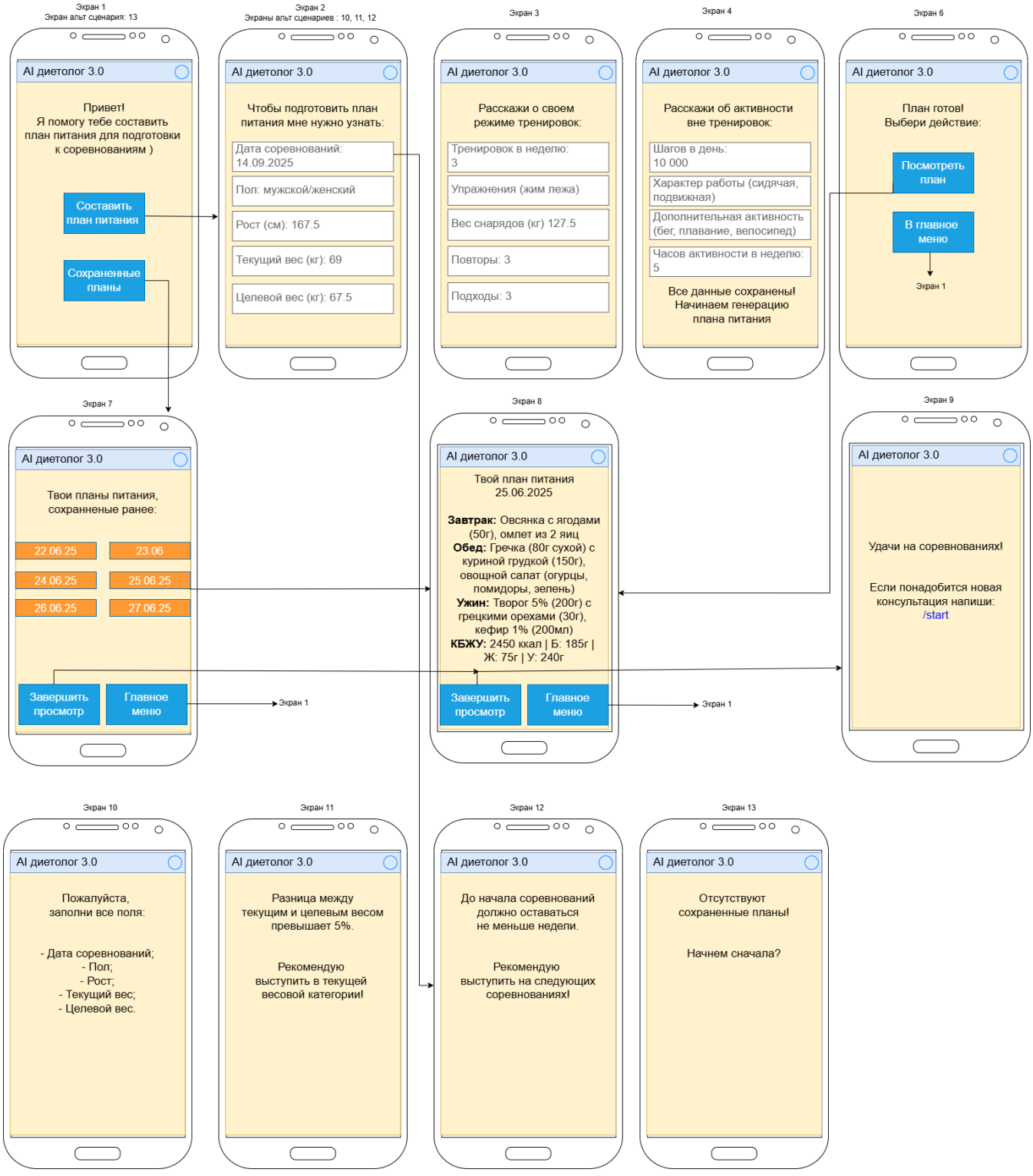
AI-диетолог 3.0

1. User story

Как спортсмен, я хочу получить от ИИ агента ежедневный план питания, чтобы подготовиться к выступлению на соревнованиях в требуемом весе.

2. Макеты



3. Use case

Use Case: составить план питания
Идентификатор: UC-01

Главный актор: спортсмен, готовящийся к соревнованиям
Вспомогательный актор: ИИ агент (диетолог)

Предусловия: спортсмен авторизован в системе

Ограничения: составление плана питания возможно не позднее недели до начала соревнований, текущий вес спортсмена отличается от требуемого веса не более чем на 5%.

Триггер: спортсмен заходит на платформу и выбирает опцию «Составить план питания»

Основной сценарий:

1. Система приветствует спортсмена и предлагает ему ввести дату соревнований, указать пол, рост, текущий вес спортсмена и требуемый вес.
2. Спортсмен вводит требуемые параметры.
3. ИИ агент начинает задавать вопросы в формате интервью, охватывающие следующие блоки:

- режим тренировок (количество тренировок в неделю, основные упражнения, вес снарядов, кг, повторы и подходы);
- физическая активность помимо тренировок (количество шагов в день, характер работы, дополнительная активность, количество часов активности в неделю).

4. Спортсмен отвечает на вопросы каждого из блоков в текстовом формате.
5. ИИ агент анализирует ответы и в завершение интервью:

Генерирует план питания спортсмена на день, следующий за днем запроса. В сгенерированном плане питания указаны приемы пищи (завтрак, обед, ужин), рецепты блюд с граммовкой продуктов на каждый из приемов пищи и общий КБЖУ для всех приемов пищи, для достижения требуемого веса спортсменом до начала соревнований с учетом его индивидуальных особенностей.

Альтернативный сценарий:

2a. Спортсмен не вводит требуемые для начала работы параметры.
3a. Система предлагает ввести параметры ещё раз.
Без введения каждого из параметров генерация плана питания не начинается.

Альтернативный сценарий:

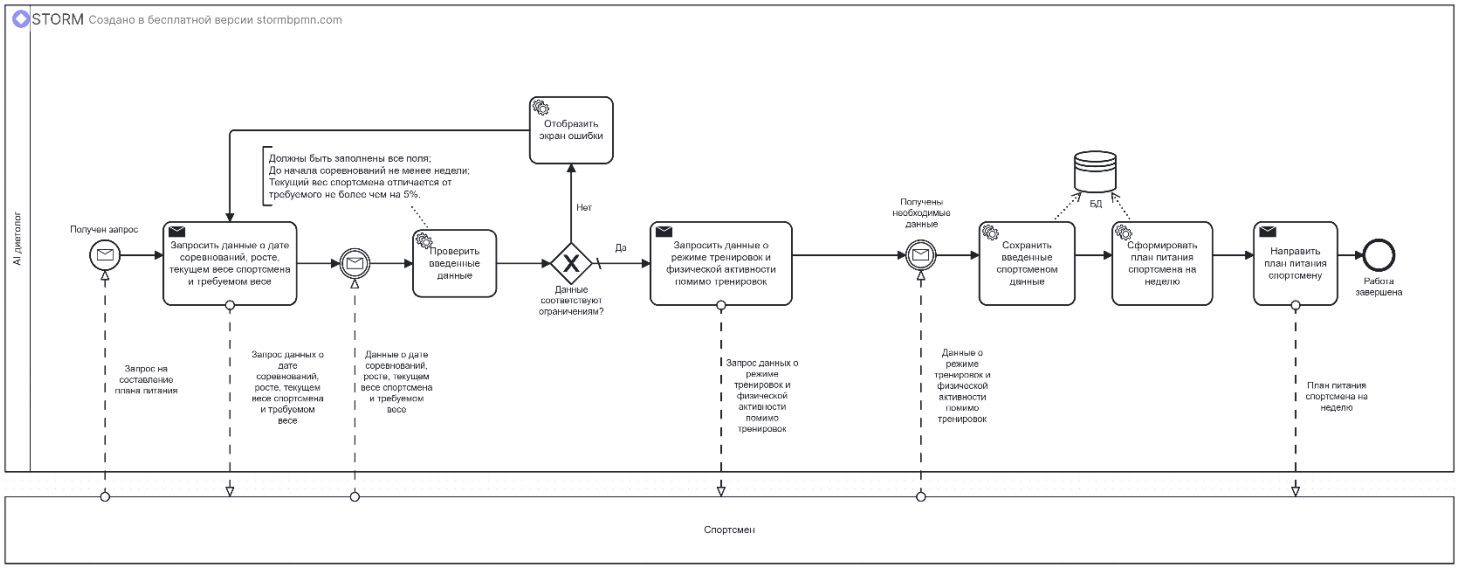
2b. Спортсмен вводит дату соревнований, до которых остается менее недели.
3b. Система сообщает спортсмену, что до начала соревнований должно оставаться не менее недели и предлагает выступить на следующих соревнованиях или ввести новую дату соревнований.
Без введения корректной даты соревнований генерация плана питания не начинается.

Исключительный сценарий:

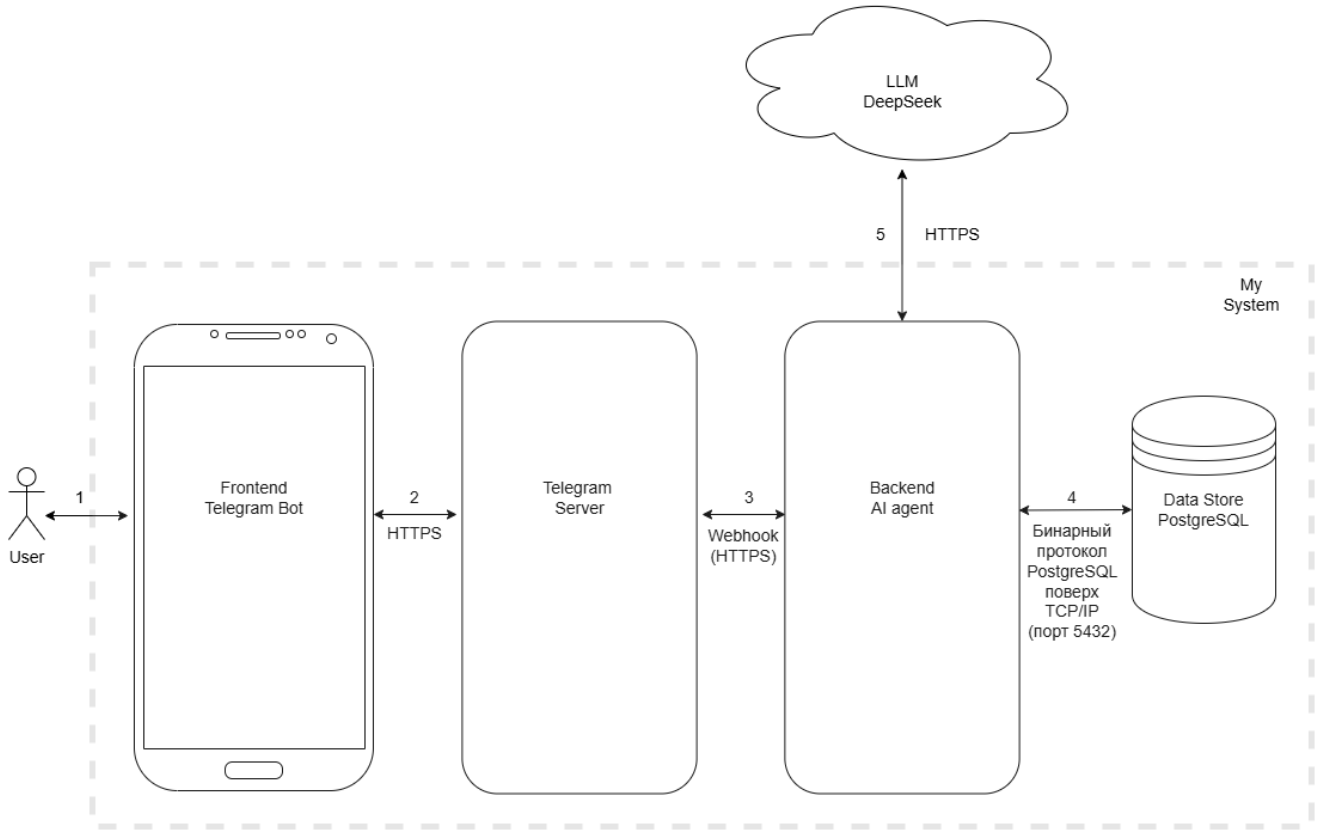
2b. Спортсмен вводит требуемый вес, отличающийся от текущего веса более чем на 5%.
3c Система предлагает выступить спортсмену в текущем весе или ввести новый вес.
Без введения корректного требуемого веса генерация плана питания не начинается.

Гарантии успеха: спортсмен получает план питания на день и может использовать его для подготовки к выступлению на соревнованиях в требуемом весе.

