

|  |  |
| --- | --- |
|  |  |
| Valutaváltó projektfeladat |  |
|  |  |
|  | *2025*  *2/14 EC* |
|  | Ágoston Attila  Kardos Zoltán  Tóth Sándorné |

Tartalomjegyzék

[Valutaváltó 1](#_Toc195107780)

[A feladat célja 4](#_Toc195107781)

[Beadandó vizsgaremek 4](#_Toc195107782)

[A honlap témája 6](#_Toc195107783)

[A történet, amiből kiindultunk: 6](#_Toc195107784)

[Kommunikáció és tárterület management 8](#_Toc195107785)

[Az adatbázisterv 9](#_Toc195107786)

[Az adatbázis implementálása a backend számára 9](#_Toc195107787)

[A backend rendszer 10](#_Toc195107788)

[A fejlesztés folyamata-backend 11](#_Toc195107789)

[Előkészítés 11](#_Toc195107790)

[Projektek 11](#_Toc195107791)

[Applikáció és az admin felület összekötése 11](#_Toc195107792)

[Adatbázis 12](#_Toc195107793)

[Adatbázis láthatósága 13](#_Toc195107794)

[Adatok adatbázisból történő megjelenítése, küldése JSON formátumban 13](#_Toc195107795)

[REST API 13](#_Toc195107796)

[Serializer 14](#_Toc195107797)

[Nézetek 14](#_Toc195107798)

[urls.py beállítások 15](#_Toc195107799)

[A fejlesztés folyamata – a frontend 16](#_Toc195107800)

[A környezet 16](#_Toc195107801)

[Adatok 16](#_Toc195107802)

[Reszponziv felületek 16](#_Toc195107803)

[A weboldalak 17](#_Toc195107804)

[Átváltás - valto.html 18](#_Toc195107805)

[Küldés – kuldo.html 18](#_Toc195107806)

[Diagramok – diagram.html 19](#_Toc195107807)

[Bejelentkezés – templates login.html 20](#_Toc195107808)

[Regisztráció – templates signup.html 20](#_Toc195107809)

[Deviza küldése – templates home.html 20](#_Toc195107810)

[Tranzakciók listája – templates tranzakciok.html 21](#_Toc195107811)

[Üzemeltetés 22](#_Toc195107812)

[Fejlesztői környezet kialakítása: 22](#_Toc195107813)

[Python, Django, Django RestFramework 22](#_Toc195107814)

[GitHub 22](#_Toc195107815)

[Visual Studio Code 22](#_Toc195107816)

[Docker 23](#_Toc195107817)

[Linuxweb 23](#_Toc195107818)

[Tesztelés 23](#_Toc195107819)

[Fejlesztési lehetőségek 23](#_Toc195107820)

[Irodalomjegyzék 24](#_Toc195107821)

[Mellékletek 24](#_Toc195107822)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A feladat célja A projektfeladatunk a Debreceni SZC Mechwart András Gépipari és Informatikai Technikum szoftvertesztelő és programozó felnőttképzési tanfolyam záró munkája.  A munka célja olyan reszponzív honlap készítése, mely képes az MNB által kezelt valuták átváltására az aktuális árfolyamon, valamint lehetőséget biztosit a regisztrált felhasználók számára valuta küldés megvalósítására. | | | |  | |
|  | Beadandó vizsgaremek A.) A vizsgázóknak minimum 2, maximum 3 fős fejlesztői csapatot alkotva kell a vizsgát megelőzően egy komplex szoftveralkalmazást lefejleszteniük.  B.) A szoftveralkalmazásnak az aláábbi elvárásoknak kell megfelelni:  1. Életszerű, valódi problémára nyújt megoldást.  2. Adattárolási és -kezelési funkciókat is megvalósít.  3. RESTful architektúrának megfelelő szerver és kliens oldali komponenseket egyaránt tartalmaz.  4. A kliens oldali komponens vagy komponensek egyaránt alkalmasak asztali és mobil eszközökön történő használatra. Mobil eszközre kifejlesztett kliens esetén natív mobil alkalmazás, vagy azzal hozzávetőlegesen megegyező felhasználói élményt nyújtó webes kliens egyaránt alkalmazható.  használ, míg az adminisztrációs felület natív asztali alkalmazásként készül el). |  | | |  | |
|  | Asztali eszközökre fejlesztett kliens oldali komponensnél mindenképpen szükséges webes megvalósítás is, de emellett opcionálisan natív, asztali alkalmazás is a csomag része lehet. (pl. A felhasználóknak szánt interfész webes megjelenítést használ, míg az adminisztrációs felület natív asztali alkalmazásként készül el).  5. A forráskódnak a tiszta kód elveinek megfelelően kell készülnie.  6. A szoftver célját, komponenseinek technikai leírását, működésének műszaki feltételeit és használatának rövid bemutatását tartalmazó dokumentáció is része a csomagnak.  A megosztott anyagnak tartalmaznia kell az alábbiakat:  • A szoftver forráskódja,  • Natív asztali alkalmazások esetén a program telepítőkészlete,  • Az adatbázis adatbázismodell-diagramja,  • Az adatbázis export fájlja (dump),  • A szoftveralkalmazás dokumentációja,  • A tesztekhez végzett kód, valamint a teszteredmények dokumentációja. | | |  | | | | |  |
|  | A honlap témája | | A feladat kiírás úgy szólt, hogy a valós életben létező problémának informatikai eszközökkel történő megoldását kell megvalósítani. A projektmunka első fázisában ezért megegyeztünk, hogy mi egy valutaváltással foglalkozó honlapot készítünk. Ahhoz, hogy a váltás napi árfolyamon történhessen, az MNB API-járól nyerjük ki az árfolyamokat. A váltási funkciót kiegészítettük egy mini felhasználókezelési rendszerrel, valamint a bejelentkezett felhasználók számára pénzküldési felületet is biztosítunk. A harmadik megvalósult egység az árfolyamdiagrammok, mely fixen tartalmaz egy pár fontosabb valutát, de a honlap használói a rendelkezésre álló egyéb valuták közül is választhatnak megjelenítendőt.  A honlap reszponzív, igy mind asztali, mind mobileszközön megjeleníthető. | | |  | |
|  | A történet, amiből kiindultunk: Debrecen belvárosában valutaváltással foglalkozó ismerős keresett meg, hogy a biznisz kissé visszaesett, ezért szeretne egy weblapot készíttetni velünk, hogy népszerűsítse vállalkozását.  **Kérése volt, hogy népszerűsítsük az új szolgáltatását a deviza tranzakciót és korrektül tájékoztassuk az ügyfeleit az árfolyamokról és a költségekről.** | | | | |  | |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| A munka folyamata |
|  |

A valutaváltás a fizikailag megfogható készpénzes tranzakciókra vonatkozik, ami csak és kizárólag személyesen vehető igényben az irodában. A deviza tranzakciókat számlára történő utalásokkal non materiális módon valósítja meg. A költségek mindkét esetben azonosak. A különböző pénznemek árfolyamai a mindenkori Magyar Nemzeti Bank legfrissebb hivatalos devizaárfolyamaival azonosak és magyar forintra történő átszámítás az elszámolás módja. Csak az MNB által kezelt devizanemekben történik a szolgáltatás.

A feladatot elvállaltuk a kis teamünkkel, akik **Tóth Sándorné Márta** - *projektvezető, data analist,* **Ágoston Attila** *– szofrverfejlesztő*, és Kardos Zoltán - *dizájner*.

