "Checked out successfully", "type": "check_out" }

Errors: 400 if QR missing/invalid or membership inactive or already completed both check-in/out. 403 if user is trainer or not a member.

PUT /api/attendance/checkout

Description: Mark checkout time for a trainee by specifying gym_id (alternative to scan).
Auth: Authenticated user (get_current_user) — trainers blocked.
Query param / body param: gym_id (string)
Behavior: finds today's attendance record for member_id matching member_{user.id} (note: code uses "member_id": f"member_{user.id}" — this assumes member ids were created in that pattern; be careful) and updates check_out_time to now.
Success: { "message": "Checkout recorded successfully" }
Errors: 404 if check-in not found.

Important: This endpoint uses member_id constructed as member_{user.id} when searching for attendance; ensure your member documents follow that naming convention or modify the logic.

GET /api/attendance/my-history

Description: Get attendance history (latest 30) for the authenticated trainee.
Auth: Trainee (get_current_trainee)
Response: list of attendance records with id field (converted from _id), sorted descending by check_in_time.

GET /api/attendance/gym-stats

Description: Get attendance stats for gym manager. Supports optional ?date=YYYY-MM-DD query parameter to fetch stats for a specific date. Returns counts and lists for the selected date and the last week.
Auth: Gym Manager (get_current_gym_manager) and ensures active subscription.

Query param: date (optional; format YYYY-MM-DD)
Response:

{

"selected_date": "2025-11-04",

"today_count": 10,

"week_count": 72,

"today_records": [... attendance objects ...]

}

Errors: 400 for invalid date format.

Payment routes

All under /api/payments

POST /api/payments/create-order

Description: Create Razorpay payment order for a member. Inserts pending payment doc.
Auth: Authenticated user (get_current_user)
Request body: PaymentCreate — inferred fields:

{

"member_id": "member_...",

"amount": 500, // in INR (float or int)

"payment_type": "string enum" // code uses `.value` so it's likely an enum

}

Behavior:

- Verifies member exists.
- Creates razorpay order via razorpay_client.order.create.
- Inserts a payment document with status PENDING.

Success response:

{

"order_id": "order_razorpayid",

"amount": 50000, // amount in paise as returned by razorpay

"currency": "INR",

"key_id": "RAZORPAY_KEY_ID",

"payment_id": "pay_..."

}

Errors: 404 if member not found.

GET /api/payments/checkout (returns HTML)

Description: Creates a Razorpay order and serves an auto-opening Razorpay checkout HTML page. Intended for usage via WebBrowser.openBrowserAsync from Expo frontends.
Auth: Public (not protected in code)
Query params: plan (string), amount (int)
Behavior:

- Creates order, returns HTML page with embedded Razorpay Checkout JS which calls /api/payments/verify on success.
  Response: HTMLResponse content (the checkout page).

POST /api/payments/verify

Description: Verify payment webhook/response from Razorpay and update the gym's subscription (finds latest gym as fallback) and insert payment record with status SUCCESS.
Auth: Public (the checkout HTML posts here; no signature check other than razorpay_client.utility.verify_payment_signature)
Request body: JSON with:

{

"razorpay_payment_id": "pay_...",

"razorpay_order_id": "order_...",

"razorpay_signature": "signature",

"plan": "starter|pro|premium",

"amount": "500"

}

Behavior:

- Verifies signature with Razorpay util.
- Finds a gym (current code uses gym = await db.gyms.find_one(sort=[("registration_date", -1)]) — latest gym) and updates its subscription plan and expiry (30 days).
- Inserts a success payment doc.
  Response: { "message": "Payment verified successfully and subscription updated" }
  Errors: 400 for invalid signature; 404 if no gym found to apply payment.

Important: Current implementation uses the latest registered gym to apply subscription — for production you should link the payment to the correct gym (e.g., include gym_id in the payment payload or get from session).

GET /api/payments/gym/all

Description: Get all payments for gym manager (no subscription enforcement).
Auth: Gym Manager (get_current_gym_manager)
Response: payments array for the gym (converted _id -> id).

GET /api/payments/gym-payments

Description: Get successful payments for the gym manager (enforces active subscription).
Auth: Gym Manager (get_current_gym_manager) and ensure active subscription.
Response: recent payments (up to 100) for the gym with member_name attached where possible.

GET /api/payments/my-payments

Description: Get payment history for the current trainee.
Auth: Trainee (get_current_trainee)
Response: list of payments for the member with id fields.

Plan routes (Workout & Diet)

All under /api/plans or /api/gyms/start-free-trial

POST /api/plans/workout

Description: Create a workout plan for a member (Gym Manager only).
Auth: Gym Manager (get_current_gym_manager) & ensures active subscription.
Request body: WorkoutPlanCreate — inferred fields:

```
{
  "member_id": "member_...",
  "plan_name": "string",
  "workout_days": [
    { /* day model e.g., day name, exercises — code uses `.dict()` */ }
  ]
}
```

Behavior:

- Verifies member belongs to the gym.

- Inserts workout_plans doc with trainer_id set to user.id.
  Success response: { "message": "Workout plan created successfully", "plan_id": "workout_..." }

GET /api/plans/workout/my-plan

Description: Get workout plan for the current trainee.
Auth: Trainee (get_current_trainee)
Behavior: finds member by user_id, fetches plan by member_id.
Response: plan object (or { message: "No workout plan found", plan: null }).

POST /api/plans/diet

Description: Create diet plan for a member (Gym Manager only).
Auth: Gym Manager & ensure active subscription.
Request body: DietPlanCreate — inferred fields:

```
{

  "member_id": "member_...",

  "plan_name": "string",

  "daily_meals": [ { /* meal model */ } ],

  "total_calories": 2000

}
```

Success response: { "message": "Diet plan created successfully", "plan_id": "diet_..." }

GET /api/plans/diet/my-plan

Description: Get diet plan for current trainee.
Auth: Trainee (get_current_trainee)
Response: diet plan object (or No diet plan found message).

