Here's a comprehensive outline of an \*\*Auth API\*\* using \*\*JWT\*\* for a microservices architecture with \*\*Node.js\*\* and \*\*TypeScript\*\*. This design addresses JWT's downsides, such as lack of revocation and refresh mechanisms.

### \*\*Key Features\*\*

1. \*\*User Authentication (Login)\*\*: Issues a short-lived access token and a long-lived refresh token.

2. \*\*Token Refresh\*\*: Provides a mechanism to refresh access tokens using a refresh token.

3. \*\*Token Revocation\*\*: Adds a blacklist to revoke compromised tokens.

4. \*\*Logout\*\*: Revokes tokens by adding them to a blacklist.

5. \*\*Role-Based Access Control (RBAC)\*\*: Adds scopes/roles to the tokens.

6. \*\*Expiration Handling\*\*: Short-lived access tokens reduce exposure risks.

---

### \*\*Endpoints\*\*

Here’s a breakdown of the endpoints needed:

1. \*\*POST /auth/register\*\*

- Registers a new user.

2. \*\*POST /auth/login\*\*

- Authenticates a user and issues an access token and refresh token.

3. \*\*POST /auth/refresh\*\*

- Issues a new access token using a refresh token.

4. \*\*POST /auth/logout\*\*

- Revokes a refresh token and optionally invalidates access tokens.

5. \*\*GET /auth/profile\*\*

- Returns the authenticated user’s profile (protected route).

6. \*\*POST /auth/revoke\*\*

- Admin endpoint to revoke tokens for a user.

---

### \*\*Implementation\*\*

Here’s an implementation in Node.js/TypeScript:

#### \*\*1. Directory Structure\*\*

```plaintext

src/

├── controllers/

│ ├── auth.controller.ts

├── middlewares/

│ ├── auth.middleware.ts

├── models/

│ ├── user.model.ts

│ ├── tokenBlacklist.model.ts

├── routes/

│ ├── auth.routes.ts

├── services/

│ ├── auth.service.ts

│ ├── token.service.ts

├── utils/

│ ├── jwt.utils.ts

├── app.ts

```

#### \*\*2. Token Configuration\*\*

Use \*\*jsonwebtoken\*\* for token management.

```typescript

// src/utils/jwt.utils.ts

import jwt from 'jsonwebtoken';

const ACCESS\_TOKEN\_SECRET = process.env.ACCESS\_TOKEN\_SECRET || 'your\_access\_secret';

const REFRESH\_TOKEN\_SECRET = process.env.REFRESH\_TOKEN\_SECRET || 'your\_refresh\_secret';

export const generateAccessToken = (userId: string, roles: string[]) => {

return jwt.sign({ userId, roles }, ACCESS\_TOKEN\_SECRET, { expiresIn: '15m' });

};

export const generateRefreshToken = (userId: string) => {

return jwt.sign({ userId }, REFRESH\_TOKEN\_SECRET, { expiresIn: '7d' });

};

export const verifyAccessToken = (token: string) => {

return jwt.verify(token, ACCESS\_TOKEN\_SECRET);

};

export const verifyRefreshToken = (token: string) => {

return jwt.verify(token, REFRESH\_TOKEN\_SECRET);

};

```

#### \*\*3. Auth Service\*\*

Handle user and token logic.

```typescript

// src/services/auth.service.ts

import bcrypt from 'bcrypt';

import { UserModel } from '../models/user.model';

import { TokenBlacklistModel } from '../models/tokenBlacklist.model';

import { generateAccessToken, generateRefreshToken, verifyRefreshToken } from '../utils/jwt.utils';

export const registerUser = async (email: string, password: string) => {

const hashedPassword = await bcrypt.hash(password, 10);

const newUser = await UserModel.create({ email, password: hashedPassword });

return newUser;

};

export const loginUser = async (email: string, password: string) => {

const user = await UserModel.findOne({ email });

if (!user || !(await bcrypt.compare(password, user.password))) {

throw new Error('Invalid credentials');

}

const accessToken = generateAccessToken(user.id, user.roles);

const refreshToken = generateRefreshToken(user.id);

return { accessToken, refreshToken };

};

export const refreshToken = async (token: string) => {

const payload = verifyRefreshToken(token);

const blacklisted = await TokenBlacklistModel.findOne({ token });

if (blacklisted) {

throw new Error('Token is revoked');

}

const newAccessToken = generateAccessToken(payload.userId, payload.roles);

return newAccessToken;

};

export const logoutUser = async (refreshToken: string) => {

await TokenBlacklistModel.create({ token: refreshToken });

};

