SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR,
INFORMATIKA SZAK

SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM

F1 Ticket Manager - Online jegyek kezelése

DIPLOMADOLGOZAT

Témavezető:
Dr. Márton Gyöngyvér,
Egyetemi tanár
Györfi Ágnes,
Egyetemi tanársegéd

Végzős hallgató:
Kovács Bence

2023

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
SPECIALIZAREA INFORMATICĂ

UNIVERSITATEA
SAPIENTIA

F1 Ticket Manager - Gestiunea tichetelor

LUCRARE DE DIPLOMĂ

Coordonator științific:
Dr. Márton Gyöngyvér,
Profesor universitar
Györfi Ágnes,
Asistent universitar

Absolvent:
Kovács Bence

2023

SAPIENTIA HUNGARIAN UNIVERSITY OF TRANSYLVANIA
FACULTY OF TECHNICAL AND HUMAN SCIENCES
COMPUTER SCIENCE SPECIALIZATION

SAPIENTIA
HUNGARIAN UNIVERSITY
OF TRANSYLVANIA

F1 Ticket Manager - Online ticket management

BACHELOR THESIS

Scientific advisor:
Dr. Márton Gyöngyvér,
Full Professor
Györfi Ágnes,
Assistant professor

Student:
Kovács Bence

2023

Declaraţie

Subsemnata/ul ....., absolvent(ă) al/a specializării ……., promoţia…… cunoscând prevederile Legii Educaţiei Naţionale 1/2011 şi a Codului de etică şi deontologie profesională a Universităţii Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licenţă/proiect de diplomă/disertaţie se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informaţiile şi datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea,

Data: Absolvent

Semnătura……

Kivonat

Napjainkban az online jegyvásárlás rohamos elterjedése figyelhető meg a technológia fejlődésével és az internet elérhetőségének növekedésével. Egy online platformon keresztül az emberek ma már kényelmesen és gyorsan tudnak jegyeket vásárolni különféle eseményekre, mint például koncertekre, színházi előadásokra vagy sporteseményekre, mint például a Formula-1.

Az online jegyvásárlás számos előnnyel jár. A vásárlók egyszerűen és kényelmesen böngészhetnek és választhatnak a széles körű jegyválaszték közül. Az online platformok részletes információkat nyújtanak az eseményekről, beleértve a dátumokat, helyszíneket és leírásokat. Emellett az online jegyvásárlás lehetővé teszi a jegyek összehasonlítását, árak és típusok kiválasztását, ami segít a vásárlóknak a legjobb lehetőség megtalálásában.

A technológiai fejlesztések, mint például a biztonságos online fizetési rendszerek és az elektronikus jegyek, hozzájárultak az online jegyvásárlás elterjedéséhez. A vásárlók könnyedén és biztonságosan fizethetnek az online platformokon (webshop) keresztül, és elektronikus jegyet kapnak, amelyet mobil eszközükön vagy nyomtatható formában mutathatnak fel az eseményen. Az online jegyvásárlás megkönnyíti az eseményekre való részvételt, hiszen a vásárlóknak nem kell hosszú sorokban állniuk a jegypénztáraknál.

Rezumat

În prezent, se observă o răspândire rapidă a achiziționării de bilete online, odată cu dezvoltarea tehnologică și creșterea accesului la internet. Prin intermediul unei platforme online, oamenii pot cumpăra bilete confortabil și rapid pentru diverse evenimente, cum ar fi concerte, spectacole de teatru sau evenimente sportive, precum Formula 1.

Achiziționarea de bilete online vine cu numeroase avantaje. Cumpărătorii pot naviga și alege cu ușurință dintr-o gamă largă de opțiuni de bilete. Platformele online oferă informații detaliate despre evenimente, inclusiv date, locații și descrieri. De asemenea, achiziționarea de bilete online permite compararea prețurilor și selecționarea diferitelor tipuri de bilete, ceea ce ajută cumpărătorii să găsească cea mai bună opțiune.

Dezvoltările tehnologice, cum ar fi sistemele de plată online sigure și biletele electronice, au contribuit la răspândirea achiziționării de bilete online. Cumpărătorii pot plăti ușor și în siguranță prin intermediul platformelor online (webshop) și primesc biletele electronice, pe care le pot prezenta pe dispozitivele lor mobile sau sub formă printată la eveniment. Achiziționarea de bilete online facilitează participarea la evenimente, deoarece cumpărătorii nu mai trebuie să stea în rânduri lungi la casele de bilete.

Abstract

Currently, the rapid spread of online ticket purchasing can be observed due to technological advancements and the increased accessibility of the internet. Through an online platform, people can now conveniently and quickly purchase tickets for various events such as concerts, theater performances, or sports events like Formula 1.

Online ticket purchasing comes with numerous advantages. Customers can easily browse and choose from a wide range of ticket options. Online platforms provide detailed information about events, including dates, venues, and descriptions. Additionally, online ticket purchasing allows for ticket comparisons, price and type selections, which help customers find the best options available.

Technological developments, such as secure online payment systems and electronic tickets, have contributed to the widespread adoption of online ticket purchasing. Customers can easily and safely make payments through online platforms (webshop) and receive electronic tickets that can be presented on their mobile devices or in printable form at the event. Online ticket purchasing simplifies event attendance, as customers no longer have to wait in long queues at ticket counters.

Tartalomjegyzék

1. fejezet

Jelölések

Rövidítés	Angol megnevezés	Magyar megnevezés
F1	Formula One	Formula-1/Forma-1
FIA	International Automobile Federation	Nemzetközi Automobil Szövetség
F1TM	F1 Ticket Manager	F1 Ticket Manager
EDI	Electronic Data Interchange	Elektronikus Adatcsere
PDF	Portable Document Format	Hordozható Dokumentum Formátum
QR Code	Quick Response Code	Quick Response-kód (=gyors válasz)
E2EE	End-To-End Encryption	Végpontok Közötti Titkosítás
PIN	Postal Index Number	Postai Indexszám
DOM	Document Object Model	Dokumentum Objektum Modell
OSS	Open-Source Software	Nyílt Forráskódú Szoftver
NoSQL	Not only Structured Query Language	Nem csak Strukturált Lekérdezőnyelv
API	Application Programming Interface	Alkalmazásprogramozási Felület
CDN	Content Delivery Network	Tartalomelosztó Hálózat
IDE	Integrated Development Environment	Integrált Fejlesztői Környezet
DBMS	Database Management Systems	Adatbázis-kezelő Rendszer (ABKR)
UI	User Interface	Felhasználói Felület
UX	User Experience	Felhasználói Élmény
HTML	HyperText Markup Language	Hiperszöveges Jelölőnyelv
CSS	Cascading Style Sheets	Lépcsőzetes Stíluslapok
HTTP	HyperText Transfer Protocol	Hiperszöveges Szállítási Protokoll
BCM	Block Cipher Mode	Blokktitkosítási Mód
AES	Advanced Encryption Standard	-
RSA	Rivest–Shamir–Adleman	-

1.1. táblázat. Jelölések

2. fejezet

Bevezető

2.1. Témaválasztás indoklása

Napjainkban az online jegyvásárlás nagy előretörést ért el a technológia fejlődésével az egyre szélesebb körben történő bankkártyás internetes vásárlások következtében. Egy online platformon keresztül az emberek ma már kényelmesen és gyorsan tudnak jegyeket vásárolni különféle eseményekre tekintettel arra, hogy percek alatt el tudjuk végezni a világ bármely pontjából a nap bármely időpontjában. Az online jegyvásárlás számos előnnyel jár, amelyeknek köszönhetően egyre népszerűbbé válik.

Az egyik legfontosabb előny az online jegyvásárlás esetén az, hogy a vásárlók egyszerűen és kényelmesen böngészhetnek és választhatnak a széles körű jegyválaszték közül. Az online platformok részletes információkat biztosítanak az eseményekről, így a potenciális vásárlók teljes körű tájékoztatást kapnak az eseményről, mielőtt eldöntenék, hogy vásárolnak-e jegyet. Tehát a vásárlók magabiztosan dönthetnek arról, hogy melyik eseményre szeretnének jegyet vásárolni, anélkül, hogy bármilyen korlátozásba ütköznének.

A technológiai fejlesztések, mint például a biztonságos online fizetési rendszerek (2.1), amelyek az EDI rendszerek alkomponenseként ismert, és az elektronikus jegyek, hozzájárultak az online jegyvásárlás népszerűségéhez. A vásárlók könnyedén és biztonságosan fizethetnek az online platformokon keresztül, és elektronikus jegyet kapnak, amelyet mobil eszközükön vagy nyomtatható formában mutathatnak fel az eseményen. Az elektronikus jegyek további előnye, hogy nehezen veszíthetőek el vagy semmisülhetnek meg, így a vásárlók biztonságban tudhatják az értékes jegyeiket. Itt fontos megemlíteni, hogy ezen jegyek tárolása és biztonságban tartása további adatbiztonsági kérdéseket vet fel, amellyel foglalkozunk a dolgozat keretében. A platformokon keresztül a vásárlók egyszerűen választhatják ki a kívánt eseményt, a megfelelő ülőhelyet vagy jegytípust, és azonnal megvásárolhatják a jegyüket néhány kattintással. Ez időt és energiát takarít meg a vásárlók számára, ami napjainkban egy lényeges tényező.

