**1. Ключевой функционал, который вы, вероятно, хотите воспроизвести**

1. **Создание теста/опроса** в удобной форме (веб-интерфейс, админ-панель).
2. **Разнообразные типы вопросов**:
   * Единственный выбор (radio button)
   * Множественный выбор (checkbox)
   * Открытый ответ (текстовые поля)
   * Сопоставление (matching)
   * Упорядочивание (drag-and-drop)
   * Другое (числовой ввод, шкалы, «правда или ложь» и т.д.)
3. **Гибкие настройки**:
   * Таймер на весь тест или на каждый вопрос.
   * Разрешённое количество попыток.
   * Показывать или не показывать правильные ответы после теста.
   * Случайный порядок вопросов и/или вариантов ответов.
   * Прохождение теста только по ссылке или по коду доступа.
4. **Подсчёт и отображение результатов**:
   * Сразу (после прохождения).
   * После проверки (если вопросы требуют ручной проверки).
   * Гибкая система начисления баллов (например, несколько правильных ответов = несколько баллов).
5. **Аналитика**:
   * Сводная статистика по попыткам.
   * Процент правильных/неправильных ответов.
   * Среднее время прохождения и т.д.
6. **Пользовательские роли**:
   * **Администратор/Автор** теста: создаёт и редактирует тест, просматривает результаты.
   * **Проходящий** (ученик/пользователь): даёт ответы, видит итоговый результат (в зависимости от настроек).

**2. Архитектура приложения и рабочий процесс**

**2.1. Компоненты системы**

1. **Frontend-приложение** (SPA или SSR):
   * Инструменты: React / Vue / Angular / и т.д.
   * Позволяет:
     + Администратору/автору создавать/редактировать тесты.
     + Пользователю — проходить тест, видеть результат.
2. **Backend** (с REST API или GraphQL):
   * Инструменты: NestJS / Express / Django / Laravel / и т.д.
   * Основная логика:
     + Хранение тестов, вопросов, вариантов ответов в базе.
     + Обработка отправленных ответов, расчёт баллов.
     + Управление ролями, авторизацией (через JWT или другой механизм).
     + Сервис статистики (хранение результатов, генерация отчётов).
3. **База данных**:
   * Реляционная (PostgreSQL / MySQL) или NoSQL (MongoDB).
   * Хранит:
     + Таблицу/коллекцию «quizzes» (информация о тесте),
     + Таблицу/коллекцию «questions» (вопросы + их настройки),
     + Таблицу/коллекцию «options» (варианты),
     + Таблицу/коллекцию «results» (результаты прохождения).
4. **Сервер (hosting/облачные сервисы)**:
   * Развёртывание в Docker-контейнерах или Heroku, AWS, GCP и т.д.

**2.2. Жизненный цикл теста**

1. **Создание квиза**:
   * Автор задаёт общие настройки (название, описание, время, количество попыток и т.д.).
   * Устанавливает доступ (общедоступный или по коду/паролю/аккаунту).
2. **Добавление вопросов**:
   * Для каждого вопроса указывается тип (single/multiple/…).
   * Указываются варианты ответов, какая опция(и) правильная(ие), при необходимости — сложность, вес (баллы).
   * Для «открытых» или «эссе» вопросов — указываются критерии оценивания.
3. **Публикация теста**:
   * Тест становится доступным по публичной ссылке / защищённой ссылке / в личном кабинете.
4. **Пользовательское прохождение**:
   * Пользователь авторизуется (или нет — если тест доступен всем).
   * Нажимает «Начать» — при необходимости включается таймер.
   * Отвечает на вопросы. Может быть предусмотрена навигация (перейти к следующему/предыдущему вопросу, пропустить вопрос).
   * Отправляет ответы.
5. **Проверка**:
   * Система автоматически проверяет (если все вопросы машинно проверяемые).
   * Либо автор (или преподаватель) вручную проверяет открытые вопросы.
6. **Отображение результата**:
   * Показать общий балл, детализировать по вопросам, показать правильные ответы (или скрыть — по настройкам).
7. **Сбор и просмотр статистики**:
   * Автор может посмотреть, сколько человек прошли, их результаты, слабые места.
   * Пользователь видит свой итоговый результат (историю прохождений).

