

**Vak Bottyán János Katolikus  
Műszaki és Közgazdaság Technikum,  
Gimnázium és Kollégium**



**54 213 05 Szoftverfejlesztő  
szakképesítés**

**Mini Napló**

**Készítette: Silkó Levente**

**2023**

# Tartalomjegyzék

<b>Tartalomjegyzék</b> .....	2
<b>Előszó</b> .....	3
<b>Bevezető</b> .....	4
Megvalósíthatósági elemzés.....	5
Elvárások: .....	5
Megvalósíthatósági elemzés:.....	5
Opcionális (ha nem helyi szerverről fut):.....	5
Alkalmazni kívánt programozási nyelv(ek): .....	6
Követelmények (funkcionális) .....	6
Követelmények (nem funkcionális): .....	7
Működés:.....	8
Felhasználói felület:.....	8
<b>Felhasználói dokumentáció</b> .....	9
Bevezetés: .....	9
Felhasználói felület.....	10
Bejelentkezési képernyő:.....	10
Főoldal felépítése.....	13
Órarend oldal felépítése .....	14
Értékelések oldal felépítése: .....	16
Hiányzások oldal kinézete .....	19
Felhasználó profil oldalának felépítése .....	20
Felhasználói felület kiegészítő elemei .....	21
<b>Fejlesztői dokumentáció</b> .....	24
Specifikációk.....	24
Bevezetés .....	25
Problémák.....	26
Megoldások.....	26
Projekt mappa szerkezete .....	27
Statisztika.....	28
Bevezetés a program kódba .....	28
A fő fájl.....	29
Adatok lekérdezése .....	36
Kulcsszavak magyarázata.....	38
Programhoz használt könyvtárak .....	40
Fejlesztői környezet.....	41
Használt fejlesztési modell .....	42
További fejlesztési lehetőségek.....	43
Felhasznált irodalom .....	44
Felhasznált eszközök .....	44
Plágium nyilatkozat .....	45

# Előszó

Az oktatás iránt elkötelezett diákok számára rendkívül fontos az iskolai információkhoz való hozzáférés, mint például az adott napon tartandó órák, az elért jegyek vagy más hasonló információk. A kormány által kifejlesztett Kréta alkalmazás erre nyújt megoldást, azonban a felhasználók tapasztalhatták, hogy az alkalmazás kinézete és funkcionalitása kissé elavult.

Ezen a ponton a jelenlegi projekt célja, hogy modern dizájn stílusokat, elemeket és újra gondolt elrendezéseket használjon, hogy frissítse és élettél tölthesse meg a felhasználói felületet, miközben egyszerűbbé teszi annak használatát. Ennek eredményeként az alkalmazás használata még hatékonyabb és kényelmesebb lesz a felhasználók számára.

# Bevezető

A záródolgozat témája egy mobil-os elektronikus napló alkalmazás létrehozása, amely a saját tapasztalatból fakadó problémákat kívánja megoldani, miközben az alkalmazás felületét újra gondolja és egyszerűsíti. Az alkalmazás célja olyan megoldást biztosítani, amely az átlagos felhasználó számára is vonzó, modern és letisztult. Eközben hatékonyan kezeli a régi problémákat. Tervezésnél fő szempontok között sorakozott, hogy mindenkinek egyértelmű kezelést, könnyű navigálást, és gyors működést biztosítson. Az alkalmazáshoz választott technológiákkal a hatékony, és megbízható működést szolgáltat a felhasználó számára.

## Megvalósíthatósági elemzés

### Elvárások:

- Felhasználó saját fiókjával be tud jelentkezni.
- Megtudja tekinteni a legutóbbi eseményeket, jegyeket.
- Láthatja az aznapi óráit
- Jegyeit tantárgyak szerint átlagolva megtekintheti, és diagram(okkal) kiegészítve.
- Hiányzásait követheti.
- Általános információkat megnézheti a fiókjáról.

### Megvalósíthatósági elemzés:

- Humán erőforrás: 1 fejlesztő, óraszám: ~180
- Hardver erőforrás:

### Minimum:

- Operációs rendszer: Android 11 (ios, windows, web)
- Memória: 1GB
- Háttértár: Minimum 300MB
- Periféria: Érintőképernyő

### Ajánlott:

- Operációs rendszer: Android 13 (ios, windows, web)
- Memória: 4GB
- Háttértár: 1GB
- Periféria: Érintőképernyő

### Opcionális (ha nem helyi szerverről fut):

- Kréta szerverek

### Alkalmazni kívánt programozási nyelv(ek):

- Dart
- Flutter (keretrendszer)
- GetX (state / route menedzser)
- Hive (adatbázis)

### Szoftver erőforrás:

- Visual Studio Code (v1.76.2)
- Android Studio Emulator

### Üzemeltetés:

- Csak az alkalmazás telepítésére van szükség

### Karbantartás:

- Nem kell biztosítani

### Megvalósítás:

- Kb. 180 óra, Költség: Nincs (csak villanyszámla, wifi/mobilnet [3MB])

### Követelmények (funkcionális)

Bejelentkezési oldal megjelenítésekor:

- Szövegdobozokkal való interakció,
- Gomb bejelentkezéshez.

Főoldal megjelenésekor:

- Alsó Navigációs Menü segítségével navigálás.
- A Profil oldalon jobb fent kijelentkezés gomb

Oldalválasztás végrehajtása:

- Navigációs menüre való érintéssel

Hatékonyság:

- Az alkalmazás könnyedén folyamatosan fut.
- Gyors válaszidő
- Pontos visszajelzés

Megbízhatóság:

- Hibalehetőség maximum szerver oldalról történhet.

Követelmények (nem funkcionális):

Biztonság:

- Biztonságos adatkommunikáció a telefontal.
- Biztonságos szerver kérések.

Hordozhatóság:

- Telefonon használva lehetséges

Felhasználhatóság:

- Megfelelő instrukciók biztosítása

Környezet: -

### Működés:

- Gyors működés
- Megbízható használat
- Biztonságos működés
- A hibalehetőségek kizárása
- Gyakori használat

### Felhasználói felület:

- modern design
- Átlátható felhasználói felület
- Könnyen kezelhető
- Interaktív elemek
- Intuitív felület
- Animált váltások



# Felhasználói dokumentáció

## Bevezetés:

Az alábbi mobil alkalmazás, egy létező és a közoktatásban ismert digitális naplót (Kréta) alkalmazást, próbálja újra gondolni, modern dizájn trendekkel, és bizonyos esetekben fejleszteni a már meglévő funkcionalitást.

Az alábbi dokumentáció azt a célt szolgálja, hogy egyszerűen és érthetően, elmagyarázza az alkalmazás felépítését és használatát. Illetve arra szolgál, hogy bemutassa az összes interakciót, amivel a felhasználó találkozhat.

## Felhasználói felület dizájnok:

A felhasználó felület megtervezésében nagy szerepet játszott az úgynevezett Glassmorphism, egy viszonylag új és divatos dizájn trend a webdesign területén, amely az átlátszó, üvegszerű hatásokra épül. Továbbá az alkalmazás felületének fókusza főleg a letisztultságot vette célba, hogy a felhasználónak ne okozzon gondot az eligazodás az alkalmazáson belül.

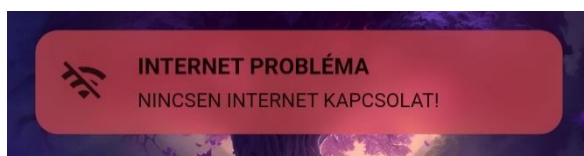
## Mi is az a Glassmorphism?

A Glassmorphism trend azért is különleges, mert a tervezők általában árnyékokat, mélységet és dimenziót használnak, hogy az elemek kiemelkedjenek a háttérből, míg a Glassmorphism a hatásnak köszönhetően éppen az átlátszó felületek és az áttűnő háttér miatt fokozza a mélységet és dimenziót.

Összességében a Glassmorphism egy modern, látványos és letisztult megjelenésű dizájn trend, amely a minimalista és átlátszó hatásokat használja, hogy az elemek dinamikusabbak és szembe ötlőbbek legyenek.