2.1. ábra. Online fizetési rendszerek

További előnyeként egy ilyen platformnak megemlítendő, hogy a szervezők számára is jelentősen hatékonyabbá teszi a rendelések nyomon követését, statisztikák készítését, amelyeket felhasználhatnak értékesítési jelentések és a marketing javításához. Emellett az online jegyvásárlás lehetőséget nyújt a szervezőknek arra is, hogy célzottan reklámozzák az eseményüket, így nagyobb látogatottságot érhetnek el.

A jelenleg is működő hivatalos platform, ahol direkt módon juthatunk hozzá jegyekhez az F1-es versenyhétvégékre, az F1 Experiences. Itt gyorsan és kényelmesen vásárolhatjuk meg a kívánt jegyünket, amelyet a sikeres rendelés és kifizetés után emailben kapjuk meg PDF formátumban, amely tartalmazza a megvásárolt jegy(ek)et, egyedi azonosítókat és QR kódokat. Az email továbbá tartalmazza a számlázási adatokat. Az eseményre érve, a belépő kapuknál, egy erre a célra kihelyezett okos eszközzel megtörténik a QR kód olvasása és hitelesítése. Pozitív eredmény esetén beléphetünk az esemény helyszínére.

A fent említett folyamat megengedi, hogy ezek az elektronikus jegyek átruházhatóak a tulajdonos által bárki számára. Ezzel persze önmagában nincs probléma, mivel ezt a szabadságot meg kell lehessen adni a felhasználóknak, hogy bizonyos esetekben más személy tudjon részt venni a vásárló helyett, így nem veszik kárba a vásárlás. Ez a rendszer viszont teret ad egy olyan biztonsági kérdésnek, amelyet jelenleg a felhasználó felelősségére van bízva, miszerint ezt a kódot akár hetekkel, hónapokkal a használatuk előtt kapnak meg a felhasználók elektronikus levél formájában és azt bárki megszerezheti, akinek hozzáférése van a fiókhoz. Rosszabb esetekben, egy szándékos kibernetikai támadás esetén is eltulajdoníthatják és felhasználhatják. Ennek persze kisebb a valószínűsége, viszont ami egy aggasztó tény, hogy a felhasználók nagy része nem megfelelő módon kezeli az adatainak a biztonságos tárolását és számos esetben fellelhetőek olyan emberi hibák, amelyeket kihasználnak az adathalászok, hogy hozzáférjenek a megvásárolt jegyekhez és saját célokra használják fel, többnyire illegális módon kereskedni velük.

Gyakran megtörténik, hogy egy felhasználó több oldalra is ugyan azokat a bejelentkezési adatokat adja meg a regisztráció során. Ez többségében a személyes email fiók felhasználó nevével és jelszójával megegyezik és ezt az adathalászok is figyelembe veszik.

Egy másik sebezhetőség, hogy számos esetben egy fiókhoz több személy is hozzáfér, így már nem beszélhetünk biztonságos adattárolásról. Megemlítendő viszont, hogy az email szolgáltatók biztosítanak E2EE-t az elektronikus levelek küldésekor és fogadásakor.

Az F1 Ticket Manager webes alkalmazás célja az alapvető jegyvásárlási funkcionalitások biztosítása, valamint a teljes vásárlási folyamat biztonságosabbá tétele. Ennek megvalósítására számos technológiai fejlesztés és programozói technika létezik. Az alkalmazás fejlesztésénél beépítésre került PIN kódok használata, amely egy emelt szintű biztonságot nyújt a felhasználó számára, valamint EGY? titkosítási algoritmus, amely az eredeti adatok azonnali visszafejtését hivatott megnehezíteni.

2.2. Elméleti megalapozás és szakirodalmi áttekintő

2.2.1. Webes alkalamazás felépítése

Az F1TM a React JavaScript programozási nyelv keretrendszerével valósult meg a Microsoft Visual Studio Code IDE-ben. Az alkalmazás használ harmadik féltől származó könyvtárakat is, amelyek felgyorsítják a fejlesztési folyamatot, mivel előre le van implementálva számos funkcionalitás. Ezek általában több fejlesztő által használtak és vannak tesztelve, ezért többségében gyorsabbak és biztonságosabbak.

Lévén, hogy az alkalmazás egy webes platform, a struktúrája két fő részből áll: frontend és backend. Ezen projekt keretében első sorban a frontend implementációján volt a hangsúly.

A frontend az a része a webes alkalmazásnak, amellyel a felhasználók közvetlenül interakcióba lépnek. Ez a rész felelős a UI megjelenítéséért és a felhasználói interakciók kezeléséért. A frontend általában a böngészőben fut, és a felhasználó által látott elemeket jeleníti meg, például az oldalak, űrlapok, gombok, navigációs elemek stb. A frontend tervezésekor figyelembe kell venni a UX aspektusait is. Ezek azért felelnek, hogy a felhasználói élmény a lehető legjobb legyen a weboldal böngészése során. Itt első sorban a letisztultság, átláthatóság és az összezavaró elemek elkerülése a legfőbb cél.

A frontend technológiák közé tartozhatnak:

- HTML: Az alapvető struktúrát és tartalmat határozza meg a weboldalakhoz.
- CSS: A megjelenítést és a stílust adja a weboldalaknak, mint például a színek, betűtípusok, elrendezés stb.
- JavaScript: A dinamikus és interaktív funkciókért felelős, például animációk, eseménykezelés, adatmanipuláció. Gyakran használnak frontend keretrendszereket, például a React JS-t, Next.js-t vagy Angular-t, amelyek segítenek az alkalmazás fejlesztésében és szervezésében.

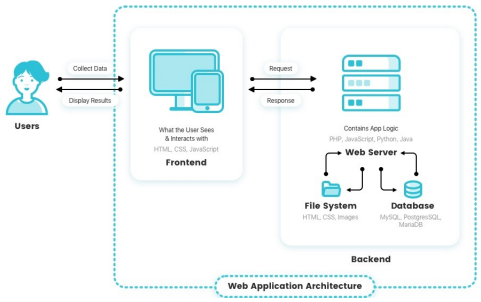
A backend a szerveroldali logikát és adatkezelést végzi. Ez a rész felelős az adatbáziskezelésért, a logika végrehajtásáért, a felhasználói kérések feldolgozásáért és a válaszok generálásáért. A backend nem közvetlenül látható vagy interaktív a felhasználók számára, viszont folyamatosan kommunikál a frontend-el az API-kon keresztül.

A backend technológiák közé tartozhatnak:

- Szerveroldali programozási nyelvek: Python, Ruby, Java, PHP stb.

- Keretrendszerek: Node.js, Django, Ruby on Rails, Laravel stb., amelyek segítenek az alkalmazás fejlesztésében és a szerveroldali logika megvalósításában.
- Adatbázis-kezelő rendszerek: MySQL, PostgreSQL, MongoDB stb., amelyek tárolják az alkalmazás adatai és lehetővé teszik ezek lekérdezését és manipulálását.

A frontend és backend között kommunikáció történik HTTP kérések és válaszok segítségével. A frontend kéréseket küld a backendnek, például adatlekérdezések vagy műveletek végrehajtása érdekében. Ezek a backend(ek) API végpontjain keresztül történnek A backend feldolgozza ezeket a kéréseket, és visszaküldi a válaszokat a frontendnek (2.2). Nagyon alkalmazások esetében megtörténhet, hogy a frontend több backend szerverről kéri le az információkat. Ez az felhasználó számára nem feltűnő, mivel egy megfelelő UX-szel rendelkező frontend minden esetben egy státuszjelző elemet helyez a betöltés idejére (animáció), függetlenül attól, hogy éppen melyik szerverrel történik a kommunikáció.



2.2. ábra. Webes alkalmazás architekúrája

2.2.2. A React JavaScript keretrendszer

Az elmúlt években a React JS jelentős népszerűségre tett szert a webfejlesztés területén. A React egy nyílt forráskódú JavaScript keretrendszer, amelyet a Facebook (Meta) fejlesztett ki, és célja a felhasználói felületek könnyű és hatékony megvalósítása. A React alapvetően egy komponens alapú megközelítést kínál, amely lehetővé teszi a fejlesztők számára, hogy újra felhasználható, önálló építőelemeket hozzanak létre, amelyeket könnyedén kombinálhatnak egymással a komplexebb felhasználói felületek elkészítése érdekében.

A React (2.3) rendkívül népszerűvé vált a fejlesztők körében számos előnye miatt. Elsőként említhetjük a hatékony Virtual DOM implementációját, amely lehetővé teszi az alkalmazások gyors és hatékony frissítését. A Virtual DOM a weboldal megjelenítéséhez használt valós DOM virtuális reprezentációja. Amikor változás történik az adatokban, a React a Virtual DOM-on keresztül kiszámítja az optimális frissítéseket, majd ezeket a változtatásokat csak a valós DOM-ra alkalmazza. Ez a megközelítés jelentős sebességjavulást eredményez a webalkalmazásokban.

A második fontos előny a komponens alapú megközelítés, amely lehetővé teszi a fejlesztők számára a komponensek újra felhasználását és a kód modularizációját. A React komponensek önmagukban zárt egységek, amelyek különböző feladatokat elláthatnak a felhasználói felületeken, például gombok, űrlapok, listák stb. Az egyedi komponensek

könnyedén kombinálhatók egymással, így a fejlesztőknek nem kell újra megírniuk a kódot, hanem egyszerűen felhasználhatják a meglévő komponenseket.

Emellett más technológiai óriások is felfedezték a React előnyeit. Például a Netflix, a PayPal, az Airbnb és a Dropbox is a React-et használja az alkalmazásaik fejlesztéséhez. Ezek a vállalatok magas forgalommal és komplex felhasználói felületekkel rendelkeznek, és a React segítségével könnyedén kezelhetik ezeket a kihívásokat.

