### **Задача**

Создать API endpoint, который:

1. Принимает POST-запросы без авторизации с сообщением из внешнего мессенджера содержащим следующие параметры:
   * external\_message\_id:string (уникальный внешний идентификатор сообщения - 16 байт)
   * external\_client\_id:string (уникальный внешний идентификатор клиента - 16 байт)
   * client\_phone:string (номер телефона с +7. Пример - +79194961182. Допустим, что номера могут быть только российские (12 символов))
   * message\_text:string (текст сообщения - до 4096 символов)
   * send\_at:integer (дата-время отправки в формате unixtime).
2. Сохраняет данные в 3 таблицы:
   * **clients**: информация о клиентах.
   * **dialogs**: диалоги, связанные с клиентами.
   * **messages**: сообщения внутри диалогов.
3. Обрабатывает дубликаты, обеспечивает целостность данных и высокую производительность.

### **Требования**

1. **Условия**
   * Данные поступают с высокой скоростью (1k RPS)
   * Вебхуки могут содержать неполные данные (в таком случае логгируем ошибку и отбрасываем их)
   * Данные не гарантируют упорядоченности по времени отправки
   * Данные могут дублироваться (Повторные вебхуки, с повторяющимся message\_id)
2. **Структура БД:**
   * Продумать связи между таблицами, индексы, ограничения уникальности.
3. **Логика:**
   * Для каждого клиента (определяемого external\_client\_id) создается уникальная запись в clients.
   * Каждый клиент имеет ровно один диалог (создается при первом сообщении).
   * Сообщения добавляются в диалог с проверкой уникальности external\_message\_id.
4. **Производительность:**
   * Решение для обработки высокой нагрузки
   * Минимизация запросов к БД
5. **Дополнительно:**
   * Валидация входных данных
   * Обработка ошибок

### **Ожидаемое решение**

1. **Код:**
   * Проект в виде набора PHP-скриптов или на фреймворке (Yii2/Laravel).
   * Использование Postgresql для постоянного хранения данных (Допустимо другое решение, но его необходимо обосновать. Оно обязательно должно быть реляционным)
   * Описание структуры БД в виде миграции.
   * Пример реализации с использованием PDO или ORM.
   * Наличие phpDoc к классам, атрибутам, методам
2. **Архитектура:**
   * Разделение на слои (контроллеры, сервисы, модели).
3. **Пояснения:**
   * Описание выбранных подходов к обработке нагрузки.
   * Обоснование структуры БД и индексов.
   * Возможные улучшения (кеширование, шардирование, репликация, что-то ещё).

### **Критерии оценки**

1. Корректность работы (целостность данных, обработка дубликатов, решение проблем конкурентной обработки).
2. Производительность.
3. Чистота кода.
4. Масштабируемость (возможность обработки 10k+ RPS).
5. Документация и пояснения.

Результат должен быть предоставлен в виде репозитория на GitHub/GitLab.