I'll help you generate comprehensive API documentation for the authentication APIs in the FoodShare application. Let me first explore the codebase to understand the authentication implementation.Let me analyze the authentication-related code in the FoodShare application to document the Authentication APIs properly.

Based on my analysis of the codebase, I can see that the application has a well-structured authentication service that handles login, registration, and logout functionality. The authentication APIs are defined primarily in authService.js.

FoodShare API Documentation - Authentication APIs

1. Authentication APIs

1.1 User Login

Endpoint URL: `/api/auth/login`

HTTP Method: `POST`

Request Headers:

```
Content-Type: application/json
```

Request Body Schema:

```json
{
  "email": "user@example.com",
  "password": "securePassword123"
}
```

Response Body Schema:

```json
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": "u12345",
    "name": "John Doe",
    "email": "user@example.com",
    "role": "donor",
    "status": "Active",
    "createdAt": "2023-11-10T15:30:00Z"
  }
}
```

Authentication Requirements: None (Public API)

Status Codes and Messages:

- `200 OK`: Successful login
- `400 Bad Request`: Invalid input data
- `401 Unauthorized`: Invalid credentials
- `403 Forbidden`: Account disabled or pending verification
- `500 Internal Server Error`: Server error

Description: Authenticates a user with email and password credentials. Returns a JWT token and basic user information upon successful authentication.

1.2 User Registration

Endpoint URL: `/api/auth/register`

HTTP Method: `POST`

Request Headers:

```
Content-Type: application/json
```

Request Body Schema:

```json
{
  "name": "Jane Smith",
  "email": "jane.smith@example.com",
  "password": "securePassword123",
  "role": "beneficiary",
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "state": "CA",
    "zipCode": "12345",
    "country": "USA"
  },
  "phone": "+1234567890"
}
```

Response Body Schema:

```json
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": "u67890",
    "name": "Jane Smith",
```

```
    "email": "jane.smith@example.com",
    "role": "beneficiary",
    "status": "Active",
    "createdAt": "2023-11-15T10:20:30Z"
  }
}
```

Authentication Requirements: None (Public API)

Status Codes and Messages:

- `201 Created`: Successfully registered
- `400 Bad Request`: Invalid input data or validation errors
- `409 Conflict`: Email already registered
- `500 Internal Server Error`: Server error

Description: Registers a new user with the provided information. Returns a JWT token and user data upon successful registration. For NGO accounts, the status will be set to "Pending Verification" until approved by an admin.

1.3 User Logout

Endpoint URL: `/api/auth/logout`

HTTP Method: `POST`

Request Headers:

```
Authorization: Bearer {token}
```

Request Body Schema: Empty

Response Body Schema:

```json
{
  "message": "Logout successful"
}
```

Authentication Requirements: JWT Token Required

Status Codes and Messages:

- `200 OK`: Successfully logged out
- `401 Unauthorized`: Invalid or missing token
- `500 Internal Server Error`: Server error

Description: Invalidates the user's session token on the server side. The client should also remove the token from local storage.

1.4 Token Refresh

Endpoint URL: `/api/auth/refresh-token`

HTTP Method: `POST`

Request Headers:

```
Content-Type: application/json
```

Request Body Schema:

```json
```

{

"token": "current_jwt_token"

}

```

**Response Body Schema**:

```json

{

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"expiresAt": "2023-12-15T10:20:30Z"

}

```

Authentication Requirements: Valid but potentially expired JWT Token

Status Codes and Messages:

- `200 OK`: Successfully refreshed token

- `401 Unauthorized`: Invalid token

- `403 Forbidden`: Refresh not allowed (token revoked)

- `500 Internal Server Error`: Server error

Description: Refreshes an expired or soon-to-expire JWT token without requiring the user to log in again. The client should replace the old token with the new one for subsequent API calls.

1.5 Update User Profile

Endpoint URL: `/api/auth/profile`

HTTP Method: `PUT`

Request Headers:
```

```
Authorization: Bearer {token}
Content-Type: application/json
```

Request Body Schema:

```json
{
  "name": "Jane Updated Smith",
  "address": {
    "street": "456 Oak Avenue",
    "city": "New City",
    "state": "NY",
    "zipCode": "67890",
    "country": "USA"
  },
  "phone": "+10987654321"
}
```

Response Body Schema:

```json
{
  "user": {
    "id": "u67890",
    "name": "Jane Updated Smith",
    "email": "jane.smith@example.com",
    "role": "beneficiary",
    "status": "Active",
    "address": {
      "street": "456 Oak Avenue",
```

```
    "city": "New City",
    "state": "NY",
    "zipCode": "67890",
    "country": "USA"
  },
  "phone": "+10987654321",
  "updatedAt": "2023-11-18T14:25:30Z"
 }
}
```


Authentication Requirements: JWT Token Required

Status Codes and Messages:

- `200 OK`: Successfully updated profile
- `400 Bad Request`: Invalid input data
- `401 Unauthorized`: Invalid or missing token
- `500 Internal Server Error`: Server error

Description: Updates the authenticated user's profile information. Only certain fields can be updated (cannot change email or role through this endpoint).