Описание модуля Performance Review (360 + Self)

АРХИТЕКТУРА

Модуль предназначен для проведения регулярных оценок сотрудников с элементами самооценки и оценки коллег. Система строится вокруг набора связанных таблиц, каждая из которых отвечает за хранение данных о сотрудниках, временных циклах, вопросах и результатах ответов.

Таблица сотрудников

Базовая таблица сотрудников (Employer) содержит все ключевые атрибуты: ID, ФИО, дату рождения, почту, дату устройства и должность.
Эти данные формируют профиль сотрудника и служат для:

- автоматического запуска **нулевой самооценки** в день трудоустройства;
- генерации **одноразовых ссылок** для участия в оценке коллег (без отдельной системы авторизации);
- отправки напоминаний и уведомлений о предстоящих оценочных циклах.

Самооценка при устройстве — это первая базовая точка, которая фиксирует исходное состояние навыков сотрудника. Она будет использоваться как стартовая метка для отслеживания динамики развития.

Таблица тестовых циклов

Таблица с датами тестов хранит информацию о плановых циклах оценки (1, 3, 6, 12, 18, 24 месяцев).
Сроки можно изменять вручную — например, если компания решит проводить промежуточные проверки. Каждая запись имеет собственный ID, который используется при формировании связи с таблицей ответов.

Таблицы Hard и Soft Skills

Существуют две таблицы — для технических и поведенческих компетенций.
Структура у них идентичная: категории, подкатегории, вопросы и описания. Каждый вопрос имеет уникальный ID и текстовое описание (grade description), которое определяет, что именно оценивается.

Сотрудник (или его коллега) выставляет оценку от 1 до 10, где 1 — «не владеет навыком», 10 — «экспертный уровень».
Система позволяет масштабировать количество категорий и вопросов без изменения архитектуры базы.

Таблица ответов (Answer Table)

Основная рабочая таблица. Содержит результаты всех проведённых оценок, включая самооценки.
Каждая запись включает:

- ID сотрудника, которого оценивают;

- ID респондента (того, кто оценивает);
- ID вопроса и категории;
- временную метку обновления (update_time);
- значение оценки (grade).

При создании цикла оценки система автоматически формирует записи по всем сотрудникам и вопросам, присваивая каждой оценке значение 0. Это обозначает, что ответ ещё не дан.
Если сотрудник или коллега проходит тест, запись обновляется с реальной оценкой (от 1 до 10).

Логика определения типа оценки простая:
если ID employer = ID respondent, значит это **self-test**;
если не совпадает — **оценка коллеги (peer review)**.

Логика отслеживания активности

Проверка активности осуществляется по нулевым записям (grade = 0).
Если сотрудник не участвует в оценке коллег, это считается понижением его вовлечённости. В дальнейшем эти данные будут использоваться при пересчёте его soft-навыков и метрик командного взаимодействия.

РЕЗУЛЬТАТ И МЕТРИКИ

Система Performance Review служит инструментом для комплексной оценки сотрудника с использованием самооценки, обратной связи от коллег и динамики развития во времени. Результаты формируются автоматически на основании данных, собранных в Answer Table, и выводятся во FrontEnd-приложении в виде интерактивных графиков и блоков с интерпретацией.

Механизм формирования результатов

После завершения теста по каждому периоду система агрегирует данные из ответов всех респондентов (включая самооценку).
Для каждого вопроса и категории рассчитывается **средний балл**:
если, например, три коллеги оценили сотрудника в 3, 7 и 8, итоговое значение составит 6.
Такие средние значения вычисляются отдельно:

- по **самооценке** (self-score),
- по **оценке коллег** (peer average),
- и по каждой категории (hard/soft skills).

Фронтенд получает уже агрегированные данные и визуализирует:

1. Текущую самооценку сотрудника.
2. Среднюю оценку коллег за тот же период.
3. Динамику изменений по периодам — 0 (вход), 1, 3, 6, 12 месяцев.

