МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

Факультет информационных технологий
Кафедра «Инфокогнитивные технологии»

КУРСОВОЙ ПРОЕКТ

на тему: *«Разработка веб-приложения для анализа безопасности маршрутов на основе данных о дорожно-транспортных происшествиях»*

Направление подготовки «Проектирование баз данных»
Профиль «Разработка и интеграция бизнес-приложений»

Выполнил:

студент группы 241-362

Яковлев

Артур Олегович

27.06.2026 _____

(подпись)

Москва 2026

1. ВВЕДЕНИЕ

1.1. Цель и задачи проекта

Целью данной работы является разработка веб-приложения для анализа безопасности маршрутов на основе данных о дорожно-транспортных происшествиях (ДТП) с использованием картографического сервиса Яндекс.Карты.

В рамках проекта необходимо было решить несколько ключевых задач. Прежде всего, требовалось разработать систему загрузки и обработки данных о ДТП, которая могла бы работать с большими объемами информации. Далее необходимо было реализовать алгоритм формирования опасных зон на основе плотности ДТП, который бы позволял визуализировать участки дорог с повышенной аварийностью. Важной задачей стала интеграция с API Яндекс.Карт для построения маршрутов между заданными точками. Также требовалось разработать алгоритм анализа пересечений маршрутов с опасными зонами для оценки безопасности различных вариантов пути. Помимо этого, необходимо было создать удобный пользовательский интерфейс для визуализации данных, реализовать систему авторизации и хранения истории маршрутов, а также провести оптимизацию производительности системы для обеспечения быстрой работы с большими объемами данных.

1.2. Описание системы

Разработанное веб-приложение предоставляет пользователям комплексный инструмент для анализа безопасности маршрутов. Основная функциональность включает визуализацию опасных зон на карте Москвы на основе данных о ДТП, где каждая зона отображается цветом в зависимости от уровня опасности. Пользователи могут строить маршруты между двумя точками с использованием API Яндекс.Карт, получая несколько вариантов пути. Для каждого варианта система автоматически вычисляет оценку безопасности на основе пересечений с опасными зонами, что позволяет сравнивать маршруты и выбирать наиболее безопасный. Для авторизованных пользователей доступна функция сохранения

истории маршрутов, что упрощает повторный анализ часто используемых путей.