4. BPMN



5. Архитектура



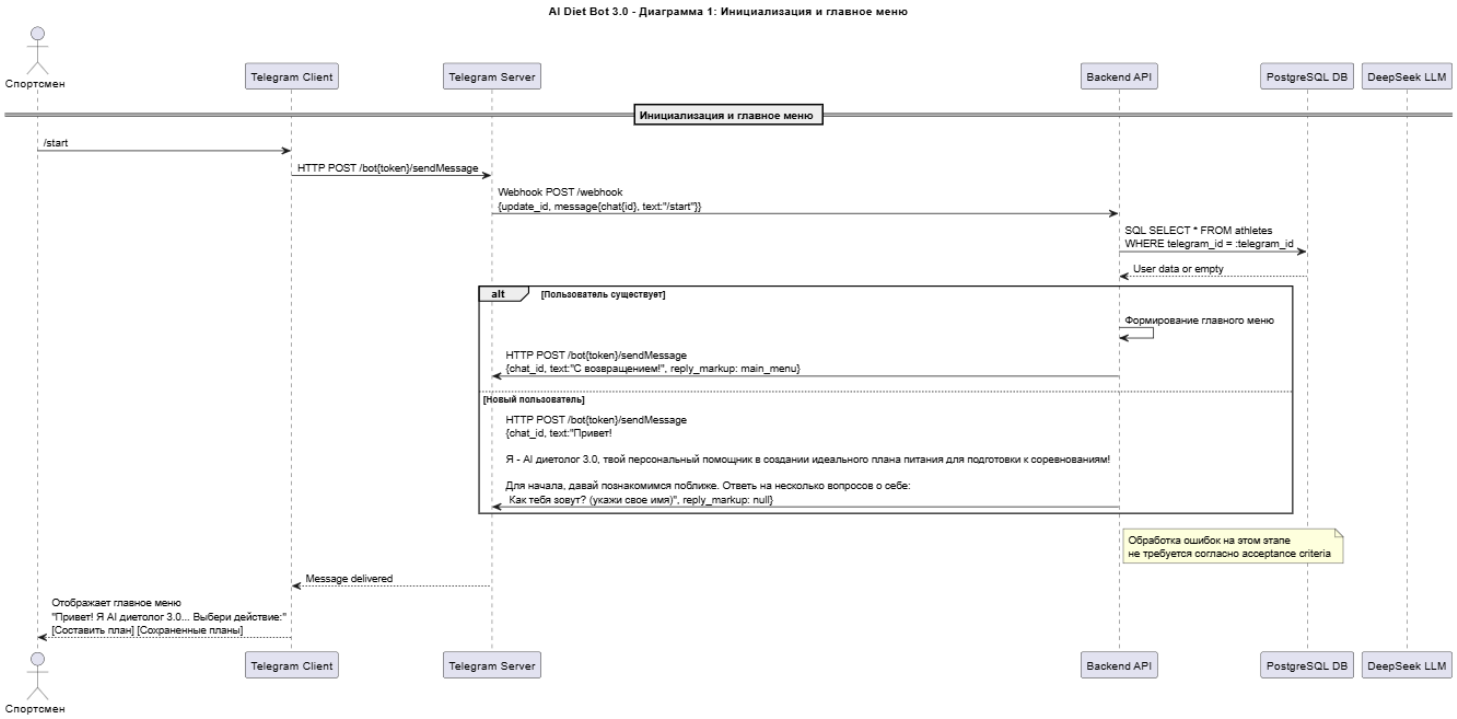
6. Диаграммы последовательностей

Полный код PlantUML для диаграммы последовательности взаимодействия спортсмена с AI диетологом 3.0 доступен по ссылке:
https://drive.google.com/file/d/1vca7ScqPMobpH3d896uoElg0530IyEU-/view?usp=sharing

1. Инициализация и главное меню

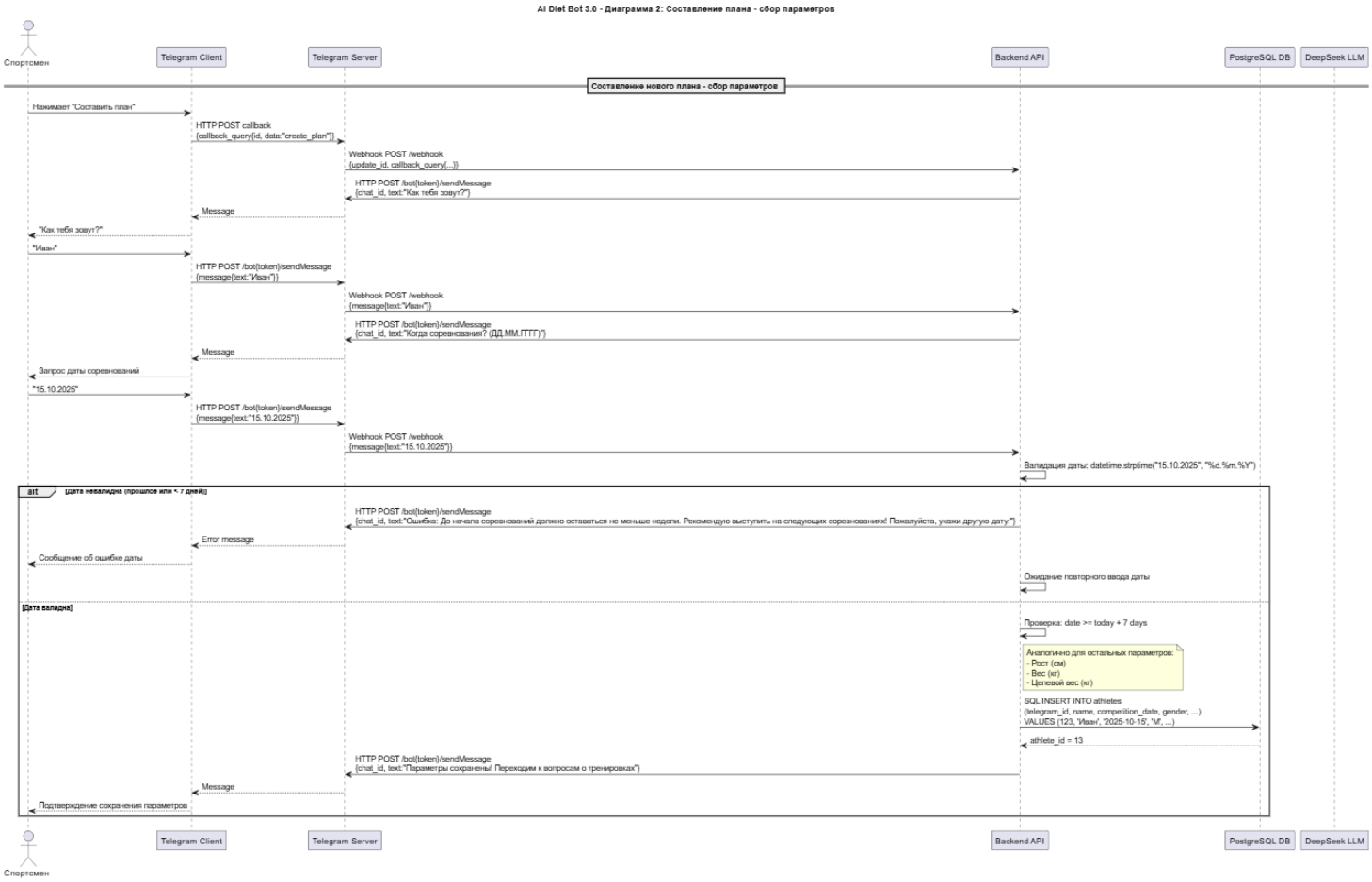
- Пользователь запускает бота командой **/start** через Telegram клиент
- Telegram клиент отправляет HTTP POST запрос на сервер Telegram: **/bot{token}/sendMessage**
- Сервер Telegram через webhook передает запрос в Backend: POST **/webhook** с данными **{update_id, message{chat{id}, text:"/start"}}**
- Backend выполняет SQL запрос к базе данных PostgreSQL: **SELECT * FROM athletes WHERE telegram_id = :telegram_id**
- Если пользователь существует в базе данных:
  • Backend формирует главное меню
  • Backend отправляет HTTP POST запрос на сервер Telegram: **/bot{token}/sendMessage с текстом "С возвращением!" и reply_markup: main_menu**
- Если пользователь новый:
  • Backend отправляет HTTP POST запрос на сервер Telegram: **/bot{token}/sendMessage с текстом "Привет! Я - AI диетолог 3.0, твой персональный помощник... Как тебя зовут?" и reply_markup: null**
- Сервер Telegram доставляет сообщение пользователю
- Пользователь получает главное меню с выбором действий:

"Привет! Я AI диетолог 3.0 и помогу тебе составить план питания для подготовки к соревнованиям! Выбери действие: [Составить план] [Сохраненные планы]"



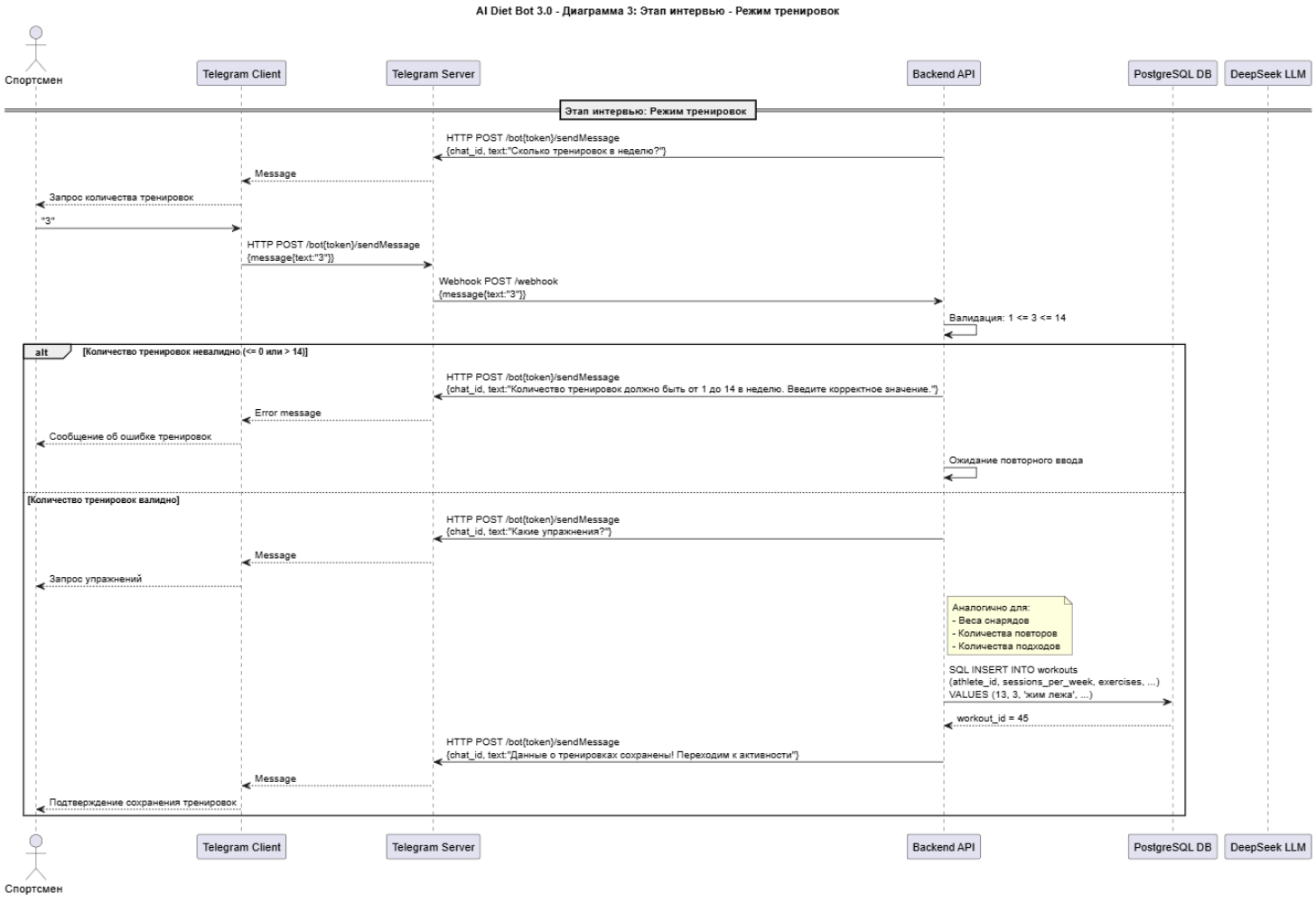
2. Составление нового плана - сбор параметров

- Пользователь нажимает **inline-кнопку [Составить план]** в Telegram клиенте
- Telegram клиент отправляет HTTP POST **callback** запрос на сервер Telegram: **{callback_query{id, data:"create_plan"}}**
- Сервер Telegram через webhook передает запрос в Backend: POST **/webhook с данными callback_query**
- Backend отправляет HTTP POST запрос на сервер Telegram: **/bot{token}/sendMessage с текстом "Как тебя зовут?"**
- Пользователь вводит ответ (например, "Иван")
- Telegram клиент отправляет HTTP POST запрос на сервер Telegram: **/bot{token}/sendMessage с текстом ответа**
- Сервер Telegram через webhook передает ответ в Backend
- Backend отправляет следующий вопрос: **"Когда соревнования? (ДД.ММ.ГГГГ)"**
- Пользователь вводит дату (например, "15.10.2025")
- Backend выполняет валидацию даты: **проверка формата ДД.ММ.ГГГГ и что дата >= сегодня + 7 дней**
- Если дата невалидна (в прошлом или менее 7 дней до соревнований):
  • Backend отправляет сообщение об ошибке: **"Ошибка: До начала соревнований должно оставаться не меньше недели..."**
  • Backend ожидает повторного ввода даты
- Если дата валидна:
  • Backend продолжает запрос остальных параметров: **рост (см), вес (кг), целевой вес (кг)**
  • Backend выполняет SQL INSERT запрос в таблицу athletes: **INSERT INTO athletes (telegram_id, name, competition_date, gender, ...) VALUES (...)**
  • Backend отправляет подтверждение: **"Параметры сохранены! Переходим к вопросам о тренировках"**



