МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ»
(национальный исследовательский университет)»

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

КУРСОВАЯ РАБОТА

по дисциплине «Базы данных»

на тему: «Разработка информационной системы для трекинга целей и привычек»

Выполнил:
студент группы М8О-308Б-23 _____ / Власко М. М. /

Проверил:
преподаватель _____ / _____ /

Оценка: _____ **Дата:** _____

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность выбранной для разработки темы обусловлена прежде всего отсутствием чрезмерно большого количества подобных систем (среди немногочисленных примеров Habit Tracker [1]), в особенности с открытым исходным кодом, что открывало достаточно обширные возможности для самостоятельного исследования и разработки вопроса. Кроме того, особенности предметной области, в значительной мере подходящей для данной темы, обеспечивали выполнение основных требований к работе (объём предметной области и т. д.).

Объектом проводимого исследования является автоматизация процесса управления личными целями и привычками пользователя. Предмет исследования – возможности выбранных системы управления базами данных (СУБД) и веб-фреймворка для организации требуемой функциональности информационной системы.

В рамках исследования проводились анализ доступных СУБД и веб-фреймворков, анализ предметной области, проектирование структуры базы данных, разработка на её основе backend-приложения, тестирование и отладка разработанной информационной системы.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Выбор СУБД и веб-фреймворка

В качестве системы управления базой данных была выбрана PostgreSQL, как одна из наиболее эффективных и требуемая по техническому заданию работы.

В качестве веб-фреймворка выбран Django и его расширение Django REST Framework (Python) [2], как один из наиболее простых в изучении и использовании (что связано в значительной мере со сжатыми сроками разработки), и позволяющий эффективно реализовать многие тривиальные операции по взаимодействию с базой данных, каковых в разрабатываемой системе большинство. Для разработки требуемой документации в формате Swagger использовалась библиотека drf-spectacular – стандартное решение для генерации документации в Django REST Framework [3].

1.2 Анализ предметной области и постановка задачи

Задачей данного исследования является разработка информационной системы, которая будет обеспечивать автоматизированное управление целями и привычками пользователей, возможность создания записей о привычках и целях, а также о фактах соблюдения или несоблюдения привычки, и о прогрессе, а также факте достижения конкретной цели. Кроме того, система должна обеспечивать социальное взаимодействие пользователей внутри, а именно, возможность подписки пользователей друг на друга (с целью отслеживания взаимного прогресса), а также участия в соревнованиях (челленджах), которые в данной реализации будут проводится на предмет максимального прогресса по достижению целей пользователями.

Таким образом, предметная область прежде всего включает в себя сущности пользователя, цели и привычки; записи о соблюдении привычки и записи о прогрессе в достижении цели; челленджа и категории (по которой могут группироваться цели, привычки и челленджи).

2 ПРОЕКТНАЯ ЧАСТЬ

2.1 Описание архитектуры системы

Общая архитектура системы была выбрана в соответствии с традиционной для Django схемой – реализация API (эндпоинтов) логически разделена на несколько приложений (программных модулей Python), каждое из которых реализует работу в основном с отдельной конкретной сущностью. Таким образом, программный код сгруппирован и структурирован, что упрощает его понимание и поддержку.

Ядром системы на основе Django является основной модуль, который содержит несколько ключевых подмодулей: asgi, wsgi, urls и settings.

Модули asgi и wsgi обеспечивают интерфейс (точки входа) для развертывания веб-приложения на сервере.

Модуль urls – это корневой маршрутизатор проекта, который объединяет и определяет все URL-маршруты, реализованные в отдельных приложениях.

В модуле settings определены глобальные настройки системы, например, в случае разработанной системы в этом модуле подключается конкретная база данных, подключаются приложения (системные и собственные), настраивается IP-адрес и порт сервера, аутентификация и общий вид интерфейса Swagger.

Остальные приложения имеют следующую структуру:

— migrations – директория, в которой хранятся автоматически сгенерированные модули, обеспечивающие последовательное и согласованное применение изменений к базе данных с использованием встроенных средств Django;

— models.py – модуль, в котором объявляются модели базы данных, представляемые в качестве классов Python, наследуемых от специального класса django.db.models.Model, на основе которых генерируются модули миграций и, в конечном счёте, таблицы в базе данных. Эти модели в дальнейшем используются в запросах к базе данных как с помощью ORM, так

и «сырых» запросов;

— serializers.py – модуль, в котором объявляются специальные классы – сериализаторы, которые обеспечивают автоматизированную обработку (в том числе валидацию) данных для передачи из обработчика запросов (эндпоинта) в базу данных и наоборот;

— urls.py – модуль, который определяет URL-маршруты для каждого эндпоинта, которые впоследствии импортируются в глобальный модуль urls;

— views.py – модуль, в котором реализуются обработчики запросов.

