Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Online könyvkölcsönző alkalmazás készítése a Simonyi Károly Szakkollégium számára

Szakdolgozat

Készítette
Fehér János

Konzulens
dr. Ekler Péter

2020. december 5.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Fehér János*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot 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, 2020. december 5.

Fehér János
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 LaTeX alapú, a *TeXLive* TeX-implementációval és a PDF-LaTeX fordítóval működőképes.

Abstract

This document is a LaTeX-based 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. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX 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.

A Simonyi Károly Szakkollégiumban jelenleg üzemel egy könyvtár, ahonnan a hallgatók különféle tankönyveket és szórakoztató irodalmat van lehetőségük kölcsönözni. Ennek az adminisztrációja egy megosztott Google Docs táblázaton keresztül történik, ami azonban a publikum felé nem nyilvános, a hallgatók a könyvtárban lévő könyvek elérhetőségét csak személyesen, vagy az üzemeltetőknek írt emailen keresztül tudják ellenőrizni. Ez nagyban hátráltatja, hogy a hallgatókban tudatosuljon a könyvtár létezése, valamint megnehezíti annak karbantartását és a kölcsönzés menetét is.

Ezen helyzet adta a motivációt arra, hogy készítsek egy, a hallgatók által könnyen hozzáférhető, és az üzemeltetők által kényelmesen menedzselhető webes alkalmazást a szakkollégium számára. A cél egy reszponzív, web alapú felület tervezése és elkészítése a könyvtár könnyű használhatósága érdekében. (TODO: ezt talán szebben fogalmazni)

TODO: szebben fejezetek: - Használt technológiák kiválasztása - frontend, backend, db, rest/graphql - alkalmazás felépítése - adatbázis - db séma (dbdiagram export, prisma schema, prisma migrate) - prisma (schema file, migrations) - backend - nextjs file-based routing, middleware-ek - frontend - components/ mappa, file based routing - kódmegosztás - lib/ mappa, prisma típusok frontenden - alkalmazás működése - authentikáció, authorizáció - adatlekérés a backendtől: useSWR - prisma - kosár - order leadása, menedzselése - keresés: postgres full text search, kihívások, megoldások - fájlfeltöltés: amazon S3

- tesztek (???) - CI - github actions, eslint - hosting, jövőbeli deploy (kubernetes, docker) - jövőbeli bővítési lehetőségek - real-time chat websocket segítségével - hosted / dockerben futó fuzzy search (elastic, algolia) - authsch integráció a user-pass mellé

2. fejezet

A LaTeX-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 LaTeX-ben a sablon segítségével tapasztalataink szerint jóval hatékonyabb, mint egy WYSWYG (*What You See is What You Get*) típusú szövegszerkesztő esetén (pl. Microsoft Word, OpenOffice).

2.1. Címkék és hivatkozások

A LaTeX dokumentumban címkéket (`\label`) rendelhetünk ábrákhoz, táblázatokhoz, fejezetekhez, listákhoz, képletekhez stb. Ezekre a dokumentum bármely részében hivatkozhatunk, a hivatkozások automatikusan feloldásra kerülnek.

A sablonban makrókat definiáltunk a hivatkozások megkönnyítéséhez. Ennek megfelelően minden ábra (*figure*) címkéje `fig:` kulcsszóval kezdődik, míg minden táblázat (*table*), képlet (*equation*), fejezet (*section*) és lista (*listing*) rendre a `tab:`, `eq:`, `sec:` és `lst:` kulcsszóval kezdődik, és a kulcsszavak után tetszőlegesen választott címke használható. Ha ezt a konvenciót betartjuk, akkor az előbbi objektumok számára rendre a `\figref`, `\tabref`, `\eqref`, `\sectref` és `\listref` makrókkal hivatkozhatunk. A makrók paramétere a címke, amelyre hivatkozunk (a kulcsszó nélkül). Az összes említett hivatkozástípus, beleértve az `\url` kulcsszóval bevezetett web-hivatkozásokat is a `hyperref`[1] csomagnak köszönhetően aktívak a legtöbb PDF-nézegetőben, rájuk kattintva a dokumentum megfelelő oldalára ugrik a PDF-néző vagy a megfelelő linket megnyitja az alapértelmezett böngészővel. A `hyperref` csomag a kimeneti PDF-dokumentumba könyvjelzőket is készít a tartalomjegyzékből. Ez egy szintén aktív tartalomjegyzék, amelynek elemeire kattintva a nézegető behozza a kiválasztott fejezetet.

2.2. Ábrák és táblázatok

Használjunk vektorgrafikus ábrákat, ha van rá módunk. PDFLaTeX használata esetén PDF formátumú ábrákat lehet beilleszteni könnyen, az EPS (PostScript) vektorgrafikus képformátum beillesztését a PDFLaTeX közvetlenül nem támogatja (de lehet konvertálni, lásd később). Ha vektorgrafikus formában nem áll rendelkezésünkre az ábra, akkor a veszteségmentes PNG, valamint a veszteséges JPEG formátumban érdemes elmenteni. Figyeljünk arra, hogy ilyenkor a képek felbontása elég nagy legyen ahhoz, hogy nyomtatásban is megfelelő minőséget nyújtson (legalább 300 dpi javasolt). A

[1] Segítségével a dokumentumban megjelenő hivatkozások nem csak dinamikussá válnak, de színezhetők is, bővebbet erről a csomag dokumentációjában találunk. Ez egyúttal egy példa lábjegyzet írására.

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.

Kapcsolási rajzok legtöbbször kimásolhatók egy vektorgrafikus programba (pl. CorelDraw) és onnan nagyobb felbontással raszterizálva kimenthatők PNG formátumban. Ugyanakkor kiváló ábrák készíthetők Microsoft Visio vagy hasonló program használatával is: Visio-ból az ábrák közvetlenül PDF-be is menthetők.