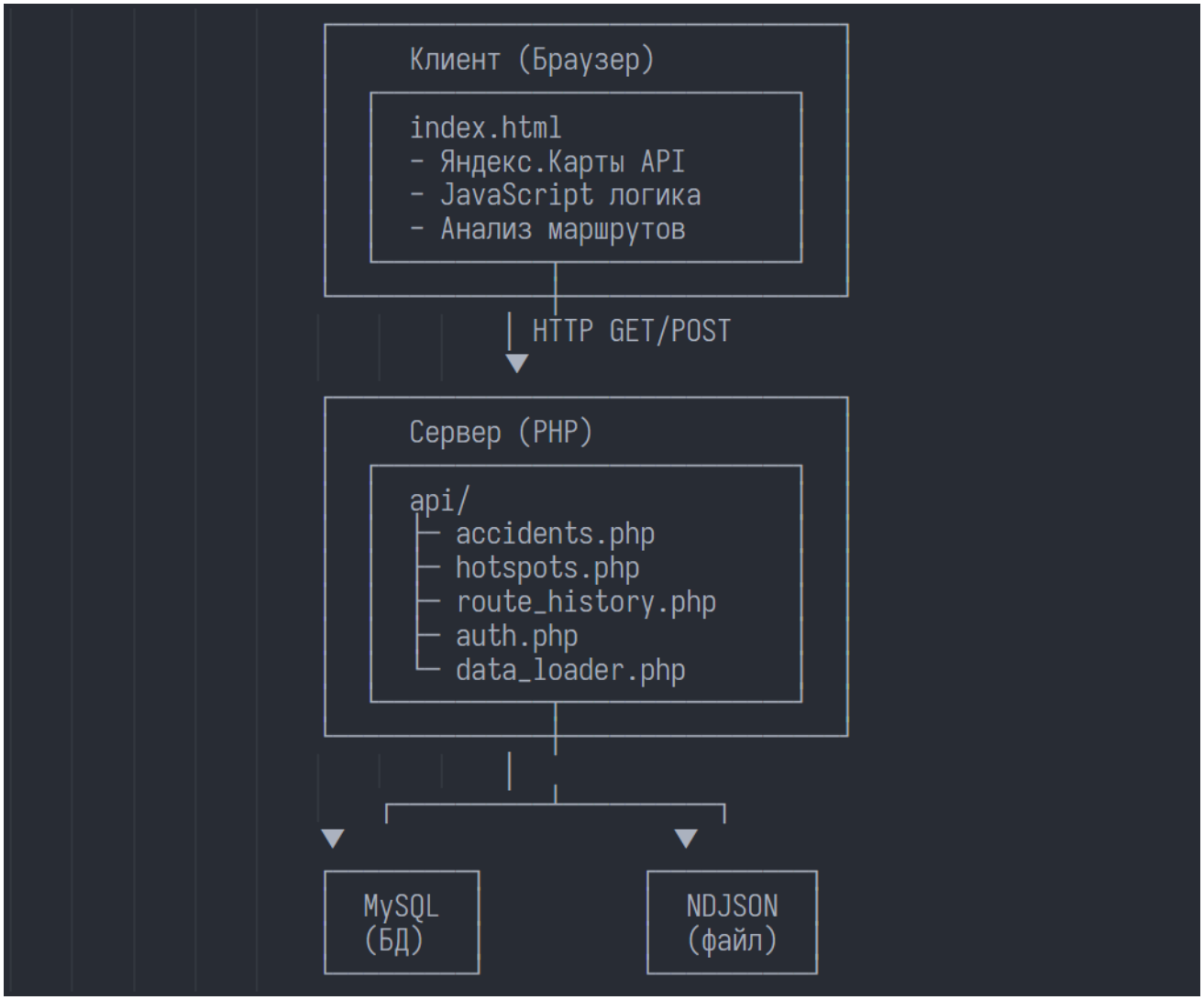
1.3. Технологический стек

В качестве технологий для frontend были выбраны HTML5 и CSS3 для структуры и стилизации интерфейса, JavaScript (ES6+) для реализации клиентской логики, а также Яндекс.Карты API 2.1 для отображения карты и построения маршрутов. Backend реализован на PHP 7.4+ для серверной логики, а для хранения данных используется MySQL 8.0+ с поддержкой spatial типов, что необходимо для эффективной работы с географическими координатами. Исходные данные о ДТП хранятся в формате NDJSON (Newline Delimited JSON), что позволяет обрабатывать большие файлы построчно без загрузки всего содержимого в память.

2. ВВЕДЕНИЕ

2.1. Общая схема системы

Система построена по клиент-серверной архитектуре. Клиентская часть (frontend) отвечает за отображение карты, взаимодействие с пользователем и анализ маршрутов. Серверная часть (backend) предоставляет API для получения данных о ДТП и опасных зонах.



2.2. Структура проекта

Проект организован следующим образом:

```
course_project2/
├── api/                          # Backend API
│   ├── config.php                # Конфигурация системы
│   ├── db.php                    # Подключение к БД
│   ├── data_loader.php           # Универсальный загрузчик данных
│   ├── accidents.php             # API для получения ДТП
│   ├── hotspots.php              # API для получения опасных зон
│   ├── route_history.php         # API для истории маршрутов
│   ├── auth.php                  # API для авторизации
│   └── cache_helper.php          # Утилиты для кэширования
├── index.html                    # Главная страница приложения
├── login.php                     # Страница входа
├── register.php                  # Страница регистрации
├── schema.sql                    # Схема базы данных
├── schema_v2.5.sql               # Дополнения к схеме (авторизация)
└── moskva.ndjson                 # Исходные данные о ДТП
```

2.3. Компоненты системы

Frontend часть системы состоит из нескольких ключевых компонентов. Основной компонент — это карта (index.html), которая служит для визуализации данных и взаимодействия с пользователем. Панель управления содержит форму ввода адресов и настройки параметров анализа, такие как размер ячейки сетки и порог опасности. Панель результатов отображает результаты анализа маршрутов с оценкой безопасности каждого варианта. Панель пользователя показывает информацию о текущем пользователе и историю его маршрутов.

Backend компоненты включают универсальный класс DataLoader, который обеспечивает единый интерфейс для загрузки данных из базы данных или файла, что позволяет легко переключаться между источниками данных. API endpoints реализованы в виде RESTful API для получения данных о ДТП и опасных зонах. Система авторизации обрабатывает регистрацию пользователей, вход в систему и управление сессиями.

