IdomSoft

Fejlesztési módszertan

FLORA
Alkalmazás-fejlesztési
Környezet

Tartalomjegyzék

1 Bevezetés ... 3
2 Általános elvárások ... 3
2.1 Funkcionális alkalmasság ... 3
2.2 Teljesítmény hatékonyság ... 4
2.3 Kompatibilitás ... 4
2.4 Használhatóság ... 4
2.5 Megbízhatóság ... 5
2.6 Biztonság ... 5
2.7 Karbantarthatóság ... 6
2.8 Hordozhatóság ... 7
3 Műszaki elvárások ... 8
3.1 Leállítási jelzések kezelése ... 8
3.2 Állapotmentesség ... 8
3.3 Feature flags ... 8
3.4 Ellenőrző HTTP kérések ... 9
3.5 Hálózati kiesés kezelése ... 9
3.6 Dinamikus címzés ... 9
3.7 Adatok megfelelő védelme ... 9
3.8 Változás kezelés: adatbázis séma ... 9
3.9 Változás kezelés: verzióváltozások ... 10
3.10 Hálózati kapcsolatok időtartama ... 10
4 Alkalmazás tervezési ajánlások ... 10

1 Bevezetés

Jelen dokumentumban az ÁAFK-ban betartandó alkalmazástervezési, alkalmazásfejlesztési alapkövetelményeket ismertetjük, részletezve az architektúra tervezése és az alkalmazás fejlesztése során betartandó alapelveket, általános és IT szakmai szabályokat, technológiai elvárásokat, forráskód formázási és dokumentációs elvárásokra, erőforrás méretezési szempontokra, teljesítmény és megbízhatósági skálázásra, terhelési megfontolásokra vonatkozóan.

Az ÁAFK-ban sokféle alkalmazás fejlesztése megvalósítható, de az ÁAFK leginkább a cloud native architekturális elvek mentén fejlesztett alkalmazásokat támogatja, ezek esetében áll rendelkezésre a legnagyobb rugalmasság, a legtöbb szolgáltatás. Az ÁAFK-ban definiált megfelelőségi szintek közül a legmagasabb az ilyen elvek mentén fejlesztett alkalmazásokra vonatkozik, és ezekre vonatkoznak a legszigorúbb követelmények is. Az alábbiakban áttekintjük az alkalmazásokkal szemben támasztott követelményeket, illetve ajánlásokat. Ezek a követelmények és ajánlások elsősorban a legmagasabb megfelelőségi szintre vonatkoznak. A kevesebb megfelelőségi szempontot teljesítő alkalmazások esetében nem minden követelmény és ajánlás releváns.

2 Általános elvárások

Jelen dokumentumban egy olyan alap követelményrendszer megfogalmazására került sor, amely értelmezhető a projektek keretében megvalósítandó legtöbb szoftverrendszerre.

Az egyes fejlesztendő alkalmazásokra (szoftvertermékekre) vonatkozó részletes követelményrendszert a jelen dokumentumban összeállított alapkövetelmények figyelembevételével készült el a Követelményrendszer.

2.1 Funkcionális alkalmasság

1. Az alkalmazásnak (szoftvertermékének) olyan funkciókat kell biztosítani, amelyek teljes mértékben megfelelnek a specifikációban meghatározott és az azokból eredő igényeknek.
2. Az alkalmazásnak (szoftvertermékének) szolgáltatnia kell az igényeknek megfelelő eredményeket a megkívánt pontossággal és meg kell könnyítenie a specifikált feladatok elvégzését.
3. Az alkalmazás (szoftvertermék) átadás-átvételi folyamatában átvételi tesztet kell végezni (User Acceptance Test - UAT), amelynek során ellenőrizni kell a funkcionális és a technikai elvárások teljesülését. Vizsgálni kell az alkalmazás helyességét, megbízhatóságát, az implementáció és a specifikáció közti viszonyt. A felhasználói

teszt mellett üzemeltetői átvételi tesztet is végezni kell. Az UAT-ről készült jegyzőkönyvnek be kell kerülnie az átvételi dokumentációba.