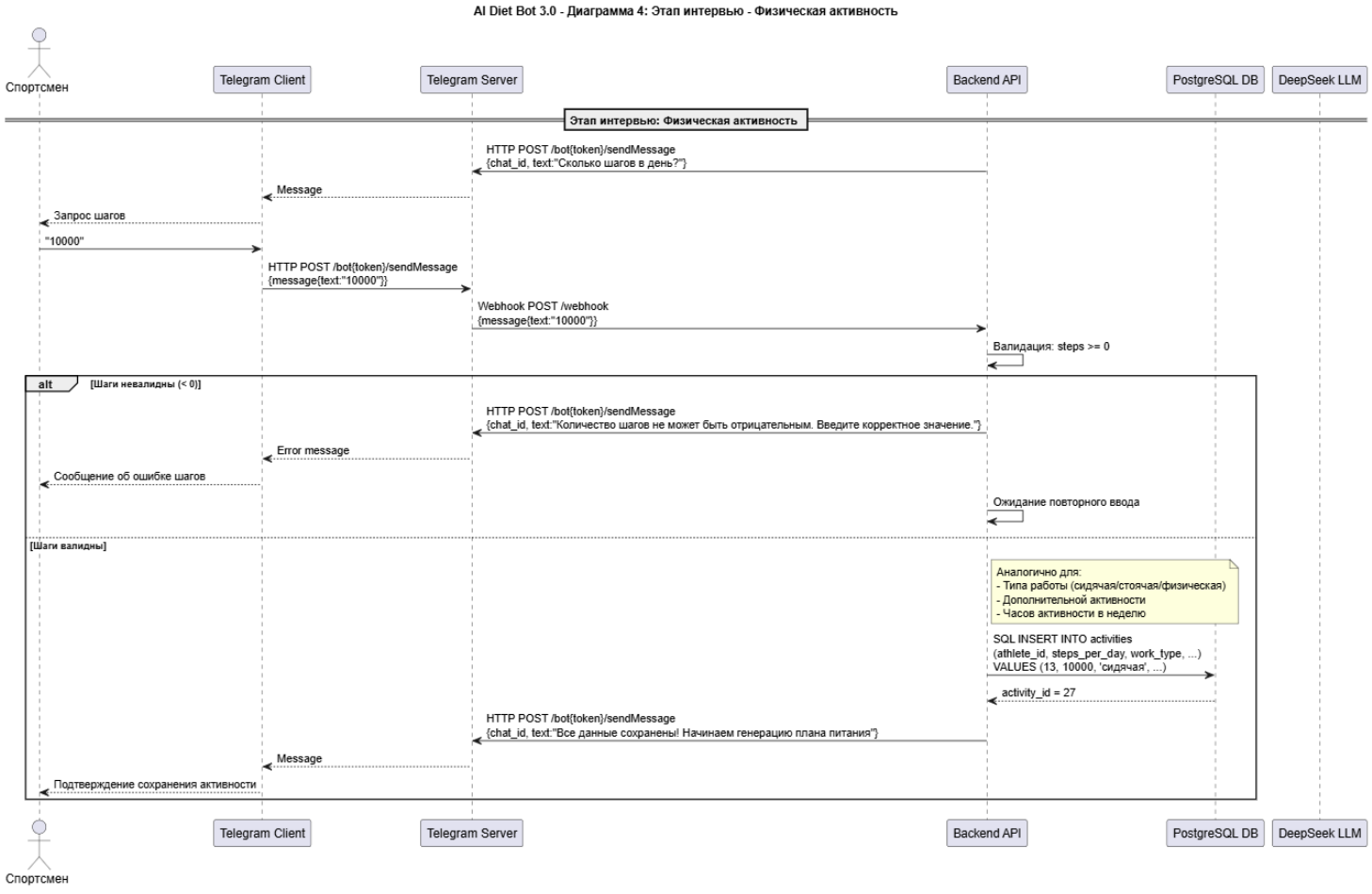
3. Этап интервью: Режим тренировок

- Backend отправляет HTTP POST запрос на сервер Telegram: **/bot{token}/sendMessage с текстом "Сколько тренировок в неделю?"**
- Пользователь вводит ответ (например, "3")
- Backend выполняет валидацию: **проверка что 1 <= количество тренировок <= 14**
- Если количество тренировок невалидно (<= 0 или > 14):
  • Backend отправляет сообщение об ошибке: **"Количество тренировок должно быть от 1 до 14 в неделю..."**
  • Backend ожидает повторного ввода
- Если количество тренировок валидно:
  • Backend продолжает запрос остальных параметров**: упражнения, вес снарядов, количество повторов, количество подходов**
  • Backend выполняет **SQL INSERT запрос в таблицу workouts: INSERT INTO workouts (athlete_id, sessions_per_week, exercises, ...) VALUES (...)**
  • Backend отправляет подтверждение: "Данные о тренировках сохранены! Переходим к активности"



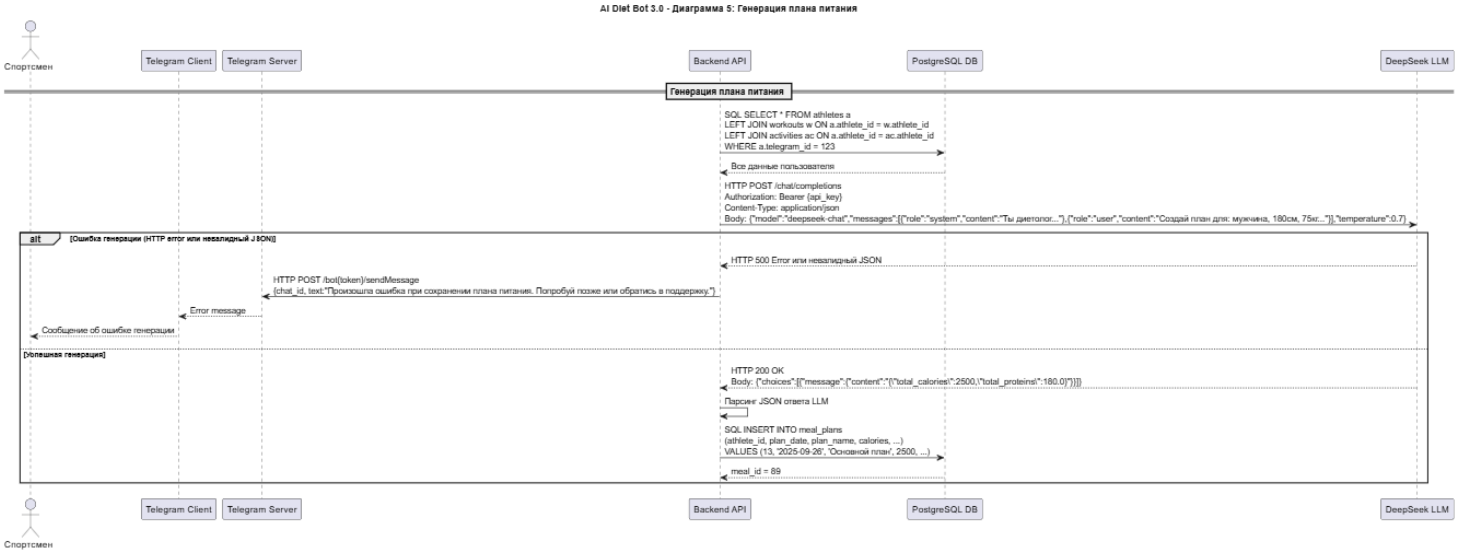
4. Этап интервью: Физическая активность

- Backend отправляет HTTP POST запрос на сервер Telegram**: /bot{token}/sendMessage с текстом "Сколько шагов в день?"**
- Пользователь вводит ответ (например, "10000")
- Backend выполняет валидацию: **проверка что steps >= 0**
- Если шаги невалидны (< 0):
  • Backend отправляет сообщение об ошибке: **"Количество шагов не может быть отрицательным..."**
  • Backend ожидает повторного ввода
- Если шаги валидны:
  • Backend продолжает запрос остальных параметров: **тип работы (сидячая/стоячая/физическая), дополнительная активность, часов активности в неделю**
  • Backend выполняет SQL **INSERT запрос в таблицу activities: INSERT INTO activities (athlete_id, steps_per_day, work_type, ...) VALUES (...)**
  • Backend отправляет подтверждение: **"Все данные сохранены! Начинаем генерацию плана питания"**



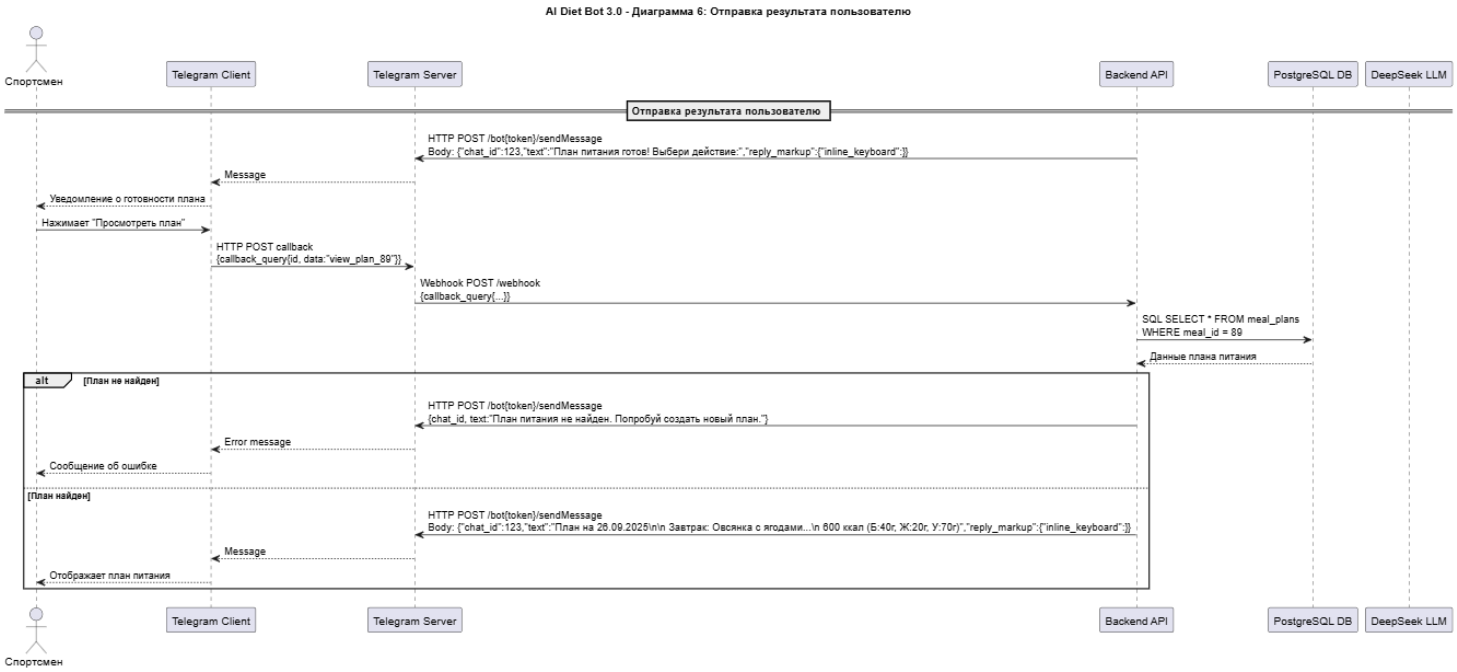
5. Генерация плана питания

- Backend выполняет SQL SELECT запрос к базе данных: **SELECT * FROM athletes a LEFT JOIN workouts w ON a.athlete_id = w.athlete_id LEFT JOIN activities ac ON a.athlete_id = ac.athlete_id WHERE a.telegram_id = :telegram_id**
- Backend отправляет HTTP POST запрос к DeepSeek LLM: **/chat/completions с Authorization: Bearer {api_key}** и телом запроса, содержащим системный промпт и агрегированные данные пользователя
- Если происходит ошибка генерации (HTTP error или невалидный JSON ответ):
  • Backend отправляет HTTP POST запрос на сервер Telegram: **/bot{token}/sendMessage с текстом "Произошла ошибка при сохранении плана питания. Попробуй позже или обратись в поддержку."**
  • Пользователь получает сообщение об ошибке генерации
- Если генерация успешна:
  • Backend парсит JSON ответ от LLM, содержащий: **общее количество калорий, белков, жиров, углеводов, приемы пищи (завтрак, обед, ужин)**
  • Backend выполняет SQL INSERT запрос в таблицу meal_plans: **INSERT INTO meal_plans (athlete_id, plan_date, plan_name, calories, proteins, fats, carbs, breakfast, lunch, dinner) VALUES (...)**



