

# JEGYZŐKÖNYV

Webes adatkezelő környezetek

Féléves feladat

Telefon/Laptop szervíz

**Készítette:** Bodó Tamás

**Neptunkód:** HDK6NX

**Dátum:** 2025. november

Miskolc, 2025

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>2</b>
<b>2. Feladat leírása</b>	<b>2</b>
<b>3. Alapvető adatszerkezeti modellek és fileok</b>	<b>2</b>
3.1. Az adatbázis ER modell tervezése . . . . .	2
3.2. Az adatbázis konvertálása XDM modellre . . . . .	4
3.3. Az XDM modell alapján XML dokumentum készítése . . . . .	5
3.4. Az XML dokumentum alapján XMLSchema készítése . . . . .	6
<b>4. Java adatszerkezet megvalósítása DOM API segítségével</b>	<b>7</b>
4.1. Adatolvasás . . . . .	7
4.2. Adat-lekérdezés . . . . .	8
4.3. Adatmódosítás . . . . .	9

# **1. Bevezetés**

A feladat célja az XML technológiák alkalmazásának gyakorlati bemutatása egy valós, jól strukturált adatmodellen keresztül. A beadandó témája egy laptop- és telefon szerviz adatainak leírása, modellezése és feldolgozása. A projekt két nagy részre tagolódik: az első részben az adatok modellezése és az XML, illetve XSD állományok elkészítése történik, míg a második részben ezeknek a fájloknak a feldolgozása valósul meg Java nyelven, DOM technológiával. A feladat célja, hogy a hallgató elsajátítsa az XML dokumentumok szerkezetének megtervezését, validálását, és ezek programozott feldolgozását. A projekt során létrejött fájlok (ER modell, XDM modell, XML dokumentum, XSD séma, DOM feldolgozó Java osztályok) együttesen mutatják be az XML technológia teljes életciklusát, az adattervezéstől egészen a feldolgozásig.

# **2. Feladat leírása**

Feladatnak egy telefon- és laptopszerviz működéséhez kapcsolódó adatnyilvántartó rendszer megtervezését és megvalósítását választottam. A rendszer célja, hogy átlátható formában kezelje a javítási folyamatok során keletkező adatokat, beleértve az ügyfelek eszközeit, a hibabejelentéseket, a szerelési lépéseket és a felhasznált alkatrészeket. Ezek az információk jól strukturálhatóak, ezért alkalmasak XML-alapú adatmodell kialakítására.

A munkát azzal kezdtem, hogy felépítettem az adatbázist leíró ER-modellt és ennek XDM megfelelőjét. Ezek a diagramok adták az alapját a végleges XML dokumentumnak, amelyet ehhez készített XML séma biztosít, meghatározva az elemek felépítését, kapcsolatait és a hivatkozások érvényességi szabályait. Az elkészült XML állományt validációval ellenőriztem, hogy megfelel-e a séma által előírt szerkezetnek.

A gyakorlati részben a DOM API-t használtam az adatok programozott kezelésére. Betölтtem az XML dokumentumot, a csomópontokból kiolvastam a szükséges információkat, bizonyos adatokat módosítottam, majd az így frissített adatstruktúrát egy új fájlban mentettem el. A feladat ezzel lefedi egy teljes adatmodell kialakítását és annak későbbi, programozott feldolgozását is, bemutatva, hogyan lehet a szervizben keletkező adatokat egységes és kezelhető formában tárolni.