2.2 Teljesítmény hatékonyság

1. Az alkalmazásnak (szoftverterméknek) hatékonyan, takarékosan kell használnia a futtató környezet erőforrásait, a nagyobb terhelések esetén azonban képesnek kell lennie a rendelkezésére álló erőforrások maximális kihasználására is.
2. Kiemelten fontos az adatbázis-tervezés általános alapelveinek betartása: normalizálás, funkcionális függőségek, a lehető legkisebb redundanciára törekvés (relációs adatbázis esetén).
3. Az alkalmazásnak (szoftverterméknek) meg kell felelnie a specifikációban meghatározott performancia elvárásoknak.
4. Az alkalmazás automatizált szkriptek segítségével legyen fordítható, tesztelhető és build-elhető, közvetlenül a forráskódból.
5. A hosszan futó műveletek legyenek külön modulba leválasztva, és aszinkron módon lecsatolva a felhasználói felületet kiszolgáló kódtól.

2.3 Kompatibilitás

1. Az alkalmazásnak (szoftverterméknek) képesnek kell lennie az együttműködésre környezetével.
2. Az alkalmazásnak (szoftverterméknek) hatékonyan kell ellátnia a feladatát osztott erőforrású környezetben is anélkül, hogy káros hatással lenne az infrastruktúra többi elemére.
3. Az alkalmazásnak (szoftverterméknek) kliens oldalon platform-független technológia alkalmazásával kell készülnie és web alapú felhasználói felületen keresztül kell elérhetőnek lennie.
4. Az alkalmazásnak (szoftverterméknek) a kormányzati infrastruktúrába – előre nem tervezett, illetve indokolatlan infrastruktúra fejlesztés nélkül – integrálhatónak kell lennie, kiemelt elvárás a megrendelőnél rendelkezésre álló üzemeltetői kompetenciával és kapacitással stabilan működtethető technológiák használata.
5. Mindemellett azonban a kód komplexitását célszerű a lehető legalacsonyabb szinten tartani, a konkrét célt nem szolgáló, funkció nélküli összetett hierarchiák és függőségek kerülendők.
6. A kód nem tartalmazhat statikus hivatkozásokat, a működéshez szükséges valamennyi információ adatbázisban vagy konfigurációban tárolandó. (pl. IP címek, hardcode-olt könyvtárhivatkozások, plaintext jelszavak, „magic constant” -ok).

2.4 Használhatóság

1. Felhasználóbarát megjelenés:
    1. A kezelése legyen ergonomikus, logikus, következetes, könnyen áttekinthető;
    2. A kezelőfelületek legyenek jól olvashatóak, az aktuális képernyő felbontáshoz igazodók;
    3. Magyar nyelvű felhasználói felület;
    4. A felhasználók számára is érthető magyar nyelvű (hiba)üzenetek.

2. Felhasználóbarát működés:
    1. Kezelése legyen könnyen megtanulható, ne igényeljen informatikai ismeretet;
    2. Törekedni kell a felhasználók által végzett műveletek egyszerűségére, a manuálisan végzendő műveletek minimalizálására;
    3. A rendszer kezelőfelületein olyan ellenőrzések, figyelmeztetések beépítése szükséges, amelyek lehetőség szerint védik a felhasználókat a nem szabályszerű vagy nem szándékosan végzett műveletek végrehajtásától;
    4. Gyors válaszidők;
    5. A specifikációban meghatározott performancia elvárásoknak való megfelelés.
3. Felhasználóbarát dokumentáció:
    1. Magyar nyelvű, a rendszer valós működését leíró felhasználói dokumentáció;
    2. A felhasználók számára könnyen értelmezhető fogalmazás, az értelmezést segítő ábrák használata;
    3. Informatikai technikákkal publikálható formátum (pl. network share, web link).

2.5 Megbízhatóság

1. A szoftverrendszernek tartósan a specifikációban elvárt funkcionalitást és teljesítményt biztosítva kell működnie.
2. Robusztus, stabil működés, jól bevált technikák, technológiák használata.
3. Magas rendelkezésre állást, skálázhatóságot biztosító környezetbe (HA/LB) illeszthetőség biztosítása (akár rétegenként is).