```

#### \*\*4. Middleware\*\*

Protect routes and check permissions.

```typescript

// src/middlewares/auth.middleware.ts

import { Request, Response, NextFunction } from 'express';

import { verifyAccessToken } from '../utils/jwt.utils';

export const authenticate = (req: Request, res: Response, next: NextFunction) => {

const token = req.headers.authorization?.split(' ')[1];

if (!token) return res.status(401).json({ message: 'Unauthorized' });

try {

const payload = verifyAccessToken(token);

req.user = payload;

next();

} catch (err) {

return res.status(403).json({ message: 'Invalid or expired token' });

}

};

```

#### \*\*5. Routes\*\*

Wire up the API endpoints.

```typescript

// src/routes/auth.routes.ts

import express from 'express';

import \* as AuthController from '../controllers/auth.controller';

import { authenticate } from '../middlewares/auth.middleware';

const router = express.Router();

router.post('/register', AuthController.register);

router.post('/login', AuthController.login);

router.post('/refresh', AuthController.refresh);

router.post('/logout', authenticate, AuthController.logout);

router.get('/profile', authenticate, AuthController.profile);

export default router;

```

#### \*\*6. Auth Controller\*\*

Define endpoint logic.

```typescript

// src/controllers/auth.controller.ts

import { Request, Response } from 'express';

import \* as AuthService from '../services/auth.service';

export const register = async (req: Request, res: Response) => {

const { email, password } = req.body;

const user = await AuthService.registerUser(email, password);

res.status(201).json(user);

};

export const login = async (req: Request, res: Response) => {

const { email, password } = req.body;

const tokens = await AuthService.loginUser(email, password);

res.json(tokens);

};

export const refresh = async (req: Request, res: Response) => {

const { refreshToken } = req.body;

const accessToken = await AuthService.refreshToken(refreshToken);

res.json({ accessToken });

};

export const logout = async (req: Request, res: Response) => {

const { refreshToken } = req.body;

await AuthService.logoutUser(refreshToken);

res.status(204).send();

};

export const profile = (req: Request, res: Response) => {

res.json({ user: req.user });

};

```

#### \*\*7. Blacklist Model\*\*

Handle token revocation.

```typescript

// src/models/tokenBlacklist.model.ts

import mongoose from 'mongoose';

const tokenBlacklistSchema = new mongoose.Schema({

token: { type: String, required: true },

createdAt: { type: Date, default: Date.now, expires: '7d' }, // Expire in 7 days

});

export const TokenBlacklistModel = mongoose.model('TokenBlacklist', tokenBlacklistSchema);

```

---

### \*\*Fixing JWT Downsides\*\*

1. \*\*Token Revocation\*\*: Use the `TokenBlacklistModel` to revoke refresh tokens.

2. \*\*Short Token Lifetimes\*\*: Access tokens expire after 15 minutes.

3. \*\*Refresh Mechanism\*\*: Refresh tokens allow secure re-authentication.

4. \*\*Role-Based Access\*\*: Include roles in the JWT payload and check them in the middleware.

5. \*\*Secure Storage\*\*: Store tokens in \*\*HTTP-only cookies\*\* or secure storage mechanisms (not in localStorage).

This setup balances performance and security for a microservices architecture. Let me know if you'd like help with further customization or deployment!