### **Тестовое задание**

#### **Описание задания**

Во вселенной «Кокоди» существует карточная игра, где несколько пользователей **собирают очки**, разыгрывая специальные карты. Для упрощённого прототипа выделим следующие элементы:

1. **Игрок** (User):  
   * У каждого игрока есть уникальный **id** и **имя**.
   * Игрок может участвовать в нескольких партиях, но внутри одной партии он участвует только один раз.
2. **Партия** (Game Session):  
   * Имеет уникальный **id**.
   * Может содержать от 2 до 4 игроков (включительно).
   * Имеет состояние (например, «waitForPlayers», «inProgress», «finished»).
   * Хранит **колоду** (deck) специальных карт.
   * Хранит историю **ходов** (turns) для отладки/аналитики.
3. **Карты** (Cards):  
   * Бывают двух типов:  
     + **Points Card**: При розыгрыше даёт игроку X очков.
     + **Action Card**: Оказывает эффект на других игроков (например, заставляет пропустить ход или отнять у игрока X очков).
   * У каждой карты есть уникальное название (name) и числовой параметр (value), используемый по-разному в зависимости от типа карты.
4. **Ход** (Turn):  
   * В каждый ход активный игрок берёт верхнюю карту колоды.
   * Если это **Points Card**, игроку добавляются очки, равные value.
   * Если это **Action Card** — нужно применить уникальный эффект (см. раздел “Правила для Action Card” ниже).
   * После применения эффекта карта **сбрасывается** (т. е. из колоды удаляется).
   * В конце хода проверяется, не достиг ли игрок лимита в 30 очков (или любой другой заданной константы) — если да, игра завершается.
5. **Правила для Action Card**:  
   * Если это карта «Block» с value = 1, то **следующий** игрок пропускает ход (в логике нужно предусмотреть, как это отразится на порядке ходов).
   * Если это карта «Steal» с value = N, то активный игрок **отнимает** у выбранного соперника N очков (не позволяя при этом уходить в минус: минимум 0).
   * Если это карта «DoubleDown» с value = 2, то активный игрок удваивает свою текущую сумму очков, но не более чем до 30.

Все «уникальные эффекты» нужно **продумать** и корректно **реализовать** внутри кода.

#### **Задача**

Нужно написать **микросервис на Kotlin** (любой фреймворк на выбор), который реализует **регистрацию** игроков, **управление** партиями и **логику** ходов.

Минимальный набор требований:

1. **Регистрация и авторизация**:  
   * Endpoint для **создания** пользователя (login, пароль, имя).
   * Хранение пользователей в SQLite/PostgreSQL (или в памяти, если нет времени на полноценную БД), но с чёткой структурой таблиц/классов.
   * При первом знакомстве с системой пользователь регистрируется, при повторном — авторизуется (предпочтительно JWT).
2. **Управление партиями**:  
   * Endpoint для **создания** новой партии. Создатель становится первым игроком.
   * Endpoint для **присоединения** к партии по её id, если в ней меньше 4 игроков.
   * Endpoint/метод для **старта** партии (если игроков хотя бы 2).
   * При старте генерируется/заполняется **колода**: придумайте от 5 до 10 уникальных карт (карты должны включать оба типа “points” и “actions”). Порядок колоды случайный.
3. **Ходы**:  
   * Endpoint POST /games/{gameId}/turn (или аналог). Должен:  
     1. Проверять, что сейчас ход **данного** игрока.
     2. «Вытягивать» верхнюю карту из колоды и применять её эффект.
     3. Пропускать ход или менять очки другим игрокам, если это Action Card.
     4. Записывать ход (какая карта, на кого эффект, сколько очков теперь у игрока и т. п.).
     5. Проверять условие завершения игры (если у кого-то стало >= 30 очков).
   * Если игра завершена, вызов этого эндпоинта должен возвращать ошибку или сообщение «Game is finished».
4. **Получение статуса**:  
   * Endpoint, возвращающий текущее состояние партии: список игроков с их очками, состояние колоды (сколько карт осталось), статус (в процессе / завершена), кто ходит следующим и т. п.
5. **Тесты**:  
   * Хотя бы **пара тестов** (юнит или интеграционные) на:  
     1. Создание партии и присоединение игроков.
     2. Пример «хода», где разыгрывается Action Card, и проверяется, что очки действительно убыли/увеличились как надо.

#### **Детали, которые нужно продумать самому**

* Где и как хранить текущую очередь ходов (учитывая «Block», пропуски и т. п.).
* Как обрабатывать ситуацию, когда у соперника недостаточно очков (при отнятии).
* Как **не допустить** повторных ходов одного игрока без очереди.
* Как именно представлять данные: структуры, таблицы, объекты.
* Формат входных/выходных запросов (JSON-схемы).
* Какой способ авторизации/аутентификации использовать.

#### **Требования к проекту**

1. **Код на Kotlin**.
2. Сервер (можно Ktor, Spring Boot, Micronaut — на выбор).
3. Минимум зависимостей. Можно использовать любую библиотеку для JSON-сериализации.
4. **Docker**-файл или инструкция, как запустить.
5. Краткая документация (README.md) — какие endpoint’ы есть и для чего.

#### **Критерии оценки**

* **Полнота** реализации требований (регистрация, создание партий, ходы, счётчик очков, конечные условия).
* **Аккуратность** кода: нейминг, структура классов, понятность.
* Наличие хоть каких-то **юнит/интеграционных тестов**.
* Удобство запуска (настроенный gradle/maven, Docker, инструкция).
* Умение **не усложнять**: хорошая архитектура «ровно в меру».

#### **Что проверяет задание**

* Знание Kotlin на уровне классов, функций, коллекций и пр.
* Понимание основ объектно-ориентированного проектирования, умение продумать модель данных.
* Умение работать с веб-фреймворком (Ktor/Spring Boot).
* Умение организовать простую логику игры и учесть исключительные ситуации.
* Базовую работу с тестами и, по возможности, с базой данных.