Návrh

Analýza lingvistických dát

Pavol Freivolt

Tamara Savková

Júlia Gablíková

Michal Knor

FMFI UK

*TIS*

**Obsah**

[1 Úvod 2](#_Toc530243579)

[2 Rozdelenie na časti 2](#_Toc530243580)

[2.1 Popis modulov 2](#_Toc530243581)

[2.1.1 Frontend webovej aplikácie 2](#_Toc530243582)

[2.1.2 Backend webovej aplikácie 2](#_Toc530243583)

[2.1.3 Spúšťací a ukončovací modul 2](#_Toc530243584)

[2.1.4 Čítací modul 3](#_Toc530243585)

[2.1.5 Upravovací modul 4](#_Toc530243586)

[2.1.6 Slabikovací modul 6](#_Toc530243587)

[2.1.7 Počítací modul 7](#_Toc530243588)

[2.1.8 Výsledkový modul 8](#_Toc530243589)

[2.1.9 Zápis naslabikovaného textu 9](#_Toc530243590)

[2.1.10 Zápis tabuliek 10](#_Toc530243591)

[2.2 Súčasne bežiace procesy 11](#_Toc530243592)

[2.2.1 Moduly a vlákna 11](#_Toc530243593)

[2.2.2 Komunikácia medzi vláknami 11](#_Toc530243594)

[2.3 Dáta a formáty súborov 12](#_Toc530243595)

[2.3.1 Funkčné súbory aplikácie 12](#_Toc530243596)

[2.3.2 Konfiguračné súbory jazyka 12](#_Toc530243597)

[2.3.3 Analyzovaný text 13](#_Toc530243598)

[2.3.4 Výsledné súbory modulu Zápis naslabikovaného textu 13](#_Toc530243599)

[2.3.5 Výsledné súbory modulu Zápis tabuliek 14](#_Toc530243600)

[3 Používateľské rozhranie (obrázky) 14](#_Toc530243601)

[4 UML diagramy 16](#_Toc530243602)

[5 Využité technológie 17](#_Toc530243603)

# Úvod

Tento dokument je návrhom architektúry systému vyvíjaného na analýzu lingvistických dát. Je určený primárne pre vývojový tím a obsahuje všetky informácie potrebné na implementáciu softvéru. Názvy konštánt a atribútov v tomto návrhu nie sú záväzné, slúžia predovšetkým na zlepšenie predstavy o funkcionalite. Systém bude implementovaný v jazyku Python a niektoré časti návrhu sa môžu odkazovať na jeho funkcie a triedy.

# Rozdelenie na časti

## Popis modulov

### Frontend webovej aplikácie

Frontend modul tvorí vizuálnu časť programu a slúži na jednoduchšiu manipuláciu s programom.

Implementácia:

* modul je zložený z dvoch stránok:

1. stránka kde sa text nahrá pomocou súboru
2. stránka kde sa text zadá cez textové pole

* obe stránky obsahujú nadpis „Analýza lingvistických dát“, navigáciu, pomocou ktorej sa dá prechádzať medzi stránkami, „selekt“ kde sa dá vybrať jazyk, „Submit“ tlačidlo a pätičku stránky
* používateľ vkladá dáta pomocou súboru, ktorý nahrá alebo priamo vloží text do textového poľa
* modul obsahuje dva „selekty“ (prvý je prístupný len pri nahrávaní súboru, druhý vždy):
  1. „encoding“ , z ktorého sa dá vybrať kódovanie zo zoznamu podporovaných kódovaní
  2. „language“ , z ktorého sa dá vybrať jazyk zo zoznamu jazykov
* modul dáta posiela do „backendu“ pomocou formulára po odkliknutí tlačidla „Submit“.

### Backend webovej aplikácie

Backend modul pracuje s dátami ktoré sú mu zaslané z frontednu pomocou formuláru cez metódu $\_POST.

Implementácia:

* ak bol súbor nahratý cez formulár, tak ho modul uloží do priečinku temp\_files/*random\_name*
* ak bol vstup zadaný cez textové pole tak tento vstup zapíše do súboru, tomuto súboru dá meno podľa prvého slova textu a následne tento súbor uloží do priečinku temp\_files*/random\_name*
* modul spustí Spúšťací a ukončovací modul, ktorý vygeneruje výstupné súbory
* po ukončení procesu modul zabalí výstupné súbory (.zip), vynúti stiahnutie súborov a vymaže všetky dočasné súbory

### Spúšťací a ukončovací modul

Spúšťací a ukončovací modul je určený na inicializovanie objektov, spustenie modulov na vláknach a ukončenie modulov.

Argumenty:

* file\_name  - názov súboru s textom na spracovanie
* file\_path – cesta k vygenerovanému priečinku temp\_files/*random\_name*
* encoding - typ kódovania súboru
* language - jazyk v ktorom je súbor napísaný

**Postup**

Vytvoria sa objekty:

* data - objekt typu ConfigData, ktorému poskytne názov konfiguračného súboru zvoleného jazyka
* pipe\_read\_clean - objekt typu Pipe
* pipe\_clean\_syll - objekt typu Pipe
* pipe\_syll\_count - objekt typu Pipe
* pipe\_count\_txt - objekt typu Pipe

Inicializujú sa moduly:

* Upravovací modul s atribútmi: pole obsahujúce pipe\_read\_clean a pipe\_clean\_syll
* Slabikovací modul s atribútmi: pole obsahujúce pipe\_clean\_syll a pipe\_syll\_count
* Počítací modul s atribútmi: pole obsahujúce pipe\_syll\_count a pipe\_count\_txt, data, file\_path + file\_name
* modul Zápis naslabikovaného textu s atribútmi: pole obsahujúce pipe\_count\_txt, file\_path

Spustenie práce modulov a ich ukončenie:

* na každom module sa zavolá funkcia start().
* zavolá Čítací modul a atribútmi: pole obsahujúce pipe\_read\_clean, ďalej atribúty file\_path + file\_name, kódovanie vstupného textu, Data.
* na každom module sa zavolá funkcia join().

### Čítací modul

Čítací modul je určený na vytváranie zoznamu objektov tried TextPunctuation a End na základe vstupných argumentov.

Argumenty:

* Pipes – pole s jedným prvkom typu Pipe (pipe\_read\_clean);
* Adresa vstupného textu (file\_path);
* Kódovanie vstupného textu (encoding);
* Objekt triedy ConfigData s konfiguračnými dátami (data).

Modul je objektom triedy ReadModule, ktorá je podtriedou triedy Module.

Daný modul vytvára objekty triedy TextPunctuation (podtrieda triedy Word), ktoré:

* reprezentujú “slovo” – jednotlivú oddelenú prázdnym symbolov postupnosť symbolov zo vstupného textu,
* majú atribúty:
  + text “slova” – zoznam symbolov slova (\_text),
  + zoznam špeciálnych označení pre interpunkčné znamienka – zoznam označení priradených ku každému symbolu (\_punct).

Tiež modul vytvára objekt triedy End, ktorý označuje, že text, ktorý je v súbore s názvom vstupného textu, bol prečítaný a modul skončil svoju prácu.

Špeciálnymi označeniami sú cifry, ktoré označujú úlohu symbolu v slove:

* interpunkčne znamienka (konštanta PUNCT - 5),
* prenos (konštanta HYPHEN - 6).