2.6 Biztonság

1. A szoftverrendszerben biztosítani kell a vele kezelt, feldolgozott és a benne tárolt adatok védelmét.
2. Biztosítani kell, hogy az adatokhoz csak az arra feljogosított (megfelelő jogosultsággal rendelkező) személyek, technikai rendszerelemek férhessenek hozzá.
3. Adatvédelmi, adatbiztonsági jogszabályokat, alapelveket nem sértő technikák alkalmazása.
4. Magyar és Európai Uniós jogszabályok és egyéb normák figyelembe vétele (titkosítás, adatvédelem stb.).
5. Naplózás (pl. biztonsági, felhasználói szokások, performancia):
    1. A naplózó funkcióknak célhoz kötöttnek, erőforrás takarékosnak, továbbá könnyen visszakereshetőnek és értelmezhetőnek kell lennie;
    2. Az adatkezeléssel járó felhasználói és egyéb tevékenységeknek minden esetben hitelesen naplózottnak kell lennie, a letagadhatatlanság és a számon kérhetőség érdekében;
    3. Az adatkezeléssel járó tevékenységeket naplózó funkció nem lehet megkerülhető, kikapcsolható;
    4. A felhasználói szokások és a performancia naplózásának az üzemeltetési feladatokhoz is felhasználhatónak kell lennie. Az erőforrások takarékos használata érdekében célszerű, ha a működése ki/be kapcsolható;
    5. A naplóállományoknak önmagukban is menthetőnek, archiválhatónak kell lennie.
6. Címtárintegrált jogosultság kezelés.

7. A jogosultságkezelő funkciónak alkalmasnak kell lennie az aktuális jogosultságok lekérdezésére, a jogosultságváltozások történeti visszakereshetőségére személyenként és jogosultságonként is.
8. A jogosultságkezelésnek hitelesnek és megkerülhetetlennek kell lennie, biztosítania kell az egyes szerepkörök hozzáférési szintjeinek szétválaszthatóságát.
9. Védelmi mechanizmusok kialakítása az alkalmazás sérülékenységi kockázatainak minimalizálása érdekében (OWASP Top 10).
10. Az alkalmazásba épített védelmi mechanizmusokon fennakadt támadásgyanús eseményeket is naplózni kell. (biztonsági naplóban) Az eseményről a naplózó funkción keresztül értesíteni kell küldeni pl. az alkalmazásgazdának.
11. A tranzakciók során minden esetben biztosítani kell az adatok integritását.
12. Az adatkezelési műveletek után az adatok minden esetben konzisztens állapotban legyenek.

2.7 Karbantarthatóság

1. A rendszert úgy kell kialakítani, hogy az könnyen javítható, módosítható, továbbfejleszthető legyen.
2. A rendszermodulok tesztelhetők, újrafelhasználhatók, a fejlesztett kódok könnyen értelmezhetők, elemezhetők legyenek.
3. Továbbfejleszthetőség biztosítása.
4. Az alkalmazott technikák és technológiák semmilyen módon nem korlátozhatják a versenyt a későbbi módosításra, továbbfejlesztésre kiírt pályázat estén.
5. A meghatározott kódformázási szabályoknak való megfelelés.
6. A forráskód legyen egyszerű, áttekinthető, könnyen értelmezhető, duplikációktól mentes és könnyen karbantartható.
7. Az egyes elemek szerepének, működésének megértését segítő commentek használata.
8. Egységes, leíró (beszédes) névkonvenciók használata.
9. SOLID alapelvek figyelembevétele:
    1. Egyszeres felelősség elve: egy osztály vagy modul egy, és csak egy felelősséggel rendelkezzen
    2. Nyitva zárt alapelv: egy osztály vagy modul, legyen nyílt a kiterjesztésre, de zárt a módosításra.
    3. Liskov-féle helyettesíthetőségi alapelv: minden osztály legyen helyettesíthető a leszármazott osztályával anélkül, hogy a program helyes működése megváltozna.
    4. Interfészek szétválasztásának elve: több specifikus interface jobb, mint egy általános.
    5. Függőséginverzió alapelve: a kód függjön absztrakcióktól, ne konkrét implementációktól. Dependency injection használata.
10. Széles körben elterjedt verziókezelő technológia használata. (pl. Git) A szoftvertermék átadásakor az aktuális release-hez tartozó forráskódokat a verziókezelőből exportált állományként át kell adni.
11. Csak olyan, széles körben elterjedt technológiák (pl. framework) használhatók, amelyek bárki számára elérhetőek, nem kötődnek a fejlesztést végző céghez vagy annak érdekköréhez. [megjegyzés: a kivételeket szabályozni kell]

