МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет Прикладной информатики
Образовательная программа «Мобильные и сетевые технологии»

ОТЧЕТ
по домашней работе №1

по теме:

ТЕХНИЧЕСКИЙ ДИЗАЙН МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ ФИТНЕС-ПЛАТФОРМЫ

Студент: В.С. Корчагин

Преподаватель:
Преподаватель ФПИН Д.И. Добряков

Санкт-Петербург 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель: разработать технический дизайн микросервисной архитектуры фитнес-платформы.

Задачи:

- определить состав микросервисов и их зоны ответственности;
- спроектировать взаимодействие сервисов через REST API с использованием JWT-аутентификации;
- продумать внутреннюю компонентную структуру каждого сервиса (контроллеры, бизнес-логика, хранение данных);
- разработать модель базы данных для каждого микросервиса;
- проиллюстрировать ключевые сценарии работы системы с помощью диаграмм последовательности.

1 Ход выполнения работы

1.1 Общая архитектура платформы

Платформа построена по микросервисной архитектуре: приложение разделено на несколько изолированных сервисов, каждый из которых отвечает за свой функциональный домен. Выделены следующие основные микросервисы:

- Auth-Service – сервис аутентификации и управления пользователями. Отвечает за регистрацию новых пользователей, вход в систему, хранение учетных записей и ролей, а также выдачу JWT-токенов для доступа к другим сервисам.
- Workout-Service – сервис тренировок и тренировочных планов. Управляет каталогом упражнений и программ тренировок, позволяет создавать планы тренировок и хранит связи между планами и упражнениями.
- Progress-Service – сервис отслеживания прогресса пользователей. Сохраняет показатели здоровья и активности пользователя (например, вес, шаги, потребление воды) и информацию о том, какие тренировочные планы пользователь выполняет (активные планы).
- Order-Service – сервис заказов и платежей. Обрабатывает заказы пользователей (например, покупка платных программ тренировок или товаров) и информацию об их оплате.
- Blog-Service – сервис блог-платформы. Позволяет публиковать посты о фитнесе и здоровье, а также добавлять к ним комментарии.

Каждый микросервис полностью изолирован: у него своя кодовая база, собственная база данных, API и модель данных. Прямого доступа к данным других сервисов нет – любые межсервисные взаимодействия осуществляются только через REST API вызовы по HTTP. Для внутренних запросов используется библиотека HTTP-клиента (в реализации – axios), что обеспечивает слабую связанность и независимость компонентов.

JWT-токен (JSON Web Token) используется для авторизации между сервисами. Auth-Service при успешном логине выдает клиенту JWT. Далее клиент прикрепляет этот токен к запросам в другие сервисы. Каждый сервис проверяет валидность токена (например, с помощью общего секретного ключа или публичного ключа) и извлекает из него идентификатор пользователя и роль, чтобы авторизовать запрос. Таким образом, аутентификация централизована, но проверка токена выполняется на уровне каждого сервиса. Это устраняет необходимость общей сессии и позволяет сервисам оставаться статическими и масштабируемыми.

Обмен данными между сервисами осуществляется через публичные REST API. Если одному сервису требуется информация, относящаяся к другому домену, он делает HTTP-запрос к соответствующему сервису вместо прямого обращения к его базе данных. Например, Blog-Service не хранит подробных данных о пользователях – при необходимости он запрашивает Auth-Service, чтобы получить информацию об авторе поста или комментария по его user_id. Аналогично Order-Service обращается к Auth-Service для получения сведений о пользователе, оформившем заказ, а Progress-Service запрашивает Workout-Service для получения деталей тренировочного плана по его идентификатору. Такой подход гарантирует, что каждый сервис управляет только своими данными, а междоменные связи реализованы через сетевые вызовы. Это исключает жесткие межмодульные зависимости на уровне базы данных и сохраняет целостность внутри каждого сервиса.

На рисунке Рисунок 1 представлена схема общей архитектуры, демонстрирующая основные сервисы, их базы данных и взаимодействие между ними по REST. Стрелками обозначены направления ключевых запросов.

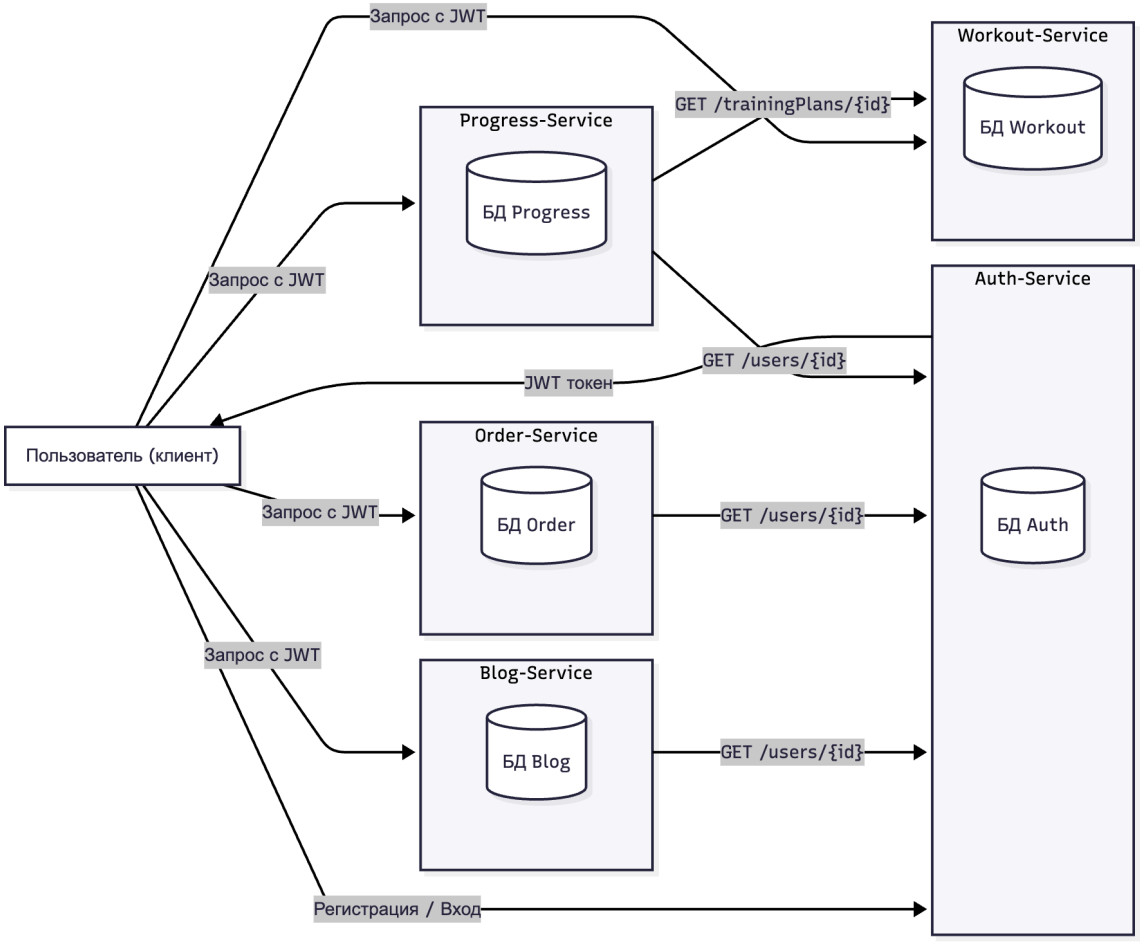

Рисунок 1 — Схема общей архитектуры платформы: микросервисы, их базы данных и взаимодействие (REST API)

1.2 Компонентная структура микросервисов

В каждом сервисе применяется единый шаблон проектирования внутренней структуры. Web-слой контроллеров обрабатывает входящие HTTP-запросы, слой сервисов инкапсулирует бизнес-логику и работу с данными, DTO используются для передачи данных между слоями и валидации, а утилиты выполняют вспомогательные функции (например, хеширование паролей, выдача токенов). Ниже рассмотрена структура компонентов каждого микросервиса.

1.2.1 Auth-Service: аутентификация и пользователи

Ответственность: Auth-Service отвечает за регистрацию новых пользователей, их аутентификацию (логин) и управление учетными записями. Он выдает JWT-токены для доступа к другим сервисам. Также сервис управляет

ролями пользователей (например, обычный пользователь, администратор), определяя права доступа.

Основные компоненты:

- **Контроллеры:** AuthController (регистрация и логин) и UserController (операции с профилями пользователей, получение и изменение информации о пользователе).
- **Сервисы:** AuthService (логика аутентификации – проверка пароля, генерация JWT) и UserService (управление сущностью пользователя – поиск, создание, обновление пользователей). Оба сервиса наследуют базовый класс с общими методами (BaseService с операциями findAll, findOne, create, update, remove).
- **DTO:** объекты для передачи данных, определяющие формат и ограничения входных данных. Например, RegisterUserDTO, LoginDTO (для регистрации и входа) и DTO для сущности пользователя. DTO выполняют валидацию (через библиотеку class-validator) – например, проверяют формат email и длину пароля при регистрации.
- **Утилиты:** PasswordHasher для безопасного хеширования паролей и проверки введенного пароля, JWTUtil для создания JWT-токена при входе и валидации токена. В сервис встроен механизм авторизации запросов: специальный authorizationChecker (например, при использовании routing-controllers), который проверяет JWT из заголовка и решает, допускать ли пользователя к защищенному ресурсу. Он использует JWTUtil для проверки подписи токена и извлечения из него id и роли пользователя.