Funkcie modulu:

* Modul rozdeľuje vstupný text na “slova” - jednotlivé oddelené prázdnym symbolov postupnosti symbolov zo vstupného textu.
* Tiež modul nastavuje špeciálne označenia pre zodpovedajúci objekt triedy TextPunctuation každého “slova”:
  + Interpunkčne znamienka - označenie ostatných pomlčiek a znakov *. , : ; ? ! [ ] ( ) { } ⟨ ⟩ — ― ‹ › « » “ ” " "* ;
  + Prenos  - označenie pomlčky, použitej pre rozdelenie “slova” na riadky, aby sa dalo ich spracovať.
* Po jednom pridáva objekty triedy TextPunctuation alebo End do dátovej štruktúry queue. Ak “slovo” je posledné v texte, modul pridá do dátovej štruktúry queue objekt triedy End, inak objekt triedy TextPunctuation.

Implementácia:

* Modul vytvorí pomocné premenné:
  + zoznam pre objekty triedy TextPunctuation objekty, vytvárané počas spracovania,
  + pracovný buffer pre text “slova”,
  + pracovný buffer pre špeciálne označenia “slova”,
  + boolean premenná, ktorá označuje či načítané “slovo” je prenesene na nový riadok cez pomlčku.
* Ďalej modul otvorí vstupný súbor na základe jeho mena a kódovania a číta ho celý po jednom symbole.
* Ak načítaný symbol nie je medzera, novy riadok alebo tabulátorová zarážka, potom pomocná boolean premenná sa nastaví na False a každý symbol sa snaží pridať do pracovných bufferov:
  + Ak symbolom je pomlčka, potom sa ošetria prípady prenášania textu na novy riadok a napísania slov cez pomlčku. Symbol pomlčky a špeciálne označenie prenosu sa pridajú do pracovných bufferov. Pomocná boolean premenná sa nastaví na True. Pokus o priradenie symbolu do pracovných bufferov skončí.
  + Ak symbol patrí do ostatných interpunkčných znamienok, tak symbol a špeciálne označenie interpunkčných znamienok sa pridajú do pomocných bufferov. Pokus o priradenie symbolu do pracovných bufferov skončí.
  + Ak proces pokračuje, symbol a špeciálne označenie sa pridajú do pomocných bufferov. Pokus o priradenie symbolu do pracovných bufferov skončí.
* Ak ďalší načítaný symbol je medzera, novy riadok alebo tabulátorová zarážka, začne sa spracovanie údajov v pracovných bufferoch:
  + Ak pracovný buffer nie je prázdny, potom:
    - Ak pomocná boolean premenná je nastavená na True a ďalší načítaný symbol nie je medzera, potom v pracovnom bufferi pre špeciálne označenia nastavenie poslednej hodnoty sa nastaví na označenie pre prenos.
  + Ďalej sa vytvorí objekt triedy TextPunctuation na základe obsahu pracovných bufferov a pridá sa do zoznamu pre získané objekty triedy TextPunctuation.
* Keď sa načíta cely text:
  + Ak v pracovných bufferoch ešte sú nejaké dáta, vytvorí sa objekt triedy TextPunctuation na základe obsahu pracovných bufferov a pridá sa do zoznamu pre získané objekty triedy TextPunctuation.
  + Modul pošle objekt triedy End do dátovej štruktúry queue.

### Upravovací modul

Upravovací modul je určený na prípravu k slabikovaniu slova, reprezentovaného objektom triedy TextPhonotypes, na základe získaných z queue pre vstup (pipes[0]) konfiguračných dát.

Argumenty:

1. pipes - pole s dvoma prvkami typu Pype (pipe\_read\_clean, pipe\_clean\_syll)
2. data - objekt s konfiguračnými dátami triedy ConfigData

Modul je objektom triedy CleanModule, ktorá je podtriedou triedy ThreadModule.

* modul dedí metódu run(), ktorú je nutné implementovať
* modul dedí metódu \_\_init\_\_(self, pipes), ktorá okrem iného vytvára thread s targetom run()
* modul dedí metódy start() a join(), ktoré slúžia na prácu s threadom modulu

Funkcie modulu:

* Získava z queue pre vstup objekty triedy TextPunctuation a postupne vytvára z nich trojice (predchádzajúci, aktuálny a nasledujúci objekt), spracováva trojicu, a aktualizuje trojicu po získaní ďalšieho slova (“predchádzajúci” dostáva hodnotu “aktuálneho”, “aktuálny” - “nasledujúceho”, a “nasledujúci” - nového slova”).
* Ak podľa objektu s konfiguračnými dátami slová, ktoré neobsahujú slabiky, sa spájajú s predchádzajúcim alebo nasledujúcim slovom, modul vykoná tieto operácie.
* Modul bude ignorovať slová, ktoré v strede obsahujú znaky, ktoré nepatria do jazyka.
* Modul bude ignorovať interpunkčné znaky, ktoré sú na začiatku alebo konci slova.
* Modul nastaví špeciálne označenia pre písmená textu slova, aby zodpovedali fonetickému typu:
  + 0 pre konsonantné písmena (konštanta CONS),
  + 1 pre sonórne písmena (konštanta SONOR),
  + 2 pre samohlásky (konštanta VOWEL),
  + 3 pre špeciálne písmená, ktoré sa spájajú s predchádzajúcou slabikou (konštanta SPEC\_PREC),
  + 4 pre špeciálne písmená, ktoré sa spájajú s nasledujúcou slabikou(konštanta SPEC\_FOLL).
* Výsledný objekt triedy TextPhonotypes bude obsahovať v texte slova len povolene písmena (podľa objektu s konfiguračnými dátami) a len špeciálne označenia pre fonetický typ písmena.
* Ak podľa objektu s konfiguračnými dátami niektoré písmená zmenia svoj fonetickej typ, potom modul vykoná tieto zmeny.

Implementácia:

* Modul získava z queue pre vstup objekty triedy TextPunctuation a postupne vytvára z nich trojice (predchádzajúci, aktuálny a nasledujúci objekt).
* Spracováva trojicu, ak jej druhý element je objektom triedy TextPunctuation:
  + Nastaví pomocné premenne:
    - Pracovný buffer pre text slova,
    - Pracovný buffer pre špeciálne označenia slova,
    - Pomocná mapa, ktorá je určená na vykonanie zmeň fonetického typu písmen slova. Ukladá informácie o tom, či také zmeny treba bude robiť na konci spracovania objektu slova a pole s indexmi, ktoré označujú pre aké písmeno treba bude robiť zmeny.
  + Prechádza každým symbolom a špeciálnym označením objektu slova.
    - Ak buffer pre text slova nie je prázdny, špeciálne označenie  je ‘interpunkčné znamienko’ a zároveň tento symbol je *[ ( { ⟨ ‹ « ‘ “* alebo *'"*, potom tento symbol sa ignoruje a nepridáva do pracovných bufferov. Načíta ďalší symbol a špeciálne označenie.
    - Ak špeciálne označenie je ‘prenos’ a ďalší symbol je písmeno, potom tento symbol sa ignoruje a nepridáva do pracovných bufferov. Načíta ďalší symbol a špeciálne označenie.
    - Ak špeciálne označenie  je ‘interpunkčné znamienko’ a symbol nie je žiaden z - —, potom sa skontroluje či ďalšie symboly až po koniec slova, ci ďalšie špeciálne označenia sú tiež ‘interpunkčné znamienka’. Ak nie, vráti trojicu: “predchádzajúci” objekt, null objekt, “nasledujúci” objekt. Ak áno, vráti trojicu: “predchádzajúci” objekt, upravený objekt slova (text a označenia slova sú zmenené podľa obsahu pracovných bufferov), “nasledujúci” objekt
    - Ďalej symbol sa zmení na malý (lowercase)
    - Prebehne kontrola, ci symbol mení svoj foneticky typ (podľa objektu s konfiguračnými dátami). Ak áno, pomocná mapa uloží index tohto slova a informáciu o tom, že tieto zmeny treba bude vykonať.
    - Prebehne kontrola, ci symbol patrí do malých písmen jazyka (podľa objektu s konfiguračnými dátami). Ak áno, malý symbol sa pridá do pracovných bufferov. Tiež sa zisti fonetická reprezentácia písmena (podľa objektu s konfiguračnými dátami), a priradí sa do pracovných bufferov pre zvuky. Ak nie, vráti trojicu: “predchádzajúci” objekt, null objekt, “nasledujúci” objekt.
  + Na záver, ak dĺžka textu “aktívneho” objektu slova je 1 a tento text netvorí slabiky (podľa objektu s konfiguračnými dátami), potom:
    - Ak objekt sa pridáva k predchádzajúcemu objektu (podľa objektu s konfiguračnými dátami), potom text a špeciálne označenia “aktuálneho” objektu sa pridávajú na koniec textu a špeciálnych označení “predchádzajúceho” objektu.
    - Ak objekt sa pridáva k nasledujúcemu objektu (podľa objektu s konfiguračnými dátami), potom text a špeciálne označenia “aktuálneho” objektu sa pridávajú na začiatok textu a špeciálnych označení “nasledujúceho” objektu.
    - Vráti trojicu objektov: “predchádzajúci” objekt, null objekt, “nasledujúci” objekt.
  + Vráti trojicu objektov: “predchádzajúci” objekt, upravený objekt (text a označenia slova sú zmenené podľa obsahu pracovných bufferov), “nasledujúci” objekt triedy TextPhonotypes.
* Aktualizuje trojicu objektov po získaní ďalšieho slova (“predchádzajúci” dostáva hodnotu “aktuálneho”, “aktuálny” - “nasledujúceho”, a “nasledujúci” - nového slova”) a začína spracovávať ďalšiu trojicu.
* Ak z queue pre vstup bol získaný objekt triedy End, potom:
  + Trojica objektov sa aktualizuje posledný krát,
  + Vykoná sa  posledné spracovanie trojice objektov triedy TextPhonotypes,
  + Spracovaný objekt triedy TextPhonotypes sa pridá do queue pre výstup (pipes[1]).
  + Objekt End sa pridá do queue pre výstup.
  + Práca modulu skončí.

### Slabikovací modul

Slabikovací modul je určený na rozdelenie slov na slabiky.

Argumenty:

1. pipes – pole dĺžky 2 obsahujúce objekty typu Pipe v nasledujúcom poradí [pipe\_clean\_syll, pipe\_syll\_count]

Modul vyberie z pipe\_in (= pipe[0]) objekt typu TextPhonotypes. Slovo uložené v obdržanom objekte pod atribútom \_text bude rozdelené na slabiky vzhľadom na atribút \_phonotypes, ktorý definuje sonoritu jednotlivých písmen. Následne bude vytvorené slovo SyllablesPhonotypes, ktoré bude vložené do pipe\_out (= pipe[1]).

**Postup**

Dedenie:

* modul dedí metódu run(), ktorú je nutné implementovať
* modul dedí metódu \_\_init\_\_(self, pipes), ktorá okrem iného vytvára thread s targetom run()
* modul dedí metódy start() a join(), ktoré slúžia na prácu s threadom modulu

Metóda run() – inicializácia:

* modul vytvorí dve premenné pipe\_in, ktorá odkazuje na pipes[0] a pipe\_out, ktorá odkazuje na pipes[1]
* modul si do premennej running vloží hodnotu True
* následne modul spustí cyklus podmienený hodnotou premennej running (while running), v ktorom beží zvyšok programu

Získanie objektu TextPhonotypes z pipe\_in :

* modul sa najprv pokúsi získať pipe\_in (pipe\_in.acquire())
* po získaní pipe\_in sa modul pokúsi vybrať z pipe\_in prvok. Ak sa mu to nepodarí, čaká (pipe\_in.wait()). Ak sa mu to podarí, uvoľní pipe\_in a pokračuje

Rozdelenie slova TextPhonotypes.\_text na slabiky (1. iterácia):

* modul si vytvorí premennú A, ktorá bude poľom polí integerov
* modul si vytvorí premennú B – pole integerov
* postupne prechádza pole TextPhonotypes.\_phonotypes a vkladá tieto prvky do B
* ak modul narazil pri prechádzaní na konštantu s hodnotou 2, vloží ju do B, celé pole B vloží do A, pole B vyprázdni a pokračuje
* po prečítaní celého TextPhonotypes.\_phonotypes pokračuje 2. iteráciou

Príklad: TextPhonotypes.\_phonotypes = [2,1,0,2,1], z toho A = [[2],[1,0,2],[1]]

Rozdelenie slova TextPhonotypes.\_text na slabiky (2. iterácia):

* modul prechádza prvky poľa A počnúc A[1] (budeme tieto prvky (polia integerov) označovať písmenom A[i])
* ak nastane situácia, že nejaký prvok v A[i] má vyššiu hodnotu ako prvok, ktorý za ním nasleduje, tento prvok a všetky pred ním sú vybraté z A[i] a vložené na koniec poľa A[i-1]
* ak má prvok A[i][j] hodnotu konštanty SPEC\_PREC alebo SPEC\_FOLL, teda je to špeciálny znak, neporovnáva sa, namiesto toho sa vykoná operácia uvedená nižšie a prvok A[i][j-1] sa porovnáva s prvkom A[i][j+1] (teda A[i][j] sa v porovnávaní preskočil)
* ak prechádzame posledný prvok poľa A a tento prvok A[i] obsahuje len prvky s hodnotou 0, A[i] zrušíme a všetky jeho prvky presunieme do A[i-1]

Príklad: A = [[2],[1,0,2],[1]], z toho A = [[2,1], [0,2,1]]

Práca so špeciálnymi znakmi SPEC\_PREC a SPEC\_FOLL:

* ak má znak hodnotu SPEC\_PREC, skontroluje sa, či sa pred ním v slabike niečo nachádza, ak nie, tento znak sa vyberie zo slabiky a pripojí sa na koniec predchádzajúcej slabiky (vloží sa na koniec predchádzajúce poľa)
* ak má znak hodnotu SPEC\_FOLL, netreba robiť nič, pretože týmto znakom určite nekončí slabika (SPEC\_SUC určite nemá hodnotu 2)

Rozdelenie slova TextPhonotypes.\_text na slabiky (3. iterácia):