**3. Проектирование сущностей и данных (пример на NestJS + SQL)**

Приведу пример, как **одна** из возможных схем может выглядеть в TypeORM (либо в любой другой ORM).

**3.1. Таблица Quiz**

* **id** (pk)
* **title** (string)
* **description** (text, nullable)
* **isPublic** (boolean) — доступен ли тест всем без авторизации
* **accessCode** (string, nullable) — код доступа (если требуется)
* **timeLimit** (number, nullable) — время в секундах (на весь тест)
* **attemptsAllowed** (number, nullable) — сколько попыток доступно (или null = бесконечно)
* **shuffleQuestions** (boolean) — случайный порядок вопросов
* **showCorrectAnswers** (boolean) — показать правильные ответы после прохождения
* **createdAt**, **updatedAt** — timestamps

**3.2. Таблица Question**

* **id** (pk)
* **quizId** (fk -> Quiz.id)
* **text** (string) — формулировка вопроса
* **type** (enum: single\_choice, multiple\_choice, text, matching, ordering, …)
* **score** (number) — количество баллов за правильный ответ
* **order** (number) — позиция вопроса (если не используем shuffle)
* **createdAt**, **updatedAt**

**3.3. Таблица Option**

* **id** (pk)
* **questionId** (fk -> Question.id)
* **text** (string) — текст варианта
* **isCorrect** (boolean) — правильный ли вариант
* **order** (number) — порядок варианта (если хотим фиксировать)
* **createdAt**, **updatedAt**

**3.4. Таблица UserQuizResult**

* **id** (pk)
* **userId** (fk -> User.id)
* **quizId** (fk -> Quiz.id)
* **score** (number) — итоговый балл
* **maxScore** (number) — максимальный балл (могут быть суммарные баллы за вопросы)
* **startTime**, **endTime** (datetime) — время начала и завершения
* **status** (enum: in\_progress, completed, evaluating, …)
* **createdAt**, **updatedAt**

**3.5. Таблица UserQuestionAnswer**

* **id** (pk)
* **userQuizResultId** (fk -> UserQuizResult.id)
* **questionId** (fk -> Question.id)
* **chosenOptionIds** (array of strings/ids) — массив выбранных вариантов, если type = multiple\_choice
* **textAnswer** (string, nullable) — для текстовых ответов
* **score** (number, nullable) — балл за конкретный вопрос (если уже проверено)
* **createdAt**, **updatedAt**

Для сложных типов (matching/ordering) можно либо расширить это поле (хранить объект JSON), либо завести отдельные таблицы.

**4. Endpoints (REST)**

Ниже — **общее** представление о том, какие маршруты могут потребоваться:

1. **[POST] /quizzes** — создать тест.
2. **[GET] /quizzes** — получить список тестов (можно с фильтрами: title, isPublic, пагинация).
3. **[GET] /quizzes/{id}** — получить детальную информацию о тесте (с вопросами, опционально без правильных ответов).
4. **[PATCH] /quizzes/{id}** — изменить тест (для автора/админа).
5. **[DELETE] /quizzes/{id}** — удалить тест.
6. **[POST] /quizzes/{id}/start** — начать прохождение теста (создать запись в UserQuizResult, статус in\_progress).
7. **[POST] /quizzes/{id}/answer** — отправить ответы на один вопрос (или сразу на все).
   * Могут быть разные подходы: помногу или покомпонентно.
8. **[POST] /quizzes/{id}/submit** — завершить тест (фиксируем конец, высчитываем результат).
9. **[GET] /quizzes/{id}/result** — получить результаты (для пользователя, прошедшего тест, или для админа).

Плюс отдельные маршруты для создания/обновления вопросов и вариантов, если нужна детальная ручная настройка в админке.

