UNIVERSITATEA „SAPIENTIA” DIN CLUJ-NAPOCA

FACULTATEA DE ŞTIINŢE TEHNICE ŞI UMANISTE,

TÎRGU-MUREŞ

SPECIALIZAREA CALCULATOARE

PointerApp

PROIECT DE DIPLOMĂ

Coordonator ştiinţific:

Ş.l.dr.ing. Szántó Zoltán

Absolvent:

Szász Nimród János

2023

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA **Viza facultăţii:**
Facultatea de Ştiinţe Tehnice şi Umaniste din Târgu Mureş
Specializarea: **Calculatoare**

LUCRARE DE DIPLOMĂ

Coordonator ştiinţific:	Candidat: S**zász Nimród János**
ş.l. dr. ing. Szántó Zoltán	Anul absolvirii: **2023**

a) Tema lucrării de licenţă:
PointerApp

b) Problemele principale tratate:
- Studiu bibliografic asupra metodelor de prezentare
- Dezvoltarea aplicatiilor mobile, care permit extragerea datelor din senzorii dispozitivului
- Studierea metodelor alternative prin care o prezentare Powerpoint poate fi controlata
- Communicare wireless intre telefon si PC

c) Desene obligatorii:
- Schema bloc al aplicației
- Diagrame UML privind software-ul realizat.

d) Softuri obligatorii:
- Applicatie pt extragerea datelor de la sensorii (ex. accelerometru) telefonului mobil realizat in Flutter
- Proiectarea si Implementarea protocolului de communicare telefon-PC
- Aplicatie desktop dezvoltat in python care acepta datele de la telefon, si transmite comenziile pt. Powerpoint

e) Bibliografia recomandată:

- Tkachuk, V., Yechkalo, Y., Semerikov, S., Kislova, M., & Khotskina, V. (2020). Exploring student uses of mobile technologies in university classrooms: Audience response systems and development of multimedia. CEUR Workshop Proceedings.
- Budiman, A., & Triono, J. (2016). Smart and Simple Classroom Presentation Tools. International Journal of Advanced Research in Computer Science, 7(7).
- Windmill, E. (2020). Flutter in action. Simon and Schuster.
- Zammetti, F. (2019). Practical Flutter. Berkeley, CA: Apress.

f) Termene obligatorii de consultaţii: săptămânal
g) Locul şi durata practicii: Universitatea „Sapientia” din Cluj-Napoca,
Facultatea de Ştiinţe Tehnice şi Umaniste din Târgu Mureş

Primit tema la data de: 31.03.2022
Termen de predare: 28.06.2023

Semnătura Director Departament

Semnătura coordonatorului
Szántó Zoltán

Semnătura responsabilului programului de studiu

Semnătura candidatului
Szász

Declarație

Subsemnatul Szász Nimród János, absolvent al specializării Calculatoare, promoția 2023 cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Târgu Mureș,

Data: 27.06.2023 Szász Nimród János

Semnătura……

PointerApp

Extras

În prezent, toată lumea are un smartphone care poate rezolva multe probleme cu diverse aplicații. Astfel de aplicații utile sunt, de exemplu, aplicațiile pentru plata facturilor noastre, dar putem găsi și multe aplicații pentru gestionarea sarcinilor noastre zilnice.

Aplicațiile mobile nu numai că ne fac viața de zi cu zi mai ușoară, dar pot fi și utile în instituțiile educaționale. Cele mai utilizate aplicații mobile în școli și universități sunt diverse calculatoare, traductoare și aplicații de luare a notelor.

Cu toate acestea, aplicațiile pentru prezentarea prezentărilor PowerPoint nu sunt răspândite printre oameni, astfel încât numărul lor este foarte mic și nu sunt cele mai bine întreținute sau nu au multe caracteristici care ar face utilizarea acestor aplicații semnificativă în afară de faptul că utilizatorul nu trebuie să păstreze un dispozitiv separat cu ei.

Teza mea își propune să proiecteze un sistem care permite utilizarea pointerelor laser pe diapozitivele prezentării PowerPoint cu ajutorul smartphone-ului nostru indiferent dacă avem un telefon Android sau IOS. În plus, ar trebui să permită comutarea între diapozitive și desenarea pe acestea. Sistemul constă din două componente: o parte este serverul, care primește și procesează datele trimise de telefonul nostru; cealaltă parte este telefonul, care detectează mișcările mâinii utilizatorului cu ajutorul accelerometrului din telefon și trimite și apăsarea butoanelor afișate către server prin WiFi. Datele necesare pentru conectarea la server sunt posibile prin scanarea codului QR generat de server.

Cuvinte cheie: Aplicație mobilă, PowerPoint, pointer cu laser

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR
SZÁMÍTÁSTECHNIKA SZAK

PointerApp

DIPLOMADOLGOZAT

Témavezető:
Dr. Szántó Zoltán
egyetemi adjunktus

Végzős hallgató:
Szász Nimród János

2023

Kivonat

Napjainkban minden ember rendelkezik okostelefonnal, amelyek segítségével rengeteg problémát megoldhat különböző alkalmazásokkal. Ilyen hasznos alkalmazások például a számláink kifizetésére szolgáló alkalmazások, de akár feladataink vezetésére is rengeteg alkalmazást találunk.

A telefonos alkalmazások nem csak a mindennapjainkat könnyítik meg, hanem akár a tanügyi intézményekben is segítségünkre lehetnek. A legtöbb telefonos alkalmazás, amit előszeretettel használnak az iskolákban és az egyetemeken: a különböző számológépek, fordítók és a jegyzetelésre szolgáló alkalmazások.

Viszont a PowerPoint bemutatására szolgáló alkalmazások az emberek körében nincsenek elterjedve, ezért a számuk igen kevés és nem a legjobban karbantartottak vagy nem rendelkeznek sok lehetőséggel, amely értelmet adna ezen alkalmazások használatának azon kívül, hogy nem kell a felhasználó egy külön eszközt magánál tartson.

A dolgozatom célja egy olyan rendszer tervezése, amely segítségével PowerPoint bemutatók diáin lehessen lézer mutatót használni az okostelefonunk segítségével attól függetlenül, hogy Android-os vagy IOS-es telefonunk van. Ezen felül a diák közötti váltást, illetve ezekre történő rajzolást is lehetővé tegye. A rendszer két komponensből épül fel: az egyik része a szerver, amely fogadja, illetve feldolgozza a telefonunk által küldött adatokat; a másik része a telefon, amely a felhasználó kézmozdulatait érzékeli a telefonban található gyorsulásmérő segítségével, illetve a megjelenített gombok lenyomását is, amelyeket elküld a szervernek WiFi-n keresztül. A szerverhez történő kapcsolódáshoz szükséges adatokat a szerver által generált QR kód beolvasása teszi lehetővé.

Kulcsszavak: Mobil alkalmazás, PowerPoint, lézermutató.

Abstract

Nowadays, everyone has a smartphone that can solve many problems with various applications. Such useful applications are, for example, applications for paying our bills, but we can also find many applications for managing our day-to-day tasks.

Mobile applications not only make our everyday lives easier but can also be helpful in educational institutions. The most commonly used mobile applications in schools and universities are various calculators, translators, and note-taking applications.

However, applications for presenting PowerPoint presentations are not widespread among people, so their number is very small and they are not the best maintained or do not have many features that would make the use of these applications meaningful other than that the user does not have to keep a separate device with them.

My thesis aims to design a system that allows laser pointers to be used on PowerPoint presentation slides with the help of our smartphone regardless of whether we have an Android or IOS phone. In addition, it should allow switching between slides and drawing on them. The system consists of two components: one part is the server, which receives and processes data sent by our phone; the other part is the phone, which detects the user's hand movements with the help of the accelerometer in the phone and also sends the pressing of displayed buttons to the server via WiFi. The data required to connect to the server is made possible by scanning the QR code generated by the server.

Keywords: Mobile application, PowerPoint, laser pointer.

Tartalomjegyzék

1. Bevezető 11
1.1. Célkitűzések 12
2. Elméleti megalapozás és szakirodalmi tanulmány 13
2.1. Ismert hasonló alkalmazások 14
3. A rendszer specifikációi 16
3.1. Felhasználói követelmények 16
3.2. Rendszerkövetelmények 17
3.2.1. Funkcionális követelmények 17
3.2.2. Nem funkcionális követelmények 17
4. A rendszer architektúrája és tervezése 18
4.1. Kommunikációs protokoll 18
4.2. Számítógépes alkalmazás 21
4.2.1. Python 21
4.2.2. Számítógépes alkalmazás leírása 22
4.2.3. A felhasználói felület 24
4.2.4. WebSocket Szerver 27
4.2.5. Az érkező üzenetek feldolgozása 29
4.3. Telefonos alkalmazás 34
4.3.1. Flutter 34
4.3.2. Telefonos alkalmazás leírása 36
4.3.3. A telefonos alkalmazás felhasználói felülete 38
4.3.4. Telefonos alkalmazás navigációjának megvalósítása 40
4.3.5. QR kód beolvasása 41
4.3.6. Gyorsulásmérő szenzor 43
4.3.7. Mutató irányítása és a funkciók megvalósítása 45
5. Összefoglaló 50
5.1. Továbbfejlesztési lehetőségek 50
6. Irodalomjegyzék 51