Első lépésként Tóth Sándorné elkészítette a projekthez az induló adatbázist, feltöltötte az induláshoz szükséges adatokkal (alapértelmezett értékek) és ezt az adatbázist adta át a backendes munkálatokat végző Ágoston Attilának.

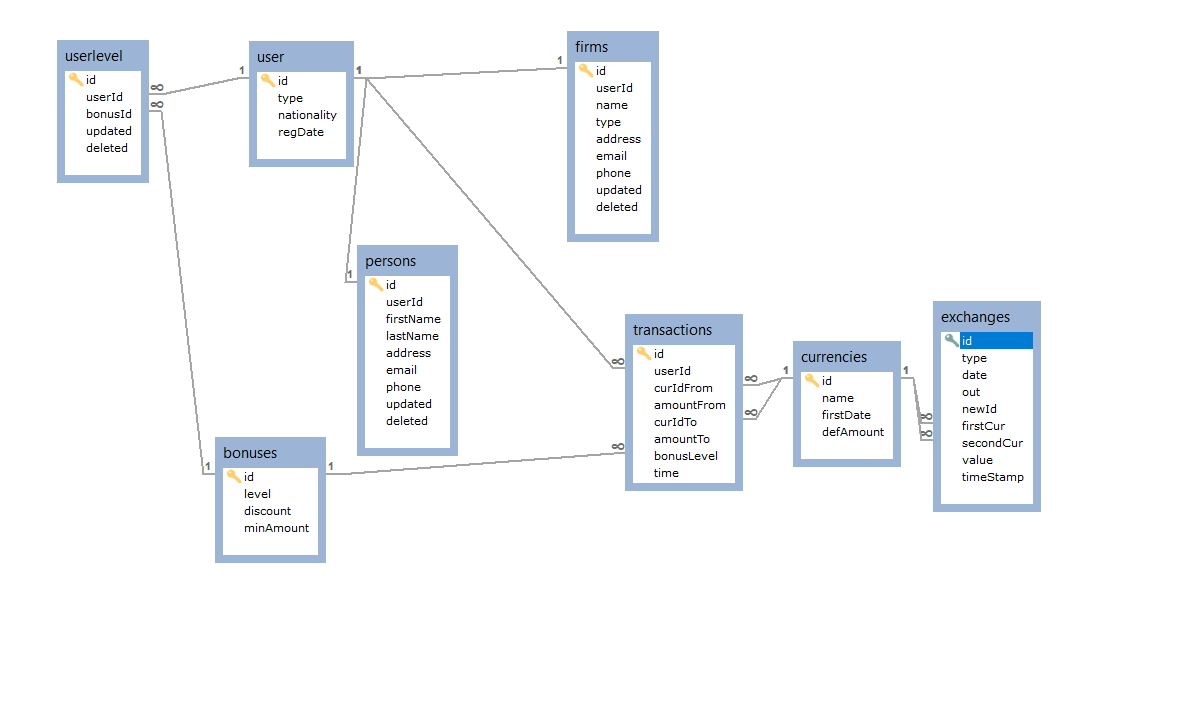
Attila létrehozta a backendes rendszert django segítségével. A backend rendszer részei a view-k, a serializer-ek, a felhasználókezeléshez szükséges részek, illetve néhány template, amit majd a frontend részben fognak használni. Itt történnek a számítási feladatok végzései is.

A frontend megvalósulása elsősorban Kardos Zoltán munkáját dicséri. Ő alkotta meg a honlapstruktúrát, dolgozta ki a tárhelyhasználatot, és a dizájn is az ő munkáját dicséri. A frontendhez használt alapvető módszerek a html, css, js és a bootstrap voltak.

## Kommunikáció és tárterület management

A munkafolyamat koordinálására a kedvenc eszközünk a közvetlen kapcsolat, valamint a facebook Messenger volt. A munka a GitHubon folyt, a repository-nk a [lukrecia602/project](https://github.com/lukrecia602/project) címen érhető el. A végső honlap a <http://fomix.hu/aladar/valto.html> címen található, a backend-es részek a [azenhazam.mywire.org:8800](http://azenhazam.mywire.org:8800/) szerveren futnak.

## Az adatbázisterv

Az első lépés, mint minden teljesen új informatikai fejlesztés esetében, az induló adatbázis tervezése és implementációja volt. Ezt a feladatot Tóth Sándorné végezte. Az induláskor megegyeztünk, hogy az adatokat az MNB által közzétett árfolyamokból vesszük. Igy az adatbázisstruktúrát alapvetően ezekhez igazítottuk. Mivel szükséges volt a projektben felhasználókezelést is megvalósítani, ezért ezeket az adatbáziselemek kiegészültek az ehhez szükséges elemekkel. A megvalósult adatbázis sémája az 1. ábrán található.

## Az adatbázis implementálása a backend számára

Az adatbázis elvi tervezése a MySQL rendszerben történt. Itt lettek kialakítva a táblák szerkezetei, a kapcsolatok, illetve a szükséges adat típusok. Az előre meghatározható (fix értékű) mezők adatai is itt kerültek rögzítésre. Ezután adaptáltuk a rendszert a django számára SQLite formátumba. Az adaptáció során a két rendszer felépítéséből és működéséből származó különbségek megvalósítása után a DB Browser alkalmazás konverziós funkciójával fordítottam át a rendszert SQLite formátumba. Az SQLite formájú induló adatbázis a mellékletekben megtalálható.

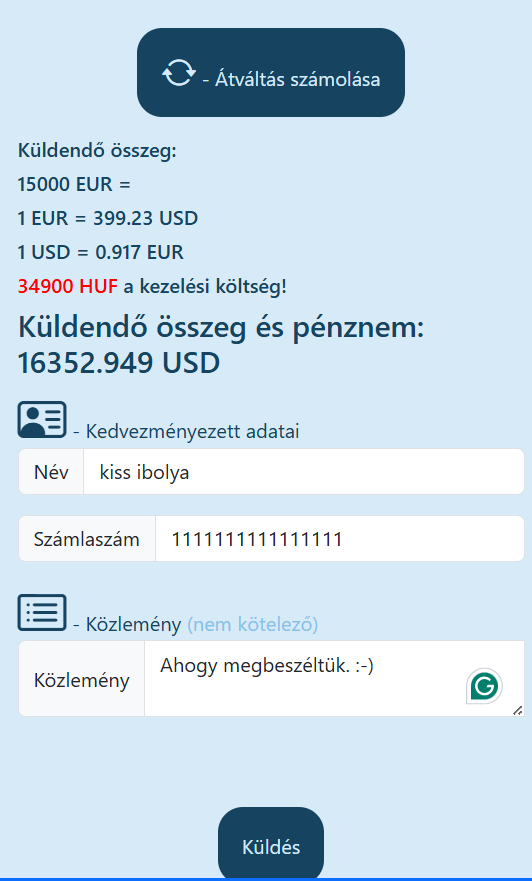
## A backend rendszer

Ágoston Attila készítette a backend egy részét django használatával. Négy template-re volt szükség, amelyek a frontendben külön html oldalt képviselnek. Ezek az oldalak alapvetően a regisztrált felhasználók által látogatott oldalak. Ezeken történik a felhasználók regisztrációja, bejelentkezése, és a valuta küldés. A felhasználók megtekinthetik a tranzakciós adataikat, törölhetik a fiókjukat. Ezekhez az oldalakhoz elkészítette a szükséges metódusokat, modelleket.

