Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Дисциплина: Интернет-технологии и веб-программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

СЕРВИС ДЛЯ ОРГАНИЗАЦИИ СОВМЕСТНЫХ ПОЕЗДОК (CARPOOLING).

БГУИР КП 1-53 01 02 219 ПЗ

Студент М.Н. Рубаха
Руководитель Н.В. Хаджинова

Минск 2025

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет информационных технологий и управления

УТВЕРЖДАЮ
Заведующий кафедрой

_____
(подпись)
_____2025г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Рубаха Максиму Николаевичу*

1. Тема проекта Сервис для организации совместных поездок (carpooling)
2. Срок сдачи студентом законченного проекта *24 декабря 2025 г.*
3. Исходные данные к проекту *Поставлена задача спроектировать и разработать сервис, который предназначен для организации совместных поездок пользователей с уменьшенной итоговой ценой поездки.Сервис должен предоставить пользователям удобное приложение, где они смогут зарегистрироваться, оформить поездку в заданном направлении, найти попутчиков и оплатить итоговую поездку.*
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке):

Введение
1. Анализ предметной области
2. Проектирование системы
3. Программная реализация системы
Заключение
Список использованных источников
Приложение А (обязательное)
Ведомость курсового проекта

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

_____

6. Консультант по проекту (с обозначением разделов проекта *Н. В. Хаджинова*

7. Дата выдачи задания *13 сентября 2025 г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

раздел 1 к 01.10 – 10 %;

раздел 2 к 10.11 – 25 %;

раздел 3 к 01.12 – 30 %;

раздел 4 к 01.12 – 15 %

оформление пояснительной записки и графического материала к 10.12 – 20 %

*Защита курсового проекта с **10.12** по **17.05.2025**г.*

РУКОВОДИТЕЛЬ *Н. В. Хаджинова*

(подпись)

Задание принял к исполнению *М. Н. Рубаха*

(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Карпулинг (совместное использование автомобиля) — это эффективный инструмент современной транспортной логистики, позволяющий нескольким людям использовать один автомобиль для совершения поездки по схожему маршруту. В условиях стремительной урбанизации, постоянного роста количества личного транспорта и перегруженности дорожной инфраструктуры, проблема оптимизации пассажирских перевозок становится как никогда актуальной. Чтобы обеспечить снижение транспортных расходов, уменьшить экологическую нагрузку и разгрузить трафик, необходимы современные автоматизированные системы поиска попутчиков.

Хорошо спроектированный веб-сервис для организации совместных поездок способен автоматизировать процесс взаимодействия между водителем и пассажирами: от геопозиционирования и построения оптимального маршрута до автоматического расчета стоимости и распределения расходов. Это не только упрощает планирование поездок для частных лиц, но и повышает эффективность использования транспортных средств, что особенно важно в условиях растущих цен на топливо и высоких требований к мобильности населения.

Системы организации поездок должны включать в себя широкий спектр функциональных возможностей — от визуализации маршрута на интерактивной карте до механизмов ролевого доступа (пассажир, водитель, администратор). Также система должна поддерживать обработку транзакций и статусов в реальном времени, позволяя пользователям оперативно находить попутчиков, находящихся в непосредственной близости, и координировать свои действия.

Современное веб-приложение для карпулинга должно обеспечивать комплексное управление жизненным циклом поездки, включая создание заявки, поиск совпадений по координатам, оплату и завершение маршрута. Интерфейс должен быть интуитивно понятным, адаптивным для мобильных устройств и обеспечивать мгновенную обратную связь без полной перезагрузки страницы, что достигается за счет использования асинхронных запросов (*AJAX*) и динамического обновления контента.

В данной курсовой работе рассматривается проектирование и реализация полнофункционального клиент-серверного веб-приложения для организации совместных поездок, включающего реляционную базу данных, серверную часть на языке *PHP* и интерактивную клиентскую часть с интеграцией картографических сервисов..

1 АНАЛИЗ ИССЛЕДУЕМОЙ ОБЛАСТИ

1.1 Описание предметной области

Концепция совместного использования ресурсов, и в частности транспортных средств, имеет глубокие исторические корни. Изначально этот процесс носил стихийный характер (автостоп), однако с развитием индустриализации и ростом городов возникла потребность в более организованных формах перемещения. Существенный импульс развитию карпулинга дали топливные кризисы второй половины *XX* века, когда экономия ресурсов стала приоритетной задачей для населения и государств. Именно тогда в США и Европе начали появляться первые организованные клубы попутчиков, координируемые через доски объявлений и телефонные службы.

Настоящая революция в данной сфере произошла с повсеместным внедрением интернета и технологий *GPS*-навигации. Переход от бумажных списков и диспетчерских служб к цифровым платформам позволил автоматизировать процесс поиска попутчиков, сделав его мгновенным и географически точным. Современная парадигма *Sharing Economy* (экономика совместного потребления) диктует новые правила: владение автомобилем перестает быть обязательным условием мобильности, уступая место временному доступу к транспортному средству.

Ключевой задачей современных информационных систем в этой области является решение алгоритмической проблемы сопоставления маршрутов. Система должна не просто соединить двух людей, но и определить, насколько их пути пересекаются географически и хронологически. Это требует обработки геопространственных данных в реальном времени, расчета расстояний (например, по формуле гаверсинуса) и оптимизации логистики, чтобы минимизировать отклонения от оптимального маршрута водителя.

Автоматизированные системы карпулинга сегодня решают комплекс социальных и экономических задач: они способствуют снижению трафика в мегаполисах, уменьшают углеродный след за счет сокращения количества автомобилей на дорогах и позволяют пользователям существенно экономить на ежедневных поездках, разделяя расходы на топливо и амортизацию.

1.2 Обзор аналогов

Для проектирования конкурентоспособного веб-приложения был проведен анализ существующих на рынке решений. Изучены популярные платформы, предоставляющие услуги такси и поиска попутчиков, с целью выявления ключевых функциональных паттернов и пользовательских сценариев.

