

Általános információk, a diplomaterv szerkezete

A diplomaterv szerkezete a BME Villamosmérnöki és Informatikai Karán:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletes és pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő L^AT_EX-diplomaterv sablon dokumentum tartalma.

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindehol 2,5 cm, baloldalon 1 cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel.

Minden oldalon – az első négy szerkezeti elem kivételével – szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális alaosztással számozzuk, maximálisan 3 alaosztás mélységben (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2. táblázat. vagy képletnél (3.2)). A fejezetcímeket igazítsuk balra, a normál szövegnél viszont használjunk sorkiegyenlítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazítsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le (A L^AT_EX ehhez kézenfekvő megoldásokat nyújt).

Az irodalomjegyzék szövegközi hivatkozása történhet a Harvard-rendszerben (a szerző és az évszám megadásával) vagy sorszámozva. A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben (a L^AT_EX-sablon a Bib_TE_X segítségével mindezt automatikusan kezeli). Minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknel a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóirat címeket csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internet hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Fontos:

- A szakdolgozat készítő/diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás Karunkon ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető!
- Mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet!

Jó munkát, sikeres szakdolgozat készítést, ill. diplomatervezést kívánunk!

Feladatkiírás

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal *helyett*, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatba már nem kell beleszerkeszteni ezt a feladatkiírást.



M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Mikroszolgáltatásokra épülő architektúra fejlesztésének és tesztelésének támogatása

DIPLOMATERV

Készítette
Borlay Dániel

Konzulens
Szatmári Zoltán

2016. május 9.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Borlay Dániel*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2016. május 9.

Borlay Dániel
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon Markdown leírónyelven készült, Pandoc rendszerrel fordítható le $\text{T}_{\text{E}}\text{X}$ Live vagy $\text{MiK}_{\text{T}}\text{E}_{\text{X}} \text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ disztribúciókkal.

Abstract

This document is a skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. The skeleton was implemented in Markdown and can be compiled with Pandoc, using the T_EX Live and or the MiK_TE_X L^AT_EX compiler.

1. fejezet

Bevezetés

A bevezető tartalmazza a diplomaterv-kiírás elemzését, történelmi előzményeit, a feladat indokoltságát (a motiváció leírását), az eddigi megoldásokat, és ennek tükrében a hallgató megoldásának összefoglalását.

A bevezető szokás szerint a diplomaterv felépítésével záródik, azaz annak rövid leírásával, hogy melyik fejezet mivel foglalkozik.

2. fejezet

Mikro szolgáltatások

2.0.1. Definíció:

Nem találtam konkrét definíciót, de a mikro szolgáltatás egy olyan architektúráis modellezési mód, amikor a tervezett rendszert/alkalmazást kisebb funkciókra bontjuk, és önálló szolgáltatásokként, önálló erőforrásokkal, valamilyen jól definiált interfészen keresztül tesszük elérhetővé.

2.0.2. Technológiáról:

A mikro szolgáltatás architektúra kiépítéséhez sokféle szétválasztási módot használnak, amik közül van olyan amit a tervezési folyamat közben felmerülő főneveket, vagy igéket használnak fel, de abban megegyeznek, hogy a funkcionálitást bontják fel. Ezekkel az [integrációval foglalkozó részben](#) olvashatunk bővebben.

A mikro szolgáltatások tervezése során a következő szempontok szerint szokták megtervezni a rendszert:

- Milyen szolgáltatásokat tud nyújtani a rendszer
- Lehetséges műveletek felsorolása (igék amik a rendszerrel kapcsolatosak)
- Lehetséges erőforrások vagy entitások felsorolása (főnevek alapján szétválasztás)
- Lehetséges use-case-ek szétválasztása (felhasználási módszerek elválasztása)
- A felbontott rendszert hogyan kapcsoljuk össze
- Pipeline-ként egy hosszú folyamatot összeépítve és az információt áramoltatva
- Elosztottan, igény szerint meghívva az egyes szolgáltatásokat
- Egyes funkciókat összekapcsolva nagyobb szolgáltatások kialakítása (kötegelés)
- A kommunikáció a felhasználóval
- Egy központi szolgáltatáson keresztül, ami a többivel kommunikál
- Add-hoc minden szolgáltatás külön hívható

Ezekkel a lépésekkel meg lehet alapozni, hogy az általunk készítendő rendszer hogyan is lesz kialakítva, és milyen paraméterek mentén lesz felvágva. A választást segíti a témában elterjedt fogalom, a scaling cude[1], ami azt mutatja, hogy az architektúráis terveket milyen szempontok mentén lehet felosztani.

2.1. ábra. Scaling Cube

Ahogy a képen is látható a meghatározó felbontási fogalmak, az adat menti felbontás, a tetszőleges fogalom menti felbontás, illetve a klónozás.

Az adat menti felbontás annyit tesz, hogy a szolgáltatásokat annak megfelelően bontjuk fel, hogy az egyes szolgáltatások csak adatbázissal, vagy csak web kiszolgálással foglalkozzanak, vagy csak a felhasználói adatok esetleg a tanulók jegyeit felügyelik. Ez a mérce a mikro szolgáltatás architektúrák esetén nem annyira fontos, mivel a szolgáltatásoknak erőforrásaikat tekintve is el kell különülniük, így nem éri meg erőforrások vagy adat mentén vágni.

A tetszőleges fogalom menti felbontás annyit tesz hogy elosztott rendszert hozunk létre tetszőleges funkcionalitás szerint. Erre épít a mikro szolgáltatás architektúra is, mivel a lényege pont az egyes funkciók atomi felbontása.

A harmadik módszer arra tér ki, hogy hogyan lehet egy architektúrát felosztani, hogy skálázható legyen. Itt a klónozhatóság, avagy az egymás melletti kiszolgálás motivál. Ez a micro-service-eknél kell, hogy teljesüljön, mivel adott esetben a load balancer alatt tudnunk kell definiálni több példányt is egy szolgáltatásból.

