**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/](https://supabase.com/pricing) 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