## Stílus megjelenése az alkalmazásban:

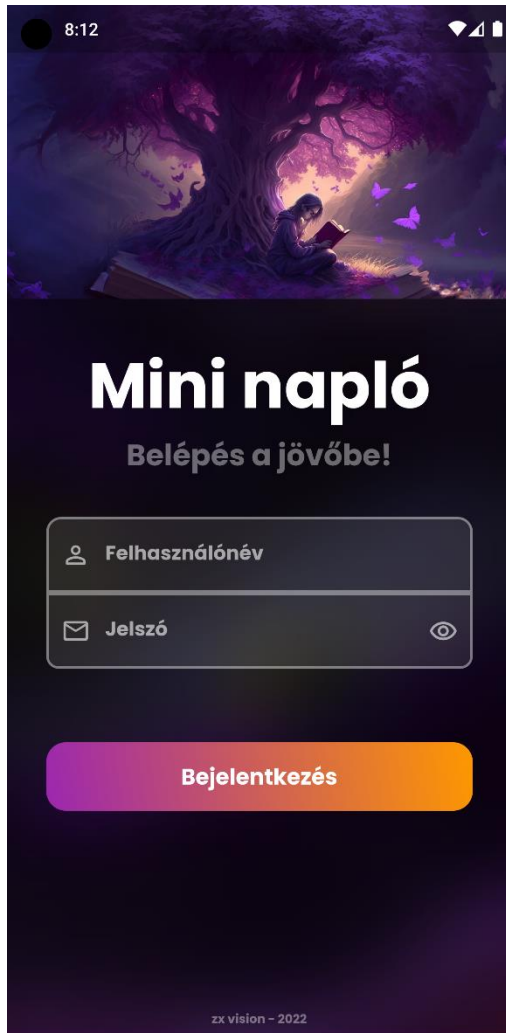
- Bejelentkezési képernyő szöveg dobozok
- Alsó navigációs menü
- Oldalakon található csempék (pl: Órarend oldal csempéje)
- Internet indikátor
- kijelentkezés gomb



Példának az internet indikátort hoztam, mivel ezzel a legkönnyeb demonstrálni az üveg hatás.

## Felhasználói felület

### Bejelentkezési képernyő:



### Felépítése

Fejléc kép: ezt a képet a Midjourney AI-al készítettem. Röviden a működéséről: Megjuk az AI-nak hogy miket szeretnél a képeden látni, specifikus beállításokkal, mint például a képarány, ezek után generál 4 darab képet, és kiválaszthatod, hogy a négy kép közül melyiket kiválaszthatjuk hogy melyiket szeretnénk felskálázni, vagy új verziókat generálni.

Az interfész egy úgy nevezett Stack-en helyezkedik el, és a felhasználó felület és a háttér között van egy homályosítás effekt és ezen helyezkedik a Cím, Szlogen stb.

Két darab szöveg dobozzal (flutterben TextField-nek hívják) történik a bejelentkezéshez szükséges adatok megadása.

A bejelentkezésre kattintás után elindul az adatok validációja, ezután kérést intéz a kréta szerverek felé, és ha minden sikeres akkor át helyez a fő képernyőre.

### Előfordulható hibák:

- Üres felhasználónév szövegdoboz
- Üres jelszó szövegdoboz
- Hibás felhasználónév vagy jelszó (szerver hívásnál derül ki)
- Nincsen internet

### Hibák megjelenítése a felhasználónak:

Előfordulható hibák a gomb szövegének kicserélésével jelenítődnek meg.

Mini napló

Belépés a jövőbe!

Felhasználónév

Oktatási azonosító

Jelszó

Üres Felhasználónév mező!

Mini napló

Belépés a jövőbe!

Felhasználónév

7238403752

Jelszó

Üres Jelszó mező!

Mini napló

Belépés a jövőbe!

Felhasználónév

7238403752

Jelszó

...

Hibás Jelszó, vagy Felhasználónév

INTERNET PROBLÉMA  
NINCSEN INTERNET KAPCSOLAT!

Mini napló

Belépés a jövőbe!

Felhasználónév

Jelszó

Nincsen internet!

Sikeres bejelentkezést követően a felhasználó átkerül a főoldalra, és itt az alsó navigációs menüvel lehet lépegetni az oldalak között.

Az alsó menün az adott oldalak lesznek elérhetőek:

Oldalak:

- Főoldal
- Órarend
- Értékelések
- Hiányzások
- Felhasználó profil

A legtöbb oldalon hasonló felépítés található: egy fejléc, tartalom és az alsó navigációs menü.

Az alsó Navigációs menü felépítése:



Minden oldalt egy ikon jelez, ami rákattintáskor válik aktívvá.

Rákattintás után a kiválasztott ikon színe felerősödik, és alatta megjelenik egy kis lila aláhúzás. Ezek szerepe, hogy a felhasználó mindig tudja, hogy melyik oldalon is jár.

Ezek mellett, egy animáció is elindul kattintáskor.

## Főoldal felépítése



A képernyő tetején a fejléc látható, amely az oldal nevét tartalmazza. Alatta egy halványabb felirat tudatja a felhasználóval, hogy miket is lát az adott oldalon.

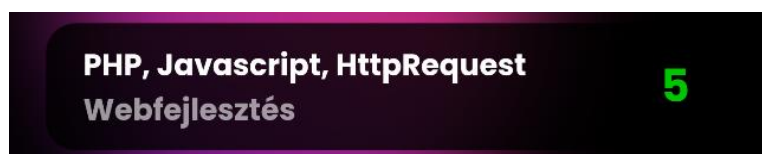
A fejléc alatt van a tartalom rész, ahol úgy nevezett csempéket láthat a felhasználó.

Egy csempe egy tanár által beírt jegyet jelenít meg.

Alul az alsó navigációs menü található, ami minden oldalnál megtalálható elem.

Mindezek mögött egy Rive animációs háttér van, amiben 3 pötty mozgását lehet megfigyelni.

Egy csempe felépítése:

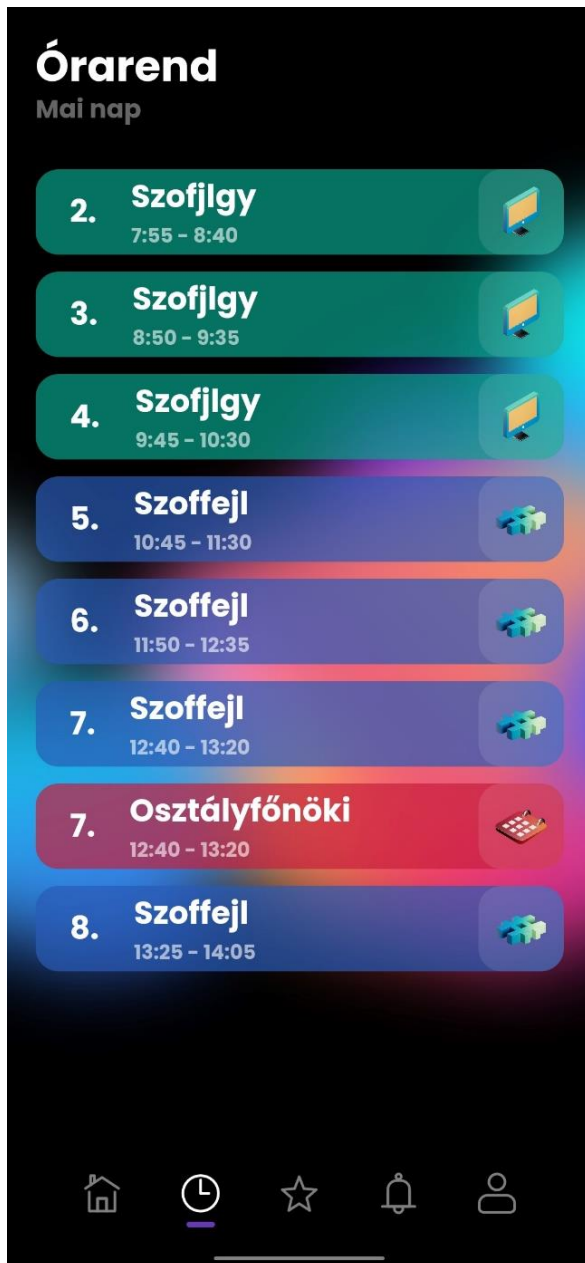


A csempe enyhén átlátszó ezáltal lehet látni a háttérben lévő animációt.

Csempén megtalálható tartalom:

- Leírás
- Tantárgy
- Osztályzat

### Órarend oldal felépítése



A fejlécen az oldal neve, és hogy mit is lát pontosan a felhasználó, ez mindig az aktuális napot jeleníti meg.

