7. Справочник по API

Содержание

1. Введение
2. Общие принципы работы с API
3. Аутентификация и авторизация
4. API управления устройствами
5. API данных пассажиропотока
6. API управления пользователями
7. Real-time интерфейсы
8. Примеры интеграции
9. Ограничения и квоты

1. Введение

1.1. Назначение справочника

Данный справочник содержит описание программных интерфейсов (API) системы BusStory, предназначенных для интеграции с внешними системами и разработки клиентских приложений. API построено на современных принципах REST с поддержкой real-time обновлений.

1.2. Архитектура API

Основные компоненты:

- **REST API** - основной интерфейс для CRUD операций
- **WebSocket API** - real-time обновления и события
- **GraphQL Endpoint** - гибкие запросы для сложной аналитики
- **Webhook API** - уведомления о событиях системы

Базовая структура:

```
API Gateway → Authentication → Rate Limiting → Business Logic → Database
     ↓              ↓               ↓              ↓             ↓
  Routing      JWT Tokens     Request Quotas   Processing    Data Storage
```

1.3. Версионирование API

Текущая версия: v1.2.0

Поддерживаемые версии:

- v1.2.x - текущая стабильная версия
- v1.1.x - поддерживается до конца 2025 года
- v1.0.x - deprecated, поддержка до середины 2025 года

2. Общие принципы работы с API

2.1. Базовые URL

Production окружение:

```
https://api.busstory.example.com/v1/
```

Staging окружение:

```
https://api-staging.busstory.example.com/v1/
```

2.2. Форматы данных

Входные данные:

- JSON для POST/PUT запросов
- Query parameters для GET запросов
- Multipart/form-data для загрузки файлов

Выходные данные:

- JSON для всех ответов
- Стандартная структура ответов с метаданными
- Поддержка пагинации для списков

Пример стандартного ответа:

```
{
  "success": true,
  "data": {
    "items": [...],
    "meta": {
      "total": 150,
      "page": 1,
      "limit": 20,
      "pages": 8
    }
  },
```

```
  "timestamp": "2025-06-30T10:30:00.000Z"
}
```

2.3. HTTP коды ответов

Успешные операции:

- `200 OK` - успешный запрос
- `201 Created` - ресурс создан
- `204 No Content` - успешное удаление

Ошибки клиента:

- `400 Bad Request` - неверный запрос
- `401 Unauthorized` - требуется аутентификация
- `403 Forbidden` - доступ запрещен
- `404 Not Found` - ресурс не найден
- `429 Too Many Requests` - превышен лимит запросов

Ошибки сервера:

- `500 Internal Server Error` - внутренняя ошибка
- `502 Bad Gateway` - ошибка шлюза
- `503 Service Unavailable` - сервис недоступен

3. Аутентификация и авторизация

3.1. Методы аутентификации

JWT Token Authentication (рекомендуется):

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

API Key Authentication:

```
X-API-Key: your-api-key-here
```

3.2. Получение токена доступа

Endpoint: `POST /auth/login`

Запрос:

```
{
  "username": "user@example.com",
```

```
  "password": "secure_password"
}
```

Ответ:

```
{
  "success": true,
  "data": {
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "refresh_token": "refresh_token_here",
    "expires_in": 3600,
    "user": {
      "id": "user123",
      "username": "user@example.com",
      "role": "operator"
    }
  }
}
```

3.3. Обновление токена

Endpoint: `POST /auth/refresh`

Запрос:

```
{
  "refresh_token": "refresh_token_here"
}
```

4. API управления устройствами

4.1. Получение списка устройств

Endpoint: `GET /devices`

Параметры запроса:

- `status` - фильтр по статусу (online, offline, maintenance)
- `location` - фильтр по географической области
- `limit` - количество записей (по умолчанию 20)
- `page` - номер страницы (по умолчанию 1)

Пример запроса:

```
GET /devices?status=online&limit=10&page=1
```

Ответ:


```
{
  "success": true,
  "data": {
    "items": [
      {
        "id": "device123",
        "name": "Bus Route 101 Counter",
        "status": "online",
        "location": {
          "latitude": 47.2357,
          "longitude": 39.7015
        },
        "last_seen": "2025-06-30T10:30:00.000Z",
        "battery_level": 85,
        "accuracy": 0.96
      }
    ],
    "meta": {
      "total": 45,
      "page": 1,
      "limit": 10,
      "pages": 5
    }
  }
}
```


4.2. Получение информации об устройстве

Endpoint: `GET /devices/{device_id}`

Ответ:


```
{
  "success": true,
  "data": {
    "id": "device123",
    "name": "Bus Route 101 Counter",
    "status": "online",
    "hardware": {
      "model": "BusStory Counter v2.1",
      "firmware": "1.4.2",
      "serial_number": "BSC202500123"
    },
    "configuration": {
      "counting_zone": {
        "x": 100, "y": 50,
```

```
        "width": 200, "height": 300
      },
      "sensitivity": 0.85
    },
    "statistics": {
      "total_counted": 15420,
      "today_counted": 342,
      "accuracy": 0.94,
      "uptime": 99.2
    }
  }
}
```