Каждая новая итерация добавляется в общий график, что позволяет видеть траекторию профессионального и поведенческого роста.

Интерпретация динамики

Система отслеживает не только текущие баллы, но и **изменение между циклами**:

- Если самооценка снижается, но оценки коллег остаются стабильными или растут — это сигнал о переоценке на старте. Сотрудник начинает реалистичнее смотреть на свои навыки, и HR может предложить дополнительные курсы или менторинг.
- Если и самооценка, и коллеги растут — высокая успешность адаптации. Сотрудник нашел баланс и его вклад замечен командой.
- Если самооценка растет, а оценка коллег падает — риск разрыва восприятия: сотрудник уверен в себе, но окружение не видит подтверждения. Это повод для индивидуальной беседы с руководителем.
- Если обе метрики падают — тревожный сигнал о выгорании, дезадаптации или конфликте. Необходимо вмешательство HR или наставника.

Визуализация на фронте

Интерфейс представляет собой web-приложение (SPA или корпоративный фронт в рамках портала), где сотрудник и руководитель видят:

- графики динамики по категориям и общему индексу;
- вклад самооценки и командной оценки по каждому периоду;
- текстовые комментарии или рекомендации (генерируются HR на основе шаблонов).

Для руководителя предусмотрен отдельный дашборд, где по каждому сотруднику отражены:

- текущий уровень адаптации (в процентах от максимально возможного);
- изменение динамики по сравнению с предыдущим циклом;
- группы риска (по soft skills, вовлеченности или просадке активности в peer-оценке).

Таблица интерпретации метрик и состояния сотрудника

Сценарий поведения	Самооценка	Оценка коллег	Интерпретация состояния	Рекомендации HR / руководителю
Реалистичная переоценка	↓	↔ или ↑	Сотрудник корректирует ожидания, переходит из эйфории в реализм.	Поддерживающая обратная связь, отметить прогресс.
Успешная адаптация	↑	↑	Команда подтверждает рост, сотрудник интегрировался.	Поощрение, вовлечение в сложные задачи.
Разрыв восприятия	↑	↓	Сотрудник считает, что растет, но команда не видит подтверждения.	Индивидуальное интервью, корректировка задач.
Потеря мотивации	↓	↓	Общий спад, возможное выгорание или проблемы в коллективе.	Срочная беседа с руководителем и HR.
Недооценка потенциала	↓	↑	Сотрудник не уверен, но команда видит прогресс.	Поддержка, признание достижений, повышение уверенности.
Стагнация	↔	↔	Без изменений, уровень стабильный.	Нейтрально; можно предложить развитие или новые цели.

Хочешь, я теперь на основе этого подготовлю **структуру данных (мини-ERD, словами)**, где покажу связи между таблицами (Employer, Hard/Soft Skills, Answer Table, Review Periods, Aggregates)? Это нужно, чтобы команда потом могла на основе этого строить архитектуру БД и API.

Отлично 👍
Ниже — **структура данных (ERD-описание)** в текстовом виде, полностью адаптированная под твою модель, где **`Answers_table`** — это конечная, центральная таблица, в которой сходятся все связи.

Логическая структура базы данных (ERD словами)

1. Employer

Назначение: Хранит основную информацию о сотрудниках.
Связи: Является корневой таблицей для всех остальных сущностей, связанных с тестированием, ответами и анализом.

Поля:

- `id_employer` — уникальный идентификатор сотрудника
- `fio` — ФИО
- `birthday` — дата рождения
- `email` — почта (используется для одноразовых ссылок)
- `date_of_employment` — дата устройства на работу
- `position` — должность
- `date_of_dismissal` — дата увольнения (null, если работает)

Использование:
При найме сотрудника создается запись; по email запускается первичный self-test через одноразовую ссылку.

2. Review_Periods

Назначение: Определяет интервалы тестирования (0, 1, 3, 6, 12 месяцев и далее).
Связи: Используется для связки с ответами в `Answers_table`.