Az open-source software (OSS) közösség is hozzájárult a React népszerűségének növekedéséhez. Számos harmadik által (Third party) fél készített könyvtár és eszköz érhető el a React-hez, amelyek további lehetőségeket kínálnak a fejlesztőknek. Ilyen példa a Redux, amely egy állapotkezelő könyvtár, vagy a React Router, amely segít az alkalmazások útvonalainak kezelésében.

A JavaScript önmagában egy típusfüggetlen programozási nyelv, de egyes keretrendszerek, mint a Next.js, a Microsoft által kifejlesztett TypeScript nyelvet használják, amely már megengedi a beépített és személyre szabott típusok használatát.

2.3. ábra. React logó

2.2.3. A Google Firebase platform

Mivel a React a webes alkalmazások frontend-jének fejlesztésére szolgál, szükségem volt egy backend szerverre és egy DBMS, -re, amely kiszolgálja a frontend-et, a Google által fejlesztett Firebase platformot és API-kat építettem be az alkalmazásba. A Google Firebase (2.4) egy teljes körű fejlesztői platform, amely különféle eszközöket és szolgáltatásokat kínál a fejlesztőknek, hogy könnyedén hozzanak létre, teszteljenek és üzemeltessenek webes és mobilalkalmazásokat.

A Firebase Realtime Database egy NoSQL alapú adatbázis, amely valós idejű adatszinkronizációt tesz lehetővé az alkalmazások között. Ez azt jelenti, hogy az adatok automatikusan frissülnek minden csatlakozott eszközön, így a felhasználók valós idejű élményt élvezhetnek. Ez különösen hasznos például csevegőalkalmazások vagy valós idejű játékok fejlesztésekor.

A Firebase Authentication lehetővé teszi a felhasználók egyszerű és biztonságos hitelesítését. Támogatja az e-mail és jelszó, a szociális média hitelesítés (pl. Google, Facebook, Twitter) és más autentikációs módokat is. A Firebase emellett lehetőséget biztosít a felhasználói fiókok-, profilok kezelésére és jogosultságkezelésre is.

A Firebase Cloud Storage segítségével könnyedén tárolhatóak és kezelhetőek az alkalmazásban használt fájlok, például képek, hangfájlok vagy videók. Az egyszerű API-k lehetővé teszik a fájlok feltöltését, letöltését és megosztását. Emellett a Firebase Hosting segítségével egyszerűen és gyorsan kiszolgálhatod az alkalmazásod statikus fájljait a világ minden tájáról.

A Storage Bucket-ek egy nagy tárolóhelyet jelentenek a fájlok (például képek, hangfájlok, videók stb.) biztonságos tárolására a felhőben. A Firebase Storage lehetővé teszi a fájlok feltöltését, letöltését, törlését és megosztását egyszerű API-k segítségével. Emellett automatikusan kezeli a fájlok hozzáférési jogosultságait és a CDN révén biztosítja a gyors és megbízható fájlletöltést a felhasználóknak.

2.4. ábra. Firebase logó

2.3. Az AES titkosítás

2.3.1. Bevezető

Az AES egy blokk titkosítási algoritmus, amelyet két belga kriptográfus tervezett 1997-ben. Az algoritmus az 1990-ben meghirdetett nyilvános pályázat nyertese lett, és 2001-ben a NIST (National Institute of Standards and Technology) elfogadta az Egyesült Államok kormányzati szervezetei számára ajánlott titkosítási szabványnak. Mivel az említett pályázat elbírálása publikusan történt, ezért valószínűleg nem történt befolyás a díj megítélését illetően. Az algoritmus egy változata a Rinjdael algoritmusnak, amely eredetileg több blokkmérettel is dolgozott és ebből választották ki az AES-t, amelynek blokkmérete 128 bit.

Az AES blokk mérete 128 bit, és a kulcs mérete lehet 128, 192 vagy 256 bit. Az AES hatékonysága körülbelül 109 Mb/s, amely természetesen függ az adott hardver tulajdonságaitól, és szakértők szerint a 256 bites kulcsméret biztonsága "örök" időkre szól.

Az AES nem használja a Feistel-sémát, mint a DES (Data Encryption Standard), hanem iteratív szerkezetű. Az algoritmus a kulcsméret alapján különböző számú körökben végzi a titkosítást, amelyekhez korkulcsokat generál. A 128 bites kulcs esetén a körök száma 10, a 192 bites kulcs esetén a körök száma 12, míg a 256 bites kulcs esetén a körök száma 14.

Az AES helyettesítést és permutációt alkalmaz a blokkon belül, és véges testek felett végez aritmetikai és műveletet. Az algoritmus térnyerésére az szolgált, hogy a művelet végzés egész számokkal történik, ezért a szerzők egy olyan algoritmust fejlesztettek, ahol az összeadás, kivonás, szorzás, osztás után is halmaz béli elemet kapunk. Az AES véges testként használja a bináris együtthatós polinomokat a GF(2^n) felett, ahol n = 8 fokszámnál kisebb.

Az AES minden műveletet 8 biten végez, és az összes 30 irreducibilis polinom közül a következőt használja: $x\hat{}8 + x\hat{}4 + x\hat{}3 + x + 1$. Az AES minden bájtot a $GF(2\hat{}8) = GF(2)[x]/(x\hat{}8 + x\hat{}4 + x\hat{}3 + x + 1)$ testelemeként kezel.

Az AES-t általában három algoritmus alkotja: a kulcsgenerálás, a titkosítás és a visszafejtés. Az AES a bemeneti 128 bites állapotot (state) egy 4x4-es mátrix formájában

kezeli, és az algoritmus minden iterációja során helyettesítő és permutációs műveleteket végez el a blokkon belül.

2.3.2. Az algoritmus elmélete

Az AES a bemeneti adatokat 128 bites blokkokra bontja. A bemeneti adatokat több körön keresztül módosítja míg el nem jutunk a titkosított adatig. A módosításokat az algoritmus több lépésben hajtja végre. Ezeket a lépéseket altranszformációknak nevezzük.

Az altranszformációk három típusúak lehetnek: AddRoundKey, SubBytes és ShiftRows. Az AddRoundKey lépés során az aktuális kör kulcsát XOR művelettel adja hozzá adatblokkhoz. A SubBytes lépés a bemeneti blokk minden elemén végigmegy egy előre meghatározott nemlineáris függvénnyel. A ShiftRows lépés során az adatblokk minden sorát egy bizonyos számmal rotáljuk.

Összességében az AES titkosítási algoritmus magában foglalja a bemeneti adatok blokkokra bontását, majd a bemeneti blokkokon végrehajtja az altranszformációkat több körön keresztül, amíg meg nem kapja a titkosított adatot. Az algoritmus célja az adatok biztonságos és hatékony titkosítása a megfelelő védelem érdekében.

2.3.3. Az algoritmus gyakorlatban

Az alábbi kódrészlet az AES algoritmus egyik használati módja JavaScript programozási nyelven a CryptoJS könyvtárcsomag segítségével (2.1).

```
export const encryptData = text => {
  const salt = CryptoJS.lib.WordArray.random(128 / 8);
  const key = CryptoJS.PBKDF2(secretPass, salt, {
    keySize: 256 / 32,
    iterations: 1000,
  });
  const iv = CryptoJS.lib.WordArray.random(128 / 8);

  const encrypted = CryptoJS.AES.encrypt(JSON.stringify(text), key, {
    iv: iv,
    mode: CryptoJS.mode.CBC,
    padding: CryptoJS.pad.Pkcs7,
    tag: true,
  });

  const data = {
    ciphertext: encrypted.ciphertext.toString(CryptoJS.enc.Base64),
    iv: iv.toString(CryptoJS.enc.Base64),
    salt: salt.toString(CryptoJS.enc.Base64),
    tag: true,
  };
  return JSON.stringify(data);
};
```

2.1. kódrészlet. Titkosítás példakód.

A kód első lépése a *salt (só)* generálása, ami egy 128/8 bájt szekvencia, amely segítségével meghatározásra kerül majd a key (kulcs). A *CryptoJS.lib.WordArray.random(128 / 8)* kódsor a **CryptoJS** könyvtárban található *random()* függvény hívásával egy véletlenszerű 128-bites szekvenciát generál, majd beilleszti az adatokat a WordArray osztályba.

A következő lépésben a kulcs előállítása történik meg a *secretPass (jelszó)* és a só használatával a kulcsderiváló függvény segítségével. A kulcs előállítása a *CryptoJS.PBKDF2()* függvénnyel történik, amely egy kulcstervező függvény. Az első paraméter a titkosításhoz használt jelszó, a második argumentum a só, a harmadik pedig a kulcs hosszát és az iterációk számát határozza meg. Ennél az algoritmusnál nagyon fontos odafigyelni, hogy a jelszó szigorúan titkos információ, vagyis ezt biztonságosan ajánlott eltárolni szerver oldalon. A többi paraméter publikus, mert önmagukban nem elegendőek a titkosított szöveg visszafejtéséhez.

Az inicializáló *vektor (iv)* generálása következik. Az IV egy véletlenszerű bájt szekvencia, amelynek hossza megegyezik a blokk méretével (128 bit), és a titkosítás során használják, hogy azonos adatok esetén is véletlenszerű kimenetet generáljon.