2.4. Потоки данных

Процесс загрузки опасных зон начинается с того, что клиент запрашивает зоны для видимой области карты, передавая координаты границ видимой области. Сервер получает этот запрос, вычисляет опасные зоны на основе данных ДТП, используя алгоритм агрегации по сетке, и возвращает данные в формате GeoJSON. Клиент получает эти данные и отображает зоны на карте в виде цветных кругов, где цвет указывает на уровень опасности.

При построении маршрута пользователь вводит адреса начала и конца маршрута в форму на панели управления. Клиент использует API Яндекс.Карт для построения маршрута, получая несколько вариантов пути. После получения координат маршрута система анализирует пересечения каждого варианта с опасными зонами, вычисляя оценку безопасности. Результаты анализа отображаются в панели результатов с рекомендацией наиболее безопасного маршрута.

Сохранение истории происходит автоматически после успешного построения и анализа маршрута. Данные об адресах отправляются на сервер, который сохраняет маршрут в базе данных для авторизованного пользователя. История отображается в панели пользователя, позволяя быстро восстановить ранее проанализированные маршруты.

3. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

3.1. Схема базы данных

База данных состоит из трех основных таблиц:

Таблица `accidents` - хранит данные о ДТП:

```sql
CREATE TABLE IF NOT EXISTS accidents (
  id BIGINT PRIMARY KEY,
  dt DATETIME NULL,
  lat DOUBLE NOT NULL,
  lon DOUBLE NOT NULL,
  geom POINT NOT NULL,
  category VARCHAR(100) NULL,
  severity VARCHAR(50) NULL,
  region VARCHAR(100) NULL,
  light VARCHAR(100) NULL,
  address VARCHAR(255) NULL,
  tags JSON NULL,
  weather JSON NULL,
  nearby JSON NULL,
  vehicles JSON NULL,
  extra JSON NULL,
  KEY idx_dt (dt),
  KEY idx_category (category),
  KEY idx_severity (severity),
  KEY idx_region (region),
  SPATIAL INDEX idx_geom (geom)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Ключевой особенностью таблицы является использование пространственного типа POINT для поля `geom`, которое хранит координаты ДТП с пространственным индексом для быстрого поиска. Поля `lat` и `lon` дублируют координаты в числовом формате, что позволяет выполнять быстрые запросы без использования пространственных функций MySQL, что особенно важно для оптимизации производительности. JSON поля используются для хранения дополнительной информации о ДТП, такой как теги, погодные

условия, близлежащие объекты и информация о транспортных средствах. На часто используемых полях созданы индексы: на дате (`idx_dt`), категории (`idx_category`), тяжести (`idx_severity`) и районе (`idx_region`), что значительно ускоряет фильтрацию данных.

Таблица `users` - пользователи системы:

```
CREATE TABLE IF NOT EXISTS users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  login VARCHAR(50) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_login (login)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Таблица `route_history` - история маршрутов:

```
CREATE TABLE IF NOT EXISTS route_history (
  id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  from_address VARCHAR(255) NOT NULL,
  to_address VARCHAR(255) NOT NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
  INDEX idx_user_id (user_id),
  INDEX idx_created_at (created_at)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

3.2. Индексы и оптимизация

Для обеспечения высокой производительности системы используются различные типы индексов. Пространственный индекс `SPATIAL INDEX idx_geom (geom)` создан на поле с геометрией и используется для быстрого поиска ДТП в заданной области, что критично для работы с географическими данными. Обычные индексы включают `idx_dt` для фильтрации по датам, составной индекс `idx_lat_lon` для запросов по координатам (создается отдельно), и `idx_user_id` для быстрого получения истории маршрутов пользователя.

Оптимизация запросов достигается несколькими способами. Используются составные индексы для частых комбинаций условий, что позволяет базе данных эффективно находить нужные записи. Ограничение выборки через LIMIT предотвращает загрузку избыточных данных, а кэширование результатов

агрегации позволяет избежать повторных вычислений для одинаковых запросов.

4. РАБОТА С ДАННЫМИ ДТП

4.1. Формат данных

Исходные данные хранятся в формате NDJSON (Newline Delimited JSON) - каждая строка файла представляет собой отдельную запись о ДТП в формате JSON:

```
{"id":12345,"dt":"2023-01-15
14:30:00","lat":55.7558,"lon":37.6173,"category":"Столкновение",...}
{"id":12346,"dt":"2023-01-15
15:00:00","lat":55.7560,"lon":37.6175,"category":"Наезд",...}
```

Формат NDJSON был выбран по нескольким причинам. Во-первых, он позволяет обрабатывать файл построчно, не загружая весь файл в память, что критично для работы с большими объемами данных (файл с данными о ДТП может достигать 200 МБ). Во-вторых, этот формат удобен для потоковой обработки, что упрощает импорт данных в базу данных. В-третьих, парсинг NDJSON проще, чем обработка одного большого JSON массива, так как каждая строка представляет собой валидный JSON объект.

4.2. Загрузка данных

Система поддерживает два источника данных, что обеспечивает гибкость при разработке и развертывании. База данных MySQL используется для продакшн использования, обеспечивая высокую производительность и возможность выполнения сложных запросов с агрегацией. NDJSON файл используется для разработки и тестирования, позволяя работать без необходимости настройки базы данных.

Выбор источника данных настраивается в файле `api/config.php` через параметр `data_source`:

```php
return [
    'data_source' => 'database', // или 'file'
    'database' => [
        'host' => 'localhost',
        'dbname' => 'dtp_analysis',
        'user' => 'root',
```

```
        'password' => ''
    ],
    'file' => [
        'ndjson_path' => __DIR__ . '/../moskva.ndjson'
    ]
];
```

4.3. Класс DataLoader

Универсальный класс `DataLoader` обеспечивает единый интерфейс для работы с данными независимо от источника:

```
class DataLoader {
    private $config;
    private $dataSource;
    // Статический кэш для загрузки NDJSON один раз
    private static $accidentsData = null;
    private static $accidentsDataLoaded = false;

    public function __construct($config) {
        $this->config = $config;
        $this->dataSource = $config['data_source'];
    }

    /**
     * Загрузить данные ДТП из файла один раз (singleton)
     */
    private function loadAccidentsDataOnce() {
        if (self::$accidentsDataLoaded && self::$accidentsData !== null) {
            return self::$accidentsData;
        }

        $filePath = $this->config['file']['ndjson_path'];
        $accidents = [];

        $handle = fopen($filePath, 'r');
        if (!$handle) {
            throw new Exception("Не удалось открыть файл: $filePath");
        }

        while (($line = fgets($handle)) !== false) {
            $data = json_decode(trim($line), true);
            if ($data && isset($data['lat'], $data['lon'])) {
                $accidents[] = $data;
            }
        }
        fclose($handle);

        self::$accidentsData = $accidents;
        self::$accidentsDataLoaded = true;
```

```
        return $accidents;
    }
}
```

Класс DataLoader имеет несколько ключевых особенностей, обеспечивающих эффективную работу с данными. Статический кэш предотвращает повторную загрузку файла в рамках одного запроса, что особенно важно при работе с большими файлами. Построчное чтение файла экономит память, так как не требуется загружать весь файл целиком. Валидация данных при загрузке гарантирует, что в систему попадают только корректные записи с обязательными полями координат.

4.4. API endpoint для получения ДТП

Endpoint `api/accidents.php` возвращает данные о ДТП в указанной области:

```
// Параметры запроса
$bbox = $_GET['bbox']; // "minLon,minLat,maxLon,maxLat"
$period = $_GET['period'] ?? 'all'; // "7d", "30d", "all"
$limit = (int)($_GET['limit'] ?? 1000);

// Парсинг bounding box
$bbox_parts = array_map('trim', explode(',', $bbox));
$minLon = (float)$bbox_parts[0];
$minLat = (float)$bbox_parts[1];
$maxLon = (float)$bbox_parts[2];
$maxLat = (float)$bbox_parts[3];

// Загрузка данных
$loader = new DataLoader($config);
$accidents = $loader->getAccidents($bbox, $period, $limit);

// Фильтрация по области
$filtered = array_filter($accidents, function($acc) use ($minLat, $maxLat, $minLon,
$maxLon) {
    return $acc['lat'] >= $minLat && $acc['lat'] <= $maxLat &&
           $acc['lon'] >= $minLon && $acc['lon'] <= $maxLon;
});

// Возврат в формате GeoJSON
echo json_encode([
    'type' => 'FeatureCollection',
    'features' => array_map(function($acc) {
        return [
            'type' => 'Feature',
```