Ezek után látható az aznapi órarend

Ennek az oldala különböző Rive animációt használ, mint a többi. Ez a háttér még egy statikus képet is használ. Ami így néz ki:



Minden óra egy 'csempe'. Ami itt is ezt az úgynevezett üveghatást hozza, ezért nem takarja ki teljesen a mögötte lévő animációt



Egy csempe felépítése:

- Óra száma az aktuális napon
- Tantárgy
- Mettől meddig tart az aktuális óra
- Ikon, ami mindig a tantárgyhoz kapcsolódik

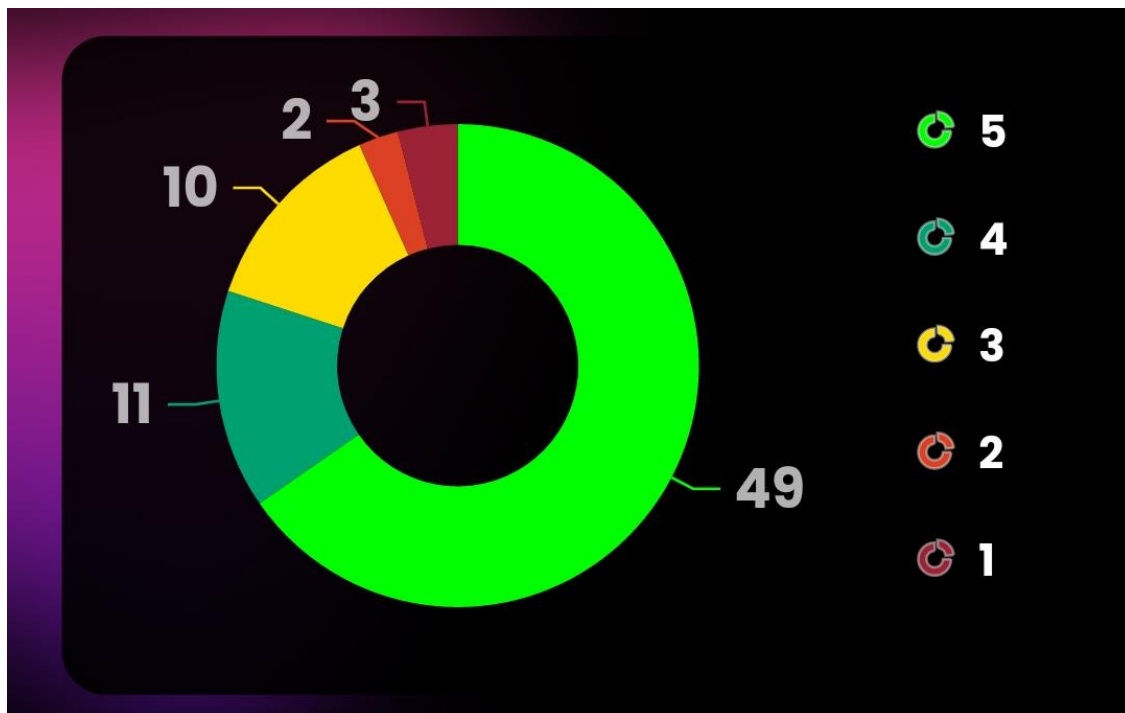
## Értékelések oldal felépítése:





# Értékelések

A fejlécben ismét az oldal címe.



Ezen az oldalon más az elrendezés, mivel itt két fő panelre van bontva.

A fent látható képen az első panel látható. Ezen a panelen egy kör grafikon helyezkedik el, ami azt mutatja, hogy melyik jegy fajtából mennyi darab van a felhasználónak. Az elrendezése, a grafikon darabszámokkal ellátva, és a jobbszélen a jegyek.

<b>Foglalkoztatás II.</b>	<b>5.00</b>
<b>Hittan</b>	<b>5.00</b>
<b>Webfejlesztés gyakorlat</b>	<b>4.92</b>
<b>Webfejlesztés</b>	<b>4.63</b>
<b>Szoftverfejlesztés</b>	<b>4.34</b>
<b>Szoftverfejlesztés gyakorlat</b>	<b>4.29</b>
<b>Foglalkoztatás I.</b>	<b>3.40</b>
<b>Szakmai idegen nyelv</b>	<b>2.75</b>

A második panelen tantárgyak szerint látható az átlag, amit két tizedes pontossággal jelenítődik meg felhasználónak, és ezek csökkenő sorrendben sorakoznak. Minden átlaghoz egy szín is társítható a kerekítés függvényében. Ennél a panelnél is megfigyelhető az érdekes felépítést tekintve, mivel szintúgy átlátszósággal játszik, de a tantárgyak között kis rések figyelhetők meg, és az elsőnek a felső jobb és bal oldalán megtalálható kerekítések, és ez csak az elsőnél van, illetve az utolsó két oldalánál az alsó és a felső elem között csak téglalap formák jelenítődnek meg

## Hiányzások oldal kinézete



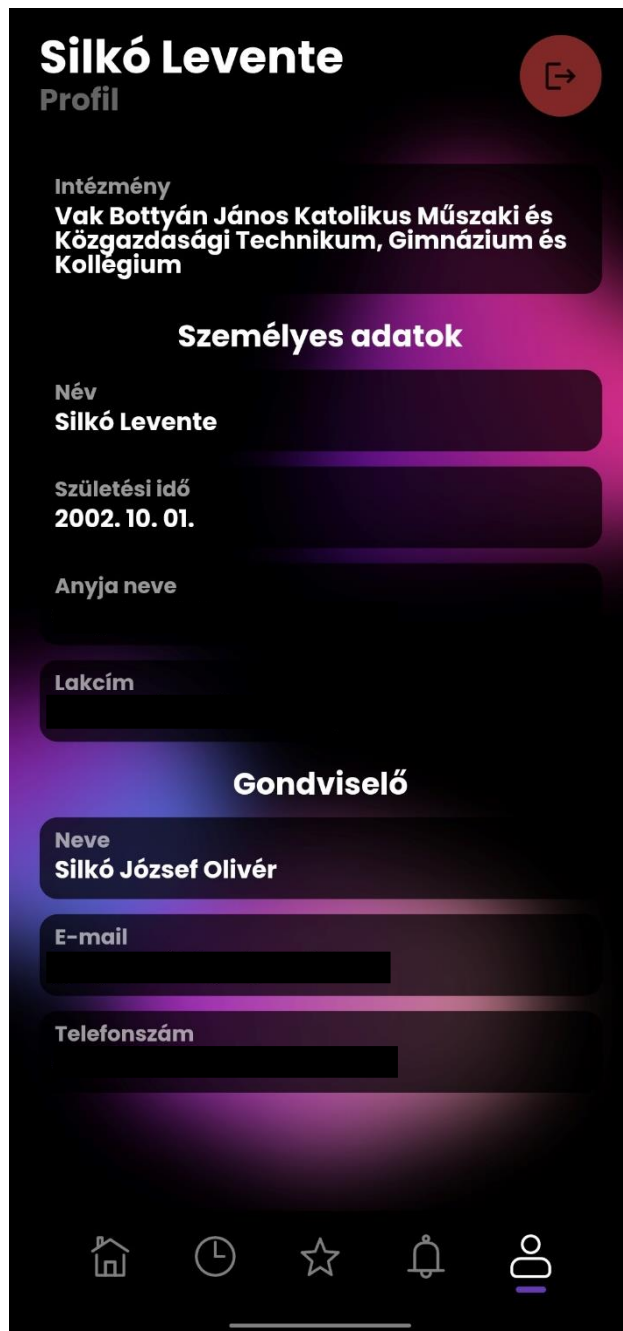
Fejlécben az oldal elnevezése

Alatta két darab csempe helyezkedik el az egyiken az igazolt hiányzások összessítése látható a felhasználó számára, míg a másikon az igazolandó hiányzások.

Ezután tanárgyanként listázva látható az adott tantárgyról való hiányzás.

Tantárgy hiányzás panelt 3 részre lehet osztani, az első rész ami azt mutatja hogy mennyi órát hiányozott a felhasználó, a második része a tantárgy neve, és a harmadik eleme az ikon.

## Felhasználó profil oldalának felépítése



The image shows a mobile app interface for a user profile. At the top, the name 'Silkó Levente' is displayed in large white text, with 'Profil' below it in a smaller font. To the right of the name is a red circular button with a white right-pointing arrow. Below this is a dark blue box containing the text 'Intézmény' followed by 'Vak Bottyán János Katolikus Műszaki és Közgazdasági Technikum, Gimnázium és Kollegium' in white. The next section is titled 'Személyes adatok' in white. It contains four input fields: 'Név' with the value 'Silkó Levente', 'Születési idő' with the value '2002. 10. 01.', 'Anyja neve' (empty), and 'Lakcím' (empty). Below this is a section titled 'Gondviselő' in white. It contains three input fields: 'Neve' with the value 'Silkó József Olivér', 'E-mail' (empty), and 'Telefonszám' (empty). At the bottom of the screen is a navigation bar with five icons: a house, a clock, a star, a bell, and a person. The person icon is highlighted with a blue underline.