Одним из наиболее известных сервисов для междугородних поездок является *BlaBlaCar*. Данная платформа фокусируется на планировании поездок заранее (за несколько часов или дней)

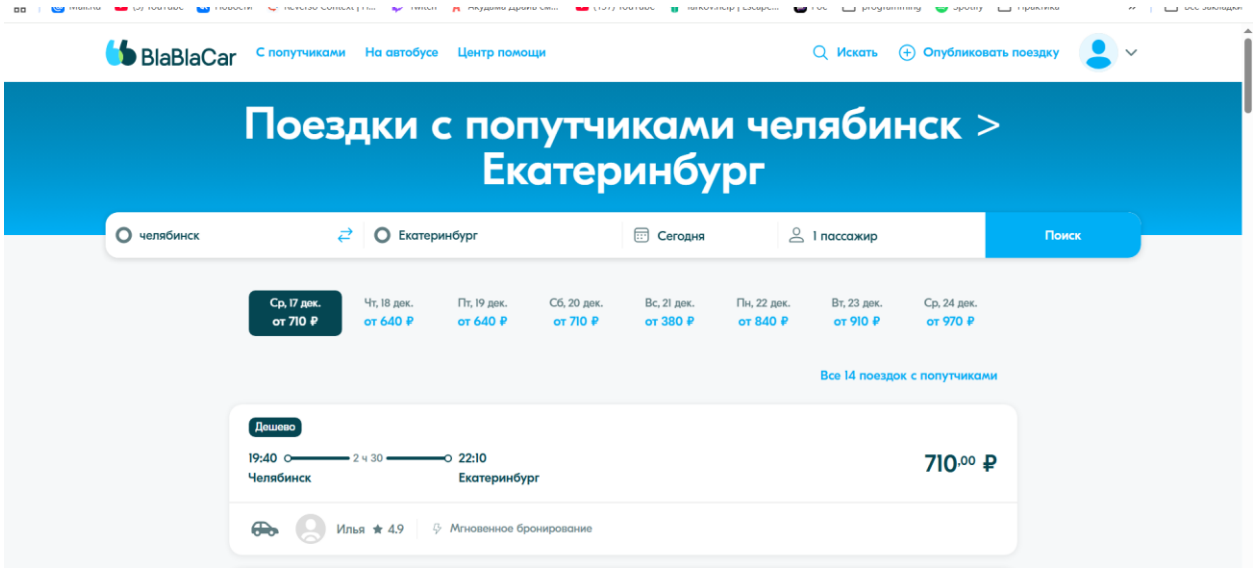

Рисунок 1.1 – Интерфейс поиска поездки на платформе *BlaBlaCar*

Пользователь указывает точки отправления и назначения, а система предлагает список водителей, едущих в том же направлении. Основной недостаток для городских условий — отсутствие динамики. Сервис плохо приспособлен для спонтанных поездок внутри города на короткие расстояния («здесь и сейчас»).

Вторым значимым аналогом являются агрегаторы такси, такие как *Uber* или *Яндекс Go*. Эти системы обеспечивают мгновенную подачу автомобиля и отслеживание в реальном времени.

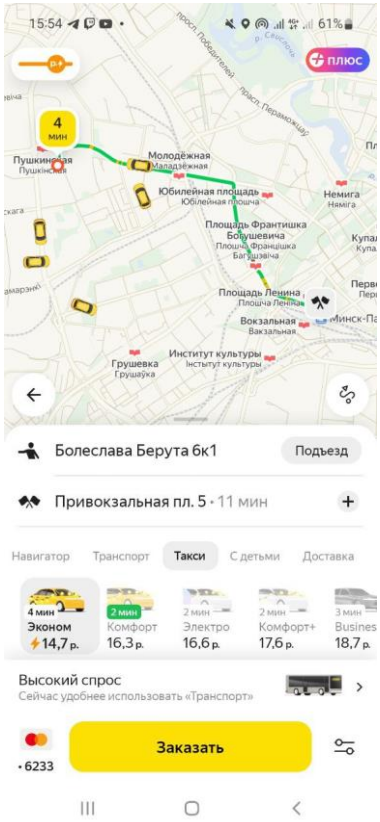

Рисунок 1.2 – Интерфейс заказа автомобиля в *Яндекс Go* (мобильная версия)

Преимуществом таких систем является высокая скорость обработки заказа и удобный картографический интерфейс. Однако классическая модель такси подразумевает оплату полной стоимости поездки одним заказчиком, что может быть экономически невыгодно для ежедневных перемещений. Функции совместных поездок (например, тариф «Вместе») существуют, но часто ограничены сложными алгоритмами ценообразования и закрытостью платформы.

Таким образом, разрабатываемая система должна быть балансом между классическим карпулингом и такси: обеспечить возможность динамического поиска попутчиков внутри города или пригорода с прозрачным разделением стоимости и визуализацией маршрута.

1.3 Постановка задачи

Целью данного курсового проекта является разработка клиент-серверного веб-приложения для организации совместных поездок, обеспечивающего автоматизированный поиск попутчиков на основе геолокации. Система должна предоставлять инструменты для создания заявок, мониторинга статуса поездки и проведения виртуальных взаиморасчетов.

Приложение должно обеспечивать реализацию трех уровней доступа: «Пассажир» (создание заявок, оплата), «Водитель» (просмотр и принятие заказов, навигация по точкам), «Администратор» (управление учетными записями водителей).

Приложение будет представлять из себя веб-сайт, который должен быть адаптирован под различные устройства, обеспечивать стабильную работу под нагрузкой 24 часа в сутки.

Серверная часть приложение должна работать на системе с операционной системой *Windows* или *Linux*. Система должна иметь минимум четыре гигабайта оперативной памяти, 50 гигабайтов хранилища для хранения информации. Приложение должно запускать в изолированной среде. Система должна иметь процессор, который должен быстро выполнять различные операции. Процессор должен обладать не менее четырех ядер и частотой не менее двух с половиной гигагерц. Также аппаратная система должна обладать постоянным и стабильным доступом к сети интернет.

Клиентская часть приложения может быть запушена на различных веб-браузерах, таких как *Chrome, Opera, Safari, Mazila FireFox* и других. Аппаратная система может быть любым устройством, обладающим стабильным доступом к сети интернет и поддерживающие данные браузеры.

Приложение должно обладать адаптивным интерфейсом, корректно отображающимся как на десктопных мониторах, так и на экранах мобильных устройств. Клиентская часть должна обеспечивать асинхронное взаимодействие с сервером (без перезагрузки страницы) для оперативного обновления статусов и координат. Безопасность данных должна обеспечиваться валидацией входных параметров и разграничением прав доступа к *API*.

Серверная часть приложения разрабатывается на языке *PHP* и функционирует под управлением веб-сервера *Apache*. В качестве системы управления базами данных выбрана реляционная СУБД *MariaDB* (*MySQL*). Клиентская часть реализуется с использованием *HTML5, CSS3, JavaScript* и библиотеки *Leaflet* для работы с картами. Система должна быть нетребовательна к ресурсам и способна функционировать на стандартном хостинге с поддержкой *PHP* 7.4 и выше.

2 ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ

2.1 Проектирование базы данных

Логическая схема базы данных представляет собой фундаментальный компонент информационной системы, определяющий структуру хранения информации, иерархию сущностей и характер связей между ними. Грамотно спроектированная модель данных обеспечивает не только целостность и непротиворечивость информации, но и высокую скорость выполнения запросов, что критически важно для геосервисов реального времени.

В рамках данного проекта была разработана реляционная схема, состоящая из взаимосвязанных таблиц. Ключевыми сущностями предметной области являются «Пользователи» (участники системы) и «Поездки» (единицы бизнес-логики).

