NeoFi Event Management API Documentation

This document provides a comprehensive overview of the NeoFi Event Management API, a RESTful API designed for collaborative event management. Built with Flask, SQLite, and JWT authentication, it supports role-based access control, event versioning, conflict detection, and changelog tracking. The API is documented using the OpenAPI 3.0 specification, accessible via an interactive Swagger UI at `/swagger`. Below, we outline the API's endpoints, usage, and evaluation of its documentation quality to ensure it is well-documented and easy to understand.

Overview

The NeoFi Event Management API enables users to:

- **Authenticate**: Register, log in, refresh tokens, and log out.
- **Manage Events**: Create, view, update, and delete events with conflict detection for overlapping events at the same location.
- **Collaborate**: Share events with users, assigning roles (Owner, Editor, Viewer).
- **Track History**: View event versions, roll back to previous versions, and compare changes.
- **Optimize**: Use rate limiting and caching for performance and security.

The API is served at `http://localhost:5000` (or your deployed URL) and includes a Swagger UI at `http://localhost:5000/swagger` for interactive testing.

Endpoints

The API is organized into four main categories: Authentication, Events, Collaboration, and Version History. All endpoints requiring authentication use JWT tokens in the `Authorization: Bearer <token>` header. Below is a summary of key endpoints, their purposes, and example usage.

Authentication

- **Register** (`POST /api/auth/register`)
- **Purpose**: Create a new user account.
- **Request**: JSON with `username`, `email`, and `password`.
- **Response**: User details, access token, refresh token, and token type.
- **Example**:
- `POST /api/auth/register`
- `Content-Type: application/json`
- `{`
- `  "username": "testuser",`
- `  "email": "test@example.com",`
- `  "password": "password123"`

- }

  Response (200):

  ```
  {
    "user": {
      "id": 1,
      "username": "testuser",
      "email": "test@example.com",
      "role": "Viewer"
    },
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "token_type": "bearer"
  }
  ```

- **Login** (`POST /api/auth/login`)
- **Purpose**: Authenticate a user and obtain tokens.
- **Request**: JSON with `username` and `password`.
- **Response**: Access token, refresh token, and token type.
- **Example**:
- `POST /api/auth/login`
- `Content-Type: application/json`
- {
- "username": "testuser",
- "password": "password123"
- }

  Response (200):

  ```
  {
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "token_type": "bearer"
  }
  ```

- **Refresh Token** (`POST /api/auth/refresh`)
- **Purpose**: Generate a new access token using a refresh token.
- **Request**: Requires a valid refresh token in the Authorization header.
- **Response**: New access token and token type.
- **Example**:
- `POST /api/auth/refresh`
- `Authorization: Bearer <refresh_token>`

  Response (200):

  ```
  {
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "token_type": "bearer"
  }
  ```

- **Logout** (`POST /api/auth/logout`)
- **Purpose**: Log out the user (client-side token cleanup recommended).
- **Request**: Requires a valid access token.

- **Response**: Confirmation message.
- **Example**: `POST /api/auth/logout Authorization: Bearer <access_token>` Response (200): `{ "message": "Logged out successfully" }`

Events

- **Create Event** (`POST /api/events`)
- **Purpose**: Create a new event with conflict detection for location overlaps.
- **Request**: JSON with `title`, `description`, `start_time`, `end_time`, `location`, `is_recurring`, and optional `recurrence_pattern`.
- **Response**: Created event details.
- **Example**:
- `POST /api/events`
- `Authorization: Bearer <access_token>`
- `Content-Type: application/json`
- `{`
- `  "title": "Team Meeting",`
- `  "description": "Weekly sync",`
- `  "start_time": "2025-06-01T10:00:00Z",`
- `  "end_time": "2025-06-01T11:00:00Z",`
- `  "location": "Room 101",`
- `  "is_recurring": false`
- `}`

  Response (200):

```
{
  "id": 1,
  "title": "Team Meeting",
  "description": "Weekly sync",
  "start_time": "2025-06-01T10:00:00Z",
  "end_time": "2025-06-01T11:00:00Z",
  "location": "Room 101",
  "is_recurring": false,
  "recurrence_pattern": null,
  "owner_id": 1
}
```