Ábrák jegyzéke

1. ábra A rendszer használati eset diagramja 16
2. ábra A rendszer architektúra diagramja 18
3. ábra WebSocket protokoll működési diagramja 20
4. ábra Adatáramlás a rendszerben 23
5. ábra A számítógépes alkalmazás felhasználói felülete 24
6. ábra A számítógépes alkalmazás felhasználói felületének elemeinek létrehozása kódban 25
7. ábra A számítógépes GUI eseménykezelése 26
8. ábra QR Kód megjelenítése 26
9. ábra WebSocket létrehozása 27
10. ábra IP cím lekérése 28
11. ábra QR kód generálása 28
12. ábra Egy fogadott üzenet feldolgozásának a lépései 29
13. ábra A szerver által fogadott akciók kezelése 30
14. ábra PowerPoint gyorsbillentyű kombinációk 31
15. ábra A szerver pont típusú adat kezelése 32
16. ábra Mutató gyorsaságának állítása 32
17. ábra Mutató mozgatása 33
18. ábra A virtuális mutató határainak a szemléltetése 34
19. ábra A telefonos alkalmazás osztály diagramja 37
20. ábra Telefonos alkalmazás felhasználói felülete 38
21. ábra Telefonos alkalmazás témájának a beállítása 39
22. ábra QRView osztály 39
23. ábra Navigációs útvonalak 40
24. ábra Navigáció használata 41
25. ábra QR kód beolvasása 42
26. ábra Telefon elhelyezkedése a koordináta rendszeren 43
27. ábra Telefonra ható erő 43
28. ábra WebSocket szerverhez csatlakozás 45
29. ábra Virtuális mutató irányításához használt üzenet létrehozása 45
30. ábra Gyorsulásmérő adatainak lekérése 46
31. ábra Üzenetek küldése a telefonos alkalmazásból 47

32. ábra Rajzolás funkció meghívása 49

1. Bevezető

A mai világban az információ átadásának rengeteg módja létezik, mint például a telefonhívás, üzenetek, különféle videók, könyvek és prezentációk. A legelterjedtebb módja az információ élőben történő bemutatására az oktatási intézményekben talán még mindig a tábla használata, természetesen előszeretettel használják a prezentációt is. Az emberek azért szeretik a táblát használni, mert egy nagyon jó interakciós felületet biztosít. Ezt úgy kell érteni, hogy a táblára nagyon gyorsan fel lehet vázolni ötleteket, megoldásokat vagy ezeket javítani, törölni egy egyszerű kréta és szivacs segítségével. Az előnyök mellett persze megvannak a hátrányai is, mint például nagyon időigényes sok adatot felírni, emellett nem lehet ezeket a felírt adatokat később megosztani vagy elmenteni.

A prezentációk nagyban hasonlítanak a táblához abból a szempontból, hogy a prezentációk segítségével is információt szeretnénk átadni embereknek interaktívabb módon, mintha egy könyvből kéne megjegyezzék az információt. A prezentációk felépítés szempontjából is hasonlítanak a táblához, mert a prezentációk során is arra törekszünk, hogy egy átlátható, viszonylag csak a kulcsszavakat leírva inkább az előadónak segítve lehessen az információt egy érthető módon átadni az embereknek.

A hasonlóságok mellett viszont ennek a módszernek is vannak előnyei és hátrányai is. Egyik nagy előnye, hogy sokkal több lehetőséget biztosít az információ bemutatásához, mint például képek, ábrák, diagramok. Természetesen táblára is lehet rajzolni ábrákat, viszont már képeket vagy nagyobb diagramokat szinte lehetetlen csak tábla segítségével bemutatni. Egy másik előny, hogy nagyon könnyen lehet menteni a prezentációkat, hogy újra felhasználhatók legyenek. A pozitívumok mellet viszont ott vannak a negatívumok is. A negatívumok közé felsorolható az interaktivitás is, mert a prezentáció irányítása vagy az arra való rajzolás kevésbé intuitív, mint a táblán a krétával történő kiemelés vagy valaminek a gyors levezetése. Erre a problémára próbálnak megoldást nyújtani a prezentációs alkalmazások.

Mivel most már szinte mindenki rendelkezik okostelefonnal, amelyek segítségével rengeteg feladatot eltudnak végezni pár kattintással, még mindig vannak olyan területek, ahol a telefonunk használata nincs teljesen elterjedve. A legtöbb ember az élete során szembe kerül egy olyan helyzettel, ahol szüksége lesz egy bemutatót tartania prezentációk segítségével, viszont ezeknek a helyzeteknek a száma elég kevés és azoknak az embereknek a száma, akik lézer mutatót is használnak a prezentációjuk alatt, hogy kiemeljék a fontosabb pontokat a bemutatójuk során még kevesebb. Ennek több oka is lehet. Az egyik ok, hogy sokkal kényelmesebb a prezentáló

számára a számítógép egerének és billentyűzetének használata egy prezentáció során, mint, hogy egy teljesen új módszert megtanuljon a prezentációk bemutatásához, ez viszont avval a hátránnyal is jár, hogy egyhelyben kell bemutatnia a prezentációt, vagy vissza kell sétálgatnia a számítógéphez, hogy például a következő diára lépjen. Egy másik ok, hogy nem mindenki szeretne beruházni egy külső eszközbe, mert magánál kellene tartson egy plusz eszközt, amelyet a nap folyamán csak a prezentáció bemutatása során használna. A harmadik ok, hogy már létező applikációk nagy része nem támogat olyan funkciókat, mint például a rajzolás, amely értelmet adna az applikáció használatának. Ezeken kívül egy másik probléma, amely felléphet az ilyen típusú alkalmazások használata során, hogy például a Microsoft alkalmazása csak akkor működik, ha feltöltjük a felhőbe a bemutatásra szánt anyagot, másképp nem tudunk interakcióba lépni a számítógépen található bemutatókkal.

Ezért szerettünk volna egy olyan alternatívát kitalálni, amely egyszerű megoldást biztosít a PowerPoint prezentációkhoz való interakcióhoz olyan emberek számára, mint például a tanárok, akik szinte minden nap prezentációkat mutatnak be vagy az olyan emberek számára, akik szeretnének interaktívabban bemutatókat tartani.

1.1. Célkitűzések

A dolgozat legfőbb célja egy olyan rendszer tervezése, amelynek segítségével PowerPoint bemutatókat tudunk irányítani a telefonunkra telepített alkalmazással, illetve a számítógépre telepített alkalmazás segítségével.

A rendszerünknek a következő célokat fogalmaztuk meg:

- telefon által rögzített gesztusok értelmezése, feldolgozása a mutató irányításához;
- egy virtuális mutatót implementálni, amely egy lézer mutatónak felel meg;
- telefon segítségével haladni a diák között előre vagy hátra;
- megoldható legyen vonalakat rajzolni vagy különböző formákat, amelyekkel ki lehet emelni részeket a prezentációból;
- olyan módot keresni a két alkalmazás egymáshoz való csatlakoztatásához, amely csak a kívánt felhasználót engedi csatlakozni;
- a rendszer feladatainak az elvégzéséhez megfelelően gyors módszert keresni az adatok küldéséhez a számítógép és telefon között;
- megoldható legyen törölni az általunk rajzolt vonalakat egy radír funkció segítségével;
- hosszútávú cél a platformfüggetlenség, rövidtávú cél Windows 11 a számítógépes applikációnak és a telefonos applikációnak az Android;
- megoldható legyen az általunk végzett módosítások mentése, mint például a rajzok, amit a virtuális mutató segítségével rajzoltunk;

2. Elméleti megalapozás és szakirodalmi tanulmány

A szakirodalmi tanulmányi kutatás során sok dolgozattal találkoztam, amelyek a külső eszközöknek a hatékonyságát vizsgálják tanítási környezetben, vagy csak általánosan a prezentációk hatékonyságát, hogy mennyire segítik az információ átvitelét és annak megértését a tradicionális módszerekhez képest. Maga a prezentáció fogalma elég elterjedt a tanítási körökben, de ezen módszer használatának az előnyei vagy hátrányai már annál kevésbé.

A PowerPoint prezentációk fontosságát az is bizonyítja, hogy már 2005-ben több mint 400 millió felhasználó naponta több mint 30 millió prezentációt készített a szakirodalom szerint [1]. Ezért tartom fontosnak a prezentációk helyes használatának megértését, hogy belátást nyerjek abba, hogy pontosan milyen funkcionalitásokat érdemes a rendszerembe beültetni.

A prezentációkészítés legjobb módjáról viszont megoszló vélemények vannak. Valaki azon a véleményen van, hogy minél egyszerűbb kell legyen egy prezentáció, mert, így sokkal könnyebben átlátható a bemutatott téma. Mások szerint pedig, ha nagyon leegyszerűsítjük, akkor a kontextus hiánya miatt sok információ elveszhet [2]. Sokak szerint a prezentáció sikerességét nagyban befolyásolja, hogy milyen színeket használunk vagy érdemes színeket használni a diáink elkészítése során [1]. Egy dologban viszont egyetértenek a szakirodalomban is: érdemes prezentációkat használni az információ hatékonyabb átadásához [1,3].

A prezentációk sok téren hasznosnak bizonyultak, például idegen nyelvek tanulása során is arra jutottak, hogy a diákok könnyeben megértették az idegen szavakat, mivel a prezentációk segítségével a nehezebb üzeneteket könnyebben megérthető pontokba foglalva hamarabb meg tudták érteni [3].