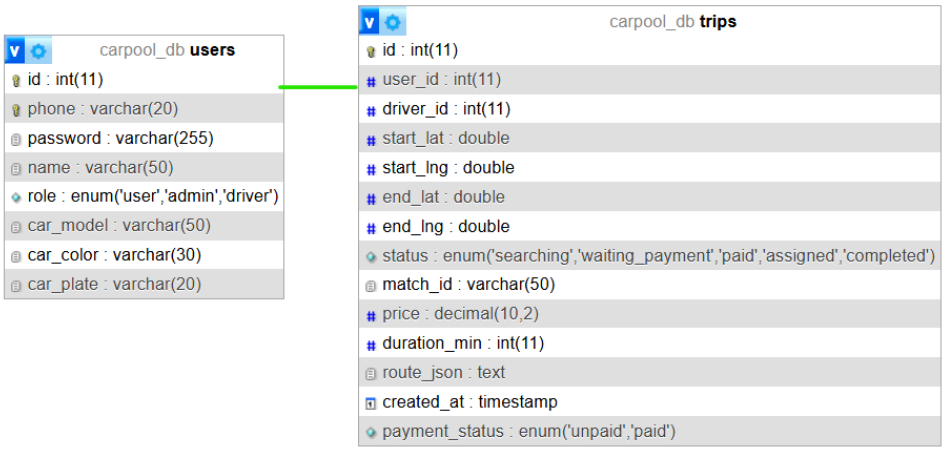

Рисунок 2.1 – Логическая схема базы данных

Структура базы данных спроектирована с учетом требований третьей нормальной формы (3НФ), что подразумевает устранение транзитивных зависимостей и избыточности данных. Атомарность атрибутов позволяет гибко управлять выборками, а использование внешних ключей (*Foreign Keys*) гарантирует ссылочную целостность.

Таблица *users* выполняет роль централизованного реестра учетных записей. В отличие от создания разрозненных таблиц для каждой категории пользователей, было принято решение использовать единую сущность с

атрибутом-дискриминатором *role*. Для пользователей с ролью «водитель» в этой же таблице предусмотрены специфические поля: *car_model* (марка авто), *car_color* (цвет) и *car_plate* (регистрационный номер). Для обычных пассажиров эти поля могут принимать значение *NULL*.

Таблица trips является динамическим реестром заявок и содержит всю информацию о жизненном цикле поездки. Она связана с таблицей пользователей двумя внешними ключами: *user_id* (кто создал заявку) и *driver_id* (кто исполняет заказ).

Координаты точек отправления и назначения хранятся в формате чисел с плавающей точкой (*start_lat*, *start_lng*, *end_lat*, *end_lng*), что необходимо для математического вычисления расстояний.

Поле *status* реализует машину состояний заявки: от первичного поиска (*searching*) и ожидания оплаты (*waiting_payment*) до назначения машины (*assigned*) и завершения (*completed*).

Атрибут *match_id* служит идентификатором группы. Если несколько пользователей объединены в одну поездку, им присваивается одинаковый строковый идентификатор, что позволяет системе обрабатывать их как единый логистический объект.

Поле *payment_status* отслеживает факт оплаты конкретным пассажиром, что позволяет реализовать логику раздельной оплаты поездки.

Поле *route_json* хранит сериализованный массив точек маршрута, что избавляет от необходимости пересчитывать путь при каждом обновлении страницы.

Такая архитектура базы данных обеспечивает необходимую гибкость для реализации алгоритмов поиска попутчиков и позволяет масштабировать систему без существенных структурных изменений.

3 РАЗРАБОТКА ВЕБ ПРИЛОЖЕНИЯ

3.1 Выбор средств реализации

Для реализации серверной логики был выбран язык программирования *PHP*. Это интерпретируемый скриптовый язык общего назначения, который де-факто является стандартом для разработки динамических веб-сайтов. Ключевыми аргументами в пользу *PHP* стали его широкая распространенность, нативная поддержка работы с базами данных и простота развертывания в среде *XAMPP*. Скрипты выполняются на стороне сервера, генерируя *JSON*-ответы для клиентской части, что позволяет скрыть бизнес-логику от конечного пользователя.

В качестве системы управления базами данных используется *MariaDB* (*MySQL*). Это реляционная СУБД, обеспечивающая высокую скорость выполнения операций чтения и записи, что критически важно для сервиса, где множество пользователей одновременно обновляют свои координаты и статусы.

Для реализации картографического функционала была выбрана библиотека с открытым исходным кодом *Leaflet.js*. В отличие от проприетарных решений (например, *Google Maps API*), *Leaflet* не требует платных ключей доступа, обладает легковесным ядром и поддерживает подключение различных поставщиков тайлов (*OpenStreetMap*). Для построения маршрутов используется плагин *Leaflet Routing Machine*, позволяющий визуализировать путь между точками.

Клиентский интерфейс построен на базе фреймворка *Bootstrap*5, который обеспечивает адаптивность верстки и кроссбраузерную совместимость. Для организации асинхронного взаимодействия (*AJAX*) используется библиотека *jQuery*, позволяющая отправлять запросы к *API* без перезагрузки страницы, создавая ощущение нативного приложения (*SPA*).

3.2 Программная реализация приложения

Для обеспечения взаимодействия с базой данных был реализован отдельный модуль конфигурации *db.php*. Это позволяет централизовать настройки подключения и избежать дублирования кода. В скрипте используется объектно-ориентированный интерфейс *mysqli*, а также принудительно устанавливается кодировка *UTF-8* для корректного

отображения кириллических символов (имен пользователей и названий остановок).

```
<?php
$host = 'localhost';
$db   = 'carpool_db';
$user = 'root';
$pass = '';

$conn = new mysqli($host, $user, $pass, $db);
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}
// Установка кодировки для поддержки кириллицы
$conn->set_charset("utf8");
session_start();
?>
```

Ядром системы является файл *api.php*, обрабатывающий все входящие запросы. Ниже приведены ключевые фрагменты программного кода с описанием алгоритмов.

Для определения близости пользователей используется математический расчет расстояния между двумя точками на поверхности сферы (земного шара). Реализована функция *getDistance*, использующая формулу гаверсинуса:

```
function getDistance($lat1, $lon1, $lat2, $lon2) {
   $earth_radius = 6371; // Радиус Земли в километрах
   $dLat = deg2rad($lat2 - $lat1);
   $dLon = deg2rad($lon2 - $lon1);

   $a = sin($dLat/2) * sin($dLat/2) +
       cos(deg2rad($lat1)) * cos(deg2rad($lat2)) *
       sin($dLon/2) * sin($dLon/2);

   $c = 2 * atan2(sqrt($a), sqrt(1-$a));
   return $earth_radius * $c;
}
```