**Silkó Levente**  
Profil

Intézmény  
**Vak Bottyán János Katolikus Műszaki és Közgazdasági Technikum, Gimnázium és Kollegium**

**Személyes adatok**

Név  
**Silkó Levente**

Születési idő  
**2002. 10. 01.**

Anyja neve

Lakcím

**Gondviselő**

Neve  
**Silkó József Olivér**

E-mail

Telefonszám

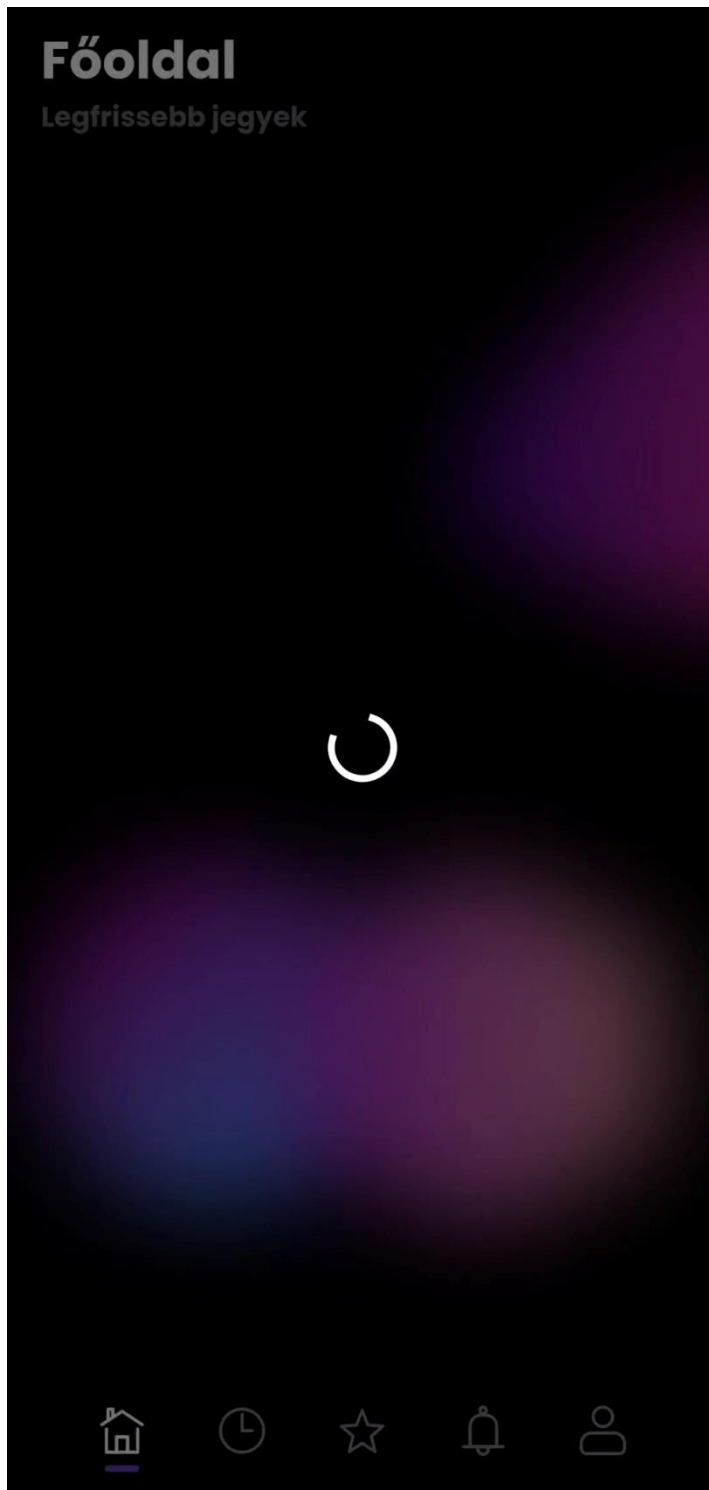
A fejlécben vélhető jelentősebb eltérés az előző oldalakhoz képest, az oldal neve helyett dinamikusan változóan megjelenítődik a felhasználó neve, alatta egy statikus profil szöveggel.

A fejlécben továbbá megtalálható egy kijelentkezés gomb, ami visszadob a bejelentkezés képernyőre.

A fejléc alatt az adott tanulóról jelenítődnek meg az információk külön csempéken.

## Felhasználói felület kiegészítő elemei

Amint az alkalmazást a felhasználó már sikeres belépés után bezárja, és következő használatnál újra nyitja egy kis középre igazított töltés indikátort lát, ez addig látható amíg az adatok betöltésre nem kerülnek, ez alatt enyhe sötétítés kerül a felhasználó felületre.



## Adat frissítés

Az adat frissítése egy kézmozdulattal történik, mint minden modern telefonos alkalmazásban manapság.

Működése főoldalon újjal történő lehúzással aktiválható.

Kinézete a kép tetején látható, lehúzásnál egy kis animáció játszódik le és ezután lép a képen látható állapotába.

Amint végződik az adatok frissítése az oldal felépítése visszatér az eredeti elrendezéséhez.



## Internet hiánya indikátor

Az alkalmazás a kréta szervereiről kéri le az adatokat ezért elhanyagolhatatlan, hogy legyen internet kapcsolat, mivel internetet igényel a bejelentkezés, adatok lekérése, és adatok frissítése, ezért fontos, hogy tudassuk a felhasználóval, hogy miért nem tud elérni egyes funkciókat, mint azt a bejelentkezés képernyőn is láthatóvá teszi a fenti indikátor és még a gomb felirata is.

Amint az internetelés megszűnik, megjelenítődik egy beúszó animációval a képen látható indikátor, ez minden oldalon állandóan látható, ha az adatok már bevannak töltve akkor az oldalak között nyugodtan navigálhatunk, de a jelzés tudatja velünk, hogy nem lehetséges az adatok frissítése, mivel nincsen internet kapcsolat.

Amint az internetelés újra vissza állt az indikátor azonnal kiúszik a képernyőről, és a funkciók ismét elérhetővé válnak.



# Fejlesztői dokumentáció

## Specifikációk

A projekt elkészítéséhez Flutter keretrendszert választottam, ami a Dart programozási nyelvvel párosul. Adatbázisnak a Hive adatbázis rendszert választottam, ami nem relációs adatbázis(noSql). Az alkalmazás egy előre elkészített ikon csomagot használ, ami letölthető, ez a 3dicons néven található meg. State-et GetX könyvtár / keretrendszerrel implementáltam.

## De mi is az a Flutter?

Flutter egy nyílt forráskódú, felhasználói felületeket építő framework, amely lehetővé teszi, hogy egyetlen kódbázis segítségével készítsük el az alkalmazásunkat minden platformon, beleértve az iOS-t, az Androidot, a webet és a desktopot is. A Flutter alkalmazásokat a Dart programozási nyelvvel írják, amelyet a Google fejlesztett ki.



A Flutter nagyon népszerűvé vált az elmúlt években azáltal, hogy egyszerűbbé tette a mobilalkalmazások és a felhasználói felületek fejlesztését. A Flutter nagyobb rugalmasságot biztosít a fejlesztőknek, mert a platformok közötti átmeneteket automatikusan kezeli, így a fejlesztőknek nem kell külön-külön megírniuk a kódot minden egyes platformhoz. A Flutter előnyei közé tartozik az, hogy gyorsabb alkalmazásfejlesztést tesz lehetővé, a felhasználói felületek magas szintű testreszabását, valamint az, hogy a fejlesztőknek nem kell ismerniük több programozási nyelvet vagy platformot. A Flutter-t sok nagyvállalat használja, például a Google, a Alibaba, a Tencent és a BMW is. Az ilyen vállalatok általában olyan nagyméretű alkalmazásokat fejlesztenek, amelyeket több platformon kell futtatni.



## Flutter app felépítése

A Flutter működése az ún. "widgetek" használatán alapul, amelyek a felhasználói felületelemek építőkövei. A widgetek különböző formátumokban és méretekben jelenhetnek meg, és különböző adatokat, mint például szövegeket, képeket és gombokat jeleníthetnek meg. A widgetek összetételével lehet összerakni az alkalmazások felhasználói felületét, és különböző animációkkal és hatásokkal lehet őket tovább testre szabni.

A Flutter nagyon népszerű fejlesztők körében, mert lehetővé teszi, hogy az alkalmazásokat gyorsan és hatékonyan fejlesszék ki, miközben magas szintű testre szabást biztosítanak a felhasználói felületeken.

## Bevezetés

Az alap eltervezés, hogy készítsék egy modernebb krétát. Ehhez a következő dolgok szükségesek, először is megfelelő eszközt kellett kiválasztani a feladathoz, sok keresgetés, és utána járás következtében a flutter-re esett a választásom, az érve, hogy platform függetlenül tudom építeni az alkalmazásomat, ez a következőket jelenti, hogy ha akarom Windows-os környezetben futtatom, vagy a weben, illetve IOS-on is vagy Androidon is bármikor le buildelhetném, és futtathatnám. Az egyediség kedvéért maradtam az Androidnál. A keretrendszer megvan, ezután abba kellett bele gondolni, hogy miket is kell megvalósítani a program „backend-jének”. Ezek a következők lennének, felhasználó interakciókat kezelni, szerver hívásokat végezni, az alkalmazás állapotát menedzselni, adatokat tárolni, és még sok mást. Belegondolva egy komplex „piac kész” alkalmazást kellett építeni. Ebben a fejlesztői dokumentációban egy le egyszerűsített bevezetés található a program kódjába.