Lehetőségeink Matlab ábrák esetén:

- Képernyőlopás (*screenshot*) is elfogadható minőségű lehet a dokumentumban, de általában jobb felbontást is el lehet érni más módszerrel.
- A Matlab ábrát a `File/Save As` opcióval lementhetjük PNG formátumban (ugyanaz itt is érvényes, mint korábban, ezért nem javasoljuk).
- A Matlab ábrát az `Edit/Copy figure` opcióval kimásolhatjuk egy vektorgrafikus programba is és onnan nagyobb felbontással raszterizálva kimenthatjük PNG formátumban (nem javasolt).
- Javasolt megoldás: az ábrát a `File/Save As` opcióval EPS *vektorgrafikus* formátumban elmentjük, PDF-be konvertálva beillesztjük a dolgozatba.

Az EPS kép az `epstopdf` programmal[2] konvertálható PDF formátumba. Célszerű egy batch-fájlt készíteni az összes EPS ábra lefordítására az alábbi módon (ez Windows alatt működik).

```
@echo off
for %%j in (*.eps) do (
  echo converting file "%%j"
  epstopdf "%%j"
)
echo done .
```

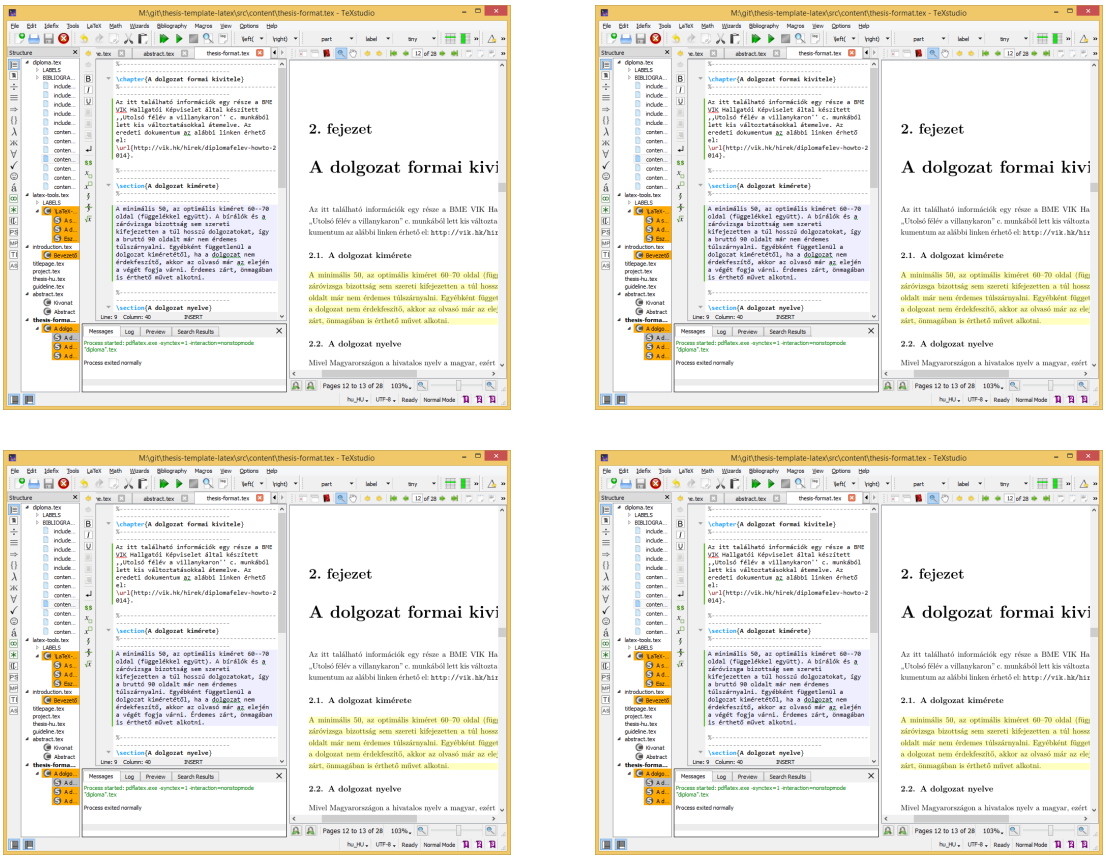
Egy ilyen parancsfájlt (`convert.cmd`) elhelyeztük a sablon `figures\eps` könyvtárába, így a felhasználónak csak annyi a dolga, hogy a `figures\eps` könyvtárba kimenti az EPS formátumú vektorgrafikus képet, majd lefuttatja a `convert.cmd` parancsfájlt, ami PDF-be konvertálja az EPS fájlt.

Ezek után a PDF-ábrát ugyanúgy lehet a dokumentumba beilleszteni, mint a PNG-t vagy a JPEG-et. A megoldás előnye, hogy a lefordított dokumentumban is vektorgrafikusan tárolódik az ábra, így a mérete jóval kisebb, mintha raszterizáltuk volna beillesztés előtt. Ez a módszer minden – az EPS formátumot ismerő – vektorgrafikus program (pl. CorelDraw) esetén is használható.

A képek beillesztésére a **??**. **??+**ben mutattunk be példát (**??**. **??+**). Az előző mondatban egyúttal az automatikusan feloldódó ábrahivatkozásra is láthatunk példát. Több képfájlt is beilleszthetünk egyetlen ábrába. Az egyes képek közötti horizontális és vertikális margót metrikusan szabályozhatjuk (2.1. ábra). Az ábrák elhelyezését számtalan tipográfiai szabály egyidejű teljesítésével a fordító maga végzi, a dokumentum írója csak preferenciáit jelezheti a fordító felé (olykor ez bosszúságot is okozhat, ilyenkor pl. a kép méretével lehet játszani).

A táblázatok használatára a 2.1 táblázat mutat példát. A táblázatok formázásához hasznos tanácsokat találunk a `booktabs` csomag dokumentációjában.

[2] a korábban említett LaTeX-disztribúciókban megtalálható



2.1. ábra. Több képfájl beillesztése esetén térközöket is érdemes használni.