* modul vytvorí výsledné pole D, ktoré bude obsahovať stringy (slabiky)
* podľa dĺžok prvkov poľa A rozkúskuje slovo vo TextPhonotypes.\_text a tieto substringy vloží do poľa D

Príklad: TextPhonotypes.\_text = “intel”, A = [[2,1],[0,2,1]], teda A[0].length = 2, A[1].length = 3, z toho D = [“in”, “tel”]

Vloženie slova SyllablesPhonotypes do pipe\_out:

* modul sa najprv pokúsi získať pipe\_out (pipe\_out.acquire())
* po získaní pipe\_out modul vloží do pipe\_out objekt typu SyllablesPhonotypes s argumentami (D, A)
* modul urobí pipe\_out.notify()
* modul uvoľní pipe\_out

Ukončenie behu modulu:

* ak modul vo fáze získania objektu z pipe\_in zistí, že získaný objekt je typu End, nastaví premennú running na False, a tento objekt vloží do pipe\_out podobne ako vkladá SyllablesPhonotypes

### Počítací modul

Počítací modul je určený na zistenie dĺžok slabík a počítanie ich výskytov.

Argumenty:

1. pipes – pole dĺžky 2 obsahujúce objekty typu Pipe v nasledujúcom poradí [pipe\_syll\_count, pipe\_count\_txt]
2. data – objekt typu ConfigData
3. file\_path – cesta k vstupnému súboru (string)

Modul vyberie z pipe\_in [= pipes[0]] pole a každú slabiku samostatne spracuje – zistí a zaznamená jej dĺžku a zaráta jej výskyt.

**Postup**

Dedenie:

* modul dedí metódu run(), ktorú je nutné implementovať
* modul dedí metódu \_\_init\_\_(self, pipes), ktorá okrem iného vytvára thread s targetom run()
* modul dedí metódy start() a join(), ktoré slúžia na prácu s threadom modulu

Metóda run() – inicializácia:

* modul vytvorí dve premenné pipe\_in, ktorá odkazuje na pipes[0] a pipe\_out, ktorá odkazuje na pipes[1]
* modul si vytvorí premenné *map\_len* (obsahuje prvky typu string: integer, kde string je slabika a integer je jej dĺžka) a *map\_freq* (obsahuje prvky typu string: integer, kde string je slabika a integer jej počet výskytov)
* modul si do premennej *running* vloží hodnotu True
* následne modul spustí cyklus podmienený hodnotou premennej *running* (while *running*), v ktorom beží zvyšok programu (okrem zavolania nasledujúceho modulu)

Získanie objektu SyllablesPhonotypes z pipe\_in:

* modul sa najprv pokúsi získať pipe\_in (pipe\_in.acquire())
* po získaní pipe\_in sa modul pokúsi vybrať z pipe\_in prvok *p* typu SyllablesPhonotypes. Ak sa mu to nepodarí, čaká (pipe\_in.wait()). Ak sa mu to podarí, uvoľní pipe\_in a pokračuje.

Spracovanie slabík:

* modul prechádza slabiky poľa *p.\_syllables*
* modul si vytvorí pomocné pole A
* zistí, či sa už slabika nachádza v map\_freq, ak áno, zvýši hodnotu pod týmto kľúčom o 1, ak nie, pridá ju do map\_freq s hodnotou 1
* ak sa slabika nenachádzala v map\_freq, nenachádza sa ani v map\_len a teda modul musí zistiť jej dĺžku podľa postupu nižšie
* modul pridá do poľa A dĺžku spracovávanej slabiky, ktorú zistí z map\_len
* po prečítaní poslednej slabiky v *p.\_syllables* vloží modul novovytvorený prvok typu SyllablesLengths s argumentom (A, p.\_syllables) do pipe\_out podľa postupu nižšie

Zistenie a evidencia dĺžky slabiky:

* modul prechádza znaky (písmená) v slabike
* modul si vytvorí premennú *syl\_len*, ktorej počiatočná hodnota je 0
* modul zo štruktúry data zistí dĺžku daného znaku vzhľadom na okolité znaky a túto dĺžku pripočíta do *syl\_len*
* modul pridá do map\_len slabiku (kľúč) s jej dĺžkou v premennej *syl\_len* (hodnota)

Vloženie SyllablesLengths do pipe\_out:

* modul sa najprv pokúsi získať pipe\_out (pipe\_out.acquire())
* po získaní pipe\_out modul vloží do pipe\_out prvok typu SyllablesLengths s argumentom (A, p.\_syllables)
* modul urobí pipe\_out.notify()
* modul uvoľní pipe\_out

Ukončenie práce:

* ak modul vo fáze získania prvku *p* z pipe\_out zistí, že *p* je typu End, nastaví premennú running na False, a teda po spracovaní tohto slova skončí cyklus
* získaný prvok p typu End modul vloží do pipe\_out podobne ako SyllablesLengths
* po ukončení cyklu modul zavolá Výsledkový modul s argumentami map\_len, map\_freq, file\_path

### Výsledkový modul

Výsledkový modul vytvára dve nové mapy za pomoci map\_len a map\_freq.

Argumenty:

1. map\_len – obsahuje prvky typu string: integer, kde string je slabika a integer je jej dĺžka
2. map\_freq – obsahuje prvky typu string: integer, kde string je slabika a integer jej počet výskytov
3. file\_path – cesta k vstupnému súboru (string)

Modul vytvorí map\_with\_rep a map\_wout\_rep za pomoci map\_len a map\_freq, kde kľúčom v map\_with\_rep je dĺžka slabiky a hodnotou je počet unikátnych výskytov danej dĺžky slabiky, zatiaľ čo v map\_wout\_rep je kľúčom dĺžka slabiky a hodnotou počet všetkých výskytov danej dĺžky slabiky.

Postup

Dedenie:

* modul dedí metódu run(), ktorú je nutné implementovať
* modul dedí metódu \_\_init\_\_(self), ktorá okrem iného vytvára thread s targetom run()

Metóda run() – inicializácia:

* modul si do premennej run vloží hodnotu True
* následne modul spustí cyklus podmienený hodnotou premennej run (while run), v ktorom beží zvyšok programu

Vytvorenie a naplnenie premennej map\_with\_rep:

* modul vytvorí map\_with\_rep
* modul prejde všetky hodnoty (dĺžka slabiky) v map\_len
* ak hodnota ešte nie je kľúčom v map\_with\_rep, tak ju vytvorí kľúč (dĺžka slabiky) a hodnotu nastaví na 1, inak hodnotu zvýši o 1

Vytvorenie a naplnenie premennej map\_wout\_rep:

* modul vytvorí map\_wout\_rep
* modul prejde postupne všetky prvky v map\_freq
* modul za pomoci map\_len zistí dĺžku daného prvku v map\_freq
* ak sa kľúč s danou dĺžkou v map\_wout\_rep nenachádza, tak vytvorí kľúč s danou dĺžkou, ktorého hodnota je hodnota daného prvku z map\_freq
* ak sa kľúč s danou dĺžkou v map\_wout\_rep už nachádza, tak k nemu pripočíta hodnotu z map\_freq

Ukončenie práce

* ak modul zapíše posledný prvok do map\_wout\_rep, tak zavolá modul ‘Zápis tabuliek’ s argumentami: map\_len, map\_freq, map\_with\_rep, map\_wout\_rep, file\_path

### Zápis naslabikovaného textu

Modul ‘Zápis naslabikovaného textu‘ slúži na zapisovanie naslabikovaného textu do súboru.