## Problémák

Az alkalmazás készítése közben az adott problémákkal szembesültem:

Kezdném azzal, hogy a flutter a dart programozási nyelvet használja, ezzel csak az a baj, hogy még a projektem előtt még nem is ismertem, tehát egy nyelv sajátosságait, illetve szintaxisát kellett elsajátítanom.

Második problémám az volt, hogy hogyan is érjem el az adott tanuló adatait a kréta szervereiről (jegyeit, órarendjét, stb)

Továbbá még problémák:

- hogyan mivel menedzseljem az alkalmazás state-jét
- mivel tároljam az alkalmazás működéséhez szükséges adatokat

## Megoldások

Szerencsémre nem nekem kellett vissza fejtenem a szerver kéréseket, hanem egy kis utána járással, rátaláltam egy github repositoryra ahol egy részletes dokumentáció található a vissza fejtett api-ról, szerencsémre egy python file-t is tartalmazott, amiben szinte mindent szerver hívást implementált a készítő, amit szinte csak átkellet írnom Dart kódra és úgymond már van is egy működő kódom, ami letudja kérdezni az adataimat, a kréta szervereiről.

A state menedzselésnek a flutter által biztosított StatefullWidget választottam elsőnek, de ez hamar bebizonyosodott, hogy ez a megközelítés nagyon limitált funkciókat biztosít. Szétnéztem, hogy mivel is tudnám megvalósítani amire nekem szükségem van, és eleinte nem tudtam melyik state menedzsert válasszam, mivel, ha megnézzük mennyi opció közül lehet választani flutter oldalán feltüntetettek közül akkor 16 darab lehetőség közül lehet választani. Mindegyiknek megvan az előnye és a hátránya is. Rengeteg video és kutatómunka után aGetX-re esett a választásom. Pár érv, hogy miért:

- Egyszerű szintaxis
- Teljesítmény
- Beépített modulok (Route menedzser, egyszerű widget-ek)

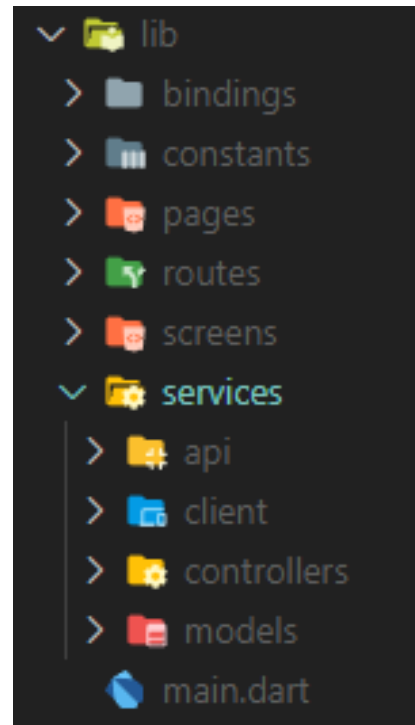
Egy olyan adatbázist választottam, amit a flutter fejlesztők szeretnek használni, ajánlani ez pedig nem más, mint a Hive. A következő dolgokban tér el egy szokványos sql-től, ez egy nosql adatbázis, tehát nincsenek relációk, legegyszerűbben úgy lehet elképzelni, mint egy szótárat, van egy kulcs és ahhoz tartozik az érték. Mivel az alkalmazás nem tárolja a tanuló adatait, hanem mindig, ha aktuális egy szerver hívással tölti a memóriába, ezért nincs szükség annak tárolására, így csak a felhasználó neve, és a jelszavát kellett eltárolnom, így tökéletesen megfelel a célnak a Hive. Előnyei közül itt is a teljesítményt és az egyszerűséget emelném ki.

## Projekt mappa szerkezete

A lib nevezetű mappa tartalmazza a program futásához szükséges összes kódot

Mappák ismertetése:

- bindings – Ez a mappa a state menedzseléshez tartozó kontrollereket tartalmazza, és ezeket az alkalmazás indításakor „inject” -eli a globális scope-ba.
- constants – A mappa kibontva több az alkalmazásban található állandó / konstans színeit, icon elérési útjait és felhasználó felület részeit (pl: betűtípusát).
- pages – A navigálható oldalak UI kódját tartalmazza, ezen felül minden oldalnak külön „widget”-jeit, esetleges adat modelljét.
- routes – Két program fájlt tartalmaz, az egyik a oldal url konstansokat, a másik fájl pedig a GetX route menedzsmentjét kezeli az alkalmazáson belül.
- screens – Azokat a UI kódokat tartalmazza, ami befedi az egész kijelzőt, ezért „képernyő” (screen), ebben a mappán összesen 3 darab screen található (error, login, main).
- services – Ez a mappa tartalmazza az úgy nevezett „bloc”-okat, azaz business logic component ez azt jelenti, hogy minden komponens, ami a logikát kezeli az appon belül.
  - api - Mappában található minden olyan kód, ami a szerverhez való kéréseket intézi (kód implementálási kommenteket az \*\_interface.dart fájlokban találhatóak).
  - client - Ebben a mappában az a kód található, ami a szervertől kapott adatokat modellekbe rendezi.
  - controllers - Itt található minden controller, ami interakcióba lép a felhasználó felülettel, például a controller, ami a bejelentkezésért felelős „login\_controller.dart”.
  - models – A szervertől visszakapott adatot (JSON), strukturált adat formátumban tárolja.



Végül egy main.dart fájl található, mely az első fájl, ami a program futtatásakor indul el. Ezzel kezdődik az alkalmazás elindulása, de erről később részletesebben.

## Statisztika

Ez a projekt a lib mappán belül jelen állapotában 47 darab .dart végződésű fájl található, amikben összesen 2687 sor program kódot írtam.

## Bevezetés a program kódba

A programban kizárólag angol nyelvet alkalmaztam, ez azt jelenti, hogy a változók, a függvények, metódusok, osztályok és a programban lévő kommentek mind angol nyelven íródtak, tehát a kód olvasásához elengedhetetlen az angol nyelv ismerete.

A kódot igyekeztem kommentálni, bár szerintem, az elnevezéseim magától értetődőek, így elég csak a metódus nevét elolvasni és már érthető is, hogy mi történik benne. A kommentek nagyrészt a „bloc” részletben vannak, a felhasználói felületében elvéve találhatóak.

A kód írása közben használtam a „best practices” -eket. Esetenként a S.O.L.I.D és a DRY elveket követtem. Végül úgy gondolom, hogy a kód rendezettsége, olvashatósága és érthetősége nem hagy kifogást maga után.

Dart egy objektumorientált programozási nyelv (OOP). Ez egy fejlesztés közben alkalmazott módszertan, amely az adatokat és az azon végrehajtandó műveleteket egységes egésszé, úgynevezett objektumokká fogja össze. Az objektum az osztály (class) kulcsszó alatt valósul meg. Az én programomban is ezt a módszertant helyeztem előtérbe.

Mint a statisztika bekezdésnél olvasható 47 darab fájl jött létre a program írása közben, így csak az igazán lényeges kódra fogok kitérni, az importálásokra nem, mivel azok általában kommentelve vannak.

## A fő fájl

```
class App extends StatelessWidget {  
  const App({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    final box = Hive.box('MainBox');  
    final String? user = box.get('username');  
  
    return GetMaterialApp(  
      title: 'Mini Napló',  
      debugShowCheckedModeBanner: false,  
      initialBinding: InitialBinding(),  
      initialRoute: user == null ? Routes.LOGIN : Routes.NAVIGATOR,  
      getPages: AppPages.routes,  
    );  
  }  
}
```

*lib/main.dart*

A lényeges rész az App osztályban történik a build metodusnál, először is a Hive lokális adattárolóból box-ból, kér egy hivatkozást a 'MainBox' kulcsú adattárolóról, aki még nem dolgozott hasonló adattárolóval, a box egy úgynevezett tábla, ha sql-ben gondolkoznánk, ebben a box-ban (doboz) tárolódik a felhasználónév és jelszava a felhasználónak, ezután kikéri a 'username' kulcsú adatot a box-ból.

Végül visszaadd egy GetMaterialApp widget-et ami az app alap paramétereit állítja be, mint például az app elnevezését, bindings-okat, ami mint említettem a kontrollereket illeszti a globális scope ba, valamint a getPages paraméternél kapja meg milyen oldalakra navigálhat az app, és végül az initialRoute ami azt takarja, hogy futáskor melyik route-ot (útvonalat) keresse, ami végül elnavigál az adott képernyőhöz. Itt viszont nincs konstans útvonal, hanem attól függően, hogy ha az adatbázisban nem található felhasználó akkor a LOGIN, azaz a bejelentkezés képernyőre vigyen, de ha már van felhasználó akkor a NAVIGATOR azaz a navigáló oldalra visz, ami felel, hogy éppen melyik oldalt mutassa az alsó navigációs menü segítségével. A vizsgálatot nem egy hagyományos if else szerkezettel végeztem, hanem egy rövidebb úgynevezett ternary operátorral, ami röviden kell egy feltétel, ami ebben az esetben a felhasználó létezése, ezután jöhet a ? operátor, amit a feltétel igaz ága, és végül : után a hamis ága. Úgy gondolom, hogy fontos megemlíteni és érthetően magyarázni az operátor működését, mivel a program kód során gyakran alkalmazott, és megjelenő elem.

