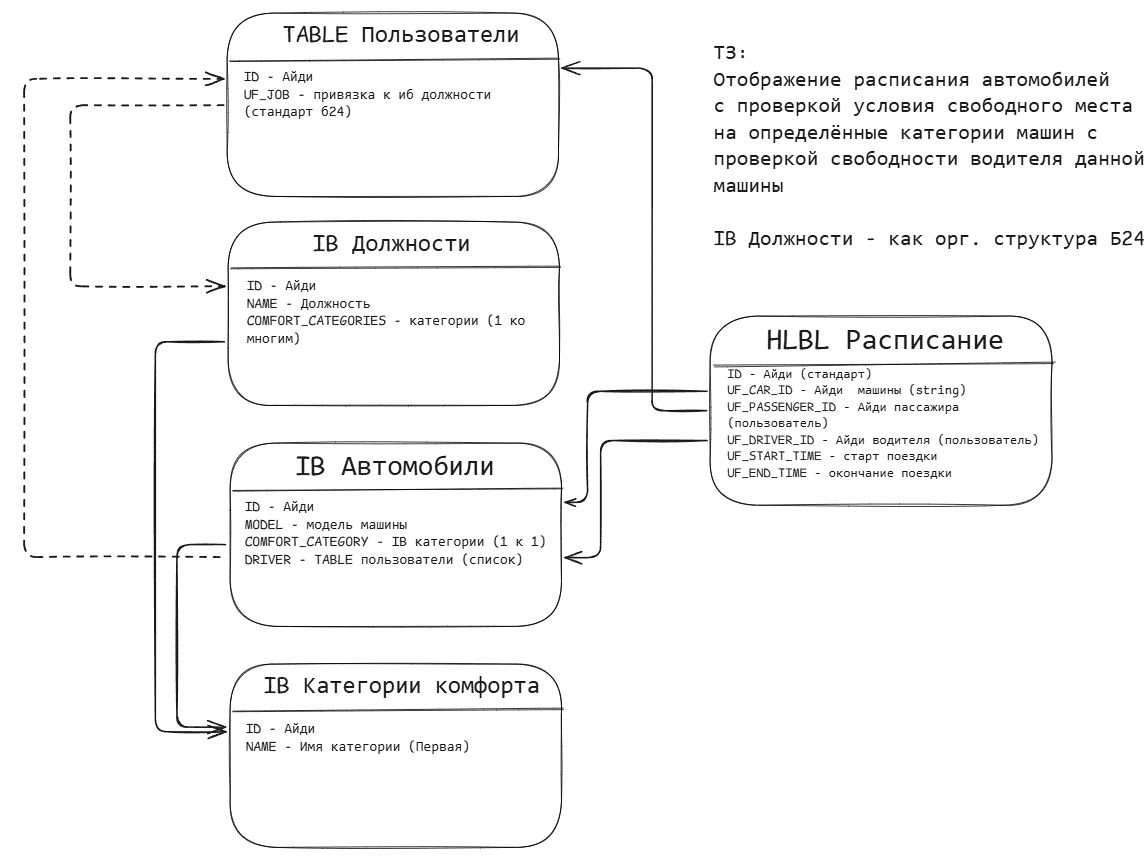
1. Схема



1. Код

Написать компонент, который выводит для текущего сотрудника список свободных автомобилей на запрошенное время (запрошенное время начала и время окончания поездки передается от клиента на сервер в get-параметрах) с указанием модели, категории комфорта, водителя

Фильтр по времени + проверка по должности на категорию и свободность водителя

Файл class.php

<?php  
if (!defined('B\_PROLOG\_INCLUDED') || *B\_PROLOG\_INCLUDED* !== true) die();  
  
use Bitrix\Main\Loader;  
use Bitrix\Main\Type\DateTime;  
use Bitrix\Main\ArgumentException;  
use Bitrix\Main\SystemException;  
  
class AvailableCarsComponent extends CBitrixComponent  
{  
 protected $userId;  
 protected $start;  
 protected $end;  
 private const *HLBL\_CODE* = 'Schedule';  
  