Argumenty:

1. pipes – pole dĺžky 1 obsahujúce objekty typu Pipe [pipe\_count\_text]
2. file\_path – cesta k vstupnému súboru (string)

Modul vytvorí v priečinku file\_path súbory ‘syllable\_text.txt’ a ‘syllable\_lengths\_text.txt’. Následne vyberá z pipe\_in (= pipe[0]) object typu SyllablesPhonotypes. Slabiky v obdržanom objekte pod atribútom \_syllables sa uložia do poľa A a ich sonorita pod atribútom \_phonotypes do poľa B. Ak pole A dosiahne dĺžku ‘n’, tak otvorí súbor ‘syllable\_text.txt’ a uloží tam obsah pola A. Ak pole B dosiahne dĺžku ‘n’, tak otvorí súbor ‘syllable\_ lengths\_text.txt’, kam uloží obsah pola B.

**Postup**

Dedenie:

* modul dedí metódu run(), ktorú je nutné implementovať
* modul dedí metódu \_\_init\_\_(self, pipes, file\_path), ktorá okrem iného vytvára thread s targetom run()

Metóda run() – inicializácia:

* v priečinku file\_path sa vytvoria súbory ‘syllable\_text.txt’ a ‘syllable\_lenghts\_text.txt’
* modul si vytvorí pomocné pole A
* modul si vytvorí pomocné pole B
* modul si do premennej run vloží hodnotu True
* následne modul spustí cyklus podmienený hodnotou premennej run (while run), v ktorom beží zvyšok programu

Získanie objektu SyllablesPhonotypes z premennej pipe\_in:

* modul sa najprv pokúsi získať pipe\_in (pipe\_in.acquire())
* po získaní pipe\_in sa modul pokúsi vybrať z pipe\_in prvok; ak sa mu to nepodarí, čaká (pipe\_in.wait()). Ak sa mu to podarí, uvoľní pipe\_in a pokračuje

Zapísanie naslabikovaného textu do súboru:

* modul postupne prechádza pole SyllablesPhonotypes.\_syllables a vkladá tieto prvky do poľa A
* ak pole A presiahne dĺžku ‘n’, tak obsah pola A zapíše do súboru ‘syllable\_text.txt’, pričom slabiky jednotlivých slov sú oddelené pomlčkou a pole A vynuluje

Zapísanie naslabikovaného textu do súboru vo forme čísel:

* modul postupne prechádza pole SyllablesPhonotypes.\_phonotypes a vkladá tieto prvky do poľa B
* ak pole B presiahne dĺžku ‘n’, tak obsah pola B zapíše do súboru ‘syllable\_lengths\_text.txt’, kde slabiky sú nahradené ich dĺžkou a oddelené pomlčkou a pole B vynuluje

Ukončenie behu modulu:

* ak modul vo fáze získania objektu z pipe\_in zistí, že získaný objekt je typu End(), nastaví premennú run na False a teda po spracovaní tohto slova ukončí svoj beh

### Zápis tabuliek

Modul má na starosti vytvorenie troch tabuľkových súborov .xls.

Argumenty:

1. map\_len – obsahuje prvky typu string: integer, kde string je slabika a integer je jej dĺžka
2. map\_freq – obsahuje prvky typu string: integer, kde string je slabika a integer jej počet výskytov
3. map\_with\_rep – obsahuje prvky typu integer1: integer2, kde integer1 je dĺžka slabiky a integer2 je počet unikátnych slabík majúcich takúto dĺžku
4. map\_wout\_rep – obsahuje prvky typu integer1: integer2, kde integer1 je dĺžka slabiky a integer2 je počet všetkých slabík majúcich takúto dĺžku
5. file\_path – cesta k vstupnému súboru (string)

Modul vytvorí v priečinku file\_path súbory:

* ‘syllables\_multiplicity.xls‘ - tabuľka s početnosťou a dĺžkou slabík
* ‘number\_of\_length\_of\_syllables\_with\_repetition.xls‘ – frekvenčná tabuľka dĺžok slabík berúc do úvahy všetky výskyty slabík
* ‘number\_of\_length\_of\_syllables\_without\_repetition.xls’ - frekvenčná tabuľka dĺžok slabík berúc do úvahy výskyt slabík v texte iba raz

**Postup**

Dedenie:

* modul dedí metódu run(), ktorú je nutné implementovať
* modul dedí metódu \_\_init\_\_(self), ktorá okrem iného vytvára thread s targetom run()

Metóda run() – inicializácia:

* modul si do premennej run vloží hodnotu True
* následne modul spustí cyklus podmienený hodnotou premennej run (while run), v ktorom beží zvyšok programu

Vytvorenie tabuľky s početnosťou a dĺžkou slabík:

* modul vytvorí pomocné pole A, ktorého prvkami budú polia o veľkosti 3
* modul prejde všetky hodnoty (početnosť) v map\_freq zostupne
* zistí aký je kľúč (slabika) danej hodnoty
* na základe kľúča zistí jeho dĺžku (dĺžka slabiky) z map\_len
* zistené údaje zapíše do poľa A, kde prvkami poľa sú polia formátu [slabika, početnosť, dĺžka slabiky]
* vytvorí súbor ‘syllables\_multiplicity.xls‘ v priečinku file\_name
* zapíše do tabuľky prvky poľa A, ktorej stĺpce sú: slabika, početnosť, dĺžka slabiky

Vytvorenie tabuľky frekvenčnej tabuľky dĺžok slabík berúc do úvahy všetky výskyty slabík:

* modul vytvorí pomocné pole A, ktorého prvkami budú polia o veľkosti 3
* modul prejde všetky hodnoty (početnosť) v map\_wout\_rep a ich súčet zapíše do pomocnej premennej S
* prejde sa každý kľúč  (dĺžka slabiky) map\_wout\_rep
* pre každý kľúč sa zistí jeho početnosť
* vypočíta sa jeho percentuálny výskyt ako početnosť/S\*100
* zistené údaje zapíše do poľa A, kde prvkami poľa sú polia formátu [dĺžka slabiky, početnosť, percentuálny výskyt]
* vytvorí súbor ‘number\_of\_length\_of\_syllables\_with\_repetition.xls‘ v priečinku file\_name
* zapíše do tabuľky prvky poľa, ktorej stĺpce sú: dĺžka slabiky, početnosť, percentuálny výskyt

Vytvorenie tabuľky frekvenčnej tabuľky dĺžok slabík berúc do úvahy výskyt slabík v texte iba raz:

* modul vytvorí pomocné pole A, ktorého prvkami budú polia o veľkosti 3
* modul prejde všetky hodnoty (početnosť) v map\_with\_rep a ich súčet zapíše do pomocnej premennej S
* prejde sa každý kľúč  (dĺžka slabiky) map\_with\_rep
* pre každý kľúč sa zistí jeho početnosť
* vypočíta sa jeho percentuálny výskyt ako početnosť/S\*100
* zistené údaje zapíše do poľa A, kde prvkami poľa sú polia formátu [dĺžka slabiky, početnosť, percentuálny výskyt]
* vytvorí súbor ‘number\_of\_length\_of\_syllables\_without\_repetition.xls‘ v priečinku file\_name
* zapíše do tabuľky prvky poľa A, ktorej stĺpce sú: dĺžka slabiky, početnosť, percentuálny výskyt

## Súčasne bežiace procesy

Niektoré moduly bežia v samostatných vláknach, a teda aplikácia môže spracovávať viacero slov v rôznych úrovniach naraz. Zoznam vlákien a modulov, ktoré sú v týchto vláknach spúšťané, je uvedený v časti Moduly a vlákna. Znakom „->“ je vyjadrená následnosť, teda postupné spúšťanie modulov v týchto vláknach. Okrem Main threadu sú všetky vlákna robené podľa architektúry Pipes and filters a to v rámci seba, aj medzi sebou. Posúvanie dát medzi vláknami je opísané v časti Komunikácia medzi vláknami.