Приложения в некоторых случаях могут использовать модули из других приложений.

Для составления swagger-документации для каждого эндпоинта используется параметризованный декоратор extend_schema, применяемый к каждому методу-обработчику и в качестве параметров принимающий описание эндпоинта, формата его параметров, возможных ответов и их примеры.

Управление веб-приложением (запуск, создание и выполнение миграций) происходи из командной строки с помощью модуля manage.py, расположенного в корне проекта. Вся система запускается локально с помощью технологий Docker и docker-compose.

2.2 Структура базы данных

Выбранная структура базы данных представлена на рисунке 1. Среди реализованных сущностей можно выделить следующие:

1) users – основная таблица, отвечающая за хранение сведений о пользователях системы, включает в себя следующие поля:

— username – уникальное имя пользователя;

— password_hash – хэш пароля;

— first_name, last_name и description – имя, фамилия и описание;

— role – роль пользователя (admin или user);

— is_active и is_public – показатели активности и доступности профиля

пользователя для просмотра другими;

— last_login – время последнего входа в систему;

2) auth_tokens – таблица, хранящая токены аутентификации, выдаваемые пользователям при входе в систему, включает поля:

— key – токен;

— expires_at – время истечения срока действия токена;

— is_active – показатель действительности токена;

— user_id – ссылка на владельца токена;

3) goals – таблица, хранящая все цели, включает поля:

— user_id – ссылка на владельца;

— title – название цели;

— description – описание цели;

— category_id – ссылка на категорию цели;

— target_value – целевой показатель;

— deadline – крайний срок достижения цели;

— is_completed и is_public – аналогично users;

4) goal_progresses – таблица, хранящая записи о прогрессе пользователя по достижению цели, включает поля:

— goal_id – ссылка на цель;

— progress_date – дата, в которую достигнут данный прогресс;

— current_value – текущее значение целевого показателя;

— notes – примечания;

5) habits – таблица, хранящая все привычки, аналогична goals, но вместо полей target_value, deadline и is_completed включает следующие:

— frequency_type – число, характеризующая длительность промежутка (в условных единицах времени), который считается периодом выполнения действий в рамках привычки;

— frequency_value – число повторений действия в течение периода;

— is_active – показатель активности привычки;

6) habit_logs – таблица, хранящая записи о факте соблюдения или несоблюдения привычки, аналогична goal_progresses, вместо поля current_value содержит поле status, принимающее следующие значения:

— completed – соблюдено;

— skipped – пропущено по уважительной причине (не рассматривается в статистике);

— failed – выполнение провалено;

7) categories – таблица, хранящая все категории, применимые к целям, привычкам и челленджам, включает поля name и description (название и описание категории);

8) challenges – таблица, хранящая данные о всех челленджах, включает поля name, description и is_active (название, описание, и состояние челленджа), а также start_date и end_date – даты начала и конца соревнования. С этой таблицей с помощью дополнительных таблиц связаны по связи «многие ко многим» категории, присущие челленджу, и цели, участвующие в челлендже. В таблице goal_challenges также хранится момент присоединения цели к соревнованию;

9) subscription – таблица, связывающая пользователей в качестве подписчика и того, на кого подписаны, по связи «многие ко многим», также хранит момент создания подписки;

10) audit_logs и batch_logs – таблицы, хранящие журнал изменения базы данных (для обычных операций и отдельно для массового импорта данных).

Более подробное описание моделей и их взаимодействия приводится далее в описании приложений.