A regisztrációt, a be- és kijelentkezést, a django beépített felhasználókezelő rendszere végzi, de ennek használatához kellett készíteni egy űrlapot, amely lehetővé teszi a művelet elvégzését. Ehhez némelyik beépített metódust módosítani kellett.

Bejelentkezés után, a felhasználó tud valutát váltani majd a váltott valutát elküldeni egy címzettnek. A váltás és a küldés egy felületen történik, de külön metódusok kezelik a két műveletet.

A váltáshoz meg kell adni egy összeget, mit, mire szeretnénk váltani, a váltás gombra kattintva a program végrehajtja a váltást, és kiszámolja a küldés költségét is majd az adatok megjelennek a képernyőn.

Váltás után el tudjuk küldeni a váltott összeget. A küldéshez meg kell adni több adatot is, például a kedvezményezett nevét és számlaszámát. A küldés gombra kattintva kerülnek lementésre a tranzakciós adatok. Ezek az adatok megtekinthetők a tranzakciók gombra kattintva.Az adatok egy modell segítségével tárolódnak el az adatbázisban és egy külön metódus felel az adatok kiválogatásáért és megjelenítéséért.

## A fejlesztés folyamata-backend

### Előkészítés

VS Code-ban megnyitottam a mappát, amiben a fejlesztés fog történni.

Létrehoztam egy új parancssor típusú terminálablakot.

Ezután következett a virtuális környezet létrehozása: python -m venv .

és az aktiválása: cd Scripts\activate

Következő lépés volt a django rendszerének a telepítése: pip install django

Frissítettem a szükséges eszközöket:

python.exe -m pip install --upgrade pip

python.exe -m pip install --upgrade setuptools

Ezzel készen állt a környezet a projectek létrehozásához.

Első lépés a django adminisztrációs projektjének a létrehozása:

django-admin startproject config.

Applikáció létrehozása:

python manage.py startapp pénzváltó.

Az adatmigráció előkészítése és végrehajtása:

python manage.py makemigrations

python manage.py migrate

Admin (superuser) létrehozása:

python manage.py createsuperuser

A mi superuserünk az admin,admin adatpárral lett létrehozva, email cím nélkül.

Fejlesztői szerver elindítása:

python manage.py runserver

A szerver futását a ctrl+c billentyűkombináció szakitja meg.

Ha sikerült, akkor be lehet jelentkezni az oldalra a http://127.0.0.1:8000/admin címen.

Ha valami nem sikerült akkor a telepített projektek listázása segitett:

pip list

### Projektek

#### Applikáció és az admin felület összekötése

Az adminisztrációs projekt fájlaiban is kell beállításokat elvégezni.

- Settings.py fájlban az INSTALLED\_APPS nevű tömbben meg kell adni az app(mappa) nevét.

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'AppNeve',

]

Az urls.py fájlban össze kell kötni az Applikáció és a Adminisztráció fájljait:

from django.contrib import admin

from django.urls import path, include (includeot beirni)

urlpatterns = [

path('admin/', admin.site.urls),

path('',include('Pénzvéltó.urls'))

]

#### Adatbázis

Az appban a models.py fájlban kell létrehozni.

Különböző adattípusokkal létrehozott adatbázis:

from django.db import models

from datetime import datetime # Dátumkezelő osztály beimportálása

class Kulcs(models.Model): # Ebben a példában erre a modellre hivatkozunk idegen kulcsként.

kulcs = models.IntegerField()

def \_\_str\_\_(self):

return f'{self.kulcs}'

class Adattipusok(models.Model):

egesszam = models.IntegerField(null=False)

tortszam = models.FloatField(null=True)

datum = models.DateTimeField(default=datetime.now())

igazhamis = models.BooleanField()

rovidszoveg = models.CharField(max\_length=100)

hosszuszoveg = models.TextField()

idegenkulcs = models.ForeignKey('Kulcs', on\_delete=models.CASCADE) #idegen kulcs a másik táblára ('A másik tábla neve', CASCADE törlés)

def \_\_str\_\_(self):

return f'{self.egesszam} {self.tortszam} {self.datum} {self.igazhamis} {self.rovidszoveg} {self.hosszuszoveg} {self.idegenkulcs}'

# az admin felületen az adatok megjelenítésének beállítása: return f'{self.mezőnév} valami szöveg {self.mezőnév}'

# itt most minden adatot megjelenítünk szóközökkel elválasztva

# null = False - Azt jelenti, hogy adatfelvitelnél, az adott értéket kötelező megadni.

# null = True - Azt jelenti, hogy adatfelvitelnél az adatot nem kötelező megadni.

# default = datetime.now() - Alapértelmezett értékként a mai dátumot állítja be. Alapértelmezett érték bármilyen adattípusnál megadható.

# max\_length=100 - Maximális karakterhossz beállítása, CharField-nél kötelező megadni.

Ha kész az adatbázismodell, vagy egy már meglévő adatbázisnak változtatunk az adatszerkezetén, akkor mindig adjuk ki a

python manage.py makemigrations és python manage.py migrate parancsokat.

#### Adatbázis láthatósága

Ahhoz, hogy az adatbázis látható legyen az admin felületen, az alábbi beállítások szükségesek.

Ezeket a beállításokat az applikációbanban, az admin.py fájlban kell megcsinálni.

from django.contrib import admin

from . models import Adattipusok, Kulcs # A modellek beimportálása a modell nevének használatával.

admin.site.register(Adattipusok) # a láthatóság beállítása (Modellneve)

admin.site.register(Kulcs)

#### Adatok adatbázisból történő megjelenítése, küldése JSON formátumban

##### REST API

Ahhoz, hogy tudjunk adatokat kiküldeni JSON formátumban egy adatbázisból, több beállítás is szükséges.

Ezeket az adminisztrációs projektben a settings.py fájlban, valamint az applikációban a serializers.py, urls.py és a views.py fájlokban csináltam.

Első körben telepítettem a rest-framework-öt az alábbi paranccsal :

pip install djangorestframework.

Utána, az adminisztrációs projektben, az settings.py-ba került a következő:

INSTALLED\_APPS = [

'pénzváltóprojekt'

'rest\_framework', #rest-framework hozzáadása

]

##### Serializer

Ahhoz, hogy REST API segítégével tudjunk adatot kiküldeni JSON formátumban, szükséges egy serializer. Az appban létre kell hozni egy serializer.py fájlt és ezután ez kerül bele:

from rest\_framework import serializers

from . models import Adattipusok

class AdattipusokSerializer(serializers.ModelSerializer):

class Meta:

model=Adattipusok # Modell neve

fields = "\_\_all\_\_" # Az adatbázis összes mezőjének behúzása

depth = 1 # Mélység beállítása

##### Nézetek

A views.py fáljba kerültek azok a függvények, amik az kommunikációját és az adatok feldolgozását végzik.

Itt most a REST API-t mutatom, ami az adatok JSON formátumba történő küldéséért felel.

from django.shortcuts import render

from rest\_framework.response import Response

from rest\_framework.decorators import api\_view

from . models import Kulcs, Adattipusok # Modellek beimportálása.

from . serializers import AdattipusokSerializer # Serializer beimportálása.

# Create your views here.

@api\_view(['GET'])

def getAllAdattipusok(request):

all = Adattipusok.objects.all() # Az adatok, tömbbe történő eltárolása.

serialized = AdattipusokSerializer(all, many=True) # Serializáció

return Response(serialized.data) # Az adatok JSON formátumban történő megjelenítése.