Az adat titkosítása a *CryptoJS.AES.encrypt()* függvénnyel történik. Az adatot először JSON formátumba alakítjuk, majd az AES algoritmust a kulcs, az IV és további paraméterek (mód, padding és tag) megadásával alkalmazzuk. A *mode* a blokktitkosítási mód (BCM) kiválasztására szolgál, ezek között található a ECB, CBC, OFB vagy CFB. A mi esetünkben a CBC (Cipher Block Chaining) mód van használva.

A *padding* a blokkok kitöltési módját határozza meg, itt a **Pkcs7** (Public Key Cryptography Standards 7-es szabványa) van használva. A Pkcs7 padding szabvány szerint, ha az üzenet hossza nem egész blokk méretű, akkor azt szükséges kiegészíteni. Minden hiányzó bájt felveszi a hiányzó bájtok számának megfelelő értéket. Ha például az utolsó blokkban 3 hiányzó bájl van, akkor a blokk utolsó bájtjai a 0x03 értéket vesz fel.

Az textitiv az inicializáló vektort tartalmazza, amit korábban generáltunk. A textittag beállítása igazra van állítva, hogy az üzenet hitelesítése (integritásának ellenőrzése) is megtörténjen a titkosítás során. A tag az üzenet elejére kerül beillesztésre a titkosítás során, és végül az eredeti üzenet végén kerül ellenőrzésre a helyessége.

Az utolsó lépés a textittitkosított adat (ciphertext), az IV, a só és az üzenet hitelesítésének értéke (tag) összekapcsolása és JSON formátumba rendezése és visszatérítése.

Az alábbi példakód a visszafejtést valósítja meg (2.2):

```
export const encryptData = text => {
  const salt = CryptoJS.lib.WordArray.random(128 / 8);
  const key = CryptoJS.PBKDF2(secretPass, salt, {
    keySize: 256 / 32,
    iterations: 1000,
  });
  const iv = CryptoJS.lib.WordArray.random(128 / 8);

  const encrypted = CryptoJS.AES.encrypt(JSON.stringify(text), key, {
    iv: iv,
    mode: CryptoJS.mode.CBC,
    padding: CryptoJS.pad.Pkcs7,
    tag: true,
  });

  const data = {
    ciphertext: encrypted.ciphertext.toString(CryptoJS.enc.Base64),
    iv: iv.toString(CryptoJS.enc.Base64),
    salt: salt.toString(CryptoJS.enc.Base64),
    tag: true,
  };
  return JSON.stringify(data);
};
```

2.2. kódrészlet. Visszafejtés példakód.

A **decrypt** függvény az **encrypt** függvénnyel ellentétes sorrendben dolgozza fel az adatokat. A paraméterként kapott titkosított szövegből JSON objektumot állít elő. Ezután ebből az objektumból megkapja a só és az iv Base64 formátumból az értékeket. A kulcs előállítása a *PBKDF2* függvénnyel történik a jelszó és a só segítségével. A titkosított adat, az iv, a só és a hitelesítési tag felhasználásával sikeresen vissza tudjuk fejteni az eredeti üzenetet.

2.3.4. Alkalmazások és érdekességek

- Az AES algoritmusnak 128, 192 és 256 bites változatai vannak, amelyek mindegyike különböző szintű biztonságot nyújt.
- Az algoritmus rendkívül hatékony, amely lehetővé teszi a titkosított adatok nagy sebességű feldolgozását.
- Nyilvánosan elérhető és ingyenesen használható, ami azt jelenti, hogy bárki használhatja és integrálhatja az alkalmazásába.
- Ennek az algoritmusnak számos könyvtárcsomagban van megvalósítva. Léteznek egyaránt publikus és privát forráskódok, amelyek közül néhány optimalizálva van a hardveres és szoftveres rendszerekhez.

- Az alkalmazása különböző területeken népszerű, például az online banki- és pénzügyi tranzakciókban, a VPN (Virtual Private Network) rendszerekben, a Wi-Fi hálózatokban, az adattároló eszközökön.
- Az AES használata akkor biztosít teljes biztonságot, ha ezt együtt használják egyéb kriptográfiai primitívekkel. Ilyen az AES-en kívül az RSA vagy a Hashmap. A kulcsfontosságú védelmi rendszerekben, mint például a HTTPS, SSH vagy TLS, használnak több rétegű védelmi mechanizmusokat az AES kiegészítéseként.
- A 256 bites titkosítási kulcsokat sokkal nehezebb brute-force módon támadni, mint egy 128 bites kulcsot. Azonban az utóbbi is olyan hosszú időbe telik kitalálni, még hatalmas számítási kapacitás mellett is, hogy az előrelátható jövőben nem lesz probléma, mert egy támadónak is hatalmas számítási kapacitásra lenne szüksége a szükséges brute-force (nyers-erő módszere) generáláshoz.
- Azonban a 256 bites kulcsokhoz is több feldolgozási teljesítmény szükséges és hosszabb ideig tarthat a generálásuk. Amikor az energiafogyasztás problémát jelent, különösen kis eszközök esetében, vagy a késleltetés valószínű, a 128 bites kulcsok jobb választásnak számítanak.

2.4. RSA!!!!

2.5. Kutatási kérdések

Az alkalmazás fejlesztése során a következő kérdésekre próbáltam válaszokat keresni:

- Hogyan lehet biztonságosabbá tenni az elektronikus jegyvásárlást és azok felhasználását?
- Mely technológiák segítségével lehet gyors és hatékony online üzletet tervezni és megvalósítani?
- Hogyan lehet a megvalósított rendszert hosszú távon karban tartani és felügyelni?
- Hogyan lehetséges a rendszer automatizálása valós időben?

2.6. Célkitűzések

3. fejezet

Rendszerspecifikáció és architektúra

3.1. Beépített szoftverek, könyvtárak

Dolgozatom ezen részében először vizsgáljunk meg olyan beépített differenciálegyenlet megoldó szoftvereket, melyeket már megemlítettem. Ezek közül én a Matlab és a Boost - Odeint beépített programját használtam és vizsgáltam meg. Mindkét esetben közönséges differenciálegyenletek (ODE) megoldására, az előre megírt algoritmus segítségével, melynek hátterében természetesen a Dormand-Prince módszer áll.

$$x = 1 + 1 + 1 + 1 \tag{3.1}$$
$$= 4. \tag{3.2}$$

$$y = mx + b$$
$$z = nw + c. \tag{3.3}$$

$$\int_0^1 \sum_a^b \prod_\alpha^\beta . \tag{3.4}$$

$$\frac{12}{34}.$$

3.1.1. Matlab - ode45

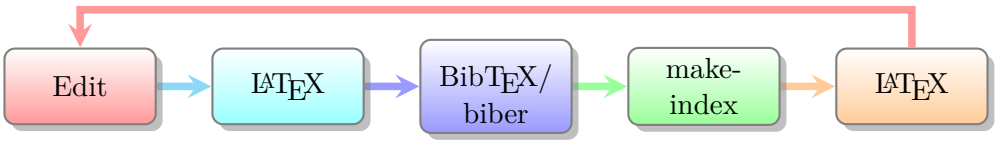
A Matlab egy programcsomag és egyben egy technikai nyelv is, mely magas szinten lehetőséget biztosít számítások elvégzésére, modellezésre, szimulációra, megjelenítésre, vizualizációra és számos más hasznos mérnöki munka elvégzésére. Esetünkben a legfontosabb, hogy könnyedén tudunk közönséges differenciálegyenleteket megoldani az ode45 program segítségével. Emellett az eredményeket egy jól megtervezett felületen ki is tudjuk ábrázolni. Az alábbi példában jól látható, hogy milyen egyszerű és kényelmes a használata:

```
f = @(t,y) y;
t = [0 10];
y0 = 1.0;
ode45(f, t, y0);
plot(t, y(:));
```

3.1. kódrészlet. Matlab példakód diff. egyenlet megoldására.

A fenti bemenetre $n = 100$ - szor lefuttattuk az algoritmust és a következő időeredményeket kaptuk:

- Futási idők **átlaga**: 0.0043 sec
- Futási idők **minimuma**: 0.0026 sec
- Futási idők **maximuma**: 0.1504 sec



3.1.2. Boost - Odeint

Egy másik előre megírt közönséges differenciálegyenlet megoldó a Boost könyvtárcsomag Odeint nevezetű könyvtára. Ez egy modern C++ nyelven írt csomag, lényeges jellemzői, hogy **nagy teljesítményre** képes és **nagyon magas szinten** (absztraktan - Template Metaprogramming) van megírva, így **rugalmas** és könnyen integrálható különböző rendszerekbe. Emellett rugalmas a bementi adatok típusát illetően is. Ugyanakkor jó tudni, hogy ez a nagyon absztrakt megvalósítás hátrány is lehet, mert bizonyos esetekben nagyon nehéz megérteni vagy megoldani egy felmerülő problémát. Most lássunk egy egyszerű példát a használatára:

```
#include <iostream>
#include <boost/numeric/odeint.hpp>

using namespace std;
using namespace boost::numeric::odeint;

typedef runge_kutta_dopri5<double> stepper_type;

void rhs( const double x , double &dxdt , const double t ) {
   dxdt = 3.0/(2.0*t*t) + x/(2.0*t);
}

void write_cout( const double &x , const double t ) {
   cout << t << '\t' << x << endl;
}

int main() {
```

```
  double x = 0.0;
  integrate_adaptive( make_controlled( 1E-12 , 1E-12 , stepper_type() ) ,
  rhs , x , 1.0 , 10.0 , 0.1 , write_cout );
}
```

3.2. kódrészlet. Odeint példakód.