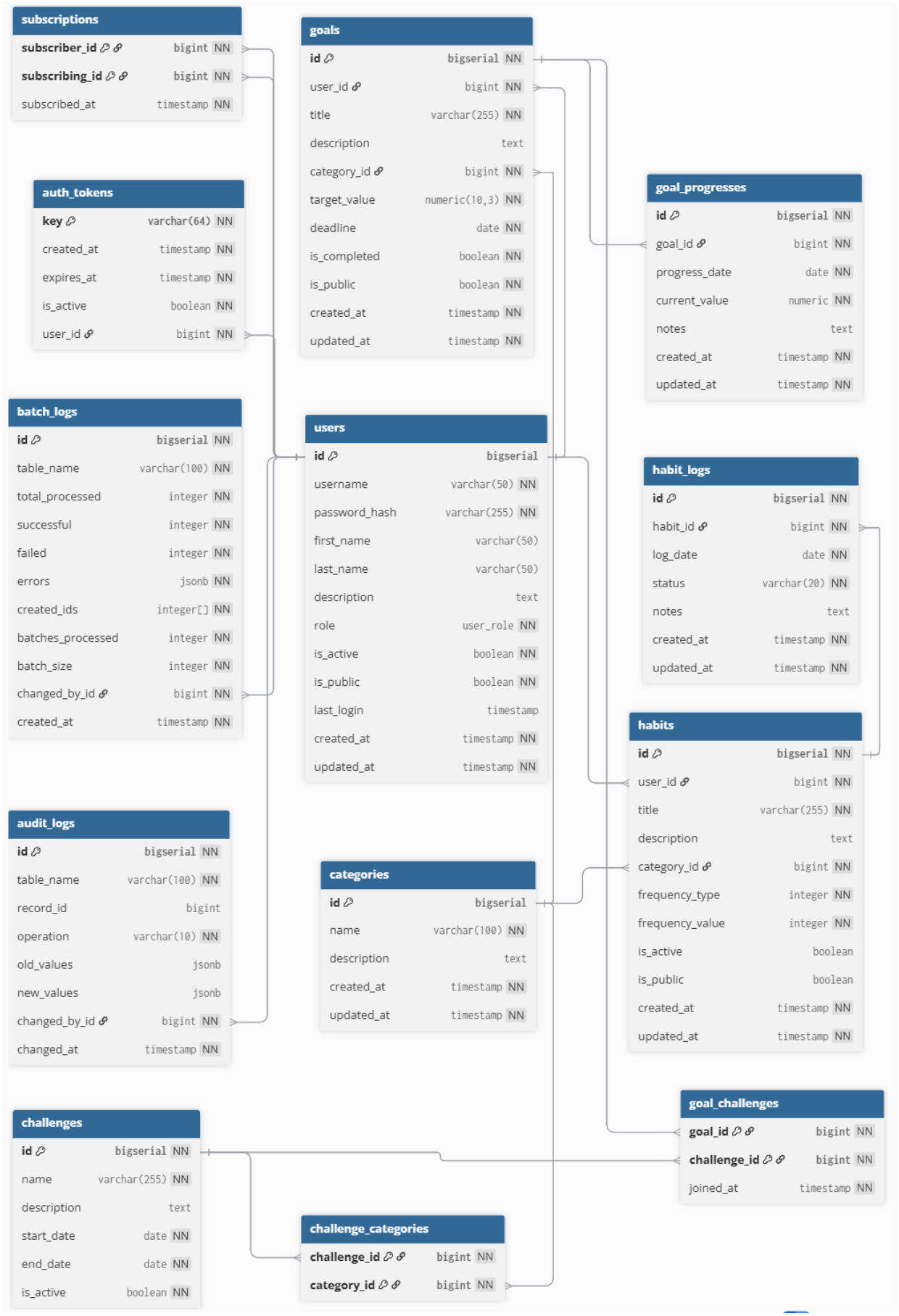

Рисунок 1 – Схема базы данных

2.3 Приложение core

Приложение отвечает за непосредственную работу с пользователями и их авторизацией. Базовые эндпоинты по созданию, удалению, полному и частичному обновлению, массовому и единичному получению данных о пользователях здесь и далее реализованы с помощью наследования класса ViewSet, который обеспечивает автоматическое выполнение указанных операций. Единственное, что требуется определить явно, это алгоритм валидации данных (в сериализаторах), документацию к эндпоинтам и распределение прав доступа. Пагинация при множественном получении также обеспечивается автоматически, для чего определяется класс со значениями имён параметров пагинации (например, limit и offset), значениями limit – максимальным и по умолчанию.

Права доступа определяются через специальные поля класса authentication_classes и permission_classes, которым присваиваются списки классов, отвечающих за аутентификацию и доступ. Эти классы реализованы в подмодуле authentication основного модуля. Класс аутентификации – TokenAuthentication – обеспечивает поиск и получения токена из заголовка http-запроса, проверку его наличия и активности в базе данных, и в случае успеха, передачу объекта аутентифицированного пользователя на проверку доступа. Классы для доступа представлены HasValidToken, IsAdminOrSelf, IsAdmin. Логика их работы заключается для первого в проверке наличия токена и того, что он не истёк, для второго в том, является ли пользователь админом или владельцем того объекта, к которому он пытается получить доступ, для третьего в простой проверке наличия административных прав. Эти классы используются во всех приложениях.

При создании пользователя используется в том числе метод класса модели set_password, который с помощью библиотеки bcrypt преобразует пароль в хэш, которые сохраняется в базе данных. Создание пользователя – единственный эндпоинт системы, не требующий для доступа ауентификации.