##### urls.py beállítások

Az applikációban hozzuk létre az urls.py-t és ezt a fájlt használjuk az adminisztrációs felület címeinek a kiterjesztésére.

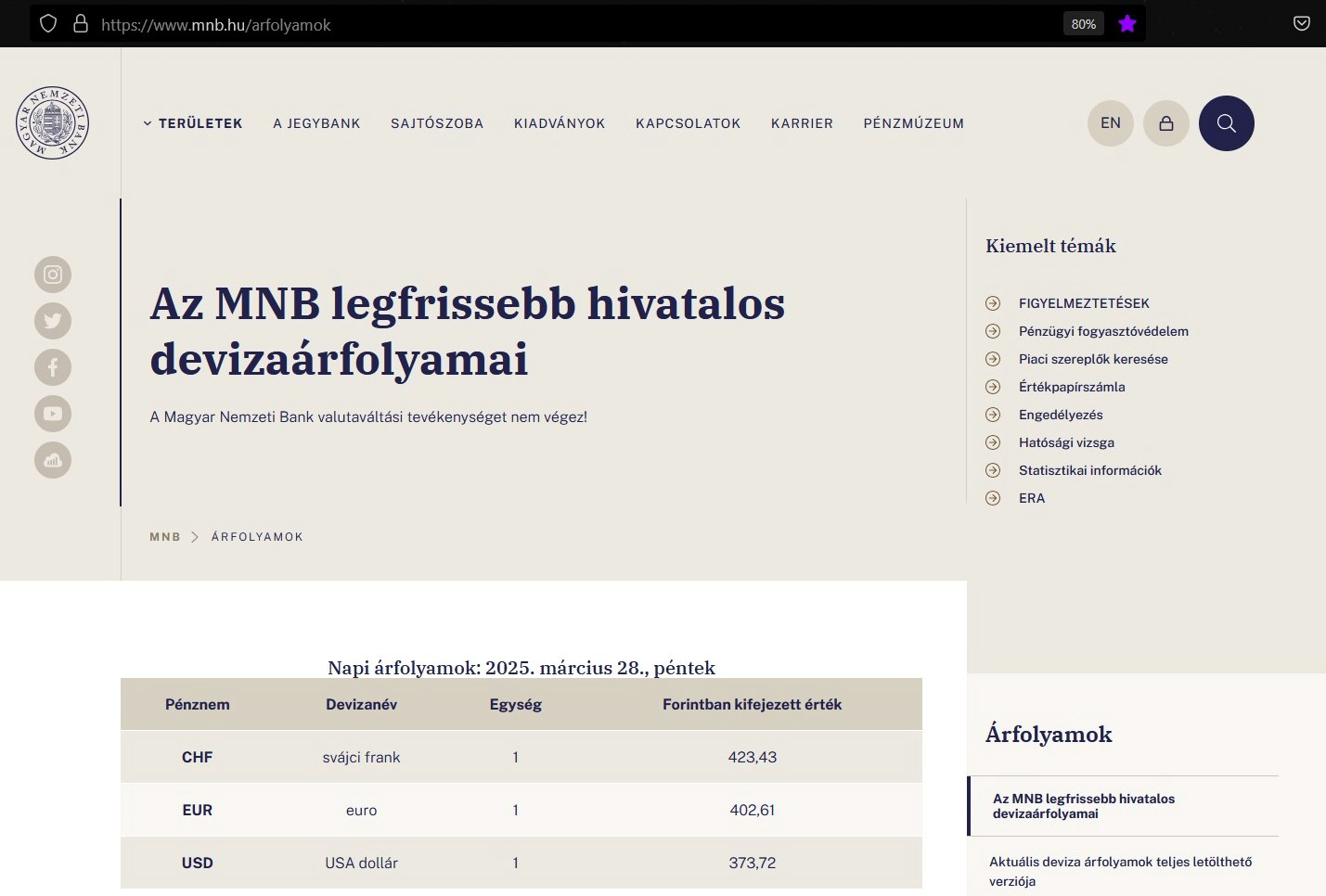
from django.urls import path

from . import views # A views.py beimportálása.

urlpatterns = [

path('adatok/', views.getAllAdattipusok, name='getAllAdattipusok'. ),

]

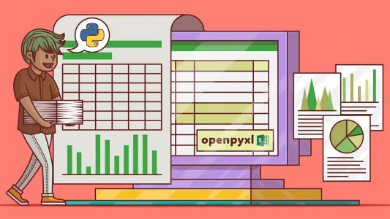
A szerver elindítása után az adatbázisban levő adatok megjelennek a képernyőn a http://127.0.0.1:8000/adatok/ címen.

## A fejlesztés folyamata – a frontend

**Kardos Zoltán** foglalkozott a felhasználói felületekkel (User Interface), és a felhasználói élményekkel, vagyis a frontend résszel, amivel a felhasználók közvetlenül találkoznak és interakcióba lépnek.

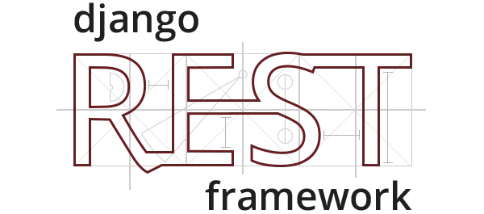
A Magyar Nemzeti Bank (MNB) a legfrissebb hivatalos devizaárfolyamait a weblapján <https://www.mnb.hu/arfolyamok> teszi közre.

### A környezet

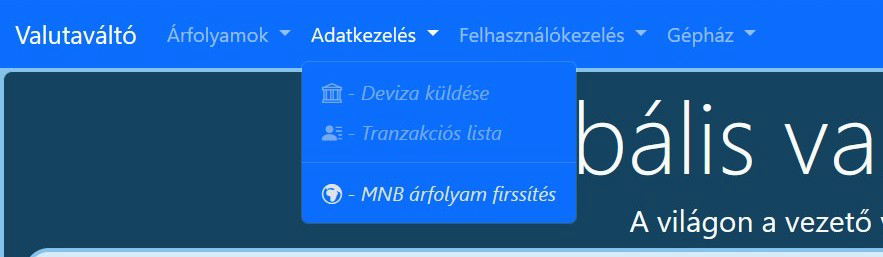
Az aktuális deviza árfolyam teljes letölthető verziója a   
<https://www.mnb.hu/Root/ExchangeRate/arfolyam.xlsx> számolótáblában érhető el. Az adatokat mnb\_deviza\_download.py állományban python függvényekkel

 lett átmigrálva a db.sqlite3 adpiatbázisunkba. A migráláshoz a Pandas nyílt forráskódú adatelemző és manipulációs python könyvtárat, és az excel munkafüzetek olvasásához az openxl könyvtárat használtuk.

#### Adatok

Miután az adatok bekerültek az adatbázisunkba keresni kellett egy lehetőséget, hogy az API adatok megfelelő módon a felhasználóhoz jussanak. A REST API-t (Django REST API) választottam, Json formátumban kommunikálja az adatokat a szerverünk a kliens eszközre. Így az adatokat serializáltam és kialakítottam a API végpontokat. Három endpoint került elkészítésre: egy az érvényes devizanemeknek, egy az utolsó legfrissebb árfolyamadatoknak az átváltás számításához és egy az árfolyamtörténet megjelenítéséhez, ami látványos grafikonnal valósult meg.