2.3. Felsorolások és listák

Számozatlan felsorolásra mutat példát a jelenlegi bekezdés:

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

Számozott felsorolást is készíthetünk az alábbi módon:

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

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 (2.1. kódrészlet).

```
\begin{enumerate}
  \item \emph{első bajusz:} ide lehetne írni az első elem kifejését,
  és ez a kifejtés így néz ki, ha több sorosra sikeredik,
  \item \emph{második bajusz:} ide lehetne írni a második elem kifejését,
```

Órajel	Frekvencia	Cél pin
CLKA	100 MHz	FPGA CLK0
CLKB	48 MHz	FPGA CLK1
CLKC	20 MHz	Processzor
CLKD	25 MHz	Ethernet chip
CLKE	72 MHz	FPGA CLK2
XBUF	20 MHz	FPGA CLK3

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

```
  \item \emph{ez meg egy szakáll:} ide lehetne írni a harmadik elem kifejését.
\end{enumerate}
```

2.1. lista. A fenti számozott felsorolás LaTeX-forráskódja

A lista keretét, háttérszínét, egész stílusát megválaszthatjuk. Ráadásul különféle programnyelveket és a nyelveken belül kulcsszavakat is definiálhatunk, ha szükséges. Erről bővebbet a `listings` csomag hivatalos leírásában találhatunk.

2.4. 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 $\operatorname{rot} E = -\frac{dB}{dt}$ differenciális alakban adott Maxwell-egyenlet felületre vett integráljából vezethető le. Látható, hogy a LaTeX-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

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \tag{2.1}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}, \tag{2.2}$$

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

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},\\ \mathbf{y} &= \mathbf{C}\mathbf{x}.\end{aligned}$$

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

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}. \tag{2.3}$$

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

$$W(s) = \frac{A}{1 + 2T\xi s + s^2T^2}. \tag{2.4}$$

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/maths/eqnarray-align-environment/`.

2.5. Irodalmi hivatkozások

Egy LaTeX dokumentumban az irodalmi hivatkozások definíciójának két módja van. Az egyik a `\thebibliograhy` környezet használata a dokumentum végén, az `\end{document}` lezárás előtt.

```
\begin{thebibliography}{9}

\bibitem{Lamport94} Leslie Lamport, \emph{\LaTeX: A Document Preparation System}.
Addison Wesley, Massachusetts, 2nd Edition, 1994.

\end{thebibliography}
```

Ezek után a dokumentumban a `\cite{Lamport94}` utasítással hivatkozhatunk a forrásra. A fenti megadás viszonylag kötetlen, a szerző maga formázza az irodalomjegyzéket (ami gyakran inkonzisztens eredményhez vezet).

Egy sokkal professzionálisabb módszer 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 konferenciacikk- stb. hivatkozások kinézetét az irodalomjegyzékben (a sablon használata esetén ezzel nem is kell foglalkoznia a hallgatónak, de az eredményt célszerű ellenőrizni). felhasznált hivatkozások adatbázisa egy `.bib` kiterjesztésű szöveges fájl, amelynek szerkezetét a A 2.2 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},
  publisher = {Panem Könyvkiadó},
  title     = {\LaTeX~kézikönyv},
  year      = {2004},
}

@article{Candy86,
  author       = {James C. Candy},
  journaltitle = {{IEEE} Trans.\ on Communications},
  month        = {01},
  note         = {\doi{10.1109/TCOM.1986.1096432}},
  number       = {1},
  pages        = {72--76},
  title        = {Decimation for Sigma Delta Modulation},
  volume       = {34},
  year         = {1986},
}

@inproceedings{Lee87,
  author    = {Wai L. Lee and Charles G. Sodini},
  booktitle = {Proc.\ of the IEEE International Symposium on Circuits and Systems},
  location  = {Philadelphia, PA, USA},
  month     = {05~4--7},
  pages     = {459--462},
  title     = {A Topology for Higher Order Interpolative Coders},
  vol       = {2},
  year      = {1987},
}

@thesis{KissPhD,
  author      = {Peter Kiss},
  institution = {Technical University of Timi\c{s}oara, Romania},
  month       = {04},
  title       = {Adaptive Digital Compensation of Analog Circuit Imperfections for Cascaded Delta-
    Sigma Analog-to-Digital Converters},
  type        = {phdthesis},
  year        = {2000},
}

@manual{Schreier00,
  author       = {Richard Schreier},
  month        = {01},
  note         = {\url{http://www.mathworks.com/matlabcentral/fileexchange/}},
  organization = {Oregon State University},
  title        = {The Delta-Sigma Toolbox v5.2},
  year         = {2000},
}

@misc{DipPortal,
  author       = {{Budapesti űMszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai
    Kar}},
  howpublished = {\url{http://diplomaterv.vik.bme.hu/}},
  title        = {Diplomaterv portál (2011. február 26.)},
}

@incollection{Mkrtychev:1997,
  author    = {Mkrtychev, Alexey},
  booktitle = {Logical Foundations of Computer Science},
  doi       = {10.1007/3-540-63045-7_27},
  editor    = {Adian, Sergei and Nerode, Anil},
  isbn      = {978-3-540-63045-6},
  pages     = {266-275},
  publisher = {Springer Berlin Heidelberg},
  series    = {Lecture Notes in Computer Science},
  title     = {Models for the logic of proofs},
  url       = {http://dx.doi.org/10.1007/3-540-63045-7_27},
  volume    = {1234},
  year      = {1997},
```

