🧪 Тестовое задание: “Мини-система турниров”

🎯 Цель

Создать минимальное backend-приложение на FastAPI, моделирующее регистрацию игроков в турниры. Задание помогает оценить:

- Понимание async, FastAPI, SQLAlchemy 2.0
- Умение работать с типами, линтерами, тестами
- Навыки структурирования кода
- Готовность к обучению и внимательность

⚙️ Технологические требования

Обязательно:

- Python 3.11+
- FastAPI
- Async SQLAlchemy 2.0+
- Alembic (миграции)
- Pydantic
- PostgreSQL
- Docker + Docker Compose
- Mypy (типизация)
- Ruff (линтер)
- Black (форматирование)
- Pytest (тесты)

Будет плюсом:

- DDD-подход (минимально)
- Асинхронная реализация всех операций
- Уверенное использование Alembic
- Чистая структура проекта (отделение слоёв)

🧩 Задача: Мини-турнирная система

Создать API для управления турнирами и регистрации игроков в них.

📌 Функциональные требования

1. **Создание турнира**
    - `POST /tournaments`
    - Принимает: название, максимальное число игроков, дата старта (UTC).
    - Возвращает: ID и детали турнира.
2. **Регистрация игрока**
    - `POST /tournaments/{tournament_id}/register`
    - Принимает: имя игрока, email.
    - Проверка:
        - Нельзя зарегистрировать больше игроков, чем указано в лимите.
        - Один email может участвовать в одном турнире только один раз.
3. **Список зарегистрированных игроков**
    - `GET /tournaments/{tournament_id}/players`
    - Возвращает список игроков с их именами и email.

📁 Структура проекта (рекомендуемая)

```
project/
├── app/
│   ├── main.py
│   ├── models/
│   │   └── tournament.py
│   ├── schemas/
│   │   └── tournament.py
│   ├── repositories/
│   │   └── tournament.py
│   ├── services/
│   │   └── tournament.py
│   ├── api/
│   │   └── tournament.py
│   ├── db.py
│   └── config.py
├── alembic/
├── tests/
│   └── test_registration.py
├── docker-compose.yml
├── Dockerfile
├── pyproject.toml
└── README.md
```

📥 Входные данные

Пример запроса на создание турнира:

```
POST /tournaments
{
  "name": "Weekend Cup",
  "max_players": 8,
  "start_at": "2025-06-01T15:00:00Z"
}
```

Пример запроса на регистрацию игрока:

```
POST /tournaments/1/register
{
  "name": "John Doe",
  "email": "john@example.com"
}
```

📤 Выходные данные

Пример успешного ответа:

```
{
  "id": 1,
  "name": "Weekend Cup",
  "max_players": 8,
  "start_at": "2025-06-01T15:00:00Z",
  "registered_players": 1
}
```

✅ Что обязательно должно быть:

- **Асинхронные обработчики** (`async def`)
- **Миграции Alembic** (миграция таблиц `tournaments`, `players`)
- **Валидации с Pydantic**
- **Ручка с проверкой лимита игроков**

- **Обработка ошибок** (422, 400 и т.п.)
- **Минимум один тест на регистрацию**
- **Линтинг и типизация без ошибок**
- **README.md с инструкцией по запуску**

Дополнительные требования

- Код должен быть читаемым и разделённым на слои (API, сервисы, репозитории, модели).
- Желательно использовать DI (зависимости через Depends).
- Никаких фреймворков кроме указанных (никакого Django, ORMs, кроме SQLAlchemy).
- Желательно покрытие тестами хотя бы основной логики.
- Приветствуется использование `loguru` (если знакомы).

Что прислать

- Ссылку на публичный GitHub/GitLab репозиторий
- Все необходимые файлы для локального запуска: `Dockerfile`, `docker-compose.yml`, `alembic`, `.env.example`
- README с командами:
  - `make dev` / `docker-compose up`
  - `alembic upgrade head`
  - `pytest`

Критерии оценки

Критерий	Вес
Корректность логики	40%
Чистота и структура кода	20%
Работа с базой и Alembic	15%
Работа с типами и линтерами	10%
Наличие тестов	10%
README и удобство запуска	5%