Диаграмма компонентов Auth-Service представлена на рисунке Рисунок 2.

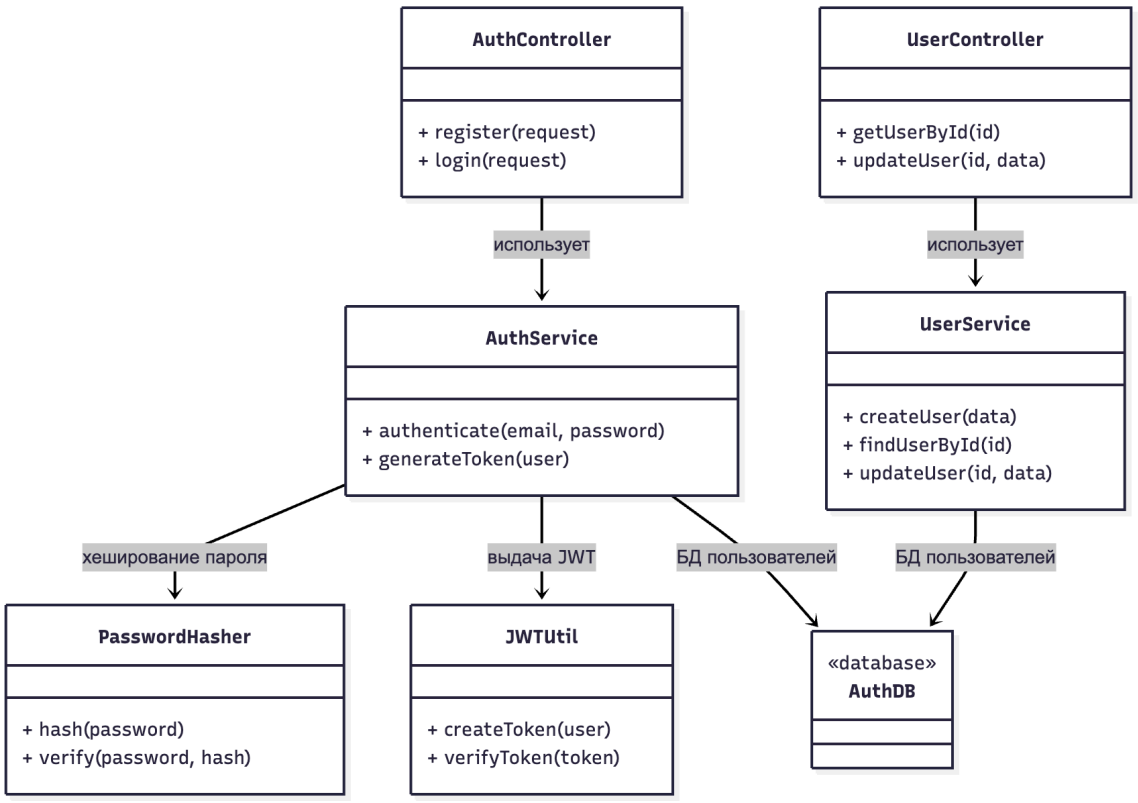

Рисунок 2 — Компонентная структура Auth-Service

База данных Auth-Service содержит таблицы пользователей и ролей. ER-модель приведена на рисунке Рисунок 3. Таблица User включает поля id, name, email, passwordHash и внешний ключ roleId (на Role). Таблица Role хранит возможные роли (id и name). Связь «роль назначается пользователям» указывает, что одному типу роли может соответствовать множество пользователей. Auth-Service автономно хранит данные учетных записей и не зависит от других сервисов.

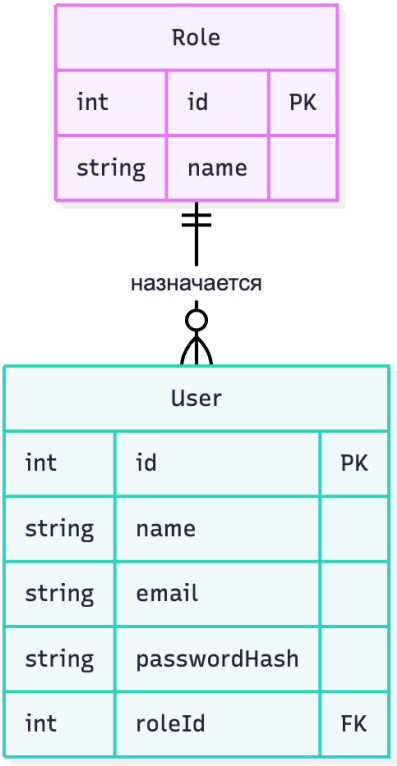

Рисунок 3 — ER-модель базы данных Auth-Service (пользователи и роли)

1.2.2 Workout-Service: тренировки и планы

Ответственность: Workout-Service управляет упражнениями и программами тренировок. В его ведении справочник упражнений (Workout) и тренировочных планов (TrainingPlan), а также состав планов (связи между планами и упражнениями). Сервис предоставляет API для получения списка упражнений, создания и редактирования тренировочных планов, добавления упражнений в план и т.п.

Основные компоненты:

– **Контроллеры:** WorkoutController (CRUD-операции над сущностью упражнения) и TrainingPlanController (управление тренировочными планами: создание нового плана, добавление упражнения в план, получение плана с составом и т.д.).

– **Сервисы:** WorkoutService и TrainingPlanService реализуют бизнес-логику для соответствующих сущностей. Например, TrainingPlanService проверяет корректность добавляемых упражнений в план, может рассчитывать продолжительность программы и др. Оба сервиса могут

использовать методы базового класса (наследовать BaseService) для стандартных операций с БД.

– **DTO:** например, CreateWorkoutDTO, CreateTrainingPlanDTO – описывают формат входных данных и выполняют их проверку (название упражнения, описание и цель плана, список упражнений в плане и др.).
– **Утилиты:** доменные утилиты минимальны (логика в основном внутри сервисов). JWT-аутентификация проверяется на защищенных маршрутах (например, создание упражнений доступно только администратору). Внешние обращения: Workout-Service может не вызывать другие сервисы напрямую, но отвечает на запросы Progress-Service (когда тому нужны сведения о тренировочном плане по id).

Компонентная структура Workout-Service показана на рисунке Рисунок 4.

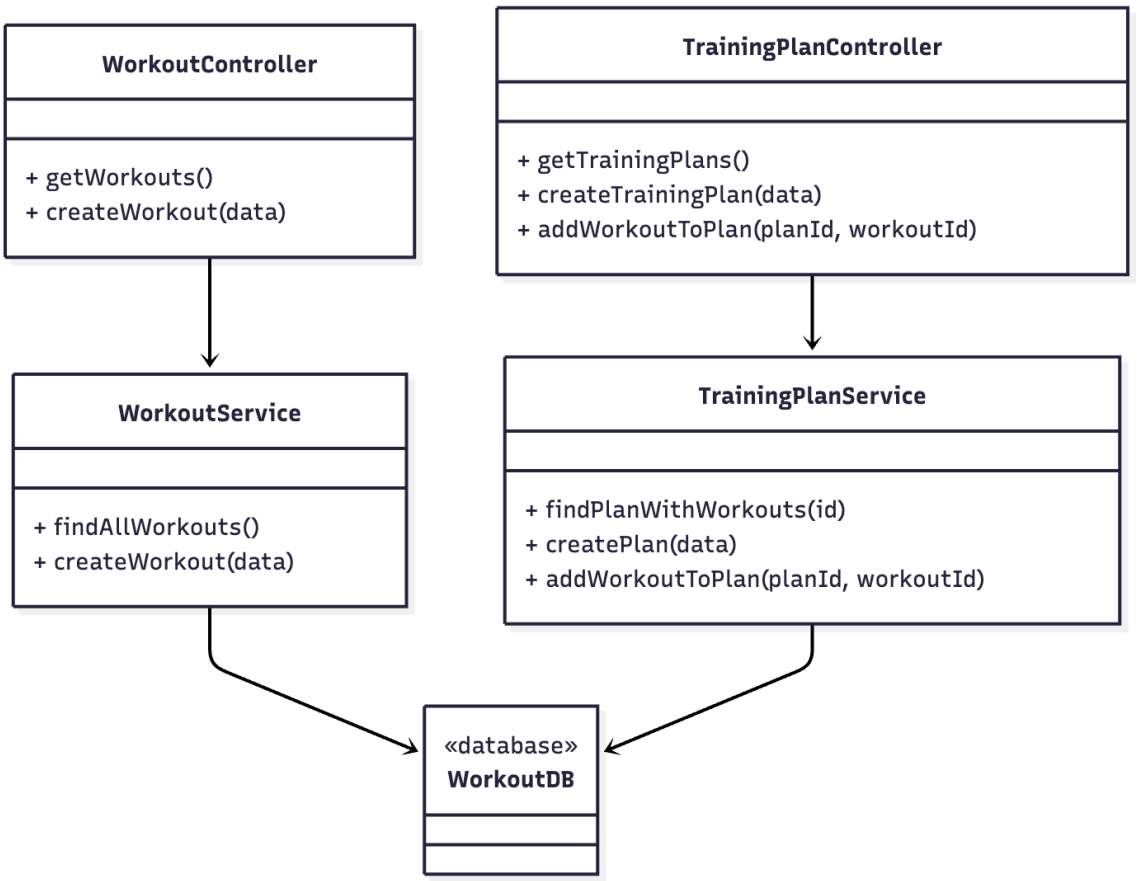

Рисунок 4 — Компонентная структура Workout-Service

База данных Workout-Service включает только данные, относящиеся к тренировкам. Модель данных состоит из таблиц Workout (упражнение: id, name, description), TrainingPlan (тренировочный план: id, name, goal) и про-

межуточной таблицы TrainingPlanWorkout (id, trainingPlanId, workoutId) для связи многие-ко-многим между планами и упражнениями. Связи показаны на рисунке Рисунок 5. Один тренировочный план может включать несколько упражнений, и одно упражнение может входить в разные планы (через таблицу TrainingPlanWorkout). В этой базе нет внешних ключей на пользователей – назначение плана пользователю выполняется в Progress-Service.