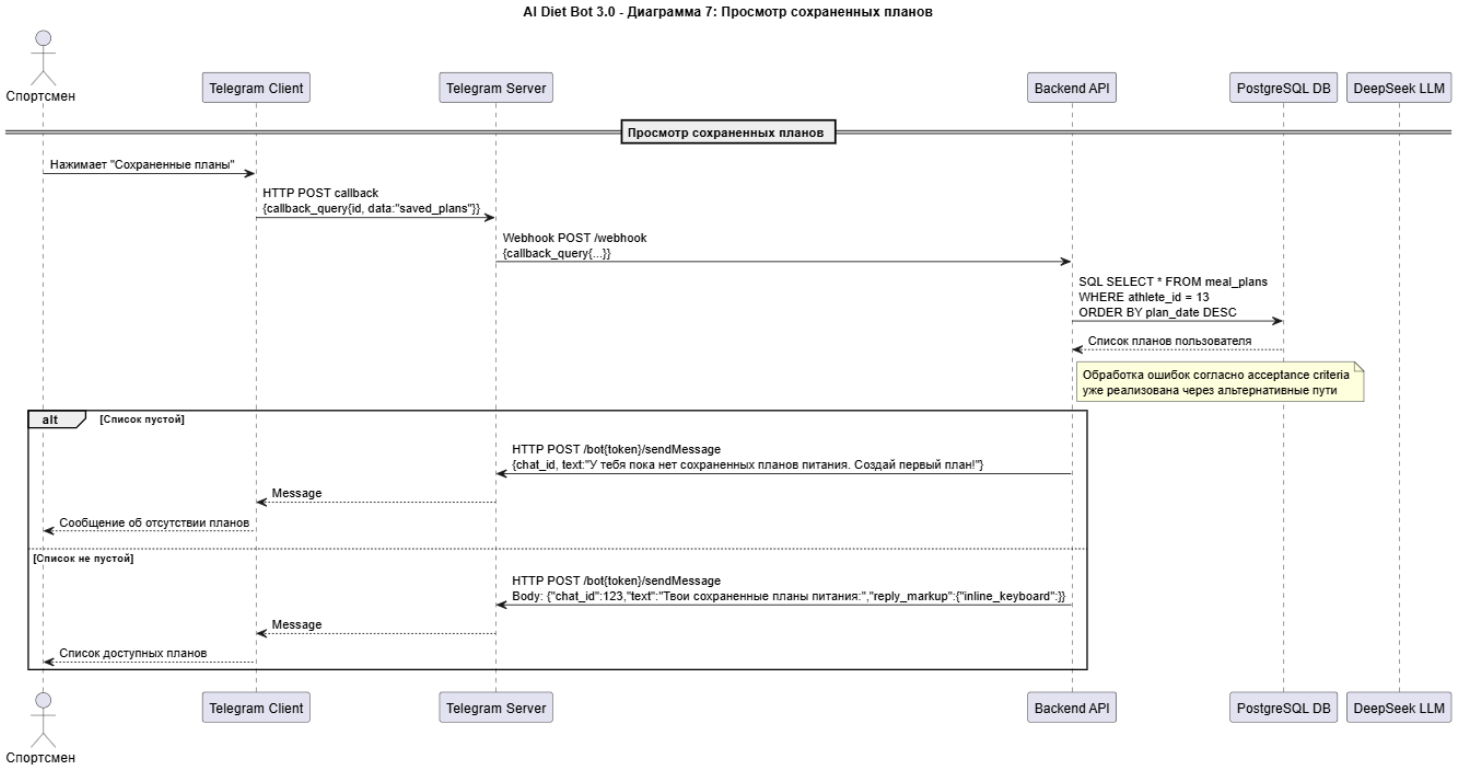
6. Отправка результата пользователю

- Backend отправляет HTTP POST запрос на сервер Telegram: **/bot{token}/sendMessage с текстом "План питания готов! Выбери действие:" и reply_markup с inline-кнопками: [{"text":"Просмотреть план","callback_data":"view_plan_{meal_id}"}, {"text":"В главное меню","callback_data":"back_to_menu"}]**
- Пользователь получает уведомление о готовности плана
- Пользователь нажимает **"Просмотреть план"**
- Telegram клиент отправляет HTTP POST callback запрос на сервер Telegram: **{callback_query{id, data:"view_plan_{meal_id}"}}**
- Сервер Telegram через webhook передает запрос в Backend
- Backend выполняет SQL SELECT запрос: **SELECT * FROM meal_plans WHERE meal_id = :meal_id**
- Если план не найден:
  • Backend отправляет сообщение об ошибке: **"План питания не найден. Попробуй создать новый план."**
- Если план найден:
  • Backend отправляет HTTP POST запрос на сервер Telegram с текстом плана питания и **reply_markup: [{"text":"Завершить просмотр","callback_data":"finish_session"}, {"text":"В главное меню","callback_data":"back_to_menu"}]**
  • Пользователь получает отображение плана питания



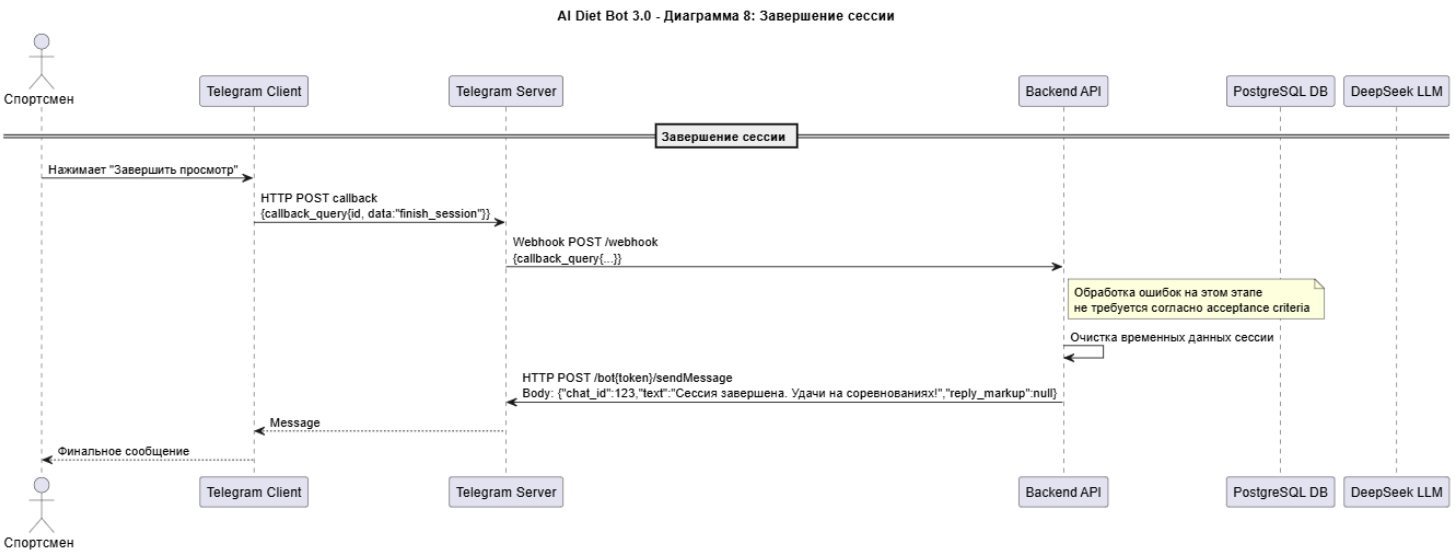
7. Просмотр сохраненных планов

- Пользователь нажимает **"Сохраненные планы"** в главном меню через Telegram клиент
- Telegram клиент отправляет HTTP POST callback запрос на сервер Telegram: **{callback_query{id, data:"saved_plans"}}**
- Сервер Telegram через webhook передает запрос в Backend
- Backend выполняет SQL SELECT запрос: **SELECT * FROM meal_plans WHERE athlete_id = :athlete_id ORDER BY plan_date DESC**
- Если список пустой:
  • Backend отправляет HTTP POST запрос на сервер Telegram: **/bot{token}/sendMessage с текстом "У тебя пока нет сохраненных планов питания. Создай первый план!"**
  • Пользователь получает сообщение об отсутствии планов
- Если список не пустой:
  • Backend отправляет HTTP POST запрос на сервер Telegram: **/bot{token}/sendMessage с текстом "Твои сохраненные планы питания:" и reply_markup с inline-кнопками планов (например, [{"text":"План от 26.09.2025","callback_data":"view_plan_89"}])**
  • Пользователь получает список доступных планов



8. Завершение сессии

- Пользователь нажимает кнопку **"Завершить просмотр"** через Telegram клиент
- Telegram клиент отправляет HTTP POST callback запрос на сервер Telegram: **{callback_query{id, data:"finish_session"}}**
- Сервер Telegram через webhook передает запрос в Backend
- Backend очищает временные данные сессии
- Backend отправляет HTTP POST запрос на сервер Telegram: **/bot{token}/sendMessage с текстом "Сессия завершена. Удачи на соревнованиях!" и reply_markup: null**
- Пользователь получает финальное сообщение с пожеланием удачи на соревнованиях



7. Модель данных

Полный скрипт для генерации базы данных доступен по ссылке:
https://drive.google.com/file/d/14bdSf1jhVyUohfaYqJ5Lk_mTl2hDDOiy/view?usp=sharing

На основе User Story и Use Case выделим основные сущности и их атрибуты:

1. Спортсмен (Athlete) - основной актор
2. План питания (MealPlan) - результат работы системы
3. Прием пищи (Meal) - завтрак/обед/ужин в плане
4. Тренировка (Workout) - режим тренировок спортсмена
5. Физическая активность (Activity) - активность вне тренировок

Теперь построим логическую модель данных с PK/FK без связей многие-ко-многим.

Логическая модель данных
athletes (Спортсмены)

Атрибут	**Описание**
athlete_id (PK)	UUID спортсмена, первичный ключ
name	Имя спортсмена
telegram_id	ID в Telegram
gender	Пол спортсмена
height	Рост спортсмена, см
current_weight	Текущий вес, кг
target_weight_category	Целевой вес, кг
competition_date	Дата соревнований
created_at	Дата и время создания в формате 2025-08-17 19:39:12.098

```sql
-- Создание таблицы спортсменов (только если не существует)
CREATE TABLE IF NOT EXISTS athletes (
    athlete_id SERIAL PRIMARY KEY,
    telegram_id BIGINT UNIQUE, -- Добавлено поле для идентификации пользователей Telegram
    name VARCHAR(100) NOT NULL,
    gender CHAR(1) CHECK (gender IN ('M', 'F')) NOT NULL,
    height DECIMAL(5,2) NOT NULL CHECK (height > 0),
    current_weight DECIMAL(5,2) NOT NULL CHECK (current_weight > 0),
    target_weight DECIMAL(5,2) NOT NULL CHECK (target_weight > 0),
    competition_date DATE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT weight_check CHECK (ABS(current_weight - target_weight) <= current_weight * 0.05)
);
```

meal_plans (Планы питания)

Атрибут	**Описание**
meal_id (PK)	UUID плана питания, первичный ключ
athlete_id (FK → athletes(athlete_id))	UUID спортсмена, внешний ключ для связи с сущностью спортсмена
meal_type	Тип приема пищи (завтрак, обед, ужин)
calories	Калораж на прием пищи, ккал
proteins	Количество белков на прием пищи, г
fats	Количество жиров на прием пищи, г
carbs	Количество углеводов на прием пищи, г
description	Описание приема пищи (наименование блюда)
plan_date	Дата плана, отображаемая спортсмену (+1 день от даты запроса на генерацию плана питания)
plan_name	Название плана
created_at	Дата и время создания в формате 2025-08-17 19:39:12.098

```
-- Создание объединенной таблицы планов питания и приемов пищи (только если не существует)
CREATE TABLE IF NOT EXISTS meal_plans (
    meal_id SERIAL PRIMARY KEY,
    athlete_id INTEGER NOT NULL REFERENCES athletes(athlete_id) ON DELETE CASCADE,
    meal_type VARCHAR(10) NOT NULL CHECK (meal_type IN ('завтрак', 'обед', 'ужин')),
    calories INTEGER NOT NULL CHECK (calories > 0),
    proteins DECIMAL(5,2) NOT NULL CHECK (proteins >= 0),
    fats DECIMAL(5,2) NOT NULL CHECK (fats >= 0),
    carbs DECIMAL(5,2) NOT NULL CHECK (carbs >= 0),
    description TEXT,
    plan_date DATE NOT NULL,
    plan_name VARCHAR(50) DEFAULT 'Основной план',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    -- Уникальное ограничение для предотвращения дублирования планов
    UNIQUE (athlete_id, plan_date, plan_name, meal_type)
);

-- Создание триггерной функции для проверки даты соревнований (только если не существует)
CREATE OR REPLACE FUNCTION check_competition_date()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.plan_date > (SELECT competition_date FROM athletes WHERE athlete_id = NEW.athlete_id) -
INTERVAL '7 days' THEN
        RAISE EXCEPTION 'План должен начинаться не позднее чем за неделю до соревнований';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Создание триггера (только если не существует)
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE tgname = 'meal_plan_date_check') THEN
        CREATE TRIGGER meal_plan_date_check
        BEFORE INSERT OR UPDATE ON meal_plans
        FOR EACH ROW EXECUTE FUNCTION check_competition_date();
    END IF;
END $$;
```

workouts (Тренировки)

Атрибут	**Описание**
workout_id (PK)	UUID тренировки, первичный ключ
athlete_id (FK → athletes(athlete_id))	UUID спортсмена, внешний ключ для связи с сущностью спортсмена
sessions_per_week	Количество тренировок в неделю
exercises	Описание упражнения
equipment_weight	Вес отягощения, кг
reps	Повторов за упражнение
sets	Подходов за упражнение
created_at	Дата и время создания в формате 2025-08-17 19:39:12.098

```sql
-- Создание таблицы тренировок (только если не существует)
CREATE TABLE IF NOT EXISTS workouts (
    workout_id SERIAL PRIMARY KEY,
    athlete_id INTEGER NOT NULL REFERENCES athletes(athlete_id) ON DELETE CASCADE,
    sessions_per_week INTEGER NOT NULL CHECK (sessions_per_week BETWEEN 1 AND 14),
    exercises VARCHAR(100) NOT NULL,
    equipment_weight DECIMAL(5,2) CHECK (equipment_weight >= 0),
    reps INTEGER CHECK (reps > 0),
    sets INTEGER CHECK (sets > 0),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

activities (Физическая активность)

Атрибут	**Описание**
activity_id (PK)	UUID физической активности, первичный ключ
athlete_id (FK → athletes(athlete_id))	UUID спортсмена, внешний ключ для связи с сущностью спортсмена
activity_type	Описание активности
duration_minutes	Продолжительность активности, мин
frequency_per_week	Количество активностей в неделю
created_at	Дата и время создания в формате 2025-08-17 19:39:12.098

```sql
-- Создание таблицы физической активности (только если не существует)
CREATE TABLE IF NOT EXISTS activities (
    activity_id SERIAL PRIMARY KEY,
    athlete_id INTEGER NOT NULL REFERENCES athletes(athlete_id) ON DELETE CASCADE,
    steps_per_day INTEGER CHECK (steps_per_day > 0),
    work_type VARCHAR(50),
    additional_activity VARCHAR(100),
    activity_hours DECIMAL(4,2) CHECK (activity_hours >= 0),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