```
}
```

2.2. lista. Példa szöveges irodalomjegyzék-adatbázisra BibTEX használata esetén.

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 `\cite` paranccsal tudunk hivatkozni, így a 2.2. kódrészlet esetén a hivatkozások rendre `\cite{Wettl04}`, `\cite{Candy86}`, `\cite{Lee87}`, `\cite{KissPhD}`, `\cite{Schreirer00}`, `\cite{Mkrtychev:1997}` és `\cite{DipPortal}`. Az egyes forrásmunkák sorszáma az irodalomjegyzék bővítésekor változhat. Amennyiben az aktuális számhoz illeszkedő névelőt szeretnénk használni, használjuk az `\acite{}` parancsot.

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 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 (TeXstudio esetén `Tools/Bibliography`).

Hogy a szövegbe ágyazott hivatkozások kinézetét demonstráljuk, itt most sorban meghivatkozzuk a [7], [2], [4], [3], [6] és az [5][3] forrásmunkát, valamint az [1] 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.

Irodalomhivatkozásokat célszerű először olyan szolgáltatásokban keresni, ahol jó minőségű bejegyzések találhatók (pl. ACM Digital Library,[4] DBLP,[5] IEEE Xplore,[6] SpringerLink[7]) és csak ezek után használni kevésbé válogatott forrásokat (pl. Google Scholar[8]). A jó minőségű bejegyzéseket is érdemes megfelelően tisztítani.[9] A sablon angol nyelvű változatában használt `plainnat` beállítás egyik sajátossága, hogy a cikkhez generált hivatkozás a cikk DOI-ját és URL-jét is tartalmazza, ami gyakran duplikátumhoz vezet – érdemes tehát a DOI-kat tartalmazó URL mezőket törölni.

[3] Informatikai témában gyakran hivatkozunk cikkeket a Springer LNCS valamely kötetéből, ez a hivatkozás erre mutat egy helyes példát.

[4] `https://dl.acm.org/`

[5] `http://dblp.uni-trier.de/`

[6] `http://ieeexplore.ieee.org/`

[7] `https://link.springer.com/`

[8] `http://scholar.google.com/`

[9] `https://github.com/FTSRG/cheat-sheets/wiki/BibTeX-Fixing-entries-from-common-sources`

2.6. A dolgozat szerkezete és a forrásfájlok

A diplomatervsablonban a TeX fájlok két alkönyvtárban helyezkednek el. Az `include` könyvtárban azok szerepelnek, amiket tipikusan nem kell szerkesztenünk, ezek a sablon részei (pl. címoldal). A `content` alkönyvtárban pedig a saját munkánkat helyezhetjük el. Itt érdemes az egyes fejezeteket külön TEX állományokba rakni.

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

1. 1 oldalas *tájékoztató* a szakdolgozat/diplomaterv szerkezetéről (`include/guideline.tex`), ami a végső dolgozatból törlendő,
2. *feladatkiírás* (`include/project.tex`), a dolgozat nyomtatott verzójában 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,
3. *címoldal* (`include/titlepage.tex`),
4. *tartalomjegyzék* (`thesis.tex`),
5. a diplomatervező *nyilatkozat*a az önálló munkáról (`include/declaration.tex`),
6. 1-2 oldalas tartalmi *összefoglaló* magyarul és angolul, illetve elkészíthető még további nyelveken is (`content/abstract.tex`),
7. *bevezetés*: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása (`content/introduction.tex`),
8. sorszámmal ellátott *fejezetek*: 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,
9. esetleges *köszönetnyilvánítás*ok (`content/acknowledgement.tex`),
10. részletes és pontos *irodalomjegyzék* (ez a sablon esetében automatikusan generálódik a `thesis.tex` fájlban elhelyezett `\bibliography` utasítás hatására, a 2.5. szakaszban leírtak szerint),
11. *függelékek* (`content/appendices.tex`).

A sablonban a fejezetek a `thesis.tex` fájlba vannak beillesztve `\include` utasítások segítségével. Lehetőség van arra, hogy csak az éppen szerkesztés alatt álló `.tex` fájlt fordítsuk le, ezzel lerövidítve a fordítási folyamatot. Ezt a lehetőséget az alábbi kódrészlet biztosítja a `thesis.tex` fájlban.

