Table of Contents

API Keys Management Service

Comprehensive Documentation for ApiKey REST Controller

The `ApiKey` REST Controller is designed to manage **CRUD operations** and additional functionalities for handling API keys in the **Parking Management System.** These keys are critical for authentication and must be included in all client requests using a custom header: `X-Api-Key: {apiKeyValue}`.

The controller interacts with the `ApiKey` entity and provides the ability to **create**, **retrieve**, **update**, **revoke**, and **delete API keys.** Only users with **administrator privileges** can generate API keys. Below, you will find all the necessary specifications and detailed information for implementing and using the available endpoints.

Endpoint Specifications

1. Create a New API Key

- **URL**: `/api/v1/api-keys/generate/{parkingId}`
- **Method**: `POST`
- **Description**: Allows administrators to create a new API key.
- **Path Parameters**:
  - `parkingId`: UUID of the parkingId to associate new key.
  - **Key Format**:
    - `keyValue`: At least 16 characters long, containing letters and digits.
- **Request Body (JSON)**:

```
{
  "scope": ["SCOPE_1", "SCOPE_2"]
}
```

- **Response (201 - Created)**:

```
{
  "id": "UUID of the API key",
  "keyValue": "unique-key-value",
  "parkingId": "UUID of the parking",
  "scope": ["SCOPE_1", "SCOPE_2"],
  "issuedBy": "UUID of the issuer",
  "revokedBy": null,
  "status": "ACTIVE"
}
```

- **Error Responses**:
  - **400 Bad Request**: Invalid request body structure or missing required fields.

2. Fetch API Key Details

- **URL**: `/api/v1/api-keys/{id}`
- **Method**: `GET`
- **Description**: Retrieves the details of a specific API key using its unique ID.
- **Path Parameters**:
  - `id`: UUID of the API key to fetch information for.
- **Response (200 - Success)**:

```
{
  "id": "UUID of the API key",
  "keyValue": "unique-key-value",
  "parkingId": "UUID of the parking",
  "scope": ["SCOPE_1", "SCOPE_2"],
  "issuedBy": "UUID of the issuer",
  "revokedBy": null,
  "status": "ACTIVE"
}
```

- **Error Responses**:
  - **404 Not Found**: API key with the specified ID does not exist.

3. List All API Keys

- **URL**: `/api/v1/api-keys/search`
- **Method**: `GET`
- **Description**: Fetches a paginated list of all registered API keys with optional filters.
- **Query Parameters**:
  - `status` (Optional): Filter results by status (`ACTIVE`, `REVOKED`).
  - `parkingId` (Optional): Filtering by parking id
  - **Filters Logic**: All specified filters are applied using `AND` logic.
  - `page` (Optional): Page number for pagination (default: `1`).
  - `size` (Optional): Number of results per page (default: `10`).
- **Response (200 - Success)**:

```json
{
  "content": [
    {
      "id": "UUID of the API key",
      "parkingId": "UUID of the parking",
      "scope": ["SCOPE_1", "SCOPE_2"],
      "status": "ACTIVE"
    }
  ],
  "page": 1,
  "size": 10,
  "totalElements": 50
}
```

4. Revoke an API Key

- **URL**: `/api/v1/api-keys/{id}/revoke`
- **Method**: `PUT`
- **Description**: Revokes an active API key, marking it as no longer valid.
- **Path Parameters**:
  - `id`: UUID of the API key to be revoked.
- **Request Body (JSON)**:

```json
{
  "revokedBy": "UUID of the current session user (filled
automatically)"
}
```

- **Response (200 - Success)**:

```json
{
  "id": "UUID of the API key",
  "keyValue": "unique-key-value",
  "parkingId": "UUID of the parking",
  "scope": ["SCOPE_1", "SCOPE_2"],
  "issuedBy": "UUID of the issuer",
  "revokedBy": "UUID of revoker",
  "status": "REVOKED"
}
```

- **Error Responses**:
  - **400 Bad Request**: API key is already revoked.
  - **404 Not Found**: API key with the specified ID does not exist.

5. Delete an API Key

- **URL**: `/api/v1/api-keys/{id}`
- **Method**: `DELETE`
- **Description**: Permanently deletes the specified API key from the system.
- **Path Parameters**:
  - `id`: UUID of the API key to be deleted.
- **Response (204 - No Content)**:
  - No content provided on success.
- **Error Responses**:
  - **404 Not Found**: API key with the specified ID does not exist.