# **3. Alapvető adatszerkezeti modellek és fileok**

## **3.1. Az adatbázis ER modell tervezése**

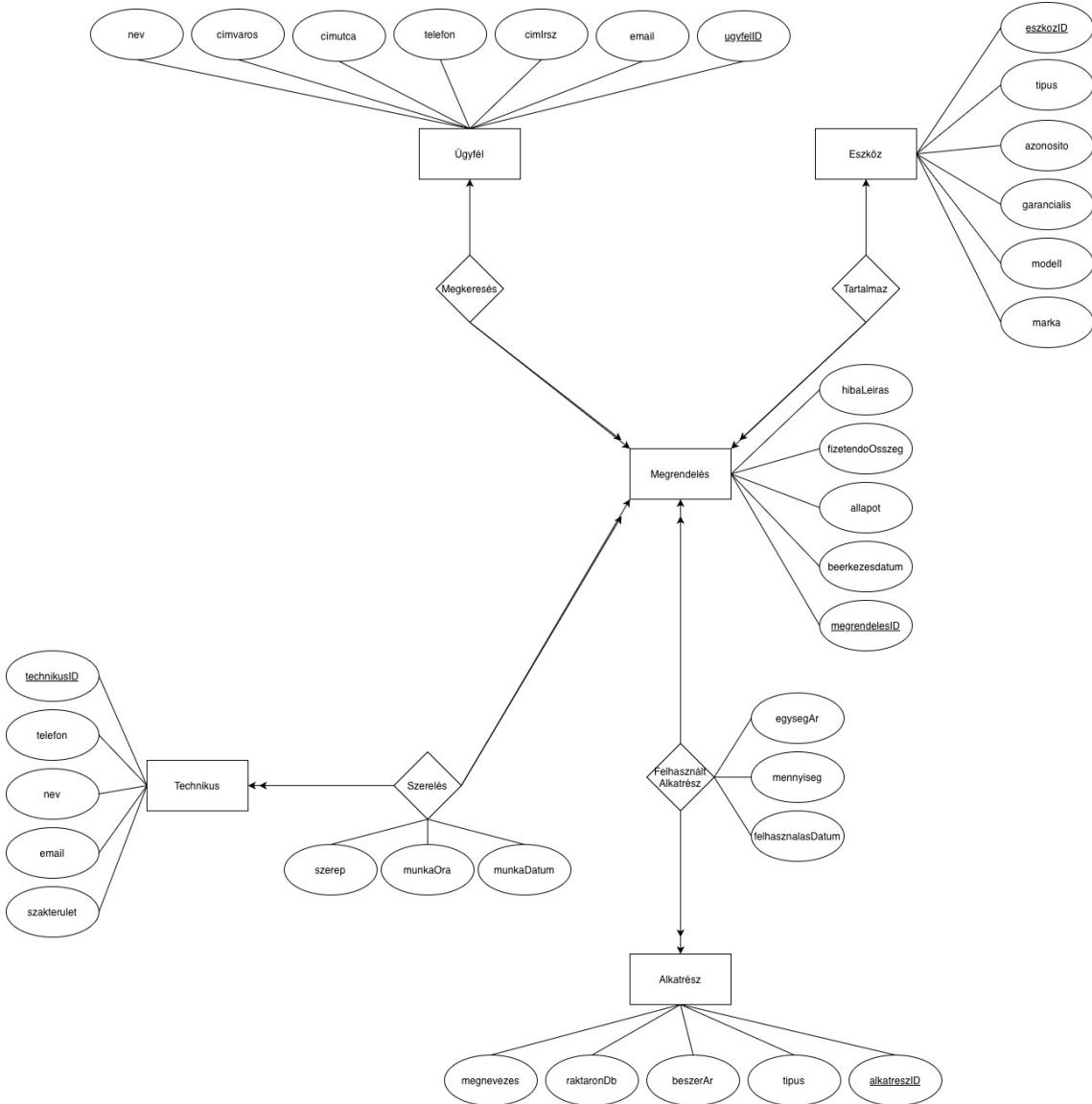
A rendszer adatmodelljének összeállításakor a javítási folyamatot helyeztem a középpontba, ezért a Megrendelés került a struktúra központi szerepébe. Ehhez az egyedhez

kapcsolódik a legtöbb további információ, hiszen a javítás minden lépése – az ügyfélről kezdve az eszközön át egészen a szerelésig és a felhasznált alkatrészekig – ehhez kötődik. A Megrendelés és az Eszköz között egyértelmű, 1:N típusú kapcsolat áll fenn, mivel egy eszközhöz a használat során több hibabejelentés is tartozhat. Az Eszközök pedig az Ügyfelekhez kötődnek, ahol szintén 1:N viszony jelenik meg: egy ügyfél több telefont vagy laptopot is bevizsgáltathat.