Az előző kódrészlet eredménye a következő:

- Futási idők **átlaga**: 0.0989 sec
- Futási idők **minimuma**: 0.09 sec
- Futási idők **maximuma**: 0.099 sec

3.2. Általam megvalósított szoftverek

```
function [yy, tt, timeSpent] = fun_dopri45(f, y0, t0, tf, tolerance)
   ...
   while t < tf
      if (t+h >= tf)
         h = tf-t;
      end

      % Calculate k1, k2, ... , k7
      k1 = h*feval(f, t+h*C(1), y);
      k2 = h*feval(f, t+h*C(2), y + A2(1)*k1);
      ...
      k7 = h*feval(f, t+h*C(7), y + A7(1)*k1 + A7(3)*k3 + ... + A7(6)*k6);

      % Calcaulate the next point
      yt = y + A7(1)*k1 + A7(3)*k3 + A7(4)*k4 + A7(5)*k5 + A7(6)*k6;
      % Calculate the error
      err = abs(E(1)*k1 + E(3)*k3 + ... + E(7)*k7);

      if max(err) < tolerance
         t = t + h;
         y = yt;
         ...
      end

      % Calculate optimal step size
      scale = 1.25*(maxErr/tolerance)^(1/5);
      if scale > 0.2
         h = h / scale;
      else
         h = 5.0*h;
      end
   end
end
```

3.3. kódrészlet. Matlab kód ode45 használata nélkül.

- Futási idők **átlaga**: 0.0043 sec
- Futási idők **minimuma**: 0.0058 sec

- Futási idők **maximuma**: 0.0287 sec

A következő szoftvert, amit bemutatok **Java** nyelven írtam objektum orientáltan, a fejlesztés során pedig Eclipse fejlesztői környezetet használtam. Ez a szoftver három egységből áll: fő -, differenciálegyenlet megoldó - és kifejezés kiértékelő egység (ezt a legnehezebb megvalósítani vagy megtalálni a megfelelő könyvtárat). Az alkalmazás nem rendelkezik grafikus felhasználói felülettel, a bemeneti adatokat egy szöveges állományból olvassa be (amelynek jól meghatározott szerkezete van) és a kívánt eredményeket a standard kimenetre írja ki. A kifejezés kiértékelő egység megírásánál felhasználtam egy előre elkészített Java könyvtárat, melyet *JEP*-nek neveznek. A szoftverben arra használtam fel, hogy egy sztringként megadott kifejezést kiértékeltem és elvégeztem a segítségével. Mindezt úgy csinálja, hogy a háttérben felépít egy kifejezésfát, aminek a leveleiben lesznek az értékek, csúcsaiban a műveletek, zárójelek, stb. (lásd az alábbi ábrát):

- Futási idők **átlaga**: 0.1422 sec
- Futási idők **minimuma**: 0.123 sec
- Futási idők **maximuma**: 0.341 sec

Harmadiként lássuk a **C++ szoftvert**, amelyet szintén objektum orientáltam valósítottam meg. Ennek a szoftvernek a szerkezete hasonló a Java szoftver szerkezetéhez. Ebben az esetben is szükség volt egy kifejezés kiértékelő könyvtárra, hogy ne kelljen egy sajátot írni. Először kipróbáltam egy *muparser* nevű könytárat, aztán egy másikat is, aminek *ExprTk* (Expression Toolkit Library) a neve. Fontos elmondani, hogy mindkét könyvtár ingyenesen elérhető és használható. Az első könyvtárat nehezebb volt hozzáadni a szoftverhez és mivel több fájlból állt több időbe került a fordítása is. A legfőbb ok, amiért mégis a másodikat használtam az volt, hogy jelentősen gyorsabb és hatékonyabb volt az én szoftveremben. Tehát végül az *ExprTk* könyvtárat használtam a jobb teljesítménye és könnyebb integrálhatósága miatt.

- Futási idők **átlaga**: 0.1086 sec
- Futási idők **minimuma**: 0.096 sec
- Futási idők **maximuma**: 0.214 sec

Végül nézzük meg az utolsó, Dormand-Prince módszeren alapuló szoftvert, amely egy **Android alkalmazás**. Mint tudjuk napjainkban a mobileszközök nagyon elterjedtek és teljesítményük is jelentősen megnőtt, lassan felveszik a versenyt a személyi számítógépekkel. Ezért mindenképp szerettem volna az algoritmust megvalósítani mobileszközökre is és megnézni itt is a teljesítményt. Mivel az Android alkalmazások fejlesztésénél Java nyelvet használunk, így könnyű dolgom volt, hiszen a Java szoftverből át tudtam venni a már jól megírt és elkülönített osztályokat. Ugyanazt a *JEP* könyvtárat használtam itt is a kifejezések kiértékelésére, így majd az eredmények összehasonlítását is könnyebbé tettem. A Java és C++ szoftverrel ellentétben rendelkezik egy kis grafikus felhasználói felülettel, de a bemeneti adatokat itt is egy szöveges állományból olvassuk be, mert nem túl kényelmes azt a sok számadatot, meg egyenletet beviteli mezőkön keresztűl beírni.

- Futási idők **átlaga**: 37.9015 sec
- Futási idők **minimuma**: 31.116 sec
- Futási idők **maximuma**: 70.876 sec

3.3. Szoftverek összehasonlítása

Az előző alfejezetekben ismertettem a már létező Dormand-Prince módszeren alapú differenciálegyenlet megoldók közül kettőt és négy saját megvalósítást is. Mindegyik esetében láthattunk futási időket és számadatokat, azonban nem láttuk ezeket egymás mellett. Ebben az alfejezetben összegezzük és összehasonlítjuk a kapott eredményeket.

Fontos, hogy minden algoritmust ugyanazon a hardveren teszteljünk, mert csak így reálisak és összehasonlíthatóak a mérési adatok. Esetünkben használt hardver konfigurációja:

- Intel Core i5-7200U, 2.50 GHz processzor, 8.00 GB RAM memória

Természetesen az Android alkamazást csak mobileszközökön lehetett vizsgálni, itt két készüléket használtam a tesztelésre:

- Motorola Moto E2, Quad-core 1.2 GHz processzor, 1 GB RAM memória
- Samsung Galaxy Core Prime, Quad-core 1.2 GHz processzor, 1 GB RAM memória

Technológia	**Átlagidő (s)**	**Min. idő (s)**	**Max. idő (s)**	**Hardver**
Matlab - ode45	0.0032	0.0029	0.0038	Intel Core i5
Boost - Odeint	0.0989	0.0900	0.1290	Intel Core i5
Matlab	0.0062	0.0060	0.0066	Intel Core i5
Java	0.2224	0.1970	0.3020	Intel Core i5
C++	0.1047	0.1010	0.1240	Intel Core i5
Android	37.9015	31.1160	70.8760	Moto E2
Android	35.9987	29.4200	87.6120	Core Prime

3.1. táblázat. Mérési eredmények $n = 10$ tesztesetre.

Technológia	**Átlagidő (s)**	**Min. idő (s)**	**Max. idő (s)**	**Hardver**
Matlab - ode45	0.0043	0.0026	0.1504	Intel Core i5
Boost - Odeint	0.0912	0.0900	0.0990	Intel Core i5
Matlab	0.0067	0.0058	0.0287	Intel Core i5
Java	0.1422	0.1230	0.3410	Intel Core i5
C++	0.1086	0.0960	0.2140	Intel Core i5
Android	–	–	–	Moto E2
Android	–	–	–	Core Prime

3.2. táblázat. Mérési eredmények $n = 100$ tesztesetre.

3.4. Differenciálegyenletek megoldása GPU-n

A továbbiakban nézzük meg a két algoritmus magjának szekvenciális és párhuzamosított változatait:

```
for (int i = 0; i < numberOfVariables; ++i) {
   mResult.at(i)[j + 1] = mResult.at(i)[j] + mInputs->getStepSize() *
   (mFunctionsParsers.at(i)->computeFunctionValue(values));
}
```

3.4. kódrészlet. Euler módszer szekvenciális kód.

```
__global__ void computeFunctionsValuesKernel(double* resultValues,
double* previousValues, double* functionValues, double stepSize, int N) {
   int i = threadIdx.x;

   if (i < N) {
      resultValues[i] = previousValues[i] + stepSize * functionValues[i];
   }
}
```

3.5. kódrészlet. Euler módszer párhuzamosított kód.

```
for (int k = 0; k < numberOfVariables; ++k) {
   mResult.at(k)[i + 1] = mResult.at(k)[i] + (mInputs->getStepSize() / 6)*
   (K[0][k] + 2 * K[1][k] + 2 * K[2][k] + K[3][k]);
}
```

3.6. kódrészlet. Runge-Kutta módszer szekvenciális kód.

```
__global__ void computeValuesKernel(double* resultValues, double* K,
double* previousValues, double stepSize, int numberOfVariables) {
   int i = threadIdx.x;

   if (i < numberOfVariables) {
      resultValues[i] = previousValues[i] + ((stepSize / 6) *
      (K[i] + 2 * K[i + 1 * numberOfVariables] +
      2 * K[i + 2 * numberOfVariables] + K[i + 3 * numberOfVariables]));
   }
}
```

3.7. kódrészlet. Runge-Kutta módszer párhuzamosított kód.