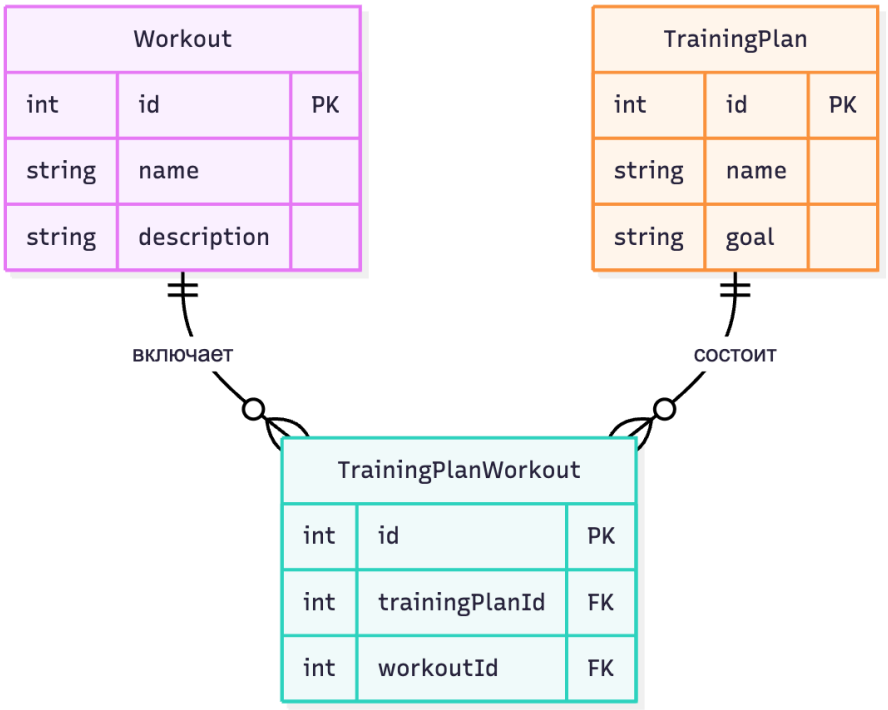

Рисунок 5 — ER-модель базы данных Workout-Service (тренировочные планы и упражнения)

1.2.3 Progress-Service: прогресс пользователя

Ответственность: Progress-Service отслеживает и сохраняет данные о состоянии пользователя и его активности. В него входят показатели здоровья (например, вес, количество шагов, выпитая вода) и сведения о том, какие тренировочные планы использует пользователь (активные планы). Сервис позволяет получать и обновлять прогресс пользователя, а также привязывать пользователя к выбранным тренировочным планам для мониторинга выполнения.

Основные компоненты:

– **Контроллеры:** ProgressController (операции с показателями прогресса: получить текущие данные, обновить вес, шаги и т.д.) и

UserTrainingPlanController (управление активными планами пользователя: назначить план, получить планы пользователя, завершить/удалить план).

- **Сервисы:** ProgressService (логика обновления и расчета прогресса – например, вычисление индекса массы тела на основе веса и роста, если хранится) и UserTrainingPlanService (управляет связью пользователя с планами – добавление нового плана, получение списка планов пользователя, удаление плана). Сервисы работают с собственной базой Progress-Service.
- **DTO:** например, UpdateProgressDTO (для обновления показателей, проверяет, что вес и шаги – положительные числа) и AddTrainingPlanDTO (для назначения плана, проверяет правильность идентификаторов; userId берется из JWT).
- **Утилиты:** Progress-Service может обращаться к внешним сервисам. При добавлении плана пользователю сервис выполняет запрос в Workout-Service (GET /trainingPlans/{id}), чтобы убедиться, что указанный план существует и получить его детали. Кроме того, JWT-токен проверяется на всех защищенных маршрутах Progress-Service, чтобы пользователь мог работать только со своими данными (либо администратор – со всеми).

На рисунке Рисунок 6 изображены основные компоненты Progress-Service.

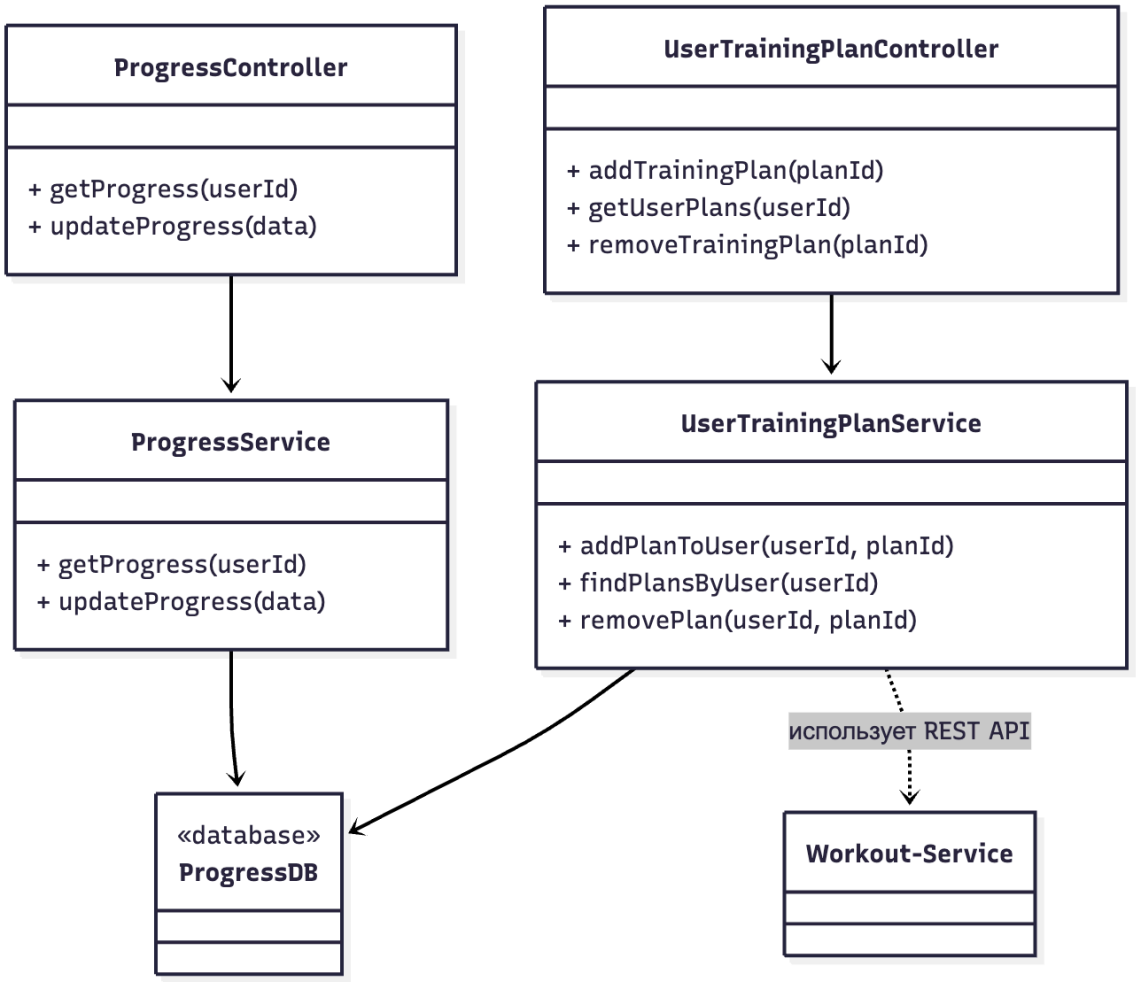

Рисунок 6 — Компонентная структура Progress-Service

База данных Progress-Service представлена двумя сущностями: UserProgress и UserTrainingPlan (рисунок Рисунок 7). В UserProgress хранятся текущие показатели пользователя (id, userId, вес, шаги, вода и др.). В UserTrainingPlan фиксируются назначенные пользователю тренировочные планы (id, userId, trainingPlanId). Внешние связи: userId соответствует идентификатору пользователя в Auth-Service, а trainingPlanId – идентификатору плана в Workout-Service, однако эти поля хранятся как обычные данные (без внешних ключей). При необходимости Progress-Service запрашивает информацию о плане или пользователе во внешних сервисах по этим идентификаторам.