2.1. Architektúrális minták

Mint korábban láthattuk vannak bizonyos telepítési módszerek, amik mentén szokás a mikro szolgáltatásokat felépíteni. Van aki az architektúrális tervezési minták közé sorolja a mikro szolgáltatás architektúrát, azonban nem lehet élesen elkülöníteni, mivel valamilyen csatolási módszerre szükség van, ami nem specifikus a mikro szolgáltatás-ek esetén, viszont más architektúrális mintákra jellemző.

Ilyen a Pipes and filter architektúrális minta [2], aminek a lényege, hogy a funkciókra bontott architektúrát az elérni kívánt végeredmény érdekében különböző módokon összekötünk. Ebben a módszerben az adat folyamatosan áramlik az egyes alkotó elemek között, és lépésről lépésre alakul ki a végeredmény. Elég olcsón kivitelezhető architektúrális minta, mivel csupán sorba kell kötni hozzá az egyes szolgáltatásokat, azonban nehezen lehet optimalizálni, és könnyen lehet, hogy olyan részek lesznek a feldolgozás közben, amik hátráltatják a teljes folyamatot.

Egy másik elosztott rendszerekhez kitallált minta a subscriber/publisher[3], amely arra alapszik, hogy egy szolgáltatásnak szüksége van valamilyen adatra vagy funkcióra, és ezért feliratkozik egy másik szolgáltatásra. Ennek az lesz az eredménye, hogy bizonyos szolgáltatások bizonyos más szolgáltatásokhoz fognak kötődni, és annak megfelelően fognak egymással kommunikálni, hogy milyen feladatot kell végrehajtaniuk.

2.1.1. Hivatkozások:

[1] Chris Richardson: [The Scale Cude](#)

[2] Microsoft: [Pipes and Filters Pattern] (<https://msdn.microsoft.com/en->

us/library/dn568100.aspx)

[3] Microsoft: [Publish/Subscribe](#)

2.1.2. Előnyök-hátrányok:

A mirco-service architektúrák a monolitikus architektúra ellentetjei, melyben az erőforrások központilag vannak jelezve, és minden funkció egy nagy interfészen keresztül érhető el. A monolitikus architektúra egyszerűen kiépíthető, könnyű tervezni és fejleszteni, azonban nehezen lehet kicserélni, nem elég robosztus, és nehezen skálázható, mivel az erőforrásokat közösen kezelik a funkciók.

Ezzel ellenzétben a mikro szolgáltatás architektúrát ugyan nehezen lehet megtervezni, hiszen egy elosztott rendszert kell megtervezni, ahol az adatátviteltől kezdve az erőforrás megosztáson keresztül semmi sem egyértelmű, viszont a későbbi tovább fejlesztés sokkal egyszerűbb, mivel külön csapatokat lehet rendelni az egyes szolgáltatásokhoz, és könnyen integrálhatók kicserélhetők az alkotó elemek. Mivel sok kis egységből áll, könnyebben lehet úgy skálázni a rendszert, hogy ne pazaroljuk el az erőforrásainkat, és ugyanakkor a kis szolgáltatások erőforrásokban is el vannak különítve, így nem okoz gondot, hogy fel vagy le skálázzunk egy szolgáltatást. Ennek az a hátránya, hogy le kell kezelni a skálázáskor a közös erőforrásokat. (Például ha veszünk egy autentikációs szolgáltatást, akkor ha azt fel skálázzuk, meg kell tartanunk a felhasználók listáját, így duplikálni kell az adatbázist, és fenntartani a konzisztenciát) Ugyan csak előnye a mirco-service architektúrának, hogy különböző technológiákat lehet keverni vele, mivel az egyes szolgáltatások különböző technológiákkal különböző platformon is futhatnak.

2.1.3. Kommunikáció:

A szolgáltatások közötti kommunikáció nincs lekötve de jellemző a REST-es API, vagy a webservice-re jellemző XML alapú kommunikáció. Minden szolgáltatáshoz tartozik egy önálló interfész, amin keresztül a többi szolgáltatás kommunikálhat vele, és minden funkcióját el lehet érni. Ennek az interfésznek olyannak kell lennie, hogy az implementáció szabadon változtatható legyen, és ne kelljen más szolgáltatásokat megváltoztatni, ha a saját szolgáltatásunkat változtatjuk. Ez segíti a több csapattal való munkát, és lehetővé teszi hogy teljesen függetlenül létezzenek a szolgáltatások.

2.1.4. Példák:

Amazon - minden Amazon-nal kommunikáló eszköz illetve az egyes funkciók implementációja is szolgáltatásokra van szedve, és ezeket hívják az egyes funkciók (vm indítás, törlés, mozgatás, stb.)

eBay - Különböző műveletek szerint van felbonva a a funkcionalitás, és ennek megfelelően külön szolgáltatásként érhető el a fizetés, megrendelés, szállítási információk, stb.

NetFlix - A nagy terhelést elkerülendő bizonyos streaming szolgáltatásokat átlalakítottak, hogy a mikro szolgáltatás architektúra szerint működjön.

Mintapéldák: <http://eventuate.io/exampleapps.html>

2.1.5. Hivatkozások:

- [1] Peter Van Garderen: *Archivematica: Using Micro-Services and open-source software to deliver a comprehensive digital curation solution*
- [2] Manfred Sneps-Sneppe, Dmitry Namiot: *Micro-service Architecture for Emerging Telecom Applications*
- [3] Chris Richardson (Kong): *Pattern: Microservices Architecture*
- [4] Jose Ignacio Fernández-Villamor, Carlos Á. Iglesias, Mercedes Garijo: *Microservices: Lightweight services descriptors for REST architectural style*
- [5] Dimirty Namiot, Manfréd Sneps-Sneppe: *On Mirco-services Architecture*
- [6] Augusto Ciuffoletta: *Automated deployment of a microservice-based monitoring infrastructure*
- [7] Reagan Moore: *Towards a Theory of Digital Preservation*
- [8] Sebastián Peyrott: *An Introduction to Microservices, Part 1*
- [9] James Lewis, Martin Fowler: *Microservices a definition of this new architectural term*

3. fejezet

Technológiai áttekintés

Az integrációhoz olyan technológiákat lehet használni, melyek lehetővé teszik az egyes szolgáltatások elkülönült működését.