A javítás folyamatát részletesebbé teszi a Szerelés egyed bevezetése, amely a Megrendelés és a Technikus közé ékelődik be. Így a folyamat nem csupán egy hivatkozás a szerelőre, hanem külön tárolja a munkavégzéshez kapcsolódó adatokat is, például a szerepkört, a munkára fordított időt és a dátumot. Ez lehetővé teszi, hogy később akár statisztikák vagy teljesítményelemzések is készülhessenek.

A felhasznált alkatrészek kezelése N:M kapcsolatot igényel a Megrendelés és az Alkatrész között, mivel egy javításhoz több különböző elem is felhasználható, és egy alkatrész-típus számos javításban előfordulhat. Ezt a kapcsolati szerepet a felhasznált alkatrész egyed tölti be, amely önálló tulajdonságokkal is rendelkezik, például a felhasznált mennyiséggel, az egységárral és a felhasználás dátumával.

A modell kialakításánál arra törekedtem, hogy a szerkezet részletes legyen, mégse váljon indokolatlanul túlterhelteké. A fő egyedek tulajdonságai így kellően informatívak, ugyanakkor nem tartalmaznak felesleges mezőket, így a feladat követelményeinek minden szerkezeti, minden tartalmi szempontból megfelelően alakul a modell.



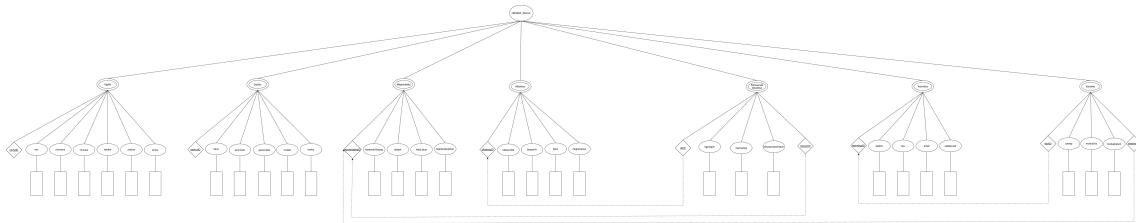
1. ábra. A szervíz ER modellje

### 3.2. Az adatbázis konvertálása XDM modellre

Az ER modell elkészítése után a következő lépés az volt, hogy az adatstruktúrát XDM formára alakítsam át. Ennek során a hangsúly az entitások közötti relációkról a dokumentum alapú, hierarchikus felépítésre helyeződött át. A modellhez egy központi, mesterséges gyökérelemet hoztam létre, amely a teljes telefon- és laptopszerviz nyilvántartást fogja össze, és amely alatt minden fő elem önálló gyermekként jelenik meg. Így külön elemként szerepel az ügyfél, az eszköz, a megrendelés, a technikus, az alkatrész, valamint a szerelések és a felhasznált alkatrészek listája.

Az XDM struktúrában a kulcsértékek attribútumként jelennek meg, míg a normál tulaj-

donságok külön elemeket alkotnak. A kapcsolatok megjelenítése is eltér az ER modellben megszokottól: itt nem gráf jellegű összekötések szerepelnek, hanem a hivatkozások idegen kulcsra mutató attribútumok formájában válnak láthatóvá. A teljes modell ezáltal jól követi az XML dokumentum működési logikáját, mivel a hangsúly a hierarchián, a gyermek-szülő viszonyokon és az egyes rekordok közötti referenciaértékeken van.



