Техническое задание

Для реализации функционала по заказу курьера для доставки корреспонденции контрагентам, необходимо реализовать хранение и получение данных по заявкам.
На данный момент уже реализованы фронтенд и бэкенд.

Задача:
1. Используя сервис https://supabase.com/ нужно поднять облачную базу данных PostgreSQL.

2. Для доступа к данным в базе данных должен быть создан пользователь
логин: netocourier
пароль: NetoSQL2022
права: полный доступ на схему public, к information_schema и pg_catalog права только на чтение, предусмотреть доступ к иным схемам, если они нужны.

3. Должны быть созданы следующие отношения:

courier: --данные по заявкам на курьера
id uuid PK
from_place varchar --откуда
where_place varchar --куда
name varchar --название документа
account_id uuid FK --id контрагента
contact_id uuid FK --id контакта
description text --описание
user_id uuid FK --id сотрудника отправителя
status enum -- статусы 'В очереди', 'Выполняется', 'Выполнено', 'Отменен'. По умолчанию 'В очереди'
created_date date --дата создания заявки, значение по умолчанию now()

account: --список контрагентов
id uuid PK
name varchar --название контрагента

contact: --список контактов контрагентов
id uuid PK
last_name varchar --фамилия контакта
first_name varchar --имя контакта
account_id uuid FK --id контрагента

user: --сотрудники
id uuid PK
last_name varchar --фамилия сотрудника
first_name varchar --имя сотрудника
dismissed boolean --уволен или нет, значение по умолчанию "нет"

4. Для генерации uuid необходимо использовать функционал модуля uuid-ossp, который уже подключен в облачной базе.

5. Для формирования списка значений в атрибуте status используйте create type ... as enum

6. Для возможности тестирования приложения необходимо реализовать процедуру insert_test_data(value), которая принимает на вход целочисленное значение.
Данная процедура должна внести:
value * 1 строк случайных данных в отношение account.
value * 2 строк случайных данных в отношение contact.
value * 1 строк случайных данных в отношение user.
value * 5 строк случайных данных в отношение courier.
- Генерация id должна быть через uuid-ossp
- Генерация символьных полей через конструкцию SELECT repeat(substring('абвгдеёжзийклмнопрстуфхцчшщьыъэюя',1,(random()*33)::integer),(random()*10)::integer);
Соблюдайте длину типа varchar. Первый random получает случайный набор символов из строки, второй random дублирует количество символов полученных в substring.
- Генерация булева типа происходит через 0 и 1 с использованием оператора random.
- Генерацию даты и времени можно сформировать через select now() - interval '1 day' * round(random() * 1000) as timestamp;
- Генерацию статусов можно реализовать через enum_range()

7. Необходимо реализовать процедуру erase_test_data(), которая будет удалять тестовые данные из отношений.

8. На бэкенде реализована функция по добавлению новой записи о заявке на курьера:
```
function add($params) --добавление новой заявки
   {
      $pdo = Di::pdo();
      $from = $params["from"];
      $where = $params["where"];
      $name = $params["name"];
      $account_id = $params["account_id"];
      $contact_id = $params["contact_id"];
      $description = $params["description"];
      $user_id = $params["user_id"];
      $stmt = $pdo->prepare('CALL add_courier (?, ?, ?, ?, ?, ?, ?)');
      $stmt->bindParam(1, $from); --from_place
      $stmt->bindParam(2, $where); --where_place
      $stmt->bindParam(3, $name); --name
      $stmt->bindParam(4, $account_id); --account_id
      $stmt->bindParam(5, $contact_id); --contact_id
      $stmt->bindParam(6, $description); --description
      $stmt->bindParam(7, $user_id); --user_id
      $stmt->execute();
   }
```
Нужно реализовать процедуру add_courier(from_place, where_place, name, account_id, contact_id, description, user_id),

которая принимает на вход вышеуказанные аргументы и вносит данные в таблицу courier
Важно! Последовательность значений должна быть строго соблюдена, иначе приложение работать не будет.