A kód bemutatása során, azzal a folyamattal szeretném bemutatni ahogyan egy új felhasználó találkozik a felhasználó felülettel. Így a bejelentkezés képernyő következik, csak a login\_screen.dart fájl magában 224 sor kódot tartalmaz, és ez csak az oldal kinézetéért felelős, ehhez még tartozik egy controller, ami a logikáért felelős ez a services/controllers mappán belül a login\_controller.dart fájl. A felhasználó interakcióba tud lépni a szöveg dobozokkal, a gombbal és a visibility ikonnal. Ezek mögött a következő kód áll. Hogy dolgozni tudjunk a flutterben lévő szöveg dobozokkal, létre kell hozni egy TextEditingControllert szöveg dobozonként.

```
late final TextEditingController usernameController;
late final TextEditingController passwordController;

@override
void onInit() {
  super.onInit();
  usernameController = TextEditingController();
  passwordController = TextEditingController();
}

@override
void onClose() {
  usernameController.dispose();
  passwordController.dispose();
  super.onClose();
}
```

*[lib/services/controllers/login\\_controller.dart](#)*

Ez a kód részlet a controller fájlban található, itt először a változók deklarálása történik, viszont megfigyelhető, hogy nem kapnak értéket, ezért a változók előtt egy „late” kulcs szó található, ez azt mondja meg hogy a változó értéke később lesz deklarálva, utána „final” ami a Dart ban olyan mint a const JavaScript-ben azaz csak egyszer lehet értéket adni az adott változónak, ezt egy type hint követ ami azt mondja meg hogy milyen típusú értéket fog felvenni a változó, ez igazából opcionális, de egy „good practise” nek mondható mivel pontosan megmondja mit tartalmaz az adott változó, ezáltal könnyebben hiba kezelhető, mivel ebben az esetben később lesz megadva az értéke, illetve az IDE-t is segíti hogy milyen metódusokat használhatsz az adott változóra és végül a változó neve. A változóknak azért nem adatom meg az értéket alaptól, mert arra csak akkor lenne szükség ha egyből valamit megjelenítenénk a szövegdobozban, de itt erre nincsen szükség, ehelyett csak akkor adok értéket amikor a controller létre jön, azaz az onInit() metódusban.

Az onClose() metódus szerepe, hogy amint elnavigálunk a bejelentkezés képernyőről, akkor a controller megszűnik, mivel kitörlik a memóriából, hogy ne foglaljon feleslegesen helyet, jobb teljesítmény érdekében, a controller kitörlése dinamikusan történik GetX vezérlésével, a alapértelmezett onClose() metódussal, de mivel hogy a loginController osztály örököli a GetxController ős osztályt így egy @override dekorátorral felül tudjuk írni ezt a metódust amivel megmondjuk hogy törlésnél a kontrollereket is törölje a memóriából, a teljesítmény érdekében.

## A Bejelentkezés gomb mögött álló logikát kezelő metódus

```

void signUserIn() async {
  if (!Get.find<NetworkService>().hasConnection) {
    buttonText("Nincsen internet!");
    return;
  }

  final String username = usernameController.text;
  final String password = passwordController.text;

  // assert problems && validate input
  if (!_studentIsValid(username, password)) return;

  final apiService = Get.find<ApiService>();
  final loginSuccess = await apiService.createUser(
    username: username,
    password: password,
    institute: cv.instituteCode,
  );

  if (!loginSuccess) {
    buttonText("Hibás Jelszó, vagy Felhasználónév");
    return;
  }

  buttonText("Sikeres bejelentkezés");
  await apiService.initData();
  await _addStudentToDb(username, password, apiService.bearerAsMap);

  Get.parameters['relogin'] == null ? Get.offNamed(Routes.NAVIGATOR) : Get.back();
}

```

*lib/services/controllers/login\_controller.dart*

A `signUserIn()` metódus egy aszinkron (`async`) metódus, erre azért van szükség mivel történik szerverrel való kommunikáció ezért fontos hogy „`await`” -elni tudjuk, azaz megvárni amíg teljesen vissza tér a válasz. A metódus „early return”-el dolgozik, ez azért fontos hogy ne legyen sok egymásba ágyazás, ezért szimplán csak térjen vissza ha valami nem megfelelő, ezáltal olvashatóbbá téve a kódot. Már a metódus elején található is egy ilyen „early return”, ami azért felelős, hogy megnézi, hogy van-e internet kapcsolat, `Get.find<NetworkService>()` a globális scope-ban megkeresi a service-t és vissza adja az osztály példányt, Ez a service azért felelős hogy folyamatosan vizsgálja az internet állapotát, ezzel dolgozik az internet hiánya indikátor is. Ezen az osztályon meghívódik a `.hasConnection` „getter” lekérdező metódus, ennek értéke logikai (bool) azaz igaz vagy hamis lehet, igaz ha van internet, ellenkezőképpen hamis. Tehát ha az értéke hamis azaz nincsen internet, akkor a `!` – bang nevezetű operátorral megfordítom az értéket, tehát igazzá válik ezáltal lefut az `if` és át írja a gomb szövegét amit a felhasználó lát, majd vissza tér. Ha van internet akkor két változóban eltárolom a szövegdoboz szövegét, amit egy privát validáló metódusnak adok paraméterben, ez a metódus az üres dobozokra szűr, ha mind a két darab doboz tartalmaz szöveget akkor nincs probléma és nem tér vissza ennél az elágazásnál a metódus.

Mivel, hogy van szöveg ezért létrehozok egy User osztályt az ApiService en belül segítségével, ami a createUser() metódussal történik

```
Future<bool> createUser({  
  required String username,  
  required String password,  
  required String institute,  
}) async {  
  _user(User(  
    user: username,  
    password: password,  
    institute: institute,  
  ));  
  await user.init();  
  
  return await user.login();  
}
```

*lib/services/controllers/api\_service.dart*

Ez a képen látható. a metódus fogad három paraméter, a required szó azt mondja, hogy ezeket muszáj megadni, készít egy User példányt a biztosított paraméterekkel, majd egy reaktív privát változóba rakja (\_user()) hogy később is elérhető legyen ez az osztály példány. Ezután inicializálódik a user osztály, ami azért felelős, hogy előállítsa a szerver felé történő lekérdezésekhez szükséges header-eket, És végül visszatér a beléptetési metódus eredményével, ami igaz vagy hamis lehet attól függően, hogy sikeres a bejelentkezés vagy sem.



## login metódus implementációja a KretaAPI ős osztályon belül

```
@override
Future<bool> login() async {
  final response = await http.post(
    Uri.parse(Kreta.IDP + KretaEndpoints.token),
    headers: headers,
    body: FetchHelper.loginBody(user, password, institute),
  );

  final res = jsonDecode(response.body);
  if (!res.containsKey("access_token") && res["access_token"] == null) {
    return false;
  }

  bearer = Bearer.fromMap(res);
  headers["Authorization"] = bearer.accessToken;
  reqHeaders = FetchHelper.reqHeaders(bearer.accessToken);

  return true;
}
```

*lib/services/api/kreta\_api.dart*

Röviden a működéséről, a megadott adatokkal egy http kérést intéz a kréta végpontjához (endpoint), és ha a visszatért válaszban található „access\_token” és nem „null” értéket vesz fel akkor létezik ilyen felhasználó és sikeres belépés történt, ha nem sikeres a metódus egy hamis logikai válasszal visszatér, de ha sikeres akkor készít egy Bearer példányt a visszaadott JSON-ből, ez azért fontos mivel ennek segítségével kérhetőek le az adott felhasználói adatai (pl.: jegyei) a későbbi szerver kéréseknél, ezután még elkészíti a későbbi lekérdezéseknél majd használandó header-eket, ha minden sikeres egy igaz értékkel vissza tér a függvény.

```

final loginSuccess = await apiService.createUser(
  username: username,
  password: password,
  institute: cv.instituteCode,
);

if (!loginSuccess) {
  buttonText("Hibás Jelszó, vagy Felhasználónév");
  return;
}

buttonText("Sikeres bejelentkezés");
await apiService.initData();
await _addStudentToDb(username, password, apiService.bearerAsMap);

Get.parameters['relogin'] == null ? Get.offNamed(Routes.NAVIGATOR) : Get.back();

```

*lib/services/controllers/login\_controller.dart*

Vissza térve a `signUserIn()` metódushoz egy gyors validálás után. ha sikeres a bejelentkezés tudatja a felhasználóval a gomb szövegének kicserélésével. Majd az adatok lekérdezése történik, ezt követi egy privát metódus, aminek a feladata, hogy elmentse a felhasználó adatait a Hive lokális adatbázisba. És végül elnavigál a főoldalra, ha első belépés, ha csak felhasználó váltás akkor csak szimplán visszadob.