Имя пользователя и пароль при создании и обновлении валидируются на

предмет использования только разрешённых символов. Пароль обновляется с помощью отдельного эндпоинта, так же, как и роль, причём роль изменить может только админ.

Вход в систему выполняется посредством проверки соответствия передаваемых имени и пароля, создания нового токена в таблице auth_tokens и возврата этого токена в качестве ответа клиенту. Далее этот токен может передаваться клиентом в заголовке запросов, например, с помощью интерфейса Swagger. Выход осуществляется путём установки для токена текущего пользователя показателя активности в false. Удаление деактивированных токенов для админов предусмотрено в качестве отдельного эндпоинта.

2.4 Приложения goals и habits

Эти приложения одновременно отвечают за обработку целей и состояний их прогресса, привычек и записей об их соблюдении. Для всех таблиц стандартные операции также реализованы через ViewSet.

Пользователи могут создавать и обновлять записи только у себя, а получать только свои и публичные записи других пользователей. Админы (здесь и далее) могут выполнять абсолютно любые запросы.

Кроме того, нельзя создать состояние прогресса у цели с истекшим дедлайном или у достигнутой цели, невозможно изменить владельца или категорию цели или привычки (что сделано для корректной работы системы челленджей).

Все поля, кроме описаний и примечаний, у рассматриваемых таблиц являются обязательными, в том числе и связь с конкретной категорией.

Среди дополнительных эндпоинтов можно выделить получение всех целей или привычек, имеющих данную категорию или данного владельца. Дополнительные эндпоинты реализуются либо как новые (не переопределённые) методы ViewSet, либо как методы отдельных классов – наследников APIView.

2.5 Приложение subscriptions

Приложение отвечает за создание и удаление подписок (записей в таблице subscriptions). Не допускается создание подписки на самого себя и дублирование подписок (id подписчика и объекта подписки выступают в роли составного ключа). Пользователь может подписать или отписать только самого себя, просматривать подписки только публичных пользователей.

Приложение также обеспечивает получение списков объектов пользователей-подписчиков и объектов подписки у данного пользователя, а также проверку наличия конкретной подписки.

2.6 Приложение challenges

Приложение обеспечивает полный цикл работы системы челленджей: управление ими, присоединение целей и получение базовой аналитики (в том числе, рейтинга лидеров). Создавать челленджи и управлять ими (помимо участия) могут только админы. В челленджах участвуют непосредственно цели, а пользователи участвуют косвенно (то есть один пользователь может участвовать сразу с несколькими целями).

Все челленджи построены на едином принципе: лидирует цель, у которой минимальная разница между целевым значением и ближайшим к целевому значением прогресса. Отличия заключаются в категориях, которые имеет челлендж, и, следовательно, которые могут иметь участвующие цели, а также в сроках проведения.

В челленджах могут участвовать только публичные недостигнутые цели публичных пользователей, только в активных челленджах. Присоединение возможно только в период между началом и концом соревнования.

Специальные эндпоинты обеспечивают выборку записей по взаимной принадлежности, например, цели и пользователи, участвующие в данном челлендже, челленджи, в которых участвует данный пользователь или которым присуща данная категория. В отличие от предыдущих описанных эндпоинтов, в которых взаимодействие с базой данных происходило с помощью ORM,

данные запросы выполняются в «сыром» виде, с использованием метода Model.objects.raw, с безопасной передачей параметров в качестве аргументов метода. Пример запроса для последнего эндпоинта приведён в листинге 1.

Листинг 1

```
challenges = Category.objects.raw('''
                SELECT c.id, c.name, c.description, c.target_value,
                c.start_date, c.end_date, c.is_active,
                c.created_at, c.updated_at
                FROM challenges c
                JOIN challenge_categories cc ON c.id = cc.challenge_id
                WHERE cc.category_id = %s''', (category_id,))

paginator = self.pagination_class()
page = paginator.paginate_queryset(challenges, request)

if page is not None:
       serializer = ChallengeSerializer(page, many=True)
       return paginator.get_paginated_response(serializer.data)

serializer = ChallengeSerializer(challenges, many=True)
return Response(serializer.data)
```

На этом примере видно, что использование метода raw (а не connection.cursor().execute, например), обеспечивает при использовании сериализатора автоматическое преобразование результата «сырого» запроса к пригодному для отправки клиенту виду всего за несколько строк.

Ключевыми эндпоинтами приложения являются получения рейтинга целей и пользователей в порядке лидирования в данном челлендже. Для пользователей соревновательным показателем является минимальная из минимальных разниц (описанных выше) между целевым и текущим показателями, среди всех его участвующих целей. Запрос, связанный с рейтингом пользователей, приведён в листинге 2. В обоих запросах используется функция calculate_goal_progress, возвращающая минимальные разницы между целевым и текущим показателями целей.