**5. Логика проверки ответов**

1. **single\_choice** (один правильный вариант):
   * Сравнить chosenOptionId с isCorrect = true.
   * Если совпало — пользователь получает балл (score вопроса).
2. **multiple\_choice** (несколько правильных вариантов):
   * У вопроса может быть несколько isCorrect = true.
   * Сравнить массив выбранных пользователем вариантов с эталонным.
   * Балл может начисляться:
     + Только если пользователь выбрал все правильные и ни одного лишнего.
     + Частичные баллы, если не всё выбрано, но часть есть.
3. **text**:
   * Либо автоматическая проверка (например, сравниваем строки, либо с учётом регистра, либо с использованием алгоритмов близости),
   * Либо ручная оценка преподавателем (статус evaluating -> completed).
4. **matching / ordering**:
   * Хранить правильный порядок/соответствия.
   * Сравнивать массив пользователя с эталонным.
   * При необходимости частичные баллы.
5. **итоговый счёт**:
   * Суммируется балл за каждый вопрос, максимальный балл = сумма всех score вопросов.

**6. Расширенные возможности, как в OnlineTestPad**

1. **Перемешивание вопросов** и **вариантов** (shuffling):
   * На бэкенде (генерируем случайный порядок при start).
   * Или на фронтенде (получаем список и сортируем случайным образом).
2. **Временные ограничения**:
   * Время на весь тест (timeLimit) или на отдельные вопросы.
   * Отслеживать время на фронтенде (таймер).
   * При отправке на бэкенд сверять, не вышло ли время.
3. **Показ/Сокрытие правильных ответов**:
   * Флаг showCorrectAnswers.
   * Если false, то пользователь видит только общие результаты (баллы), но не видит детальную инфу.
4. **Разрешить пересдачу** (несколько попыток):
   * Поле attemptsAllowed и запись в UserQuizResult.
   * Если кол-во попыток исчерпано, не позволяем start новый тест.
5. **Разделение на секции** (в одном тесте может быть несколько блоков):
   * Можно добавить таблицу Section для логической группировки вопросов.

**7. UX и UI**

Как в OnlineTestPad, часто делают:

1. **Админка/Конструктор** теста — drag&drop вопросы, интерфейс с предпросмотром, настройка дизайна.
2. **Публичная страница прохождения**:
   * Шаги теста, прогресс-бар, таймер.
   * Возможность вернуться к предыдущим вопросам (или нет, если запрещено настройками).
   * Подсказки (если разрешено).
3. **Отчёты**:
   * Графики, диаграммы, фильтры по датам, экспорт в Excel/PDF.

**8. Как приступить к созданию подобного проекта**

1. **Спецификация и ТЗ**: максимально детально прописать, какие типы вопросов нужны, какие настройки важны, как должны выглядеть результаты и отчёты, где будет хоститься.
2. **Выбрать стек**:
   * Node.js (NestJS/Express) / Python (Django/Flask) / PHP (Laravel) и т.д.
   * В базе данных учесть, что таблицы для вопросов могут получиться сложными — возможно, понадобится NoSQL (MongoDB) с массивами вложенных опций.
3. **Создать базовую структуру** (контейнеры, auth-модуль, user-модуль, quiz-модуль).
4. **Реализовать CRUD для квизов и вопросов** + логику прохождения/проверки.
5. **Добавить UI**: на React/Vue/Angular (или использовать шаблоны admin panel).
6. **Тестировать и дорабатывать** (т. к. систему тестирования всегда нужно много проверять и фиксировать мелкие нюансы).

**9. Резюме**

Чтобы реализовать систему, похожую на [OnlineTestPad](https://onlinetestpad.com):

1. Разработайте **гибкую модель данных** (Quiz, Question, Option, Result).
2. Учтите **разные типы вопросов**, работу с ограничениями (таймер, попытки, шифрование ссылок, коды доступа).
3. Реализуйте **автоматическую или ручную проверку** результатов.
4. Добавьте **аналитику** и **статистику** (отчёты, графики).
5. Сделайте **удобную админ-панель** для создания тестов и **понятный интерфейс** для прохождения тестов.
6. Позаботьтесь об авторизации, правах доступа и защите данных (особенно, если у вас персональные данные пользователей).

Таким образом, вы получите профессиональную, масштабируемую систему Quiz, аналогичную онлайн-платформам вроде OnlineTestPad.

**1. Общая информация**

**Цель**:  
Создать веб-платформу для конструирования, проведения и анализа онлайн-тестов/опросов. Система должна предоставлять удобный интерфейс для создания тестов с различными типами вопросов и гибкими настройками, а также модуль для отображения результатов и аналитики.