 /\*  
 📘 Предполагаемая структура данных:  
  
 🧩 Инфоблоки (IBLOCKS):  
 1. Должности (Positions)  
 - ID (int) — идентификатор должности  
 - NAME (string) — название должности  
 - COMFORT\_CATEGORIES (multiple link) — привязка к "Категории комфорта"  
  
 2. Автомобили (Cars)  
 - ID (int) — идентификатор автомобиля  
 - NAME (string) — модель автомобиля  
 - COMFORT\_CATEGORY (link) — привязка к категории комфорта  
 - DRIVER (link to user) — водитель, закреплённый за машиной  
  
 3. Категории комфорта (ComfortCategories)  
 - ID (int) — идентификатор категории  
 - NAME (string) — название категории (Первая, Вторая и т.д.)  
  
 🚀 Highload-блок (HLBL):  
 Schedule  
 - UF\_CAR\_ID (string/int) — ID автомобиля  
 - UF\_DRIVER\_ID (int) — ID водителя  
 - UF\_PASSENGER\_ID (int) — ID сотрудника, который поедет  
 - UF\_START\_TIME (datetime) — время начала поездки  
 - UF\_END\_TIME (datetime) — время окончания поездки  
  
 👤 Пользователи (TABLE: b\_user)  
 - ID (int)  
 - UF\_JOB (link to IBLOCK Positions)  
  
 ⚙️ Логика связей:  
 - Каждый пользователь имеет должность (UF\_JOB) (Стандарт Битрикс24)  
 - Должность определяет доступные категории комфорта  
 - Каждая машина относится к одной категории комфорта  
 - Каждая машина имеет только одного водителя  
 - Один водитель может быть закреплён только за одной машиной  
 - В расписании (Schedule) фиксируются занятые машины на определённый интервал времени  
  
 📅 Условия работы компонента:  
 - Пользователь передаёт GET-параметры ?start=HH:MM:SS&end=HH:MM:SS  
 - Компонент определяет должность пользователя → категории комфорта → машины этих категорий  
 - Из списка машин исключаются те, что уже заняты в указанный интервал времени  
 - Результат — список доступных для бронирования автомобилей  
 \*/  
  
  
 public function executeComponent(): void  
 {  
 try {  
 $this->initModules();  
 $this->getUser();  
 $this->parseTimes();  
  
 $positionId = $this->getUserPositionId();  
 if (!$positionId) {  
 throw new \DomainException('Для пользователя не задана должность (UF\_JOB).');  
 }  
  
 $comfortIds = $this->getCategoryIds($positionId);  
 if (empty($comfortIds)) {  
 $this->viewResult(  
 []  
 );  
 return;  
 }  
  
 $cars = $this->getCarsByComfortIds($comfortIds);  
 if (empty($cars)) {  
 $this->viewResult(  
 []  
 );  
 return;  
 }  
  
 $busyCars = $this->getBusyCarsAndDrivers($this->start, $this->end);  
  
 // Финальный вывод  
 $this->viewResult(  
 $this->filterAvailableCars($cars, $busyCars)  
 );  
  
 } catch (\Exception $e) {  
 print\_r($e->getMessage());  
 }  
 }  
  
 protected function getUserPositionId(): ?int  
 {  
 $user = \CUser::*GetByID*($this->userId)->fetch();  
  
 if (!$user) {  
 return null;  
 }  
  
 return $user['UF\_JOB'] ?? null;  
 }  
  
 protected function getCategoryIds(int $positionId): array  
 {  
 $categories = [];  
  