Egy másik érv amiért sok helyen szeretnek PowerPoint prezentációkat használni, mert könnyebb az emberek figyelmét fenntartani, ha prezentációkat használunk, mintha csak szóban adnánk át az információt. Ezt azzal indokolják, hogy a prezentációkat dinamikusabbá lehet tenni különféle grafikai elemekkel is akár vagy képekkel és videókkal, amelyek szintén nagyban javítják a hallgatók figyelmének a fenntartását [3]. Az emberek figyelmének a fenntartását az is nagyban elősegíti, ha jobban interakcióba tudnak lépni a bemutatást tartó egyénnel. Ezt az bizonyítja, hogy a járvány időszakában készítettek felméréseket, amelyek során arra a következtetésre jutottak, hogy a diákok jobban szeretik, ha a tanár is szerepel a prezentáció diáin, mivel jobban meg tudják érteni az anyagot és a bemutatott anyagra is könnyebben tudnak koncentrálni [4].

A mobiltelefonok hasznosságát is fontos figyelembe vennünk, mivel ahogy a szakirodalomban is olvasható, [5] egyre jobban elterjedt a mobilos applikációk használata a tanítás

során. Ez annak köszönhető, hogy a mobiltelefon szinte mindenki számára elérhető. A nagy elérhetősége miatt rengeteg kutatást végeztek a téren. A legtöbb kutatás azt vizsgálta, hogy mivel lehetne a diákok figyelmét lekötni, illetve a diákok képességeit javítani. A kutatás során arra jutottak, hogy a mobilos applikációknak fontos szerepe lehet egy modern tanítási rendszerben, mert a használatuk kimutatható részben hozzájárult a diákok eredményeinek a javulásához [5]. A kutatások során [6] az is kiderül, hogy napjainkban azoknak a prezentációt segítő eszközöknek a többsége, amelyeket az emberek inkább használni szeretnek, vezeték nélküli eszközök. Ezek az eszközök azért terjedtek el, mert segítik a prezentációk folyékonyabb bemutatását, mivel az ember nincs hozzákötve a géphez és ez által a testbeszéddel is hozzá tud járulni az információ hatékonyabb átadásához.

A szakirodalmi kutatásom során olyan kutatásokat is kaptam [7], amelyek azt bizonyították, hogy mivel az emberi agy is két főbb csatornán fogadja az információt: verbális, illetve vizuális módon, ezért is hasznos lehet a prezentációk használata információ átadására, mivel a bemutató nem csak szóban, hanem ahogy már említettem akár fotókkal, ábrákkal vagy lézer mutató használatával is hozzá tud járulni az információ könnyebb átadásához.

2.1. Ismert hasonló alkalmazások

A PowerPoint-ot kisegítő alkalmazások száma elég nagy, viszont a legtöbb ismert hasonló alkalmazás nagy része nem oldja meg a célkitűzésekben felsorolt problémákat. Léteznek alkalmazások, amelyek szintén mobiltelefont használnak a PowerPoint prezentációk vezérléséhez, viszont vannak olyan alkalmazások is, amelyek más külső eszközöket használnak a prezentációk vezérléséhez.

A mobiltelefont használó megoldások sokkal elterjedtebbek: a legtöbb ilyen alkalmazás is a telefon szenzorjait felhasználva próbálja a diákat irányítani. A legtöbb ember számára ismert talán maga a Microsoft PowerPoint alkalmazása[1] a telefonokhoz, amely segítségével a telefonunkat használhatjuk lézer pointerként a bemutatóink során. Az alkalmazás nagy hátránya, hogy muszáj Microsoft felhasználói fiókkal rendelkeznünk a használatához, ezen felül csak olyan prezentációkat tudunk vezérelni, amely fel van töltve a felhőbe is. Léteznek más alkalmazások is, amelyek nem kérnek regisztrációt vagy nem szükséges a használatukhoz a felhőben tárolnunk a

[1] https://support.microsoft.com/en-us/office/present-a-slide-show-from-a-smartphone-and-use-a-laser-pointer-for-emphasis-485e8618-98ed-472b-84fa-443e23ea602a

prezentációinkat. Az ilyen típusú alkalmazások száma elég nagy, ahogy a következő cikkben [2] is olvasható [9], de a legtöbb ilyen alkalmazás nagy hátránya, hogy rég voltak frissítve, így újabb telefonokon problémákba ütközhetünk a használatuk során. Egy másik problémája az ilyen típusú alkalmazásoknak, hogy nem mindenki számára elég egyszerű a használatuk, mivel csatlakozniuk kell a számítógépükhöz, amely során lehet ip címeket kell majd beállítaniuk és ez a folyamat egy átlag felhasználó számára nehézkes is lehet.

Egy másik módszer, amelyet előszeretettel használnak az emberek az maga egy fizikai mutató. Ezek a mutatók általában Bluetooth segítségével kapcsolódnak a számítógépünkhöz, hogy virtuális lézer mutatóként szolgáljanak. Hátránya ennek a megoldásnak, hogy be kell ruháznunk egy külső eszközbe, amelyet csak erre a célra használhatunk.

A szakirodalmi tanulmányi kutatásom során találkoztam olyan eszközzel, amely kézre szerelhető [6]. Az eszköz a teljes alkar mozdulatait figyeli és ezeket a mozdulatokat alakítja át különböző funkciók végrehajtásához. A rendszer előnye, hogy az ember teljes keze szabad marad, így semmi nem akadályozza a prezentáló mozdulatait. Viszont a rendszer hátrányának tartom magát az eszközt, mert egy plusz beruházás a felhasználónak. Ezen felül sok lépéses a rendszer beüzemelése, amely sokat ronthat a felhasználói élményen. Egy másik problémája ennek a megoldásnak, hogy több szenzort is felhasználnak az irányításhoz, amely más embereknél akár kalibrálást is igényelhet, amely szintén megnehezítené a használatát. Evvel szemben a legtöbb telefon kis eltéréssel megegyező képességű szenzorokat használ, így márkától függetlenül a mért értékek egy telefonos rendszer esetén nagy valószínűséggel ugyanabba a tartományba esnének bele.

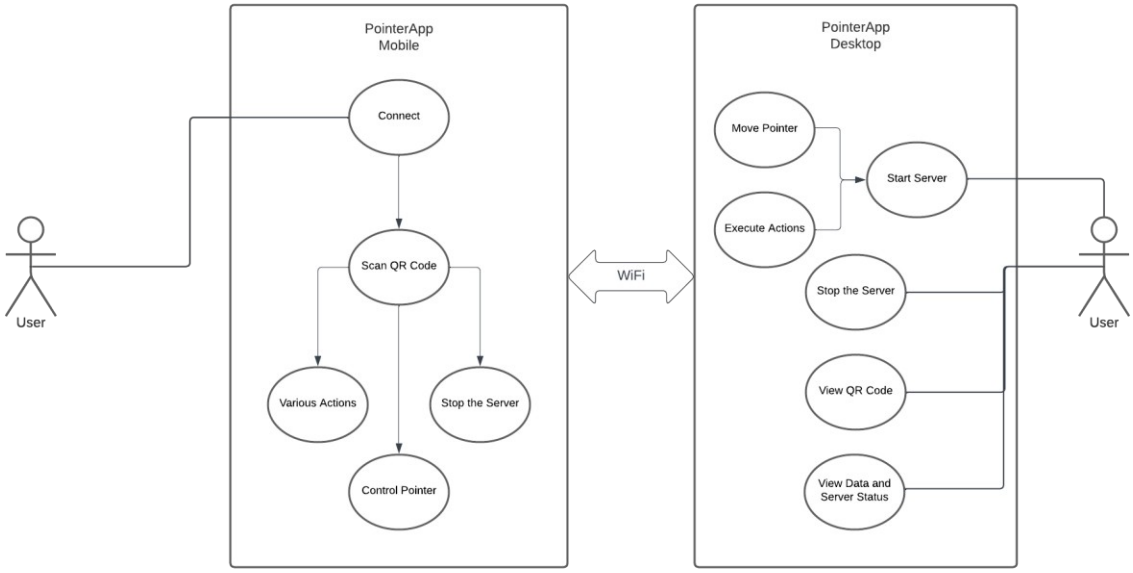
[2] https://en.softonic.com/top/remote-control-powerpoint-apps

3. A rendszer specifikációi

A rendszer specifikációja két nagy részre osztható fel: a telefonos alkalmazásra, illetve a számítógépes alkalmazásra. A telefonos alkalmazás segítségével tudja a felhasználó irányítani a prezentációt, míg a számítógépes alkalmazás felel a telefon által küldött parancsok feldolgozásában és végrehajtásában.

3.1. Felhasználói követelmények

Ahogy az 1. ábrán is látható a rendszernek egy típusú felhasználója van. Ez a felhasználó tudja kezelni mind a telefonos alkalmazást és a számítógépes alkalmazást is.



1. ábra A rendszer használati eset diagramja

A számítógépes alkalmazásban tudja a felhasználó elindítani a szervert, amely fogadja az üzeneteket, illetve végrehajtja ezek tartalmát, mint például a mutató irányítása vagy a diák közötti navigáció. Ezen felül a felhasználónak lehetősége van a szerver leállítására is a számítógépes alkalmazásban. Ebben a felhasználói felületben tudja megnézni a szerver állapotát és a fogadott üzeneteket is. Az utolsó funkció, ami elérhető a felhasználó számára az a telefon kapcsolódásához szükséges QR kód megnyitása.

A telefonos alkalmazás megnyitásakor ahogy az 1. ábrán is látható: a felhasználó a kezdő képernyőre kerül, ahol a csatlakozáshoz, a kezdőképernyőn lévő gomb által átnavigálhat a QR kód beszkenneléséhez. Ezen az oldalon automatikusan megnyílik egy kamera felület, amely

segítségével a számítógépen látható QR kód beszkennelése után át tud lépni az irányító felületre. A mutató irányítása ezen az irányító oldalon van megvalósítva. Ezen az oldalon a felhasználónak lehetősége van a lézer mutató használatához és a különböző akciók használatához: a rajzolás funkció kiválasztásához, a prezentációhoz tartozó diák előre, illetve hátra való mozgatásához. Emellett lehetősége van a telefonos alkalmazásból a szerver leállítására is.