2. ábra. A szervíz XMD modellje

### 3.3. Az XDM modell alapján XML dokumentum készítése

Az XML állomány szerkezete a korábban elkészített XDM modell hierarchiáját követi. A dokumentum egyetlen gyökérelemmel indul, amely a szerviz teljes adatállományát fogja össze, és alatta jelennek meg a különálló egységek – például az ügyfelek, az eszközök, a megrendelések, a technikusok és az alkatrészek. Ezek minden önálló elemként szerepelnek, és egyedi azonosítót kapnak attribútum formájában. Az a struktúra, amely az ER modellben még kapcsolatokkal volt leírva, itt referenciaértékekben jelenik meg: az idegen kulcsok azonosítói egy-egy attribútumban jelennek meg, így teremtenek kapcsolatot az elemek között.

A dokumentumban néhány tulajdonság ismétlődhet vagy több elemben is megjelenhet, azonban az egyértelműség érdekében minden adat külön elemet kap. A felépítés lényegében azt mutatja meg, hogyan jelennek meg az XML-ben a rendszer főbb adatsorai. Az alábbi példarészlet bemutatja, hogyan tárolódik egy szervizbe érkezett eszköz adata:

Listing 1. Adatszerkezet-részlet egy eszköz bejegyzéséből

```

1 <eszköz eszközID="E001" ügyfelID="U001">
2   <tipus>telefon</tipus>
3   <marka>Samsung</marka>
4   <modell>Galaxy S22</modell>
5   <azonosito>IMEI-99887766</azonosito>
6   <garancialis>igen</garancialis> </eszköz>
```

Ebben a szakaszban jól látható, hogy az eszközhez tartozó adatokat külön gyermekelemekben tárolja a dokumentum, míg a hivatkozás az ügyfélre attribútumként kerül

megadásra.

### 3.4. Az XML dokumentum alapján XMLSchema készítése

Az XSD séma az elkészült XML struktúrájához igazodik, és meghatározza a dokumentumban megengedett elemeket, azok sorrendjét, adattípusait és a köztük lévő kapcsolatokat. A séma három meghatározó funkcióra épül: az elsődleges kulcsok deklarálására, az idegen kulcs hivatkozások létrehozására, valamint az egyes összetett elemekhez tartozó komplex típusok definiálására.

Az elsődleges kulcsokat a dokumentumban attribútumként jelenítettem meg, és az XSD-ben ezekhez külön kulcsdefiníció tartozik. Az idegen kulcsokra hivatkozó attribútumok esetében keyref elemek biztosítják, hogy a hivatkozott azonosítóval valóban létezzen a megfelelő elemben. A komplex típusokban kerül meghatározásra, hogy az adott elem milyen alárendelt elemeket tartalmaz, milyen adattípusokkal rendelkezhetnek, illetve milyen attribútumokkal bővülnek.

Az alábbi részlet például azt szemlélteti, hogyan épül fel egy eszköztípus leírása a sémában:

Listing 2. Az XSD séma részlete

```
1 <xs:complexType name="EszkozType">
2   [xs:sequence](xs:sequence)
3     <xs:element name="tipus" type="xs:string"/>
4     <xs:element name="marka" type="xs:string"/>
5     <xs:element name="modell" type="xs:string"/>
6     <xs:element name="azonosito" type="xs:string"/>
7     <xs:element name="garancialis" type="xs:string"/>
8   </xs:sequence>
9   <xs:attribute name="eszközID" type="xs:ID" use="required"/>
10  <xs:attribute name="ugyfelID" type="xs:IDREF" use="required"/>
11 </xs:complexType>
```

Ez a szerkezet jól mutatja az XSD egyik fő szerepét: pontosan meghatározza, milyen adat kerülhet egy adott elembe, hogyan kapcsolódik más adatrészkekhez, és milyen formátumban kell megjelennie az XML dokumentumban.