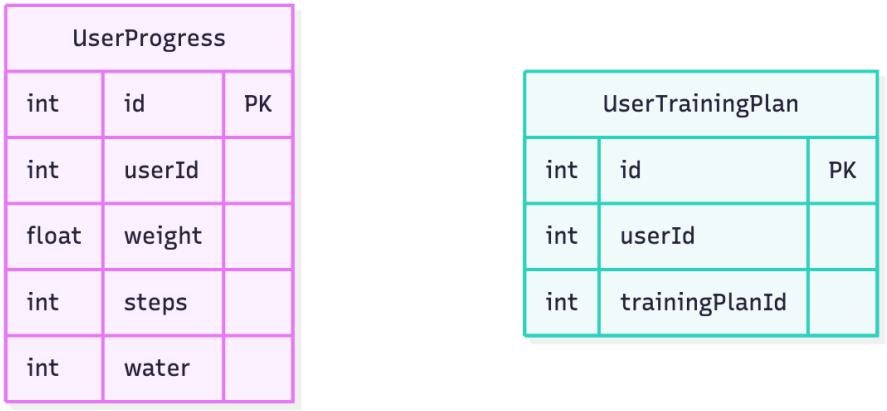

Рисунок 7 — ER-модель базы данных Progress-Service (показатели и планы пользователя)

1.2.4 Order-Service: заказы и платежи

Ответственность: Order-Service отвечает за оформление заказов на платформе и их оплату. В рамках фитнес-платформы это может включать покупку премиум-подписки, платных тренировочных программ или товаров. Сервис хранит заказы (Order) и информацию об их оплате (Payment). Он обеспечивает создание новых заказов, изменение статусов (например, пометку «оплачен») и получение сведений о сделанных заказах.

Основные компоненты:

– **Контроллеры:** OrderController (создание нового заказа, получение информации о заказе, инициирование оплаты; в простом случае функциональность оплаты может быть реализована методом OrderController) и, возможно, отдельный PaymentController для подтверждения платежа и проверки статуса, если бы логика оплаты выносилась отдельно.

– **Сервисы:** OrderService (бизнес-логика работы с заказами – расчёт суммы, создание записи заказа, привязка к пользователю) и PaymentService (логика оплаты – эмуляция платёжного процесса, изменение статуса платежа, взаимодействие с платёжным шлюзом, если предполагалось). Оба сервиса работают с базой данных Order-Service.

– **DTO:** например, CreateOrderDTO (состав заказа, сумма, идентификатор пользователя из JWT) и PaymentDTO (данные об оплате – способ, параметры).
– **Утилиты:** могут включать интеграцию с внешней платёжной системой (в учебном проекте эмулируется), а также утилиту для генерации идентификаторов/номеров заказов. Order-Service может обращаться к Auth-Service для получения информации о пользователе (например, email для отправки чека) или проверки существования пользователя. JWT обеспечивает авторизацию: обычный пользователь может создавать заказ и просматривать свои заказы, администратор — видеть чужие и менять статусы.

Компонентная схема Order-Service приведена на рисунке Рисунок 8.

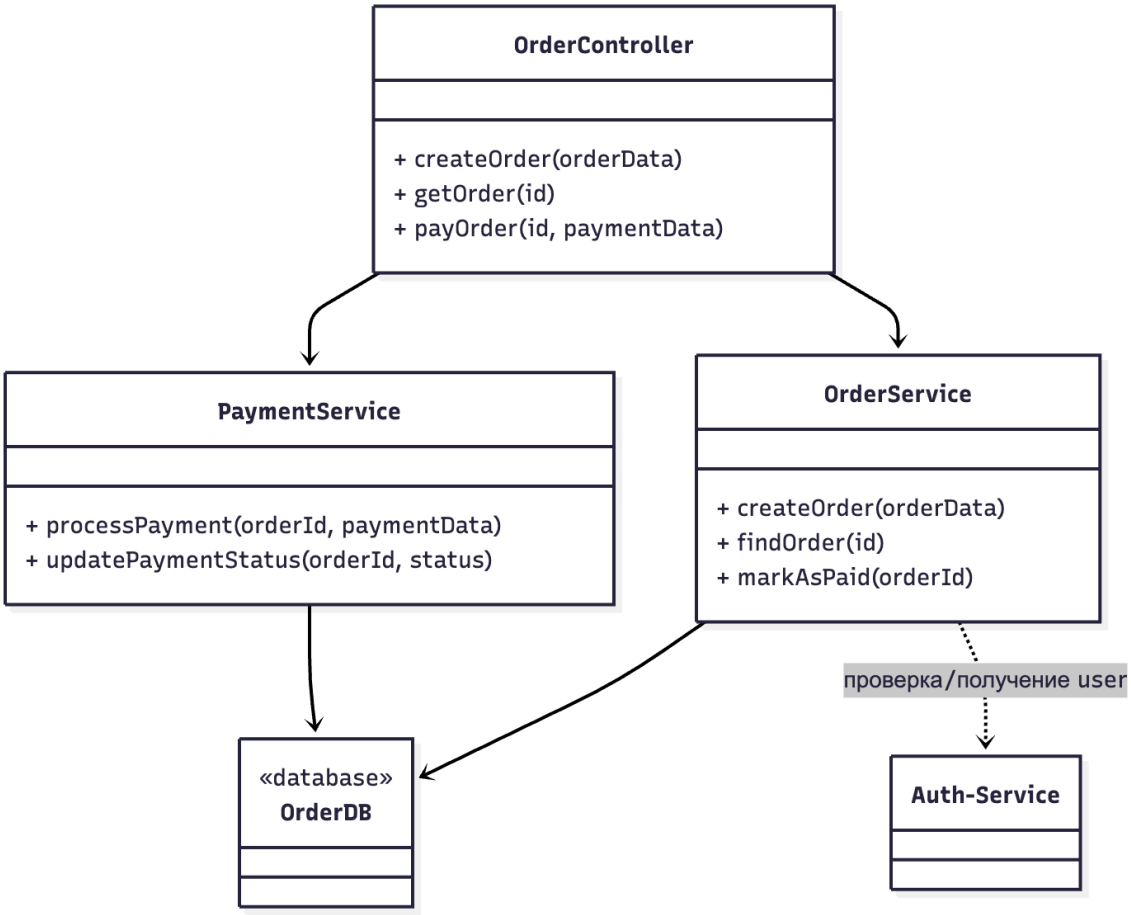

Рисунок 8 — Компонентная структура Order-Service

База данных Order-Service включает связанные таблицы Order и Payment (рисунок Рисунок 9). В таблице Order хранятся сведения о заказе: id, userId

(идентификатор пользователя из Auth-Service), общая сумма, дата создания, статус заказа (например, NEW, PAID, CANCELLED). Таблица Payment хранит детали оплаты: id, orderId, статус платежа (Pending, Completed и др.), способ оплаты. Связь между Order и Payment – один к одному (каждый заказ имеет единственную запись оплаты). При создании заказа ему присваивается статус NEW и создаётся связанная запись Payment (например, со статусом Pending). После успешной оплаты Order-Service обновляет статус заказа на PAID, а платежа – на Completed. Поле userId используется для связывания заказа с пользователем, но для получения подробностей о пользователе (имя, email) при необходимости Order-Service выполняет REST-запрос в Auth-Service.

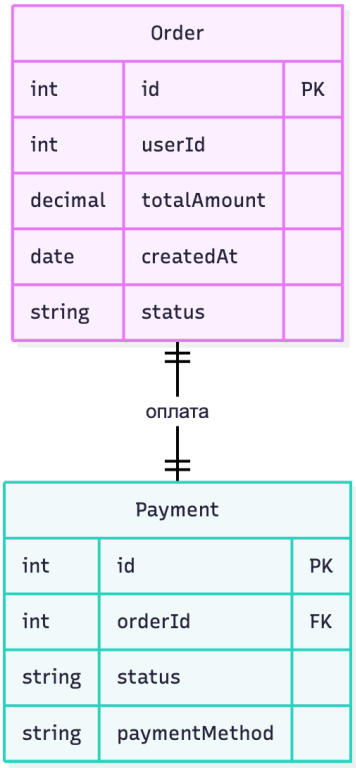

Рисунок 9 — ER-модель базы данных Order-Service (заказы и платежи)

1.2.5 Blog-Service: посты и комментарии

Ответственность: Blog-Service реализует блог-платформу: даёт возможность пользователям читать публикации, создавать новые посты о тренировках и здоровье, а также оставлять комментарии к постам. Он управляет сущностями постов (BlogPost) и комментариев (BlogComment), связывая комментарии с соответствующими постами.