3.2. Rendszerkövetelmények

3.2.1. Funkcionális követelmények

- a felhasználó a rendszer használatához meg kell nyisson egy prezentációt Microsoft PowerPoint-ban;
- a rendszer funkcióinak a használatához a prezentációt el kell indítania a PowerPoint alkalmazásban a diavetítés funkció segítségével;
- a helyes QR kód megnyitásához először el kell indítania a szervert a számítógépes alkalmazásban;
- a felhasználó meg kell győződjön arról, hogy mindkét eszköz ugyanahhoz a WiFi hálózathoz van csatlakozva;
- a telefonos alkalmazásban a szerver leállítása után csak a számítógépes alkalmazásban lehet a szervert újraindítani;

3.2.2. Nem funkcionális követelmények

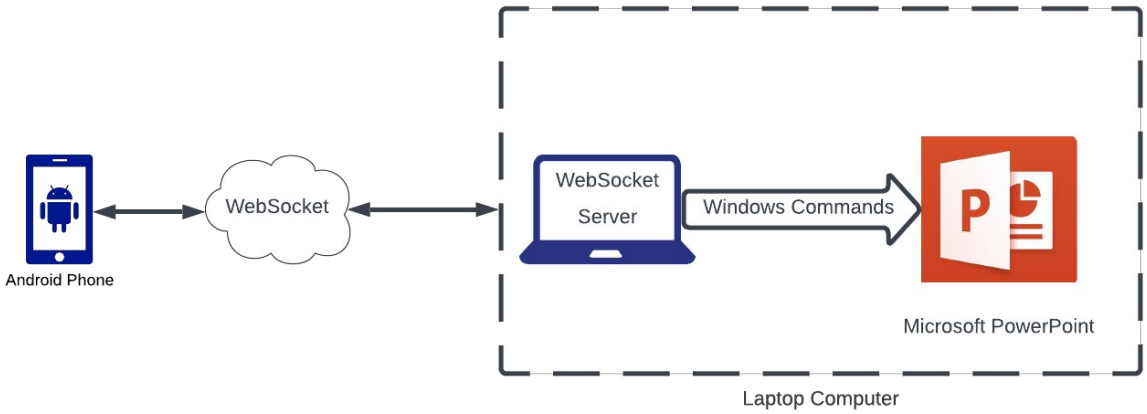
3.2.2.1 Számítógépes alkalmazás

- az operációs rendszer: Windows 11 vagy Windows 10;
- Python 3.11;
- a megfelelő könyvtárak telepítése: asyncio, os, threading, json, pyautogui, PySimpleGUI, websockets, pydirectinput, socket, qrcode, PIL Image;
- WiFi hálózathoz való kapcsolódás lehetősége;

3.2.2.2 Telefonos alkalmazás

- Android 12.0 (ennél újabb);
- minimum API szint 31;
- minimum 100 MB tárhely;

4. A rendszer architektúrája és tervezése



2. ábra A rendszer architektúra diagramja

A rendszer architektúrája három nagy komponensből épül fel, ahogy ez a 2. ábrán is látható a telefonos applikáció, amely Flutter[3] keretrendszerben, Dart nyelven íródott és főbb interakciós felületként szolgál a felhasználó és a rendszer között.

A második komponens számítógépes alkalmazás, amely egy Python[4] nyelvben írt WebSocket szerverként működik a rendszerünkben és a telefon által küldött adatokat dolgozza fel. Az adatok feldolgozása után pedig Windows utasításokkal vezérli a prezentációs alkalmazást. A telefon és a számítógép közötti kommunikációt pedig a WebSocket internetes kommunikációs protokoll végzi el.

A harmadik komponens pedig a prezentációs alkalmazás, amely a mi esetünkben a Microsoft PowerPoint és a számítógépes alkalmazás által létrehozott parancsokat hajtsa végre.

4.1. Kommunikációs protokoll

A rendszer megvalósításához egy fontos lépés a megfelelő kommunikációs protokoll kiválasztása a számítógép és a telefon közötti kapcsolat létrehozásához. A kommunikációs protokollok választása során a legfontosabb szempont a gyorsaság volt, mivel az adatok átviteli sebessége nagyban befolyásolja a fő célként megadott mutató megvalósítását is. Ezért esett a választásunk inkább a WiFi kapcsolaton történő adatátvitelre, mivel a szakirodalomban végzett

[3] https://flutter.dev/

[4] https://www.python.org/

kutatásokban is, arra döntésre jutottak [6], hogy inkább az WiFi-n történő adatátvitel ajánljak, a Bluetooth-on keresztül történő adatátvitelt helyet a kapcsolat sebességé miatt.

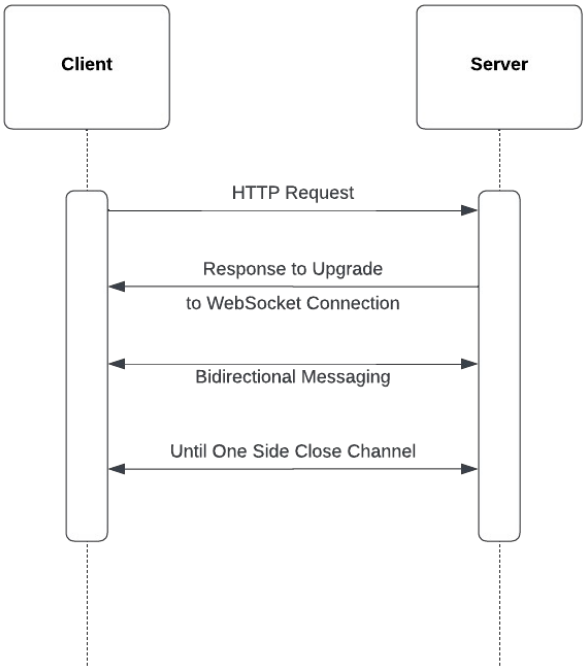
A választásunk tehát a WiFi kapcsolatra esett, de még el kellet döntsük, hogy milyen kommunikációs protokollt fogunk kiválasztani az interneten történő adatátvitel során. Az első opciónak a ZMQ-t (Zero Message Queue) választottuk ki. A ZMQ [5] egy aszinkron üzenet küldő könyvtár, amely üzenet sor segítségével valósítsa meg a kommunikációt két vagy akár több entitás között. A nagy előnye a ZMQ-nak az a nevében is szereplő zéró, aminek a jelentése az, hogy az üzenetek elosztásához nincs szükség egy különálló komponensre úgynevezett broker-re ezáltal is csökkentve a késleltetést. Ezen felül a ZMQ egy full-duplex, kétirányú protokoll, ami azt jelenti, hogy egyszere tud üzenetet fogadni és küldeni ugyanazon a kapcsolaton egy entitásnak. Ez a tulajdonsága a későbbi funkciók bevezetése során lenne fontos a rendszerünk szempontjából.

Viszont a rendszer fejlesztésének az elején egy problémába ütköztünk, amely megakadályozta ennek a protokollnak a használatát a rendszerünkben. A probléma, amely

[5] https://zeromq.org/get-started/

megakadályozta a használatát az volt, hogy a Flutter-es verzió elég kezdetleges állapotban volt, ezért egy másik kommunikációs protokoll mellet döntöttünk.

A választásunk a WebSocket[6] kommunikációs protokollra eset. A WebSocket is egy kétirányú full-duplex protokoll, mint a ZMQ és kliens, szerver kommunikációs modellt valósít meg a TCP protokollra építve. A WebSocket a kapcsolat megteremtéséhez egy HTTP kérést küld



3. ábra WebSocket protokoll működési diagramja

a kliens a szervernek, a kérésre a szerver válaszol egy HTTP üzenettel és a kapcsolatot WebSocket kapcsolatra fejleszti, ahogy ezt a 3. ábra is mutatja [7], de a HTTP kapcsolattal ellentétben nem kell három irányú kézfogást végrehajtson a kapcsolat megteremtéséhez [8]. Miután létrejön a kapcsolat a kliens és a szerver között, a TCP kapcsolat nem fejeződik be addig amíg az egyik fél meg nem

[6] https://proxyscrape.com/blog/websocket-vs-http

[7] https://en.wikipedia.org/wiki/WebSocket

szünteti azt. Ennek a protokollnak nagy előnye, hogy valós idejű adatcserét biztosít a kliens és a szerver között, amely különösen hasznos a mi rendszerünk esetében.

4.2. Számítógépes alkalmazás

4.2.1. Python

A Python a világon a leggyorsabban növekvő programozási nyelvek közé tartozik, ahogy a cikkben is olvasható a népszerűségének a kulcsa, hogy szinte bármilyen területen lehet használni mind adattudomány vagy akár a mesterséges intelligencia fejlesztésének a területén is [9]. A Python nagy népszerűségének egy másik titka, hogy a nyelv szintaxisa egyszerű, könnyen olvasható, így a kezdő programozók tetszését is hamar elnyeri.

Python egy interpretált nyelv, ami azt jelenti, hogy a kód közvetlenül futtatható anélkül, hogy szükség lenne fordításra [10]. Ennek a tulajdonságának az a nagy előnye, hogy könnyen lehet platformtól függetlenül bármely operációs rendszeren futtatni.