Nézzünk meg pár mérési eredményt a következő differenciálegyenlet rendszerre:

$$\begin{cases} y_1'(t,y) = y_1 \\ \vdots \\ y_n'(t,y) = y_n \\ y_1'(t_0) = 1.0 \\ \vdots \\ y_n'(t_0) = 1.0 \end{cases}, t \in [0,100], n = 5 \tag{3.5}$$

	CPU sec (Intel Core i5)		**GPU sec (GeForce 940MX)**	
	Euler	Runge-Kutta	Euler	Runge-Kutta
$h = 10.0$	0.006	0.036	0.587	0.036
$h = 1.0$	0.074	0.523	0.841	0.490
$h = 0.1$	0.689	4.987	1.565	3.399
$h = 0.01$	7.202	52.800	14.321	50.110
$h = 0.001$	71.071	500.999	98.445	321.965

3.3. táblázat. Mérési eredmények $n = 5$ egyenlet esetén.

$$\begin{cases} y_1'(t,y) = y_1 \\ \vdots \\ y_n'(t,y) = y_n \\ y_1'(t_0) = 1.0 \\ \vdots \\ y_n'(t_0) = 1.0 \end{cases}, t \in [0,100], n = 10 \tag{3.6}$$

	CPU sec (Intel Core i7)		**GPU sec (GeForce 950M)**	
	Euler	Runge-Kutta	Euler	Runge-Kutta
$h = 10.0$	0.039	0.256	1.070	0.222
$h = 1.0$	0.320	2.482	0.994	2.087
$h = 0.1$	3.042	23.962	4.143	20.978
$h = 0.01$	30.812	241.977	35.009	206.842
$h = 0.001$	305.092	2394.930	344.485	2067.870

3.4. táblázat. Mérési eredmények $n = 10$ egyenlet esetén.

4. fejezet

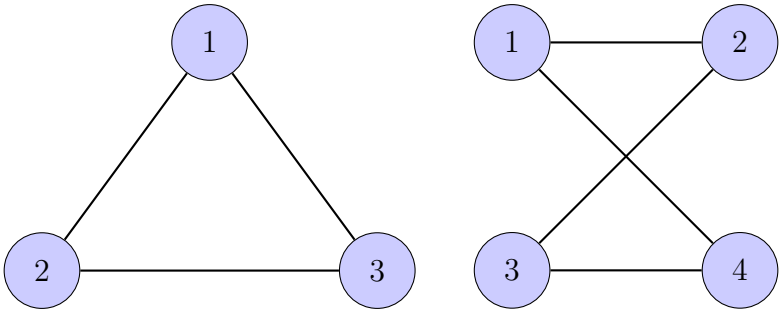
Eredmények, következtetések

4.1. Beépített szoftverek esetén

A 3. fejezetben bemutattam két beépített, előre megvalósított szoftvert, melyek a Matlab ode45 programja és a Boost - Odeint könyvtár. Az elért eredmények és tesztek azt mutatják, hogy a Matlab ode45 differenciálegyenlet megoldója sokkal gyorsabb és hatékonyabb, mint az Odeint. Ha a tesztesetekben mért átlagidőket összehasonlítjuk láthatjuk (3.3. alfejezet), hogy az ode45 $20 - 30$ -szor gyorsabb a Odeintnél. Ez talán annak is köszönhető, hogy a Matlab egy nagyon komoly szoftver, amelynek programcsomagjain mérnökök és programozók százai (vagy akár ezrei) dolgoznak, így természetes, hogy az algoritmusok jobban optimizáltak és hatékonyabbak az ingyenes szoftvereknél. A továbbiakban összegezzük, hogy a két technológiának milyen előnyei és hátrányai vannak vagy éppen miért érdemes/nem érdemes használni őket, lásd [LS15]

Matlab ode45 előnyei:

- nagyon egyszerű a használata, nem igényel komoly programozási ismereteket
- könnyű beépíteni és összekötni más Matlab programokkal
- a kapott eredményeket mátrix vagy vektor típusokban téríti vissza, ami könnyűvé teszi az eredmények további kezelését
- az eredményeket grafikus felületen azonnal meg tudjuk jeleníteni
- jobb eredményeket produkált, mint az Odeint
- jól dokumentált, sok példa van a használatára



4.1. ábra. Egyszerű gráf TIKZ segítségével

Matlab ode45 hátrányai:

- komoly hátránya az Odeinttel szemben, hogy **fizetni kell a használatáért**
- nagyon sok memóriát használ, komolyan igénybeveszi a számítógép erőforrásait

Odeint előnyei:

- ingyenes és nyílt forráskódú, használható személyi és kereskedelmi célokra egyaránt
- nagyon rugalmas, absztrak, így könnyedén változtatható a bemeneti adatok típusa vagy struktúrája
- C++ nyelven írodott, támogatva a modern programozási technológiákat (Generikus programozás, Template Metaprogramming)

Odeint hátrányai:

- használata nehezebb, mint a Matlab ode45 programé, szükséges a C++ programozási nyelv ismerete
- a kapott eredményekkel nem olyan könnyű bánni, mint a Matlab esetében
- absztraktsága miatt nehéz a felmerülő problémákat megoldani
- dokumentáltsága jóval szegényesebb, mint a Matlabé

4.2. Saját szoftverek esetén

Saját szoftverek esetében sikerült négy különböző technológia segítségével megvalósítani a Dorman-Prince differenciálegyenlet megoldó algoritmust (lásd 3.2. alfejezet). A felhasznált technológiák: Matlab, Java, C++ és Android voltak.

Ezen technológiák közül az Androidos szoftverrel kapcsolatban már előzetes félelmeink voltak, mivel azt feltételeztük, hogy bármennyire is fejlettek napjainkban a mobileszközök, mégis hardveresen nem lesznek elegek ahhoz, hogy versenybe tudjanak szállni a számítógépekkel. Néhány teszt után feltételezéseink beigazolódtak, láthatjuk a 3.3. alfejezet teszteseteiben is, hogy az Android szoftver mennyire gyengén teljesített (mind a Motorola Moto E2, mind a Samsung Galaxy Core Prime esetében). Összehasonlítva a többi algoritmussal az átlagidőket nézve $150 - 160$ - szor lassabb a Javanál és $300 - 350$ - szer a C++ - nál! Tehát azt a következtetést vonhatjuk le, hogy nem érdemes mobileszközökön differenciálegyenleteket oldani, mivel túlságosan nagy a hardver igénye és egyelőre ezen a téren nem képesek tartani a lépést a számítógépekkel.

A további három technológia közül (Matlab, Java, C++) meglepő módon itt is a Matlab teljesített a legjobban, igaz, hogy ebben az esetben már nem használtuk az ode45 programot, hanem megírtam én a saját függvényemet. Ez a teljesítményen is meglátszott, mert az általam írt függvény nem tudott jobban teljesíteni a tesztek alatt, mint az ode45. Mindezek ellenére $35 - 40$ - szer gyorsabb volt a Javanál és megközelítőleg $15 - 20$ - szor gyorsabb a C++ - nál.

A Java és C++ szoftvereket összehasonlítva elmondhatom, hogy az esetek többségében a C++ körülbelül 2 - szer volt gyorsabb a Javanál, ami megfelel az előzetes elvárásoknak.

Amit nagyon fontosnak tartok kihangsúlyozni, hogy az általam megvalósított C++ szoftver a tesztek során nagyon jól teljesített, felvette a versenyt az Odeint könyvtárral és az elért átlagok is csak nagyon kicsivel maradnak el az Odeint által produkált eredményektől (3.3. alfejezet).

Továbbá megvalósítottam az Euler és Runge-Kutta módszerek párhuzamosított változatait is CUDA technológia segítségével. Ebben az esetben a tesztek azt mutatták, hogy az Euler módszer esetében többe kerül a sok CPU és GPU memória közötti másolás művelete, mint amennyit nyerünk a számítások elvégzése során. Tehát ebben az esetben ez a fajta párhuzamosítási megközelítés nem éri meg. Ezzel ellentétben a Runge-Kutta módszer esetében a megközelítés eredményesnek bizonyult abban az esetben, ha az egyenletek száma nagy és a lépésköz kicsi. A 3.4 alfejezetben láthattuk, hogy abban az esetben ha az egyenletek száma $n = 10$ és a lépésköz $h = 0.001$, a GPU-n megközelítőleg 5 és fél perccel hamarabb lefutott az algoritmus, mint a CPU-n. Ezzel a párhuzamosítási módszerrel nem tudtuk kihasználni a videókártya által nyújtott maximális számítási kapacitást, de így is jelentős különbséget sikerült elérni a futási időket nézve, lásd [Kó8].

4.3. Összességében

A fentieket összegezve elmondhatom, hogy megéri előre megírt szoftvereket vagy könyvtárakat használni differenciálegyenletek megoldására. Nagyon megkönnyíthetik az eltünket egyszerűségükkel és nagy teljesítményükkel. Viszont fontos elmondani, hogy használatuk problémákkal is járhat, például fizetni kell értük vagy nem lehet belenyúlni az

algoritmusokba kedvünk szerint, esetleges fellépő hibák esetén nagyon nehéz lenyomozni a hiba forrását (vagy szinten lehetetlen).

