Day 10: Authentication & Network Protocols

This day focuses on **identity, security, and network-layer fundamentals** that are mandatory for production-grade web applications and frequently tested in backend/system-design interviews.

> Constraint respected: **No Docker usage**. All integrations are shown for direct/local or cloud-managed services.

---

1. Authentication Overview (Why This Matters)

Modern systems separate concerns:

- **Authentication** → Who are you?
- **Authorization** → What are you allowed to do?

Industry-standard approach: - Use **external Identity Providers (IdP)** like Auth0 / Firebase - Backend validates **JWT tokens** - Roles & permissions enforced at API level

---

2. Auth0 Integration (RBAC Enabled)

Auth0 Architecture

```
Client → Auth0 (Login)
       → JWT Access Token
       → Backend API (Role checks)
```

Auth0 Setup Steps

1. Create **Auth0 Application (SPA / Regular Web App)**
2. Create **API** (Audience)
3. Enable **RBAC** and "Add Permissions in the Access Token"

Example Roles & Permissions

- Roles: `admin`, `user`
- Permissions: `read:data`, `write:data`

Backend JWT Verification (Python / FastAPI)

```python
from fastapi import Depends, HTTPException
from jose import jwt

AUTH0_DOMAIN = "your-domain.auth0.com"
API_AUDIENCE = "your-api-identifier"
ALGORITHMS = ["RS256"]


def verify_token(token: str):
    try:
        payload = jwt.decode(
            token,
            key=PUBLIC_KEY,
            algorithms=ALGORITHMS,
            audience=API_AUDIENCE,
            issuer=f"https://{AUTH0_DOMAIN}/"
        )
        return payload
    except Exception:
        raise HTTPException(status_code=401, detail="Invalid token")
```

Role-Based Access Control

```python
def require_role(role: str):
    def checker(payload=Depends(verify_token)):
        if role not in payload.get("permissions", []):
            raise HTTPException(status_code=403)
    return checker
```

✔️Checklist: Auth0 RBAC working

---

3. Firebase Authentication

Supported Providers

- Email / Password
- Google OAuth
- GitHub OAuth

Firebase Setup

1. Create Firebase project

2. Enable providers in **Authentication → Sign-in methods**
3. Download service account JSON

Verify Firebase ID Token (Backend)

```
import firebase_admin
from firebase_admin import auth, credentials

cred = credentials.Certificate("serviceAccount.json")
firebase_admin.initialize_app(cred)

def verify_firebase_token(token: str):
    return auth.verify_id_token(token)
```

✔️Checklist: Firebase Auth (Email, Google, GitHub)

---

4. Email Protocols – SMTP & IMAP

Protocol Roles

Protocol	Purpose
SMTP	Sending emails
IMAP	Reading emails

---

5. SMTP – Email Sending (HTML Templates)

Python SMTP Example

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

msg = MIMEMultipart("alternative")
msg["Subject"] = "Welcome"
msg["From"] = "no-reply@app.com"
msg["To"] = "user@example.com"

html = """
<h2>Welcome!</h2>
```

```
<p>Your account is ready.</p>
"""

msg.attach(MIMEText(html, "html"))

with smtplib.SMTP_SSL("smtp.gmail.com", 465) as server:
    server.login("email@gmail.com", "APP_PASSWORD")
    server.send_message(msg)
```

✔️Checklist: Email sending with HTML templates

---

6. IMAP – Email Reading

```
import imaplib
import email

mail = imaplib.IMAP4_SSL("imap.gmail.com")
mail.login("email@gmail.com", "APP_PASSWORD")
mail.select("inbox")

status, messages = mail.search(None, "ALL")
for num in messages[0].split():
    _, data = mail.fetch(num, "(RFC822)")
    msg = email.message_from_bytes(data[0][1])
    print(msg["Subject"])
```

---

7. SSL / TLS, HTTPS, SSH – Core Concepts

SSL / TLS

- Encrypts data in transit
- Uses public/private key cryptography
- Prevents MITM attacks

HTTPS

- HTTP + TLS
- Certificates issued by trusted CAs

SSH

- Secure remote server access
- Key-based authentication

8. SSL Certificate Installation (HTTPS)

Using Let’s Encrypt (Conceptual Steps)

1. Obtain certificate
2. Configure web server (Nginx / Apache)
3. Force HTTPS

Example Nginx Snippet

```
server {
  listen 443 ssl;
  ssl_certificate /etc/ssl/fullchain.pem;
  ssl_certificate_key /etc/ssl/privkey.pem;
}
```

✔️Checklist: HTTPS enabled

---

9. CORS (Cross-Origin Resource Sharing)

Why CORS Exists

- Browser security policy
- Prevents unauthorized cross-origin calls

CORS Configuration (FastAPI)

```
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=["https://myfrontend.com"],
    allow_credentials=True,
    allow_methods=["GET", "POST"],
    allow_headers=["Authorization", "Content-Type"],
)
```

✔️Checklist: CORS whitelisted origins

---

10. Web Security Best Practices

- Always use HTTPS
- Short-lived access tokens
- Rotate secrets
- Secure cookies (`HttpOnly`, `Secure`)
- Validate JWT audience & issuer
- Rate limit auth endpoints

Day 10 Completion Status

- ☑️Auth0 RBAC integration
- ☑️Firebase Auth providers enabled
- ☑️SMTP email sending (HTML)
- ☑️SSL/TLS & HTTPS configured
- ☑️CORS with whitelisted origins

Key Interview Insight

> Authentication is not a feature — it is **infrastructure**. Poor auth design compromises the entire system.

Next logical step: **Day 11 – Observability, Logging & Monitoring** (Prometheus, OpenTelemetry, distributed tracing)

Or integrate **Auth + Caching + DB** into one secure backend.