Egy másik nagy előnye a Pythonnak, hogy nyílt forráskódú, így rengetek úgynevezett ízesítés létezik belőle, amelyek mind más problémákat oldanak meg. A nyílt forráskódúsága ahhoz is hozzájárul, hogy a fellelhető hibák nagyon hamar kerülnek javításra, ami biztonsági szempontból is előnyös és emelet rengeteg újdonság érkezik hozzá, ami ugyanúgy növeli a népszerűséget és a használhatóságát.

A nagy népszerűségének talán a legnagyobb előnye, hogy a gazdag közösség révén rengeteg fejlesztői támogatást, dokumentációt és könyvtárakat kínál, amelyek segítségével szinte bármit el lehet készíteni a Python nyelv használatával. Igy lehet benne grafikus felületeket készíteni könyvtárak használatával vagy egy kommunikációs szerverként is funkcionálhat egy rendszerben.

4.2.2. Számítógépes alkalmazás leírása

A számítógépes alkalmazás, ahogy már említettem Python nyelvben íródott. Azért eset a Python nyelvre a választásunk, mert a nagy népszerűsége miatt rengeteg csomag elérhető hozzá, amely megkönnyíti a különböző funkciók implementálását.

Az alkalmazás egy több szálon futó program, a fő szálon fut a felhasználói felület a GUI és egy másik szálon a WebSocket szerver. A többszálúság megvalósításához a threading [8] csomagot használtam fel.

Az alkalmazás indítása után három opció lehetséges: elindítható a szerver, leállítható a szerver, amivel be is záródik az alkalmazás és megnézhető telefon csatlakoztatásához szükséges QR kód, amely megnézése után visszakerülünk az alkalmazás fő ablakára.

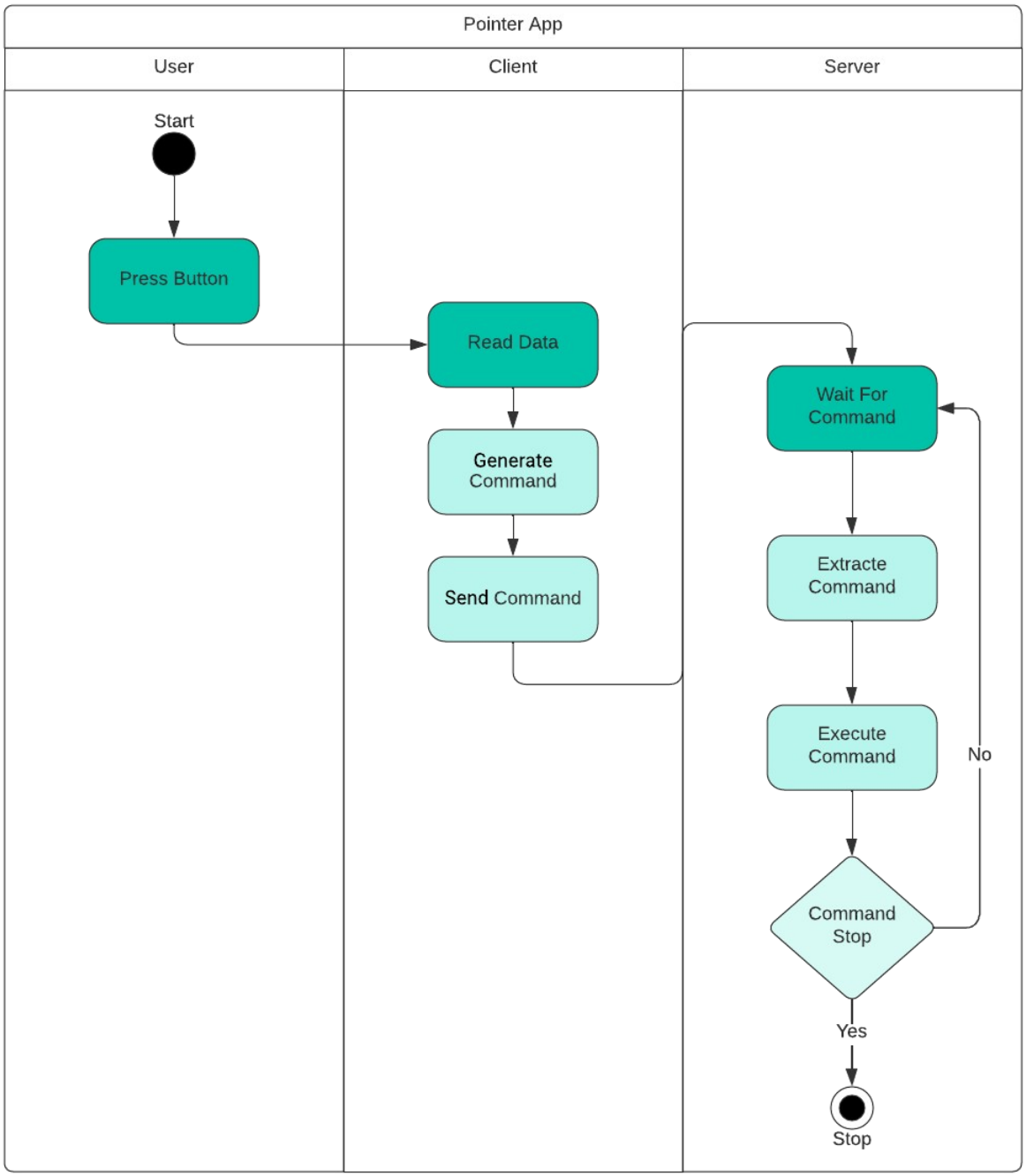
A szerver elindítása után az alkalmazás fogadja a WebSocket kapcsolaton át küldött üzeneteket. A számítógépes alkalmazás által fogadott üzenetek kezelésének a lépései a 4. ábrán látható aktivitás diagramon kerülnek bemutatásra.

Az üzenetek kiindulópontja a rendszerünkben, a felhasználó és az általa generált üzeneteket kell feldolgozza a szerver. A 4. ábrán is megfigyelhető, hogy a felhasználó által létrehozott akcióra reagál a telefonos alkalmazás, amely a rendszerünkben kommunikációs szempontból egy kliensként funkcionál. A telefonos alkalmazás szerepe az adatok beolvasása és ezeknek az adatoknak a felhasználásával, parancsok létrehozása, amelyeket továbbítani tud a számítógépes Python alkalmazásnak.

A számítógépes alkalmazás az érkező üzeneteket először fel kell dolgozza, ahhoz, hogy végre tudja hajtani az üzenetekben lévő parancsokat. Az üzenetek feldolgozása során a számítógépes alkalmazás megnézi, hogy milyen parancs érkezet a telefonos alkalmazástól. Miután tudja a parancsot, végre is hajtja annak a parancsnak a tartalmát és visszamegy a parancs várási állapotába. A parancs várási állapotba csak, akkor nem megy vissza, ha a kapott parancs a szerver megállítását kéri.

[8] https://docs.python.org/3/library/threading.html

Ezeknek az állapotoknak a működésének a részletes leírása, a szerver leírásánál kerül majd bemutatásra.



4. ábra Adatáramlás a rendszerben

4.2.3. A felhasználói felület

A felhasználói felület, amely az 5. ábrán látható felel minden lehetséges funkció kezeléséhez amely, elérhető a felhasználó számára.

5. ábra A számítógépes alkalmazás felhasználói felülete

A felhasználói felület PySimpleGUI[9] csomag felhasználásával van megvalósítva. Azért választottuk a számítógépes alkalmazás felhasználói felületének a megvalósításához a PySimpleGUI csomagot, mert nagyon könnyedén és gyorsan lehet benne felhasználói felületeket megtervezni. Emellett pedig lehetővé teszi az elemek rugalmas elrendezését és lehetőség van a felhasználói felületi elemek méretének, elhelyezkedésének és megjelenítésének testreszabására. A rengeteg dokumentációs anyag révén, pedig nagyon könnyen elsajátítható a csomag használata.

A felhasználói felület megkötése, hogy csak az alkalmazás fő szálán futhat. Maga a felület megvalósítása the_gui() függvényben van megírva. A the_gui() függvény tartalmazza: a alkalmazás témáját, az elemeket egy elrendezésben úgynevezett layout-ban, az ablak (window) létrehozását amely tartalmazza az elemeket és az események (events) lekezelését is. Ezeket szeretném most részletesen is bemutatni.

Az első lépés a felhasználói felület létrehozása során a téma (theme) és az elrendezés létrehozása. Az elrendezés tartalmazza azokat az elemeket, amelyeket meg szeretnénk jeleníteni a felhasználói felületen. Ahogy 6. ábrán is látható az alkalmazás témáját fekete színre állítottam a

[9] https://www.pysimplegui.org/en/latest/

sg.theme() függvény segítségével, ez a függvény a PySimpleGUI előre megírt függvénye. Ez után létrehoztam az elrendezést egy layout listában. A layout lista tartalmazza a szövegmezőt, amely a fogadott adatokat jeleníti meg, emelet a szerver állapota is ebben a szövegmezőben figyelhető meg.

A layout lista ahogy a 6. ábrán is látható tartalmazza a három gombot, amelyeket az sg.Button() függvény hoz létre. Minden gomb más funkció végrehajtásáért felel, amit majd az

```
def the_gui():
    sg.theme('Black')  # give our window a spiffy set of colors

    layout = [[sg.Text('Output Text', size=(40, 1))],
              [sg.Output(size=(110, 20), font=('Helvetica 10')),
               sg.Button('Start', button_color=('black', 'darkslateblue')),
               sg.Button('Stop', button_color=('black', 'firebrick'))],
              [sg.Button('Open QR Code', button_color=('black', 'azure4'))]]

    window = sg.Window("Websocket Server", layout, font=('Helvetica', ' 13'), default_button_element_size=(8, 2),
                       use_default_focus=False, finalize=True)
```

6. ábra A számítógépes alkalmazás felhasználói felületének elemeinek létrehozása kódban

esemény kezelésnél részletesebben is kifejtek. Az is megfigyelhető a 6. ábrán, hogy az elemeknek itt lehet állítani a tulajdonságait, mint például: a méretét, színét vagy az elem által megjelenített szöveg tartalmát.