12. Többrétegű architektúra (pl. Prezentációs réteg, Üzleti logika réteg, Perzisztencia réteg) A szükséges rétegek számát és funkcióját az egyes feladathoz igazítottan, többek közt a várható terhelés figyelembe vételéve, a robusztusság, skálázhatóság és a hatékony, gazdaságos üzemeltethetőség érdekében kell meghatározni.
13. Preferált a perzisztens állapot mentes üzleti logikai és prezentációs réteg kialakítása. Az alkalmazás üzleti logikáját megvalósító modul rugalmasan legyen skálázható, ne kelljen az alkalmazás specifikus állapot szinkronizációt figyelembe venni a skálázás során.
14. Moduláris felépítés
    1. A kialakított szoftverrendszer különálló, egymással csak a saját interfészükön kommunikáló, lazán csatolt modulok strukturált halmazaként legyen kialakítva.
    2. Az alkalmazás mikro szolgáltatás tervezési elvek mentén kerüljön kialakításra (Microservices Architecture). A tervezés és fejlesztés során törekedni kell arra, hogy a rendszer egységbe zárt (self-contained) legyen
    3. Az egyes rétegekben a funkciókat megvalósító rendszerkomponensek interfészeit dokumentált, biztonságos, uniformizált módon kell kialakítani. A modulok közti kölcsönös függőséget minimalizálni kell.
    4. A moduláris felépítésnek biztosítania kell a szoftverrendszer hatékony továbbfejleszthetőségét, módosíthatóságát a meglévő, jól bevált modulok (kódok) újrafelhasználhatóságát, újhasznosíthatóságát.
    5. Az alkalmazás legyen képes kezelni, ha egy-egy modul átmenetileg nem elérhető. Ilyen helyzetben ne omoljon össze az alkalmazás.
    6. Az logikai és prezentációs réteg elemei jól definiált szolgáltatások mentén függjenek a környezetüktől (pl. adatbázis, IDM). A szolgáltatások elérési paramétereit környezeti változókból töltse be.
    7. A hosszan futó műveletek legyenek külön modulba leválasztva, és aszinkron módon lecsatolva a felhasználói felületet kiszolgáló kódtól.
15. Az alkalmazás automatizált szkriptek segítségével legyen fordítható és build-elhető, közvetlenül a forráskódból. A reprodukálhatóság érdekében a fejlesztést végzőnek a fordításhoz és build-eléshez szükséges állományokat, eljárásokat és egyéb információkat át kell adnia.

2.8 Hordozhatóság

1. Olyan technológiák és technikák alkalmazásával kell a szoftverrendszert elkészíteni, amelyek minimális erőforrás ráfordítással lehetővé teszik, a szoftverrendszer átköltöztetését és működtetését egy másik infrastruktúra környezetbe.
2. OpenSource technológia elemek előnyben részesítése, alkalmazása.
3. A kód nem tartalmazhat statikus hivatkozásokat (pl. IP cím, elérési út)
4. A lehető legnagyobb mértékben automatizálni kell a telepítés és beállítás folyamatát.
5. A telepítési, áttelepítési, eltávolítási leírásnak aktuálisnak, részletesnek, folyamatszerűnek és követhetőnek kell lenni. Tartalmaznia kell a függőségeket és a várható kimeneteket.

3 Műszaki elvárások

Már az alkalmazás tervezési szakaszában figyelembe kell venni a hatékony üzemeltethetőség, továbbfejleszthetőség, módosíthatóság szempontjait, ennek részletei kerültek kifejtésre az alábbi fejezetben.