8. Модель данных

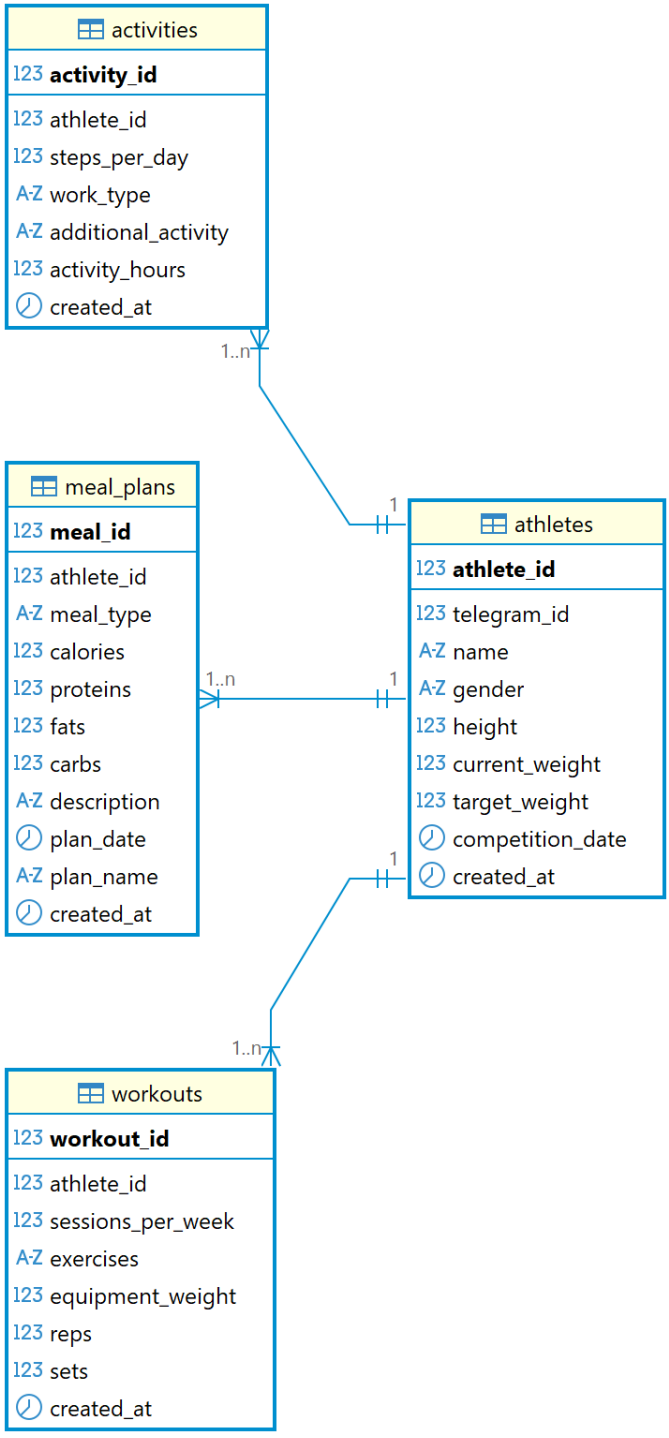
Нормализация до 3NF

Модель соответствует 3NF, так как:

1. Все атрибуты атомарны
2. Нет частичных зависимостей от составного ключа (все PK простые)
3. Нет транзитивных зависимостей - все неключевые атрибуты зависят только от PK

Дополнительные проверки реализованы через CHECK-ограничения:

- Проверка разницы веса (не более 5%)
- Проверка даты начала плана (не позднее чем за неделю до соревнований)
- Проверка типов данных и допустимых значений



9. REST. Табличный вид

POST /users - Создание нового пользователя

Создает запись пользователя в системе.

Request: UserCreate schema (body)

Название параметра	Тип данных	Описание	Обязательность параметра
telegram_id	integer	ID пользователя в Telegram	Да
name	string	Имя пользователя	Да
gender	string	Пол (М/Ж)	Да
height	number	Рост в см	Да
current_weight	number	Текущий вес в кг	Да
target_weight	number	Целевой вес в кг	Да
competition_date	string	Дата соревнований (YYYY-MM-DD)	Да

Response: Данные созданного пользователя (User)

Название параметра	Тип данных	Описание	Обязательность параметра
athlete_id	integer	ID пользователя в системе	Да
telegram_id	integer	ID пользователя в Telegram	Да
name	string	Имя пользователя	Да
gender	string	Пол (М/Ж)	Да
height	number	Рост в см	Да
current_weight	number	Текущий вес в кг	Да
target_weight	number	Целевой вес в кг	Да
competition_date	string	Дата соревнований (YYYY-MM-DD)	Да
created_at	string	Дата создания (ISO 8601 2024-01-15T10:30:00Z)	Да

GET /users - Получение списка пользователей

Возвращает список всех зарегистрированных пользователей.

Request (query)

Название параметра	Тип данных	Описание	Обязательность параметра
limit	integer	Лимит пользователей (default: 10)	Нет
offset	integer	Смещение (default: 0)	Нет

Response: Массив объектов User, для каждого из объектов массива

Название параметра	Тип данных	Описание	Обязательность параметра
athlete_id	integer	ID пользователя в системе	Да
telegram_id	integer	ID пользователя в Telegram	Да
name	string	Имя пользователя	Да
gender	string	Пол (М/Ж)	Да
height	number	Рост в см	Да
current_weight	number	Текущий вес в кг	Да
target_weight	number	Целевой вес в кг	Да
competition_date	string	Дата соревнований (YYYY-MM-DD)	Да
created_at	string	Дата создания (ISO 8601 2024-01-15T10:30:00Z)	Да

GET /users/{telegram_id} - Получение пользователя по Telegram ID

Возвращает данные конкретного пользователя.

Request (path)

Название параметра	Тип данных	Описание	Обязательность параметра
telegram_id	integer	ID пользователя в Telegram	Да

Response: Данные пользователя (User)

Название параметра	Тип данных	Описание	Обязательность параметра
athlete_id	integer	ID пользователя в системе	Да
telegram_id	integer	ID пользователя в Telegram	Да
name	string	Имя пользователя	Да
gender	string	Пол (М/Ж)	Да
height	number	Рост в см	Да
current_weight	number	Текущий вес в кг	Да
target_weight	number	Целевой вес в кг	Да
competition_date	string	Дата соревнований (YYYY-MM-DD)	Да
created_at	string	Дата создания (ISO 8601 2024-01-15T10:30:00Z)	Да

POST /meal-plans - Генерация плана питания

Генерирует новый план питания для пользователя.

Request: MealPlanGenerateRequest (body)

Название параметра	Тип данных	Описание	Обязательность параметра
telegram_id	integer	ID пользователя в Telegram	Да

Response: Возвращает сгенерированный план (MealPlan)

Название параметра	Тип данных	Описание	Обязательность параметра
plan_id	integer	ID плана	Да
athlete_id	integer	ID пользователя в системе	Да
plan_date	string	Дата плана (YYYY-MM-DD)	Да
plan_name	string	Название плана	Да
total_calories	number	Общая калорийность	Да
total_proteins	number	Белки (г)	Да
total_fats	number	Жиры (г)	Да
total_carbs	number	Углеводы (г)	Да
meals	**array [Meal]**	Список приемов пищи	Да
Meal			
meal_id	integer	ID приема пищи	Да
meal_type	string	Тип (завтрак/обед/ужин)	Да
description	string	Описание блюда	Да
calories	number	Калории	Да
proteins	number	Белки (г)	Да
fats	number	Жиры (г)	Да
carbs	number	Углеводы (г)	Да

GET /meal-plans - Получение планов питания пользователя

Возвращает список планов питания для указанного пользователя.

Request (query)

Название параметра	Тип данных	Описание	Обязательность параметра
telegram_id	integer	ID пользователя в Telegram	Да
limit	integer	Лимит планов (default: 10)	Нет
offset	integer	Смещение (default: 0)	Нет

Response: Массив объектов MealPlanSummary, для каждого из объектов массива

Название параметра	Тип данных	Описание	Обязательность параметра
plan_id	integer	ID плана	Да
plan_date	string	Дата плана (YYYY-MM-DD)	Да
plan_name	string	Название плана	Да
total_calories	number	Общая калорийность	Да

GET /meal-plans/{plan_id} - Получение детальной информации о плане

Возвращает полную информацию о конкретном плане питания.

Request (path)

Название параметра	Тип данных	Описание	Обязательность параметра
plan_id	integer	ID плана питания	Да

Response: Детальная информация о плане (MealPlanDetail)

Название параметра	Тип данных	Описание	Обязательность параметра
plan_id	integer	ID плана	Да
athlete_id	integer	ID пользователя в системе	Да
plan_date	string	Дата плана (YYYY-MM-DD)	Да
plan_name	string	Название плана	Да
total_calories	number	Общая калорийность	Да
total_proteins	number	Белки (г)	Да
total_fats	number	Жиры (г)	Да
total_carbs	number	Углеводы (г)	Да
meals	**array [Meal]**	Список приемов пищи	Да
Meal			
meal_id	integer	ID приема пищи	Да
meal_type	string	Тип (завтрак/обед/ужин)	Да
description	string	Описание блюда	Да
calories	number	Калории	Да
proteins	number	Белки (г)	Да
fats	number	Жиры (г)	Да
carbs	number	Углеводы (г)	Да

POST /webhook - Обработка входящих сообщений от Telegram

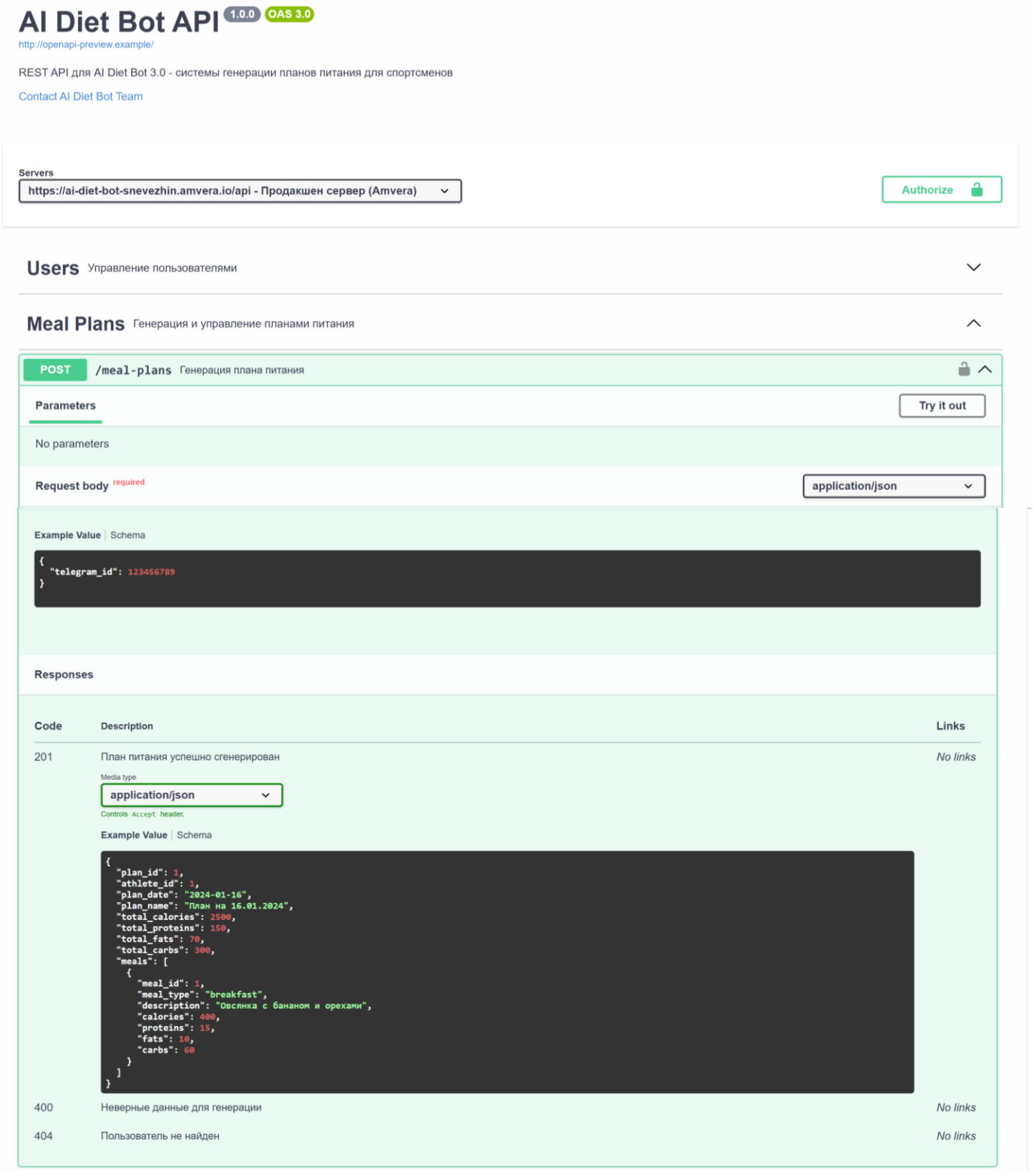
Основной endpoint для взаимодействия с Telegram Bot.
Обрабатывает: сообщения пользователей, Callback-запросы от inline-кнопок, команды бота (/start)

Request: TelegramUpdate (body)

Название параметра	Тип данных	Описание	Обязательность параметра
update_id	integer	ID обновления	Да
message	object	Текстовое сообщение	Да
message_id	integer	ID сообщения	Да
chat	object	Информация о чате	Да
id	integer	ID чата	Да
first_name	string	Имя пользователя	Да
last_name	string	Фамилия пользователя	Да
username	string	username пользователя	Да
type	string	Тип чата (private/group)	Да
text	string	Текст сообщения	Да
date	integer	ID приема пищи	Да
callback_query	object	Callback от inline-кнопки	Да
id	string	ID для callback	Да
from	object	Информация о пользователе	Да
data	string	Данные callback (/create_plan)	Да
message	object	Сообщение, к которому привязана кнопка	Да

10. Swagger

Полный текст спецификации OpenAPI в формате yaml доступен по ссылке:
https://drive.google.com/file/d/1vca7ScqPMobpH3d896uoElg0530IyEU-/view?usp=sharing



GET `/meal-plans` Получение планов питания пользователя

Parameters | Cancel

Name	Description
telegram_id * required integer *(query)*	ID пользователя в Telegram 8899900
limit integer *(query)*	10
offset integer *(query)*	0

Execute | Clear

Responses

Curl

```
curl -X 'GET' \
  'https://ai-diet-bot-snevezhin.amvera.io/api/meal-plans?telegram_id=8899900&limit=10&offset=0' \
  -H 'accept: application/json'
```

Request URL

```
https://ai-diet-bot-snevezhin.amvera.io/api/meal-plans?telegram_id=8899900&limit=10&offset=0
```

Server response

Code	Details
Undocumented	**Failed to fetch.** **Possible Reasons:** • CORS • Network Failure • URL scheme must be "http" or "https" for CORS request.