A következő feladatokra kellenek technológiák: * Hogyan lehet feltelepíteni egy önálló szolgáltatást? (telepítés) * Hogyan lehet összekötni ezeket a szolgáltatásokat? (automatikus környezet felderítés) * Hogyan lehet fenntartani, változtatni a szolgáltatások környezetét? (integrációs keretrendszer) * Hogyan lehet skálázni a szolgáltatást? (skálázás) * Hogyan lehet egységesen használni a skálázott szolgáltatásokat? (load balance, konzisztencia fenntartás) * Hogyan lehet virtualizáltan ezt kivitelezni? (virtualizálás) * A meglévő szolgáltatásokat hogyan tartsuk nyilván? (service registry) * Hogyan figyeljük meg a rendszert működés közben (monitorozás, loggolás)

3.0.6. Telepítés:

A microservice-eket valamilyen módon létre kell hozni, egy hosthoz kell rendelni, és az egyes elemeket össze kell kötni. A szolgáltatások telepítéséhez olyan technológiára van szükség amivel könnyen elérhetünk egy távoli gépet, és könnyen kezelhetsük az ottani erőforrásokat. Ehhez a legkézenfekvőbb megoldás a Linux rendszerek esetén az SSH kapcsolaton keresztül végrehajtott Bash parancs, de vannak eszközök, amikkel ezt egyszerűbben és elosztottabban is megtehetjük.

- **Jenkins:** A Jenkins egy olyan folytonos integráláshoz kifejlesztett eszköz, mellyel képesek vagyunk különböző funkciókat automatizálni, vagy időzítetten futtatni. A Jenkins egy Java alapú webes felülettel rendelkező alkalmazás, amely képes bash parancsokat futtatni, Docker konténereket kezelni, build-eket futtatni, illetve a hozzá fejlesztett plugin-eken keresztül, szinte bármire képes. Támogatja a fürtözést is, így képesek vagyunk Jenkins slave-eket létrehozni, amik a mester szerverrel kommunikálva végzik el a dolgukat. A microservice architektúrák esetén alkalmas a szolgáltatások telepítésére, és tesztelésére.
- **ElasticBox:** Egy olyan alkalmazás, melyben nyilvántarthatjuk az alkalmazásainkat, és könnyen egyszerűen telepíthetjük őket. Támogatja a konfigurációk változását, illetve számos technológiát, amivel karban tarthatjuk a környezetünket. (Docker,

Puppet, Ansible, Chef, stb) Együtt működik különböző cloud megoldásokkal, mint az AWS, vSphere, Azure, és más környezetek. Hasonlít a Jenkins-re, csupán ki van élezve a microservice architektúrák vezérlésére. (Illetve fizetős a Jenkins-el ellentétben) Mindent végre tud hajtani ami egy microservice alkalmazáshoz szükséges, teljes körű felügyeletet biztosít.

Egyéb lehetőség, hogy a fejlesztő készít magának egy olyan szkriptet, ami elkészíti számára a micro-service architektúrát, és lehetővé teszi az elemek dinamikus kicserélését. (ad-hoc megoldás)

3.0.7. Környezet felderítés:

Az egyes szolgáltatásoknak meg kell találniuk egymást, hogy megfelelően működhessen a rendszer, azonban ez nem mindig triviális, így szükség van egy olyan alkalmazásra, amivel felderíthetjük az aktív szolgáltatásokat.

- **Consul:** A Hashicorp szolgáltatás felderítő alkalmazása, amely egy kliens-szerver architektúrának megfelelően megtalálja a környezetében lévő szolgáltatásokat, és figyeli az állapotukat (ha inaktívvá válik egy szolgáltatás a Consul észreveszi). Ez az alkalmazás egy folyamatosan választott mester node-ból és a többi slave node-ból áll. A mester figyeli az alárendelteket, és kezeli a kommunikációt. Egy új slave-et úgy tudunk felvenni, hogy a consul klienssel kapcsolódunk a mesterre. Ha automatizáltan tudjuk vezényelni a feliratkozást, egy nagyon erős eszköz kerül a kezünkbe, mivel eseményeket küldhetünk a szervereknek, és ezekre különböző feladatokat hajthatunk végre.

3.0.8. Integrációs keretrendszer:

A telepítéshez és a rendszer állapotának a fenntartásához egy olyan eszköz kell, amivel gyorsan egyszerűen végrehajthatjuk a változtatásainkat, és ha valamit változtatunk egy szolgáltatásban, akkor az összes hozzá hasonló szolgáltatás értesüljön a változtatásról, vagy hajtson végre ő maga is változtatást.

- **Puppet:** Olyan nyílt forrású megoldás, amellyel leírhatjuk objektum orientáltan, hogy milyen változtatásokat akarunk elérni, és a Puppet elvégzi a változtatásokat. Automatizálja a szolgáltatás változtatásának minden lépését, és egyszerű gyors megoldást szolgáltat a komplex rendszerbe integráláshoz.
- **Chef:** A Chef egy olyan konfiguráció menedzsment eszköz ami nagy mennyiségű szerver számítógépet képes kezelni, fürtözhető, és megfigyeli az alá szervezett szerverek állapotát. Tartja a kapcsolatot a gépekkel, és ha valamelyik konfiguráció nem felel meg a definiált receptkönyvnek (amiben definiálhatjuk az elvárt környezeti paramétereket) akkor változtatásokat indít be, és eléri, hogy a szerver a megfelelő konfigurációval rendelkezzen. Népszerű konfiguráció menedzsment eszköz, amiz könnyedén használhatunk integrációhoz, illetve a szolgáltatások cseréjéhez, és karbantartásához.