A fő elem, amely összefogja a felhasználói felület többi elemét az ablak (window). Az ablak létrehozása az sg.Window() függvényen keresztül történik. Ebben a függvényben adjuk meg az ablakunk címét, itt rendeljük hozzá az elrendezést is. Emelet itt lehet beállítani, hogy az alap gomb méret mekkora legyen vagy, hogy induláskor kerüljön fókuszba egy elem vagy sem. Az utolsó lépés az ablak megjelenítéséhez a finalize paraméter igazra állítása, ami azt jelzi, hogy befejeztük az ablak paramétereinek változtatását és megjeleníthető a felhasználó számára.

A következő lépés az események lekezelése. Maga a felhasználói felület egy végtelen ciklusban fut ahol a window.read() függvénnyel nézi a bejövő eseményeket és azok értékeit ahogy ez a 6. ábrán is megfigyelhető. A windows.read() függvény egy event, value változóba kimenti a beérkező esemény nevét és az eseménnyel érkező értéket ha van. Ez után három if segítségével vizsgálja, hogy milyen esemény érkezett meg. Az eseményeket a Start, Open QR Code, Stop gomb vagy az ablak bezárása kezdeményezheti.

Az első if ágban a 7. ábrán, azt vizsgálja az alkalmazás, hogy az event változóba Stop érték érkezett, amely a Stop gomb megnyomására jöhet létre, vagy az ablakot zárta be a felhasználó, ha valamelyik igaz, akkor az asyncio.get_event_loop().stop() függvény hivásal megálitja a

WebSocket szervert és kilép a végtelen ciklusból ezt követően a window.close() függvénnyel lezárja az ablakot és megáll alkalmazás működése.

A második if ágban a Start gomb lenyomása van kezelve, a gomb lenyomásakor a threading.Thread(target=run, daemon=True).start() függvénnyel egy új szálon végrehajtja a run()

```
while True:  # The Event Loop
    event, value = window.read()
    if event in (sg.WIN_CLOSED, 'Stop'):  # quit if exit button or X
        asyncio.get_event_loop().stop()
        break

    if event == 'Start':
        threading.Thread(target=run, daemon=True).start()
        print("Server started")

    if event == 'Open QR Code':
        im = Image.open("connect_qrcode.png")
        im.show()

window.close()
```

7. ábra A számítógépes GUI eseménykezelése

függvényt, amely tartalmazza a WebSocket elindítását. Ezen kívül a szövegdobozba kiírja a print segítségével, hogy a szerver elindult.

Az utolsó if ágban az Open QR Code gomb lenyomása van lekezelve, amely a qrcodeGenerate() függvény által generált png állományt nyissa meg a PIL[10] csomag Image.open()

8. ábra QR Kód megjelenítése

[10] https://pypi.org/project/Pillow/

függvény segítségével és a show() függvénnyel jeleníti meg a felhasználó számára ahogy ez a 8. ábrán is látható.

4.2.4. WebSocket Szerver

A WebSocket szerver a run() függvényben kerül létrehozásra egy külön szálon amelyet, már említett Start gomb lenyomásakor hoz létre az alkalmazás. A szerver létrehozásának első lépése egy eseményhurok létrehozása (event loop) az asyncio.new_event_loop() függvénnyel. Azért van szükség az eseményhurokra, mert a szerver az eseményhurok segítségével tudja lekezelni az aszinkron műveleteket. Ezenkívül az eseményhurok lehetővé teszi, hogy a szerver egyszerre több kapcsolatot kezeljen és folyamatosan figyelje az érkező üzeneteket vagy eseményeket anélkül, hogy blokkoló módon várakozna. Az eseményhurok létrehozása után be kell állítani a hurkot a jelenlegi szálra az asyncio [11] csomag asyncio.set_event_loop() függvényével, ahogy ez a 9. ábrán is látható.

```
def run():
    # must set a new loop for asyncio
    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)
    # setup a server

    # get ip
    ip_address, port = ipGenerate()

    # generate qr code
    qrcodeGenerate(ip_address, port)

    loop.run_until_complete(websockets.serve(listen, ip_address, port))
    # keep thread running
    loop.run_forever()
```

9. ábra WebSocket létrehozása

A következő lépés a QR kód generálásához szükséges ip cím és port lekérése. Ezeknek a változóknak a beállítását az ipGenerate() függvény valósítsa meg. A függvény, amely tartalma látható a 10. ábrán, első lépésként létrehoz a socket[12] csomag használatával egy UDP socket-et.

[11] https://pypi.org/project/asyncio/

[12] https://docs.python.org/3/library/socket.html

A socket-el csatlakozik a Google egyik DNS szerverére, sikeres kapcsolat után pedig az s.getsocketname() függvény első visszatérített értékét kimenti az ip_address változóba. Azért van szükség a Google DNS szerverétől lekérni a számítógép ip címét, mert ha például a felhasználó használ virtuális gépet vagy más hasonló típusú alkalmazást, akkor az ip címek listája a számítógépen változhat és nem tudná mindig kivenni a rendszer a megfelelő ip címet a listából mivel nem tudná megfelelő ip cím helyzetét a listában. Miután kimentette a számítógép azt a címet,

```
def ipGenerate():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.connect(("8.8.8.8", 80))
    ip_address = s.getsockname()[0]  # get ip address

    port = os.environ.get('PORT', 8080)
    port = str(port)
    return ip_address, port
```

10. ábra IP cím lekérése

amin keresztül van lehetősége az interneten kommunikálni, a port beállítása történik meg. A port beállítása történhet környezeti változóból kapott értékel vagy ha nincs beállítva környezeti változó akkor az alapértelmezett értéket téríti vissza.

Az utolsó előtti lépés a WebSocket indítása előtt, a QR kód kigenerálása ahhoz, hogy már említett Open QR Code gomb lenyomásakor a felhasználó láthassa a kódot, amit be kell olvasón a telefonal. A kód generálását a qrcodeGenerate() függvény végzi, amely bemenetként megkapja az ip címet illetve a port-ot. A függvényen belül amint azt a 11. ábra is mutassa, a QR kód generálását a qrcode csomag make() függvénye végzi el, amit aztán egy png fájlba ment ki az alkalmazás img.save() függvény használatával. Ezt a png fájlt nyissa meg majd a Open QR Code gomb.

Az utolsó lépés a WebSocket szerver tényleges elindítása. A szervert a eseményhurokban indítsuk el a websockets.serve() függvényhívással. Ahhoz, hogy a szerver folyamatosan

```
def qrcodeGenerate(ip_address, port):
    img = qrcode.make(ip_address + ":" + port)
    type(img)  # qrcode.image.pil.PilImage
    img.save("connect_qrcode.png")
```

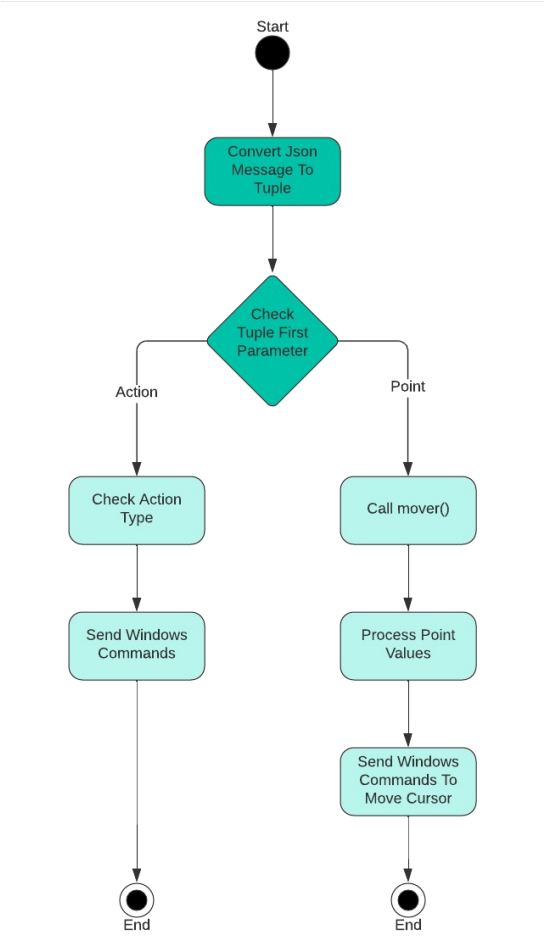
11. ábra QR kód generálása

hallgatózzon és kezelje a WebSocket kapcsolatokat, a loop.run_forever() függvényhívás valósítja meg. A websockets.serve() függvényben észrevehető egy listen nevű paraméter, amely annak a függvény nevét mondja meg, amelyiket bejövő kapcsolat esetén meghív a szerver.

A listen függvény egy async függvény, amelynek a feladata az üzenetek hallgatása és visszaküldése a telefonnak. A visszaküldés a rendszer működéséhez nem szükséges, viszont a tesztelés során nagy segítséget jelentett. Az érkező üzeneteket egy asyncron for ciklusban adja át a messageCheck() függvénynek, amelynek a szerepe az üzenetek feldolgozása. Emellett itt kerül kiíratásra az érkező üzenet a számítógépes alkalmazás felhasználói felületére és ebben a for ciklusban küldi vissza az üzenetet a telefon számára is tesztelés esetén.

