Experiment 7.2 JWT Authentication for Secure Banking API Endpoints

Aim: To implement JWT-based authentication and role-based authorization for secure banking API endpoints using Express.js.

Theory:

1. JWT (JSON Web Token) securely transmits claims between parties and is commonly used for stateless authentication.
2. Tokens contain a payload, a signature, and optionally header metadata; the signature ensures integrity.
3. Expiration and secret keys are crucial to reduce token theft and replay attacks.
4. Role-based authorization enforces access control for endpoints like account details, transfers, and admin operations.
5. Secure banking APIs should protect sensitive operations (e.g., transfers) with authentication, validation, and logging.
6. HTTPS, input validation, rate limiting, and secure storage of secrets complement JWT for production-grade security.

Code Implementation:

```
const express=require('express');
const jwt=require('jsonwebtoken');
const bodyParser=require('body-parser');

const SECRET='replace-with-secure-secret';
const TOKEN_EXPIRY='1h';

function logger(req,res,next){
  const start=Date.now();
  res.on('finish',()=>{
    const duration=Date.now()-start;
    console.log(`${req.ip} ${req.method} ${req.originalUrl} ${res.statusCode} ${duration}ms`);
  });
  next();
}

function authenticateToken(req,res,next){
  const auth=req.headers['authorization'];
  if(!auth||!auth.startsWith('Bearer ')) return res.status(401).json({error:'Unauthorized'});
  const token=auth.slice(7);
  jwt.verify(token,SECRET,(err,decoded)=>{
    if(err) return res.status(403).json({error:'Forbidden'});
    req.user=decoded;
    next();
  });
}

function authorizeRole(...allowedRoles){
  return (req,res,next)=>{
    const role=req.user&&req.user.role;
    if(!role||!allowedRoles.includes(role)) return res.status(403).json({error:'Forbidden'});
    next();
  };
}
```

```
const app=express();
app.use(bodyParser.json());
app.use(logger);

const users=[{id:1,username:'alice',password:'password',role:'user',accountBalance:5000},{id:2,usernam

app.post('/login',(req,res)=>{
  const {username,password}=req.body;
  const user=users.find(u=>u.username===username&&u.password===password);
  if(!user) return res.status(400).json({error:'Invalid credentials'});
  const payload={id:user.id,username:user.username,role:user.role};
  const token=jwt.sign(payload,SECRET,{expiresIn:TOKEN_EXPIRY});
  res.json({token});
});

app.get('/accounts/:id',authenticateToken,(req,res)=>{
  const id=Number(req.params.id);
  const user=users.find(u=>u.id===id);
  if(!user) return res.status(404).json({error:'Not found'});
  if(req.user.id!==id&&req.user.role!=='admin') return res.status(403).json({error:'Forbidden'});
  res.json({id:user.id,username:user.username,accountBalance:user.accountBalance});
});

app.post('/transfer',authenticateToken,(req,res)=>{
  const {from,to,amount}=req.body;
  const sender=users.find(u=>u.id===from);
  const receiver=users.find(u=>u.id===to);
  if(!sender||!receiver) return res.status(404).json({error:'Account not found'});
  if(req.user.id!==from) return res.status(403).json({error:'Forbidden'});
  const amt=Number(amount);
  if(isNaN(amt)||amt<=0) return res.status(400).json({error:'Invalid amount'});
  if(sender.accountBalance<amt) return res.status(400).json({error:'Insufficient funds'});
  sender.accountBalance-=amt;
  receiver.accountBalance+=amt;
  res.json({message:'Transfer successful',from:from,to:to,amount:amt});
});

app.get('/admin/overview',authenticateToken,authorizeRole('admin'),(req,res)=>{
  const overview=users.map(u=>({id:u.id,username:u.username,accountBalance:u.accountBalance}));
  res.json({overview});
});

const port=process.env.PORT||3001;
app.listen(port,()=>console.log(`Server running on port ${port}`));
```