Основные компоненты:

- **Контроллеры:** BlogPostController (операции с постами – получение списка, публикация нового поста, редактирование, удаление) и BlogCommentController (операции с комментариями – добавление комментария к посту, удаление или изменение комментария).
- **Сервисы:** BlogPostService и BlogCommentService инкапсулируют логику блога. BlogPostService, например, может получать пост вместе с комментариями (findPostWithComments), а BlogCommentService – логику добавления комментария (например, автоматическое проставление текущего пользователя как автора).
- **DTO:** используются для проверки данных постов и комментариев (CreatePostDTO, AddCommentDTO и др., проверяют длину текста, наличие необходимых полей).
- **Утилиты:** Blog-Service активно взаимодействует с Auth-Service для обогащения данных о пользователях. При выдаче списка постов сервис получает имя автора каждого поста через запрос в Auth-Service по authorId. Аналогично, при отображении комментариев запрашивается Auth-Service для получения имени автора комментария. Это может быть реализовано через HTTP-клиент (axios). JWT-токен используется для авторизации создания/удаления контента: публиковать посты и комментировать могут только авторизованные пользователи, а удалять или изменять пост/комментарий может либо автор (владелец), либо админ (проверка выполняется по userId и роли).

На рисунке Рисунок 10 показана структура компонентов Blog-Service.

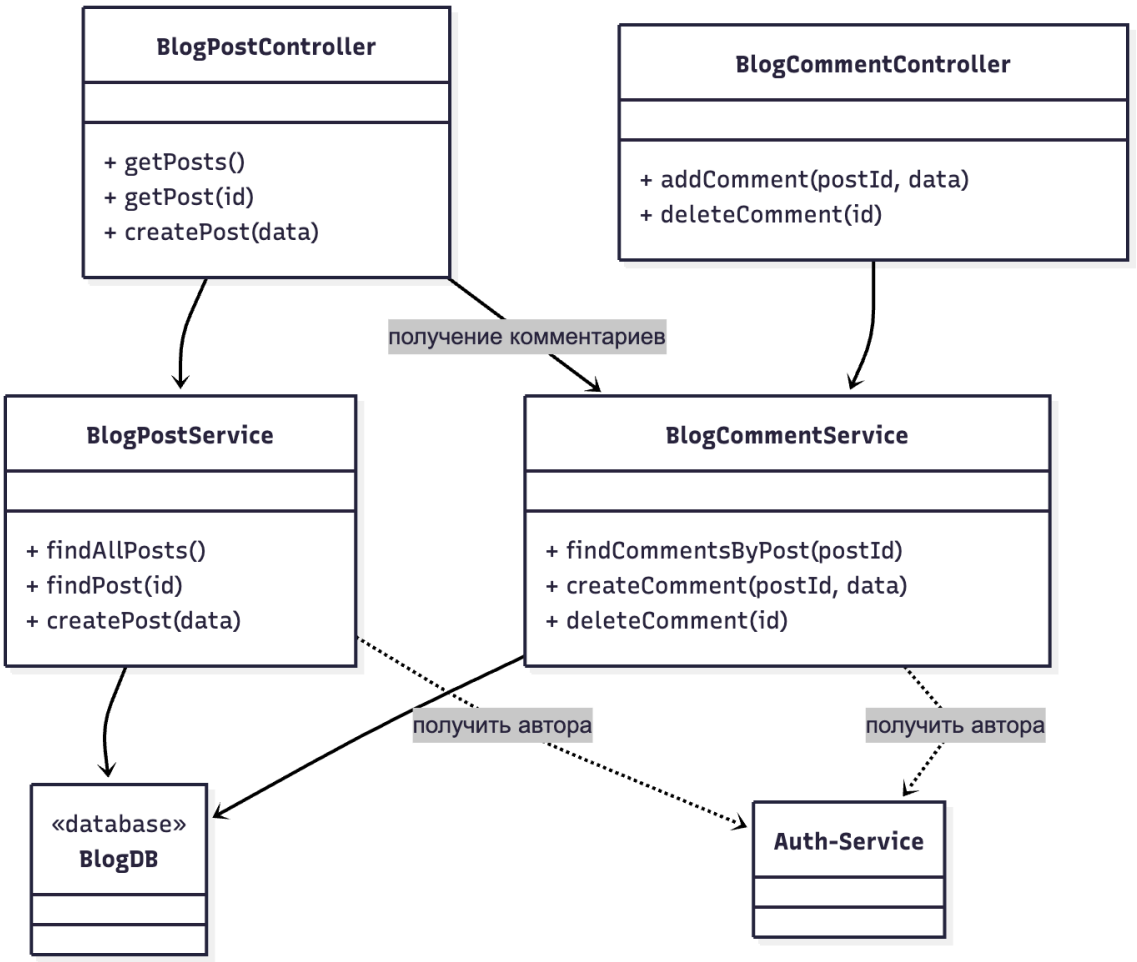

Рисунок 10 — Компонентная структура Blog-Service

База данных Blog-Service хранит только данные блога – посты и комментарии (рисунок Рисунок 11). Таблица BlogPost содержит публикации (id, title, content, authorId – идентификатор автора поста из Auth-Service). Таблица BlogComment хранит комментарии (id, postId, content, authorId – идентификатор автора комментария из Auth-Service). Внутренняя связь: один пост может иметь много комментариев (отношение 1 ко многим между BlogPost и BlogComment). Поля authorId в обеих таблицах являются внешними по отношению к Auth-Service, однако внутри Blog-Service они хранятся как обычные числовые поля (без ключевых связей). При необходимости отображения данных о авторе Blog-Service выполняет запросы в Auth-Service по указанным идентификаторам.

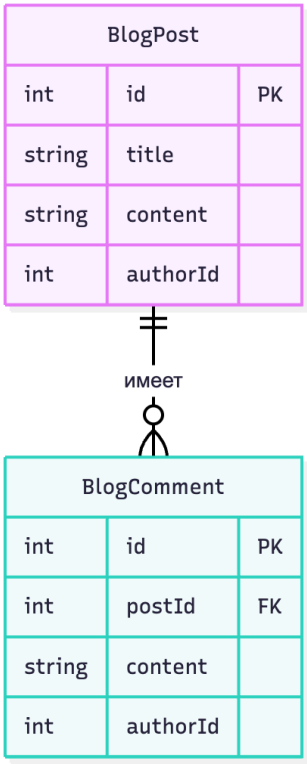

Рисунок 11 — ER-модель базы данных Blog-Service (посты и комментарии)

1.3 Диаграммы пользовательских сценариев

Далее приведены диаграммы последовательности, иллюстрирующие ключевые пользовательские сценарии работы системы. Эти диаграммы показывают, как распределяются запросы между сервисами и какие действия выполняются на каждом шаге.

1. Регистрация пользователя и вход (login)

На этой диаграмме показано, как новый пользователь регистрируется в системе, а затем выполняет вход для получения JWT токена:

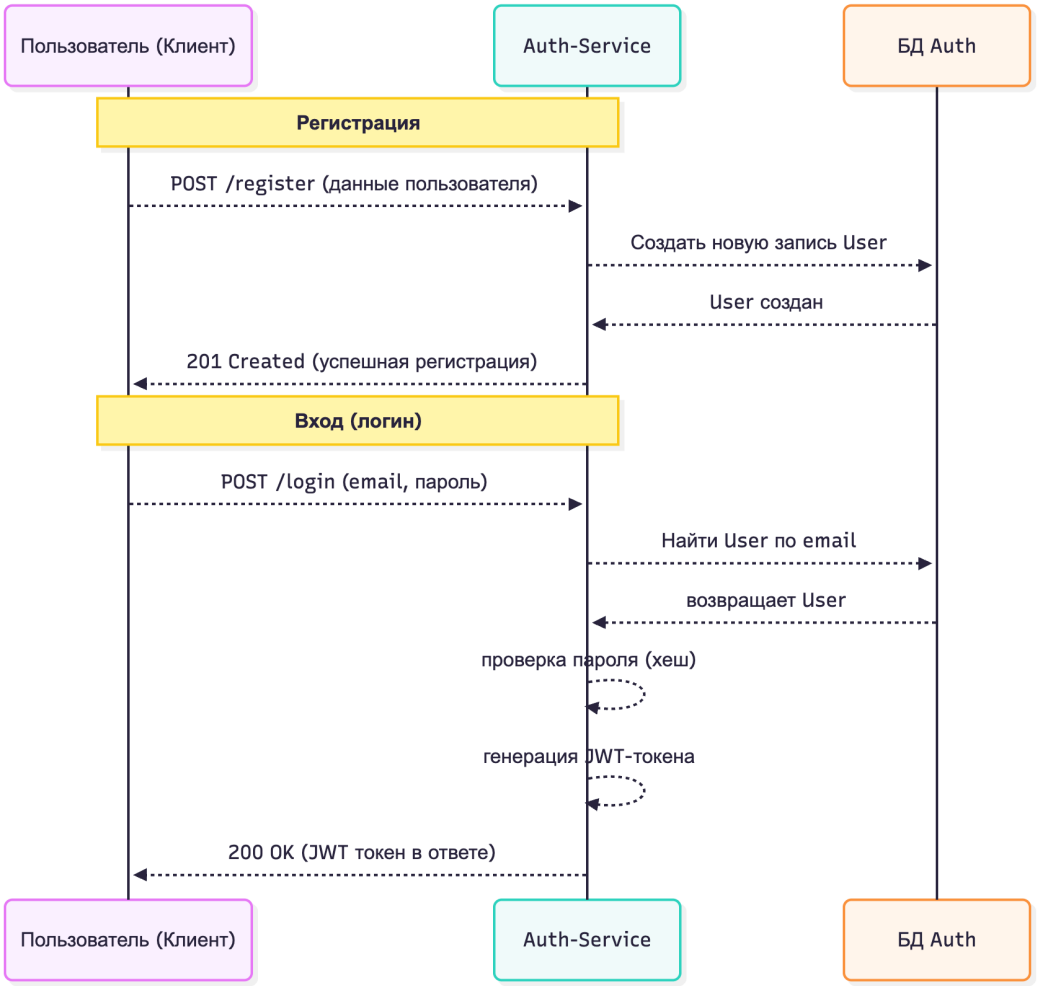

Рисунок 12 — Диаграмма последовательности — регистрация и вход пользователя

Объяснение: Пользователь отправляет запрос на регистрацию (эндпойнт `/register`) в Auth-Service, передавая свои данные (имя, email, пароль). Auth-Service сохраняет нового пользователя в своей базе (создает запись User) и возвращает подтверждение успешной регистрации. Затем пользователь выполняет вход: отправляет запрос `/login` с email и паролем. Auth-Service находит пользователя по email, проверяет правильность пароля (сравнивает хеш с помощью PasswordHasher) и при успешной проверке генерирует JWT-токен. Этот токен возвращается клиенту. В дальнейшем клиент будет использовать полученный JWT для доступа к защищенным ресурсам других сервисов.

2. Применение тренировочного плана пользователем

Диаграмма на рисунке Рисунок 13 отражает, как пользователь выбирает тренировочный план и «назначает» его себе для выполнения (привязка плана пользователю в Progress-Service):

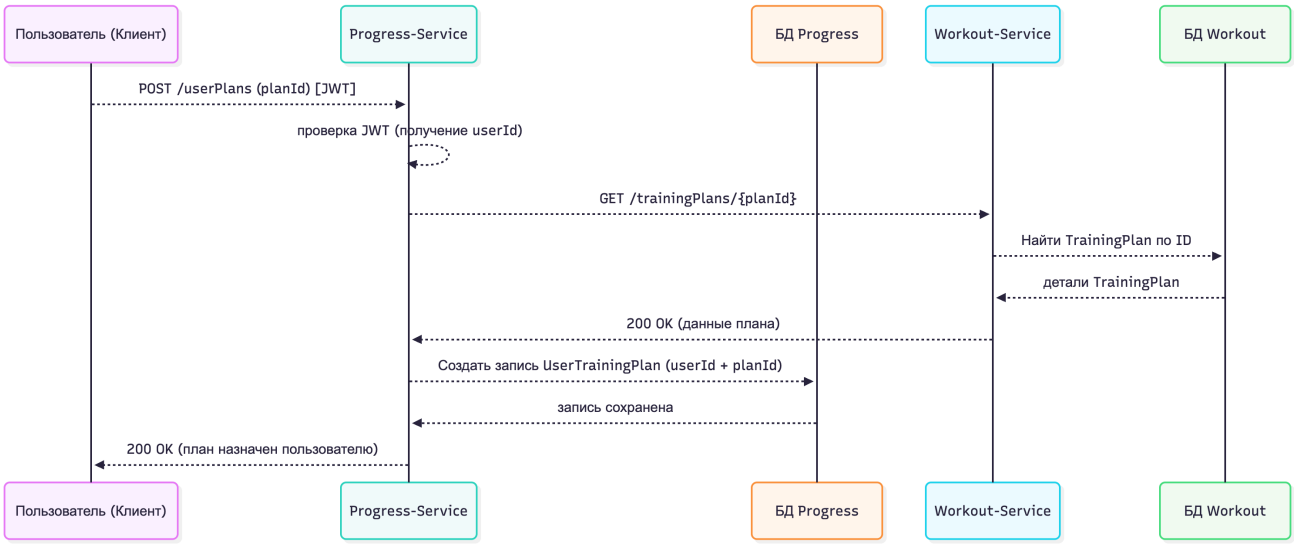

Рисунок 13 — Диаграмма последовательности — назначение пользователю тренировочного плана

Объяснение: Авторизованный пользователь (с JWT) вызывает метод Progress-Service для назначения себе тренировочного плана: запрос `POST /userPlans` с идентификатором плана. Progress-Service проверяет JWT и извлекает userId пользователя. Затем сервис обращается во внешний Workout-Service (запрос `GET /trainingPlans/{planId}`), чтобы убедиться в существовании указанного плана и получить его детали. Workout-Service ищет план по ID в своей базе и возвращает данные (например, название, описание). После этого Progress-Service сохраняет новую запись в своей базе (UserTrainingPlan, связывающую пользователя и выбранный план) и возвращает подтверждение успешного назначения. С этого момента Progress-Service «знает», какой план выполняет пользователь, и может отслеживать прогресс по нему.

3. Оформление заказа и оплата

На рисунке Рисунок 14 показан процесс покупки пользователем платной услуги или товара на платформе — от создания заказа до его оплаты:

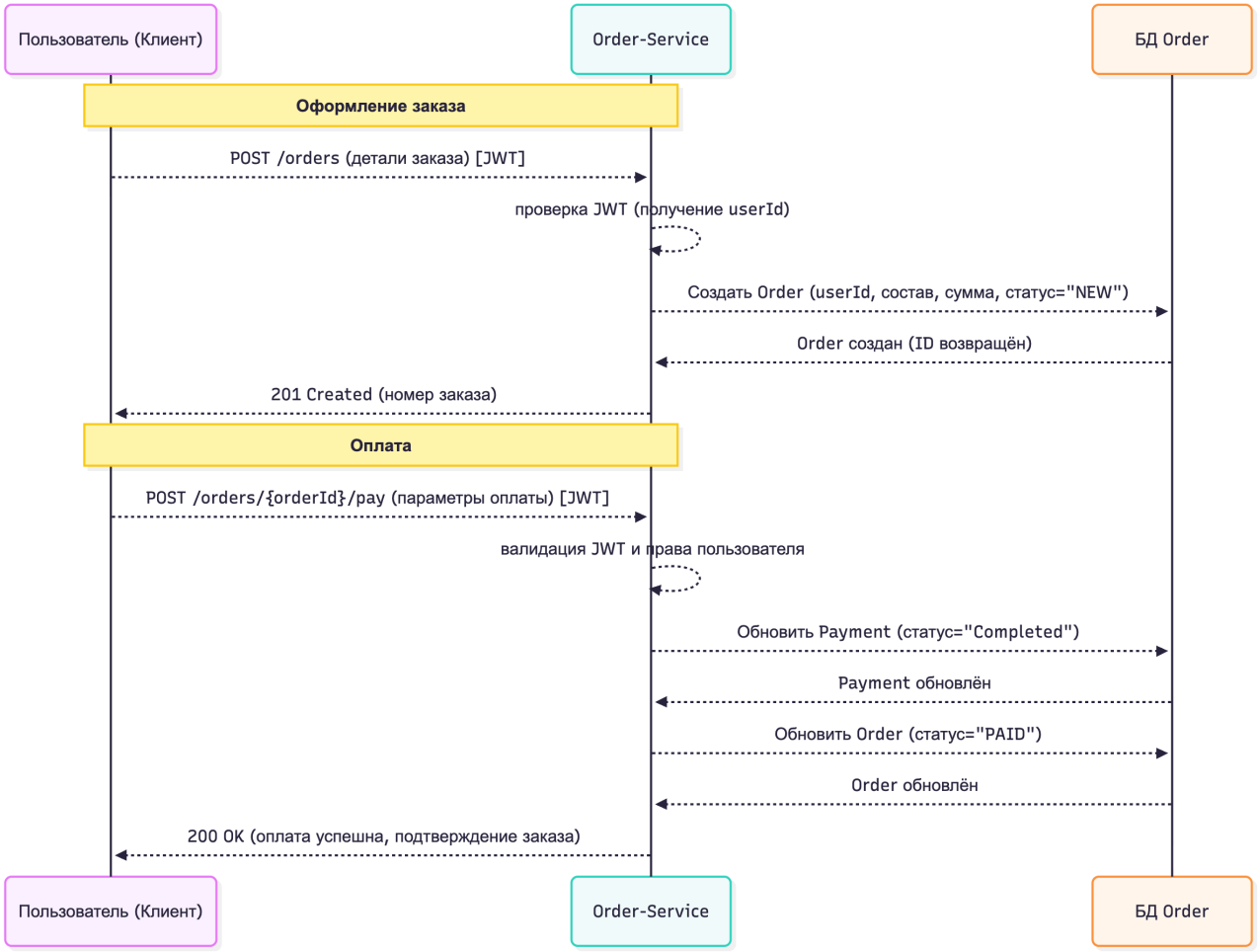

Рисунок 14 — Диаграмма последовательности — оформление заказа и его оплата

Объяснение: Пользователь инициирует оформление заказа, отправляя запрос `POST /orders` в Order-Service (передаются детали заказа, например выбранная услуга; JWT включён). Order-Service извлекает из токена userId и создает новый заказ в своей базе (статус «NEW») с привязкой к пользователю. В ответ сервис возвращает информацию о созданном заказе (например, уникальный идентификатор). Далее пользователь осуществляет оплату этого заказа — в реальности через внешнюю систему, но здесь упрощённо вторым запросом. Клиент отправляет `POST /orders/{orderId}/pay` (данные оплаты, JWT) в Order-Service. Сервис проверяет права (что запрос от того же пользователя) и затем обновляет запись оплаты: устанавливает статус Payment в «Completed». Одновременно связанный заказ помечается как «PAID». Order-Service возвращает успешный результат (подтверждение оплаты). Таким образом, заказ сначала

создаётся (как неоплаченный), затем после отдельного шага оплаты переводится в состояние «оплачен».