Центральным элементом логики является алгоритм создания поездки и поиска попутчиков. При поступлении новой заявки система сохраняет её со статусом *searching*, а затем сканирует базу данных на наличие совместимых маршрутов. Если расстояние между точками старта составляет менее 2 км, а между точками финиша — менее 7 км, алгоритм объединяет пользователей в группу. Фрагмент кода, реализующий создание и поиск (*action = create_trip*):

```
$dist = getDistance($slat, $slng, $elat, $elng);
$price = round(3.00 + ($dist * 0.60), 2);
$dur = max(5, round(($dist / 40) * 60));

$stmt = $conn->prepare("INSERT INTO trips (user_id, start_lat, start_lng, ... status) VALUES (?, ?, ?, ..., 'searching')");
$stmt->execute();
$trip_id = $conn->insert_id;

$sql = "SELECT * FROM trips WHERE status='searching' AND id != $trip_id";
$res = $conn->query($sql);

while($row = $res->fetch_assoc()) {
  $dist_s = getDistance($slat, $slng, $row['start_lat'], $row['start_lng']);
  $dist_e = getDistance($elat, $elng, $row['end_lat'], $row['end_lng']);

  if ($dist_s <= 2 && $dist_e <= 7) {
    $match_id = uniqid('m');
    $route = [
      ['lat'=>$slat, 'lng'=>$slng, 'name'=>'Точка сбора'],
      ['lat'=>$row['end_lat'], 'lng'=>$row['end_lng'], 'name'=>'Остановка 1'],
      ['lat'=>$elat, 'lng'=>$elng, 'name'=>'Остановка 2']
    ];
    $route_json = json_encode($route, JSON_UNESCAPED_UNICODE);

    $conn->query("UPDATE trips SET status='waiting_payment', match_id='$match_id', route_json='$route_json' WHERE id=$trip_id");
    $conn->query("UPDATE trips SET status='waiting_payment', match_id='$match_id', route_json='$route_json' WHERE id={$row['id']}");
    break;
  }
}
```

Для обеспечения интерактивности используется механизм *Long Polling* (длинный опрос). Клиентская часть каждые несколько секунд обращается к методу *check_status*, который возвращает актуальное состояние заказа. Это позволяет водителю мгновенно узнать о новом заказе, а пассажиру — о прибытии машины. Реализация проверки статуса для водителя с агрегацией данных о пассажирах:

```
<?php
if ($action == 'check_status') {
  $sql = "SELECT ... FROM trips WHERE driver_id=$uid AND status != 'completed' LIMIT 1";
```

```
    $res = $conn->query($sql);
    if ($trip = $res->fetch_assoc()) {
        $passengers = [];
        $total_price = 0;

        if ($trip['match_id']) {
            $mid = $trip['match_id'];
            $users_res = $conn->query("SELECT u.name, u.phone, t.price FROM trips t JOIN
users u ON t.user_id = u.id WHERE t.match_id='$mid'");
            while($u = $users_res->fetch_assoc()) {
                $total_price += $u['price'];
                $passengers[] = $u;
            }
        }
        $trip['passengers'] = $passengers;
        $trip['total_price'] = $total_price;
        echo json_encode(['status'=>'success', 'data'=>$trip]);
    } else {
        echo json_encode(['status'=>'no_trip']);
    }
}?>
```

Важной частью клиентской логики является визуализация маршрута. В файле *dashboard.php* реализована функция *drawRoute*, которая принимает *JSON*-объект с координатами точек, парсит его и использует библиотеку *Leaflet Routing Machine* для построения векторной линии пути на карте. Данная функция также управляет маркерами, очищая карту от предыдущих меток перед отрисовкой новых, что предотвращает утечки памяти и визуальный мусор. Фрагмент *JavaScrip*t-кода, отвечающий за отрисовку маршрута:

```
function drawRoute(json) {
    if(!json) return;
    let pts = JSON.parse(json);
    )
    if(!routingControl) {
        let wps = pts.map(p => L.latLng(p.lat, p.lng));
        if(startMarker) map.removeLayer(startMarker);
        routingControl = L.Routing.control({
            waypoints: wps,
            show: false,
            draggableWaypoints: false,
            addWaypoints: false,
            lineOptions: {
```

```
            styles: [{color: '#10b981', opacity: 0.8, weight: 6}]
        },
        createMarker: function() { return null; }
    }).addTo(map);
    map.fitBounds(L.latLngBounds(wps).pad(0.2));
  }
}
```

Визуальное оформление приложения вынесено в отдельный файл стилей *style.css*. Использован подход *Mobile First*, обеспечивающий корректное отображение на смартфонах. Для создания современного интерфейса применены переменные *CSS* (*:root*) и эффект *Glassmorphism* (полупрозрачный фон с размытием) для плавающих панелей управления поверх карты. Фрагмент таблицы стилей:

```
:root {
  --primary: #10b981;
  --dark: #1f2937;
  --glass: rgba(255, 255, 255, 0.95);
}
#map {
  position: absolute;
  top: 0; left: 0; right: 0; bottom: 0;
  z-index: 1;
}
.status-card {
  position: absolute; top: 15px;
  background: var(--glass);
  backdrop-filter: blur(10px);
  border-radius: 20px;
  box-shadow: 0 10px 30px rgba(0,0,0,0.15);
  z-index: 1000;
}
```

Такая реализация обеспечивает четкое разделение ответственности: сервер занимается вычислениями и хранением данных, а клиентский браузер отвечает за рендеринг карты и интерфейса.

4 ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

Взаимодействие с программным комплексом начинается со страницы авторизации. При первом запуске веб-приложения пользователю предлагается либо войти в существующий аккаунт, либо зарегистрироваться.

На этапе регистрации система запрашивает имя, номер телефона и пароль. Для удобства ввода реализована маска номера телефона, автоматически подставляющая код страны «+375». Это минимизирует вероятность ошибки при вводе контактных данных.