```
\includeonly{
  guideline,%
  project,%
  titlepage,%
  declaration,%
  abstract,%
  introduction,%
  chapter1,%
  chapter2,%
  chapter3,%
  acknowledgement,%
  appendices,%
}
```

Ha az alábbi kódrészletben az egyes sorokat a `%` szimbólummal kikommentezzük, akkor a megfelelő `.tex` fájl nem fordul le. Az oldalszámok és a tartalomjegyék természetesen csak akkor billennek helyre, ha a teljes dokumentumot lefordítjuk.

2.7. Alapadatok megadása

A diplomaterv alapadatait (cím, szerző, konzulens, konzulens titulusa) a `thesis.tex` fájlban lehet megadni.

2.8. Új fejezet írása

A főfejezetek külön `content` könyvtárban foglalnak helyet. A sablonhoz 3 fejezet készült. További főfejezeteket úgy hozhatunk létre, ha új TEX fájlt készítünk a fejezet számára, és a `thesis.tex` fájlban, a `\include` és `\includeonly` utasítások argumentumába felvesszük az új `.tex` fájl nevét.

2.9. Definíciók, tételek, példák

Definíció 1 (Fluxuskondenzátor térerőssége). Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. ▪

Példa 1. *Példa egy példára. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.*

Tétel 1 (Kovács tétele). Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. ▪

3. fejezet

Használt technológiák kiválasztása

3.1. Verziókezelés

A fejlesztés során fontos a változások átlátható, könnyű követése, a szoftver különböző verzóinak követése.

Erre a mára már kvázi ipari sztenderdnek számító `git` nevű szoftvert vettem igénybe. A forráskód felhőben történő tárolására a GitHub szolgáltatását használtam.

3.2. Frontend

3.2.1. JavaScript keretrendszer

A frontendhez használt technológia kiválasztásánál két fő szempontot tudunk megkülönböztetni. Az egyik az egyes backend rendszerek által támogatott, szerveroldalon renderlet, template engine-t használó megoldás, a másik a külön frontend framework használata. Utóbbi jóval nagyobb szabadságot és funkcionalitást ad a fejlesztő kezébe, és lehetőséget ad a legújabb technikák és könyvtárak használatára.

Emiatt úgy döntöttem, hogy az alkalmazás ezen részét a négy legnépszerűbb keretrendszer (React, Angular, Vue és Svelte) egyikével fogom megvalósítani. Ezek a keretrendszerek alapvető filozófiában és funkcionalitásban lényegében megyegyeznek, azonban az elérhető könyvtárak mennyisége és minősége a React esetében a legnagyobb, emiatt végső soron erre esett a választásom.

3.2.2. CSS keretrendszer

A CSS keretrendszer kiválasztása során két fő irányvonal jött szóba: az ún. atomic CSS, illetve az előre már elkészített komponenseket kínáló könyvtárak. Míg előbbi lehetővé teszi a teljesen egyedi design írását, utóbbi jóval nagyobb fokú kényelmet és a felület gyorsabb elkészítését teszi lehetővé.

Mivel jelen alkalmazás esetében nem volt szempont egy egyedi design elkészítése, ezért az utóbbi megoldás használata mellett döntöttem. A kiválasztott framework végül a Chakra UI lett, mely nagy mennyiségű kész, egyszerűen bővíthető és magas fokú accessibility-t kínáló komponensekkel rendelkezik.

3.3. Backend

A backend keretrendszer kiválasztása során a legfontosabb szempont az volt, hogy minél jobban képes legyen integrálódni a frontend keretrendszerhez. Mindenképpen szerettem volna elkerülni a külön repository használtatát. Ennek fő okai, hogy mind a fejlesztés,

mind a deployment jelentősen egyszerűsödik, valamint közös könyvtár esetén a közös kódrészletek megosztása is triviálissá válik.

A fentiek miatt a lehetőségeim a NodeJS alapú megoldásokra szűkítettem le. Egy népszerű, és általam már kipróbált megoldás az Express keretrendszer, azonban a React világában létezik egy, az igényeimnek méginkább megfelelő framework: a NextJS. Ez egy React-ra épülő, SSR-t (TODO: lábjegyzet) támogató keretrendszer, amely rendelkezik egy úgynevezett API routes nevű funkcióval.

Ennek segítségével a React alkalmazásunkon belül készíthető egy API réteg, amely a fordítás után szerveroldalon futó függvényekké lesz átalakítva. Ezzel lényegében megspóroljuk, hogy külön API-t és hozzávaló elérési, illetve deployment környezetet kelljen létrehozni, miközben egy Express-hez hasonló interfészt tudunk használni. Előnye továbbá, hogy rendkívül egyszerűvé teszi a frontend és a backend közötti kódmegosztást, csökkentve ezzel a duplikációkat és erősítve a type safety-t.

3.4. Adatbázis

Az adatbázis architektúra kiválasztása esetén két fő irányvonal volt a meghatározó: a hagyományos relációs adatbázis (pl. PostgreSQL, MySQL) és a NoSQL megoldások (pl. MongoDB, Google Firestore, AWS DynamoDB).

A technológia kiválasztása során fontos szempont volt az ACID elvek követése, a relációk egyszerű kezelése és a minél nagyobb típusbiztosság elérése a backend kódjában.

Ezeknek a szempontoknak a hagyományos relációs adatbázisok felelnek meg, ezen belül a PostgreSQL-re esett a választásom, mivel ez egy általam már ismert, sok funkciót támogató, inegyenes és nyílt forráskódú megoldás.

3.4.1. Az adatbázis elérése

Miután a backend és adatbázis technológiák és könyvtárak kiválasztásra kerültek, szükséges a kettő közötti kommunikációt biztosító könyvtár meghatározására. NodeJS környezetben rendkívül nagy választék áll rendelkezésre, ezeket két nagy csoportra lehet bontani: ORM és query builder.

Míg az előbbi megoldás egy erős absztrakciós réteget képez az adatbázis szerkezete és a programkód között, addig a query builder egy, az SQL-hez közelebbi interfészt kínál a felhasználónak. Ez utóbbi előnye, hogy a felhasználó által írt kód közelebb van a végső soron lefutó SQL-hez, így átláthatóbbak az egyes lekérések végeredményei.

A query builder megoldások közül is kiemelkedik a Prisma, amely egy NodeJS környezetben működő adatbázis kliens, ami a hansúlyt a type safety-re helyezi. Az adatbázis sémát egy speciális .prisma kiterjesztésű fájlban tudjuk megírni, majd ezt a Prisma migrációs eszközével tudjuk az adatbázisunkba átvezetni. (TODO: lábjegyzet hogy a migráció még csak experimental) Ezután lehetőségünk van a séma alapján a sémából TypeScript típusokat generálni, melyek használata nagyban megkönnyíti és felgyorsítja a fejlesztés menetét.

```
const user = new User()
user.firstName = "Bob"
user.lastName = "Smith"
user.age = 25
await repository.save(user)
```

3.1. lista. Adatbázis beillesztés TypeORM környezetben

```
const user = await prisma.user.create({
  data: {
    firstName: "Bob",
    lastName: "Smith",
```