6. Validate API Key

- **URL**: `/api/v1/api-keys/validate/{keyValue}`
- **Method**: `GET`
- **Description**: Validates the provided API key and retrieves its details if valid.
- **Path Parameters**:
  - `keyValue`: API key to be validated.
- **Response (200 - Success)**:

```json
{
  "id": "UUID of the API key",
  "keyValue": "unique-key-value",
  "parkingId": "UUID of the parking",
  "scope": ["SCOPE_1", "SCOPE_2"],
  "issuedBy": "UUID of the issuer",
  "revokedBy": null,
  "status": "ACTIVE"
}
```

- **Error Responses**:
  - **404 Not Found**: API key with the specified ID does not exist.

Request Validation Rules

1. **Key Uniqueness**: The `keyValue` field must be globally unique.
2. **Valid UUID Format**:
   - Fields such as `parkingId`, `issuedBy`, and `revokedBy` must conform to the UUID standard.
   - `revokedBy` field is populated automatically with the session user's UUID; cannot be set manually.
3. **Field Constraints**:
   - `scope` field must not be empty and must include at least one valid permission.
   - Valid values for `scope`: `['SCOPE_1', 'SCOPE_2']`. The list of permissions is synchronized with the system's database.
   - `status` must be one of the accepted values: `ACTIVE`, `INACTIVE`, or `REVOKED`.

Controller Class Outline

The `ApiKeyController` class implements the following endpoints:

- `@PostMapping("/api/v1/api-keys/generate/{parkingId}")`: Create a new API key.
- `@GetMapping("/api-keys/search")`: List all API keys (paginated).
- `@PutMapping("/api-keys/{id}/revoke")`: Revoke an API key by ID.
- `@DeleteMapping("/api-keys/{id}")`: Delete an existing API key by ID.
- `@DeleteMapping("/api-keys/validate/{keyValue}")`: Validate an API key

Example Use Case

Revoking an API Key

1. **Client Action**:
    - The administrator sends a request to revoke an API key with ID `1234`.
2. **System Validation**:
    - Confirms the key exists and its current status is `ACTIVE`.
3. **Outcome**:
    - API key's status is updated to `REVOKED`.
    - The `revokedBy` field is populated automatically with the session user's UUID; cannot be set manually.

Example Request:

```
PUT /api/v1/api-keys/1234/revoke
Content-Type: application/json
```

Example Response:

```json
{
  "id": "1234",
  "keyValue": "unique-key-value",
  "parkingId": "UUID of the parking",
  "scope": ["SCOPE_1", "SCOPE_2"],
  "issuedBy": "UUID of the issuer",
  "revokedBy": "admin-uuid",
  "status": "REVOKED"
}
```

Security and Logging

- This API uses secure authentication mechanisms such as tokens or API key validation.

- All sensitive operations (creation, revocation, deletion) should be logged for auditing purposes.

Notes:

1. Proper error handling ensures robust fault tolerance.
2. This API design is compliant with REST principles for scalability and maintainability.

Parking Creation Process

Endpoint: POST /parking/creation

Description:

This endpoint is used to create a new parking and associate it with an owner. To successfully interact with this endpoint, the request must include a valid JWT token for authentication and authorization.

Requirements:

- **JWT Token**: A valid JSON Web Token (JWT) must be included in the request headers for authentication.
- **User Role**:
    - The user must have the `PARKING_OWNER` role in the system.
    - The user must also have the necessary permissions associated with this role.

Purpose:

This operation is restricted to users assigned the `PARKING_OWNER` role, ensuring that only authorized individuals can associate a new parking entity to the specified owner.

Request body to create new parking:

```
{
  "name": "Central Parking",
  "address": {
    "city": "New York",
    "countryCode": "US",
    "postalCode": "10001",
    "street": "5th Avenue",
    "buildingNumber": "12A",
    "latitude": 40.712776,
    "longitude": -74.005974,
    "isBelongToAnyInstitution": true,
      "institutionName": "Big Corp"
    }
  }
```

Authorization and Authentication System Documentation

This document provides comprehensive guidelines for frontend developers on how to implement authorization and authentication in the system using REST APIs. The system relies on **JWT (JSON Web Tokens)** for secure communication. This guide outlines the authorization workflow, available API endpoints, error handling, and example usage to help you integrate seamlessly into the system.

Overview

The authorization system is designed to securely manage user authentication and protect access to resources using tokens. It supports:

- **Credentials-based authentication**: Login using username and password.
- **JWT-based authorization**: Protect API requests using an access token (short-lived).
- **Token refreshing**: Obtain a new access token when the old one expires without re-authentication.

Authorization Workflow

1. **Authentication**: A user logs in with their credentials and receives two tokens:

- **Access Token**: Used for accessing protected resources.
- **Refresh Token**: Used to generate a new access token when the old one expires.