4.3. Обновление конфигурации устройства

Endpoint: `PUT /devices/{device_id}/config`

Запрос:

```
{
  "counting_zone": {
    "x": 120, "y": 60,
    "width": 180, "height": 280
  },
  "sensitivity": 0.90,
  "reporting_interval": 30
}
```

4.4. Отправка команды устройству

Endpoint: `POST /devices/{device_id}/commands`

Доступные команды:

- `restart` - перезагрузка устройства
- `calibrate` - калибровка алгоритмов
- `update_firmware` - обновление прошивки
- `reset_counters` - сброс счетчиков

Запрос:

```
{
  "command": "restart",
  "parameters": {
    "force": true
```

```
  }
}
```

5. API данных пассажиропотока

5.1. Получение данных в реальном времени

Endpoint: `GET /passenger-flow/current`

Параметры:

- `device_ids` - список ID устройств (через запятую)
- `time_window` - временное окно в секундах (по умолчанию 300)

Ответ:

```
{
  "success": true,
  "data": {
    "timestamp": "2025-06-30T10:30:00.000Z",
    "devices": [
      {
        "device_id": "device123",
        "current_flow": {
          "in": 3,
          "out": 1,
          "net": 2
        },
        "last_5_minutes": {
          "in": 12,
          "out": 8,
          "net": 4
        }
      }
    ],
    "summary": {
      "total_in": 3,
      "total_out": 1,
      "total_net": 2
    }
  }
}
```

5.2. Исторические данные

Endpoint: `GET /passenger-flow/history`

Параметры:

- `start_date` - начальная дата (ISO 8601)
- `end_date` - конечная дата (ISO 8601)
- `device_ids` - список устройств
- `granularity` - детализация (minute, hour, day)

Пример запроса:

```
GET /passenger-flow/history?start_date=2025-06-29T00:00:00Z&end_date=2025-06-30T23:59:59Z&granularity=hour
```

5.3. Аналитические отчеты

Endpoint: `POST /reports/generate`

Типы отчетов:

- `daily_summary` - ежедневная сводка
- `weekly_overview` - еженедельный обзор
- `route_analysis` - анализ маршрутов
- `peak_hours` - часы пик

Запрос:

```
{
  "report_type": "daily_summary",
  "date": "2025-06-30",
  "device_ids": ["device123", "device456"],
  "format": "json"
}
```

6. API управления пользователями

6.1. Получение списка пользователей

Endpoint: `GET /users`

Требует роль администратора

Ответ:

```
{
  "success": true,
  "data": {
    "items": [
      {
        "id": "user123",
```

```
        "username": "operator@example.com",
        "role": "operator",
        "status": "active",
        "last_login": "2025-06-30T09:15:00.000Z",
        "created_at": "2025-01-15T10:00:00.000Z"
      }
    ]
  }
}
```

6.2. Создание пользователя

Endpoint: `POST /users`

Запрос:

```
{
  "username": "newuser@example.com",
  "password": "secure_password",
  "role": "operator",
  "profile": {
    "first_name": "Имя",
    "last_name": "Фамилия",
    "department": "Диспетчерская"
  }
}
```

6.3. Управление ролями

Endpoint: `PUT /users/{user_id}/role`

Запрос:

```
{
  "role": "administrator",
  "permissions": {
    "devices": ["read", "write", "configure"],
    "reports": ["read", "generate"],
    "users": ["read", "write"]
  }
}
```

7. Real-time интерфейсы

7.1. WebSocket подключение

URL: `wss://api.busstory.example.com/ws`

Инициализация:

```
const ws = new WebSocket('wss://api.busstory.example.com/ws');

ws.onopen = function() {
  // Аутентификация
  ws.send(JSON.stringify({
    type: 'auth',
    token: 'your-jwt-token'
  }));
};
```

7.2. Подписка на события

Типы событий:

- `device_status` - изменения статуса устройств
- `passenger_count` - новые данные подсчета
- `system_alert` - системные уведомления
- `user_activity` - активность пользователей

Подписка:

```
ws.send(JSON.stringify({
  type: 'subscribe',
  events: ['device_status', 'passenger_count'],
  filters: {
    device_ids: ['device123', 'device456']
  }
}));
```

7.3. Обработка событий

Пример события:

```
{
  "type": "passenger_count",
  "device_id": "device123",
  "timestamp": "2025-06-30T10:30:15.000Z",
  "data": {
    "in": 2,
    "out": 0,
    "confidence": 0.92,
    "zone": "door_1"
  }
}
```