- **List Events** (`GET /api/events`)
- **Purpose**: Retrieve paginated events the user has access to, filterable by date range.
- **Parameters**: `page` (default: 1), `per_page` (default: 10), `start_date`, `end_date` (ISO 8601).
- **Response**: List of events with pagination metadata.
- **Example**:
- `GET /api/events?page=1&per_page=10&start_date=2025-06-01T00:00:00Z&end_date=2025-06-30T23:59:59Z`
- `Authorization: Bearer <access_token>`

  Response (200):

```
{
  "events": [
```

```
    {
      "id": 1,
      "title": "Team Meeting",
      "description": "Weekly sync",
      "start_time": "2025-06-01T10:00:00Z",
      "end_time": "2025-06-01T11:00:00Z",
      "location": "Room 101",
      "is_recurring": false,
      "recurrence_pattern": null,
      "owner_id": 1
    }
  ],
  "total": 1,
  "page": 1,
  "per_page": 10
}
```

- **Get Event** (`GET /api/events/{id}`)
- **Purpose**: Retrieve details of a specific event.
- **Parameters**: `id` (event ID).
- **Response**: Event details.
- **Example**:
- `GET /api/events/1`
- `Authorization: Bearer <access_token>`

  Response (200):

```
{
  "id": 1,
  "title": "Team Meeting",
  "description": "Weekly sync",
  "start_time": "2025-06-01T10:00:00Z",
  "end_time": "2025-06-01T11:00:00Z",
  "location": "Room 101",
  "is_recurring": false,
  "recurrence_pattern": null,
  "owner_id": 1
}
```

- **Update Event** (`PUT /api/events/{id}`)
- **Purpose**: Update an event's details (Owner or Editor only).
- **Parameters**: `id` (event ID).
- **Request**: JSON with updated event fields.
- **Response**: Updated event details.
- **Example**:
- `PUT /api/events/1`
- `Authorization: Bearer <access_token>`
- `Content-Type: application/json`
- `{`
- `  "title": "Updated Meeting",`
- `  "description": "Updated weekly sync",`
- `  "start_time": "2025-06-01T10:00:00Z",`
- `  "end_time": "2025-06-01T11:00:00Z",`
- `  "location": "Room 102",`
- `  "is_recurring": false`

- `}`

  Response (200):

  ```
  {
    "id": 1,
    "title": "Updated Meeting",
    "description": "Updated weekly sync",
    "start_time": "2025-06-01T10:00:00Z",
    "end_time": "2025-06-01T11:00:00Z",
    "location": "Room 102",
    "is_recurring": false,
    "recurrence_pattern": null,
    "owner_id": 1
  }
  ```

- **Delete Event** (`DELETE /api/events/{id}`)
- **Purpose**: Delete an event (Owner only).
- **Parameters**: `id` (event ID).
- **Response**: Confirmation message.
- **Example**:
- `DELETE /api/events/1`
- `Authorization: Bearer <access_token>`

  Response (200):

  ```
  {
    "message": "Event deleted successfully"
  }
  ```

- **Batch Create Events** (`POST /api/events/batch`)
- **Purpose**: Create multiple events, skipping conflicts.
- **Request**: JSON array of event objects.
- **Response**: List of created events.
- **Example**: `POST /api/events/batch Authorization: Bearer <access_token> Content-Type: application/json [{ "title": "Meeting 1", "description": "First meeting", "start_time": "2025-06-01T10:00:00Z", "end_time": "2025-06-01T11:00:00Z", "location": "Room 101", "is_recurring": false }, { "title": "Meeting 2", "description": "Second meeting", "start_time": "2025-06-01T12:00:00Z", "end_time": "2025-06-01T13:00:00Z", "location": "Room 102", "is_recurring": false }]` Response (200): `[{ "id": 1, "title": "Meeting 1", "description": "First meeting", "start_time": "2025-06-01T10:00:00Z", "end_time": "2025-06-01T11:00:00Z", "location": "Room 101", "is_recurring": false, "recurrence_pattern": null, "owner_id": 1 }, { "id": 2, "title": "Meeting 2", "description": "Second meeting", "start_time": "2025-06-01T12:00:00Z", "end_time": "2025-06-01T13:00:00Z", "location": "Room 102", "is_recurring": false, "recurrence_pattern": null, "owner_id": 1 }]`

Collaboration

- **Share Event** (`POST /api/events/{id}/share`)
- **Purpose**: Grant a user access to an event with a role (Owner only).
- **Parameters**: `id` (event ID).