Поля:

- `id_period` — уникальный ID периода
- `month_period` — значение в месяцах
- `start_date`, `end_date` — диапазон проведения (может изменяться админом)

Использование:
Позволяет гибко управлять временем рассылок и построением отчетов в динамике.

3. Hard_Skills

Назначение: Справочник профессиональных компетенций, проверяемых в тестах.
Связи: Каждая запись может быть связана с несколькими вопросами (через `id_question`).

Поля:

- `id_question_category` — категория (напр. «Frontend», «DevOps», «Data Analysis»)
- `category_description` — краткое описание тематики
- `id_question` — ID вопроса внутри категории
- `grade_description` — текст вопроса (например: «Оцените умение анализировать пользовательские требования»)

4. Soft_Skills

Назначение: Справочник личностных и поведенческих компетенций.
Связи: Структурно аналогичен Hard_Skills, но относится к «поведенческому» типу.

Поля:

- `id_question_category`
- `category_description`
- `id_question`
- `grade_description` (например: «Как быстро сотрудник адаптируется в новых условиях?»)

5. Answers_Table

Назначение: Центральная таблица системы. Хранит все ответы, как по self-test, так и по peer-review.

Поля:

- `id_answer` — уникальный ID записи
- `id_employer` — сотрудник, который оценивается
- `id_respondent` — сотрудник, который заполняет тест (если совпадает с `id_employer` — self-test)
- `id_period` — ссылка на период тестирования (`Review_Periods`)
- `question_type` — `hard` или `soft` (для унификации)
- `id_question_category` — категория вопроса
- `id_question` — конкретный вопрос
- `grade` — оценка от 0 до 10
- `update_time` — дата обновления записи

Особенности логики:

- При создании нового цикла тестирования система автоматически создает все вопросы по категориям с `grade = 0`.
- После прохождения теста происходит `update` — ответы получают реальные значения.
- `grade = 0` означает, что сотрудник не прошел тест или проигнорировал запрос (используется для KPI активности).

6. Results_View (виртуальная таблица / API-агрегат)

Назначение: Не хранится физически, но создается как запрос или API endpoint. Используется фронтендом для построения графиков динамики и отчетов.

Формируется из `Answers_table` с группировкой:

- по сотруднику (`id_employer`),
- по периоду (`id_period`),
- по типу (`hard` / `soft`),

- и по источнику оценки (`self` / `peer`).

Расчетные поля:

- `avg_self_grade` — средний балл самооценки
- `avg_peer_grade` — средний балл от коллег
- `total_avg` — итоговый общий средний балл
- `delta_self_peer` — разница между самооценкой и оценкой коллег
- `trend` — изменение показателя относительно предыдущего периода (в %)

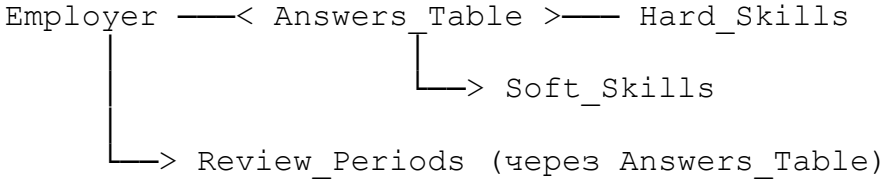
7. Review_Logs (опционально)

Назначение: Служебная таблица для фиксации рассылок и статусов.

Поля:

- `id_log`
- `id_employer`
- `id_period`
- `email_sent` (true/false)
- `link_expiry_date`
- `status` (в процессе / завершено / пропущено)

Визуально (словесная ERD-схема)



(Answers_Table — узел, связывающий все остальные таблицы)

ОПИСАНИЕ API

API 1 — Инициация цикла Performance Review / Self-Test

Цель

Автоматически определить, наступил ли момент для запуска анкетирования (по дате трудоустройства или по плановым периодам ревью),
создать записи с начальными значениями (`grade = 0`) в `Answers_table`,
и подготовить одноразовые ссылки для прохождения формы.