Progress tracking routes

POST /api/progress

Description: Log progress entry for current trainee (weight, body fat, measurements, photos, notes).
Auth: Trainee (get_current_trainee)
Request body: ProgressLogCreate — inferred fields:

```
{

  "weight": 68.5,

  "body_fat_percentage": 15.2,
```

"measurements": { "waist": 30, "chest": 38, ... },

"photos": ["base64 or url strings"],

"notes": "optional text"

}

Success response:

{ "message": "Progress logged successfully", "progress_id": "prog_..." }

Errors: 404 if no membership found.

GET /api/progress/my-history

Description: Get progress logs for current trainee (up to 50).
Auth: Trainee (get_current_trainee)
Response: list of progress log objects with id fields.

AI Assistant routes

All under /api/ai

POST /api/ai/chat

Description: Send a message to the AI fitness assistant (wraps GPTChat utility). Saves both user and assistant messages to chat_messages.
Auth: Authenticated user (get_current_user)
Request body: ChatRequest — inferred:

{ "message": "string" }

Behavior: calls GPTChat().send_message(message) (async) and stores entries in DB.
Success response:

{ "response": "AI generated reply text", "timestamp": "2025-11-04T..." }

Errors: 500 on AI service error.

GET /api/ai/chat-history

Description: Get chat history for the authenticated user (up to 100 messages).
Auth: Authenticated user (get_current_user)
Response: list of messages sorted ascending by timestamp, each with id field.

Root & health

GET /api/

Description: API root for the router.
Auth: None
Response:

{

"message": "FitDesert API",

"version": "1.0.0",

"status": "running"

}

GET /api/health

Description: Health check endpoint.
Auth: None
Response:

{ "status": "healthy" }

GET / (app root, duplicated)

Description: Another root endpoint at app-level defined near bottom.
Response: { "status": "running" }

Example usage / samples

Login & use token

1. POST /api/auth/login with JSON body { "email": "...", "password": "..." }
2. Response contains session_token. For subsequent requests include either:
   - Cookie session_token=<token> or
   - Header Authorization: Bearer <session_token>

Create Gym (Head Admin)
POST /api/gyms/create?owner_email=owner@example.com&password=ownerpass with JSON gym body:

{

"name": "Desert Gym",

"address": "Street 123",

"city": "City",

"state": "State",

"phone": "9999999999",

"email": "owner@example.com"

}

Mark attendance (QR scan)
POST /api/attendance/scan with JSON:

{ "qr_code": "fitdesert://gym/gym_1700000000.123/attendance" }

Start Razorpay checkout (web)
Open: GET /api/payments/checkout?plan=starter&amount=500 (returns HTML page that auto-opens the Razorpay checkout and posts result to /api/payments/verify).

Errors, edge-cases & important implementation notes

1. **Model definitions missing:** The code depends heavily on Pydantic models (UserCreate, GymCreate, MemberCreate, etc.). For exact field lists, include models.py. I inferred common fields from usages.
2. **Parameter locations:** Some function signatures accept non-body parameters together with a Pydantic body. In FastAPI, non-body parameters become **query** parameters by default. Example: create_gym_by_admin(request, gym_data: GymCreate, owner_email: str, password: str) — so owner_email and password are expected as query parameters. Document callers accordingly.
3. **Session handling & auth helpers:** The code uses helper functions from auth_utils like get_current_user, get_current_gym_manager, etc. These determine how to pass tokens (cookie vs Authorization header). Keep consistent across frontends.
4. **Payment -> gym mapping:** POST /api/payments/verify currently finds the latest registered gym and applies the subscription to it. **This is a security & correctness issue** — you should pass gym_id (or manager context) when creating/verifying payments to ensure the payment maps to the correct gym.
5. **Attendance scanning regex:** Attendance scan uses regex re.search(r"gym_(\d+(?:\.\d+)?)", qr_code) to extract numeric portion and then creates gym_{group}. If QR format changes, scanning will break. Consider storing gym_id explicitly as gym_<timestamp> in QR payload.
6. **Member id conventions:** Several places construct ids like member_{timestamp}; elsewhere some code searches by member_{user.id} — ensure id generation & lookup conventions match.
7. **Timezones:** The code stores datetimes in UTC datetime.now(timezone.utc) and sometimes adds IST_OFFSET for display. Decide on a single canonical storage and conversion pattern.
8. **Rate limits & security:** No rate limiting, CSRF mitigations, or strict CORS policy are present (CORS is wide open allow_origins=["*"]). For production, restrict CORS, enable CSRF protections or token-based headers, validate inputs thoroughly.

Files / Collections used by the API

- db.users
- db.user_sessions

- db.gyms
- db.members
- db.attendance
- db.workout_plans
- db.diet_plans
- db.progress_logs
- db.payments
- db.chat_messages

Recommended small follow-ups I can do for you (pick any)

- Generate an **OpenAPI (YAML/JSON)** spec derived from these routes (useful for Swagger / Postman).
- Produce a **compact README** (one-line per endpoint) for quick reference.
- Extract **exact model schemas** if you upload models.py — I'll replace "inferred" fields with exact model fields and types.
- Create **Postman collection** (JSON) or **curl examples** for key flows (register → login → create gym → add member → scan attendance → payment).

If you'd like, I'll now:

- Convert this into a single .md file and provide it for download, or
- Generate the OpenAPI YAML/JSON version from this documentation.

Which would you like next?