4. Просмотр и комментирование постов блога

Диаграмма на рисунке Рисунок 15 демонстрирует взаимодействие при просмотре пользователем блога и добавлении нового комментария к посту:

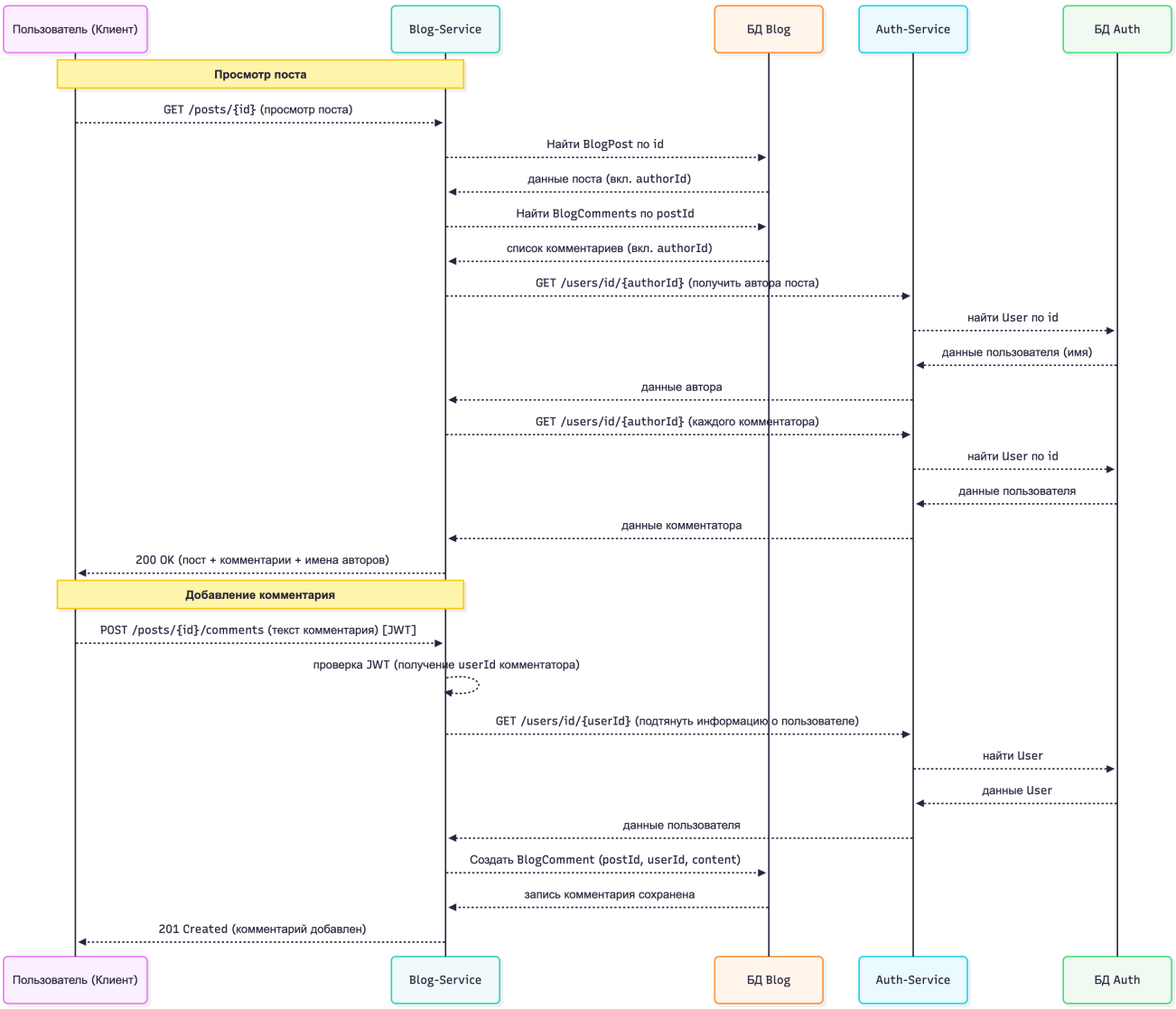

Рисунок 15 — Диаграмма последовательности — просмотр поста и добавление комментария

Объяснение: В первой части сценария пользователь запрашивает отображение конкретного поста блога (`GET /posts/{id}`). Blog-Service считывает из своей базы данных содержимое поста и связанные с ним комментарии. Поскольку в базе Blog-Service о пользователях хранится только идентификатор (authorId), сервис делает запросы в Auth-Service, чтобы получить информацию

о пользователях: сначала о авторе поста по его id, затем о каждом авторе комментария. Auth-Service возвращает необходимые данные (например, имя пользователя), и Blog-Service формирует ответ, содержащий данные поста, список комментариев и имена авторов. Во второй части сценария пользователь (с JWT) отправляет запрос `POST /posts/{id}/comments` для добавления комментария. Blog-Service проверяет токен и получает из него userId — идентификатор текущего пользователя, который будет автором комментария. Сервис создаёт новую запись комментария в базе (связывая его с указанным постом и userId автора) и возвращает успешный ответ. Теперь комментарий сохранён и будет отображён при следующем запросе поста (с подставленным именем автора через Auth-Service).

5. Отслеживание прогресса (показатели здоровья и выполнение плана)

Наконец, рисунок Рисунок 16 иллюстрирует, как пользователь просматривает свои данные прогресса, включая активные планы тренировок:

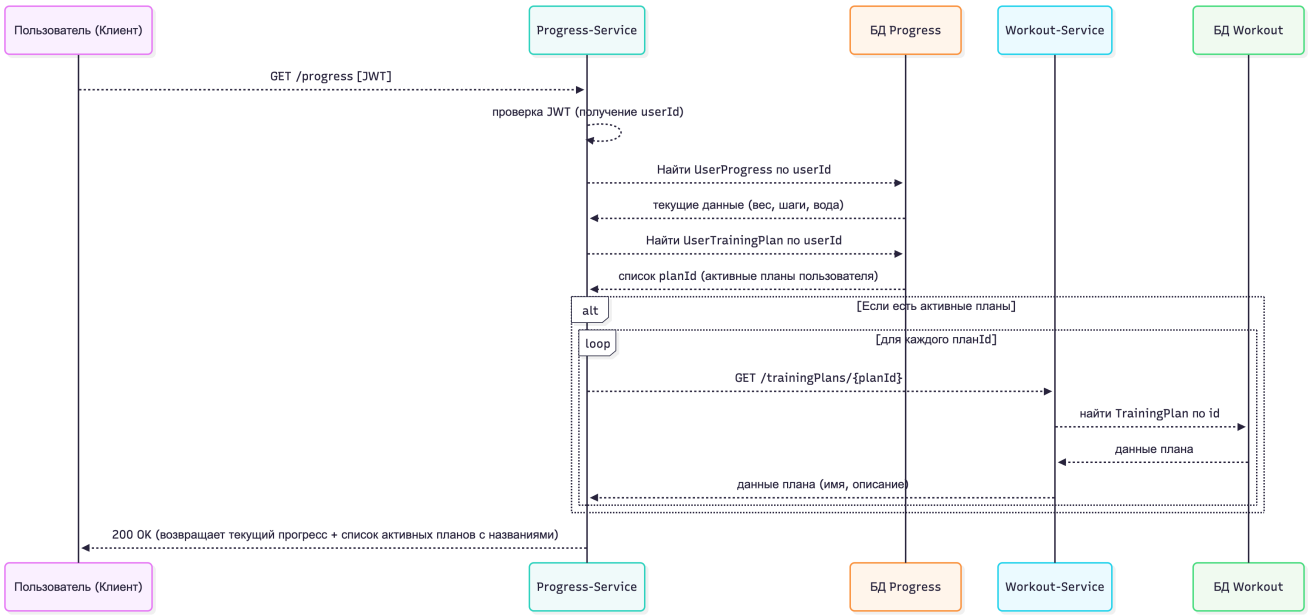

Рисунок 16 — Диаграмма последовательности — получение данных прогресса пользователя

Объяснение: Пользователь запрашивает свои текущие показатели (`GET /progress` с JWT). Progress-Service валидирует токен и узнаёт userId. Затем сервис

получает из своей базы запись UserProgress данного пользователя (текущий вес, шаги, вода и т.д.). Также извлекаются все записи UserTrainingPlan этого пользователя — то есть идентификаторы тренировочных планов, которые пользователь сейчас выполняет. Если есть активные планы, Progress-Service для каждого выполняет внешний запрос в Workout-Service (`GET /trainingPlans/{id}`) и получает детали плана (название, цель и пр.). Собрав информацию, Progress-Service возвращает клиенту совокупные данные: текущие показатели здоровья пользователя и список активных тренировочных программ с их названиями. На стороне клиента эта информация отображается, позволяя пользователю отслеживать свой прогресс. При изменении показателей (например, обновлении веса) происходит аналогичный цикл: пользователь отправляет новые данные, сервис обновляет запись UserProgress и при необходимости возвращает обновлённую информацию.