Триггер

API вызывается **по cron-задаче один раз в сутки** (например, в 00:10).
При вызове он проверяет даты и создает тестовые записи, если наступил период ревью.

Условная логика

1. Проверка типа цикла

- Если **текущая дата = дате трудоустройства (`date_of_employment`)** → создается **нулевой self-test** только для этого сотрудника.
- Если **текущая дата совпадает с началом одного из Review_Periods (1, 3, 6, 12 мес)** →
  создается **цикл peer-review** для всех сотрудников, где оцениваемым является один, а респондентами — все остальные активные сотрудники.

Входные данные

(на стороне backend cron / без ручного запроса)

```
{
  "current_date": "2025-10-19"
}
```

Основная логика выполнения

1. **Получить всех активных сотрудников**

```
SELECT * FROM Employer
WHERE date_of_dismissal IS NULL;
```

4. **Для каждого сотрудника рассчитать дни с даты трудоустройства:**

```
days = DATEDIFF(current_date, date_of_employment)
```

6. **Если days = 0 → создать Self-Test**
   - Создать в `Answers_table` набор записей:
     - `id_employer = id сотрудника`
     - `id_respondent = id сотрудника`
     - `id_period = 0`
     - `grade = 0`
   - Сгенерировать одноразовую ссылку (валидность 24 часа)
   - Отправить email:

«Привет, [Имя]! Пройди стартовую самооценку в рамках Performance Review.
Ссылка активна 24 часа.»

7. **Если days совпадает с одним из контрольных периодов (30, 90, 180, 365...)**
    - Для этого сотрудника создается **peer-review цикл**:
        - `id_employer = id сотрудника`
        - `id_respondent = id коллеги` (все активные сотрудники, кроме оцениваемого)
        - `id_period = соответствующий период`
        - `grade = 0`
    - Для каждого респондента создается одноразовая ссылка и отправляется письмо:

      «Вам предложено оценить коллегу [Имя]. Ссылка активна 24 часа.»

Создание записей в БД

Для каждой новой проверки создается набор записей с `grade = 0`:

```
INSERT INTO Answers_Table (
  id_employer, id_respondent, id_period, question_type,
  id_question_category, id_question, grade, update_time
)
SELECT
  e.id_employer, r.id_respondent, p.id_period, q.type,
  q.id_question_category, q.id_question, 0, NOW()
FROM Employer e
JOIN Employer r ON e.id_employer != r.id_employer
JOIN Review_Periods p ON p.month_period = {matched_period}
JOIN Questions q ON q.active = 1;
```

(для self-test `r.id_respondent = e.id_employer`)

Выходные данные (ответ API)

```
{
  "status": "success",
  "created_self_tests": 3,
  "created_peer_reviews": 25,
  "emails_sent": 28
}
```

Состояния после вызова

Состояние	Описание
`grade = 0`	Запись создана, тест не пройден
`grade > 0`	Ответ внесён
отсутствует запись	Тест еще не инициирован

Дополнительные условия

- Если запись с таким `id_employer`, `id_respondent` и `id_period` **уже существует**, новая не создаётся.
- API должен логировать каждый запуск в `Review_Logs` с отметкой, для кого были созданы проверки.
- Если рассылка не удалась (ошибка SMTP, expired link generation) → запись всё равно создаётся,
  но помечается `status = pending_email`.

API 2 — Приём и обработка ответов Performance Review

Цель

Принять ответы, отправленные сотрудником через одноразовую ссылку, обновить записи в `Answers_table`, зафиксировать время, и при необходимости изменить статус рассылки в `Review_Logs`.

Триггер

Пользователь открывает персональную ссылку на анкету →
Front-End подтягивает список вопросов (через API #3, который будет позже).
После заполнения и отправки форма делает `POST` запрос к этому API.