Responses

Code	Description	Links
200	Список планов питания Media type: application/json Controls Accept header. Example Value | Schema	*No links*

```
[
  {
    "plan_id": 1,
    "plan_date": "2024-01-16",
    "plan_name": "План на 16.01.2024",
    "total_calories": 2500
  }
]
```

GET `/meal-plans/{plan_id}` Получение детальной информации о плане питания

Parameters | Try it out

Name	Description
plan_id * required integer *(path)*	ID плана питания plan_id

Responses

Code	Description	Links
200	Детальная информация о плане	*No links*

Media type: application/json

Controls Accept header.

Example Value | Schema


```
{
  "plan_id": 1,
  "athlete_id": 1,
  "plan_date": "2024-01-16",
  "plan_name": "План на 16.01.2024",
  "total_calories": 2500,
  "total_proteins": 150,
  "total_fats": 70,
  "total_carbs": 300,
  "meals": [
    {
      "meal_id": 1,
      "meal_type": "breakfast",
      "description": "Овсянка с бананом и орехами",
      "calories": 400,
      "proteins": 15,
      "fats": 10,
      "carbs": 60
    }
  ]
}
```


404	План питания не найден	*No links*

Telegram Webhook

Обработка входящих сообщений от Telegram

POST `/webhook` Обработка входящих сообщений от Telegram

Parameters | Try it out

No parameters

Request body required — application/json

Example Value | Schema


```
{
  "update_id": 123456789,
  "message": {
    "message_id": 123,
    "chat": {
      "id": 123456789,
      "first_name": "Иван",
      "last_name": "Иванов",
      "username": "ivanov",
      "type": "private"
    },
    "text": "/start",
    "date": 1700000000
  },
  "callback_query": {
    "id": "123456789",
    "from": {
      "id": 123456789,
      "first_name": "Иван",
      "last_name": "Иванов",
      "username": "ivanov"
    },
    "data": "create_plan",
    "message": {
      "message_id": 123,
      "chat": {
        "id": 123456789
      }
```


Responses

Code	Description	Links
200	Сообщение успешно обработано	*No links*

Media type: application/json

Controls Accept header.

Example Value | Schema


```
{
  "status": "ok"
}
```


400	Неверный формат сообщения	*No links*
500	Ошибка обработки сообщения	*No links*

11. Критерии приемки

Функциональность: Инициализация нового пользователя

Сценарий 1: Новый пользователь запускает бота
Дано: Новый пользователь открывает Telegram бота AI диетолог 3.0
Когда: Пользователь отправляет команду /start
Тогда: Telegram бот отображает приветственное сообщение “Привет! Я AI диетолог 3.0, твой персональный помощник в создании идеального плана питания для подготовки к соревнованиям! Для начала, давай познакомимся поближе. Ответь на несколько вопросов о себе: Как тебя зовут? (укажи свое имя)”

Сценарий 2: Процесс знакомства с новым пользователем
Дано: Новый пользователь получил приветственное сообщение
Когда: Пользователь последовательно вводит все параметры: - Имя: “Станислав” - Дата соревнований: “15.10.2025” - Пол: “М” - Рост: “180.5” - Текущий вес: “75.5” - Целевой вес: “72.0”
Тогда: Telegram бот сохраняет все параметры и отображает сообщение “Параметры сохранены! Переходим к вопросам о тренировках ”

Сценарий 3: Ввод невалидной даты соревнований
Дано: Пользователь вводит параметры
Когда: Пользователь указывает дату соревнований, до которых осталось меньше недели или дату соревнований в прошлом
Тогда: Telegram бот отображает сообщение “Ошибка: До начала соревнований должно оставаться не меньше недели. Рекомендую выступить на следующих соревнованиях! Пожалуйста, укажи другую дату:”

Сценарий 4: Ввод невалидного пола
Дано: Пользователь вводит параметры
Когда: Пользователь указывает неопределенный пол (заложены М/Ж Муж/Жен Мужской/Женский M/F Male/Female)
Тогда: Telegram бот отображает сообщение “Неверный формат данных. Пожалуйста, укажи свой пол (М/Ж):”

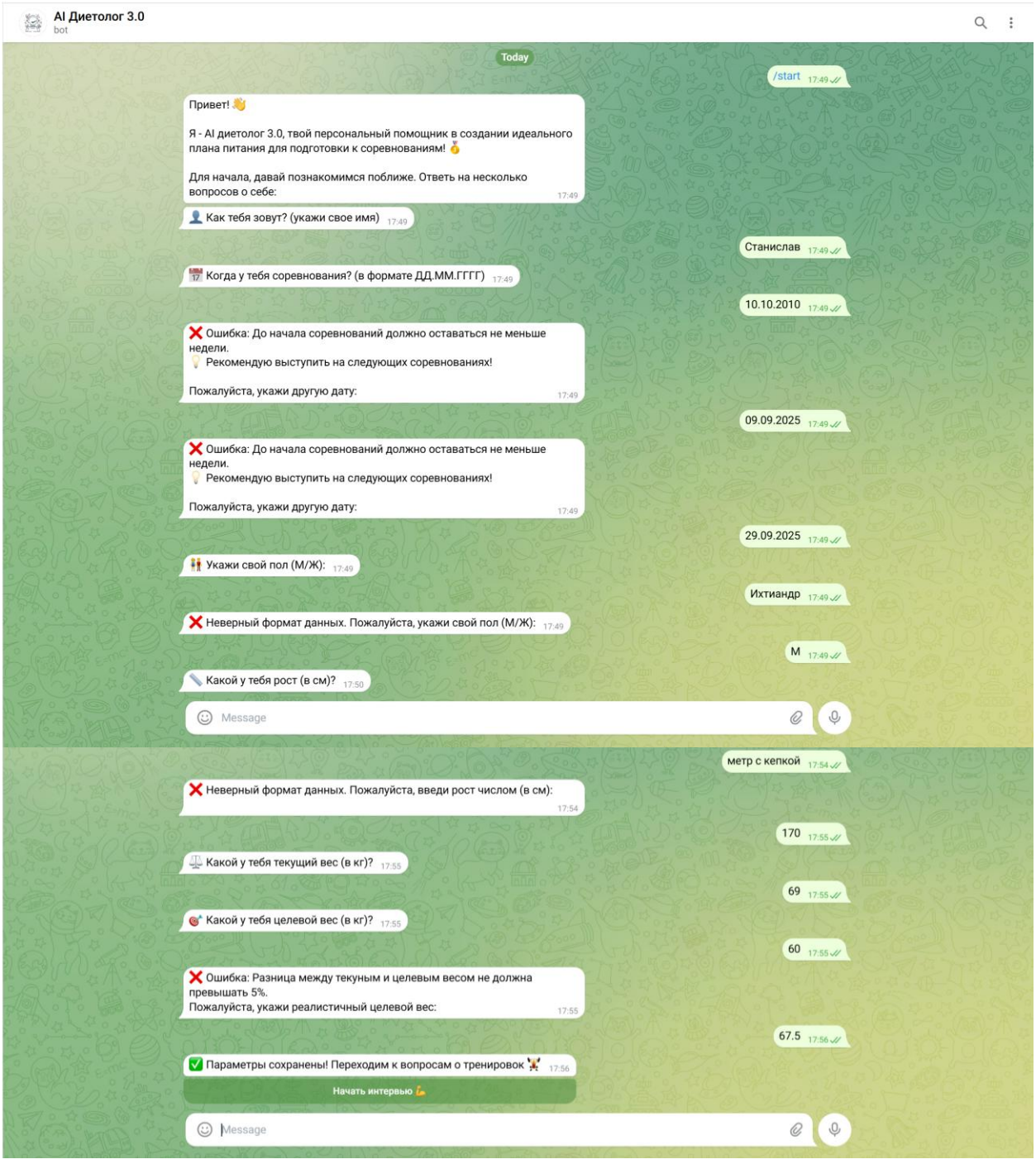
Сценарий 5: Ввод невалидного формата данных
Дано: Пользователь вводит параметры
Когда: Пользователь указывает рост в неправильном формате “сто восемьдесят”
Тогда: Telegram бот отображает сообщение “Неверный формат роста. Введите число с точкой (например: 180.5)”

Сценарий 6: Невалидная разница весов

Дано: Пользователь вводит параметры

Когда: Пользователь указывает текущий вес “60.0” и целевой вес “80.0” (разница более 5%)

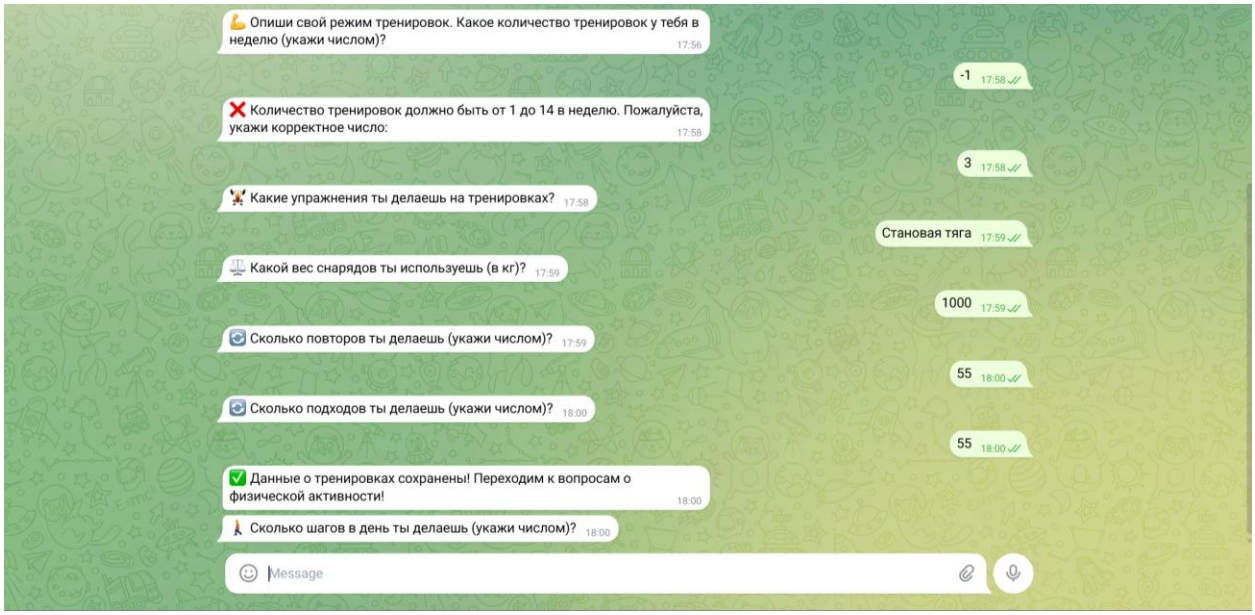
Тогда: Telegram бот отображает сообщение “Разница между текущим и целевым весом не должна превышать 5%. Введите корректные значения.”



Функциональность: Этап интервью - Режим тренировок

Сценарий 7: Успешный ввод данных о тренировках
Дано: Пользователь завершил ввод параметров
Когда: Пользователь вводит валидные данные о тренировках: - Количество тренировок: “4” - Упражнения: “жим лежа, приседания” - Вес снарядов: “125.5” - Повторы: “8” - Подходы: “4”
Тогда: Telegram бот сохраняет данные и переходит к этапу вопросов о физической активности

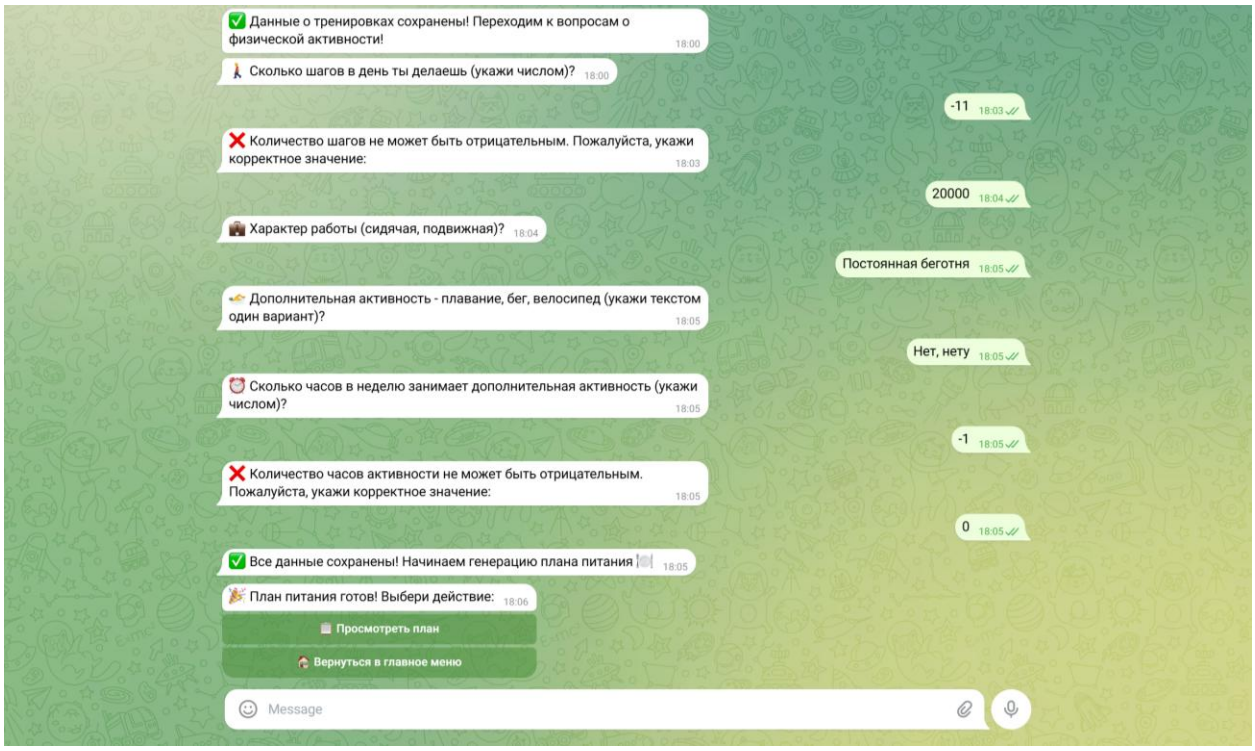
Сценарий 8: Ввод невалидного количества тренировок
Дано: Пользователь вводит данные о тренировках
Когда: Пользователь указывает количество тренировок “0” или “-1”
Тогда: Telegram бот отображает сообщение “Количество тренировок должно быть от 1 до 14 в неделю. Введите корректное значение.”