**Основные роли пользователей**:

1. **Администратор/Автор**:
   * Создаёт и редактирует тесты.
   * Управляет вопросами и настройками теста.
   * Просматривает результаты и аналитику.
   * Может управлять пользователями (если предусмотрено управление списком участников).
2. **Пользователь (ученик/респондент)**:
   * Проходит тест (может потребоваться авторизация или код доступа).
   * Получает результат (в зависимости от настроек теста).

**2. Типы вопросов**

Минимальный набор:

1. **С одним правильным ответом (Single Choice)**
   * Формат radio button: пользователь выбирает из списка вариантов ровно один.
2. **С несколькими правильными ответами (Multiple Choice)**
   * Формат checkboxes: пользователь может выбрать несколько вариантов, правильных может быть более одного.
3. **Открытый вопрос (Text / Essay)**
   * Текстовое поле для ответа. Может требовать ручной проверки или автоматической (если допустимы точные ответы по шаблону).
4. **Упорядочивание (Ordering)**
   * Нужно расставить заданные элементы (варианты) в правильной последовательности.
5. **Сопоставление (Matching)**
   * Нужно сопоставить элементы из левой колонки (A) с элементами из правой колонки (B).
6. **Числовой ввод (Numeric Input)**
   * Пользователь вводит число в поле, результат сравнивается с эталоном (± допуск, если требуется).
7. **Шкалы (Scale / Range)**
   * Выбор значения на шкале (например, от 1 до 5). Может быть как вопрос-«оценка», так и вопрос с одним правильным значением.
8. **«Правда или ложь» (True/False)**
   * Быстрый формат: одно утверждение, два варианта («Верно» / «Неверно»).

Возможные дополнительные форматы (по желанию):

* **Загрузка файла (File Upload)** — пользователь прикрепляет файл; итоговая оценка даётся вручную.
* **Вопрос с изображением / аудио / видео** — вложенные медиафайлы (и варианты ответов, возможно, тоже с медиа).

**3. Важные настройки теста**

1. **Время**:
   * **Ограничение по времени на весь тест** (timeLimit, в минутах или секундах).
   * **Ограничение по времени на вопрос** (при необходимости).
   * **Старт и конец доступности теста** (например, тест доступен с 01.01.2025 по 15.01.2025).
2. **Количество попыток (attemptsAllowed)**:
   * Можно ограничить 1–N попыток или разрешить неограниченное количество.
   * Если попыток несколько, нужна логика выбора результата (лучший, последний или среднее).
3. **Случайный порядок (Randomization)**:
   * **Перемешивание вопросов** (shuffleQuestions).
   * **Перемешивание вариантов ответов** (shuffleOptions).
   * Возможность фиксировать порядок для определённых вопросов/вариантов (например, если от этого зависит смысл).
4. **Оценка и показ правильных ответов**:
   * **Автоматическая проверка** (для выборов с явно указанными правильными вариантами).
   * **Ручная проверка** (для эссе или файлов).
   * **Показ правильных ответов** сразу после ответа/по завершении/никогда.
   * **Частичное оценивание** (например, при multiple choice).
5. **Доступ**:
   * **Публичный** — доступен всем по ссылке (без авторизации).
   * **Только по коду** (accessCode) — пользователь вводит код, чтобы начать.
   * **Только авторизованным** — требуется учётная запись (роль user).
   * **Дополнительные ограничения** (только определённым группам пользователей).
6. **Отображение и дизайн**:
   * Количество вопросов на одной странице (все сразу или пошагово).
   * Возможность возвращаться к предыдущим вопросам.
   * Включение подсказок / комментариев (если нужно).

**4. Результаты и отчёты**

1. **Отображение индивидуального результата**:
   * Суммарный балл (score).
   * Количество правильных/неправильных ответов.
   * Детальный разбор по каждому вопросу (если это разрешено настройками).
   * Время прохождения (сколько ушло минут/секунд).
   * Статус: «пройден» / «не пройден» / «на проверке».
2. **Отчёты для администратора**:
   * **Список всех участников**, проходивших тест, их итоговые баллы, время попытки.
   * **Диаграммы** (например, гистограмма распределения баллов).
   * **Фильтры** по дате (прохождения), пользователю, группе, результату.
   * **Экспорт** в Excel / CSV / PDF.
   * **Анализ эффективности вопроса** (какой процент пользователей ответил верно/неверно).
3. **Уведомления** (опционально):
   * Автор получает уведомление (email/SMS/внутреннее) о прохождении теста.
   * Пользователь получает уведомление о результатах (если тест проверяется вручную).