Входные данные

```
{
  "token": "ad1e1b02-41d3-42aa-bcf2-b917b3b4fd1e",
  "answers": [
    {
      "id_question": 101,
      "grade": 8
    },
    {
      "id_question": 102,
      "grade": 6
    },
    {
      "id_question": 103,
      "grade": 9
    }
  ]
}
```

- `token` — одноразовый уникальный ключ, выданный по email (валиден 24 часа).
- `answers` — массив из ID вопросов и оценок от 1 до 10.

Основная логика выполнения

1. Проверка валидности токена

- Проверить срок действия токена в таблице `Review_Logs`.
- Если `expired` → вернуть ошибку:
- `{ "status": "error", "message": "Link expired" }`

2. Определить контекст

По токену извлечь:

- `id_employer` (кого оценивают),

- `id_respondent` (кто оценивает),
- `id_period`,
- `question_type` (hard / soft).

Это позволит точно обновить нужные строки в `Answers_table`.

3. Проверка целостности данных

Для каждой записи убедиться, что существует строка с `grade = 0`
(созданная ранее при вызове API #1).
Если нет → создать новую запись (fallback на случай сбоев рассылки).

4. Обновление данных

По каждой паре `id_employer`, `id_respondent`, `id_question`, `id_period` обновить поле `grade` и время обновления:

```
UPDATE Answers_Table
SET grade = {grade}, update_time = NOW()
WHERE id_employer = {employer}
  AND id_respondent = {respondent}
  AND id_question = {id_question}
  AND id_period = {period};
```

5. Изменение статуса токена

После успешной обработки всех ответов:

```
UPDATE Review_Logs
SET status = 'completed', completed_at = NOW()
WHERE token = {token};
```

Сценарии

Сценарий	Поведение
Self-Test (`id_employer = id_respondent`)	Обновляются только ответы текущего сотрудника
Peer-Review (`id_employer ≠ id_respondent`)	Обновляются ответы коллеги
Частичное заполнение	Можно сохранять промежуточно (draft), при `save_mode = "partial"`
Повторное открытие токена	Если статус `"completed"`, возвращается предупреждение: `"already_submitted"`

Выходные данные

```
{
  "status": "success",
  "updated_count": 25,
  "message": "Answers successfully submitted"
}
```

Особенности и проверки

- Все ответы валидируются:
  `grade` ∈ [1,10]
  `answers.length` ≥ 1
  Несуществующие `id_question` → игнорируются.
- API обязательно логирует результат (успех/ошибку) в отдельную таблицу `Audit_Log` (опционально).
- Повторная отправка по тому же токену не допускается.

Пример потока (в связке с API #1):

```
[API #1] → создаёт записи с grade=0
   ↓
[Email link sent]
   ↓
[User opens form → submits → API #2]
   ↓
[Grades updated in Answers_Table]
   ↓
[Review_Logs: status = completed]
```

API 3 — Получение данных для анкеты по одноразовой ссылке

Цель

По токену выдать полную структуру анкеты:

- данные оцениваемого сотрудника,
- список вопросов (hard и soft skills),
- категориальную структуру,
- сведения о типе теста (self / peer),
- и статус валидности ссылки.

Триггер

Frontend открывает форму по ссылке из письма.
В URL передается токен:

```
https://review.wink.app/form?token=ad1e1b02-41d3-42aa-bcf2-b917b3b4fd1e
```

Приложение делает `GET` запрос к API:

```
GET /api/review/form-data?token=ad1e1b02-41d3-42aa-bcf2-b917b3b4fd1e
```

Входные данные

```
{
  "token": "ad1e1b02-41d3-42aa-bcf2-b917b3b4fd1e"
}
```

Основная логика выполнения

1. Проверка токена

- Найти токен в таблице `Review_Logs`.
- Проверить:
    - срок действия (`link_expiry_date > NOW()`),
    - статус (`pending` или `completed`).