### Moduly a vlákna

* ***Main thread***: Spúšťací a ukončovací modul -> Čítací modul (-> Spúšťací a ukončovací modul)
* ***Thread1***: Upravovací modul
* ***Thread2***: Slabikovací modul
* ***Thread3***: Počítací modul -> Výsledkový modul -> Zápis tabuliek
* ***Thread4***: Zápis naslabikovaného textu

### Komunikácia medzi vláknami

* ***Spúšťací a ukončovací modul***: Od modulu Backend webovej aplikácie získava dáta, na základe ktorých vytvára všetky štruktúry potrebné na komunikáciu, spúšťa moduly a volá Čítací modul. Štruktúry a potrebné dáta dáva modulom k dispozícii pri ich spustení/zavolaní a viac s nimi nemanipuluje. Čaká, kým všetky vlákna skončia, aby mohol ukončiť aj svoj beh.
* ***Main thread – Thread1***: Čítací modul komunikuje s Upravovacím modulom pomocou pipe\_read\_clean.
* ***Thread1 – Thread2***: Upravovací modul komunikuje so Slabikovacím modulom pomocou pipe\_clean\_syll.
* ***Thread2 – Thread3***: Slabikovací modul komunikuje s Počítacím modulom pomocou pipe\_syll\_count.
* ***Thread3 – Thread4***: Počítací modul komunikuje s modulom Zápis naslabikovaného textu pomocou pipe\_count\_txt.

## Dáta a formáty súborov

### Funkčné súbory aplikácie

Aplikácia bude implementovaná v jazyku Python, a teda všetky jej fuknčné súčasti (okrem webovej) musia byť vo formáte **py**.

### Konfiguračné súbory jazyka

Súbory, v ktorých sú uložené pravidlá pre prácu s jazykom analyzovaného textu, budú vo formáte **json**.

**Štruktúra**:

{

"language": [

{

"name": "language\_name",

"writing\_system": "latin/cyrillic"

}

],

"uppercase": [

"uppercase\_letter1", "uppercase\_letter2", ...

],

"lowercase": [

{

"lowercase\_letter1": { "phoneme-length": length or -1, "sign": "SONOR/CONS/VOWEL"},

"lowercase\_letter2": { "phoneme-length": length or -1, "sign": "SONOR/CONS/VOWEL"},

...

}

],

"VOWEL": ["vowel1", "vowel2", ...],

"SONOR": ["sonor1", "sonor2", ...],

"CONS": ["cons1", "cons2", ...],

"special\_signs": ["sign1", "sign2", ...],

"digraphs": {

"letter1": ["letter1", "letter2", ...],

"letter2": ["letter1", "letter2", ...],

...

},

"changes": [

{

"letter1": {

"preceding": "SONOR/CONS/VOWEL",

"following": "SONOR/CONS/VOWEL",

"becomes": "SONOR/CONS/VOWEL"

},

"letter2": {

"preceding": "SONOR/CONS/VOWEL",

"following": "SONOR/CONS/VOWEL",

"becomes": "SONOR/CONS/VOWEL"

},

...

}

],

"zero-syllable\_words": [

{

"word1": "to\_following/to\_precedent",

"word2": "to\_following/to\_precedent",

...

}

],

"spec\_sound\_length": [

{

"letter1": {

"0": {

"preceding": "SONOR/CONS/VOWEL",

"following": "SONOR/CONS/VOWEL"

},

"1": {

"preceding": "SONOR/CONS/VOWEL",

"following": "SONOR/CONS/VOWEL"

},

"2": {

"preceding": "SONOR/CONS/VOWEL",

"following": "SONOR/CONS/VOWEL"

}

},

"letter2": {

"0": {

"preceding": "SONOR/CONS/VOWEL",

"following": "SONOR/CONS/VOWEL"

},

"1": {

"preceding": "SONOR/CONS/VOWEL",

"following": "SONOR/CONS/VOWEL"

},

"2": {

"preceding": "SONOR/CONS/VOWEL",

"following": "SONOR/CONS/VOWEL"

}

},

...

}

]

}

### Analyzovaný text

Vstupný text musí byť vo formáte **txt** a v jednom z kódovaní: **ASCII**, **Windows-1250**, **Windows-1251**, **UTF-8**, **UTF-16**, **UTF-32**. Jazyk súboru musí byť podporovaný.

### Výsledné súbory modulu Zápis naslabikovaného textu

Súbory budú vo formáte **txt** a v kódovaní **UTF-8**.

***syllable\_text.txt***: Bude obsahovať iba znaky abecedy uvedené v konfiguračnom súbore jazyka textu, medzery a pomlčky, pričom medzi pomlčkami a medzerami musí byť vždy minimálne jeden znak abecedy.

***syllable\_lengths\_text.txt***: Bude obsahovať iba číslice oddelené medzerami a pomlčkami.

### Výsledné súbory modulu Zápis tabuliek

Súbory budú vo formáte **xls**.

***syllables\_multiplicity.xls***: Tabuľka, v ktorej prvý stĺpec obsahuje text slabiky, druhý stĺpec obsahuje dĺžku slabiky a tretí stĺpec obsahuje počet výskytov slabiky v texte. Slabiky sú usporiadané podľa početnosti zostupne.

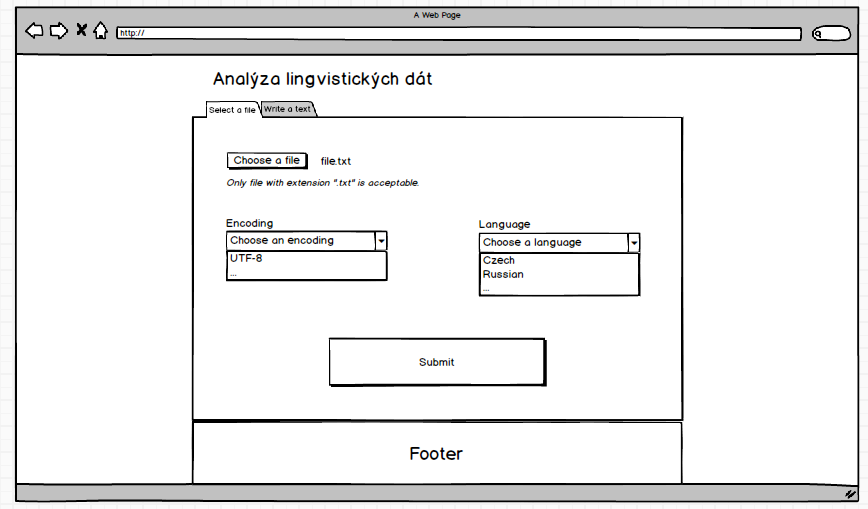
***count\_of\_length\_of\_syllables\_with\_repetition.xls****:* Tabuľka, v ktorej prvý stĺpec je dĺžka slabík a druhý stĺpec je počet slabík tejto dĺžky s opakovaním. Dĺžky sú usporiadané vzostupne.