```
    age: 25
  }
})
```

3.2. lista. Adatbázis beillesztés Prisma segítségével

todo: táblázat

ORM query builder paradigma oop funkcionális programozás séma definíció ts class+dekorátorok változó adatelérés példányosított query interface, chain-elt függvények objektumon keresztül

3.5. REST és GraphQL

A fentieken túl utolsó lépésként szükséges volt eldönteni a frontend és a backend közötti kommunikáció módját. A REST architektúra mellett ugyanis megjelent a GraphQl, ami az eddigi lehetőségekhez képest egy flexibilisebb megoldást kínál az adatok elérésére. A két technológia előnyeit és hátrányait az alábbi táblázatban foglaltam össze.

TODO: táblázat

A fenti szempontokat figyelembe véve végül a hagyományos REST architektúra alkalmazása mellett döntöttem.

3.6. TypeScript

A frontend illetve a Node.js világában lehetőségünk van arra, hogy a JavaScript helyett egy arra lefordítható egyéb nyelvet használjunk fejlesztés közben. Az egyik ilyen lehetőség a TypeScript, amely egy már több éve jelenlévő, gyorsan fejlődő alternatíva többek között olyan funkciókkal, mint a static type check, type inference és magas szintű IDE integráció.

Az általam választott technológiák közül a Prisma kifezezetten épít a TypeScript nyújtotta előnyökre, illetve a Next.js integrációhoz elegendő egy konfigurációs fájlt létrehozni, így evidens volt, hogy az alkalmazást TypeScript segítségével írjam meg.

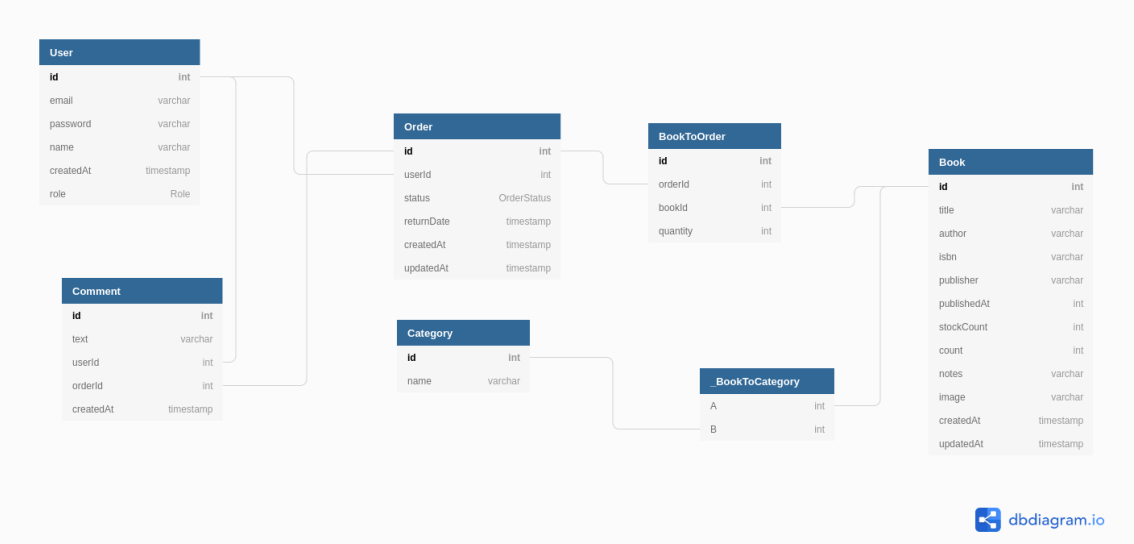
4. fejezet

Az alkalmazás felépítése

4.1. Adatbázisséma

4.1.1. Tervezés

Az adatbázisséma tervezéséhez a dbdiagram.io nevű platformfüggetlen, webes ER diagram tervező szoftvert használtam. Ez egy saját fejlesztésű, DBML nevű DSL nyelvet használ a séma leírására, és lehetővé teszi ennek exportálását különféle formátumokba.



4.1. ábra. Az adatbázisséma ER diagramja.

A séma tervezése során a Prisma által használt elnevezési konvenciókat használtam megkönnyítve a két technológia közötti átjárhatóságot.

4.1.2. Implementáció

A séma adatbázisba történő átvezetésére két lehetőségünk van. Az egyik, hogy a dbdiagram oldalról lehetőségünk van .sql kiterjesztésű fájlt letölteni, ezt a létrehozott adatbázisunkon futtatni, majd a Prisma introspect funkcióját használva legenerálni hozzá a Prisma schema fájlt a backendünk számára. A másik, egyszerűbb megoldás a Prisma migrate[1] használata. Ez esetben nekünk manuálisan kell létrehozni a Prisma schema fájlt a korábbi diagram alapján, majd a

[1] A dolgozat írása idejében ez a funkció még experimental státuszban volt, de a használata során nem ütköztem problémákba.

```
prisma migrate save --experimental
prisma migrate up --experimental
```

parancsokat kiadva létrehozzuk és futtatjuk a Prisma migrációt. Ez utóbbi megoldás előnyei, hogy egy központi helyen tudjuk kezelni a séma változásait, valamint ezt adatbázis-agnosztikus módon tehetjük meg.

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model Book {
  id          Int           @id @default(autoincrement())
  title       String
  author      String?
  isbn        String?
  publisher   String?
  publishedAt Int?
  stockCount  Int?          @default(1)
  count       Int?          @default(1)
  notes       String?
  image       String?
  createdAt   DateTime?     @default(now())
  updatedAt   DateTime?     @default(now())
  orders      BookToOrder[]
  categories  Category[]
}

model BookToOrder {
  id       Int   @id @default(autoincrement())
  orderId  Int
  bookId   Int
  quantity Int?  @default(1)
  books    Book  @relation(fields: [bookId], references: [id])
  orders   Order @relation(fields: [orderId], references: [id])

  @@unique([bookId, orderId], name: "BookToOrder_book_order_unique")
}

model Category {
  id    Int    @id @default(autoincrement())
  name  String
  books Book[]
}

model Comment {
  id        Int       @id @default(autoincrement())
  text      String?
  createdAt DateTime? @default(now())
  userId    Int
  orderId   Int
  order     Order     @relation(fields: [orderId], references: [id])
  user      User      @relation(fields: [userId], references: [id])
}

model Order {
  id         Int           @id @default(autoincrement())
  userId     Int
  returnDate DateTime?
  status     orderstatus?  @default(PENDING)
  createdAt  DateTime?     @default(now())
  updatedAt  DateTime?     @default(now())
  user       User          @relation(fields: [userId], references: [id])
  books      BookToOrder[]
  comments   Comment[]
}
```

```
model User {
  id        Int       @id @default(autoincrement())
  email     String    @unique
  password  String
  name      String?
  createdAt DateTime? @default(now())
  role      userrole? @default(BASIC)
  comments  Comment[]
  orders    Order[]
}

enum orderstatus {
  PENDING
  RENTED
  RETURNED
  LATE
}

enum userrole {
  BASIC
  ADMIN
  EDITOR
}
```

4.1. lista. Séma leírása a Prisma DSL-ben

A Prisma DSL-ben a dbdiagram.io-hoz hasonló módon tudjuk felvenni a relációkat. Ennek nagy előnye, hogy a táblák közötti kapcsolatokat egyszerűen tudjuk modellezni, amit a `prisma migrate` át tud vezetni az adatbázisunkba.

Ahogy a fenti kódrészletben is látszik, a Category és a Book modellek közötti kapcsolatot biztosító kapcsolótábla nem jelenik meg a Prisma schema fájlban. Ez egy úgynevezett implicit kapcsolat, melyet a Prisma motorja automatikusan kezel és a migráció során létrehozza a megfelelő táblát a két entitás között.

4.1.3. Kapcsolódás az adatbázishoz

A Prisma esetében az adatbáziskapcsolatot egy connection string segítségével tudjuk megadni, amit alapesetben a megadott környezeti változóból (`DATABASE_URL`) olvas ki. Ennek nagy előnye, hogy könnyen tudunk lokális és remote adatbázisok között váltani, illetve egyszerűvé teszi a production-ben lévő kód és az adatbázis közötti kapcsolat létrehozását.

4.2. A backend felépítése

4.2.1. Next.js API routes

A Next.js keretrendszer a 9-es verzió óta lehetővé teszi szerveroldali kód írását az alkalmazásunkhoz. Ennek segítségével a `pages/api` mappába helyetett fájljaink szolgálnak backendként. Minden ide helyezett fájl egyben a nevének megfelelő API végpont lesz, tehát például a `pages/api/books.ts`-ben lévő kódot a pages/api/books URL-en keresztül tudjuk elérni.