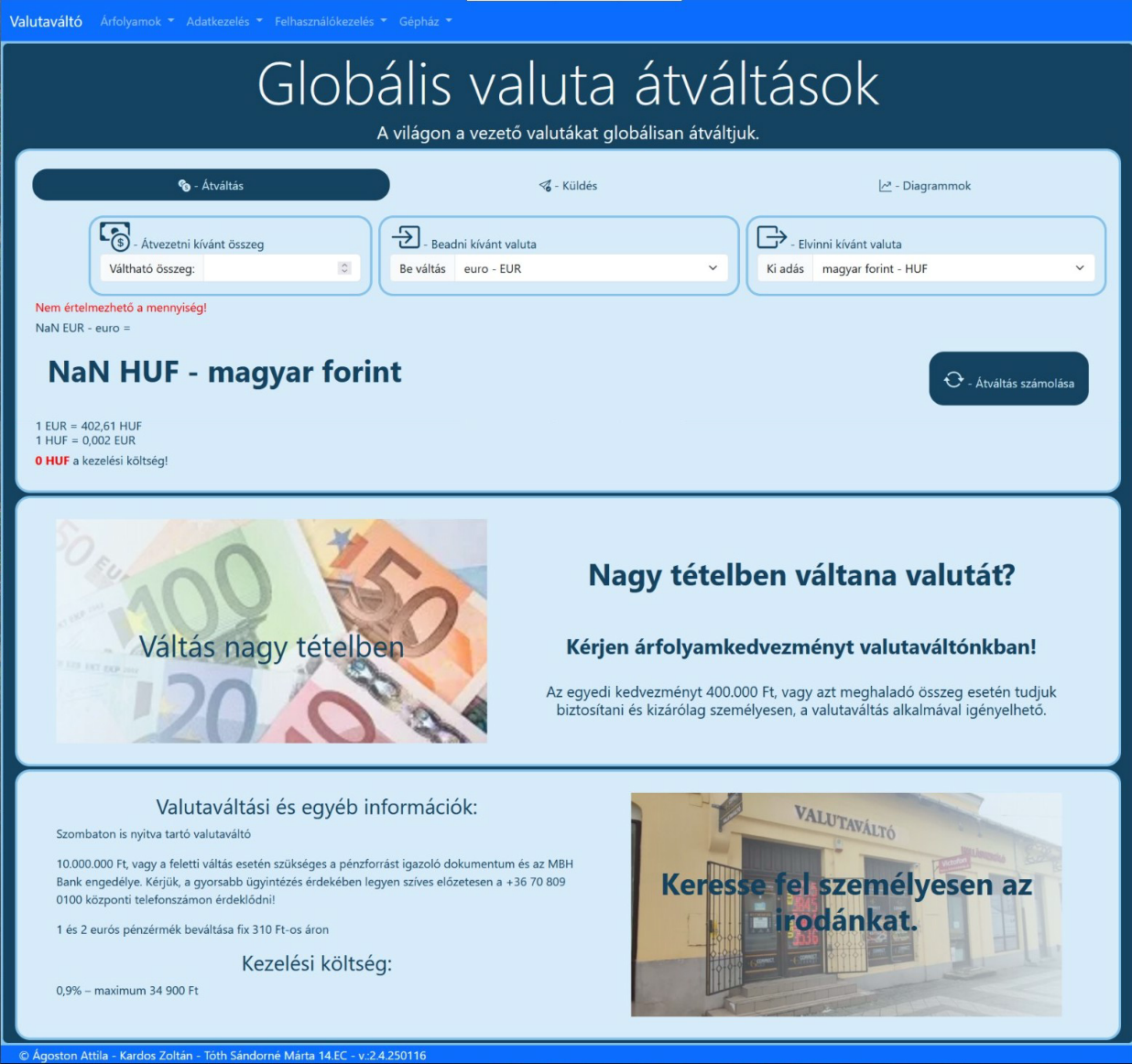
#### Reszponziv felületek

A felhasználói felület kialakításánál figyelembe lett véve, hogy számos funkciót kell megvalósítani a szoftvernek. A menürendszert a hagyományos vízszintesen elhelyezkedő főmenüvel és az azokból függőlegesen lenyíló almenükkel lett megvalósítva.

Ezt a struktúrát a kisebb kijelzőjű eszközökhöz szendvicsmenüvé alakítottuk át. A menürendszer és a megjelenítés reszponzív kialakításához a Bootsrap keretrendszert használtuk.

### A weboldalak

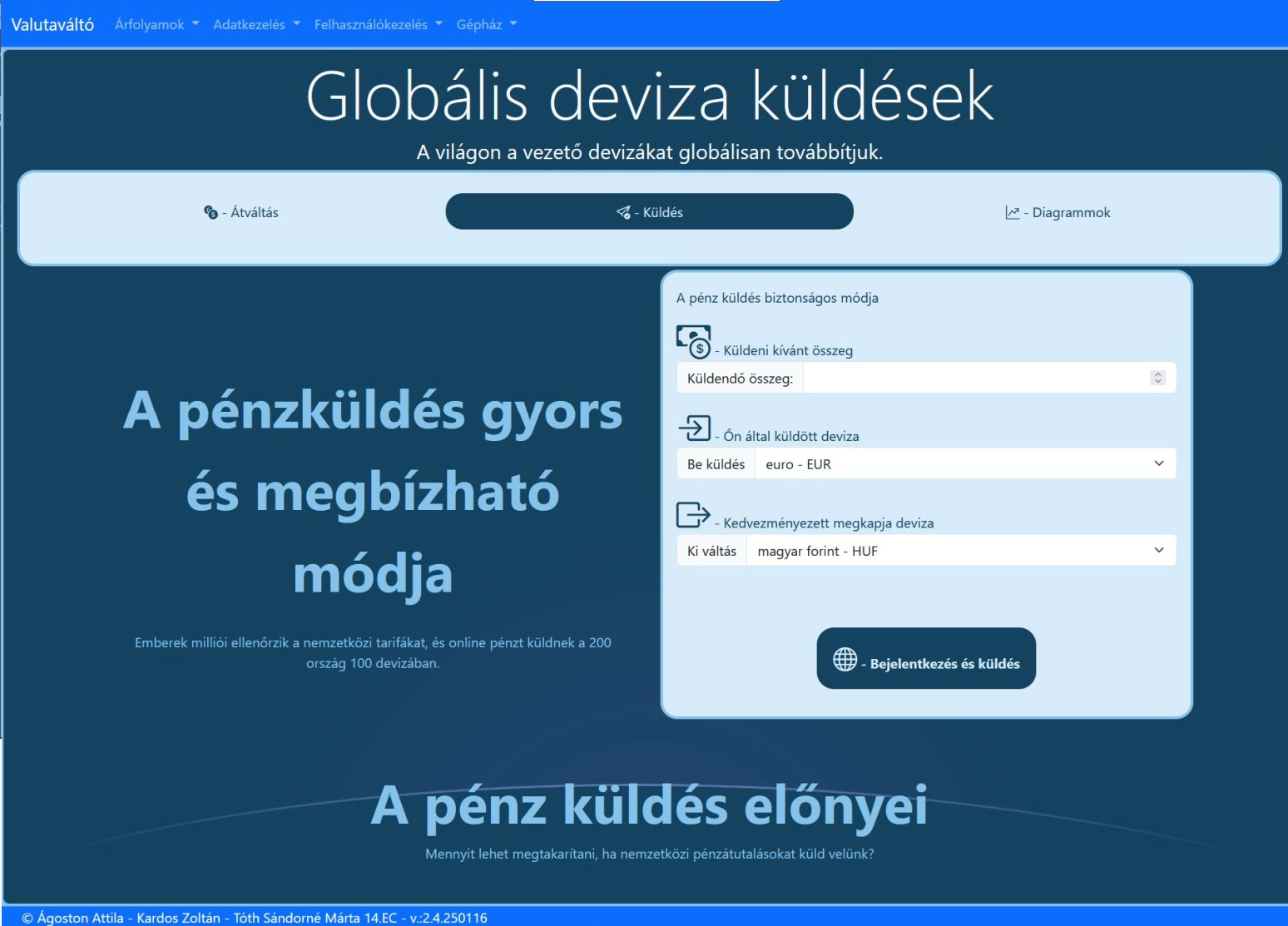
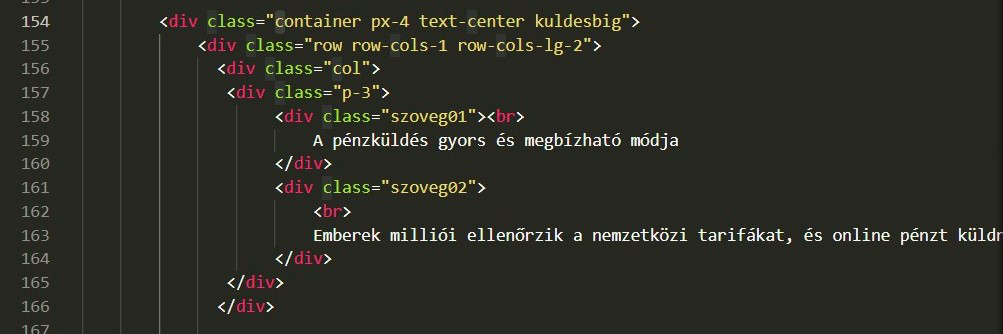
A weboldalakat sima HTML lapokból építettem fel CSS formázással (**Bootsrap** keretrendszerrel) és az interakciókat JavaScript végzi el azoknál az oldalaknál ahol nincs bejelentkezési köztelezettség, szabadon látogathatóak. Ezeknél az oldalaknál a szerver minimális terhelése, mentesítése, a felhasználói oldal erőteljesebb igénybevétele volt a cél. Így a három bejelentkezést nem igénylő oldal a **valto.html**, **kuldo.html** és a **diagram.html** esetében a webszerver (Apache Web Server) feladatán kívül minden feladat a felhasználó eszközén valósul meg.