Если просрочен или уже завершен → вернуть ошибку:

```
{ "status": "error", "message": "Link expired or already used" }
```

2. Получить контекст токена

По токену извлекаются:

- `id_employer` — кого оценивают
- `id_respondent` — кто оценивает
- `id_period` — период
- `question_type` — hard/soft (или оба, если тип mixed)

На основе этих данных подготавливается структура опроса.

3. Формирование структуры вопросов

- Из таблиц `Hard_Skills` и `Soft_Skills` выбрать все активные вопросы.
- Объединить их в единый JSON-объект, группируя по категориям.
- Если записи для данного периода и пары (employer–respondent) уже существуют в `Answers_table`,
  подтянуть их текущие значения `grade` (например, если сотрудник уже начал заполнять и сохранил draft).

Пример SQL-запроса:

```
SELECT
  q.id_question, q.grade_description, q.id_question_category,
  c.category_description, a.grade
FROM Questions q
LEFT JOIN Answers_Table a
  ON a.id_question = q.id_question
  AND a.id_employer = {employer}
  AND a.id_respondent = {respondent}
  AND a.id_period = {period}
JOIN Categories c
  ON c.id_question_category = q.id_question_category;
```

Выходные данные

```
{
  "status": "success",
  "review_type": "self",
  "review_period": "3 months",
  "employer": {
    "id": 15,
    "fio": "Иванов Сергей Петрович",
    "position": "Frontend Developer"
  },
  "respondent": {
    "id": 15,
    "fio": "Иванов Сергей Петрович"
  },
  "questions": {
    "hard_skills": [
      {
        "category": "Frontend",
        "items": [
          { "id": 101, "question": "Понимание основ React.js",
"grade": 0 },
          { "id": 102, "question": "Работа с REST API", "grade":
0 }
        ]
      }
    ],
    "soft_skills": [
      {
        "category": "Командное взаимодействие",
        "items": [
          { "id": 201, "question": "Умение адаптироваться в
новой команде", "grade": 0 },
          { "id": 202, "question": "Навыки аргументированного
общения", "grade": 0 }
        ]
      }
    ]
  },
  "link_valid_until": "2025-10-20T12:00:00Z"
}
```

Особенности и сценарии

Сценарий	Поведение
Self-test	В блоке `employer` и `respondent` одинаковый ID
Peer-review	Разные ID, в интерфейсе отображается “Оценка коллеги [Имя]”
Draft-заполнение	Если ранее сохранены ответы (`grade > 0`), они подгружаются для редактирования
Просроченный токен	API возвращает ошибку, фронт отображает уведомление об истечении ссылки
Повторный доступ после завершения	API возвращает `"status": "completed"` и заблокированный режим отображения (readonly)

Связь с другими API

API	Роль
API #1	Создает записи `grade=0` и генерирует токен
API #2	Принимает и сохраняет ответы
API #3	Отдает форму и данные для заполнения

API 4 — Расчёт агрегатов и динамики развития сотрудника

Цель

Агрегировать результаты всех прохождений тестов по каждому сотруднику:

- вычислить средний балл самооценки и оценки коллег;
- определить динамику по периодам;
- сформировать структуру данных для отображения графиков и аналитики на фронтенде.

Триггер

Выгрузка отчета на фронт:

`GET /api/review/analytics?employer_id=15`
или для конкретного периода:

`GET /api/review/analytics?employer_id=15&period_id=3`

Входные параметры

Поле	Тип	Описание
`employer_id`	int	ID сотрудника, для которого собирается аналитика
`period_id`	int (опционально)	Период (1, 3, 6, 12 месяцев и т.д.) — если не задан, возвращаются все

Поле	Тип	Описание
skills_type	enum('hard','soft','all')	По умолчанию 'all' — фильтр по типу навыков

Основная логика

1. Получаем все записи из `Answers_table` для сотрудника

```
SELECT
    a.id_period,
    a.id_question,
    q.id_question_category,
    q.skill_type, -- hard / soft
    a.grade,
    a.id_respondent,
    a.id_employer,
    a.update_time
FROM Answers_Table a
JOIN Questions q ON q.id_question = a.id_question
WHERE a.id_employer = {employer_id}
```