```
            'geometry' => [
                'type' => 'Point',
                'coordinates' => [$acc['lon'], $acc['lat']]
            ],
            'properties' => $acc
        ];
    }, $filtered)
], JSON_UNESCAPED_UNICODE);
```

Оптимизация работы endpoint достигается несколькими способами. Фильтрация данных на сервере уменьшает объем передаваемых данных по сети, что особенно важно при работе с большими наборами данных. Ограничение количества записей через параметр `limit` предотвращает перегрузку клиента и обеспечивает быстрый отклик системы. Кэширование результатов для повторяющихся запросов позволяет избежать повторных вычислений и значительно ускоряет работу приложения.

5. ФОРМИРОВАНИЕ ОПАСНЫХ ЗОН

5.1. Формат данных

Опасные зоны формируются на основе агрегации ДТП по сетке. Алгоритм начинается с разбиения области на ячейки сетки, где размер ячейки задается параметром `grid` (по умолчанию 1000 метров). Для каждой ячейки вычисляется количество ДТП, что позволяет определить плотность аварийности на участке.

Определение уровня опасности происходит на основе плотности ДТП на единицу площади. Низкий уровень опасности (желтый цвет) соответствует плотности 0.1-0.2 ДТП на ячейку, средний уровень (оранжевый) — 0.2-0.3 ДТП на ячейку, а высокий уровень (красный) — плотности ≥ 0.3 ДТП на ячейку. Для уменьшения визуального шума и фокусировки на действительно проблемных участках применяется фильтрация: отображаются только ячейки с количеством ДТП не менее `threshold` (по умолчанию 5).

5.2. Визуализация на карте

На клиенте опасные зоны отображаются в виде цветных кругов:

```
function loadHotspotsOnce() {
    const bounds = map.getBounds()
```

```
    const bbox = [
        bounds[0][1], // minLon
        bounds[0][0], // minLat
        bounds[1][1], // maxLon
        bounds[1][0], // maxLat
    ].join(',')

    const gridSize = parseInt(document.getElementById('gridSize').value) || 1000

    fetch(`api/hotspots.php?bbox=${bbox}&period=all&threshold=5&grid=${gridSize}`)
        .then(response => response.json())
        .then(data => {
            hotspotsCollection.removeAll()

            data.features.forEach(feature => {
                const props = feature.properties
                const coords = feature.geometry.coordinates

                // Определение цвета по уровню опасности
                let color = '#fff9c4' // желтый - низкий
                if (props.riskLevel === 'high')
                    color = '#e74c3c' // красный
                else if (props.riskLevel === 'medium') color = '#f39c12' // оранжевый

                // Создание круга
                const circle = new ymaps.Circle(
                    [
                        [coords[1], coords[0]], // центр
                        props.radius, // радиус в метрах
                    ],
                    {},
                    {
                        fillColor: color,
                        fillOpacity: 0.6,
                        strokeColor: color,
                        strokeWidth: 2,
                    }
                )

                hotspotsCollection.add(circle)
            })
        })
}
```

6. СИСТЕМА АВТОРИЗАЦИИ

Опасные зоны формируются на основе агрегации ДТП по сетке. Алгоритм начинается с разбиения области на ячейки сетки, где размер ячейки задается параметром `grid` (по умолчанию 1000 метров). Для каждой ячейки вычисляется

количество ДТП, что позволяет определить плотность аварийности на участке.

Система авторизации использует PHP сессии и хэширование паролей:

```php
function handleLogin($pdo) {
    $login = trim($_POST['login']);
    $password = $_POST['password'];

    // Поиск пользователя
    $stmt = $pdo->prepare("SELECT id, login, password_hash FROM users WHERE login = ?");
    $stmt->execute([$login]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$user || !password_verify($password, $user['password_hash'])) {
        http_response_code(401);
        echo json_encode(['error' => 'Неверный логин или пароль']);
        return;
    }

    // Установка сессии
    $_SESSION['user_id'] = $user['id'];
    $_SESSION['user_login'] = $user['login'];

    echo json_encode([
        'success' => true,
        'user' => ['id' => $user['id'], 'login' => $user['login']]
    ]);
}
```

Безопасность системы авторизации обеспечивается несколькими механизмами. Использование функции `password_hash()` с алгоритмом bcrypt гарантирует, что пароли хранятся в зашифрованном виде и не могут быть восстановлены даже при компрометации базы данных. Проверка пароля через `password_verify()` обеспечивает безопасное сравнение введенного пароля с хэшем в базе данных.

7. ОПТИМИЗАЦИЯ ПРОИЗВОДИТЕЛЬНОСТИ

7.1. Оптимизация запросов к БД

Изначальная реализация выполняла отдельный SQL-запрос для каждой ячейки сетки. Для области размером 1°×1° с сеткой 1000 метров это означало выполнение примерно 10,000 запросов, что приводило к превышению времени выполнения PHP скрипта (120 секунд) и делало систему непригодной для

использования.

Решение проблемы заключалось в использовании одного SQL-запроса с группировкой вместо множества отдельных запросов. Это позволило базе данных эффективно обработать все ячейки за один проход, используя оптимизацию запросов на уровне СУБД. Сравнение подходов:

```
$sql = "
    SELECT
        FLOOR((lat - ?) / ?) as lat_idx,
        FLOOR((lon - ?) / ?) as lon_idx,
        COUNT(*) as cnt
    FROM accidents
    WHERE lat BETWEEN ? AND ? AND lon BETWEEN ? AND ?
    GROUP BY lat_idx, lon_idx
    HAVING cnt >= ?
";
$stmt = $pdo->prepare($sql);
$stmt->execute([...]);
$results = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

7.2. Кэширование данных

Кэширование API ответов реализовано через файловую систему. Функция `getCachedResponse` проверяет наличие кэш-файла и его актуальность (время жизни задается параметром TTL, по умолчанию 3600 секунд). Если кэш актуален, данные возвращаются из файла без выполнения запроса к базе данных. Функция `saveCachedResponse` сохраняет результаты запросов в файлы, используя MD5 хэш параметров запроса в качестве имени файла.

7.3. Оптимизация на клиенте

На клиентской стороне применяются техники ленивой загрузки и дебаунсинга. Ленивая загрузка данных реализована через флаг `hotspotsLoaded`, который предотвращает повторную загрузку опасных зон, если они уже были загружены. Это особенно важно при перемещении по карте, когда пользователь может многократно изменять видимую область.

Дебаунсинг запросов используется для оптимизации загрузки данных при изменении границ карты. Вместо немедленной загрузки данных при каждом изменении границ, система ждет 500 миллисекунд после последнего изменения.

Если за это время границы снова изменились, таймер сбрасывается. Это предотвращает избыточные запросы к серверу при быстром перемещении по карте и значительно снижает нагрузку на сервер

8. ТЕСТИРОВАНИЕ И РЕЗУЛЬТАТЫ

8.1. Функциональное тестирование

Функциональное тестирование системы включало проверку всех основных компонентов. Тестирование построения маршрутов подтвердило корректность работы с API Яндекс.Карт: система успешно строит маршруты между двумя адресами, получает несколько вариантов маршрута и правильно отображает их на карте. Тестирование анализа безопасности показало, что алгоритм корректно рассчитывает пересечения маршрутов с опасными зонами, правильно вычисляет оценку безопасности каждого маршрута и корректно сравнивает несколько маршрутов между собой. Тестирование системы авторизации подтвердило работоспособность регистрации новых пользователей, входа в систему и сохранения истории маршрутов с последующей загрузкой.

8.2. Производительность

Измерения производительности показали высокую скорость работы системы. Загрузка опасных зон для видимой области карты занимает менее 1 секунды, что обеспечивает плавную работу интерфейса. Построение маршрута через API Яндекс.Карт занимает от 1 до 3 секунд в зависимости от сложности маршрута и загрузки серверов Яндекс. Анализ маршрута на клиенте выполняется менее чем за 0.5 секунды, что практически незаметно для пользователя.

Оптимизация запросов к базе данных дала впечатляющие результаты. До оптимизации система не справлялась с обработкой больших областей, превышая лимит времени выполнения в 120 секунд. После оптимизации время выполнения сократилось до менее 1 секунды, что представляет собой улучшение более чем в 120 раз.

8.3. Примеры работы системы

Система успешно обрабатывает маршруты по Москве различной сложности и длины. Тестирование на реальных маршрутах показало, что система корректно определяет опасные зоны на основе исторических данных о ДТП и предоставляет пользователю обоснованные рекомендации по выбору наиболее безопасного маршрута. Визуализация опасных зон на карте позволяет пользователю интуитивно понимать распределение аварийности по городу, а сравнение нескольких вариантов маршрута помогает принимать обоснованные решения о выборе пути.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ:

1. Документация Яндекс.Карты API 2.1 - https://yandex.ru/dev/maps/jsapi/
2. PHP Manual - https://www.php.net/manual/ru/
3. MySQL Spatial Data Types - https://dev.mysql.com/doc/refman/8.0/en/spatial-types.html
4. MDN Web Docs - JavaScript - https://developer.mozilla.org/ru/docs/Web/JavaScript