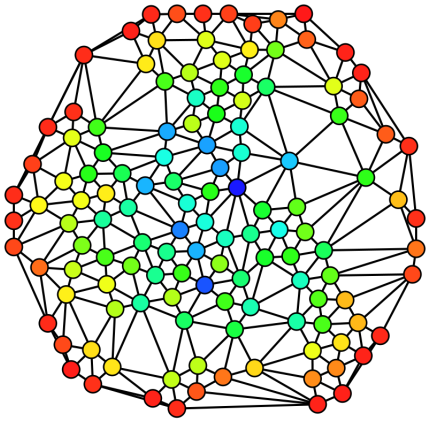
Saját algoritmusok terén bátran elmondhatom, hogy megéri a C++ technológiát választani és ezen a vonalon továbbhaladni egy esetleges saját könyvtár megírása, megvalósítása felé. Láthattuk, hogy az általam megírt C++ szoftver is felvette a versenyt az Odeint könyvtárral, ami szintén C++ technológiát alkalmaz, lásd [Ant07].

Egy másik vonal, amit érdemes sokkal jobban felderíteni az a CUDA technológiával és grafikus kártyával történő differenciálegyenlet megoldása. Láthattuk, hogy egy kis párhuzamosítás is jelentős időbeli különbséget jelenthet bizonyos algoritmusok esetében. Annak tudatában is, hogy a differenciálegyenletek megoldása nem a legjobban adatpárhuzamosítható feladatok közé sorolható azt mondom, hogy megéri ezzel a technológiával foglalkozni. Ennek kapcsán a legnagyobb motiváció számomra a jövőre nézve a parciális differenciálegyenletek párhuzamosításának megvalósítása és tanulmányozása, mivel ezeknél az egyenleteknél jobban ki lehet használni a videókártya rácsos szerkezetét.

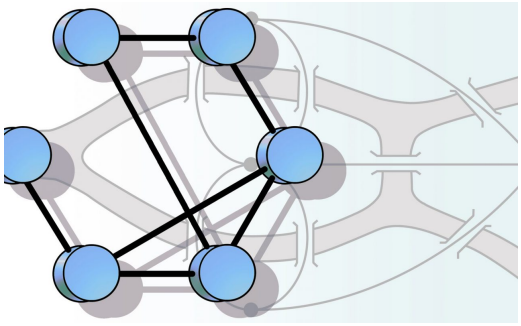
5. fejezet

Számítógépes elemzés

Ebben a fejezetben bemutatom az előző fejezetben elviekben ismertetett modelleket példákon és valós adatokon keresztül. A szoftver segítségünkre lesz abban, hogy megoldjuk a felmerülő differenciálegyenleteket és egyenletrendszereket, végül az eredményeket grafikus felületen is megtekinthetjük.



5.1. ábra. Gráf



5.2. ábra. Gráf 2

,ð„ï,

,ð„ï,,ð„ï,

6. fejezet

Szoftver

1. algoritmus: Euclidean Algorithm

Input: Integers $a \geq 0$ and $b \geq 0$
Output: GCD of a and b

```
1 while b ≠ 0 do
2 |   r ← a mod b;
3 |   a ← b;
4 |   b ← r;
5 end
```

2. algoritmus: How to write algorithms

Data: this text
Result: how to write algorithm with LaTeX2e

```
1  initialization;
2  while not at end of this document do
3  |   read current;
4  |   if understand then
5  |   |   go to next section;
6  |   |   current section becomes this one;
7  |   else
8  |   |   go back to the beginning of current section;
9  |   end
10 end
```

3. algoritmus: IntervalRestriction

Data: $G = (X, U)$ such that G^{tc} is an order.
Result: $G' = (X, V)$ with $V \subseteq U$ such that G'^{tc} is an interval order.

```
1  begin
2  |  V ⟵ U
3  |  S ⟵ ∅
4  |  for x ∈ X do
5  |  |  NbSuccInS(x) ⟵ 0
6  |  |  NbPredInMin(x) ⟵ 0
7  |  |  NbPredNotInMin(x) ⟵ |ImPred(x)|
8  |  end
9  |  for x ∈ X do
10 |  |  if NbPredInMin(x) = 0 and NbPredNotInMin(x) = 0 then
11 |  |  |  AppendToMin(x)
12 |  |  end
13 |  end
15 |  while S ≠ ∅ do
REM|  |  remove x from the list of T of maximal index
18 |  |  while |S ∩ ImSucc(x)| ≠ |S| do
19 |  |  |  for y ∈ S − ImSucc(x) do
20 |  |  |  |  { remove from V all the arcs zy : }
21 |  |  |  |  for z ∈ ImPred(y) ∩ Min do
22 |  |  |  |  |  remove the arc zy from V
23 |  |  |  |  |  NbSuccInS(z) ⟵ NbSuccInS(z) − 1
24 |  |  |  |  |  move z in T to the list preceding its present list
25 |  |  |  |  |  {i.e. If z ∈ T[k], move z from T[k] to T[k − 1]}
26 |  |  |  |  end
27 |  |  |  |  NbPredInMin(y) ⟵ 0
28 |  |  |  |  NbPredNotInMin(y) ⟵ 0
29 |  |  |  |  S ⟵ S − {y}
30 |  |  |  |  AppendToMin(y)
31 |  |  |  end
32 |  |  end
33 |  |  RemoveFromMin(x)
34 |  end
35 end
```

6.1. A szoftver bemutatása

Az általam írt szoftver egy **Java nyelv**en, **NetBeans IDE 8.0.1** fejlesztői környezetben írt asztali alkalmazás, amelynek fő funkcionalitása a kezdetiérték-probléma típusú differenciálegyenletek numerikus megoldása és ezen megoldások grafikus felületen való ábrázolása. A fő funkcionalitás mellett a szoftver tartalmaz még két kisebb funkcionalitást is, ezek közül az egyik a kétdimenziós függvényábrázolási lehetőség, a másik pedig a háromdimenziós függvények megjelenítésének lehetősége.

Az szoftver a differenciálegyenletek megoldásához a 3. fejezetben leírt numerikus eljárásokat alkalmazza.

A grafikus felhasználói felület megalkotásához a **Swing** (Java) komponens készletet használtam. A Swing használatával célom az volt, hogy egy felhasználóbarát és könnyen kezelhető felületet hozzak létre, amelyen a felhasználó könnyedén eligazodhat. Továbbá e komponenskészlet használata mellett szól az is, hogy a későbbiekben bemutatásra kerülő könyvtárak, melyek az ábrázolás megvalósítására használtam szintén Swing komponensekkel vannak megvalósítva.

Felhasználói felület, valamint a három funkcionalitás bemutatása képekben:

4. algoritmus: Switch használata

```
switch order do
    case bloody mary do
        Add tomato juice;
        Add vodka;
        break;
    case hot whiskey do
        Add whiskey;
        Add hot water;
        Add lemon and cloves;
        Add sugar or honey to taste;
        break;
    otherwise do
        Serve wine;
    end
end
```

6.2. A szoftver megírásához használt könyvtárak

A szoftver elkészítésénél szükségem volt néhány előre megírt osztálykönyvtárra, amelyek megkönnyítették a munkámat. Ezekről tudni kell, hogy nyílt forráskódúak, tehát bárki számára elérhetőek az interneten, továbbá azt is, hogy ezek is Java nyelvben íródtak, hasonlóan, mint az általam írt alkalmazás. A továbbiakban szeretném bemutatni ezeket a könyvtárakat és azt, hogy mire- és hogyan használtam fel őket.

- JMathPlot (https://sites.google.com/site/mulabsltd/products/jmathplot):
  - Java könyvtár, amelyet interaktív megjelenítésre, ábrázolásra fejlesztettek

  - gyors és könnyű utat biztosít tudományos adatok megjelenítésére Swing komponensek segítségével (nem használ openGL-t)
  - az általa biztosított saját komponenseket úgy lehet használni, mint bármely más Swing komponenst
  - a számomra legfontosabb tulajdonsága az, hogy két- és háromdimenziós ábrázolási lehetőséget biztosít, ezt használtam fel az alkalmazásomban

- JMathArray (https://sites.google.com/site/mulabsltd/products/jmatharray):
  - olyan Java könyvtár, amely alapvető matematikai, lineáris algebrai műveleteket biztosít számunkra
  - a könyvtár által biztosított statikus metódusok tömbökre alkalmazhatóak
  - a szoftverben arra használtam, hogy egy megadott intervallum két végpontja között egy bizonyos lépésközzel haladva egy tömböt tudjak feltölteni (inkrementálás)

- JEP (http://www.cse.msu.edu/SENS/Software/jep-2.23/doc/website/):
  - szintén egy Java könyvtár, amelyet különböző elemzésekre és kiértékelésekre fejlesztettek
  - segítségével egy szövegként (sztring-ként) megadott kifejezést könnyedén kiértékelhetünk, elvégezhetünk
  - a szövegként megadott kifejezésből a háttérben egy kifejezésfát épít fel, majd a későbbiekben ennek a fának a segítségével dolgozik
  - emellett sok általános matematikai függvény és konstans is bele van építve, amiket szintén könnyedén elérhetünk
  - az általam fejlesztett szoftverben a függvények sztringként adhatók meg egy beviteli mezőn keresztül, a JEP könyvtárat ezen függvények „parszolására” használtam fel

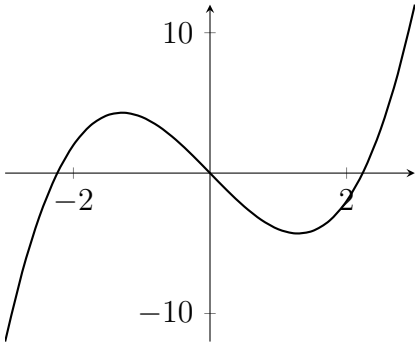
6.3. Diagramok

6.3.1. Use Case diagram

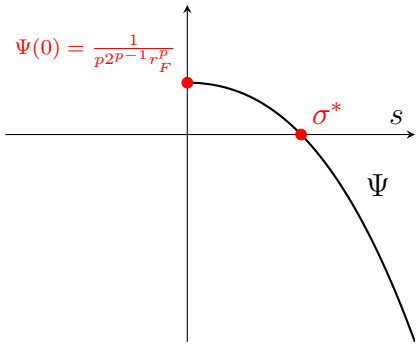
6.3.2. Osztálydiagram

A szoftver szerkezetileg két nagyobb részből (csomagból) áll, az egyik a felhasználói felület megalkotásához szükséges osztályokat tartalmazza, a másik pedig a differenciálegyenletek megoldására szolgáló osztályokat és a parszer osztályt, mely egy sztringként megadott függvény kiértékelésére szolgál.