- **Request**: JSON with `user_id` and `role` (Owner, Editor, Viewer).
- **Response**: Permission details.
- **Example**:
- `POST /api/events/1/share`
- `Authorization: Bearer <access_token>`
- `Content-Type: application/json`
- `{`
- `  "user_id": 2,`
- `  "role": "Editor"`
- `}`

  Response (200):

  ```
  {
    "user_id": 2,
    "role": "Editor"
  }
  ```

- **List Permissions** (`GET /api/events/{id}/permissions`)
- **Purpose**: View all user permissions for an event.
- **Parameters**: `id` (event ID).
- **Response**: List of permissions.
- **Example**:
- `GET /api/events/1/permissions`
- `Authorization: Bearer <access_token>`

  Response (200):

  ```
  [
    {
      "user_id": 1,
      "role": "Owner"
    },
    {
      "user_id": 2,
      "role": "Editor"
    }
  ]
  ```

- **Update Permission** (`PUT /api/events/{id}/permissions/{user_id}`)
- **Purpose**: Update a user's role for an event (Owner only).
- **Parameters**: `id` (event ID), `user_id` (user ID).
- **Request**: JSON with `role`.
- **Response**: Updated permission.
- **Example**:
- `PUT /api/events/1/permissions/2`
- `Authorization: Bearer <access_token>`
- `Content-Type: application/json`
- `{`
- `  "role": "Viewer"`
- `}`

  Response (200):

```
{
  "user_id": 2,
  "role": "Viewer"
}
```

- **Delete Permission** (`DELETE /api/events/{id}/permissions/{user_id}`)
- **Purpose**: Revoke a user's access to an event (Owner only).
- **Parameters**: `id` (event ID), `user_id` (user ID).
- **Response**: Confirmation message.
- **Example**: `DELETE /api/events/1/permissions/2 Authorization: Bearer <access_token>` Response (200): `{ "message": "Permission deleted successfully" }`

Version History

- **Get Event Version** (`GET /api/events/{id}/history/{version_id}`)
- **Purpose**: Retrieve a specific version of an event.
- **Parameters**: `id` (event ID), `version_id` (version number).
- **Response**: Version details.
- **Example**:
- `GET /api/events/1/history/1`
- `Authorization: Bearer <access_token>`

  Response (200):

```
{
  "version": 1,
  "title": "Team Meeting",
  "description": "Weekly sync",
  "start_time": "2025-06-01T10:00:00Z",
  "end_time": "2025-06-01T11:00:00Z",
  "location": "Room 101",
  "is_recurring": false,
  "recurrence_pattern": null,
  "modified_by": 1,
  "modified_at": "2025-06-01T09:00:00Z"
}
```

- **Rollback Event** (`POST /api/events/{id}/rollback/{version_id}`)
- **Purpose**: Restore an event to a previous version (Owner only).
- **Parameters**: `id` (event ID), `version_id` (version number).
- **Response**: Updated event details.
- **Example**:
- `POST /api/events/1/rollback/1`
- `Authorization: Bearer <access_token>`

  Response (200):

```
{
  "id": 1,
  "title": "Team Meeting",
  "description": "Weekly sync",
  "start_time": "2025-06-01T10:00:00Z",
  "end_time": "2025-06-01T11:00:00Z",
```

```
  "location": "Room 101",
  "is_recurring": false,
  "recurrence_pattern": null,
  "owner_id": 1
}
```