**5. Хостинг и инфраструктура**

1. **Где будет развёрнуто**:
   * **Облачное решение** (AWS, GCP, Azure, DigitalOcean) с возможностью масштабирования.
   * **VPS / VDS**: например, Hetzner, OVH, Selectel и т.д.
   * **Docker-контейнеры**: возможно, использовать Docker Compose для бэкенда, фронтенда, БД.
2. **База данных**:
   * SQL (PostgreSQL, MySQL) или NoSQL (MongoDB).
   * При больших нагрузках — горизонтальное масштабирование.
   * При средней нагрузке — одна инстанция + резервные копии.
3. **Домены и сертификаты**:
   * Привязать собственный домен (например, quiz.example.com).
   * Установить SSL-сертификат (HTTPS) для безопасности.
4. **CI/CD**:
   * Настроить автоматическую сборку и развёртывание (GitLab CI, GitHub Actions, Jenkins).
5. **Резервное копирование** (backup):
   * Регулярная выгрузка базы данных.
   * Хранение бэкапов в отдельном хранилище (S3, Google Cloud Storage).

**6. Техническое задание (ТЗ): основные модули и задачи**

Ниже — **основные модули**, которые необходимо реализовать, и **краткое описание** того, что входит в ТЗ.

**6.1. Модуль аутентификации и авторизации (Auth)**

* Регистрация и вход пользователя (JWT или иной способ).
* Роли: Admin, User (возможно, расширение: Teacher, Student).
* Доступ к созданию/редактированию тестов только для Admin/Teacher.

**6.2. Модуль «Quiz»**

1. **CRUD для тестов**:
   * Создать тест (title, description, настройки).
   * Получить список тестов (с фильтрами).
   * Получить детальный тест (включая вопросы).
   * Обновить тест (изменить настройки).
   * Удалить тест (архивация или полное удаление).
2. **CRUD для вопросов**:
   * Создание, редактирование, удаление вопросов внутри теста.
   * Настройка типа вопроса.
   * Работа с опциями (если single/multiple choice).
3. **Прохождение теста**:
   * Маршрут/метод start — инициализация попытки (UserQuizResult).
   * Логика сохранения ответов (по вопросу или сразу всё).
   * Таймер (при необходимости) — реализация на фронтенде + проверка на бэкенде.
4. **Проверка и подсчёт результатов**:
   * Автоматический подсчёт при machine-checkable вопросах.
   * Ручная проверка для эссе/файлов.
   * Подсчёт итогового score, статус («пройден» / «не пройден»), процент правильных ответов.

**6.3. Модуль «Результаты и отчёты»**

* Хранение попыток в таблице UserQuizResult.
* Возможность администратора/автора просматривать:
  + Список попыток.
  + Детали конкретной попытки (какие ответы дал пользователь).
* Генерация отчётов:
  + Сводная статистика (процент прохождения, средний балл и т.д.).
  + Экспорт в Excel / CSV.
* Визуализация (графики, диаграммы).

**6.4. Модуль «Настройки системы» (опционально)**

* Настройка доступности теста (дата начала/окончания).
* Настройка уведомлений (email-шаблоны).
* Управление пользователями (если нужно).

**7. Дополнительные требования и пожелания**

1. **UI/UX**:
   * Интуитивно понятный интерфейс для создания тестов (drag-and-drop, предварительный просмотр вопроса).
   * Адаптивная верстка (мобильные устройства, планшеты, десктоп).
2. **Безопасность**:
   * Хранение паролей в хешированном виде.
   * Защита от SQL-инъекций, XSS, CSRF.
   * Защита от перебора кодов доступа.
3. **Производительность**:
   * Возможность прохождения теста большим количеством пользователей одновременно.
   * При необходимости — кэширование популярных тестов (Redis).
4. **Логи**:
   * Запись действий пользователей (кто создал/изменил тест, время прохождения).
   * Системные логи для быстрого обнаружения ошибок.
5. **Масштабируемость**:
   * Горизонтальное масштабирование backend (несколько инстансов за балансировщиком).
   * Распределённая БД при больших объёмах (шардирование).