2. Разделяем на self / peer

- Если `a.id_respondent = a.id_employer` → self-test
- Иначе → peer-test

3. Агрегация по категориям и периодам

Для каждого `id_question_category` и `id_period`:

- `avg_self_grade` = средний балл самооценки
- `avg_peer_grade` = средний балл коллег
- `delta` = `avg_peer_grade - avg_self_grade` (отклонение восприятия)
- `trend` = разница между текущим и предыдущим периодом

Пример SQL (упрощённый):

```
SELECT
  q.id_question_category,
  q.skill_type,
  a.id_period,
  AVG(CASE WHEN a.id_employer = a.id_respondent THEN a.grade
END) AS avg_self_grade,
  AVG(CASE WHEN a.id_employer <> a.id_respondent THEN a.grade
END) AS avg_peer_grade
FROM Answers_Table a
JOIN Questions q ON q.id_question = a.id_question
WHERE a.id_employer = {employer_id}
GROUP BY q.id_question_category, q.skill_type, a.id_period;
```

Расчёт динамики

После выборки по периодам данные сортируются по `id_period` (1, 3, 6, 12, …) и к каждому новому периоду добавляется:

```
trend_self = avg_self_grade_current - avg_self_grade_previous
trend_peer = avg_peer_grade_current - avg_peer_grade_previous
```

Выходные данные

```
{
  "status": "success",
  "employer_id": 15,
  "fio": "Иванов Сергей Петрович",
  "position": "Frontend Developer",
  "analytics": [
    {
      "skill_type": "hard",
      "category": "Frontend Development",
      "periods": [
        { "period": "start", "self": 6.0, "peer": null,
"trend_self": null, "trend_peer": null },
        { "period": "1m", "self": 7.0, "peer": 6.5,
"trend_self": +1.0, "trend_peer": null },
        { "period": "3m", "self": 8.0, "peer": 7.2,
"trend_self": +1.0, "trend_peer": +0.7 }
      ]
    },
    {
      "skill_type": "soft",
      "category": "Коммуникация",
      "periods": [
        { "period": "start", "self": 8.5, "peer": null },
        { "period": "1m", "self": 8.0, "peer": 7.8 },
        { "period": "3m", "self": 8.7, "peer": 8.2 }
      ]
    }
  ],
  "averages": {
    "overall_self": 7.7,
    "overall_peer": 7.1,
    "delta": -0.6
  }
}
```

Интерпретация метрик

Показатель	Описание	Риски / возможности
`avg_self_grade`	Средняя самооценка сотрудника	Резкое падение может указывать на демотивацию

Показатель	Описание	Риски / возможности
`avg_peer_grade`	Средняя оценка коллег	Низкая динамика — проблемы с интеграцией в команду
`delta`	Разница восприятия (peer - self)	Если `delta < -1`, сотрудник себя переоценивает
`trend_self`	Изменение самооценки во времени	Рост — положительная адаптация
`trend_peer`	Изменение оценки коллег	Рост — улучшение командного восприятия
`avg_delta_dynamics`	Среднее изменение разницы self/peer	Нормализация — стабилизация восприятия

Связь с другими API

API	Назначение
API #1	Создаёт записи grade=0 и инициирует процесс
API #2	Принимает ответы от сотрудников
API #3	Отдает данные анкеты по токену
API #4	Рассчитывает агрегаты и тренды для аналитики

API 5 — Расчёт Adaptation Index

Цель

Сформировать **единый числовой показатель (0–100)**, отражающий общую динамику адаптации сотрудника, на основе:

- самооценок (`avg_self_grade`);
- оценок коллег (`avg_peer_grade`);
- изменений во времени (`trend_self`, `trend_peer`);
- различий между самооценкой и восприятием командой (`delta`);
- наличия пропусков тестов (ответов с `grade = 0`).