 $categoryQuery = \Bitrix\Iblock\Elements\ElementPositionsTable::*getList*([  
 'filter' => [  
 'ACTIVE' => 'Y',  
 '=ID' => $positionId  
 ],  
 'select' => [  
 'CATEGORY\_ID' => 'COMFORT\_CATEGORIES.VALUE'  
 ],  
 ])->fetchAll();  
  
 foreach ($categoryQuery as $category) {  
 if (!in\_array($category['CATEGORY\_ID'], $categories)) {  
 $categories[] = (int)$category['CATEGORY\_ID'];  
 }  
 }  
  
 return $categories;  
 }  
  
 protected function getCarsByComfortIds(array $comfortIds): array  
 {  
 return \Bitrix\Iblock\Elements\ElementCarsTable::*getList*([  
 'filter' => [  
 'ACTIVE' => 'Y',  
 '=CATEGORY\_ID' => $comfortIds  
 ],  
 'select' => [  
 'ID',  
 'NAME',  
 'DRIVE\_ID' => 'DRIVER.VALUE',  
 'CATEGORY\_ID' => 'COMFORT\_CATEGORY.VALUE'  
 ],  
 ])->fetchAll();  
 }  
  
 protected function getBusyCarsAndDrivers(DateTime $start, DateTime $end): array  
 {  
 $hlblId = $this->getHLBLIdByCode();  
 $hlblock = \Bitrix\Highloadblock\HighloadBlockTable::getById($hlblId)->fetch();  
 $entity = \Bitrix\Highloadblock\HighloadBlockTable::*compileEntity*($hlblock);  
 $entityDataClass = $entity->getDataClass();  
  
 return $entityDataClass::*getList*([  
 'select' => ['UF\_CAR\_ID', 'UF\_DRIVER\_ID'],  
 'filter' => [  
 [  
 'LOGIC' => 'AND',  
 ['<UF\_START\_TIME' => $end],  
 ['>UF\_END\_TIME' => $start],  
 ],  
 ],  
 ])->fetchAll();  
 }  
  
 protected function getHLBLIdByCode(): string  
 {  
 $result = \Bitrix\Highloadblock\HighloadBlockTable::getList([  
 'filter' => [  
 '=NAME' => $this::*HLBL\_CODE* ],  
 'select' => [  
 'ID'  
 ]  
 ]);  
  
 return $result->fetch()['ID'];  
 }  
  
 protected function filterAvailableCars(array $cars, array $busyCars): array  
 {  
 if (empty($busyCars)) {  
 return $cars;  
 }  
  
 $busyCarIds = array\_column($busyCars, 'UF\_CAR\_ID');  
  
 return array\_values(array\_filter($cars, function ($car) use ($busyCarIds) {  
 return !in\_array($car['ID'], $busyCarIds);  
 }));  
 }  
  
 private function initModules(): void  
 {  
 if (!Loader::*includeModule*('iblock')) {  
 throw new SystemException('Модуль iblock не подключён');  
 }  
 if (!Loader::*includeModule*('highloadblock')) {  
 throw new SystemException('Модуль highloadblock не подключён');  
 }  
 }  
  
 private function getUser(): void  
 {  
 $this->userId = \Bitrix\Main\Engine\CurrentUser::*get*()->getId();  
 if (!$this->userId) {  
 throw new SystemException('Пользователь не авторизован');  
 }  
 }  
  
 private function parseTimes(): void  
 {  
 $startTime = trim($\_GET['start'] ?? '');  
 $endTime = trim($\_GET['end'] ?? '');  
  
 $currentDate = (new DateTime())->format("Y-m-d");  
  
 if (!$startTime || !$endTime) {  
 throw new ArgumentException('Не заданы параметры start или end в GET запросе');  
 }  
  
 $this->start = new DateTime($currentDate . ' ' . $startTime, 'Y-m-d H:i:s');  
 $this->end = new DateTime($currentDate . ' ' . $endTime, 'Y-m-d H:i:s');  
  
 if ($this->end->getTimestamp() <= $this->start->getTimestamp()) {  
 throw new ArgumentException('Параметр end должен быть позже start.');  
 }  
 }  
  
 private function viewResult(mixed $text): void  
 {  
 echo "<pre>";  
 print\_r($text);  
 echo "</pre>";  
 }  
}