Támogatja továbbá a backend-oldali dinamikus routing-ot, azaz például a `pages/api/books/[id].tsx` fájl a `pages/api/<id>` URL-nek felel meg, ahol az `id` változót alábbi módon érhetjük el:

```
export default function handler(req, res) {
  const {
    query: { id },
  } = req

  res.end(`Book: ${id}`)
}
```

4.2. lista. Next.js dinamikus routing

4.2.2. next-connect

Alapesetben a Next.js csak egy egyszerű interface-t biztosít nekünk, amin keresztül elérhetjük a HTTP kérés request és response objektumokat annak kezeléséhez. Ez azonban nezhézkessé teszi a különböző kérések feldolgozását (pl. GET, POST és PUT), valamint különbőző kódrészletek egyszerű újrafelhasználását.

Ennek kényelmesebbé tételére döntöttem a next-connect könyvtár használata mellett, amellyel a fenti igények könnyedén megvalósíthatóak. Az alábbi két kódrészletben szeretném bemutatni a főbb különbségeket.

```
export default function handler(req: NextApiRequest, res: NextApiResponse) {
  if (req.method === 'GET') {
    res.statusCode = 200
    res.setHeader('Content-Type', 'application/json')
    res.end(JSON.stringify({ name: 'John Doe' }))
  } else if (req.method === 'POST') {
    // Process a POST request
  }
}
```

4.3. lista. Default Next.js API routes

```
import nextConnect from "next-connect"

const handler = nextConnect<NextApiRequest, NextApiResponse>()

handler
  .get((req, res) => {
    res.json({ name: 'John Doe' })
  })
  .post((req, res) => {
    // Process POST request
  })

export default handler
```

4.4. lista. Kérés kezelése next-connect segítségével

4.2.2.1. Middleware támogatás

A Next.js alapesetben nem rendelkezik beépített middleware támogatással, emiatt bizonyos kódrészletek újrahasználása körülményes lehet. A next-connect azonban ezt a folyamatot rendkívül egyszerűvé teszi, így könnyen lehet védett útvonalakat létrehozni például csak bejelentkezett felhasználók számára.

```
import nextConnect from "next-connect"
import requireLogin from "middleware/requireLogin"
import requireAdmin from "middleware/requireAdmin"
const handler = nextConnect<NextApiRequest, NextApiResponse>()

handler
  .get((req, res) => {
    res.json({ name: 'John Doe' })
  })
  .use(requreLogin)
  .post((req, res) => {
    // Process POST request
  })
  .use(requireAdmin)
  // Process other requests

export default handler
```

4.5. lista. Middleware kezelés next-connect segítségével

A fenti kódrészletben a POST kérés csak bejelentkezett felhasználók számára elérhető. Ezek egymás után is fűzhetőek, így komplex igények is rendkívül egyszerűen megvalósíthatóak.

4.3. A frontend felépítése

A Next.js framework a frontenden is alkalmazza a file-based routing koncepcióját. Ennek megfelelően elegendő az `src/pages/` mappában elhelyezett `.tsx` fájlban definiálnunk egy React komponenst, és a keretrendszer a megadott fájlnévnek megfelelő URL-en fogja kirenderelni az oldalunkat.

Az adminoknak elérhető oldalakat egy külön `admin` mappába helyeztem a könnyebb elkülöníthetőség érdekében.

A `components/` mappába kerültek az újrafelhasznált, illetve kiszervezett React komponensek. Az általam írt saját React hook-ok pedig az `src/lib/hooks.tsx` fájlba kerültek.

4.4. Kódmegosztás

Mivel a backend és frontend kódja egyazon git repository-ban van elhelyezve, a kettő között kódmegosztás szinte triviális.

Bármely, az `src` mappán belül elhelyezett fájl felhasználható a szerver- és kliensoldali kód számára. Ennek megfelelően az `src/lib/interfaces.ts` és `src/lib/constants.ts` fájlokban találhatóak a közösen használt interfészek és konstans változók.

Az általam használt Prisma ORM egy nagy előnye továbbá, hogy az általa generált interfészek és típusdefiníciók is használhatóak kliens- és szerveroldalon egyaránt, ezáltal csökken a bugok mennyisége, és egy esetleges sémaváltozás esetén jóval könnyebb lekövetni az okozott változásokat.

5. fejezet

Az alkalmazás működése

Lorem ipsum

6. fejezet

CI/CD

A Continuous Integration / Continuous Delivery egy manapság már általánosan használt módszer az alkalmazások folyamatos tesztelésére és publikálására.

A Simonyi Könyvtár esetében igyekeztem ezeket a megközelítéseket alkalmazni a fejlesztés és deploy esetében is.

6.1. Continuous Integration

A forráskód tárolására a GitHub-ot használtam, így CI megoldásnak evidens volt a GitHub Actions használata.

Ez esetben elegendő egy `.yaml` fájlt elhelyezni a `.github/workflows/` mappába elhelyezni, és egy `git push` parancs kiadása után automatikusan lefut a szkriptünk. Ennek az állapotát a GitHub repository webes felületén tudjuk követni.

6.1.1. Statikus kódanalízis

A fejlesztés során nagy segítséget nyújtanak azok az eszközök, amelyek segítségével a forráskódunkat még fordítás vagy futtatás előtt ellenőrizni tudjuk.

A JavaScript világában ennek egyik szinte sztenderddé vált eszköze az `eslint`, amely egy TypeScript-hez is használható linter. Ennek segítségével ki tudjuk szűrni a nem használt kódrészleteket, valamint egységes formázást írhatunk elő a forráskódra, nagyban segítve ezzel a közös munkát.

Az alkalmazásomban én is ezt a megoldást használtam, mely lokálist a `yarn lint` parancs kiadásával lokálisan, valamint a GitHub Actions build folyamataként automatikusan is futtatható.

6.2. Continuous Delivery

Az alkalmazás hostolására két szolgáltatást használtam.