#### Adat mentéséért felelős metódus

```

Future<void> _addStudentToDb(String usr, String pwd, Map brr) async {
  await mainBox.put("username", usr);
  await mainBox.put("password", pwd);
  await mainBox.put("bearer", brr);
}

```

*lib/services/controllers/login\_controller.dart*

```

class MainFrame extends StatelessWidget {
  MainFrame({super.key});

  final frameController = Get.find<FrameController>();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      extendBody: true,
      resizeToAvoidBottomInset: false,
      body: Obx(
        () => PageTransitionSwitcher(
          transitionBuilder: (child, animation, secondaryAnimation) {
            return FadeThroughTransition(
              fillColor: constBgColor,
              animation: animation,
              secondaryAnimation: secondaryAnimation,
              child: child,
            );
          },
          duration: const Duration(milliseconds: 300),
          child: AppPages.getPage(frameController.currentPageRoute),
        ),
      ),
      bottomNavigationBar: CustomNavbar(),
    );
  }
}

```

*lib/screens/main\_screen/main\_screen.dart*

Ez a MainFrame Widget (Routes.NAVIGATOR) felelős azért, hogy éppen melyik oldalt mutassa a felhasználónak amikor az alsó navigációs menü segítségével navigál.

Működéséről röviden: Egy PageTransitionSwitcher Widget felelős az éppen aktuális oldal lekezeléséért, ebben beállítható a navigálás közben lévő animáció, és annak specifikációi. Ez egy Obx() Widget-be van csomagolva, ez azért fontos mivel, ha a felhasználó másik oldalra szeretne menni a CustomNavbar() widget frissíti a frameController-ben lévő reaktív változót, és amint ez változik az Obx() Widget egy úgynevezett observer widget ami figyeli, ha a reaktív változó értéke változik-e és ha igen akkor frissíti a felhasználó felületet, és azt az oldalt jeleníti meg a felhasználónak amire kattintott a navigációs menün.

## Adatok lekérdezése

Ebben a folyamatban azt mutatom be, hogy mi történik a szervertől való kéréstől, addig, hogy a felhasználónak megjelenítőjének az adatok.

```
abstract class KretaAPI implements IKretaAPI {  
  String? user;  
  String? password;  
  String? institute;  
  late Bearer bearer;  
  late Map<String, String> headers;  
  late Map<String, String> reqHeaders;  
  
  KretaAPI({  
    this.user,  
    this.password,  
    this.institute,  
  });  
}
```

*lib/services/api/kreta\_api.dart*

A KretaAPI absztrakt osztály és paraméterei, illetve konstruktorja. Ez az ősz osztály implementálja a IKretaAPI interfészt. Az interfész megadja, hogy tulajdonképpen milyen metódusokat kell implementálnia egy adott osztálynak, további előnyei: Kód újrahasznosítás, Tesztelhetőség és Absztrakció.

Ennek az osztálynak a célja, hogy implementálja a kréta szerverrel való kommunikációt, azokra a végpontokra amire az alkalmazásnak szüksége van. Kommentekért, az IKretaAPI fájlt kell nézni (lib/services/api/kreta\_api\_interface.dart).

Példa képpen a tanulónak hiányzási adatainak lekérdezést mutatom be:

```
@override
Future<List> fetchAbsences() async {
  final response = await http.get(
    Uri.parse(Kreta.base(institute!) + KretaEndpoints.absences),
    headers: reqHeaders,
  );
  return jsonDecode(response.body);
}
```

*lib/services/api/kreta\_api.dart*

Egy http kérést intéz az alap url + absences (hiányzások) össze fűzött végponthoz, és a JSON adatot dekódolja majd a listát majd visszaadja. Egész pontosan a User osztálynak.

```
class User extends KretaAPI implements IUser {
  User({super.user, super.password, super.institute});
```

*lib/services/client/user.dart*

Mivel a User osztály örököli (inheritálja) a KretaAPI ős osztályt és így a User osztály kezeli a szerver kéréseket.

```
@override
Future<Absences> getAbsences() async {
  return Absences.fromAPI(await fetchAbsences());
}
```

*lib/services/client/user.dart*

User osztályban a getAbsences() metódus használja a fel a KretaAPI osztályban implementált fetchAbsences() metódusát. A getAbsences() metódus azért felelős, hogy strukturált adattá alakítsa át a szerver választ, majd visszaadja az ApiService-nek, ami reaktív változóban eltárolja a strukturált adatot és megjelenítse a felhasználó felületnek.

## Kulcsszavak magyarázata

**Widget** – Úgy lehet legegyszerűbben elképzelni, mint html-ben használt tag-eket, tehát a widget a felhasználó felület építőelemei más képességekkel/tulajdonsággal bír, és ezek egymásba ágyazódva hozzák létre a Fa hierarchiát.

**Privát** – Minden, ami előtt van egy aláhúzás jel: `_` privát lesz, ha például egy osztályon belül létre hozunk egy metódust aláhúzással a neve előtt ez azt jelenti, hogy azt a metódus csak az osztályon belül elérhető, így az osztályon kívülről nem lehet elérni.

**State** - A state (állapot) a Flutter keretrendszerben az alkalmazás egy részének aktuális állapotát jelenti, lehet widget state vagy app state. A widget state-t Stateful widgetekkel lehet kezelni, míg az app state kezeléséhez más megoldásokra van szükség, például a Provider vagy az én esetemben GetX könyvtár használatára. A state kezelése fontos a Flutter fejlesztés során, mivel lehetővé teszi az alkalmazás általános működésének irányítását és az alkalmazás állapotának frissítését.

**Reaktív** – A GetX segítségével lehetőség van a reaktív state menedzselésre, ez a következő képpen implementálható, létre hozunk egy változót, egy GetxController-t öröklő osztályon belül, és dekoráljuk egy `.obs` végződéssel és ezzel már Observable – Reaktívá alakítódik.

```
class LoginController extends GetxController {
  // ui variables
  final buttonText = "Bejelentkezés".obs;
  final isObscure = true.obs;
```

Mint a képen is látható a `buttonText` változó felel a bejelentkezés képernyőn található gomb szövegéért, Az `isObscure` változó a jelszó láthatóságáért. Mivel ezek a változók már dekorálva vannak `.obs`, el ezért mindig, amikor a változó értéke változik GetX értesíti a `Obx()` widgetet amiben az adott reaktív változó helyet foglal. Például a gomb:

```
child: Obx(
  () => TextButton(
    onPressed: loginController.signUserIn,
    child: Text(
      loginController.buttonText.value,
      style: const TextStyle(
        color: constFontColor,
        fontFamily: constFontFamily,
        fontSize: 18,
      ),
    ),
  ),
),
```

A reaktív state menedzselésről külön oldalakat lehetne írni komplexitása valós mély működését elemezve, de röviden így tudom összefoglalni annak, aki még soha nem hallott az adott topikról.

**Service** – A service olyan, mint egy kontroller, viszont ez nem törlődik a memóriából, így a háttérben mindig elérhető és futó kód, mint a NetworkService ami a hálózatot figyeli, és megjeleníti az indikátort, és letilt egyes funkcionálisit, mint például az adatok frissítését.

**Getter** – Osztály tulajdonság vissza adó, esetekben azért van szükség, ha privát változót szeretnénk, hogy máshonnan is elérjünk a programból.

**Inicializálódik** – Létre jön.

**Best practise** – Az olyan bevált és hatékony módszerek, eljárások és megoldások összessége, amelyeket a szakmai közösség elfogadott és javasolt a problémák megoldására és az eredményes munkavégzésre. Ezek az elvek és szabályok megkönnyítik a programozók, fejlesztők és tervezők munkáját, biztosítják a kód minőségét, biztonságát és hatékonyságát.

**Header** – A http kéréseknél fontos adatokat tartalmaznak, ami a szerver felé továbbítódik. Például egy kulcsot küldünk header-ben a szervernek, a szerver megkapja megvizsgálja, és ha a kulcs megfelel a szervernek visszaküldi az adatot.