9. На бэкенде реализована функция по получению записей о заявках на курьера:
static function get() --получение списка заявок
```
  {
    $pdo = Di::pdo();
    $stmt = $pdo->prepare('SELECT * FROM get_courier()');
    $stmt->execute();
    $data = $stmt->fetchAll();
    return $data;
  }
```
Нужно реализовать функцию get_courier(), которая возвращает таблицу согласно следующей структуры:
id --идентификатор заявки
from_place --откуда
where_place --куда
name --название документа
account_id --идентификатор контрагента
account --название контрагента
contact_id --идентификатор контакта
contact --фамилия и имя контакта через пробел
description --описание
user_id --идентификатор сотрудника
user --фамилия и имя сотрудника через пробел
status --статус заявки
created_date --дата создания заявки
Сортировка результата должна быть сперва по статусу, затем по дате от большего к меньшему.
Важно! Если названия столбцов возвращаемой функцией таблицы будут отличаться от указанных выше, то приложение работать не будет.

10. На бэкенде реализована функция по изменению статуса заявки.
function change_status($params) --изменение статуса заявки
```
  {
    $pdo = Di::pdo();
    $status = $params["new_status"];
    $id = $params["id"];
    $stmt = $pdo->prepare('CALL change_status(?, ?)');
    $stmt->bindParam(1, $status); --новый статус
    $stmt->bindParam(2, $id); --идентификатор заявки
    $stmt->execute();
  }
```
Нужно реализовать процедуру change_status(status, id), которая будет изменять статус заявки. На вход процедура принимает новое значение статуса и значение идентификатора заявки.

11. На бэкенде реализована функция получения списка сотрудников компании.
static function get_users() --получение списка пользователей

```
{
    $pdo = Di::pdo();
    $stmt = $pdo->prepare('SELECT * FROM get_users()');
    $stmt->execute();
    $data = $stmt->fetchAll();
    $result = [];
    foreach ($data as $v) {
        $result[] = $v['user'];
    }
    return $result;
}
```

Нужно реализовать функцию get_users(), которая возвращает таблицу согласно следующей структуры:
user --фамилия и имя сотрудника через пробел
Сотрудник должен быть действующим! Сортировка должна быть по фамилии сотрудника.

12. На бэкенде реализована функция получения списка контрагентов.
static function get_accounts() --получение списка контрагентов

```
{
    $pdo = Di::pdo();
    $stmt = $pdo->prepare('SELECT * FROM get_accounts()');
    $stmt->execute();
    $data = $stmt->fetchAll();
    $result = [];
    foreach ($data as $v) {
        $result[] = $v['account'];
    }
    return $result;
}
```

Нужно реализовать функцию get_accounts(), которая возвращает таблицу согласно следующей структуры:
account --название контрагента
Сортировка должна быть по названию контрагента.

13. На бэкенде реализована функция получения списка контактов.
function get_contacts($params) --получение списка контактов

```
{
    $pdo = Di::pdo();
    $account_id = $params["account_id"];
    $stmt = $pdo->prepare('SELECT * FROM get_contacts(?)');
    $stmt->bindParam(1, $account_id); --идентификатор контрагента
    $stmt->execute();
    $data = $stmt->fetchAll();
    $result = [];
    foreach ($data as $v) {
        $result[] = $v['contact'];
    }
```

```
    return $result;
  }
```
Нужно реализовать функцию get_contacts(account_id), которая принимает на вход идентификатор контрагента и возвращает таблицу с контактами переданного контрагента согласно следующей структуры:
contact --фамилия и имя контакта через пробел
Сортировка должна быть по фамилии контакта. Если в функцию вместо идентификатора контрагента передан null, нужно вернуть строку 'Выберите контрагента'.

14. На бэкенде реализована функция по получению статистики о заявках на курьера:
```
static function get_stat() --получение статистики
  {
    $pdo = Di::pdo();
    $stmt = $pdo->prepare('SELECT * FROM courier_statistic');
    $stmt->execute();
    $data = $stmt->fetchAll();
    return $data;
  }
```
Нужно реализовать представление courier_statistic, со следующей структурой:
account_id --идентификатор контрагента
account --название контрагента
count_courier --количество заказов на курьера для каждого контрагента
count_complete --количество завершенных заказов для каждого контрагента
count_canceled --количество отмененных заказов для каждого контрагента
percent_relative_prev_month -- процентное изменение количества заказов текущего месяца к предыдущему месяцу для каждого контрагента, если получаете деление на 0, то в результат вывести 0.
count_where_place --количество мест доставки для каждого контрагента
count_contact --количество контактов по контрагенту, которым доставляются документы
cansel_user_array --массив с идентификаторами сотрудников, по которым были заказы со статусом "Отменен" для каждого контрагента