- **Ansible:** A Chef-hez hasonlóan képes változtatásokat eszközölni a szerver gépeken egy SSH kapcsolaton keresztül, viszont a Chef-el ellentétben nem tartja a folyamatos kapcsolatot. Az Ansible egy tipikusan integrációs célokra kifejlesztett eszköz, amelyhez felvehetjük a gépeket, amiken valamilyen konfigurációs változtatást akarunk végezni, és egy „playbook” segítségével leírhatjuk milyen változásokat kell végrehajtani melyik szerverre. Könnyen irányíthatjuk vele a szolgáltatásokat, és definiálhatunk szolgáltatásonként egy playbook-ot ami mondjuk egy fürtnyi szolgáltatást vezérel. Ez az eszköz hasznos lehet, ha egy szolgáltatásnak elő akarjuk készíteni a környezetet.
- **SaltStack:** A SaltStack nagyon hasonlít a Chef-re, mivel ez a termék is széleskörű felügyeletet, és konfiguráció menedzsment-et kínál számunkra, amit folyamatos kapcsolat fenntartással, és gyors kommunikációval ér el. Az Ansible-höz nagyon hasonlóan konfigurálható (nem lennék meglepve ha azt használná a háttérben), szintén ágens nélküli kapcsolatot tud létesíteni, és a Chef-hez hasonlóan több 10 ezer gépet tud egyszerre karbantartani.

3.0.9. Skálázás:

A microservice architektúrák egyik nagy előnye, hogy az egyes funkciókra épülő szolgáltatásokat könnyedén lehet skálázni, mivel egy load balancert használva csupán egy újabb gépet kell beszerezni, és máris nagyobb terhelést is elbír a rendszer. Ahhoz hogy ezt kivitelezni tudjuk, szükségünk van egy terhelés elosztóra, és egy olyan logikára, ami képes megsokszorozni az erőforrásainkat. Cloud-os környezetben ez könnyen kivitelezhető, egyébként hideg tartalékban tartott gépek behozatalával elérhető. Sajnálatos módon általános célú skálázó eszköz nincsen a piacon, viszont gyakran készítenek maguknak saját logikát a nagyobb gyártók.

- **Elastic Load Balancer:** Az Amazon AWS-ben az ELB avagy rugalmas terhelés elosztó az, ami ezt a célt szolgálja. Ennek a szolgáltatásnak az lenne a lényege, hogy segítse az Amazon Cloud-ban futó virtuális gépek hibatűrését, illetve egységbe szervezi a különböző elérhetőségi zónákban lévő gépeket, amivel gyorsabb elérést tudunk elérni. Mivel ez a szolgáltatás csupán az Amazon AWS-t felhasználva tud működni, nem megfelelő általános célra, azonban ha az Amazon Cloud-ban építjük fel a microservice architektúránkat, akkor erős eszköz lehet számunkra.

3.0.10. Load balancing:

A microservice architektúrának egyik fontos eleme a terhelés elosztó, vagy valamilyen fürtözést lehetővé tevő eszköz. Ez azért fontos, mert egy egységes interfészt tudunk kialakítani a szolgáltatásaink elérésére, és könnyíti a skálázódást a szolgáltatások mentén.

- **HAProxy:** Egy magas rendelkezésre állást biztosító, és megbízhatóságot növelő terhelés elosztó eszköz. Konfigurációs fájlokon keresztül megszervezhetjük, hogy mely gépet hogyan érjük el, milyen IP címek mely szolgáltatásokhoz tartoznak, illetve

round robin módon osztja szét a kéréseket az egyes szerverek között. Ez az eszköz csak és kizárólag a HTTP TCP kéréseket tudja elosztani, de egyszerű könnyen telepíthető, és könnyen kezelhető (ha nem dinamikusan változnak a fürtben lévő gépek, mert ha igen akkor szükséges egy mellékes frissítő logika is)

- **nginx:** Az Nginx egy nyílt forráskódú web kiszolgáló és reverse proxy szerver, amivel nagy méretű rendszereket kezelhetünk, és segít az alkalmazás biztonságának megőrzésében. A kiterjesztett változatával (Nginx Plus) képesek lehetünk a terhelés elosztásra, és alkalmazás telepítésre. Nem teljesen a proxy szerver szerepét váltja ki, de képes elvégezni azt.

3.0.11. Virtualizálás:

A microservice architektúrák kialakításánál nagy előnyt jelenthet, ha valamilyen virtualizációt használunk fel a környezet kialakításához. Virtualizált környezetben könnyebb a telepítés, skálázás, és a monitorozás is egyszerűbb lehet.

- **Docker:** Egy konténer virtualizációs eszköz, amelynek segítségével egy adott kernel alatt több különböző környezettel rendelkező alkalmazásokat futtató környezetet hozhatunk létre. A Docker egy szeparált fájlrendszert hoz létre a host gépen, és abban hajt végre műveleteket. Készíthetünk vele előre elkészített alkalmazás környezeteket, és szolgáltatásokat, ami ideálissá teszi microservice architektúrák létrehozásánál. A Docker konténerek segítségével egyszerűen telepíthetjük, skálázhatjuk, és fejleszthetjük a rendszert.
- **libvirt:** Többféle virtualizációs technológiával együtt működő eszköz, amivel könnyedén irányíthatjuk a virtuális gépeket, és a virtualizálás komolyabb részét el absztahálja. Támogat KVM-em, XEN-t, VirtualBox-ot LXC, és sok más virtualizáló eszközt. Ezzel az eszközzel a környezet kialakítását szabhatjuk meg, tehát a hardware-eserőforrások megosztásában nyújt nagy segítséget.
- **kvm:** A KVM egy kernel szintű virtualizációs eszköz, amivel virtuális gépeket tudunk készíteni. Processzor szintjén képes szétválasztani az erőforrásokat, és ezzel szeparált környezeteket létrehozni. Virtualizál a processzoron kívül hálózati kártyát, háttértárat, grafikus meghajtót, és sok mászt. A KVM egy nyílt forráskódú projekt és létrehozhatunk vele Linux és Windows gépeket is egyaránt.
- **Akármilyen cloud:** Ha virtualizációról beszélünk, akkor adja magát hogy a CCloud-os környezeteket is ide értsük. Egy microservice architektúrájú programot a legcél-szerűbb valamilyen Cloud-os környezetben létrehozni, mivel egy ilyen környezetnek definíciója szerint tartalmaznia kell egy virtualizációs szintet, megosztott erőforrások, monitorozást, és egyfajta leltárat a futó példányokról. Ennek megfelelően a microservice architektúra minden környezeti feltételét lefedi, csupán a szolgáltatásokat, business logikát, és az interfészeket kell elkészítenünk. Jellemzően a Cloud-os