Листинг 2

```
WITH user_best_diffs AS (
   SELECT
```

```
        g.user_id,
        u.username,
        MIN(calculate_goal_progress(g.id)) AS user_best_min_diff,
        COUNT(DISTINCT g.id) AS total_goals,
        COUNT(DISTINCT CASE WHEN EXISTS (
       SELECT 1 FROM goal_progresses gp
       WHERE gp.goal_id = g.id
       ) THEN g.id END) AS goals_with_progress
       FROM goals g
      JOIN goal_challenges gc ON g.id = gc.goal_id
      JOIN users u ON g.user_id = u.id
      WHERE gc.challenge_id = %s
     AND g.is_public = true
    GROUP BY g.user_id, u.username
)
 SELECT
 ROW_NUMBER() OVER (ORDER BY user_best_min_diff, username) AS
user_rank,
 user_id as id,
username,
user_best_min_diff,
total_goals,
goals_with_progress,
total_goals - goals_with_progress AS goals_without_progress
FROM user_best_diffs
ORDER BY user_rank;
```

2.7 Приложение categories

Приложение обеспечивает простую функциональность по управлению категориями, реализованную целиком в рамках одного ViewSet. Управлять категориями могут только админы.

2.8 Приложение audit

Приложение обеспечивает создание и управление журналами аудита. Аудит одиночных изменений во всех основных таблицах обеспечивается с помощью триггеров, определённых для каждой таблицы через единую триггерную функцию, которая записывает имя таблицы, операции, id записи (при наличии, при отсутствии – null), старые и новые значения изменённой строки, а также id пользователя, совершающего изменения. Для обеспечения последнего, id пользователя устанавливается глобально в момент

аутентификации через класс TokenAuthentication, а после запроса сбрасывается через специальный модуль middleware.

Таблица batch_logs будет описана отдельно в разделе о массовом импорте. Для журналов аудита организованы простые эндпоинты массового получения и одиночного удаления.

2.9 Приложение analytics

Приложение не определяет новые модели и служит исключительно для выполнения сложных аналитических запросов к базе данных. Для организации запросов реализованы три представления (category_detailed_analytics, challenge_basic_analytics и user_progress_analytics), представляющие расширенную информацию о категориях, челленджах и пользователях соответственно. Например, для категорий это количества связанных целей, привычек, челленджей, их средние показатели в данной категории, популярность категории (основанная на взвешенных числах участвующих в категории целях, привычках, и челленджах, и косвенно, пользователях). Для построения представлений используются SQL-функции, необходимые для расчёта отдельных показателей. Пример функции, отвечающей за расчёт процентного отношения количества успешных соблюдений привычки к общему числу, а также создание представления user_progress_analytics приведены в листинге 3.

Листинг 3

```
CREATE OR REPLACE FUNCTION calculate_habit_consistency(habit_id_param
BIGINT)
RETURNS NUMERIC AS $$
DECLARE
    total_relevant_logs INTEGER;
    completed_logs INTEGER;
    consistency NUMERIC;
BEGIN
    SELECT
        COUNT(*) FILTER (WHERE status IN ('completed', 'failed')),
        COUNT(*) FILTER (WHERE status = 'completed')
    INTO total_relevant_logs, completed_logs
    FROM habit_logs
```

```
    WHERE habit_id = habit_id_param;
    IF total_relevant_logs > 0 THEN
        consistency := (completed_logs::NUMERIC / total_relevant_logs)
* 100.0;
    ELSE
        consistency := 0.0; -- Нет записей со статусами
    END IF;
    RETURN consistency;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION get_user_habits_consistency()
RETURNS TABLE(
    user_id BIGINT,
    avg_habit_consistency NUMERIC
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        h.user_id,
        AVG(calculate_habit_consistency(h.id)) as
avg_habit_consistency
    FROM habits h
    GROUP BY h.user_id;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE VIEW user_progress_analytics AS
SELECT
    u.id as id,
    u.username,
    u.created_at as user_joined,

    COUNT(DISTINCT g.id) as total_goals,
    COUNT(DISTINCT CASE WHEN g.is_completed THEN g.id END) as
completed_goals,
    COUNT(DISTINCT CASE WHEN g.deadline < CURRENT_DATE AND NOT
g.is_completed THEN g.id END) as overdue_goals,
    AVG(calculate_goal_completion_percentage(g.id)) as
avg_goal_progress,

    COUNT(DISTINCT h.id) as total_habits,
    COUNT(DISTINCT CASE WHEN h.is_active THEN h.id END) as
active_habits,
    COALESCE(uhc.avg_habit_consistency, 0) as avg_habit_consistency,

    COUNT(DISTINCT gc.challenge_id) as total_challenges_participated,
    COUNT(DISTINCT CASE WHEN c.end_date < CURRENT_DATE THEN c.id END)
as completed_challenges,

    COUNT(DISTINCT s1.subscriber_id) as subscribers_count,
    COUNT(DISTINCT s2.subscribing_id) as subscribing_count

FROM users u
LEFT JOIN goals g ON g.user_id = u.id
LEFT JOIN habits h ON h.user_id = u.id
LEFT JOIN get_user_habits_consistency() uhc ON uhc.user_id = u.id
```

```
LEFT JOIN goal_challenges gc ON gc.goal_id = g.id
LEFT JOIN challenges c ON c.id = gc.challenge_id
LEFT JOIN subscriptions s1 ON s1.subscribing_id = u.id
LEFT JOIN subscriptions s2 ON s2.subscriber_id = u.id
WHERE u.is_public = true
GROUP BY u.id, u.username, u.created_at, uhc.avg_habit_consistency;
```