## 4. Java adatszerkezet megvalósítása DOM API segítségével

### 4.1. Adatolvasás

A Java program célja, hogy a szerviznyilvántartás XML állományából a DOM API segítségével fastruktúraként betöltsse az adatokat, majd ezeket Java objektum-szerűen bejárhatóvá és kiírhatóvá tegye. A feldolgozás az org.w3c.dom csomag osztályaira épül: a DocumentBuilderFactory és a DocumentBuilder felel az XML dokumentum beolvasásáért, a Document pedig a teljes struktúrát képviseli.

Listing 3. DOM dokumentum létrehozása Java-ban

```
1 File xmlFile = new File("HDK6NX_XML.xml");
2
3 DocumentBuilderFactory factory = DocumentBuilderFactory.
4     newInstance();
5 DocumentBuilder dBuilder = factory.newDocumentBuilder();
6
7 Document doc = dBuilder.parse(xmlFile);
doc.getDocumentElement().normalize();
```

Ez a kód részlet felel azért, hogy a nyers XML-ből egy DOM Document objektum jöjjön létre, amelyen keresztül a továbbiakban az összes elem, attribútum és szöveges tartalom elérhető. A normalize() hívás egységesíti a whitespace-eket, így a fában való bejárás kiszámíthatóbbá válik.

Az adatbeolvasás során a HDK6NXDomRead osztály végigiterál az <ugyfel> elemeken, majd az egyes ügyfelekhez tartozó <eszköz> és <megrendeles> elemeket is feldolgozza, beleértve a szerelési adatokat és a felhasznált alkatrészeket is. A program minden szinten kiolvassa az attribútumokat és a gyermekelemek szövegét, és jól olvasható formában jelezníti meg a konzolon.

Listing 4. Ügyfelek, eszközök és megrendelések bejárása

```
1 NodeList ugyfelLista = doc.getElementsByTagName("ugyfel");  
2  
3 for (int i = 0; i < ugyfelLista.getLength(); i++)  
4 {  
5     Element ugyfelElem = (Element) ugyfelLista.item(i);  
6     String ugyfelID = ugyfelElem.getAttribute("ugyfelID");  
7     String nev      = gettextContent(ugyfelElem, "nev");  
8     String email    = gettextContent(ugyfelElem, "email");  
9     String telefon  = gettextContent(ugyfelElem, "telefon");  
10  
11 // ... eszközök, megrendelesek, szerelesek,  
12 // felhasznált_alkatresz feldolgozása ...  
13 }
```

A fenti mintában jól látszik a DOM bejárás menete: először az ügyfél azonosítója és alapadatai kerülnek kiolvasásra, majd az ügyfélhez tartozó eszközök, azokhoz kapcsolódó megrendelések, végül pedig az egyes megrendelések szerelési és alkatrész adatai. A `gettextContent` segédfüggvény gondoskodik arról, hogy egy adott elem megadott nevű gyermekének szöveges tartalma egyszerűen lekérdezhető legyen, így elkerülhetők a felesleges `NullPointerException`-ök.

## 4.2. Adat-lekérdezés

A második program, a `HDK6NXDomQuery`, nem az XML teljes bejárására, hanem célzott lekérdezések végrehajtására készült. Ugyanazt a DOM dokumentumot használja, de csak bizonyos adatrészeket emel ki, olyan formában, hogy az eredmény a felhasználó számára áttekinthető, „riport jellegű” kimenetet adjon.

A program külön metódusokban valósítja meg az egyes lekérdezéseket. Az egyik eljárás az összes ügyfél nevét és azonosítóját írja ki, egy másik a megrendeléseket listázza állapot szerint (például a „javítás alatt” vagy „befogadva” státuszú bejegyzések), egy harmadik pedig a technikusok adatait gyűjti össze. Mindez ugyanarra a DOM fára épül, de a fókusz az, hogy csak a releváns elemek kerüljenek megjelenítésre.