környezetek tartalmaznak terhelés elosztást, és skálázási megoldást is, amivel szintén erősítik a szolgáltatás alapú architektúrákat. Ilyen környezet lehet az Amazon, Microsoft Azure, Google App Engine, OpenStack, és sokan mások.

3.0.12. Service registry:

Számon kell tartani, hogy milyen szolgáltatások elérhetők, milyen címen és hány példányban az architektúránkban, és ehhez valamilyen szolgáltatás nyilvántartási eszközt kell használnunk.

- **Eureka:** Az Eureka a Netflix fejlesztése, egy AWS környezetben működő terhelés elosztó alkalmazás, ami figyeli a felvett szolgáltatásokat, és így mint nyilvántartás is megfelelő. A kommunikációt és a kapcsolatot egy Java nyelven írt szerver és kliens biztosítja, ami a teljes logikát megvalósítja. EGYütt működik a Netflix által fejlesztett Asgard nevezetű alkalmazással ami az AWS szolgáltatásokhoz való hozzáférést segíti. Ugyan ez az eszköz erősen optimalizált az Amazon Cloud szolgáltatásaihoz, de a leírás alapján megállja a helyét önállóan is. Mivel nyílt forráskódú, mintát szolgáltat egyéb alkalmazásoknak is.
- **Consul:** Korábban már említettem ezt az eszközt, mivel abban segít, hogy felismerjék egymást a szolgáltatások. A kapcsolatot vizsgáló és felderítő logikán kívül tartalmaz egy nyilvántartást is a beregisztrált szolgáltatásokról, amiknek az állapotát is vizsgálhatjuk.
- **Apache Zookeeper:** A Zookeeper egy központosított szolgáltatás konfigurációs adatok és hálózati adatok karbantartására, ami támogatja az elosztott működést, és a szerverek csoportosítását. Az alkalmazást elosztott alkalmazás fejlesztésre, és komplex rendszer felügyeletére és telepítés segítésére tervezték. A consulhoz hasonlóan működik, és a feladata is ugyan az.

3.0.13. Monitorozás, loggolás:

Ha már megépítettük a microservice architektúrát, akkor meg kell bizonyosodnunk róla, hogy minden megfelelően működik, és minden rendben zajlik a szolgáltatásokkal. Ehhez többféle módon és többféle eszközzel is hozzáférhetünk, mivel az alkalmazás hibákat egy log szerver, a környezeti problémákat egy monitorozó szerver tudja megfelelően megmutatni számunkra.

- **Zabbix:** A Zabbix egy sok területen felhasznált, több 10 ezer szervert párhuzamosan megfigyelni képes, akármilyen adatot tárolni képes monitorozó alkalmazás, ami képes elosztott működésre, és virtuális környezetekben jól használható. Ágens nélküli és ágenses adatgyűjtésre is képes, és az adatokat különböző módokon képes megjeleníteni (földrajzi elhelyezkedés, gráfos megjelenítés, stb.). Nem egészen a microservice architektúrákhoz lett kialakítva, de egy elég általános eszköz, hogy felhasználható legyen ilyen célra is.

- **Kibana + LogStash:** A Kibana egy ingyenes adatmegjelenítő és adatfeldolgozó eszköz, amit az elasticsearch fejlesztett ki, és a logstash pedig egy log server, amivel tárolhatjuk a loggolási adatainkat, és egyszerűen kereshetünk benne. Kifejezetten adatfeldolgozásra szolgál mind a két eszköz, és közvetlenül együttműködnek az elasticsearch alkalmazással.
- **Sensu:** A Sensu egy egyszerű monitorozó eszköz, amivel megfigyelhetjük a szervereinket. Támogatja Ansible Chef, Puppet használatát, és támogatja a Plugin szerű bővíthetőséget. A felülete letisztult és elég jó áttekintést ad a szerverek állapotáról. Figyel a dinamikus változásokra, és gyorsan lekezeli a változásokkal járó riasztásokat. Ezek a tulajdonságai teszik a Cloud-okban könnyen és hatékonyan felhasználhatóvá.
- **Cronitor:** Ez a monitorozó eszköz mikro-szolgáltatások és cron job-ok megfigyelésére lett kifejlesztve, HTTP-n keresztül kommunikál, és a szolgáltatások állapotát figyeli. Nem túl széleskörű eszköz, azonban ha csak a szolgáltatások állapota érdekel hasznos lehet, és segíthet a Service Registry képzésében is.
- **Ruxit:** Egy Cloud-osított monitorozó eszköz, amivel teljesítmény monitorozást, elérhetőség monitorozást, és figyelmeztetés küldést végezhetünk. Az benne a különleges, hogy mesterséges intelligencia figyeli a szervereket, és kianalizálja a server állapotát, és a figyelmeztetéseket is követi. Könnyen skálázható, és használat alapú bérezése van. Ez a választás akkor jön jól, ha olyan feladatot szánunk az alkalmazásunknak, ami esetleg időben nagyon változó terhelést mutat, és az itt kapot riasztások szerint akarunk skálázni.