На основании созданных представлений в эндпоинтах приложения выполняются простые запросы, обеспечивающие получение рейтинга категорий и челленджей по популярности, рейтинга пользователей по количеству достигнутых целей и проценту соблюдения привычек, а также по количеству подписчиков. SQL-код хранится в отдельных файлах и загружается в базу данных в отдельно созданной миграции с помощью функции RunSQL.

2.10 Массовый импорт

Массовый импорт (batch-import) данных в основные таблицы (users, goals, goal_progresses, habits, habit_logs, subscriptions, categories) был реализован, главным образом, для дальнейшего наполнения системы тестовыми данными. Для этого в соответствующих приложениях был добавлен отдельный эндпоинт, принимающий список создаваемых записей и параметр, определяющий размер «батча». Далее предлагаемые для создания записи поочерёдно валидируются, после чего прошедшие проверку записи создаются одновременно с помощью метода Model.objects.bulk_create, причём одновременно создаётся число записей, равное размеру «батча».

На время импорта триггеры логгирования отключаются, чтобы не замедлять выполнения операции, делая лог по каждой записи. Вместо этого, запись о массовом импорте помещается в таблицу batch_logs, в которой указываются число предложенных записей, число успешно созданных и не созданных записей, список ошибок и данных записей, которые не удалось создать, список id созданных записей. Эта же информация возвращается клиенту в качестве ответа. Выполнять массовый импорт могут только админы.

При создании челленджей, в частности, в параметрах запроса указываются списки id категорий и целей, ассоциируемых с этим челленджем.

Соответствующие записи в таблицах goal_challenges и challenge_categories создаются одновременно в рамках этого же запроса.

2.11 Анализ производительности

При создании таблиц на основе моделей, Django автоматически добавляет индексы к полям внешних ключей. В подавляющем большинстве случаев при поиске и фильтрации используется именно первичные и внешние ключи, а также булевы поля (в особенности is_public), которые принимают всего 2 значения, и, следовательно, к ним не имеет смысла добавлять индексы. Попытки добавить индексы к немногочисленным полям, используемым при сортировке и группировке (даты или имена) не показали какого-либо увеличения производительности, в том числе при значительном наполнении таблиц (по 5000 записей).

Таким образом, было принято решение не добавлять дополнительных индексов и ограничиться теми, которые были созданы автоматически. В качестве эксперимента были временно удалены некоторые из них, после чего произведено сравнение производительности некоторых показательных запросов (использующих фильтрацию по индексируемому полю) без индексов и с их наличием.

Для эксперимента выбрано запрос, участвующий в получении списка челленджей, в которых принимает участие данный пользователь, его тело выполнено в качестве запроса с EXPLAIN ANALYTICS дважды. На время первого выполнения удалён индекс на внешний ключ таблицы goals.user_id. Результат выполнения без и с наличием индексов приведён в листинге 4.

Листинг 4

```
EXPLAIN ANALYSE
SELECT c.id, c.name, c.description,
       c.start_date, c.end_date, c.is_active, c.created_at,
c.updated_at
FROM challenges c
JOIN goal_challenges gc ON c.id = gc.challenge_id
JOIN goals g ON g.id = gc.goal_id
WHERE g.user_id = 7777;
```

```
-- Без индекса
-- Planning Time: 6.141 ms
-- Execution Time: 3.211 ms

-- С индексом
-- Planning Time: 0.606 ms
-- Execution Time: 0.205 ms
```

Использование индекса дало ускорение выполнения запроса примерно в 16 раз. Аналогичный результат (ускорение в 6 раз), представленный в листинге 5, показало выполнения тела функции calculate_goal_progress (см. листинг 2).