Listing 5. Megrendelések szűrése állapot alapján

```
1 NodeList megrendelesek = doc.getElementsByTagName("megrendeles")
2 ;
3
4 for (int i = 0; i < megrendelesek.getLength(); i++)
5 {
6
7     Element megr = (Element) megrendelesek.item(i);
8     String allapot = getText(megr, "allapot");
9     if (keresettAllapot.equalsIgnoreCase(allapot))
10    {
11        String megrID = megr.getAttribute(
12            "megrendelesID");
13        String hiba = getText(megr, "hibaLeiras");
14        String datum = getText(megr, "beerkDatum");
15        String fizetendo= getText(megr, "fizetendoOsszeg
16            ");
17
18        System.out.println("Megrendelés ID: " + megrID);
19        System.out.println(" Hiba: " + hiba);
20        System.out.println(" Dátum: " + datum);
21        System.out.println(" Fizetendő: " + fizetendo);
22    }
23 }
```

Ebben a részben a program már nem a teljes struktúrát járja be, hanem kizárolag a `<megrendeles>` elemeket vizsgálja, és csak azokat az eseteket írja ki, amelyeknél az állapot megegyezik a paraméterként megadott értékkel. Így jól láthatóvá válnak például a még folyamatban lévő javítások, vagy a frissen befogadott munkák.

### 4.3. Adatmódosítás

A harmadik komponens, a HDK6NXDomModify feladata az, hogy a meglévő XML dokumentum tartalmát programból módosítsa, majd az eredményt egy új fájlban eltárolja. A program először ugyanúgy betölti a DOM struktúrát, majd célzottan módosít bizonyos csomópontokat, illetve új elemeket szűr be a fába.

Az egyik művelet egy konkrét megrendelés állapotát változtatja meg: a megadott `megrendelesID` alapján megkeresi az adott `<megrendeles>` elemet, majd átírja az `<allapot>` gyermekelem szövegét. Egy másik művelet új megrendelést hoz létre egy

kiválasztott ügyfél adott eszközéhez: létrejön egy új <megrendeles> elem az összes szükséges al-elemmel, és ez bekerül az eszköz gyermekei közé.

Listing 6. Megrendelés módosítása DOM-ban

```
1 NodeList megrendelesek = doc.getElementsByTagName("megrendeles")
2     ;
3 for (int i = 0; i < megrendelesek.getLength(); i++) {
4     Element megr = (Element) megrendelesek.item(i);
5     if ("M002".equals(megr.getAttribute("megrendelesID"))) {
6         Element allapotElem = (Element) megr.getElementsByTagName(
7             "allapot").item(0);
8         allapotElem.setTextContent("javítás alatt");
9     }
10 }
```

A módosítások elvégzése után a program a teljes DOM fát visszaírja egy új XML fájlba. Ehhez a Transformer osztályt használja, amely képes a Document objektumot kimeneti állományra „transzformálni”. A megfelelő beállításokkal gondoskodik arról is, hogy az eredmény UTF-8 kódolású és szépen behúzott, olvasható struktúrájú legyen.

Listing 7. Módosított XML dokumentum mentése fájlba

```
1 TransformerFactory transformerFactory = TransformerFactory.
2     newInstance();
3 Transformer transformer = transformerFactory.newTransformer();
4 transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
5 transformer.setOutputProperty(OutputKeys.INDENT, "yes");
6 transformer.setOutputProperty("{[http://xml.apache.org/xslt}
7     indent-amount](http://xml.apache.org/xslt}indent-amount)", "2
8     ");
9
10 DOMSource source = new DOMSource(doc);
11 StreamResult result = new StreamResult(new File(
12     "HDK6NX_XML_updated.xml"));
13 transformer.transform(source, result);
```

A fenti folyamat eredményeként a teljes szerviznyilvántartás módosított állapota egy új XML fájlban kerül eltárolásra. Így nemcsak olvasni és lekérdezni lehet a telefon- és laptopszerviz adatait, hanem programozottan karbantartani és bővíteni is, miközben az eredeti struktúra és az XML érvényessége is megmarad.