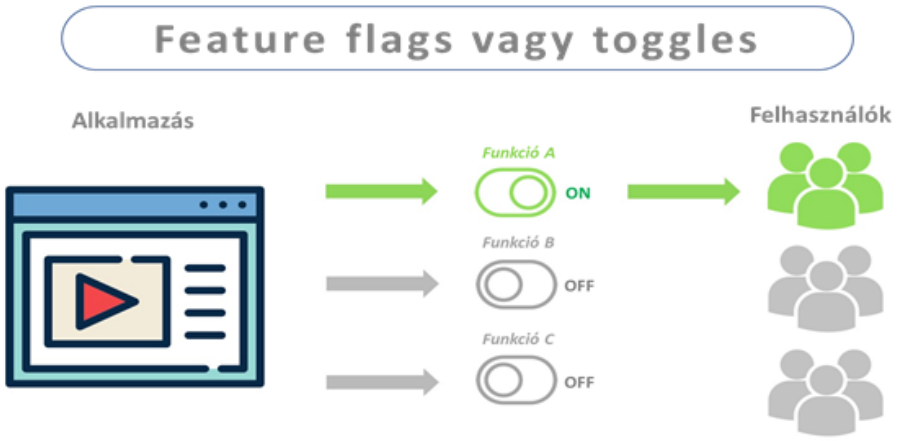
3.1 Leállítási jelzések kezelése

A fejlesztett alkalmazásnak kötelező kezelnie a leállításra utaló jelzéseket (pl. SIGTERM), ugyanis a konténerek bármikor leállíthatók és a leállítási jelzés után új tranzakciókat kezdeni nem szabad, a nyitott kapcsolatokat, állapotokat le kell zárni vagy át kell adni más munkavégző egységeknek.

3.2 Állapotmentesség

Az alkalmazás nem támaszkodhat saját file-rendszeren tárolt adatok jelenlétére (állapotmentesség). Van ugyan mód rá, hogy egy konténerben diszkre adatokat rögzítsen, de mivel a szoftver egy-egy példánya azt hiszi ő az egyetlen, aki működik, viszont valóságban több izolált példány fut, így lehetséges, hogy a diszkre másik példány írja ki az adatot, mint aki próbálja olvasni. Az egyetlen kivétel a körültekintően megvalósított cache lehet, amikor is az alkalmazás olyan adatot tárol a diszken, amit bármikor újra be tud szerezni, vagy elő tud állítani, és a file-ok jelenléte csak hálózati vagy egyéb erőforrás takarékosságot jelent, de hiányuk esetén is teljesen azonos kimenetet állít elő az alkalmazás.

3.3 Feature flags



1. *ábra - Feature flags*

Elvárás az alkalmazás készítése során úgynevezett feature flag-ekkel (technika az egyes funkciók ki- és bekapcsolására) kapcsolhatóvá tenni valamennyi felhasználói funkció elérhetőségét. Ezt a képességet az adminisztrátorok menet közben alkalmazhatják a probléma kezelés során, bizonyos korlátozások bevezetésére vagy diagnosztikai ellenőrzésekre. Előfordulhat ugyanis, hogy egyik-másik funkció átmeneti problémája miatt a felhasználók más - fontosabb műveleteket nem tudnak elvégezni, így az alkalmazás dinamikus konfigurációjával

(pl.: adatbázisban tárolt feature flag-ek) lehetővé válik a problémás funkciók kikapcsolása, és a rendszer egészének üzemben tartása. Ugyanez a funkció segítheti az új képességek bevezetését is verziófrissítések után, hiszen az új funkcióhoz szükséges adatbázis sémaváltozások elvégzése után engedélyezhetők az új funkciók.

3.4 Ellenőrző HTTP kérések

Az alkalmazásnak rendeltetésszerű működés közben képesnek kell lenni válaszolnia ellenőrző HTTP kérésekre valamelyik port-ján akkor is, ha egyébként nem HTTP kérésekkel dolgozik jellemzően. Ezeknek a kéréseknek és válaszoknak csak az a célja, hogy az üzemeltető környezet ellenőrizni tudja, az alkalmazás rendeltetésszerűen működik-e, és ha nem, akkor leállítja, és újat indít helyette.

3.5 Hálózati kiesés kezelése

A fejlesztés során az alkalmazásoknak kezelnie kell bármely hálózati kapcsolat megszakadását, és az újra-próbálkozásra olyan késleltetési módszert kell alkalmaznia, mely nem terheli túl a szolgáltatót a kapcsolat helyreállása esetén. Érdemes megfontolni a szolgáltatás természetéből adódó lehetőségeket, mint például egy űrlap kitöltésnél a beküldés kliens általi vezérlés (időnként ellenőrzi a kapcsolatot majd, ha sikeres akkor beadja a már előzetesen leadott űrlapokat).

3.6 Dinamikus címzés

Az alkalmazás számára valamennyi külső szolgáltatás dinamikus konfigurációként érhető el, tehát nem szabad host-neveket, ip-címeket, port-okat, felhasználóneveket vagy jelszavakat tárolni a forráskód mellett, hanem az alkalmazás azt környezetleíró változóból kapja meg. Amennyiben a konfiguráció változik, új alkalmazáspéldányok indulnak és a régiek leállításra kerülnek.