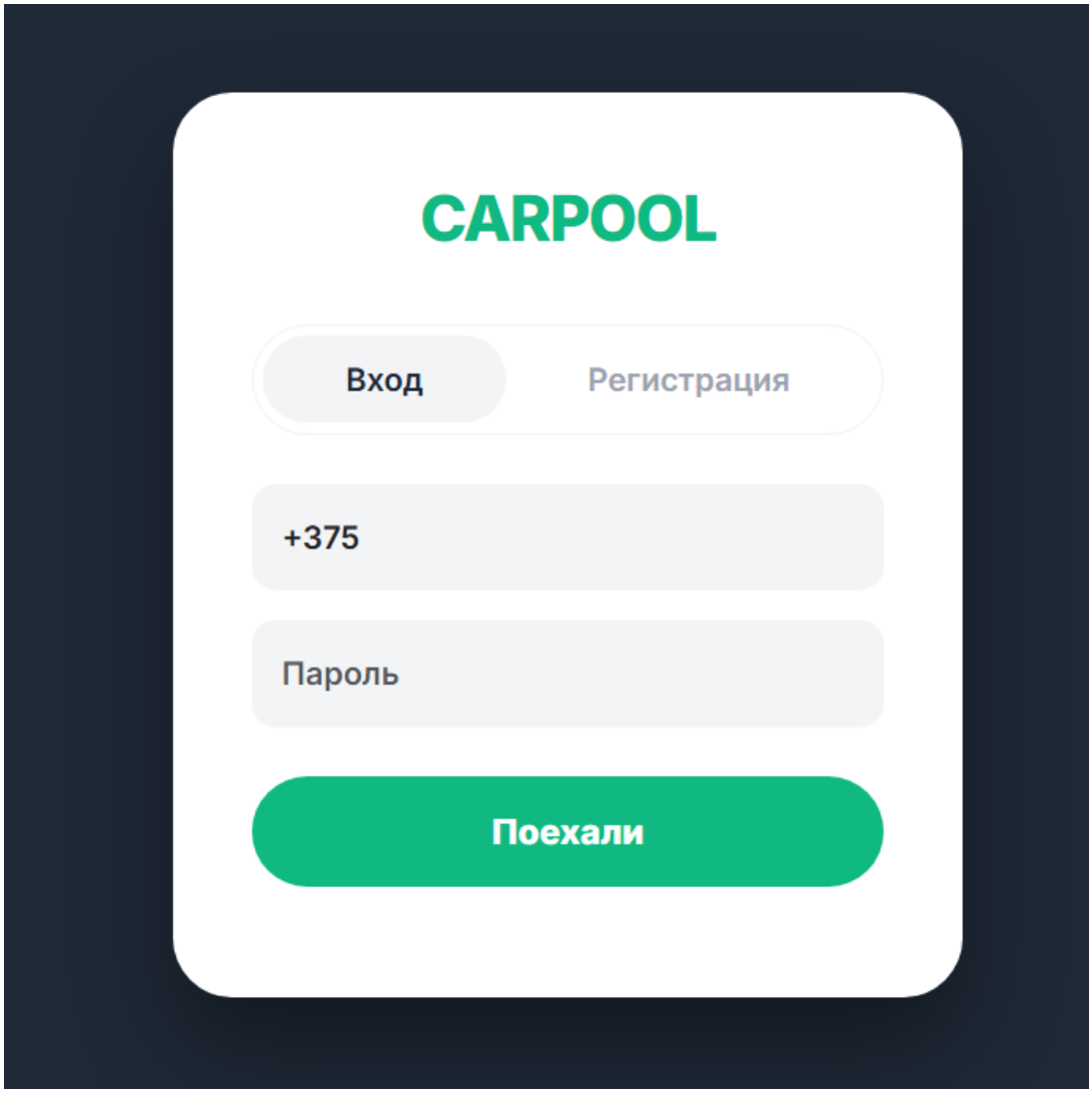

Рисунок 4.1 – Форма авторизации и регистрации пользователей

После успешной аутентификации, в зависимости от назначенной роли, происходит перенаправление на соответствующий интерфейс: панель администратора, терминал водителя или карту пассажира.

Опишем сценарий работы пассажира. Пользователь попадает на главный экран, большую часть которого занимает интерактивная карта. В нижней части расположена панель управления. Пользователь нажимает кнопки «Откуда» и «Куда», последовательно указывая точки на карте. Система отображает маркеры старта (зеленый) и финиша (красный). После нажатия кнопки «Заказать» заявка отправляется на сервер, и активируется режим поиска попутчиков. Если система находит подходящий маршрут другого пользователя, статус меняется, и появляется кнопка оплаты. Если попутчик не найден, пользователю предлагается опция «Поехать одному» с пересчетом стоимости.

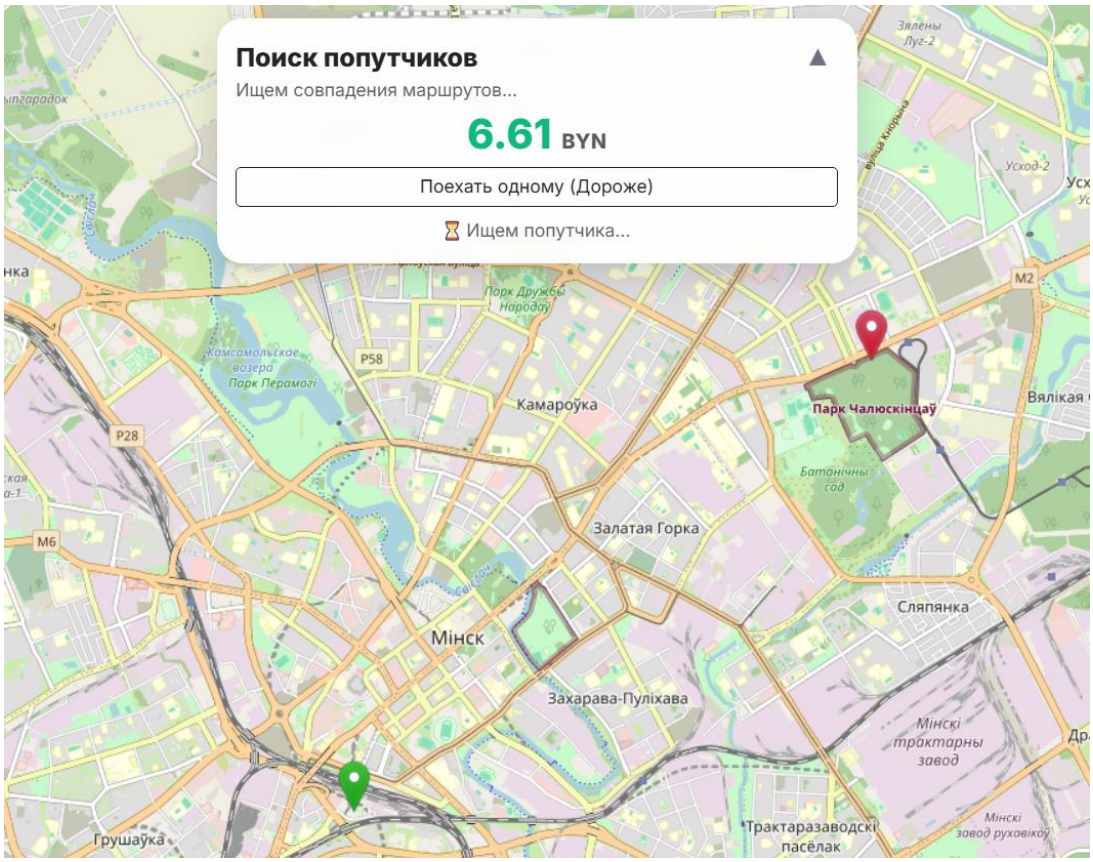

Рисунок 4.2 – Интерфейс пассажира

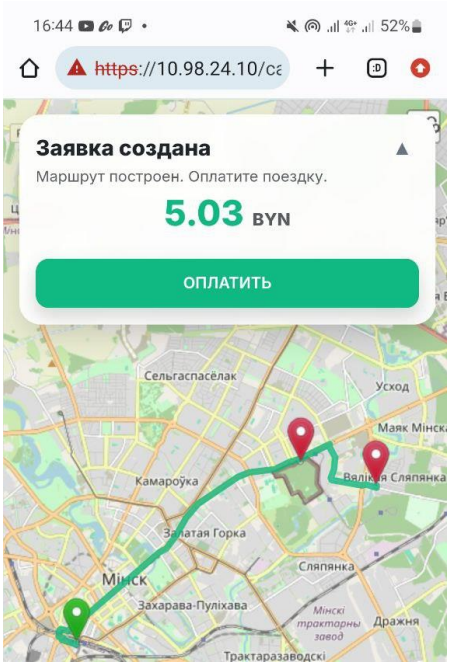

Рисунок 4.3 – Интерфейс пассажира при совместной поездке

Перейдём к сценарию водителя. интерфейс водителя оптимизирован для работы «на ходу». На главном экране отображается список доступных (уже оплаченных) заказов с указанием стоимости и примерного времени в пути.При клике на карточку заказа на карте отрисовывается предварительный маршрут (синяя линия), позволяющий оценить логистику до принятия решения. После нажатия кнопки «Принять», открывается панель активной поездки. Водитель видит список всех пассажиров (имена, телефоны) и детальный список остановок. По прибытии в конечную точку водитель нажимает «Завершить заказ», что удаляет поездку из активного списка.