3.0.14. Hivatkozások:

- [1] Sebastián Peyrott: [An Introduction to Microservices, Part 3: The Service Registry](#)
- [2] Mrina Natarajan: [3 golden rules of microservices deployments](#)
- [3] ElasticBox: [ElasticBox](#)
- [4] ElasticBox: [Kubernetes](#)
- [5] Jenkins: [ElasticBox CI](#)
- [6] Chris Richardson: [Choosing a Microservices Deployment Strategy](#)
- [7] Zohar Arad: [Effectively Monitor Your Micro-Service Architectures](#)
- [8] Nemeth Gergely: [Monitoring Microservices](#)
- [9] Cronitor: [Monitoring Microservices](#)
- [10] Ruxit: [Microservice monitoring](#)
- [11] Chris Richardson: [Service registry pattern](#)
- [12] David Liu: [Eureka at glance](#)
- [13] Hashicorp: [Consul](#)
- [14] Apache: [Apache Zookeeper](#)
- [15] Jenkins: [Jenkins](#)
- [16] Netflix: [Eureka](#)
- [17] Chef Software Inc.: [Chef](#)

- [18] Ansible: [Ansible](#)
- [19] SlatStack inc.: [SaltStack](#)
- [20] Amazon: [Elastic Load Balancing](#)
- [21] HAProxy: [The Reliable, High Performance TCP/HTTP Load Balancer](#)
- [22] Fideloper LLC: [Load Balancing with HAProxy](#)
- [23] Nginx: [Nginx](#)
- [24] Docker: [Docker](#)
- [25] Libvirt: [libvirt: The virtualization API](#)
- [26] KVM: [Kernel Virtual Machine](#)
- [27] Zabbix: [What is Zabbix](#)
- [28] Elasticsearch: [Kibana](#)
- [29] Elasticsearch: [LogStash](#)
- [30] Cronitor.io: [Cronitor](#)
- [31] dynatrace: [Ruxit overview](#)

4. fejezet

A Markdown-sablon használata

Ebben a fejezetben röviden, implicit módon bemutatjuk a sablon használatának módját, ami azt jelenti, hogy sablon használata ennek a dokumentumnak a forráskódját tanulmányozva válik teljesen világossá. Amennyiben a szoftver-keretrendszer telepítve van, a sablon alkalmazása és a dolgozat szerkesztése Markdownban a sablon segítségével tapasztalataink szerint jóval hatékonyabb, mint egy WYSWYG (*What You See is What You Get*) típusú szövegszerkesztő esetén (pl. Microsoft Word, OpenOffice).

A \LaTeX tördelőrendszer használatához képest a Markdown nyelv több megszorítást is tartalmaz, így az elkészített dokumentum általában kevésbé testreszabható. Cserébe viszont a Markdownban készült dokumentumok exportálhatók HTML vagy e-könyv (EPUB) formátumba is.

4.1. Ábrák és táblázatok

A képeket a veszteségmentes PNG, valamint a veszteséges JPEG formátumban érdemes elmenteni.

Az egyes képek mérete általában nem, de sok kép esetén a dokumentum összmérete így már szignifikáns is lehet. A dokumentumban felhasznált képfájlokat a dokumentum forrása mellett érdemes tartani, archiválni, mivel ezek hiányában a dokumentum nem fordul újra. Ha lehet, a vektorgrafikus képeket vektorgrafikus formátumban is érdemes elmenteni az újrafelhasználhatóság (az átszerkeszthetőség) érdekében.

A képek beillesztésére a Markdown eszközök (`??`, `??+`) fejezetben mutattunk be példát (`??`, `??+`). Az előző mondatban egyúttal az automatikusan feloldódó ábrahivatkozásra is láthatunk példát.

A táblázatok használatára az alábbi táblázat mutat példát.

Órajel	Frekvencia	Cél pin
CLKA	100 MHz	FPGA CLK0
CLKB	48 MHz	FPGA CLK1
CLKC	20 MHz	Processzor

Órajel	Frekvencia	Cél pin
CLKD	25 MHz	Ethernet chip
CLKE	72 MHz	FPGA CLK2
XBUF	20 MHz	FPGA CLK3

4.1. táblázat. Az órajel-generátor chip órajel-kimenetei.

4.2. Felsorolások és listák

Számozatlan felsorolásra mutat példát a jelenlegi bekezdés:

- *első bajusz*: ide lehetne írni az első elem kifejtését,
- *második bajusz*: ide lehetne írni a második elem kifejtését,
- *ez meg egy szakáll*: ide lehetne írni a harmadik elem kifejtését.

Számozott felsorolást is készíthetünk az alábbi módon:

1. *első bajusz*: ide lehetne írni az első elem kifejtését, és ez a kifejtés így néz ki, ha több sorosra sikeredik,
2. *második bajusz*: ide lehetne írni a második elem kifejtését,
3. *ez meg egy szakáll*: ide lehetne írni a harmadik elem kifejtését.

A felsorolásokban sorok végén vessző, az utolsó sor végén pedig pont a szokásos írásjel. Ez alól kivételt képezhet, ha az egyes elemek több teljes mondatot tartalmaznak.

Listákban a dolgozat szövegétől elkülönítendő kódrészleteket, programsorokat, pszeudo-kódokat jeleníthetünk meg.

```
* *első bajusz:* ide lehetne írni az első elem kifejtését, és ez a kifejtés így néz ki,
* *második bajusz:* ide lehetne írni a második elem kifejtését,
* *ez meg egy szakáll:* ide lehetne írni a harmadik elem kifejtését.
```

A listákban definiálható a használt programozási nyelv (pl. `java`, `latex`, `bash` stb.).