3.7 Adatok megfelelő védelme

A rendszer építése során számtalan kisebb-nagyobb bizalmas adat (jelszavak, titkos kulcsok, token-ek) keletkezik. Fontos azonban, hogy ezeket mindig szoftverrel generáljuk és semmilyen körülmények között ne juttassuk ki a környezetből, ne kerüljenek verziókövető rendszerbe és ne használjunk fix vagy "szokásos" jelszavakat.

3.8 Változás kezelés: adatbázis séma

Az alkalmazás tervezése során javasolt a szoftver szerkezetét és frissítési folyamatát úgy megtervezni, hogy az minden verzióváltás során folyamatos működést tegyen lehetővé. Például amennyiben az alkalmazás verzióváltás során az adatbázisséma változása is szükséges, akkor vagy a korábbi verziónak kell kezelnie az új adatbázissémával járó mellékhatásokat, vagy az új verziót kell úgy elkészíteni, hogy addig is rendeltetésszerűen tudjon működni, amíg az adatbázis séma nem frissült. Ha például az alkalmazás nem feltételezi a változásokat az adatbázisszerkezetben, akkor nem lehetséges a gördülő frissítés (rolling deployment

strategy), csak a teljes leállítás és elindítás, mely szolgáltatáskieséssel jár, és szembe megy a modern fejlesztési irányvonalakkal.

3.9 Változás kezelés: verzióváltozások

Az alkalmazás fejlesztése és frissítése során szem előtt kell tartani, hogy a gördülő telepítés során egyszerre fut a régebbi és az újabb verzió az alkalmazásból, így az átmeneti állapot során vegyes szerkezetű adatok kerülhetnek rögzítésre. Ez jellemzően nem okoz problémát, ha a fejlesztés és frissítés folyamatosan zajlik, azonban intenzív fejlesztés és ritka frissítés mellett számottevő lehet a működési eltérés, és ennek következményeit, kockázatait a fejlesztő feladata kezelni.

3.10 Hálózati kapcsolatok időtartama

A fejlesztés során javasolt kerülni a hosszútávú hálózati kapcsolatokat, hiszen maga a kapcsolat egy logikai állapot, tárolást feltételez. Inkább a rövid HTTP alapú kérdés-válasz kommunikáció a követendő, úgy, hogy a kliens sose feltételezheti, hogy ugyanaz a szerver válaszol az egymást követő kérdésekre. (állapotmentesség).

4 Alkalmazás tervezési ajánlások

Az alkalmazások tervezése során célszerű előre gondolni a felhőalapú futtatási képességek biztosításához. Manapság felhő szempontból az alkalmazásokat három fő kategóriába lehet sorolni:

- nem felhő kompatibilis
- felhő kompatibilis (cloud ready)
- felhő orientált (cloud native)

Az alkalmazások felhőalapú futtatási képességére vonatkozó extra elvárások jellemzően mind olyan tervezési elvárások, amelyeknek való megfelelés abban az esetben is igen hasznos bizonyul, ha az alkalmazást végül nem felhő alapú környezetben működtetjük. Ezért a zöldmezős fejlesztések során mindenképpen célszerű felhő orientált architektúra elveknek megfelelve alakítani a rendszer felépítését, mivel ezek az elvek általában segítik a rugalmas és hibatűrő alkalmazás architektúra kialakítását. Az alábbiakban áttekintjük a felhő alapú alkalmazás architektúrák során jelentkező fő kihívásokat és az ezekre alkalmazandó bevált tervezési mintákat[1]. Ezen minták megfelelő alkalmazása a leendő alkalmazás nem funkcionális képességeit nagy mértékben javíthatják.

A felhő alapú alkalmazás fejlesztés fő kihívásai:

- Rendelkezésre állás:

A rendelkezésre állás az időnek az a hányada, amelyben a rendszer elérhető és működőképes. Jellemzően a teljes futási idő százalékos hányadaként mérik. Befolyásolhatják rendszer hibák, infrastruktúra problémák, rosszindulatú támadások és terheltségi szintek. A rendszerek

jellemzően elvárt szolgáltatási szintek mentén üzemelnek, ezért a rendelkezésre állás mindig fontos szempont a tervezés során.

- Adat kezelés:

Felhő alapú alkalmazások tervezése során az adatkezelés mindig igen fontos szempont és sok minőségi jellemzőre van hatással. Az adatok jellemzően több helyen kerülnek letárolásra, különféle szervereken, teljesítmény, skálázási és rendelkezésre állási okokból, és ebből fakadóan számos kihívással kell szembenézni a tervezés során. Például az adatok konzisztenciáját biztosítani kell, és szükséges jellemzően az adatok megfelelő szinkronizálása az egyes tárolási helyek között.