Листинг 5

```
EXPLAIN ANALYSE
SELECT MIN(ABS(gp.current_value - g.target_value))
    FROM goal_progresses gp
    JOIN goals g ON g.id = gp.goal_id
    WHERE gp.goal_id = 8888;

-- Без индекса
-- Planning Time: 0.851 ms
-- Execution Time: 1.172 ms

-- С индексом
-- Planning Time: 0.300 ms
-- Execution Time: 0.173 ms
```

Таким образом, можно утверждать, что во многих случаях (в частности, в нашем) Django эффективно автоматизирует создание большинства важных индексов (с одной стороны, поиск по внешним ключам чаще всего используется в запросах, а с другой стороны, автоматическое создание индексов на внешние ключи – функционал именно Django, так как PostgreSQL автоматически индексирует только первичные ключи).

3 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

3.1 Контейнеризация

Система запускается в изолированном контейнере с использованием систем Docker и docker-compose. Контейнер создаётся согласно Dockerfile, на основе образа Python, с установкой компилятора gcc и системы postgres-client, а также зависимых пакетов Python.

Непосредственный запуск контейнеров, один из которых создан по Dockerfile, а другой построен на основе образа postgres, определён в файле docker-compose.yaml, где для каждого контейнера устанавливатся, в том числе входящий и выходящий порты. Также при запуске выполняются миграции в Django, и устанавливаются настройки системы (аутентификационные данные postgres, режим работы системы (debug mode), секретный ключ, используемый в Django). Эти данные берутся из окружения, например, из файла .env. Последовательность команд для сборки и запуска системы представлена в листинге 6.

Листинг 6

```
docker-compose build --no-cache
docker-compose up
docker-compose exec web bash (для перехода в консоль контейнера)
```

3.2 Тестирование

В рамках тестирования разработанной системы был создан сценарий на языке Python, который генерирует и при помощи эндпоинтов массового импорта наполняет базу данных записями. При генерации использовались библиотеки requests и faker [4] (для генерации тестовых данных). Для создания записей в зависимых таблицах использовались id созданных на предыдущих этапах записей, что позволило выполнить всю сессию без ошибок. Было оценено время выполнения каждой из операций. Наиболее продолжительное время заняло создание пользователей, что связано с многоступенчатой

валидацией и хэшированием паролей. Результаты (лог) загрузки приведены в листинге 7.

Листинг 7

```
users: 500 записей, время: 157.51 сек
categories: 30 записей, время: 0.25 сек
goals: 500 записей, время: 2.15 сек
habits: 500 записей, время: 2.16 сек
goal_progresses: 5000 записей, время: 15.50 сек
habit_logs: 5000 записей, время: 14.49 сек
challenges: 500 записей, время: 2.35 сек
subscriptions: 1000 записей, время: 1.53 сек
```

Также были протестированы на предмет работоспособности отдельные эндпоинты. Примеры запросов и ответов приведены в листинге 8.