Az implementációnál a felhasználói felület elemeit tartalmazó csomagot „View”-nak, a numerikus módszereket és a parszert tartalmazó csomagot „Model”-nek neveztem, emellett a 6.7-es ábrán megjelenik egy harmadik csomag is, amely tartalmazza a „MainClass”-t és egyben a main() metódust is. Az alábbi két diagramon láthatjuk a felsorolt csomagokat és a bennük lévő osztályokat, illetve a köztük lévő kapcsolatokat.

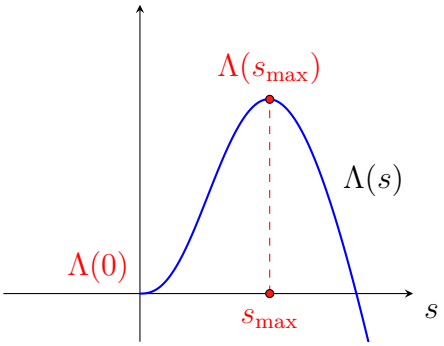


6.1. ábra. Az $x^3 - 5x$ függvény grafikus képe PGFPLOT-al

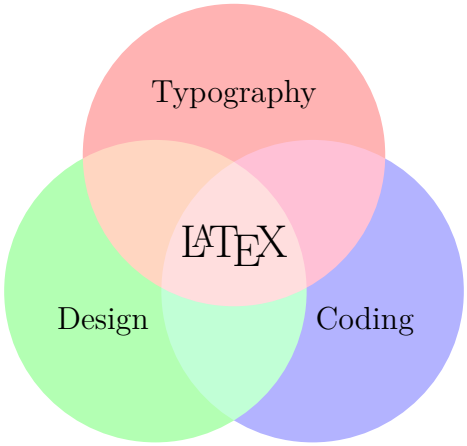


6.2. ábra. A Ψ grafikus képe

6.3.3. Szekvencia diagram



6.3. ábra. A $\Lambda(s)$ grafikus képe



6.4. ábra. Venn diagram TIKZ segítségvével

6.3.1. Definíció. *Legyen (X, d) és (Y, ρ) két metrikus tér, legyen $T : X \to Y$ egy leképezés. Azt mondjuk, hogy a T leképezés Lipschitz tulajdonságú, ha létezik egy olyan $L > 0$ szám amelyre*

$$\rho(Tx, Ty) \leq Ld(x, y) \; \forall x, y \in X.$$

Az L számot Lipschitz állandónak nevezzük.

Ha $T : X \to Y$ leképezés Lipschitz tulajdonságú, és az $L < 1$ akkor a T operátort **kontrakciónak** nevezzük. Azt mondjuk, hogy $x^* \in X$ fixpontja a T operátornak ha

$$Tx^* = x^*.$$

6.3.1. Tétel. *Banach féle fixponttétel Legyen (X, d) teljes metrikus tér és $T : X \to X$ leképezés egy kontrakció az $L < 1$ állandóval. Ekkor igazak a következő állítások:*

1. *T-nek egy és csakis egy x^* fixpontja.*
2. *Bárhogy választunk meg egy $x_0 \in X$ elemet, a $x_{k+1} = Tx_k$ sorozat konvergens és $Tx_k \to x^*$, ahol k természetes szám.*
3. *Igaz, hogy*

$$d(x_k, x^*) \leq \frac{L^k}{1-L} d(x_0, Tx_0).$$

Összefoglaló

Dolgozatomban differenciálegyenletek megoldásával foglalkoztam, amelyet különböző programozási technológiák segítségével valósítottam meg. Először ismertettem a differenciálegyenletek numerikus megoldásának elméleti alapjait, majd megvizsgáltunk és levezettünk három numerikus módszert az Euler-, a Runge-Kutta és a Dormand-Prince módszereket. Ezek közül a mai technológiákban leginkább használatos Dormand-Prince algoritmus esetében megnéztük, hogy milyen szoftverekben tálálhatjuk meg, mint alapértelmezett differenciálegyenlet megoldó. A továbbiakban ismertettem két modellt, a leukémia betegség alap modelljét és a hullámmozgás modelljét, ezzel is kihangsúlyozva a téma fontosságát, hogy mennyire fontos az időtényező bizonyos problémák egyenleteinek medoldásánál. Ezek után részletesen is mégnéztük, hogy milyen szoftvereket alkalmaztam és alkottam a differenciálegyenletek és rendszerek megoldására. A szoftvereket két kategóriába osztotottuk fel, az első a már létező szoftverek kategóriája, a másik pedig az általam megvalósított szoftverek csoportja volt. Az első kategóriában ismertettem két technológiát, a Matlab által nyúltott ode45 beépített megoldót és a Boost könyvtárcsomagban található Odeint nevű könyvtárat. Az általam írt szofverek csoportjában négy megvalósítást mutattam be ezek a Matlab, Java, C++ és Android technológiák segítségével készültek. Emellett megnéztük, hogy mennyire hatékonyan lehet párhuzamosítani a differenciálegyenletek megoldását CUDA technológia segítségével és a grafikus kártyát (GPU-t) felhasználva. Végül kiértékeltük a tesztelés során kapott eredményeket és összehasonlítottuk a különböző programokat, kiemelve azok erősségeit és gyengéit. Majd levontuk a következtetéseket, hogy melyik technológia irányában érdemes tovább haladni és melyik az, amelyikkel nem éri meg foglalkozni.

Jövőbeli terveimet illetően szeretnék jobban elmerülni a GPU-n történő differenciálegyenletek megoldásának módszereiben, valamint ezek alkalmazását kipróbálni és tanulmányozni a parciális differenciálegyenletek területén (PDE). Továbbá érdemesnek tartom a C++ szoftver továbbfejlesztését és egy differenciálegyenlet megoldó könyvtár megalkotását, amely ingyenesen használható és nyílt forráskódú lenne.

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.

Ábrák jegyzéke

Táblázatok jegyzéke

Irodalomjegyzék

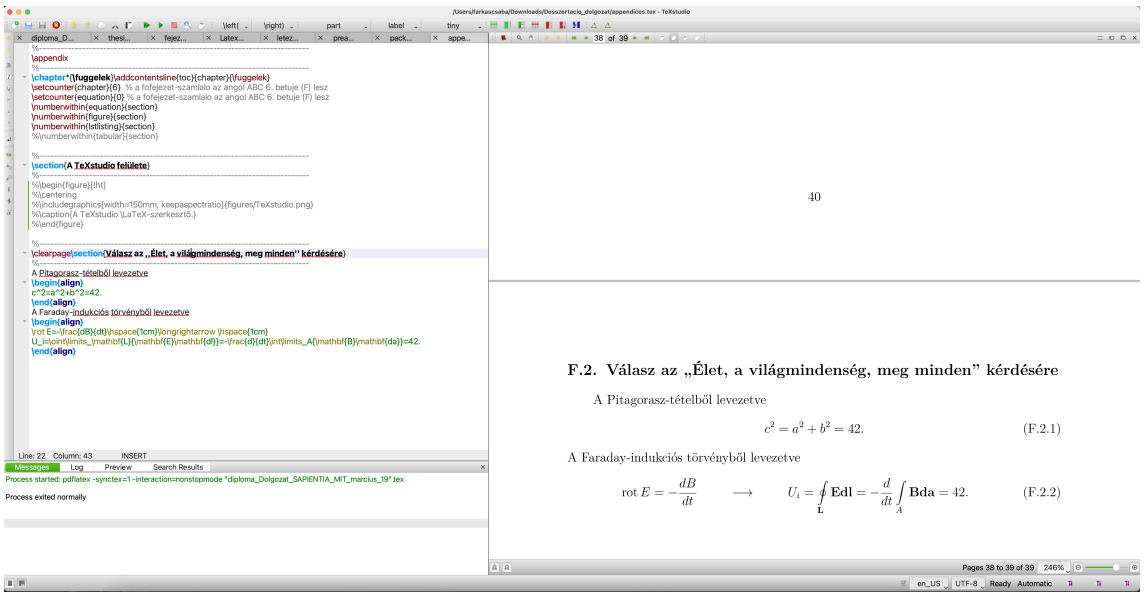
[Ant07] Margit Antal. Toward a simple phoneme based speech recognition system. *Stud. Univ. Babeş-Bolyai Inform.*, 52(2):33–48, 2007.

[KÓ8] Zoltán Kátai. Dynamic programming as optimal path problem in weighted digraphs. *Acta Math. Acad. Paedagog. Nyházi. (N.S.)*, 24(2):201–208, 2008.

[LS15] László Lovász and Balázs Szegedy. The automorphism group of a graphon. *J. Algebra*, 421:136–166, 2015.

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

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