- **Get Changelog** (`GET /api/events/{id}/changelog`)
- **Purpose**: View all version history for an event.
- **Parameters**: `id` (event ID).
- **Response**: List of version details.
- **Example**:
- `GET /api/events/1/changelog`
- `Authorization: Bearer <access_token>`

  Response (200):

```
[
  {
    "version": 1,
    "title": "Team Meeting",
    "description": "Weekly sync",
    "start_time": "2025-06-01T10:00:00Z",
    "end_time": "2025-06-01T11:00:00Z",
    "location": "Room 101",
    "is_recurring": false,
    "recurrence_pattern": null,
    "modified_by": 1,
    "modified_at": "2025-06-01T09:00:00Z"
  },
  {
    "version": 2,
    "title": "Updated Meeting",
    "description": "Updated weekly sync",
    "start_time": "2025-06-01T10:00:00Z",
    "end_time": "2025-06-01T11:00:00Z",
    "location": "Room 102",
    "is_recurring": false,
    "recurrence_pattern": null,
    "modified_by": 1,
    "modified_at": "2025-06-02T09:00:00Z"
  }
]
```

- **Compare Versions** (`GET /api/events/{id}/diff/{version_id1}/{version_id2}`)
- **Purpose**: Compare two versions of an event to highlight changes.
- **Parameters**: `id` (event ID), `version_id1`, `version_id2` (version numbers).
- **Response**: Differences between versions.
- **Example**: `GET /api/events/1/diff/1/2 Authorization: Bearer <access_token>` Response (200): `{ "diff": { "values_changed": { "root['title']": { "new_value": "Updated Meeting", "old_value": "Team Meeting" }, "root['description']": { "new_value": "Updated weekly sync", "old_value": "Weekly sync" }, "root['location']": { "new_value": "Room 102", "old_value": "Room 101" } } } }`

Error Handling

The API uses standard HTTP status codes and JSON error responses:

- **400**: Bad request (e.g., missing fields, invalid data).

```
{
  "error": "Username and password required"
}
```

- **401**: Unauthorized (e.g., invalid token).

```
{
  "error": "Unauthorized"
}
```

- **403**: Forbidden (e.g., insufficient permissions).

```
{
  "error": "Not authorized to edit"
}
```

- **404**: Not found (e.g., event or user not found).

```
{
  "error": "Event not found"
}
```

- **409**: Conflict (e.g., event scheduling conflict).

```
{
  "error": "Event conflicts with existing event at the same location"
}
```

- **422**: Validation error (e.g., invalid input format).

```
{
  "error": [
    {
      "field": "start_time",
      "message": "Invalid date format"
    }
  ]
}
```

Security

- **Authentication**: JWT tokens with 1-hour access token expiration and 30-day refresh token expiration.
- **Authorization**: Role-based access control (Owner, Editor, Viewer).
- **Rate Limiting**: Applied to sensitive endpoints (e.g., 10 login attempts per minute, 5 event creations per minute).
- **Password Security**: Salted SHA-256 hashing for stored passwords.

Accessing the Swagger UI

The API includes an interactive Swagger UI for testing and exploration:

1. Run the application:

```
docker run -d -p 5000:5000 --name neofi-app <your-dockerhub-username>/neofi-event-app:latest
```

Or, if running locally:

```
python app.py
```

2. Open a browser and navigate to `http://localhost:5000/swagger`.
3. Use the Swagger UI to:

- View endpoint details (parameters, request/response schemas).
- Test endpoints by entering data and tokens.
- See example responses and error codes.

The Swagger UI is automatically generated from the `openapi.yaml` file, ensuring consistency with the API implementation.

Evaluation: Is the API Well-Documented and Easy to Understand?

To assess whether the API documentation is well-documented and easy to understand, we evaluate it against standard criteria:

1. Completeness

- **Coverage**: The documentation covers all endpoints (authentication, events, collaboration, version history) with detailed descriptions of request parameters, request bodies, response formats, and error codes.
- **Schemas**: Clear definitions for data models (e.g., events, users, permissions) with required fields and data types.
- **Examples**: Each endpoint includes example requests and responses, making it easy to understand expected inputs and outputs.
- **Security**: Authentication and authorization requirements are explicitly stated, including token usage and role restrictions.