***count\_of\_length\_of\_syllables\_without\_repetition.xls****:* Tabuľka, v ktorej prvý stĺpec je dĺžka slabík a druhý stĺpec je počet slabík tejto dĺžky s opakovaním. Dĺžky sú usporiadané vzostupne.

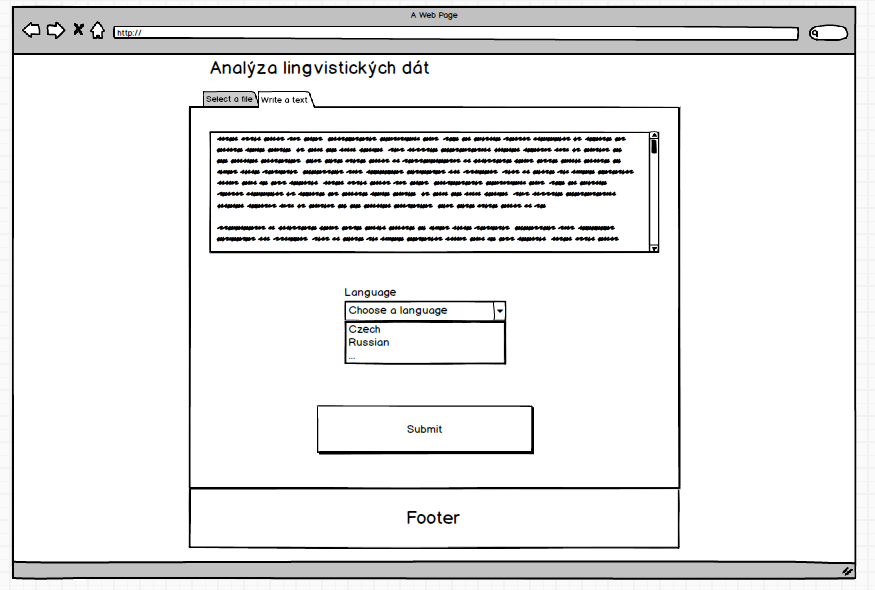
# Používateľské rozhranie (obrázky)