8. Примеры интеграции

8.1. JavaScript/Node.js клиент

```
class BusStoryAPI {
  constructor(baseUrl, apiKey) {
    this.baseUrl = baseUrl;
    this.apiKey = apiKey;
    this.token = null;
  }

  async authenticate(username, password) {
    const response = await fetch(`${this.baseUrl}/auth/login`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'X-API-Key': this.apiKey
      },
      body: JSON.stringify({ username, password })
    });

    const data = await response.json();
    this.token = data.data.access_token;
    return data;
  }

  async getDevices(filters = {}) {
    const params = new URLSearchParams(filters);
    const response = await fetch(`${this.baseUrl}/devices?${params}`, {
      headers: {
        'Authorization': `Bearer ${this.token}`,
        'X-API-Key': this.apiKey
      }
    });

    return response.json();
  }

  async getCurrentFlow(deviceIds) {
    const params = new URLSearchParams({
      device_ids: deviceIds.join(',')
    });

    const response = await fetch(`${this.baseUrl}/passenger-flow/current?
${params}`, {
      headers: {
        'Authorization': `Bearer ${this.token}`,
        'X-API-Key': this.apiKey
      }
```

```
    });

    return response.json();
  }
}

// Использование
const api = new BusStoryAPI('https://api.busstory.example.com/v1', 'your-
api-key');
await api.authenticate('user@example.com', 'password');
const devices = await api.getDevices({ status: 'online' });
```

8.2. Python клиент

```
import requests
import websocket
import json

class BusStoryAPI:
    def __init__(self, base_url, api_key):
        self.base_url = base_url
        self.api_key = api_key
        self.token = None
        self.session = requests.Session()
        self.session.headers.update({'X-API-Key': api_key})

    def authenticate(self, username, password):
        response = self.session.post(f'{self.base_url}/auth/login', json={
            'username': username,
            'password': password
        })
        data = response.json()
        self.token = data['data']['access_token']
        self.session.headers.update({'Authorization': f'Bearer
{self.token}'})
        return data

    def get_devices(self, **filters):
        response = self.session.get(f'{self.base_url}/devices',
params=filters)
        return response.json()

    def get_current_flow(self, device_ids):
        params = {'device_ids': ','.join(device_ids)}
        response = self.session.get(f'{self.base_url}/passenger-
flow/current', params=params)
        return response.json()
```

```python
# Использование
api = BusStoryAPI('https://api.busstory.example.com/v1', 'your-api-key')
api.authenticate('user@example.com', 'password')
devices = api.get_devices(status='online')
```

8.3. Webhook интеграция

Настройка webhook:

```http
POST /webhooks
Content-Type: application/json

{
  "url": "https://your-system.com/webhook/busstory",
  "events": ["device_offline", "high_passenger_flow"],
  "secret": "webhook-secret",
  "active": true
}
```

Обработка webhook:

```javascript
app.post('/webhook/busstory', (req, res) => {
  const signature = req.headers['x-busstory-signature'];
  const payload = JSON.stringify(req.body);

  // Проверка подписи
  const expectedSignature = crypto
    .createHmac('sha256', 'webhook-secret')
    .update(payload)
    .digest('hex');

  if (signature !== expectedSignature) {
    return res.status(401).send('Invalid signature');
  }

  // Обработка события
  const { type, data } = req.body;

  switch (type) {
    case 'device_offline':
      handleDeviceOffline(data);
      break;
    case 'high_passenger_flow':
      handleHighFlow(data);
      break;
  }
```

```
  res.status(200).send('OK');
});
```

9. Ограничения и квоты

9.1. Лимиты запросов (Rate Limiting)

По умолчанию:

- 1000 запросов в час на API ключ
- 100 WebSocket соединений на пользователя
- 10 одновременных webhook delivery

Для корпоративных клиентов:

- 10,000 запросов в час
- 1000 WebSocket соединений
- 100 одновременных webhook delivery

9.2. Ограничения данных

Размер запроса:

- Максимум 10MB для файловых загрузок
- Максимум 1MB для JSON запросов
- Максимум 1000 элементов в массиве

Временные окна:

- Исторические данные: максимум 1 год за запрос
- Real-time данные: максимум 24 часа
- Экспорт отчетов: максимум 100MB на файл

9.3. Мониторинг использования

Заголовки ответа:

```
X-RateLimit-Limit: 1000
X-RateLimit-Remaining: 999
X-RateLimit-Reset: 1640995200
```

Превышение лимитов:

```
{
  "success": false,
```

```
  "error": {
    "code": "RATE_LIMIT_EXCEEDED",
    "message": "Rate limit exceeded. Try again in 60 seconds.",
    "retry_after": 60
  }
}
```

Контактная информация:

ООО «ДонНовоТех»
344000, Россия, г. Ростов-на-Дону, ул. М.Горького 205
Генеральный директор: Евгений Львович Левитин
Телефон: +7 904 500 6087
Email: leojohn@yandex.ru
Веб-сайт: https://donnovotech.ru