Функциональность: Этап интервью - Физическая активность

Сценарий 9: Успешный ввод данных об активности
Дано: Пользователь завершил ввод данных о тренировках
Когда: Пользователь вводит валидные данные об активности: - Шаги в день: “10000” - Характер работы: “сидячая” - Дополнительная активность: “бег” - Часов активности: “1.5”
Тогда: Telegram бот сохраняет данные и начинает генерацию плана питания

Сценарий 10: Ввод отрицательного значения активности
Дано: Пользователь вводит данные об активности
Когда: Пользователь указывает часов активности “-1.0”
Тогда: Telegram бот отображает сообщение “Количество часов активности не может быть отрицательным. Введите корректное значение.”



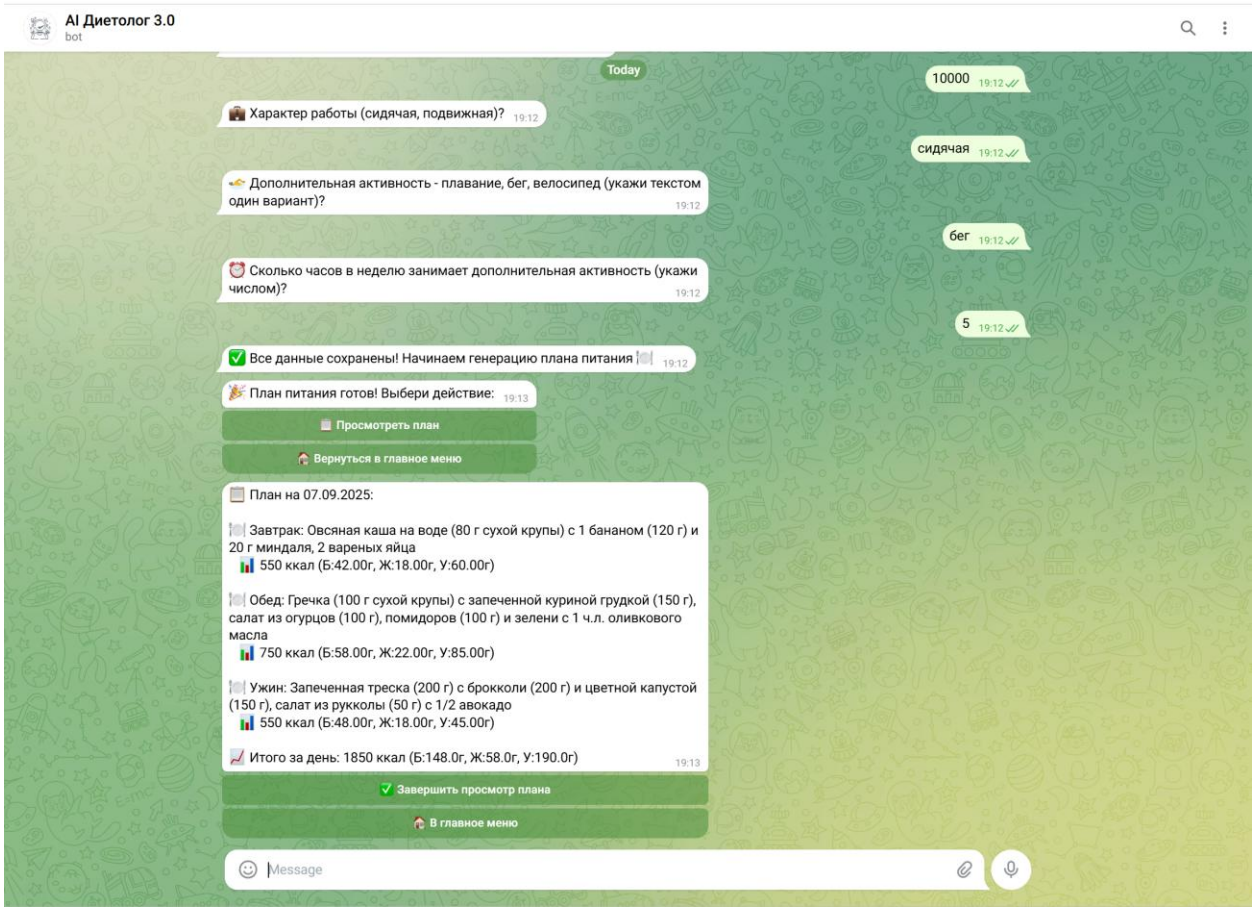
Функциональность: Генерация плана питания

Сценарий 11: Успешная генерация плана
Дано: Пользователь завершил ввод всех данных
Когда: Telegram бот отправляет запрос к LLM с агрегированными данными пользователя
Тогда: Telegram бот получает ответ от LLM и сохраняет план в базу данных с полями: - Дата начала плана (следующий день) - Общее количество калорий - Общее количество белков, жиров, углеводов - Детали приемов пищи (завтрак, обед, ужин)
Сценарий 12: Ошибка генерации плана
Дано: Пользователь завершил ввод всех данных
Когда: Telegram бот отправляет запрос к LLM
Тогда: При ошибке генерации Telegram бот отображает сообщение “Произошла ошибка при сохранении плана питания. Попробуй позже или обратись в поддержку.”

Функциональность: Отправка результата пользователю

Сценарий 13: Успешное уведомление о готовности плана
Дано: План питания успешно сгенерирован и сохранен
Когда: Telegram бот завершает генерацию плана
Тогда: Пользователь получает сообщение “План питания готов! Выбери действие: [Просмотреть план] [Вернуться в главное меню]”

Сценарий 14: Просмотр сгенерированного плана
Дано: Пользователь получил уведомление о готовности плана
Когда: Пользователь нажимает кнопку “Просмотреть план”
Тогда: Telegram бот отображает детальный план питания с: - Датой начала плана - Общей калорийностью - Содержанием белков, жиров, углеводов - Детализацией приемов пищи - Кнопками [Завершить просмотр] [Вернуться в главное меню]

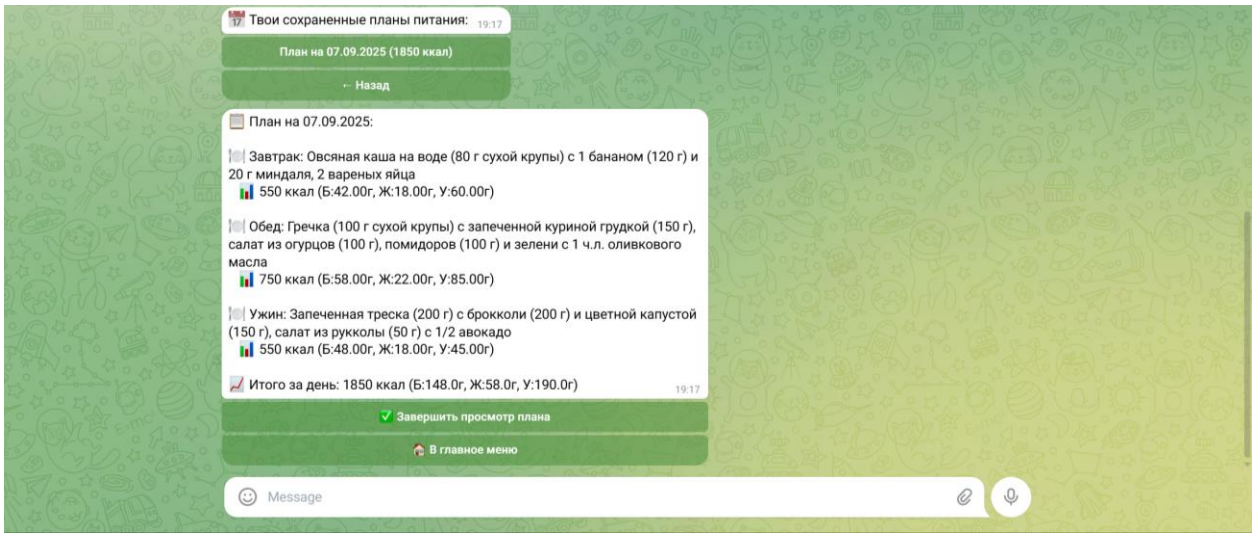


Функциональность: Просмотр сохраненных планов

Сценарий 15: Просмотр списка планов (есть сохраненные)
Дано: Пользователь находится в главном меню
Когда: Пользователь нажимает кнопку “Сохраненные планы”
Тогда: Telegram бот отображает список доступных планов с датами их создания

Сценарий 16: Просмотр списка планов (нет сохраненных)
Дано: Пользователь не имеет сохраненных планов питания
Когда: Пользователь нажимает кнопку “Сохраненные планы”
Тогда: Telegram бот отображает сообщение “У тебя пока нет сохраненных планов питания. Создай первый план!”

Сценарий 17: Просмотр конкретного плана
Дано: Пользователь просматривает список сохраненных планов
Когда: Пользователь выбирает конкретный план из списка
Тогда: Telegram бот отображает детальную информацию о выбранном плане питания



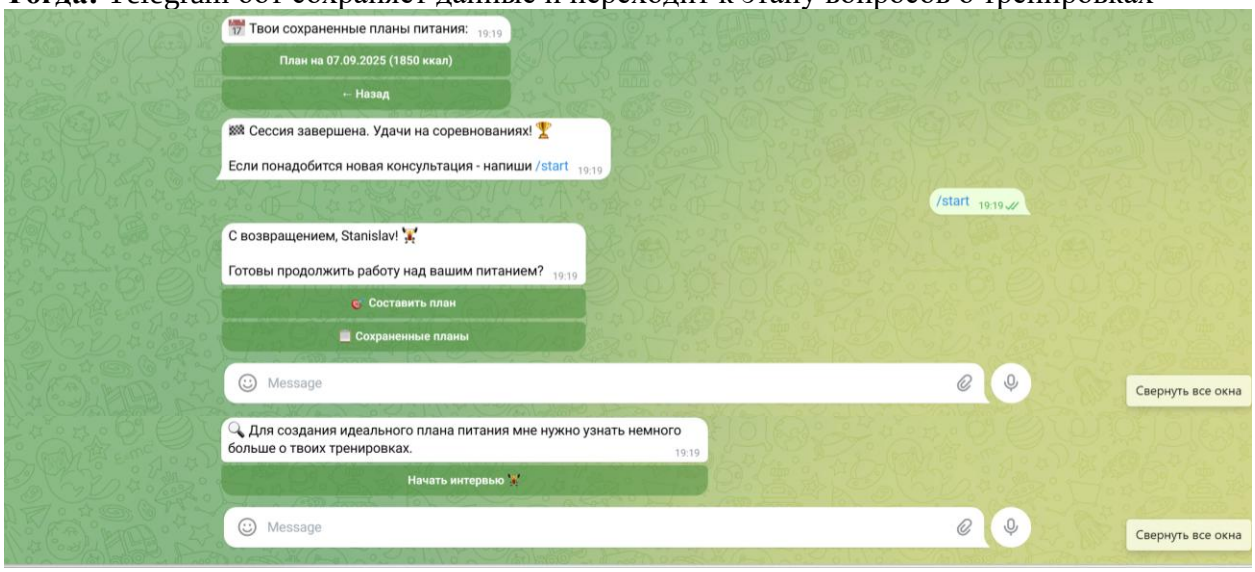
Функциональность: Завершение сессии

Сценарий 18: Завершение просмотра плана
Дано: Пользователь просматривает план питания
Когда: Пользователь нажимает кнопку “Завершить просмотр”
Тогда: Telegram бот очищает временные данные сессии и отображает финальное сообщение “Сессия завершена. Удачи на соревнованиях!”

Функциональность: Составление нового плана существующим пользователем

Сценарий 19: Существующий пользователь запускает бота
Дано: Пользователь с сохраненными данными открывает Telegram бота
Когда: Пользователь отправляет команду /start
Тогда: Telegram бот отображает главное меню с приветствием “С возвращением!” и кнопками [Составить план] [Сохраненные планы]

Сценарий 20: Существующий пользователь составляет план
Дано: Пользователь находится в главном меню
Когда: Пользователь нажимает кнопку “Составить план”
Тогда: Telegram бот сохраняет данные и переходит к этапу вопросов о тренировках



12. Нефункциональные требования

Введение

Документ описывает нефункциональные требования для Telegram бота “AI Diet Bot 3.0” - системы для создания персонализированных планов питания спортсменам. Бот построен на aiogram 3.x с интеграцией DeepSeek API и PostgreSQL базой данных.

