JWT Authentication in FastAPI & Node.js

1 . JWT Overview

- JWT (JSON Web Token) is used for secure information exchange.
- Structure: `<header>.<payload>.<signature>`
- Header: algorithm & type
- Payload: claims (user info, exp)
- Signature: ensures token integrity
- Stateless authentication: no server-side session required

2 . FastAPI JWT (JSON login)

Dependencies

```
pip install fastapi uvicorn python-jose passlib[bcrypt] python-dotenv
```

User Model & Password Hashing

```python
from pydantic import BaseModel
from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

class UserLogin(BaseModel):
    username: str
    password: str

def hash_password(password: str):
    return pwd_context.hash(password)

def verify_password(plain_password, hashed_password):
    return pwd_context.verify(plain_password, hashed_password)
```

JWT Token Utilities

```python
from datetime import datetime, timedelta
from jose import jwt, JWTError

SECRET_KEY = "YOUR_SECRET_KEY"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30
```

```
def create_access_token(data: dict):
    to_encode = data.copy()
    expire = datetime.utcnow() +
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

def verify_token(token: str):
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        return payload.get("sub")
    except JWTError:
        return None
```

Login Endpoint (JSON)

```
from fastapi import FastAPI, HTTPException, Body, status

app = FastAPI()

fake_db = {
    "shiv": hash_password("mypassword")
}

@app.post("/login")
def login(user: UserLogin = Body(...)):
    hashed_password = fake_db.get(user.username)
    if not hashed_password or not verify_password(user.password,
hashed_password):
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Incorrect username or password")
    access_token = create_access_token(data={"sub": user.username})
    return {"access_token": access_token, "token_type": "bearer"}
```

Note abou `Body(...)`**:** `Body(...)` tells FastAPI to parse the incoming request body as JSON validate it using `UserLogin` Pydantic model. `...` he indicates that the body is required. - For simple Pydantic models, **ca**you **om** `Body(...)`, and FastAPI will still parse JSON from the body default. - Us `Body(...)` allows you to add extra met `example` g `description`) for better API docs.

Manual Token Extraction from Header

```
from fastapi import Request, Depends

def get_current_user(request: Request):
    auth_header = request.headers.get("Authorization")
    if not auth_header:
        raise HTTPException(status_code=401, detail="Authorization header
missing")
```

```
    try:
        scheme, token = auth_header.split()  # splits on whitespace
        if scheme.lower() != "bearer":
            raise HTTPException(status_code=401, detail="Invalid
authentication scheme")
    except ValueError:
        raise HTTPException(status_code=401, detail="Invalid Authorization
header format")
    username = verify_token(token)
    if not username:
        raise HTTPException(status_code=401, detail="Invalid token")
    return username

@app.get("/me")
def read_me(current_user: str = Depends(get_current_user)):
    return {"user": current_user}
```

Notes: `.split()` without arguments splits on any whitespace. - This allows flexibility if extra tabs are present. - Works like OAuth2PasswordBearer but gives full control.

---

3. Node.js JWT Authentication

Dependencies

```
npm install express jsonwebtoken bcryptjs
```

Express App Setup

```
const express = require("express");
const jwt = require("jsonwebtoken");
const bcrypt = require("bcryptjs");

const app = express();
app.use(express.json());

const SECRET_KEY = "YOUR_SECRET_KEY";

const users = {
  shiv: bcrypt.hashSync("mypassword", 10)
};
```

Login Endpoint (JSON)

```
app.post("/login", (req, res) => {
  const { username, password } = req.body;
  const hashedPassword = users[username];
  if (!hashedPassword || !bcrypt.compareSync(password, hashedPassword)) {
    return res.status(401).json({ error: "Invalid username or password" });
  }
  const token = jwt.sign({ sub: username }, SECRET_KEY, { expiresIn:
"30m" });
  res.json({ access_token: token, token_type: "bearer" });
});
```

Manual Token Middleware

```
function authMiddleware(req, res, next) {
  const authHeader = req.headers["authorization"];
  if (!authHeader) return res.status(401).json({ error:
"Authorization header missing" });

  const [scheme, token] = authHeader.split(" ");
  if (scheme.toLowerCase() !== "bearer" || !token) {
    return res.status(401).json({ error: "Invalid authorization format" });
  }

  try {
    const payload = jwt.verify(token, SECRET_KEY);
    req.user = payload.sub;
    next();
  } catch {
    return res.status(401).json({ error: "Invalid token" });
  }
}

app.get("/me", authMiddleware, (req, res) => {
  res.json({ user: req.user });
});
```

Notes: - Manual extraction allows custom schemes and `req.user` ol. is - available in all protected routes. - `.split(" ")` requires exactly one space, may trim if needed.

---

4 . Key Takeaways

- JWTs are stateless, secure, and scalable.
- Can use FastAPI or Node.js with manual header extraction.
- JSON-based login is simpler for SPA/mobile apps.
- Always hash passwords and use HTTPS in production.
- `.split()` in Python is more flexible; in Node.js, `.split(" ")` is sufficient.
- Protected routes read token from header and verify it.
- Optional enhancements: refresh tokens, roles, token blacklist.