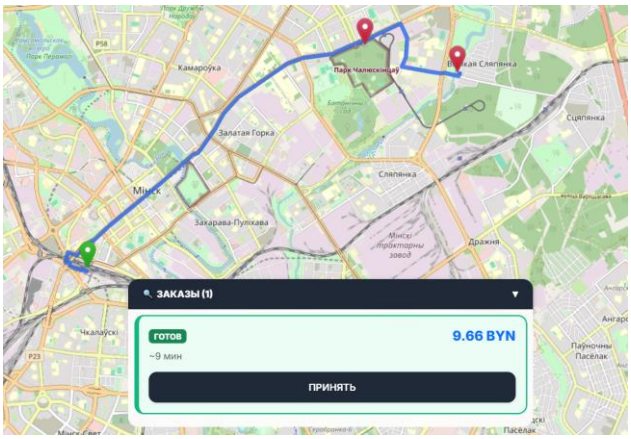

Рисунок 4.4 – Терминал водителя, просмотр маршрута

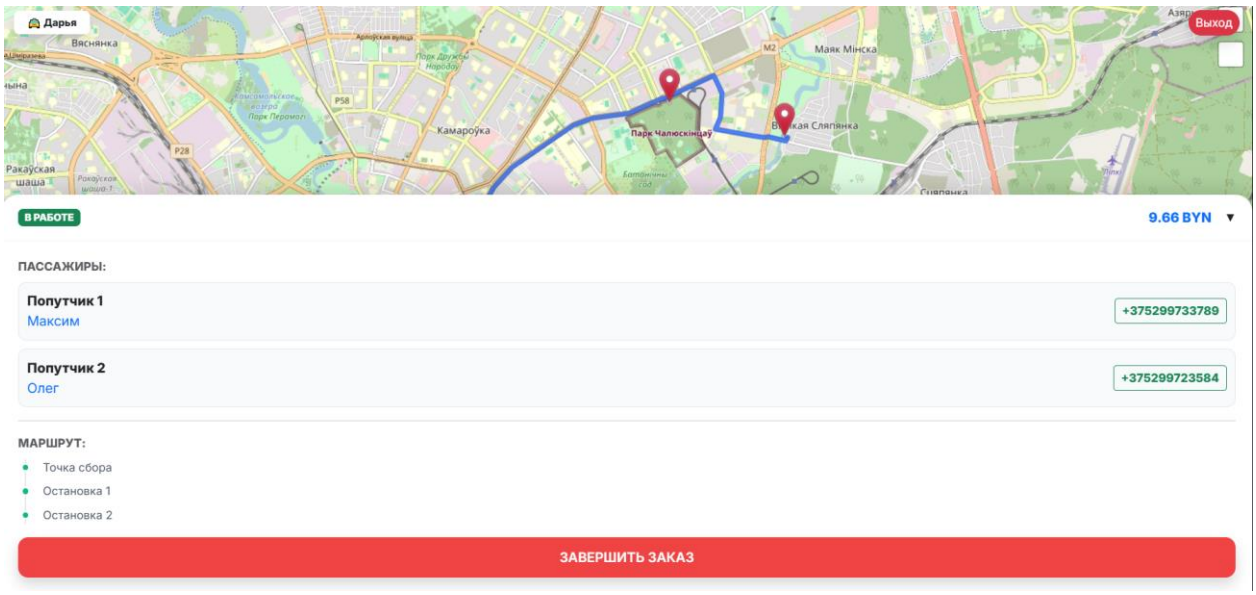

Рисунок 4.5 – Терминал водителя, активный заказ

Для управления персоналом реализована административная панель. Администратор имеет возможность просматривать список всех водителей, редактировать их данные (включая информацию об автомобиле: марка, цвет, госномер) и удалять учетные записи.