Az adatok lekérdezését JacaScript **valto.js** állományban lévő *getValutaadatok()* függvény végzi el és *.innerHTML* jelenik meg a HTML dokumentumban. A számítási feladat a váltások kalkulációja is JavaScript függvénnyel valósult meg, ezek mind a kliens rendszerét terheli.

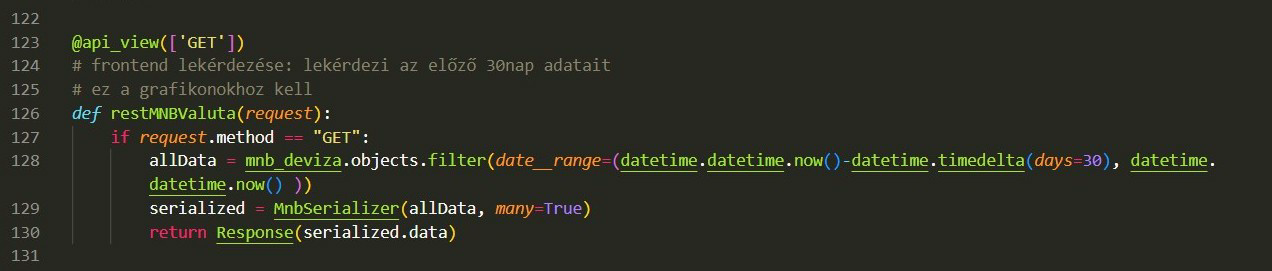
### Átváltás - valto.html

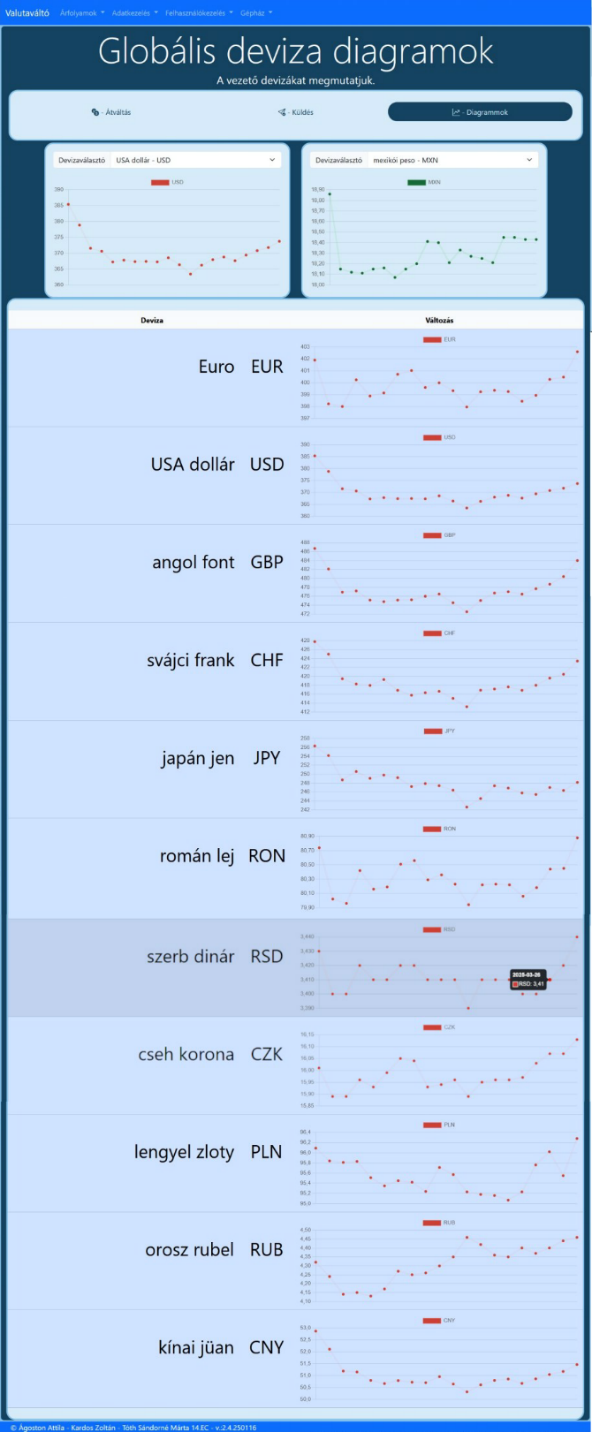
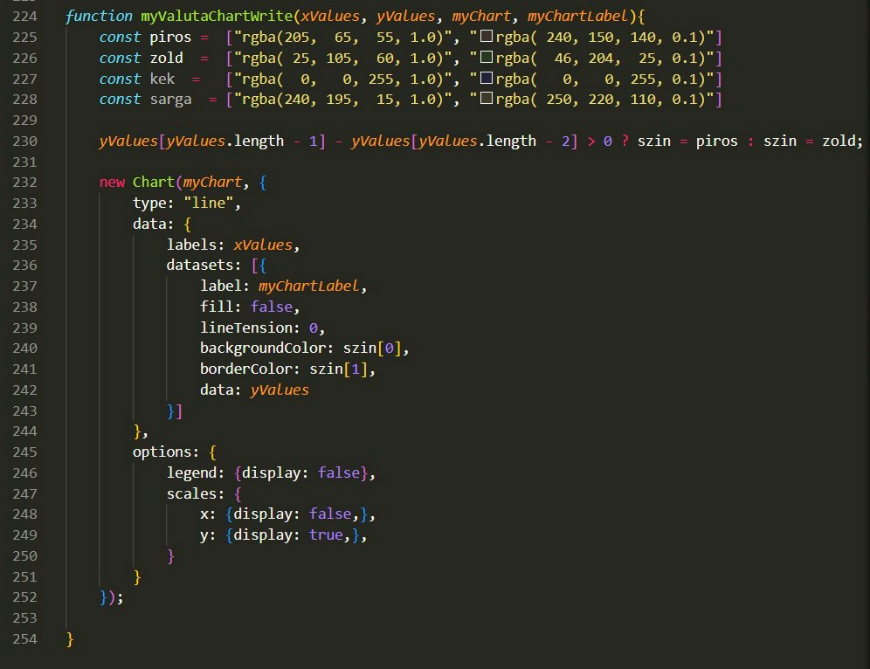
Ez az oldal a főoldal. Itt valósul meg a korrekt tájékoztatás. Ez az oldal mindenki számára szabadon elérhető. Tájékoztat avalutaváltás aktuális árfolyamáról, itt végezhető el a kalkuláció, amit *JavaScript* végez. Statikus módon tájékoztat a nagy tételben történő váltásról, a váltás költségéről, a valutaváltó elérhetőségéről mely fotóval megjelenített. A **kalkuláció** - onload eseménykezelésre - a teljes oldal betöltését követően a felhasználó kiválasztásai és a lefetchelt valutaadatokat felhasználva az **[Átváltás számolása]** gomb megnyomására kerül végrehajtásra.