1. Производительность (Performance)

NFR-PERF-001: Время отклика бота
Описание: Бот должен обеспечивать быстрый отклик на команды и сообщения пользователей
Критерии измерения:
- Время обработки простых команд (/start, /help): не более 500 мс (95-й процентиль)
- Время обработки текстовых сообщений: не более 1 секунды (95-й процентиль)
- Время генерации callback запросов: не более 800 мс (95-й процентиль)
- Время генерации плана питания через LLM: не более 120 секунд (максимальный таймаут)
Условия измерения:
- Среда: Production окружение на Amvera
- Нагрузка: до 100 одновременных пользователей
- Сеть: стабильное интернет-соединение
- Браузер: Telegram Web/Desktop версии
Инструменты: Bot analytics, Python logging, Amvera monitoring
Приоритет: Высокий
Обоснование: Скорость отклика критична для пользовательского опыта в мессенджере

NFR-PERF-002: Производительность базы данных
Описание: База данных PostgreSQL должна эффективно обрабатывать запросы
Критерии измерения:
- Время выполнения простых SELECT запросов: не более 100 мс
- Время выполнения сложных JOIN запросов: не более 300 мс
- Время вставки записей: не более 200 мс
- Максимальное количество соединений: не менее 100 одновременных подключений
Условия измерения:
- БД: PostgreSQL 13+ на Amvera
- Размер данных: до 10,000 пользователей, 50,000 планов питания
- Индексы: все основные поля проиндексированы
Инструменты: EXPLAIN ANALYZE, pg_stat_statements, Amvera DB metrics
Приоритет: Высокий
Обоснование: Производительность БД прямо влияет скорость работы бота

NFR-PERF-003: Эффективность LLM интеграции
Описание: Интеграция с DeepSeek API должна быть оптимизирована
Критерии измерения:
- Время установления соединения с API: не более 5 секунд
- Время обработки промпта: не более 108 секунд (90-й процентиль)
- Успешность запросов: не менее 95% успешных ответов
- Retry механизм: до 3 попыток при временных ошибках
Условия измерения:
- API: DeepSeek Chat API
- Размер промпта: до 4000 токенов
- Сеть: стабильное соединение с api.deepseek.com
Инструменты: Aiohttp metrics, Custom logging, DeepSeek API logs
Приоритет: Средний
Обоснование: Генерация планов - ключевая функция, требующая надежности

2. Безопасность (Security)

NFR-SEC-001: Защита пользовательских данных
Описание: Система должна обеспечивать конфиденциальность персональных данных
Критерии измерения:
- Шифрование соединения: TLS 1.2+ для всех внешних коммуникаций
- Хранение паролей/ключей: в environment variables, не в коде
- Логирование: исключение чувствительных данных из логов
- Валидация входных данных: защита от SQL injection и XSS
Условия тестирования:
- Сканирование уязвимостей: OWASP ZAP, sqlmap
- Аудит кода: проверка на наличие hardcoded credentials
- Тестирование на инъекции: SQL, NoSQL, Command injection
Инструменты: OWASP ZAP, Bandit, Safety check
Приоритет: Критический
Обоснование: Защита персональных данных пользователей и спортивных показателей

NFR-SEC-002: Аутентификация и авторизация
Описание: Контроль доступа к функциональности бота
Критерии измерения:
- Валидация Telegram WebApp данных: проверка хэшей и временных меток
- Ограничение частоты запросов: не более 10 сообщений в минуту на пользователя
- Защита от ботов: CAPTCHA или аналоги при подозрительной активности
- Аудит действий: логирование всех значимых операций
Условия тестирования:
- Тестирование на брутфорс: попытки подбора команд
- Проверка WebApp security: валидация origin и hash
- Мониторинг аномальной активности: необычные паттерны использования
Инструменты: Telegram Bot API security, Custom rate limiting
Приоритет: Высокий
Обоснование: Предотвращение злоупотреблений и несанкционированного доступа

NFR-SEC-003: Безопасность API эндпоинтов
Описание: Защита HTTP API от несанкционированного доступа
Критерии измерения:
- CORS политики: строгое ограничение разрешенных origin
- Rate limiting: не более 60 запросов в минуту с IP
- Аутентификация API: JWT или API keys для защищенных endpoints
- Валидация входных данных: проверка всех параметров запросов
Условия тестирования:
- Сканирование API: OWASP ZAP, Burp Suite
- Тестирование на инъекции: проверка всех параметров
- Проверка CORS: попытки запросов с неразрешенных доменов
Инструменты: FastAPI security features, CORS middleware, Rate limiters
Приоритет: Средний
Обоснование: API используется для интеграций и требует защиты от злоупотреблений

3. Надежность (Reliability)

NFR-REL-001: Доступность системы
Описание: Система должна быть доступна для пользователей
Критерии измерения:
- Общая доступность: не менее 99.5% в месяц (максимум 3.6 часа простоя)
- Время восстановления (MTTR): не более 15 минут после сбоя
- Мониторинг: 24/7 отслеживание статуса бота и API
- Резервное копирование: ежедневные бэкапы базы данных

Условия тестирования:
- Нагрузочное тестирование: проверка стабильности под нагрузкой
- Тестирование отказоустойчивости: имитация сбоев компонентов
- Disaster recovery: процедуры восстановления после серьезных сбоев
Инструменты: UptimeRobot, Prometheus, Amvera monitoring
Приоритет: Критический
Обоснование: Недоступность бота приводит к потере пользователей и доверия

NFR-REL-002: Отказоустойчивость компонентов
Описание: Система должна gracefully обрабатывать сбои компонентов
Критерии измерения:
- Обработка ошибок БД: retry логика при временной недоступности
- Fallback механизмы: альтернативные планы при недоступности LLM
- Graceful degradation: сохранение базовой функциональности при частичных сбоях
- Мониторинг зависимостей: отслеживание статуса внешних сервисов
Условия тестирования:
- Тестирование сетевых разрывов: имитация потери связи с API/БД
- Тестирование таймаутов: проверка обработки долгих запросов
- Тестирование ограничений ресурсов: нехватка памяти/CPU
Инструменты: Custom retry logic, Circuit breakers, Health checks
Приоритет: Высокий
Обоснование: Минимизация сбоев внешних сервисов на пользователей

NFR-REL-003: Целостность данных
Описание: Гарантия сохранности и consistency данных
Критерии измерения:
- Транзакционность: atomic операции при сохранении связанных данных
- Валидация данных: проверка целостности при чтении/записи
- Консистентность: отсутствие orphaned записей и нарушений constraints
- Бэкапы: возможность восстановления данных до последнего consistent состояния
Условия тестирования:
- Тестирование конкурентного доступа: race conditions и deadlocks
- Тестирование отказов транзакций: проверка rollback механизмов
- Валидация миграций: корректность изменения схемы БД
Инструменты: PostgreSQL transactions, Data validation, Backup verification
Приоритет: Высокий
Обоснование: Потеря или повреждение данных пользователей недопустима

4. Масштабируемость (Scalability)
NFR-SCAL-001: Горизонтальное масштабирование
Описание: Возможность обработки растущего количества пользователей
Критерии измерения:
- Максимальная нагрузка: поддержка до 1000 одновременных пользователей
- Линейное масштабирование: добавление инстансов увеличивает capacity пропорционально
- Балансировка нагрузки: равномерное распределение между инстансами
- Stateless архитектура: минимальная зависимость от shared state
Условия тестирования:
- Нагрузочное тестирование: постепенное увеличение пользователей
- Тестирование масштабирования: добавление/удаление инстансов
- Мониторинг производительности: метрики под нагрузкой
Инструменты: Locust, k6, Amvera auto-scaling
Приоритет: Средний
Обоснование: Поддержка роста пользовательской базы без деградации

NFR-SCAL-002: Масштабирование базы данных
Описание: Поддержка роста объема данных
Критерии измерения:
- Производительность БД: стабильное время ответа при росте данных
- Индексация: оптимальные индексы для частых запросов
- Partitioning: возможность сегментирования больших таблиц
- Read replicas: поддержка реплик для чтения при высокой нагрузке
Условия тестирования:
- Тестирование с большими объемами данных: 100k+ пользователей
- Проверка производительности запросов: EXPLAIN ANALYZE
- Мониторинг роста данных: прогнозирование capacity
Инструменты: PostgreSQL performance tuning, Index optimization, Monitoring
Приоритет: Средний
Обоснование: Обеспечение производительности при росте данных

5. Удобство использования (Usability)

NFR-USAB-001: Интуитивный интерфейс
Описание: Бот должен быть простым и понятным для пользователей
Критерии измерения:
- Время обучения: не более 5 минут для освоения основных функций
- Количество шагов: не более 3 кликов для основных операций
- Ясность сообщений: понятные формулировки и инструкции
- Обратная связь: индикаторы прогресса и статуса операций
Условия тестирования:
- Юзабилити-тестирование: с реальными пользователями
- A/B тестирование: сравнение разных интерфейсов
- Сбор фидбека: анкетирование и отзывы пользователей
Инструменты: Telegram Bot UI best practices, User testing, Analytics
Приоритет: Высокий
Обоснование: Удобство использования прямо влияет на удержание пользователей

NFR-USAB-002: Обработка ошибок
Описание: Понятные сообщения об ошибках для пользователей
Критерии измерения: - Ясность ошибок: понятные сообщения на русском языке - Guidance: предложения по исправлению ошибок - Consistency: единообразие формата ошибок - Logging: детальное логирование для debugging
Условия тестирования:
- Тестирование edge cases: ввод неверных данных
- Проверка сообщений: ясность и полезность для пользователя
- Мониторинг ошибок: частотность и типы ошибок
Инструменты: Custom error handlers, User feedback, Error tracking
Приоритет: Средний
Обоснование: Качественная обработка ошибок улучшает пользовательский опыт

6. Совместимость (Compatibility)

NFR-COMP-001: Совместимость с Telegram
Описание: Поддержка всех основных платформ Telegram
Критерии измерения:
- Telegram Android: полная функциональность
- Telegram iOS: полная функциональность
- Telegram Desktop: полная функциональность
- Telegram Web: полная функциональность
- Inline keyboards: корректное отображение и обработка
Условия тестирования:
- Кросс-платформенное тестирование: на всех основных клиентах

- Проверка форматов: корректность Markdown/HTML разметки
- Тестирование features: все типы сообщений и кнопок
Инструменты: Telegram test accounts, Real devices testing
Приоритет: Высокий
Обоснование: Охват максимальной аудитории пользователей Telegram

NFR-COMP-002: Совместимость API
Описание: Поддержка стандартов и обратная совместимость
Критерии измерения: - OpenAPI спецификация: соответствие OpenAPI 3.0
- JSON форматы: стандартные структуры ответов
- HTTP коды: корректное использование статусов
- Обратная совместимость: deprecated endpoints с warning
Условия тестирования:
- Валидация OpenAPI: проверка спецификации
- Тестирование клиентов: различные HTTP клиенты
- Проверка миграций: обновление версий API
Инструменты: Swagger UI, OpenAPI validators, API testing tools
Приоритет: Средний
Обоснование: Упрощение интеграций для сторонних разработчиков

7. Поддерживаемость (Maintainability)
NFR-MAINT-001: Качество кода
Описание: Чистый и поддерживаемый код
Критерии измерения:
- Покрытие тестами: не менее 70% unit тестов, 50% integration тестов
- Статический анализ: 0 critical issues в SonarQube
- Стиль кода: соответствие PEP 8 и внутренним стандартам
- Документация: README для каждого модуля, docstrings для сложных функций
- Модульность: четкие границы компонентов, слабая связанность
Условия тестирования:
- Code review: обязательный review для всех изменений
- Автоматизированные проверки: pre-commit hooks и CI checks
- Метарики качества: регулярные отчеты о качестве кода
- Рефакторинг: плановый рефакторинг сложных участков
Инструменты: Pytest, SonarQube, Flake8, Black, Mypy
Приоритет: Высокий
Обоснование: Качество кода влияет на скорость разработки и количество багов

NFR-MAINT-002: Документация
Описание: Полная и актуальная документация
Критерии измерения:
- API документация: полное покрытие всех эндпоинтов
- Архитектурная документация: диаграммы и описания компонентов
- Deployment guide: инструкции по развертыванию
- Troubleshooting: руководство по решению проблем
- Обновляемость: документация обновляется вместе с кодом
Условия тестирования:
- Проверка актуальности: соответствие документации и кода
- Юзабилити документации: понятность для новых разработчиков
- Полнота: покрытие всех значимых аспектов системы
Инструменты: Sphinx, OpenAPI, PlantUML, Markdown
Приоритет: Средний
Обоснование: Качественная документация ускоряет онбординг и снижает bus factor

NFR-MAINT-003: Мониторинг и логирование
Описание: Комплексный мониторинг работы системы
Критерии измерения:
- Логирование: структурированные логи с контекстом
- Метрики: ключевые метрики производительности и использования
- Алертинг: автоматические алерты при проблемах
- Dashboards: визуализация состояния системы
- Audit trail: логирование значимых действий пользователей
Условия тестирования:
- Проверка логов: наличие необходимой информации для debugging
- Тестирование алертов: корректность срабатывания
- Проверка метрик: точность и полнота данных
Инструменты: Prometheus, Grafana, ELK stack, Custom metrics
Приоритет: Высокий
Обоснование: Эффективный мониторинг позволяет быстро обнаруживать и решать проблемы

8. Заключение

Сводка требований

Категория	Количество требований	Приоритет
Производительность	3	Высокий
Безопасность	3	Критический
Надежность	3	Высокий
Масштабируемость	2	Средний
Удобство использования	2	Высокий
Совместимость	2	Высокий
Поддерживаемость	3	Высокий
Итого	**18**	

Приоритизация

Критический приоритет:
- Безопасность пользовательских данных
- Защита от инъекций и атак
Высокий приоритет:
- Производительность и отзывчивость
- Надежность и доступность
- Удобство использования
- Качество кода и мониторинг
Средний приоритет:
- Масштабируемость
- Документация
- Совместимость API

Метрики успеха

- **Availability:** 99.5% доступность сервиса
- **Performance:** <1сек время отклика для 95% запросов
- **Reliability:** <15 мин MTTR
- **Security:** 0 критических уязвимостей
- **Quality:** >70% покрытие тестами

Ревизии

Версия	Дата	Описание изменений
1.0	2025-09-06	Первоначальная версия требований
1.1	2025-09-09	Добавлены метрики успеха

Приложение А: Словарь терминов

AI Diet Bot - Telegram бот для создания планов питания спортсменам
LLM (Large Language Model) - Большая языковая модель (DeepSeek)
FSM (Finite State Machine) - Конечный автомат для управления диалогом
MTTR (Mean Time To Repair) - Среднее время восстановления после сбоя
SLA (Service Level Agreement) - Соглашение об уровне обслуживания
CORS (Cross-Origin Resource Sharing) - Совместное использование ресурсов между разными источниками

Приложение Б: Ссылки и ресурсы

- Telegram Bot API Documentation
- DeepSeek API Documentation
- PostgreSQL Documentation
- OWASP Security Guidelines
- Aiogram Framework Documentation

Документ подготовлен в соответствии со стандартами качества системного анализа.
Последнее обновление: 2025-09-09