Триггер

Расчёт выполняется:

- **автоматически** после закрытия тестового периода;
- **или вручную** по запросу HR/руководителя.

```
GET /api/review/adaptation_index?employer_id=15
```

Входные параметры

Поле	Тип	Описание
`employer_id`	int	ID сотрудника
`period_id`	int (опционально)	Конкретный период; если не указан — берётся последний доступный
`skills_type`	enum('hard','soft','all')	Фильтр расчёта (по умолчанию `'all'`)

Источник данных

Используются агрегированные данные из **API #4 (аналитика)**:

- `avg_self_grade`, `avg_peer_grade`
- `trend_self`, `trend_peer`
- `delta`
- `grade = 0` (если есть пропуски)

Алгоритм расчёта Adaptation Index

1. Базовый балл

Среднее значение между самооценкой и оценкой коллег:

```
base_score = (avg_self_grade + avg_peer_grade) / 2
```

Переводим шкалу из 10-балльной в 100-балльную:

```
base_score = base_score * 10
```

2. Коррекция по дельте (разнице в восприятии)

Если сотрудник себя переоценивает (delta < -1), — штраф.
Если недооценивает (delta > 1), — бонус, так как чаще всего это значит, что он скромен, но командой оценивается выше.

```
if delta < -1: adjust_delta = -abs(delta) * 2.5
elif delta > 1: adjust_delta = +abs(delta) * 1.5
else: adjust_delta = 0
```

3. Динамика роста

Рост самооценки и восприятия командой добавляют бонус:

```
adjust_trend = (trend_self + trend_peer) * 3
```

Если отрицательные — это, наоборот, штраф.

4. Штраф за пропуски

Если более 20% ответов не заполнено (`grade=0`), сотрудник получает -10 баллов:

```
if unanswered_ratio > 0.2: adjust_missing = -10
else: adjust_missing = 0
```

5. Финальный индекс

```
AdaptationIndex = base_score + adjust_delta + adjust_trend +
adjust_missing
```

Ограничиваем результат в диапазоне `0–100`:

```
if AdaptationIndex > 100: AdaptationIndex = 100
if AdaptationIndex < 0: AdaptationIndex = 0
```

Выходные данные


```
{
  "status": "success",
  "employer_id": 15,
  "fio": "Иванов Сергей Петрович",
  "position": "Frontend Developer",
  "period": "3m",
  "AdaptationIndex": 82.4,
  "components": {
    "base_score": 78.0,
    "adjust_delta": +2.0,
    "adjust_trend": +4.4,
    "adjust_missing": -2.0
  },
  "color_zone": "green",
  "interpretation": "Сотрудник стабильно растёт, оценки коллег и самооценка сбалансированы. Высокая адаптация."
}
```


Зоны восприятия (для фронта)

Зона	Диапазон	Цвет	Интерпретация
🔴 Красная	0–49	#E74C3C	Дезадаптация, потеря мотивации, низкое участие
🟠 Жёлтая	50–69	#F1C40F	Зона внимания, нестабильность или переоценка себя
🟢 Зелёная	70–85	#2ECC71	Успешная адаптация, устойчивая динамика
🔵 Синяя	86–100	#3498DB	Лидерский потенциал, высокая вовлечённость

Пример отчёта по команде

Фронтенд может строить рейтинг сотрудников по индексу:

ФИО	Период	Индекс адаптации	Зона	Динамика
Иванов С.П.	3 мес	82.4	🟢	+5.4
Петров А.Н.	3 мес	68.2	🟠	-3.1
Смирнова Л.В.	3 мес	91.0	🔵	+7.5

Связь с другими API

API	Назначение
API #1	Проверяет периоды и создаёт grade=0
API #2	Сохраняет ответы
API #3	Отдаёт анкету по токену
API #4	Агрегирует результаты и динамику
API #5	Рассчитывает индекс адаптации и определяет зону риска/успеха