### Küldés – kuldo.html

Ez az oldal a bejelentkezési funkciót hivatott megoldani. A felhasználó a deviza küldést választva jut ide. A paraméterek beállítását követően a **[Bejelentkezés és küldés]** gombot megnyomja és átnavigál a bejelentkezési oldalra, mert a küldés az regisztrációhoz kötött funkció. Ezen az oldalon a devizaváltás kalkulációját onblur="szamol()" eseményre JavaScript szamol függvényünk végzi el. Az oldal kialakítását a Bootstrap container col és a row osztályaival oldottuk meg.

### Diagramok – diagram.html

Ez az oldal a látványos megjelenítése a devizaárfolyamoknak. A fetch get előre maghatározott mennyiségű, - de paraméterezhető - visszamenőleges adatot küld a kliensnek, hogy devizaárfolyam változást tudjunk megjeleníteni.

Az oldal előre paraméterezhető módon - a JavaScriptben – jeleníti meg a kiválasztott deviza árfolyamváltozásait. A felső szekcióban elhelyezett jobb és baloldali diagram dinamikusan jeleníti meg a felhasználó által összehasonlítani kívánt devizákat. A lefetchelt adatokat tömbökben tároljuk a gyors megjelenítés érdekében. A tömbök adatait a Chart.js általános reszponzív grafikonkészítő könyvtár segítségével jelenítjük meg.

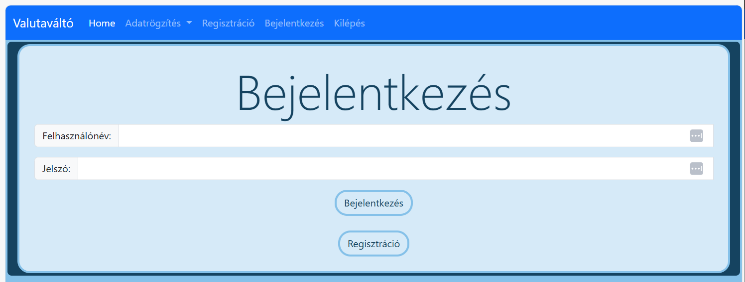
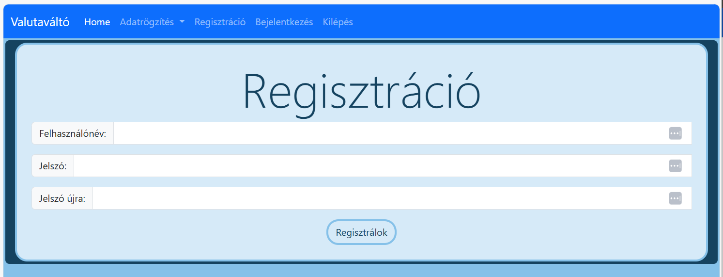
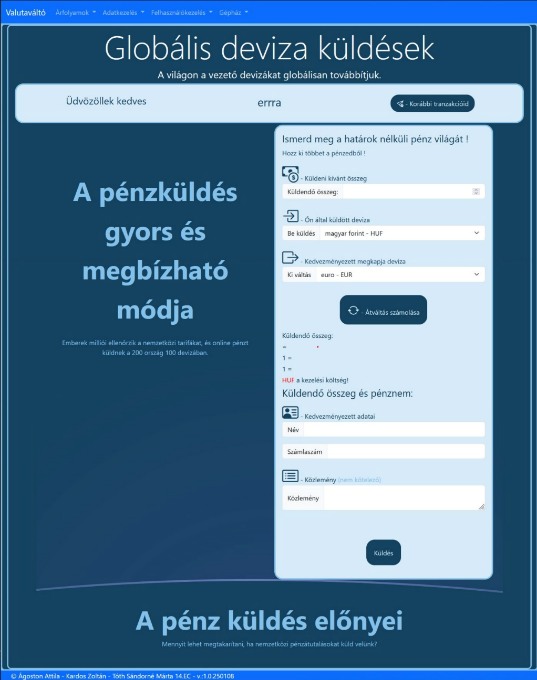
A regisztrációhoz kötött oldalak *home.html* (tényleges küldés oldala), tranzakciok.html (a bejelentkezett felhasználó korábbi tranzakciójának listázását végző oldal) Django templates -el vannak megvalósítva. A Django keretrendszer beépített sablonjaival könnyen megoldható a felhasználó kezelés. Egyszerűen a menürendszerben is használtuk a felhasználókezelését a Djangonak. A sablonok öröklését kihasználva a menürendszert és az elrendezéseket a *base.html* segítségével oldottuk meg.