Листинг 8

```
curl -X 'GET' \
  'http://127.0.0.1:8080/api/users/?limit=1&offset=0' \
  -H 'accept: application/json' \
  -H 'Authorization: 699556ee76c094dc67ea81c5ffbb6a5d61bcb0bff68ca1eedb85ea4207edbb2d'

200 OK
{
  "count": 601,
  "next": "http://127.0.0.1:8080/api/users/?limit=1&offset=1",
  "previous": null,
  "results": [
    {
      "id": 9,
      "username": "abc",
      "first_name": "John",
      "last_name": "Doe",
      "description": "strhhghghing",
      "role": "admin",
      "is_active": true,
      "is_public": true,
      "created_at": "2025-12-04T12:24:43.655906+03:00",
      "updated_at": "2025-12-19T16:12:19.349936+03:00"
    }
  ]
}


curl -X 'PATCH' \
  'http://127.0.0.1:8080/api/users/2583/' \
  -H 'accept: application/json' \
  -H 'Authorization: 699556ee76c094dc67ea81c5ffbb6a5d61bcb0bff68ca1eedb85ea4207edbb2d'
\
  -H 'Content-Type: application/json' \
  -H 'X-CSRFTOKEN: vVPTRUp91epsWda0r7t3DT6OXmbkQAuUlwq1qulMdpCwl4uRGC6aLp8WiWFD99wV' \
  -d '{
  "first_name": "Александр",
  "is_public": false
}'
```

```
200 OK
{
  "id": 2583,
  "username": "dmitrisuvorov1542",
  "first_name": "Александр",
  "last_name": "Попов",
  "description": "Прощение песенка приходить следовательно. Зачем деловой одиннадцать
аллея.",
  "role": "admin",
  "is_active": true,
  "is_public": false,
  "created_at": "2025-12-19T16:14:07.956528+03:00",
  "updated_at": "2025-12-19T16:37:43.921790+03:00"
}

curl -X 'DELETE' \
  'http://127.0.0.1:8080/api/habits/2516/' \
  -H 'accept: */*' \
  -H 'Authorization: 699556ee76c094dc67ea81c5ffbb6a5d61bcb0bff68ca1eedb85ea4207edbb2d'
\
  -H 'X-CSRFTOKEN: vVPTRUp91epsWda0r7t3DT6OXmbkQAuUlwq1qulMdpCwl4uRGC6aLp8WiWFD99wV'

204 OK


curl -X 'POST' \
  'http://127.0.0.1:8080/api/habits/log/' \
  -H 'accept: application/json' \
  -H 'Authorization: 699556ee76c094dc67ea81c5ffbb6a5d61bcb0bff68ca1eedb85ea4207edbb2d'
\
  -H 'Content-Type: application/json' \
  -H 'X-CSRFTOKEN: vVPTRUp91epsWda0r7t3DT6OXmbkQAuUlwq1qulMdpCwl4uRGC6aLp8WiWFD99wV' \
  -d '{
  "habit": 0,
  "log_date": "2025-12-19",
  "status": "completed",
  "notes": "string"
}'

400 BAD REQUEST
{
  "habit": [
    "Привычка не найдена"
  ]
}


curl -X 'GET' \
  'http://127.0.0.1:8080/api/goals/1/' \
  -H 'accept: application/json' \
  -H 'Authorization: 699556ee76c094dc67ea81c5ffbb6a5d61bcb0bff68ca1eedb85ea4207edbb2d'

404 NOT FOUND
{
  "detail": "No Goal matches the given query."
}

curl -X 'GET' \

'http://127.0.0.1:8080/api/subscriptions/is_subscribed/?subscriber_id=2773&subscribing
_id=2803' \
  -H 'accept: */*' \
  -H 'Authorization: 699556ee76c094dc67ea81c5ffbb6a5d61bcb0bff68ca1eedb85ea4207edbb2d'

200 OK
{
  "is_subscribed": true,
  "subscribed_at": "2025-12-19T13:29:12.700618Z"
}
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового работы была разработана и реализована информационная система для трекинга целей и привычек, полностью соответствующая поставленным задачам и требованиям технического задания.

В ходе выполнения работы спроектирована и реализована нормализованная реляционная база данных на PostgreSQL, включающая 11 взаимосвязанных таблиц с обеспечением целостности данных; разработан полнофункциональный RESTful API с использованием Django REST Framework, охватывающий все аспекты предметной области; реализована система аутентификации и авторизации на основе токенов с разделением прав доступа между обычными пользователями и администраторами; разработан механизм массового импорта данных с оптимизацией производительности через batch-обработку; внедрена система аудита изменений с использованием триггеров PostgreSQL; обеспечена контейнеризация решения с помощью Docker и docker-compose для простоты развертывания; проведено комплексное тестирование системы с генерацией тестовых данных, подтвердившее работоспособность всех компонентов.

Было уделено внимание вопросам производительности: проведен анализ использования индексов, доказана эффективность автоматически создаваемых Django индексов для внешних ключей.

Разработанная система подготовлена к стороннему использованию, её исходный код доступен на платформе GitHub [5].

Все поставленные цели исследования достигнуты, получен практический опыт проектирования и реализации полноценной информационной системы с использованием современных веб-технологий и систем управления базами данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Приложение «Habit Tracker: Цели» // App Store – URL: https://apps.apple.com/us/app/habit-tracker-цели/id6459830396?l=ru (дата обращения: 19.12.2025)

2. Home - Django REST framework // Django Community – URL: https://www.django-rest-framework.org/ (дата обращения: 19.12.2025)

3. drf-spectacular documentation // T. Franzel – URL: https://drf-spectacular.readthedocs.io/en/latest/ (дата обращения: 19.12.2025)

4. Welcome to Faker’s documentation! — Faker 39.0.0 documentation // Daniele Faraglia – URL: https://faker.readthedocs.io/en/master/ (дата обращения: 19.12.2025)

5. MMVlasko/bdcwn // Власко М. М. – URL: https://github.com/MMVlasko/bdcw (дата обращения: 19.12.2025)