4.2.5. Az érkező üzenetek feldolgozása

Az üzenetek feldolgozásának a lépései a 12. ábrán is látható. Az üzenetek feldolgozása egy több lépéses folyamat, amit a számítógépes alkalmazás kell elvégezen az üzenetek fogadása után.



12. ábra Egy fogadott üzenet feldolgozásának a lépései

Az üzenetek feldolgozásának első lépése a messageCheck() függvényben, az üzenetek deszerializálása JSON formátumból. Azért döntöttünk az üzenetek JSON formátumban való küldésében, mert az adatok kulcs-érték párokba vannak szervezve, ami megkönnyíti a feldolgozásukat és emelet az emberek számára is jobban átlátható az üzenetek tartalma.

Az üzenetek két típusba vannak osztályozva, az első típus az akció (actions), amelybe tartoznak a parancsok és a második típus a pontok (points) amelyek a telefon által küldött gyorsúlásmérő adatait tartalmazzák.

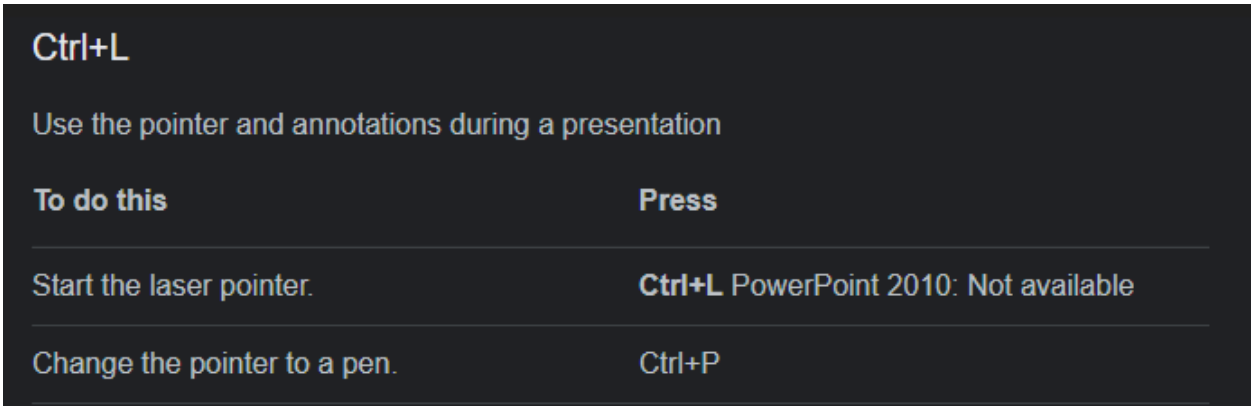
Az első típus több akciót is lefed, ahogy ezt a 13. ábra is mutatja, melyek az alkalmazás különböző funkcióit valósítsák meg. Az első akció a kalibrálás, ami minden lézermutató indítása

```
if response['type'] == 'actions':
    if response['action'] == 'calibrate':
        pyautogui.moveTo(screenWidth / 2, screenHeight / 2)  # move mouse to the center of screen

    if response['action'] == 'startLaserPointer' or response['action'] == 'stopLaserPointer':
        pyautogui.hotkey('ctrl', 'l')  # laser pointer

    if response['action'] == 'startDraw':
        pyautogui.hotkey('ctrl', 'p')  # draw with pen
        pyautogui.mouseDown(button='left')

    if response['action'] == 'stopDraw':
        pyautogui.mouseUp(button='left')
        pyautogui.hotkey('ctrl', 'p')  # draw with pen

    if response['action'] == 'pressLeft':
        pyautogui.press('left')  # press left arrow key

    if response['action'] == 'pressRight':
        pyautogui.press('right')  # press right arrow key

    elif response['action'] == 'stop':
        asyncio.get_event_loop().stop()
```

13. ábra A szerver által fogadott akciók kezelése

előtt végrehajtódik. Ennek az akciónak az a szerepe, hogy a lézermutató kezdőpontját mindig beállítja az aktuális képernyő közepére a pyautogui.moveTo() függvény használatával. Az aktuális képernyő méretét egy globális változóba menti ki az alkalmazás a pyautogui.size() függvény hívással az alkalmazás indulásakor.

A lézermutató elindítása gyorsbillentyűk (hotkey) lenyomásának a szimulálásával történik, mivel a PowerPoint-ban, ahogy a 14. ábrán is látható minden funkciónak létezik egy ilyen billentyű kombinációja, amely nagyban megkönnyíti ezeknek a funkcióknak a használatát. A lézermutató indításakor az alkalmazás egy startLaserPointer üzenetet kap a telefonos alkalmazástól, ekkor meghívja a pyautogui.hotkey() függvényt, amely billentyűzeten lévő karakterek lenyomását tudja szimulálni, hogy a prezentációs program lézermutató módra váltson. A lézermutató kikapcsolása is hasonló módon működik.



14. ábra PowerPoint gyorsbillentyű kombinációk

A rajzolás bekapcsolása, illetve kikapcsolása is a lézermutatóhoz hasonló gyorsbillentyűk használatával van megoldva, csak a rajzolás esetében szükség van az egér bal gombjának a lenyomva tartását is szimulálni a funkció használatához a pyautogui.mouseDown(button='left') függvény segítségével. A rajzolás befejezésénél pedig, ahogy a 13. ábra is mutatja, az egér gombot felengedjük és meghívjuk a rajzolásnak megfelelő gyorsbillentyű kombinációt.

A prezentáció diáinak az előre vagy hátra mozgatásának megvalósításához is a pyautogui[13] csomagot használtuk fel. A mozgatást a jobb és balra nyilak szimulálása által van megvalósítva a pyautogui.press() függvény használatával.

Az utolsó akció, ami érkezhet a szerver felé a telefonos alkalmazástól, az a szerver leállítása. A szerver leállítása a telefonos alkalmazásból, ugyanúgy az eseményhurok leállításával valósul meg, mintha megnyomnánk a Stop gombot a számítógépes alkalmazásban.

[13] https://pypi.org/project/PyAutoGUI/

A második típusú válasz 15. ábrán is megfigyelhető pontok (points), melyek a gyorsulásmérő értékeit fedik le és a mover() függvényben kerülnek feldolgozásra. A pontok

```
if response['type'] == 'points':
    mover(response)
```

15. ábra A szerver pont típusú adat kezelése

felhasználásával van megoldva a mutató mozgatása és a rajzolás alatt történő mozgatás. A pont típus mindig a startLaserPointer akció vagy a startDraw akció után érkezik és egészen addig ilyen típusú üzenetek érkeznek, amíg a két akciónak megfelelő megállítási üzenet nem érkezik meg.

A mover() függvény első lépése, a pont típusú válaszok x illetve y koordinátáinak az átalakítása kéttizedes pontosságú float értékekre. Azért van szükség az érkező értékeket kéttizedes pontosságú float értékekre alakítani, hogy a gyorsulásmérő által észlelt nagyon kicsi mozgásokat kiszűrjük, így növelve a mutató pontosságát.

A második lépésben, kiszűrjük a nagyon magas értékeket, ez azért fontos mert a felhasználó túl nagy mozdulatainak engedélyezése nagyban megnehezíti a mutató használatát. Ezt követően az alkalmazás, ahogy ez a 16. ábra is mutatja megnézi, hogy az értékek bele esnek a -7.0 és 7.0 értékek közé, ha beleesnek beszorozza az értékeket öttel, ha nem esnek bele nyolccal szorozza be és átalakítja a kapott értéket int típusú értéke mindkét esetben. Az értékek beszorzása a mutató gyorsaságát növeli meg és az értékek int típussá való átalakítása azért szükséges, mert ezek az értékek pixeleknek fognak megfelelni a mozgatás során, igy nagyon kis értékek esetén nagyon lassan mozogna a mutató. A beszorzáshoz használt értékeket, illetve a határértéket tesztelés során választottuk ki. Az értékeket azért szorozzuk két különböző számmal, hogy kisebb mozdulatoknál megmaradjon a pontosság, viszont nagyobb mozdulat esetén, lehetséges legyen a mutató pozíciójának gyors megváltoztatása. Határértéknek pedig, azért választottuk a -7, illetve 7

```
if -15.0 < xkord < 15.0 and -15.0 < ykord < 15.0:

    if -7.0 < xkord < 7.0 and -7.0 < ykord < 7.0:
        xkordint = int(xkord * 5)
        ykordint = int(ykord * 5)
    else:
        xkordint = int(xkord * 8)
        ykordint = int(ykord * 8)
```

16. ábra Mutató gyorsaságának állítása

értékeket, mert a gyorsulásmérő által mért érték a telefon függőleges helyzetében közelíti a -9.8 jobbra döntve és a 9.8 balra döntve, ebből kiindulva, pedig tesztelés által választottuk egy köztes értéket, ami a hét lett, mert, így a mutató precíz irányítása is megvalósítható és a mutató gyors mozgatása is kivitelezhető.

A harmadik lépésben, a kapott értékek berakásra kerülnek egy position nevű sorhoz, amely a MeasurementFilter osztály része. A sor mérete az alkalmazás indulásakor háromra van állítva, ezt az értéket szintén tesztelés alapján választottam ki. Az osztály szerepe a kapott értékekből egy átlagot számolni, amelyet visszatérít két int típusú változónak. Azért számolunk a kapott három értékből egy átlagot, hogy ez által is csökkentsük a beérkező zajt.