### Bejelentkezés – templates login.html

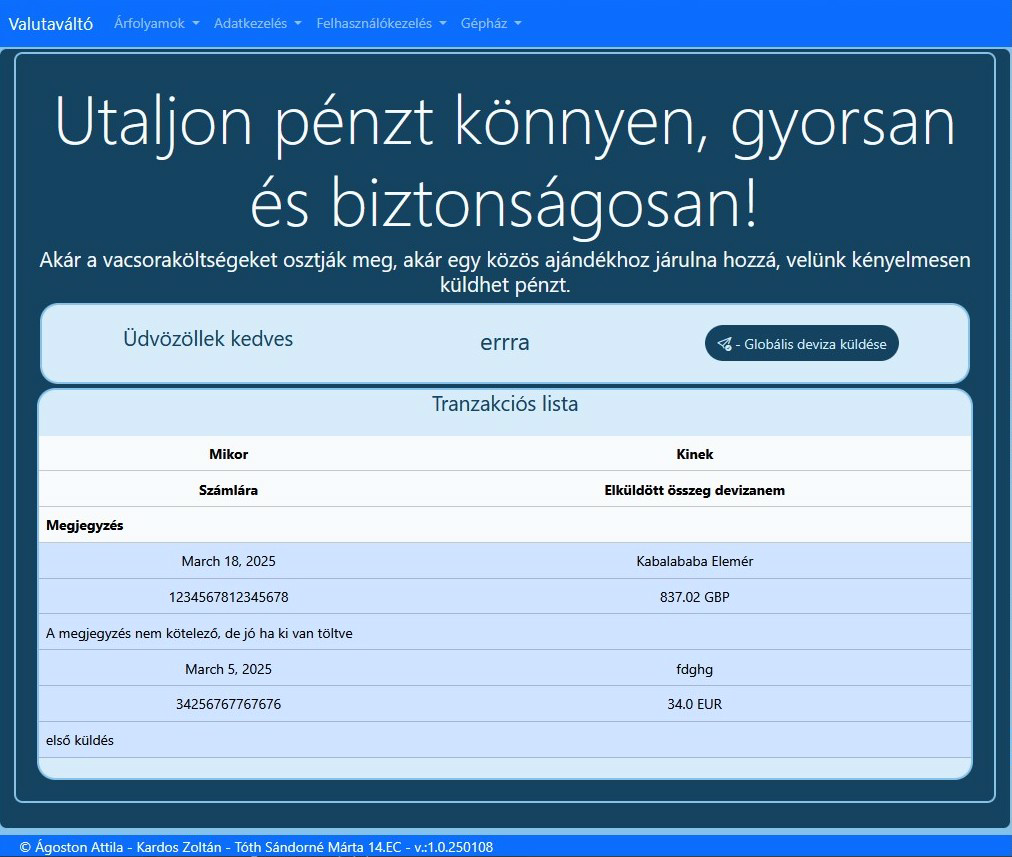
### Regisztráció – templates signup.html

### Deviza küldése – templates home.html

Zoli csak a dizájnt csinálta, hogy illeszkedjen a meglévő oldalakhoz.A honlap szerkezete a backend rendszer template-jeként lett megvalósitva.



### Tranzakciók listája – templates tranzakciok.html

A listázás reszponzív megjelenítése feladta a kérdést, hogyan lehet a töredezetlen hosszú szavakat, sorokat megjeleníteni minden eszközön. Melyik a jobb megoldás táblázat formájában, sorokban, oszlopokban talán?

### Üzemeltetés

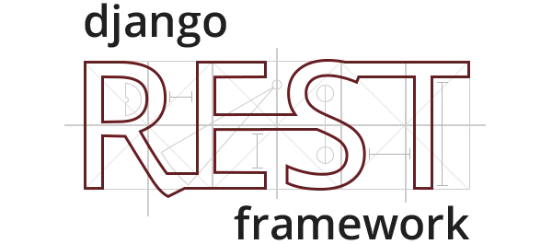
#### Fejlesztői környezet kialakítása:

##### Python, Django, Django RestFramework

A Django keretrendszer használatához python programkörnyezet szükséges. Python virtuális környezetet használunk azért, hogy a függőségekből adódó problémákat elkerüljük.

A következő python könyvtárakra lesz szükségünk:

• Django

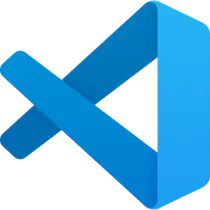
• Django requests   
• DjangoRestFramework   
• DjagoCorsHeaders   
• Pandas

• OpenPyXl

### GitHub

A team tagjaival a GitHub verziókezelő rendszert használjuk A kódjaink tárolása, megosztása egyszerűen lett megoldva, a forráskód elérhető a <https://github.com/lukrecia602/project/> címen a min brandban.

### Visual Studio Code

A forráskódokat a Visual Studio Code kódszerkesztőben hoztuk létre. Szerverüzemelés

### Docker

A Django szervert, ha belerakjuk egy docker konténerbe, akkor könnyű szerrel el tudjuk érni egy DynDNS szolgáltatás egítségével bármilyen külső hálózatról. Én a Dynu szolgáltatót preferáltam. A backend rendszert így : <http://azenhazam.mywire.org:8800/> webcímen tudjuk elérni, és a backend rendszer tesztelése is lehetséges.

### Linuxweb

A HTML, CSS, JavaScript állományokat a Linuxweb webhoszting szolgáltatóra bíztam, hogy folyamatosan elérhető legyen a számunkra. Így a projekt kezdőoldalát a <http://fomix.hu/aladar/valto.html> webcímen tudjuk elérni és a frontend rendszer tesztelése is itt lehetséges.

## Tesztelés

A tesztelés igen fontos folyamat, mely során az összeállított rendszert ellenőrizzük, hogy megbizonyosodjunk a megfelelő működésről, minőségről és teljesítményről.

• Hibák problémák felderítése:

A kalkuláció során az inputadatokat körültekintően szűrni kellett. A NaN válasz elkerülése miatt.

• Program viselkedésének ellenőrzése különböző körülmények között:

a tesztüzem során külső tesztalany a menü kivetítőn történő színbeli összeférhetetlenségre hívta fel a figyelmünket. Megállapította, hogy gyakorlatilag a háttérszín és a betűszín választása nem megfelelő magas fényviszonyoknál sem. A menü színösszeállítását lecseréltük.

• A rendszer teljesítményének és megbízhatóságának ellenőrzése:

a docker környezetben való üzemeltetés erősebb hardvert igényel a jelenlegi i3 desktoptól. Az jelenlegi oprendszer (win10) sem biztosít hosszú távú megbízható üzemeltetést.

• Tesztelés, hogy a rendszer megfelel a specifikációnak és a megrendelő igényének.

### Fejlesztési lehetőségek

• Az MNB devizaárfolyamának frissítése manuálisan történik, ezt automatizálni lehetne

pl. docker linux környezet kiaknázása egy corn job (időalapú munkamenet ütemező) segítségével.

• Valutaváltás kedvezményi rendszer bevezetése mennyiségi kedvezmények

népszerűsítése.

• Devizaváltás banki szolgáltatás biztonsági fejlesztése.

#### Irodalomjegyzék

Python: <https://www.python.org/>

Pandas: <https://pandas.pydata.org/>

Openxl: <https://pypi.org/project/openpyxl/>

Bootstrap: <https://getbootstrap.com/>

GitHub: <https://github.com/>

Docker: <https://www.docker.com/>

Dynu: <https://www.dynu.com/>

A W3Schools oldalai: [W3Schools Online Web Tutorials](https://www.w3schools.com/)

MySQL kézikönyv: [MySQL :: MySQL 8.4 Reference Manual](https://dev.mysql.com/doc/refman/8.4/en/)

ChatGPT: [Introducing ChatGPT | OpenAI](https://openai.com/index/chatgpt/)

Powerpoint tömörítő: <https://www.youcompress.com/hu/powerpoint/>

DB Browser: <https://sqlitebrowser.org/dl/>

## Mellékletek

Elérhető a <https://github.com/lukrecia602/project/blob/documentation/Mell%C3%A9kletek.docx>

címen. Tartalmazza az adatbázis dumpokat, az adatbázis sémáját, a saját készítésű kódokat és a telepítési útmutatót.