2. **Securing Requests**: The access token is sent in the `Authorization` header for all secure API calls.
3. **Refreshing Tokens**: If an access token expires, the refresh token is used to obtain a new access token without requiring the user to log in again.

API Endpoints

The following API endpoints define the authentication and authorization flow within the system.

1. Login: Token Retrieval

Authenticate the user and retrieve tokens for future requests.

- **Endpoint**: `/api/v1/security/login`
- **Method**: `POST`

Request (Login):

- **Headers**: None
- **Body**:

```
{
  "login": "<string>",
  "password": "<string>"
}
```

Login Response:

- **200 OK** (Success):

```
{
  "accessToken": "<string>",
  "accessTokenExpiry": "<datetime>",
  "refreshToken": "<string>",
  "refreshTokenExpiry": "<datetime>"
}
```

- **401 Unauthorized** (Failure): If provided credentials are invalid.

2. Access Protected Resources

Access secured endpoints by including the access token in the request header.

- **Endpoint**: Varies based on the specific resource (e.g., `/api/secure/resource`)
- **Method**: Depends on the requested resource (e.g., `GET`, `POST`, etc.)

Request (Protected Resource):

- **Headers**:

```
Authorization: Bearer <accessToken>
```

- **Body** (if required by the resource):

```
{
  "examplePayloadField": "<value>"
}
```

Protected Resource Response:

- **200 OK** (Success): Returns the requested resource.
- **Error Responses**:
  - **401 Unauthorized**: Invalid or expired access token.
  - **403 Forbidden**: User lacks the required permissions to access the resource.

3. Token Refresh

Obtain a new access token using the refresh token.

- **Endpoint**: `/api/v1/security/jwt/refresh-token`
- **Method**: `POST`

Request (Token Refresh):

- **Headers**:

```
Authorization: Bearer <refreshToken>
```

- **Body**: None

Refresh Token Response:

- **200 OK** (Success):

```
{
  "accessToken": "<string>",
  "accessTokenExpiry": "<datetime>",
  "refreshToken": "<string>",
  "refreshTokenExpiry": "<datetime>"
}
```

- **401 Unauthorized** (Failure): If the refresh token is invalid or expired.

Detailed Authorization Workflow

1. Login: Authorization Process

1. The user sends a `POST` request to `/api/v1/security/login` with their username and password.
2. The system validates the credentials and returns:

- An **access token**: Used for accessing resources.
- A **refresh token**: For generating new access tokens when the current one expires.

2. Securing Requests

1. Add the **access token** to the `Authorization` header of every request to secured resources as follows:

```
Authorization: Bearer <accessToken>
```

2. Provide any additional payload required by the specific endpoint.

3. Refreshing Tokens

1. If the system returns a `401 Unauthorized` error due to an expired access token:

- Send a `POST` request to `/api/v1/security/jwt/refresh-token` with the **refresh token** in the `Authorization` header.

2. Upon success, use the newly issued access token for subsequent requests.

3. If the refresh token is invalid or expired, prompt the user to log in again.

Example Usage Flow

1. Login to Get Tokens

Request:

```
POST /api/v1/security/login
Content-Type: application/json

{
  "login": "exampleUser",
  "password": "password123"
}
```

Response:

```
{
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cC...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cC...",
  "accessTokenExpiry": "2025-01-04T19:16:07.298428Z",
  "refreshTokenExpiry": "2025-01-05T19:11:07.298013Z"
}
```

2. Use Access Token to Fetch a Protected Resource

Request:

```
GET /api/secure/resource
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5c...
```

Response:

```json
{
  "data": "Some protected resource"
}
```

3. Refresh Expired Token

Request:

```
POST /api/v1/security/jwt/refresh-token
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5c...
```

Response:

```json
{
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cC...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cC...",
  "accessTokenExpiry": "2025-01-04T19:16:07.298428Z",
  "refreshTokenExpiry": "2025-01-05T19:11:07.298013Z"
}
```

Error Handling

Common Errors

Error Code	Description
401 Unauthorized	The token is invalid, expired, or missing.
403 Forbidden	The user does not have permission to access the requested resource.

Resolving Errors

1. **401 Unauthorized**:

- Verify that the access token is valid and included in the `Authorization` header.
- If the token is expired, refresh it using the `/api/v1/security/jwt/refresh-token` endpoint.

2. **403 Forbidden**:

- Ensure that the user has sufficient permissions for the resource.

This documentation ensures that frontend developers can effectively authenticate and authorize users, protect sensitive endpoints, and handle common error scenarios when integrating with the system.