4.3. Képletek

Ha egy formula nem túlságosan hosszú, és nem akarjuk hivatkozni a szövegből, mint például a $e^{i\pi} + 1 = 0$ képlet, *szövegközi képletként* szokás leírni. Csak, hogy másik példát is lássunk, az $U_i = -d\Phi/dt$ Faraday-törvény a $\text{rot}E = -\frac{dB}{dt}$ differenciális alakban adott Maxwell-egyenlet felületre vett integráljából vezethető le. Látható, hogy a L^AT_EX-fordító a sorközöket betartja, így a szöveg szedése esztétikus marad szövegközi képletek használata esetén is.

Képletek esetén az általános konvenció, hogy a kisbetűk skalárt, a kis félkövér betűk (\mathbf{v}) oszlopvektort – és ennek megfelelően \mathbf{v}^T sorvektort – a kapitális félkövér betűk (\mathbf{V}) mátrixot jelölnek. Ha ettől el szeretnénk térni, akkor az alkalmazni kívánt jelölésmódot célszerű külön alfejezetben definiálni. Ennek megfelelően, amennyiben \mathbf{y} jelöli a mérések vektorát, ϑ a paraméterek vektorát és $\hat{\mathbf{y}} = \mathbf{X}\vartheta$ a paraméterekben lineáris modellt, akkor a `*Least-Squares` értelemben optimális paraméterbecslő $\hat{\vartheta}_{LS} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$ lesz.

Emellett kiemelt, sorszámozott képleteket is megadhatunk, ennél az `equation` és a `eqnarray` környezetek helyett a korszerűbb `align` környezet alkalmazását javasoljuk (több okból, különféle problémák elkerülése végett, amelyekre most nem térünk ki). Tehát

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \quad (4.1)$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}, \quad (4.2)$$

ahol \mathbf{x} az állapotvektor, \mathbf{y} a mérések vektora és \mathbf{A} , \mathbf{B} és \mathbf{C} a rendszert leíró paramétermátrixok. Figyeljük meg, hogy a két egyenletben az egyenlőségjelek egymáshoz igazítva jelennek meg, mivel a mindkettőt az `&` karakter előzi meg a kódban. Lehetőség van számozatlan kiemelt képlet használatára is, például

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}.$$

Mátrixok felírására az $\mathbf{A}\mathbf{x} = \mathbf{b}$ inhomogén lineáris egyenlet részletes kifejtésével mutatunk példát:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}. \quad (4.3)$$

A `\frac` utasítás hatékonyságát egy általános másodfokú tag átviteli függvényén keresztül mutatjuk be, azaz

$$W(s) = \frac{A}{1 + 2T\xi s + s^2 T^2}. \quad (4.4)$$

A matematikai mód minden szimbólumának és képességének a bemutatására természetesen itt nincs lehetőség, de gyors referenciaként hatékonyan használhatók a következő linkek:

- http://www.artofproblemsolving.com/LaTeX/AoPS_L_GuideSym.php,
- <http://www.ctan.org/tex-archive/info/symbols/comprehensive/symbols-a4.pdf>,

- <ftp://ftp.ams.org/pub/tex/doc/amsmath/short-math-guide.pdf>.

Ez pedig itt egy magyarázat, hogy miért érdemes `align` környezetet használni: <http://texblog.net/latex-archive/math/eqnarray-align-environment/>.

4.4. Irodalmi hivatkozások

Az irodalmi hivatkozások alkalmazására javasolt a BiBTeX használata, ezért ez a sablon is ezt támogatja. Ebben az esetben egy külön szöveges adatbázisban definiáljuk a forrásmunkákat, és egy külön stílusfájl határozza meg az irodalomjegyzék kinézetét. Ez, összhangban azzal, hogy külön formátumkonvenció határozza meg a folyóirat-, a könyv-, a konferenciakik stb. hivatkozások kinézetét az irodalomjegyzékben (a sablon használata esetén ezzel nem is kell foglalkoznia a hallgatónak, de az eredményt célszerű ellenőrizni). A felhasznált hivatkozások adatbázisa egy `.bib` kiterjesztésű szöveges fájl, amelynek szerkezetét az alábbi kódrészlet demonstrálja. A forrásmunkák bevitelekor a sor végi vesszők külön figyelmet igényelnek, mert hiányuk a BiBTeX-fordító hibaüzenetét eredményezi. A forrásmunkákat típus szerinti kulcsszó vezeti be (`@book` könyv, `@inproceedings` konferenciakiadványban megjelent cikk, `@article` folyóiratban megjelent cikk, `@techreport` valamelyik egyetem gondozásában megjelent műszaki tanulmány, `@manual` műszaki dokumentáció esetén stb.). Nemcsak a megjelenés stílusa, de a kötelezően megadandó mezők is típusról-típusra változnak. Egy jól használható referencia a <http://en.wikipedia.org/wiki/BibTeX> oldalon található.

```
@BOOK{Wettl04,
  author = "Ferenc Wettl and Gyula Mayer and Péter Szabó",
  title = "\LaTeX~kézikönyv",
  publisher = "Panem Könyvkiadó",
  year = 2004
}

@ARTICLE{Candy86,
  author = "James C. Candy",
  title = "Decimation for Sigma Delta Modulation",
  journal = "{IEEE} Trans.\ on Communications",
  volume = 34,
  number = 1,
  pages = "72--76",
  month = jan,
  year = 1986,
}

@INPROCEEDINGS{Lee87,
  author = "Wai L. Lee and Charles G. Sodini",
  title = "A Topology for Higher Order Interpolative Coders",
  booktitle = "Proc.\ of the IEEE International Symposium on Circuits and Systems",
```

```

    year = 1987,
    vol = 2,
    month = may # "~4--7",
    address = "Philadelphia, PA, USA",
    pages = "459--462"
}
@PHDTHESIS{KissPhD,
    author = "Peter Kiss",
    title = "Adaptive Digital Compensation of Analog Circuit Imperfections for Cascaded D
    school = "Technical University of Timi\c{s}oara, Romania",
    month = apr,
    year = 2000
}
@MANUAL{Schreier00,
    author = "Richard Schreier",
    title = "The Delta-Sigma Toolbox v5.2",
    organization = "Oregon State University",
    year = 2000,
    month = jan,
    note = "\newline URL: http://www.mathworks.com/matlabcentral/fileexchange/"
}
@MISC{DipPortal,
    author = "Budapesti {M}űszaki és {G}azdaságtudományi {E}gyetem, {V}illamosmérnöki É
    title = "{D}iplomaterv portál (2011 február 26.)",
    howpublished = "\url{http://diplomaterv.vik.bme.hu/}",
}