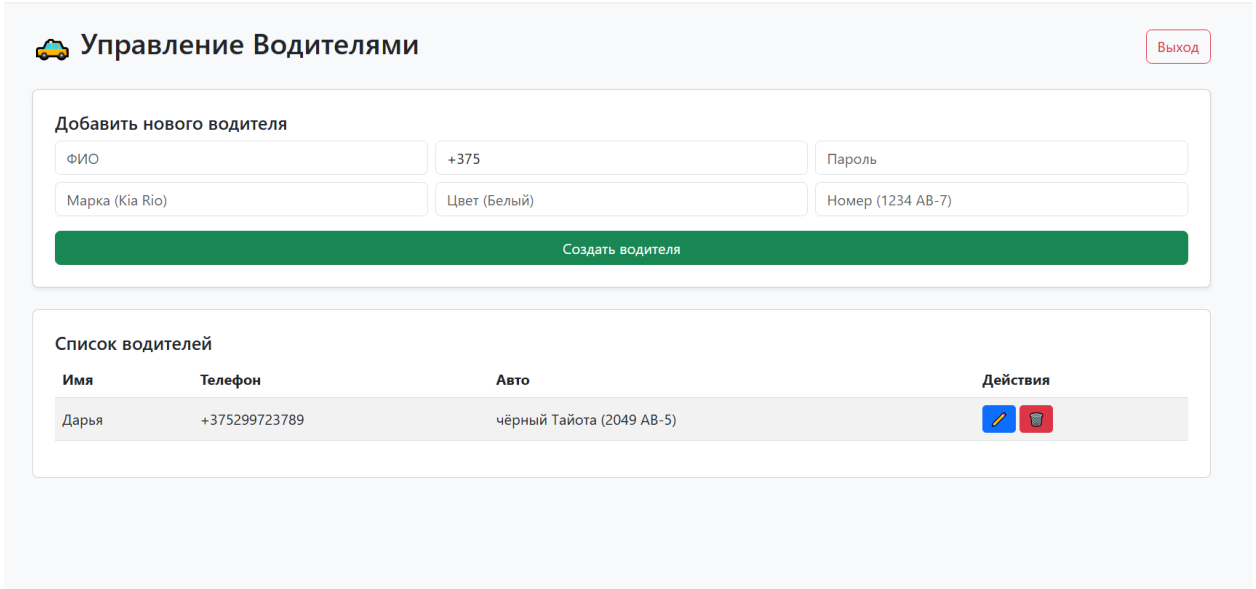

Рисунок 4.6 – Панель администратора водителей

Администрирование всего сервиса также можно производить и через панель *phpMyAdmin*, однако это не является реализацией интерфейса администратора, поэтому не рассматривается в руководстве пользователя.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была успешно решена задача проектирования и разработки автоматизированной информационной системы для организации совместных поездок (карпулинга). В результате проделанной работы разработана нормализованная схема базы данных, поддерживающая ролевую модель доступа и сложные связи между сущностями (пользователи, поездки, группы попутчиков). Создан эффективный механизм подбора попутчиков на основе геопространственных данных, позволяющий объединять маршруты в радиусе допустимого отклонения. Создано адаптивное веб-приложение с современным дизайном (*Glassmorphism*), обеспечивающее удобное взаимодействие с картой и элементами управления как на персональных компьютерах, так и на мобильных устройствах. Реализованы отдельные рабочие места для пассажира (заказ, оплата), водителя (навигация, управление заказами) и администратора.

В качестве технологического стека использовались язык *PHP*, СУБД *MariaDB*, библиотеки *Leaflet* и *jQuery*. Выбранные средства разработки позволили создать легковесное, быстродействующее приложение, не требующее установки дополнительного ПО на устройство клиента.

Практическая значимость разработки заключается в возможности использования данного сервиса для оптимизации транспортных потоков внутри города, снижения затрат на передвижение для граждан и уменьшения экологической нагрузки на окружающую среду. В перспективе система может быть масштабирована: возможно внедрение системы рейтингов, онлайн-чата между участниками поездки и интеграция с платежными шлюзами для реальных банковских транзакций.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Котеров, Д. В. *PHP* 7. В подлиннике / Д. В. Котеров, И. В. Симдянов. – Санкт-Петербург: БХВ-Петербург, 2019. – 1088 с.

[2] Флэнаган, Д. *JavaScript*. Подробное руководство / Д. Флэнаган. – 7-е изд. – Санкт-Петербург: Символ-Плюс, 2021. – 704 с.

[3] Дюбуа, П. *MySQL*. Сборник рецептов / П. Дюбуа. – Москва: Символ-Плюс, 2020. – 1056 с.

[4] Никсон, Р. Создаем динамические веб-сайты с помощью *PHP*, *MySQL, JavaScript, CSS* и *HTML5* / Р. Никсон. – 5-е изд. – Санкт-Петербург: Питер, 2019. – 816 с.

[5] Библиотека *jQuery API Documentation* [Электронный ресурс]. – Режим доступа: *https://api.jquery.com/*

[6] Фреймворк *Bootstrap*5 [Электронный ресурс]. – Режим доступа: *https://getbootstrap.com/docs/5.3/getting-started/introduction/*

[7] Документация языка *PHP* [Электронный ресурс]. – Режим доступа: *https://www.php.net/manual/ru/*

[8] Официальная документация СУБД *MariaDB* [Электронный ресурс]. – Режим доступа: *https://mariadb.com/kb/en/documentation/*

[9] Секреты *HTML CSS* [Электронный ресурс]. − Режим доступа: *https://skysmart.ru/articles/programming/secrety-HTML-CSS*

[10] Библиотека интерактивных карт *Leaflet* [Электронный ресурс]. – Режим доступа: *https://leafletjs.com/reference.html*