## Programhoz használt könyvtárak

```
# animated ui libs
rive: ^0.10.2
flutter_svg: ^2.0.1
animations: ^2.0.7
# for charts
syncfusion_flutter_charts: ^20.4.51
# for api and encoding
http: ^0.13.3
crypto: ^3.0.2
intl: ^0.18.0
# database
hive_flutter:
hive:
# state / route manager
get: ^4.6.5
# internet status
connectivity_plus: ^3.0.3
# refresh pull
liquid_pull_to_refresh: ^3.0.1
```

*pubspec.yaml*

### Fontosabb könyvtárak:

- rive – az oldalak mögötti animációért felelős
- syncfusion\_flutter\_charts – értékelés oldalon használt diagrammban használt
- http – szerver kéréseket teszi lehetővé
- hive – maga az adatbázis
- get – state / az appban való navigálást segíti
- connectivity\_plus – a NetworkService alap építő eleme, segítségével monitorozható az internet kapcsolat



## Fejlesztői környezet

### Fejlesztéshez használt számítógép:

#### Processzor – CPU

- Intel(R) Core(TM) i5-9400F CPU @ 2.90GHz

#### Videókártya – GPU

- NVIDIA GeForce GTX 1070

#### Memória – RAM

- G.SKILL 16 GB DDR4 3200 MHz

### Fejlesztéshez használt szoftverek:

Operációs rendszer: Microsoft Windows 10 HOME

Kód szerkesztők (IDE):

- Visual Studio Code (v1.76.2)
- Android Studio (Emulator miatt)

Google Chrome - Verzió: 112.0.5615.121 (Hivatalos verzió) (64 bites)

Flutter 3.7.7

Dart 2.19.4

### Fejlesztéshez / Teszteléshez használt telefonok:

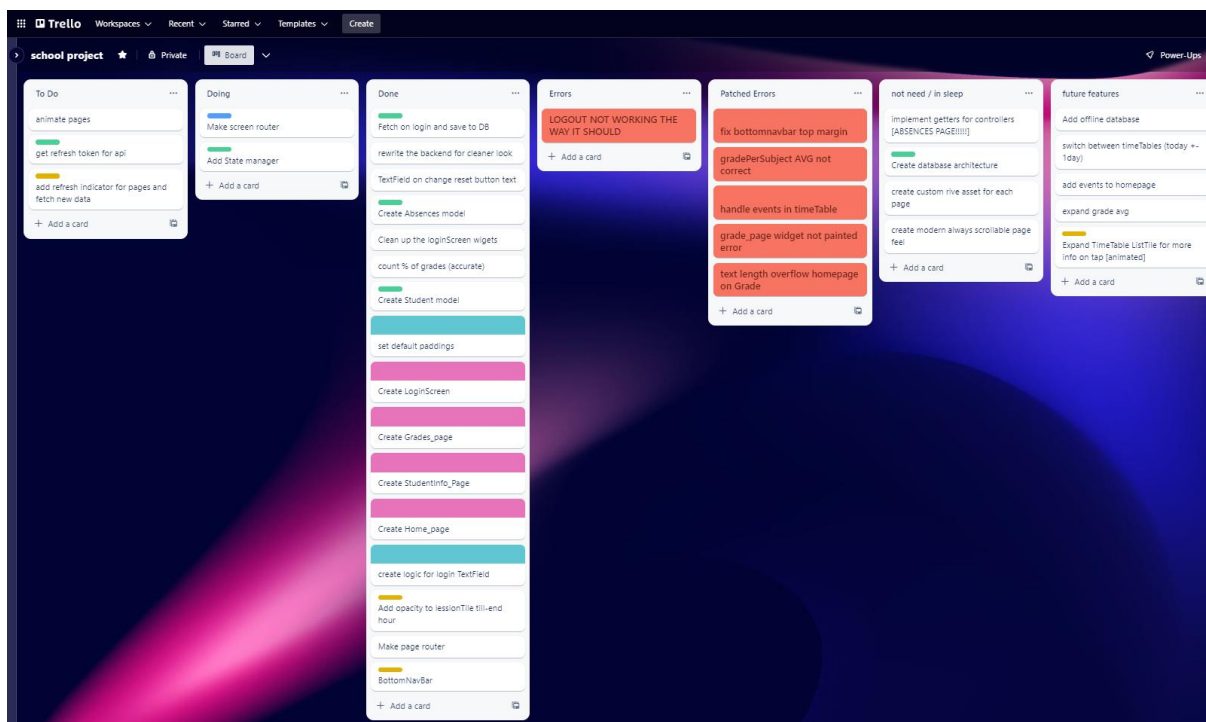
Google Pixel 6 Pro Emulator – Android 11 (API 30)

OnePlus 9 Pro – Android 13 (OxygenOS 13.0)

Redmi Note 8T – Android 11 (MIUI 12.5.5)

## Használt fejlesztési modell

Fejlesztésem során többféle fejlesztési módszertant alkalmaztam, az alkalmazás állapotától függően, alkalmaztam egy aktuális modellt. Az alkalmazás elején teljesen más ötlettel álltam elő, ami csak részben használta volna a kréta szervereit, mint hogy egy új e-naplót készítek, de rá kellett jönnöm, hogy az alap ötlet a projekt kezdés elején lévő tudásommal nem volt megvalósítható, és sokkal több időre lett volna szükség. Tehát a program elején amikor még nem pontosan tudtam, hogy összesen miket is fog tudni az alkalmazás, a Prototípus modellt alkalmaztam, tehát mindig építettem egy alap programot, és már rendes telefonra kis is buildeltem, ezt egészen a bejelentkezési képernyő, és az órarend oldal elkészüléséig alkalmaztam. De mivel nem voltam megelégedve a kinézettel, új projektet készítem, és újra gondoltam az alkalmazás kinézetét. Ekkor döntöttem el, hogy egy modernebb kinézetű e-naplót szeretnék csinálni, és ekkor tisztult ki bennem a teljes kép, mint például a kinézet, elrendezés, oldalak. Összeraktam egy program tervet egy weboldal segítségével.



Ezután megkezdődött a vízesés modellel való fejlesztés, így az alkalmazás fejlesztést biztonságosabbá téve, mivel egyszerre csak egy dolog implementálására és tesztelésére fókuszáltam, így nem kellett visszatérnem régebbi részekhez, hanem innentől már csak folyamatosan egymásra építhettem az elemeket, ezzel elérkezve a végső alkalmazásig.

## További fejlesztési lehetőségek

Mint az észrevehető a bejelentkezés képernyőn nem található iskola kiválasztásra opció, mivel ezt az alkalmazást alpból a [Vak Bottyán János Katolikus Műszaki és Közgazdasági Technikum, Gimnázium és Kollégium](#)ra optimalizálódott, azon belül is a 13.B informatikai technikumra, szóval így csak az iskola diákjai használhatják az alkalmazást. A jövőben tervezem, hogy teljesen befejezem úgy, hogy a kréta összes funkcióját implementálom (újrarendeléssel), és minden iskolában működőképes legyen. Jelenleg az alkalmazás működéséhez internet kapcsolat elengedhetetlen, mivel minden adatot jelenleg csak a memóriában tárol az alkalmazás, a bejelentkezési adatokon kívül, későbbi verziókban tervezem, hogy egy offline adatbázisba eltárolja minden adatot, ezáltal lehetővé teszi az internet kapcsolat nélküli működést is.

Következő funkciók, amiket az alkalmazásba tervezek:

- Órarend csempe kattintásra való lenyílás több információval
- Főoldalon minden esemény megjelenítése
- Órarendben a napok közötti váltogatás
- Jegyek tantárgyanként
- Többféle diagram
- Minden oldalon adat frissítés lehúzásra

## Felhasznált Irodalom

<https://github.com/bczsalba/ekreta-docs-v3>

<https://blog.logrocket.com/ultimate-guide-getx-state-management-flutter/>

<https://github.com/jonataslaw/getx>

<https://stackoverflow.com/>

<https://docs.flutter.dev/>

<https://dart.dev/>

<https://dribbble.com/search/modern-mobile-ui>

## Felhasznált eszközök

<https://pub.dev/>

<https://trello.com/>

<https://3dicons.co/>

<https://rive.app/community/1298-2487-animated-icon-set-1-color/>

<https://www.midjourney.com/>

<https://openai.com/product/dall-e-2>

# Plágium nyilatkozat

## Nyilatkozat a záródolgozat készítésére vonatkozó szabályok betartásáról

Alulírott: Silkó Levente jelen nyilatkozat aláírásával kijelentem, hogy a

### Mini Napló

című záródolgozat önálló munkám, a záródolgozat készítése során betartottam a szerzői jogról szóló 1999. évi LXXVI. törvény vonatkozó rendelkezéseit, valamint az intézmény által előírt, a záródolgozat készítésére vonatkozó szabályokat, különös tekintettel a hivatkozások és idézetek tekintetében.

Jelen nyilatkozat aláírásával tudomásul veszem, hogy amennyiben bizonyítható, hogy a záródolgozatot nem magam készítettem, vagy a záródolgozattal kapcsolatban szerzői jogsértés ténye merül fel, az intézmény megtagadja a záródolgozat befogadását és ellenem fegyelmi eljárást indíthat.

A záródolgozat befogadásának megtagadása és a fegyelmi eljárás indítása nem érinti a szerzői jogsértés miatti egyéb (polgári jogi, szabálysértési jogi, büntetőjogi) jogkövetkezményeket.

Gyöngyös, 2023.04.17.

vizsgázó aláírása