```

A stílusfájl egy .sty kiterjesztésű fájl, de ezzel lényegében nem kell foglalkozni, mert vannak beépített stílusok, amelyek jól használhatók. Ez a sablon a BiBTeX-et használja, a hozzá tartozó adatbázisfájl a mybib.bib fájl. Megfigyelhető, hogy az irodalomjegyzéket a dokumentum végére (a \end{document} utasítás elé) beillesztett \bibliography{mybib} utasítással hozhatjuk létre, a stílusát pedig ugyanitt a \bibliographystyle{plain} utasítással adhatjuk meg. Ebben az esetben a plain előre definiált stílust használjuk (a sablonban is ezt állítottuk be). A plain stíluson kívül természetesen számtalan más előre definiált stílus is létezik. Mivel a .bib adatbázisban ezeket megadtuk, a BiBTeX-fordító is meg tudja különböztetni a szerzőt a címtől és a kiadótól, és ez alapján automatikusan generálódik az irodalomjegyzék a stílusfájl által meghatározott stílusban.

Az egyes forrásmunkákra a szövegből továbbra is a @[...] paranccsal tudunk hivatkozni, így a fenti kódrészlet esetén a hivatkozások rendre [Wett104], [Candy86], [Lee87], [KissPhD], [Schreier00] és [DipPortal]. Az irodalomjegyzékben alapértelmezésben csak azok a forrásmunkák jelennek meg, amelyekre található hivatkozás a szövegben, és ez így alapvetően helyes is, hiszen olyan forrásmunkákat nem illik az irodalomjegyzékbe

írni, amelyekre nincs hivatkozás.

Mivel a fordítási folyamat során több lépésben oldódnak fel a szimbólumok, ezért gyakran többször (TeXLive és TeXnicCenter esetén 2-3-szor) is le kell fordítani a dokumentumot. Ilyenkor ez első 1-2 fordítás esetleg szimbólum-feloldásra vonatkozó figyelmeztető üzenettel zárul. Ha hibaüzenettel zárul bármelyik fordítás, akkor nincs értelme megismételni, hanem a hibát kell megkeresni. A `.bib` fájl megváltoztatáskor sokszor nincs hatása a változtatásnak azonnal, mivel nem mindig fut újra a BibTeX fordító. Ezért célszerű a változtatás után azt manuálisan is lefuttatni (TeXnicCenter esetén **Build/BibTeX**).

Hogy a szövegbe ágyazott hivatkozások kinézetét demonstráljuk, itt most sorban meghivatkozunk a (?), (?), (?), (?) és az (?) forrásmunkát, valamint az (?) weboldalt.

Megjegyzendő, hogy az ékezetes magyar betűket is tartalmazó `.bib` fájl az `inputenc` csomaggal betöltött `latin2` betűkészlet miatt fordítható. Ugyanez a `.bib` fájl hibaüzenettel fordul egy olyan dokumentumban, ami nem tartalmazza a `\usepackage[latin2]{inputenc}` sort. Speciális igény esetén az irodalmi adatbázis általánosabb érvényűvé tehető, ha az ékezetes betűket speciális latex karakterekkel helyettesítjük a `.bib` fájlban, pl. `á` helyett `\'a`-t vagy `ő` helyett `\H{o}`-t írunk.

4.5. A dolgozat szerkezete és a forrásfájlok

A diplomatervsablon (a kari irányelvek szerint) az alábbi fő fejezetekből áll:

- *tájékoztató* a szakdolgozat/diplomaterv szerkezetéről, ami a végső dolgozathoz tartozó, valamint a *feladatkiírás* (`guideline.md`), a dolgozat nyomtatott verziójában ennek a helyére kerül a tanszék által kiadott, a tanszékvezető által aláírt feladatkiírás, a dolgozat elektronikus verziójába pedig a feladatkiírás egyáltalán ne kerüljön bele, azt külön tölti fel a tanszék a diplomaterv-honlapra,
- *címoldal* (generált),
- *tartalomjegyzék* (generált),
- a diplomatervező *nyilatkozata* az önálló munkáról (generált),
- 1-2 oldalas tartalmi *összefoglaló* magyarul és angolul, illetve elkészíthető még további nyelveken is (`abstract.md`),
- *fejezetek*: *bevezetés* (feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása, `chapter1.md`), a feladatkiírás pontosítása és részletes elemzése, előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések, a tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása, a megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek (`chapter{2..n}.md`),
- összefoglalás (`summary.md`),
- esetleges *köszönetnyilvánítások* (`acknowledgement.md`),
- részletes és pontos *irodalomjegyzék* (generált),
- *függelékek* (`appendices.md`).

5. fejezet

Összefoglaló

A diplomaterv összefoglaló fejezete.

Köszönetnyilvánítás

A köszönetnyilvánítás nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Táblázatok jegyzéke

Ábrák jegyzéke

A. függelék

Függelék

A.1. Válasz az „Élet, a világmindenség, meg minden” kérdésre

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42.$$

A Faraday-indukciós törvényből levezetve

$$\text{rot} E = -\frac{dB}{dt} \quad \longrightarrow \quad U_i = \oint_{\mathbf{L}} \mathbf{E} d\mathbf{l} = -\frac{d}{dt} \int_A \mathbf{B} d\mathbf{a} = 42.$$