[11] СТП 01–2024. Стандарт предприятия. Дипломные проекты (работы). Общие требования. - Минск: БГУИР, 2024. – 178 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Программный код функции приложения

```
<?php
require 'db.php';
header('Content-Type: application/json');

$action = $_GET['action'] ?? '';

function createRouteJson($points) { return json_encode($points, JSON_UNESCAPED_UNICODE); }

function getDistance($lat1, $lon1, $lat2, $lon2) {
   $earth_radius = 6371;
   $dLat = deg2rad($lat2 - $lat1);
   $dLon = deg2rad($lon2 - $lon1);
   $a = sin($dLat/2) * sin($dLat/2) + cos(deg2rad($lat1)) * cos(deg2rad($lat2)) * sin($dLon/2) * sin($dLon/2);
   $c = 2 * atan2(sqrt($a), sqrt(1-$a));
   return $earth_radius * $c;
}


if ($action == 'auth') {
   $type = $_POST['type']; $phone = trim($_POST['phone']); $password = $_POST['password'];
   if ($type == 'register') {
      $name = trim($_POST['name']);
      $check = $conn->query("SELECT id FROM users WHERE phone='$phone'");
      if ($check->num_rows > 0) { echo json_encode(['status'=>'error', 'message'=>'Номер занят']); exit; }
      $stmt = $conn->prepare("INSERT INTO users (phone, password, name, role) VALUES (?, ?, ?, 'user')");
      $stmt->bind_param("sss", $phone, $password, $name);
      if ($stmt->execute()) {
         $_SESSION['user_id'] = $conn->insert_id; $_SESSION['role'] = 'user'; $_SESSION['name'] = $name;
         echo json_encode(['status'=>'success', 'role'=>'user']);
      }
   } else {
      $stmt = $conn->prepare("SELECT * FROM users WHERE phone=?");
      $stmt->bind_param("s", $phone);
```

```
        $stmt->execute();
        $res = $stmt->get_result();
        if ($res->num_rows > 0) {
            $row = $res->fetch_assoc();
            if ($row['password'] == $password) {
                $_SESSION['user_id'] = $row['id']; $_SESSION['role'] = $row['role']; $_SESSION['name'] = $row['name'];
                echo json_encode(['status'=>'success', 'role'=>$row['role']]);
            } else echo json_encode(['status'=>'error', 'message'=>'Неверный пароль']);
        } else echo json_encode(['status'=>'error', 'message'=>'Пользователь не найден']);
    }
}


if ($action == 'add_driver') {
    if ($_SESSION['role'] != 'admin') die();
    $stmt = $conn->prepare("INSERT INTO users (phone, password, name, role, car_model, car_color, car_plate) VALUES (?, ?, ?, 'driver', ?, ?, ?)");
    $stmt->bind_param("ssssss", $_POST['phone'], $_POST['password'], $_POST['name'], $_POST['car_model'], $_POST['car_color'], $_POST['car_plate']);
    if($stmt->execute()) echo json_encode(['status'=>'success']); else echo json_encode(['status'=>'error']);
}


if ($action == 'create_trip') {
    $uid = $_SESSION['user_id'];
    $slat = $_POST['start_lat']; $slng = $_POST['start_lng']; $elat = $_POST['end_lat']; $elng = $_POST['end_lng'];
    $dist = getDistance($slat, $slng, $elat, $elng);
    $price = round(3.00 + ($dist * 0.60), 2);
    $dur = max(5, round(($dist / 40) * 60));

    $stmt = $conn->prepare("INSERT INTO trips (user_id, start_lat, start_lng, end_lat, end_lng, price, duration_min, status) VALUES (?, ?, ?, ?, ?, ?, ?, 'searching')");
    $stmt->bind_param("iddddddi", $uid, $slat, $slng, $elat, $elng, $price, $dur);
    $stmt->execute();
    $trip_id = $conn->insert_id;

    $sql = "SELECT * FROM trips WHERE status='searching' AND id != $trip_id";
    $res = $conn->query($sql);
    while($row = $res->fetch_assoc()) {
        $dist_s = getDistance($slat, $slng, $row['start_lat'], $row['start_lng']);
        $dist_e = getDistance($elat, $elng, $row['end_lat'], $row['end_lng']);
```

```
        if ($dist_s <= 2 && $dist_e <= 7) {
            $match_id = uniqid('m');
            $route = [
                ['lat'=>$slat, 'lng'=>$slng, 'name'=>'Точка сбора'],
                ['lat'=>$row['end_lat'], 'lng'=>$row['end_lng'], 'name'=>'Остановка 1'],
                ['lat'=>$elat, 'lng'=>$elng, 'name'=>'Остановка 2']
            ];
            $route_json = createRouteJson($route);
            $new_price = round($price * 0.7, 2); $new_price2 = round($row['price'] * 0.7, 2);

            $conn->query("UPDATE trips SET status='waiting_payment', match_id='$match_id',
price=$new_price, route_json='$route_json' WHERE id=$trip_id");
            $conn->query("UPDATE trips SET status='waiting_payment', match_id='$match_id',
price=$new_price2, route_json='$route_json' WHERE id={$row['id']}");
            break;
        }
    }
    echo json_encode(['status' => 'success']);
}

if ($action == 'go_alone') {
    $uid = $_SESSION['user_id'];
    $res = $conn->query("SELECT * FROM trips WHERE user_id=$uid AND status='searching'
ORDER BY id DESC LIMIT 1");
    if($row = $res->fetch_assoc()) {
        $route = [['lat'=>$row['start_lat'], 'lng'=>$row['start_lng'], 'name'=>'Старт'],
['lat'=>$row['end_lat'], 'lng'=>$row['end_lng'], 'name'=>'Финиш']];
        $route_json = createRouteJson($route);
        $conn->query("UPDATE trips SET status='waiting_payment', route_json='$route_json'
WHERE id={$row['id']}");
        echo json_encode(['status'=>'success']);
    } else echo json_encode(['status'=>'error']);
}


if ($action == 'pay') {
    $trip_id = intval($_POST['trip_id']);
    $conn->query("UPDATE trips SET payment_status='paid' WHERE id=$trip_id");

    $trip = $conn->query("SELECT match_id FROM trips WHERE id=$trip_id")->fetch_assoc();
    if ($trip['match_id']) {
        $mid = $trip['match_id'];
        $check = $conn->query("SELECT count(*) as cnt FROM trips WHERE match_id='$mid'
AND payment_status='unpaid'");
```

```
        if ($check->fetch_assoc()['cnt'] == 0) {
            $conn->query("UPDATE trips SET status='paid' WHERE match_id='$mid'");
        }
    } else {
        $conn->query("UPDATE trips SET status='paid' WHERE id=$trip_id");
    }
    echo json_encode(['status' => 'success']);
}


if ($action == 'get_driver_orders') {

    $sql = "SELECT id, match_id, status, duration_min, route_json, SUM(price) as total_price
            FROM trips WHERE status='paid'
            GROUP BY IFNULL(match_id, id)";

    $res = $conn->query($sql);
    $orders = [];
    while($row = $res->fetch_assoc()) $orders[] = $row;
    echo json_encode(['status'=>'success', 'orders'=>$orders]);
}

if ($action == 'accept_trip') {
    $driver_id = $_SESSION['user_id'];
    $trip_id = $_POST['trip_id'];
    $trip = $conn->query("SELECT match_id FROM trips WHERE id=$trip_id")->fetch_assoc();
    if ($trip['match_id']) {
        $mid = $trip['match_id'];
        $conn->query("UPDATE trips SET status='assigned', driver_id=$driver_id WHERE
match_id='$mid'");
    } else {
        $conn->query("UPDATE trips SET status='assigned', driver_id=$driver_id WHERE
id=$trip_id");
    }
    echo json_encode(['status'=>'success']);
}


if ($action == 'check_status') {
    $uid = $_SESSION['user_id'];
    $role = $_SESSION['role'];

    if ($role == 'driver') {
```

```
    $sql = "SELECT id, match_id, start_lat, start_lng, route_json FROM trips WHERE driver_id=$uid AND status != 'completed' LIMIT 1";
    $res = $conn->query($sql);

    if ($trip = $res->fetch_assoc()) {
      $passengers = [];
      $total_price = 0;

      if ($trip['match_id']) {
        $mid = $trip['match_id'];

        $users_res = $conn->query("SELECT u.name, u.phone, t.price FROM trips t JOIN users u ON t.user_id = u.id WHERE t.match_id='$mid'");
        while($u = $users_res->fetch_assoc()) {
          $total_price += $u['price'];
          $passengers[] = $u;
        }
      } else {
        $tid = $trip['id'];
        $users_res = $conn->query("SELECT u.name, u.phone, t.price FROM trips t JOIN users u ON t.user_id = u.id WHERE t.id=$tid");
        while($u = $users_res->fetch_assoc()) {
          $total_price += $u['price'];
          $passengers[] = $u;
        }
      }

      $trip['passengers'] = $passengers;
      $trip['total_price'] = $total_price;
      echo json_encode(['status'=>'success', 'data'=>$trip]);
    } else {
      echo json_encode(['status'=>'no_trip']);
    }

  } else {
    // Пассажир
    $sql = "SELECT t.*, d.name as d_name, d.car_model, d.car_color, d.car_plate, d.phone as d_phone
        FROM trips t LEFT JOIN users d ON t.driver_id = d.id
        WHERE t.user_id=$uid AND t.status != 'completed' ORDER BY id DESC LIMIT 1";
    $res = $conn->query($sql);

    if ($row = $res->fetch_assoc()) {
      if ($row['status'] == 'waiting_payment' && $row['payment_status'] == 'paid') {
```

```
            $row['custom_message'] = "Ожидаем оплату попутчика...";
        }
        echo json_encode(['status'=>'success', 'data'=>$row]);
    } else {
        echo json_encode(['status'=>'no_trip']);
    }
  }
}

if ($action == 'finish_trip') {
  if ($_SESSION['role'] != 'driver') die();
  $trip_id = intval($_POST['trip_id']);
  $trip = $conn->query("SELECT match_id FROM trips WHERE id=$trip_id")->fetch_assoc();
  if($trip['match_id']) {
    $mid = $trip['match_id'];
    $conn->query("DELETE FROM trips WHERE match_id='$mid'");
  } else {
    $conn->query("DELETE FROM trips WHERE id=$trip_id");
  }
  echo json_encode(['status'=>'success']);
}
?>
```