1.4 Общие итоговые диаграммы

Помимо приведённых выше частных схем, были подготовлены два интегральных диаграммы для обзора всей системы. На рисунке Рисунок 17 показана общая компонентная структура решения: все пять микросервисов с основными контроллерами, сервисами и хранилищами данных на одной схеме. На рисунке Рисунок 18 представлена совокупная ER-модель данных платформы: таблицы всех сервисов сведены на одной диаграмме (названия таблиц снабжены префиксами, указывающими принадлежность сервису: AUTH, WORK, PROG, ORD, BLOG_).

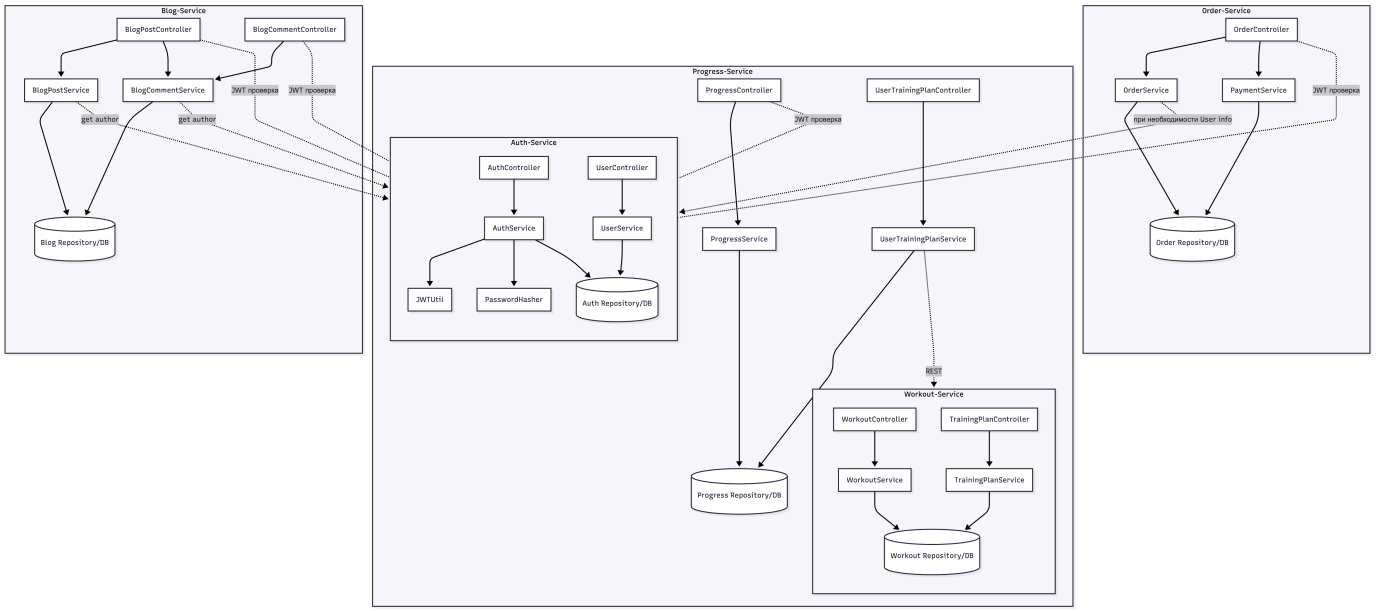

Рисунок 17 — Общая диаграмма компонентов микросервисной платформы

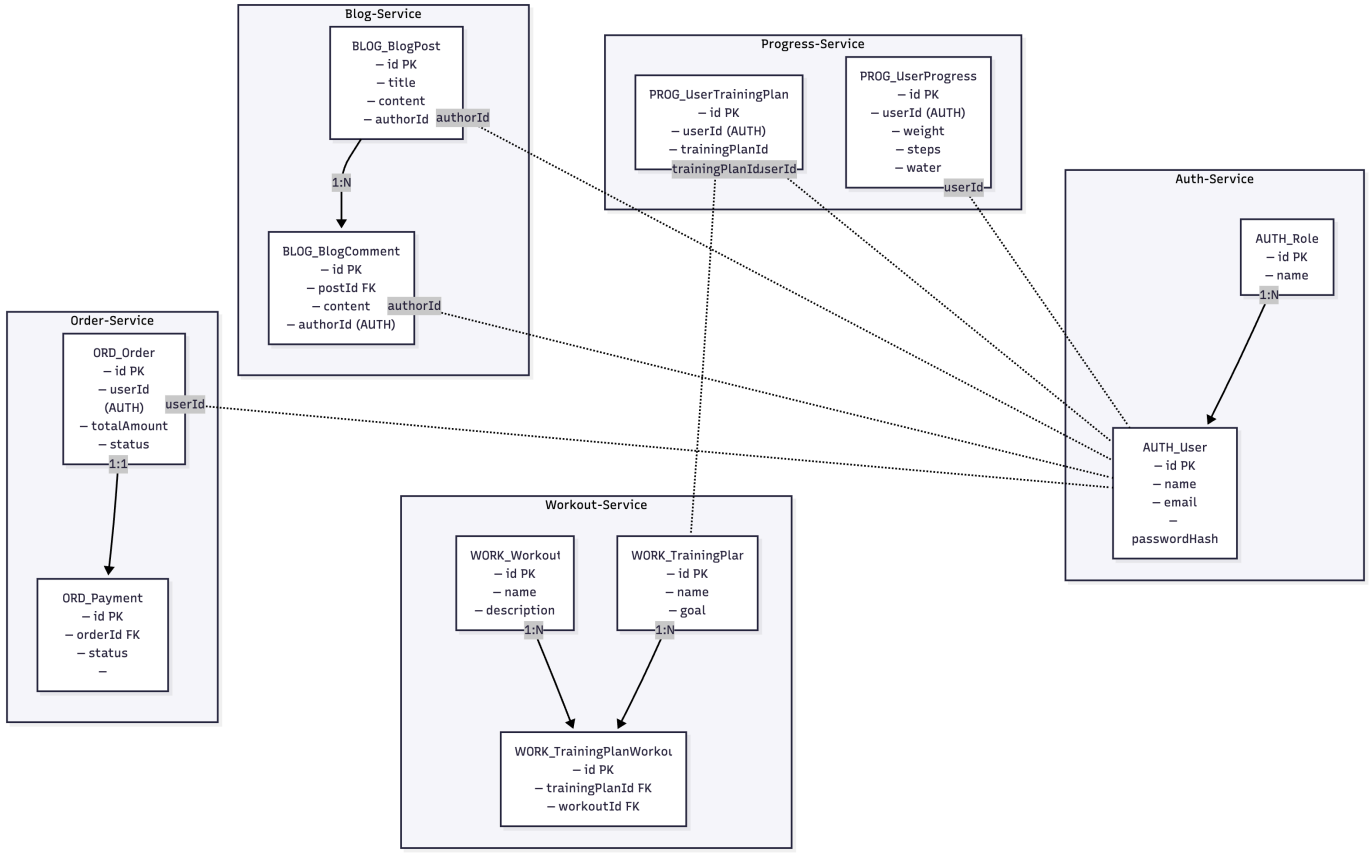

Рисунок 18 — Общая ER-диаграмма данных платформы (сгруппировано по сервисам)

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы разработан проект микросервисной архитектуры фитнес-платформы. Итоговый дизайн включает:

- разделение функциональности приложения на независимые микросервисы по доменам (Auth, Workout, Progress, Order, Blog);
- использование JWT-токенов для централизованной аутентификации пользователей при распределенной авторизации на уровне сервисов;
- полную изоляцию данных каждого сервиса (собственная база данных, отсутствие прямых внешних связей) при взаимодействии через REST API;
- единый подход к внутренней организации сервисов (контроллеры, сервисы, DTO, утилиты), обеспечивающий модульность и удобство сопровождения кода;
- разработку схем баз данных для каждого микросервиса, отражающих только необходимые внутри домена связи;
- моделирование межсервисного взаимодействия на примере ключевых пользовательских сценариев (регистрация, выполнение плана, покупка услуги, работа с блогом, отслеживание прогресса).

Все поставленные задачи выполнены, что подтверждается приложенными диаграммами. Спроектированное решение соответствует целям работы и демонстрирует преимущества микросервисного подхода: изменения в одном сервисе минимально влияют на остальные, каждый компонент можно масштабировать отдельно при росте нагрузки, а система в целом остаётся гибкой и поддерживаемой.