Authentication and Authorization Middleware Documentation

This documentation covers the implementation of a middleware that manages authentication and authorization in a Node.js application using JWT (JSON Web Tokens). The middleware ensures that protected routes are only accessible to authenticated users and performs permission-based access control for specific resources.

Table of Contents

Dependencies

```
const jwt = require("jsonwebtoken");
const User = require("../models/merchantModel");
const AffiliateUser = require("../models/affiliateUsersModels");
const dotenv = require('dotenv');
```

- `jsonwebtoken`: Used to verify the JWT tokens.
- `User`: The merchant user model.
- `AffiliateUser`: The affiliate user model.
- `dotenv`: Manages environment variables such as the JWT secret key.

---

protect Middleware

This middleware ensures that users accessing specific routes have a valid JWT token. It also handles user role-based access control by verifying user types (e.g., merchant or affiliate) and checking if the token has expired.

Skipped URLs

The following URLs are skipped and do not require authentication:

```
const skippedUrls = [
  "/api/v1/affiliate/register",
  "/api/v1/affiliate/login",
  "/api/v1/wallet/merchant-info"
];
```

Restricted URLs

These URLs are restricted to admin users only:

```
const restrictedUrls = [
  "/admin/users",
  "/admin/products",
  "/admin/orders"
];
```

Token Verification

1. The middleware checks if the JWT token is present in the `x-authorization` header.
2. If a token is present, it verifies the token using `jwt.verify` with the secret key from the environment (`process.env.JWT_SECRET`).
3. If the token is expired, a `401 Unauthorized` response is returned.
4. Based on the `type` field in the token payload, the user is fetched from the database (either from `User` or `AffiliateUser` collections).
5. If the user is not found or the token is invalid, the middleware returns an unauthorized response.

Merchant ID Verification

The middleware optionally verifies if the `merchant_id` in the request matches the `userId` in the token. If there is a mismatch, it returns an `Access denied` message.

```
const requestMerchantId = req.query.merchant_id || req.body.merchant_id ||
req.query.merchantId;
```

Finally, the user's permissions are validated through the `allowIfPermission` function before passing control to the next middleware.

---

admin Middleware

The `admin` middleware ensures that the user has admin privileges before accessing certain routes.

```
exports.admin = (req, res, next) => {
  if (req.user && req.user.isAdmin) {
    next();
```

```
    } else {
      return res.status(401).json({
        status: "fail",
        code: 401,
        message: "Not Authorized, As a Admin",
        data: {},
      });
    }
  };
```

If the user is not an admin, a `401 Unauthorized` error is returned.

---

authenticateToken Middleware

This middleware handles token authentication based on a bearer token passed in the `authorization` header.

```
exports.authenticateToken = (req, res, next) => {
  const authHeader = req.headers.authorization;
  const token = authHeader && authHeader.split(' ')[1];

  if (token == null) {
    return res.status(500).json({
      success: false,
      message: "Bearer token not provided in the header"
    });
  }

  const hardcodedToken = "f8946gntyw84769gt8y869yjh8597";
  if (token === hardcodedToken) {
    next();
  } else {
    return res.status(500).json({
      success: false,
      message: "not authorized"
    });
  }
};
```

- Extracts the token from the `authorization` header.
- Compares the token to a hardcoded value, and if they match, proceeds to the next middleware.
- If the token doesn't match, an unauthorized response is returned.

---

allowIfPermission Middleware

This middleware checks if the authenticated user has the necessary permissions to access the requested URL.

```
async function allowIfPermission(req, res, next) {
  if (!req.user || !req.user.permissions ||
!Array.isArray(req.user.permissions)) {
    return res.status(401).json({
      status: 'fail',
      code: 401,
      message: 'Not Authorized, Access Denied'
    });
  }

  const permissionsRequired = mapUrlToPermissions(req.originalUrl);
  const hasPermission = permissionsRequired.some(permission =>
req.user.permissions.includes(permission));

  if (!hasPermission) {
    return res.status(401).json({
      status: 'fail',
      code: 401,
      message: 'Not Authorized, Access Denied'
    });
  }

  next();
}
```

- Verifies that the user has the required permissions to access the URL.
- Calls `mapUrlToPermissions` to retrieve the permissions associated with the URL.
- If the user lacks the required permissions, a `401 Unauthorized` error is returned.

---

mapUrlToPermissions Function

This function maps URLs to the required permissions for that specific route.

```
function mapUrlToPermissions(url) {
  const mappings = {
    '/api/v1/transaction/': ['manage_transaction', 'all_access'],
    '/api/v1/merchant': ['manage_account', 'all_access'],
    '/api/v1/wallet/add-funds': ['manage_wallet', 'all_access'],
    // Add more mappings as needed
  };

  return Object.keys(mappings).find(key => url.startsWith(key)) ?
mappings[url] : [];
}
```

- Defines mappings between URLs and the permissions required to access those routes.
- Returns an array of permissions based on the URL pattern.

This middleware system is designed to provide flexible role-based access control, allowing you to secure routes in your application with minimal effort.