A frontend és backend közös deploymentjéhez a Vercel-t választottam. Ez rendkívül egyszerűen integrálható a Next.js keretrendszerrel, elegendő a GitHub repository-val összekötni és bármiféle extra konfiguráció nélkül elérhető lesz az alkalmazásunk egy `git push` parancs kiadása után.

A Vercel felületén lehetőségünk van a deployment státuszát ellenőrizni, korábbi deploymentre visszaállni, illetve a Next.js 10-es verziójától kezdve már különböző analitikák monitorozására is.

Az adatbázishoz a Heroku ingyenes PostgreSQL szolgáltatását vettem igénybe. Ebben az esetben elegendő az adatbázishoz kapott connection string-et a Vercel felületén a környezeti változók között megadni, és az alkalmazásunk hozzáfér az adatbázisunkhoz.

Köszönetnyilvánítás

Ez 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.

Irodalomjegyzék

[1] Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Kar: Diplomaterv portál (2011. február 26.). `http://diplomaterv.vik.bme.hu/`.

[2] James C. Candy: Decimation for sigma delta modulation. 34. évf. (1986. 01) 1. sz., 72–76. p. DOI: `10.1109/TCOM.1986.1096432`.

[3] Peter Kiss: Adaptive digital compensation of analog circuit imperfections for cascaded delta-sigma analog-to-digital converters, 2000. 04.

[4] Wai L. Lee – Charles G. Sodini: A topology for higher order interpolative coders. In *Proc. of the IEEE International Symposium on Circuits and Systems* (konferenciaanyag). 1987. 4-7 05., 459–462. p.

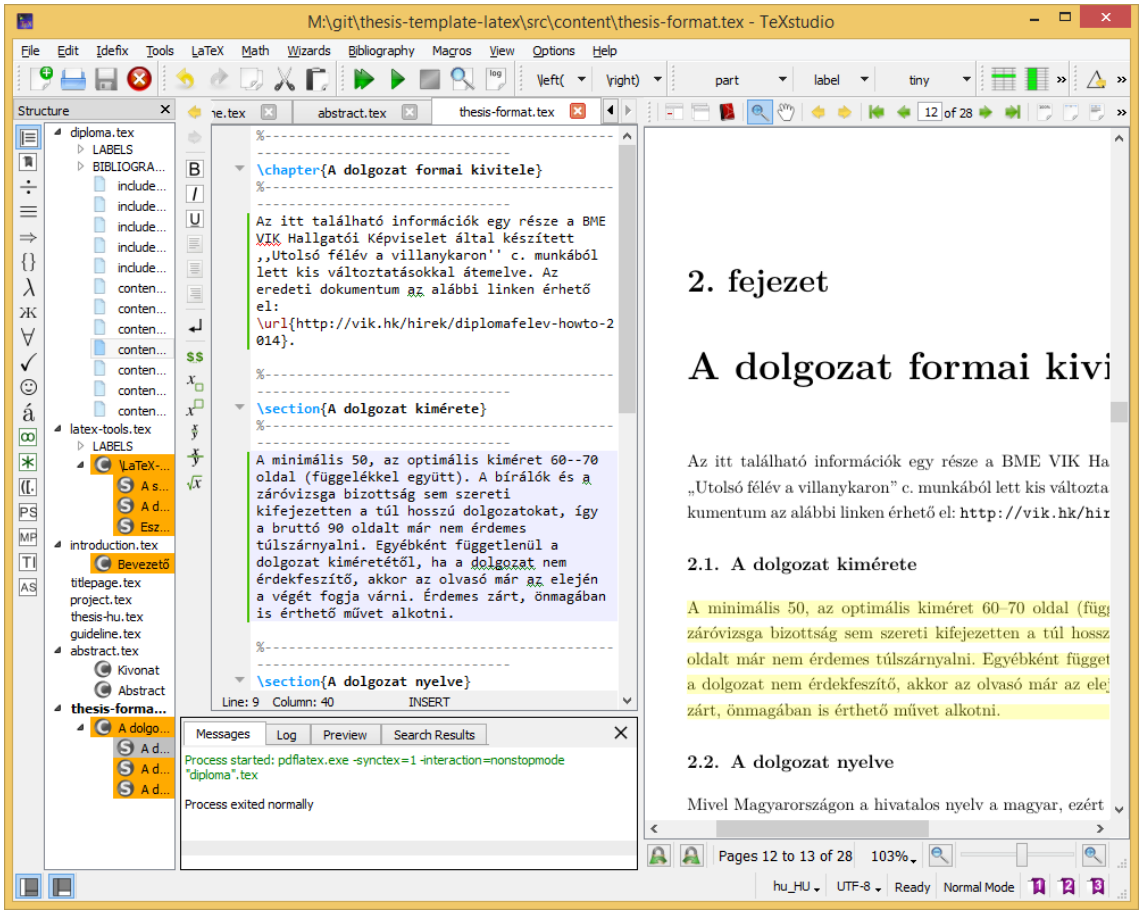
[5] Alexey Mkrtychev: Models for the logic of proofs. In Sergei Adian – Anil Nerode (szerk.): *Logical Foundations of Computer Science*. Lecture Notes in Computer Science sorozat, 1234. köt. 1997, Springer Berlin Heidelberg, 266–275. p. ISBN 978-3-540-63045-6. URL `http://dx.doi.org/10.1007/3-540-63045-7_27`.

[6] Richard Schreier: *The Delta-Sigma Toolbox v5.2*. Oregon State University, 2000. 01. `http://www.mathworks.com/matlabcentral/fileexchange/`.

[7] Ferenc Wettl – Gyula Mayer – Péter Szabó: *LaTeX kézikönyv*. 2004, Panem Könyvkiadó.

Függelék

F.1. A TeXstudio felülete



F.1.1. ábra. A TeXstudio LaTeX-szerkesztő.

F.2. Válasz az „Élet, a világmindenség, meg minden” kérdésére

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42. \tag{F.2.1}$$

A Faraday-indukciós törvényből levezetve

$$\operatorname{rot} E = -\frac{dB}{dt} \qquad \longrightarrow \qquad U_i = \oint\limits_{\mathbf{L}} \mathbf{Edl} = -\frac{d}{dt} \int\limits_{A} \mathbf{Bda} = 42. \tag{F.2.2}$$