- Tervezés és megvalósítás:

Szakszerű tervezés során fontos figyelembe venni a rendszer komponensek tervezése során a konzisztens és koherens működési elveket, a karbantarthatóságot, és az újrafelhasználhatóságot, ezáltal hatékonyabbá téve a leendő rendszer adminisztrációs és fejlesztési folyamatait. A tervezés és megvalósítás során meghozott döntések igen nagy hatással vannak a későbbi rendszer minőségére, és a teljes fenntartási költségére.

- Üzenet kezelés:

A felhő alapú rendszerek elosztott természeténél fogva igényelnek egy üzenet alapú kommunikációs réteget, amelynek segítségével összeköthetők az egyes komponensek, minél inkább lazán csatoltan, hogy a skálázás minél nagyobb léptékű lehessen. Az aszinkron üzenetek alapján megvalósított kommunikáció széles körben elterjedt, és számos előnnyel jár, azonban kihívásokat is támaszt az alkalmazás tervezése során, mert gondolni kell olyan szempontokra, mint például az üzenetek sorrendje, hibás üzenetek kezelése és idempotens működés.

- Menedzsment és monitorozás:

Felhő alapú futtató környezetben az alkalmazások jellemzően magasabb szinten izolált környezetben futnak (pl. mikrokonténerben) és gyakran nem biztosított az alacsonyszintű infrastruktúrához, vagy operációs rendszerhez való hozzáférés. Ez nehezebbé teheti a menedzsmentet és monitorozást a hagyományos alkalmazás üzemeltetéshez képest. Az alkalmazásoknak publikálniuk kell futásidejű jellemzőket, paramétereket és kontrollokat, amelyeket az adminisztrátorok és üzemeltetők monitorozhatnak, valamint ezeken keresztül menedzselhetik a rendszer működését, illetve adott esetben paraméterezhetik, hangolhatják a működését a rendszer leállítása vagy újra deploymentje nélkül.

- Teljesítmény és skálázás:

A teljesítmény a rendszer válaszadási képességének adott időn belüli mértéke, a skálázás pedig a rendszer azon képessége, hogy a teljesítmény romlása nélkül legyen képes kezelni nagyobb terheléseket, szükség esetén az erőforrások arányos növelése árán. A felhő alapú tervezési elvek mentén készített alkalmazások jellemzően fel vannak készítve különféle, változó terhelési jellemzők melletti hatékony működésre, rugalmas skálázódásra. A skálázás

kapcsán nem csak a számítási kapacitásra kell tervezni, hanem adattárolásra, üzenet kezelési infrastruktúrára, illetve további hasonló alapvető erőforrásokra.

- Rugalmasság:

A rugalmasság a rendszer azon képessége, hogy különösebb fennakadás nélkül legyen képes a hibákat kezelni és azokból helyreállni. A rendszert úgy kell megtervezni, hogy különféle területeken fellépő átmeneti vagy állandó hibák esetén is minél rugalmasabban legyen képes ezekre reagálni, lehetőleg a szolgáltatás minőségében bekövetkező csökkenés arányos legyen a hibával. A rugalmassághoz fontos a hibák mielőbbi pontos felismerése, és az azokból való helyreállás hatékony és gyors megvalósítása.

- Biztonság:

A biztonság szinte minden rendszer tervezése során alapkövetelmény. A rendszer azon képessége, hogy megelőzze a véletlen vagy szándékos károkozást, a rendszer nem rendeltetésszerű használatát. Az alkalmazásokat úgy kell megtervezni és beüzemelni, hogy biztosított legyen a rosszindulatú támadások elleni védelem, csak az engedélyezett felhasználók számára legyen elérhető, és az adatok megfelelő védelmet élvezzenek.

Az alábbiakban megvizsgáljuk, hogy a fenti kihívásokra milyen tervezési minták alkalmazásával lehet hatékonyan reagálni. A fenti kategóriákhoz tartozóan rendszerbe foglaljuk az egyes tervezési mintákat, majd a későbbi fejezetekben egyesével megvizsgáljuk részletesen az egyes tervezési mintákat.

[1] Forrás: Microsoft Azure Cloud Design Patterns