Obr. 1: Mobilné rozhranie

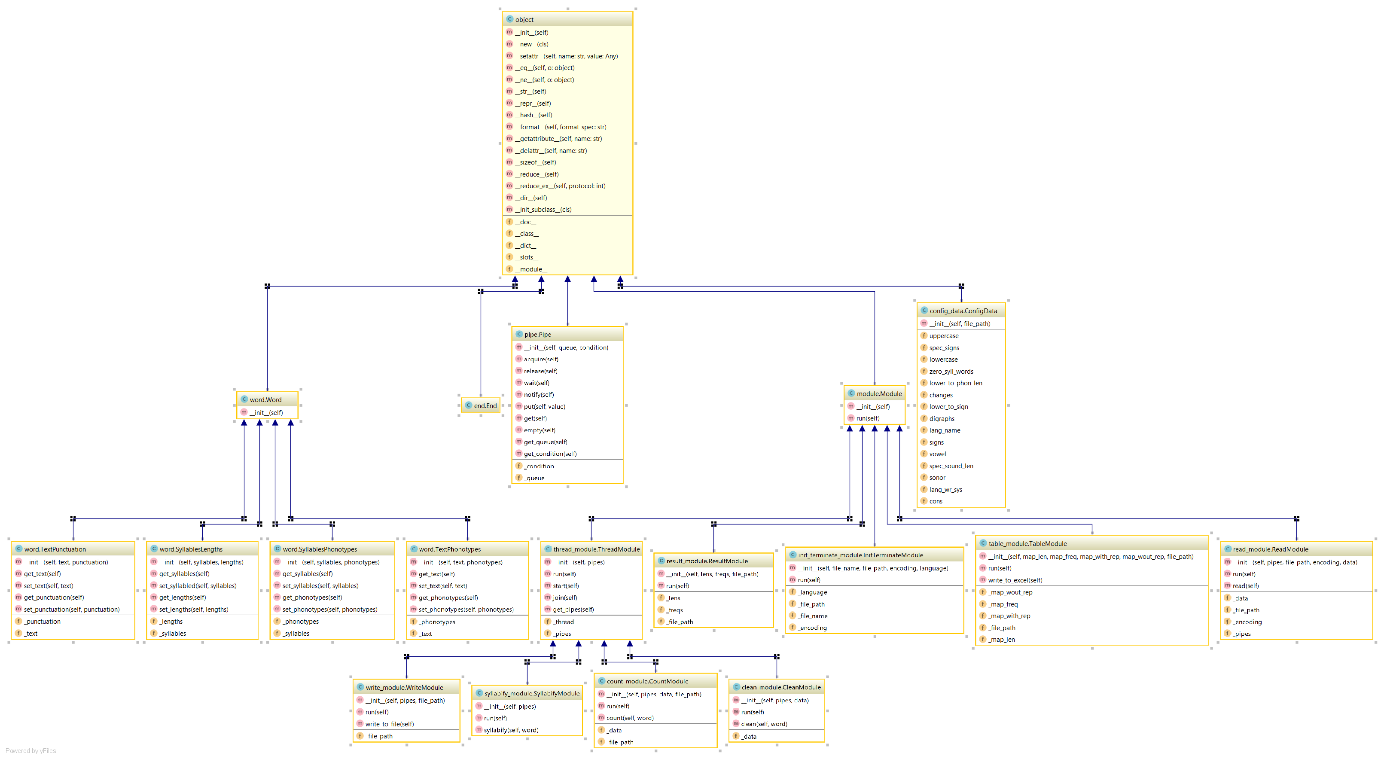


Obr. 2: Počítačové rozhranie - karta1

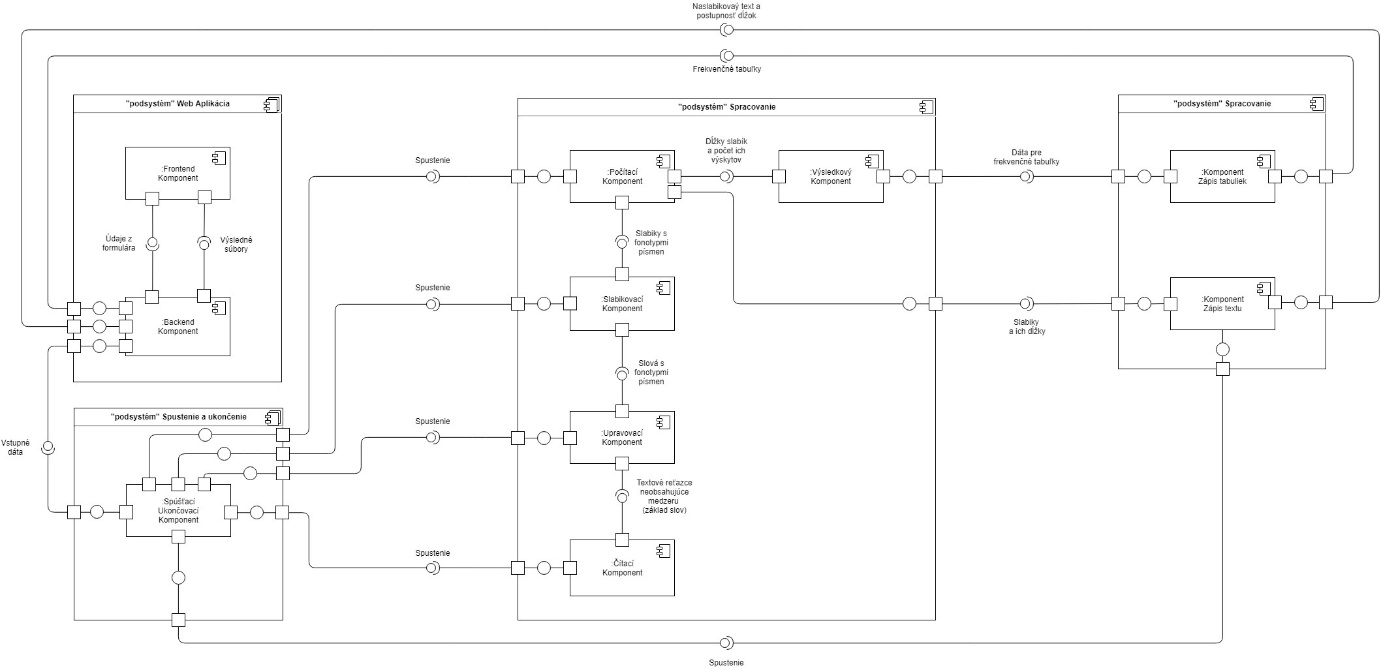


Obr. 3: Počítačové rozhranie - karta2

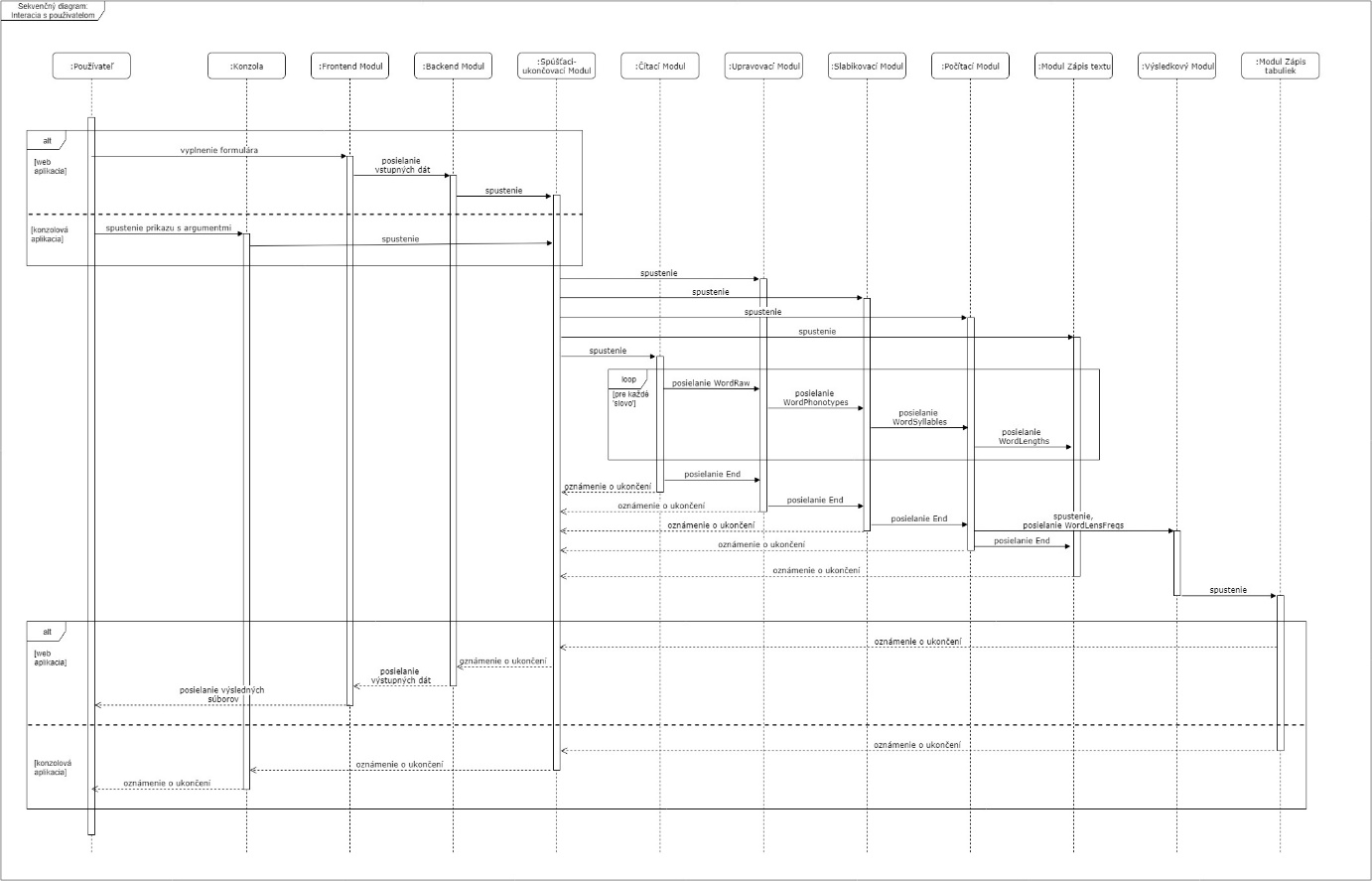
# UML diagramy



Obr. 4: Class diagram



Obr. 5: Component diagram



Obr. 6: Sequence diagram

# Využité technológie

* **CSS3** – umožňuje rozmiestnenie html objektov po stránke a úpravu celkového vizuálu stránky
* **HTML5** – umožňuje vytvoriť základnú kostru stránky
* **PHP** – umožňuje spracovať údaje z formuláru a spustiť py skript
* **JSON** – umožňuje, aby načítanie konfiguračných súborov vytvorilo objekt, s ktorým aplikácia dokáže pracovať
* **XLWT**

- knižnica slúži na vytváranie a zapisovanie súborov s príponou .xls

- zoznam hlavných príkazov:

* “wb = Workbook()” - vytvorí sa lokálna tabuľka
* “sheet1 = wb.add\_sheet(‘Sheet 1’)” – pomenuje sa list tabuľky
* “sheet1.write(row, column, ‘text’)” – zápis textu do bunky listu
* “sheet1.write(row, column, Formula(‘Sum(A1:A10)’)” – zápis vzorca do listu
* “wb.save(‘name.xls’, encoding=’UTF-8’)” – vytvorí/prepíše a uloží .xls súbor