**8. Итог**

Данная **Спецификация и ТЗ** описывает все ключевые аспекты создания системы онлайн-тестирования:

* Набор **типов вопросов** (single/multiple choice, текст, сопоставление и т.д.).
* **Настройки** теста (время, попытки, рандомизация, доступ).
* **Отчёты** и **аналитика** (статистика, экспорт, визуализация).
* **Хостинг** и инфраструктура (облачное развёртывание с резервным копированием).

В результате должна получиться многофункциональная платформа, позволяющая проводить опросы и тестирования различных форматов, обрабатывать и анализировать результаты, управлять доступом пользователей и предоставлять удобный интерфейс для всех ролей.

**1. Подготовка базы данных и сущностей (Entity/Model)**

1. **Добавить таблицы (или коллекции)** для:
   * **Quiz** (тесты/опросы)
   * **Question** (вопросы)
   * **Option** (варианты ответов)
   * **UserQuizResult** (результаты прохождения)
   * (При необходимости) **UserQuestionAnswer** (ответы на уровне вопроса)
2. **Определить связи**:
   * Quiz : Question (один-ко-многим)
   * Question : Option (один-ко-многим)
   * UserQuizResult : Quiz (многие-к-одному)
   * UserQuizResult : User (многие-к-одному)
   * UserQuestionAnswer : UserQuizResult (многие-к-одному), и т.д.
3. **Описать поля** (title, description, timeLimit, isPublic, shuffleQuestions, etc.) и необходимые индексы.

**Что делаем**:

* Создаём миграции (если используем TypeORM/Sequelize) или схемы (если Mongo/Mongoose).
* Проверяем, чтобы поля совпадали с бизнес-требованиями (время, ограничение попыток, показывать ли правильные ответы и т.д.).

**2. Создание контроллера и сервиса для Quiz**

После того, как модель данных готова, приступаем к написанию **QuizController** и **QuizService** (либо аналогичных компонентов, если другой фреймворк).

1. **QuizController** (REST-эндпоинты):
   * POST /quizzes — создание квиза
   * GET /quizzes — получение списка квизов (с пагинацией и фильтрами)
   * GET /quizzes/:id — детальный просмотр квиза
   * PATCH /quizzes/:id — обновление квиза
   * DELETE /quizzes/:id — удаление квиза
   * Дополнительно:
     + POST /quizzes/:id/questions — добавление вопросов к квизу
     + PATCH /quizzes/:id/questions/:questionId — обновление вопроса
     + DELETE /quizzes/:id/questions/:questionId — удаление вопроса
     + и т.д.
2. **QuizService** (бизнес-логика):
   * createQuiz(dto: CreateQuizDto), updateQuiz(id, dto), removeQuiz(id), …
   * addQuestion(quizId, dto), updateQuestion(quizId, questionId, dto), removeQuestion(quizId, questionId), …
   * Проверка прав (автор/админ) — можно использовать Guards или метод-обёртку (в зависимости от текущей архитектуры).

**Что делаем**:

* Создаём **DTO** (CreateQuizDto, UpdateQuizDto, CreateQuestionDto и т.д.).
* Указываем валидацию (class-validator, JOI или иные инструменты).
* Интегрируем с **Auth**: маршруты создания и обновления квизов доступны только авторизованным пользователям (и, например, только с ролью «admin» или «teacher»).

**3. Модуль Questions/Options (если нужно отдельно)**

Иногда логику CRUD для вопросов/вариантов удобнее вынести в отдельный под-модуль или в тот же QuizService, но отдельными методами. Это зависит от сложности проекта.

* **Controller**: QuizQuestionsController или просто роуты внутри QuizController.
* **Service**: QuizQuestionsService или общие методы в QuizService.

**Что делаем**:

* Создаём эндпоинты для добавления, изменения и удаления вопросов (и опций).
* Продумываем, как будем работать с «перемешиванием» вариантов, порядком вопросов и т.д.