Assessment: The documentation is complete, addressing all API functionality and edge cases (e.g., conflicts, validation errors).

2. Clarity

- **Descriptions**: Each endpoint has a clear summary and detailed description, explaining its purpose and behavior (e.g., conflict detection, versioning).
- **Language**: Uses concise, non-technical language where possible, with technical terms (e.g., JWT, ISO 8601) explained implicitly through examples.

- **Structure**: Organized into logical sections (authentication, events, etc.), making it easy to navigate.
- **Error Messages**: Standardized error responses with descriptive messages (e.g., “Event conflicts with existing event at the same location”).

Assessment: The documentation is clear, with intuitive explanations and structured organization.

3. Usability

- **Swagger UI**: The interactive Swagger UI allows developers to test endpoints directly, reducing the learning curve. It provides visual schemas, example inputs, and live responses.
- **Examples**: Realistic examples (e.g., creating an event with ISO 8601 timestamps) help developers understand usage without needing to infer formats.
- **Setup Instructions**: The `README.md` (previously provided) complements the API documentation with setup, installation, and Docker instructions, ensuring developers can quickly run the API.
- **Accessibility**: Available via `/swagger` endpoint, requiring no additional tools beyond a browser.

Assessment: The documentation is highly usable, with interactive tools and practical examples.

4. Consistency

- **Response Formats**: Consistent JSON structure across endpoints (e.g., `error` field for errors, `message` for success confirmations).
- **Naming**: Endpoint paths follow REST conventions (e.g., `/api/events/{id}`), and parameter names are intuitive (e.g., `user_id`, `version_id`).
- **Error Handling**: Uniform error codes and messages align with HTTP standards.

Assessment: The documentation maintains consistency in structure, naming, and error handling.

5. Maintainability

- **OpenAPI Specification**: The `openapi.yaml` file is machine-readable and can be updated easily as the API evolves. It’s integrated with the Flask application, ensuring synchronization.
- **Versioning**: The API version (1.0.0) is specified, allowing for future updates without breaking existing clients.
- **Extensibility**: The modular structure supports adding new endpoints or features with minimal changes to the documentation.

Assessment: The documentation is maintainable and future-proof.

Areas for Improvement

While the documentation is robust, minor enhancements could further improve it:

- **Tutorials**: Add a step-by-step guide in the Swagger UI or `README.md` for common workflows (e.g., “Create a user, log in, and schedule an event”).
- **Recurring Events**: Clarify that `recurrence_pattern` is stored but not processed (e.g., requires `python-dateutil` for expansion).
- **Rate Limit Details**: Specify exact rate limits in the documentation (e.g., “10 per minute for `/api/auth/login`”) for transparency.
- **External Links**: Include links to tools (e.g., Postman, JWT.io) for developers unfamiliar with API testing.

Usage Instructions

To use the API:

1. **Run the Application**:

- Locally: `cd C:\Users\Zirmute\Downloads\neofi_event_app\neofi_event_app .\venv\Scripts\activate python app.py`
- With Docker: `docker run -d -p 5000:5000 --name neofi-app <your-dockerhub-username>/neofi-event-app:latest`

1. **Access Swagger UI**:

- Open `http://localhost:5000/swagger`.
- Explore endpoints, test requests, and view responses.

1. **Test with Postman**:

- Import the `openapi.yaml` into Postman to generate a collection.
- Use the examples above to send requests.

1. **Authenticate**:

- Register or log in to obtain an access token.
- Include the token in the `Authorization` header for protected endpoints.

1. **Handle Errors**:

- Check response status codes and `error` fields for issues (e.g., 409 for conflicts).

Conclusion

The NeoFi Event Management API is **well-documented and easy to understand**, thanks to its comprehensive OpenAPI specification, interactive Swagger UI, and clear examples. The documentation is complete, clear, usable, consistent, and

maintainable, meeting the needs of developers integrating with the API. Minor additions (e.g., tutorials, rate limit details) could enhance it further, but it already provides a robust foundation for understanding and using the API effectively.