**4. Реализация логики прохождения теста (submit / start / finish)**

Чтобы пользователь мог **проходить** тест, нужны эндпоинты, например:

1. POST /quizzes/:id/start
   * Создаём запись в UserQuizResult (статус «in\_progress»), фиксируем время старта, userId, quizId.
2. POST /quizzes/:id/answer или PATCH /quizzes/:id/answer
   * Пользователь отправляет ответы (либо по одному вопросу, либо все сразу).
   * Сохраняем в UserQuestionAnswer (или в JSON-колонке внутри UserQuizResult).
   * Если вопросы «машинно» проверяемые, можем сразу подсчитать баллы. Если нет (эссе) — ждём ручной проверки.
3. POST /quizzes/:id/submit
   * Завершаем попытку (статус «completed»), фиксируем время окончания, рассчитываем итоговый score.
   * Возвращаем результат пользователю: сколько набрал, какие ответы правильные/неправильные (если это разрешено настройками).

**Что делаем**:

* Создаём **SubmitQuizDto**, где описаны ответы пользователя (questionId -> array of chosenOptionIds/textAnswer).
* В сервисе реализуем логику подсчёта баллов по каждому вопросу (single, multiple, text, matching и т.д.).
* Учитываем настройки (timeLimit, attemptsAllowed, randomizeQuestions, showCorrectAnswers и т.п.).

**5. Отчёты и статистика**

1. **Сводная статистика**:
   * GET /quizzes/:id/results — список попыток, их результаты, пользователь и т.д.
   * Диаграммы (на фронтенде) или экспорт в CSV/Excel/PDF (на бекенде).
2. **Детальный отчёт**:
   * GET /quizzes/:quizId/results/:resultId — полный отчёт по отдельной попытке, все ответы и баллы.

**Что делаем**:

* В **QuizService** или **ResultService** реализуем методы для вытаскивания данных, агрегации (например, средний балл, процент правильных ответов).
* При необходимости добавляем фильтрацию (по дате, по пользователю, по статусу).

**6. Тесная интеграция с Auth и User модулями**

1. **Роли и права**:
   * Убедиться, что **Админ** (или «teacher») может создавать и редактировать квизы, а **обычный пользователь** («student») — только проходить.
   * Использовать Guards и декораторы (например, @Roles('admin')).
2. **Привязка результатов к пользователю**:
   * В UserQuizResult хранить userId (FK на users).
   * При получении списка результатов фильтровать, кто может смотреть.
3. **Профиль пользователя**:
   * Можно выводить историю попыток пользователя (GET /users/me/quizzes/results) или что-то подобное.

**7. Тестирование и отладка**

1. **Unit-тесты**:
   * Покрыть основные методы сервиса (createQuiz, submitQuiz, подсчёт баллов).
2. **E2E-тесты** (если в проекте принята такая практика):
   * Проверить сценарии (автор создаёт квиз -> пользователь проходит -> результат).
3. **Swagger**:
   * Добавить документацию для всех новых эндпоинтов (Quiz, Questions, Submit и т.д.).
   * Убедиться, что у вас единый Swagger/OpenAPI JSON, включающий Auth, User, Quiz.

**8. Развёртывание (Deploy)**

1. **Подготовить миграции** (если нужна новая схема БД).
2. **Обновить конфиги** (env-файлы, переменные окружения).
3. **Запустить** новые образы (Docker) или обновить существующий сервер.
4. **Проверить** в тестовом окружении (staging/preprod) и затем вывести в **production**.

**9. Общий итог**

После того как **Auth** и **User** уже есть, логично:

1. **Определить** сущности и связи для квиза (Quiz, Question, Options, Results).
2. **Реализовать** контроллер и сервис Quiz:
   * CRUD для тестов, CRUD для вопросов.
   * Прохождение теста, отправка ответов, подсчёт результатов.
3. **Интегрировать** с Auth (проверка ролей и доступов) и User (привязка результатов).
4. **Добавить** аналитику и отчёты, экспорт данных (при необходимости).
5. **Написать тесты** и задокументировать (Swagger/OpenAPI).

Таким образом, вы получите полнофункциональный модуль «Quiz», встроенный в вашу