Az utolsó lépés, ahogy az a 17. ábrán is megfigyelhető a mutató mozgatása. A mutató mozgatásához, előbb le kell kérje a rendszer a mutató jelenlegi helyzetét. A helyzetet a

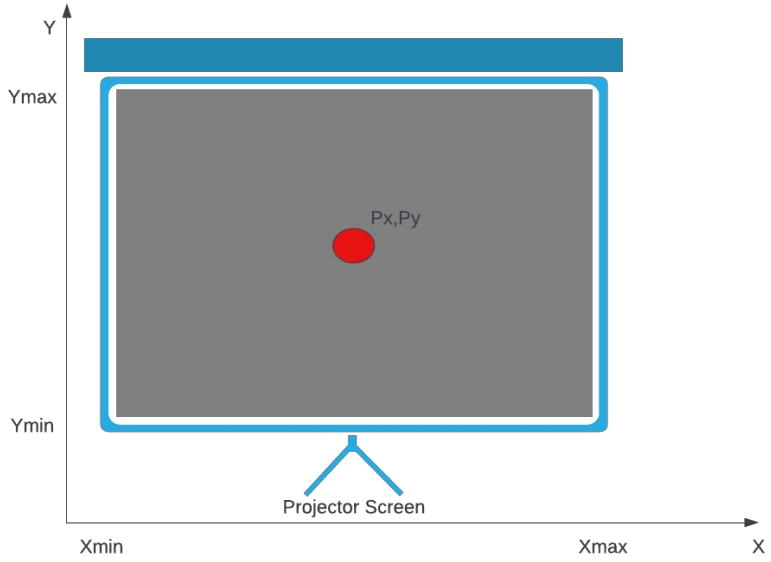
```
position.append((xkordint, ykordint))
if position.full():
    xfilter, yfilter = position.get_filtered()
    posx, posy = pyautogui.position()

# ez csak windows-al kompatibilis
pydirectinput.moveTo(int(posx - xfilter), int(posy - yfilter))
```

17. ábra Mutató mozgatása

pyautogui.position() függvény teríti vissza egy tuple-ként, mely két int típusú koordinátát tartalmaz. Miután tudjuk a mutató jelenlegi helyzetét, meghívásra kerül a pydirectinput.moveTo() függvénye, amelyben kivonjuk a jelenlegi helyzetből a telefon által kapott értéket, hogy megkapja az alkalmazás a mutató új koordinátáját, amelyre a függvény használatával beállításra is kerül.

A mutató irányítása során fontos azt is megjegyeznünk, hogy a lézermutató helyzete a mozgatás során nem haladhatja meg képernyő méretinek az értékeit. Ennek a megakadályozása miatt is döntöttünk, a mellet, hogy a mutató kezdőpontja mindig a képernyő közepe legyen és hogy a túlságosan nagy gyorsulásmérő értékeket nem vesszük figyelembe. Mivel, ha engednénk a 18. ábrán látható virtuális mutató Px illetve Py koordinátainak az értékeinek a képernyő határain kívül lévő értékeknek a felvételét, akkor nagyon nagy esély lenne arra, hogy a felhasználó hibákat tapasztalna a rendszer használata alatt. A képernyő határai automatikusan beállításra kerülnek, amikor pydirectinput.moveTo() függvényt használjuk.

Egy másik indok, amiért nem vesszük figyelembe a nagyon nagy értékeket, amelyeket a gyorsulásmérő küld, hogy a virtuális mutató irányítása folyékonyabb, kiszámíthatóbb legyen. A gyorsulásmérő értékeinek a határának a -15.0 és 15.0 értékeket választottuk tesztelés alapján, mivel nagyon nagy kimozdulás esetén kaphatok ilyen magas értékek.



18. ábra A virtuális mutató határainak a szemléltetése

A mutató mozgatásához azért van szükségünk pydirectinput csomag használatához, mert ahogy a csomag[14] leírásában is olvasható léteznek olyan alkalmazások, mint például a PowerPoint, amelyek DirectInput Scan Codes használnak az egér vagy billentyűzet utasítások kezeléséhez Windows felületen, míg a pyautogui csomag Virtual Key Codes (VKs) használ.

4.3. Telefonos alkalmazás

4.3.1. Flutter

A Flutter a Google által létrehozott és fejlesztett nyílt forráskódú programozási környezet, amely segítségével natív nyelvre forduló többplatformos mobil alkalmazásokat lehet készíteni. A Flutter egyik fő előnye, hogy a forráskód megosztható. A forráskód megosztásának a lehetősége azért előnyös, mert a fejlesztők ugyanazt a kódbázist használhatják az Android-os, az iOS-es és a Web-es alkalmazásokhoz, csökkentve ezzel a fejlesztési időt és a költségeket. Egy másik nagy

[14] https://pypi.org/project/PyDirectInput/

előnye a Flutter-nek, hogy támogassa az elő frissítéseket (hot reload), amelyek segítségével a fejlesztők azonnal láthassák a kódban történt változtatások eredményét [11].

Flutter keretrendszerben az alkalmazások Dart nyelvben vannak írva. A Dart nyelv is a Google által fejlesztett és karbantartót, viszont rengeteg nyílt forráskódú csomagot támogat, amelyek használatával szinte bármit meg lehet valósítani Dart nyelvben. A Dart nyelvet viszont nem csak Flutter-es alkalmazásokban lehet használni, hanem akár a Dart virtuális gép segítségével a Dart kódot JavaScriptre fordítva futtatható a Web-en is.

A Dart nyelv nagy előnye a fejlesztők számára, hogy támogassa az épp időben (just in time, JIT) kompilációt és az idő előtti (ahead of time, AOT) kompilációt. Az AOT kompilálás segítségével a Dart hatékonysága meg tudja közelíteni a natív kódot, ami nagyban megnöveli a Flutter sebességet. A JIT kompilálás segítségével pedig a Dart lehetővé teszi a fejlesztő számára, hogy valós időben láthassa a kódban történt változtatások eredményeit [12].

A Dart nyelv előnyei közé tartozik még, hogy egy objektumorientált nyelv, ami azt jelenti, hogy minden programozási egységet objektumonként kezel, így az objektumok használatával járó előnyök, mint például az öröklődés használható Dart nyelvben is.

A második előnye a Dart nyelvnek, hogy van beépített szemétgyűjtője (garbage collector), amely automatikusan felszabadítja a nem használt memóriát, így a fejlesztők nem kell a memória kezeléssel foglalkozzanak.

A harmadik előnye, hogy támogassa az aszinkron programozást az async/await és Future kulcsszavak használatával, így lehetővé téve például az adatcserék hatékony lebonyolítását.

Az utolsó előnye a már említett külső könyvtárak használatának lehetősége, illetve, hogy a Dart nyelv támogassa a statikus típus ellenőrzést, ezzel is hozzájárulva egy biztonságosabb kódhoz [12].

A Flutter egyik fő eleme a widgetek. Egy Flutter alkalmazásban szinte minden widgetek-ből van felépítve. A widgetek két nagy csoportba bonthatóak a stateless widgetek-re és stateful widgetek-re [12].

A statless widgetek olyan widgetek, amelyeknek a belső állapota nem változik az alkalmazás futása alatt. Ez azt jelenti, hogy a widget megjelenése ugyanaz marad, mint amit az alkalmazás indításakor megkapót. Példa egy ilyen widget-re egy egyszerű szövegdoboz.

A stateful widgetek pedig olyan widgetek amelyeknek a belső állapota változhat az alkalmazás futása alatt. Amikor az állapota változik, akkor újraépíti magát, hogy a változtatások

láthatóak legyenek a felhasználói felületen is. Példa egy ilyen widget-re egy videólejátszó felület, ahol a felhasználó megállíthatja vagy elindíthatja a videót.

A widgetek nagy előnye, hogy egymásba is lehet ágyazni őket, így nagyon könnyen létrehozhatok komplex felhasználói felületek.

Flutter használata esetén elengedhetetlen, hogy megemlítsük a Material Design-t. A Material Design-t is a Google fejlesztette ki 2014-ben, hogy egy olyan dizájn nyelvet hozzanak létre, amely egyesíti a különböző eszközökre fejlesztett alkalmazások felhasználói felületét [12].

A Material Design egy letisztult, modern, esztétikus kialakításra törekszik, animációk és intuitív interaktív felületek használatával. A Material Design alapelvei közé tartozik az anyagok, a rétegek és a világítás felhasználása. Az anyagok szerepe, hogy reprezentálják az objektumokat, amelyek a rétegek segítségével térbeli hierarchiát és mélységet adnak a dizájnnak. A világítás használatával pedig jelezni lehet a felhasználónak a kijelölt elemeket, hogy megteremtsük egy intuitív felhasználói felületet [13].

A Material Design másik alapelve az élénk színek használata, amely hozzájárul egy kontrasztos vonzó megjelenéshez. Emellett témákat is létrehozhatunk, hogy egy egységes kinézetett adjunk az alkalmazásunknak.

A dizájnba nagy hangsúly van fektetve az animációk használatára is, amelyek segítenek egy simább természetesebb interakciós felület létrehozásában a felhasználó számára. Az animációk szerepe az elemek közötti kapcsolatok átmenetek jelzése is a felhasználó számára.

Az utolsó elem a különböző betűtípusokat és méretű szövegeket testreszabhatóságának a lehetősége és ezeknek stílusokba való elhelyezése, amely segíti a fejlesztőt egy átláthatóbb, könnyebben olvasható felhasználói felület tervezésében.

4.3.2. Telefonos alkalmazás leírása

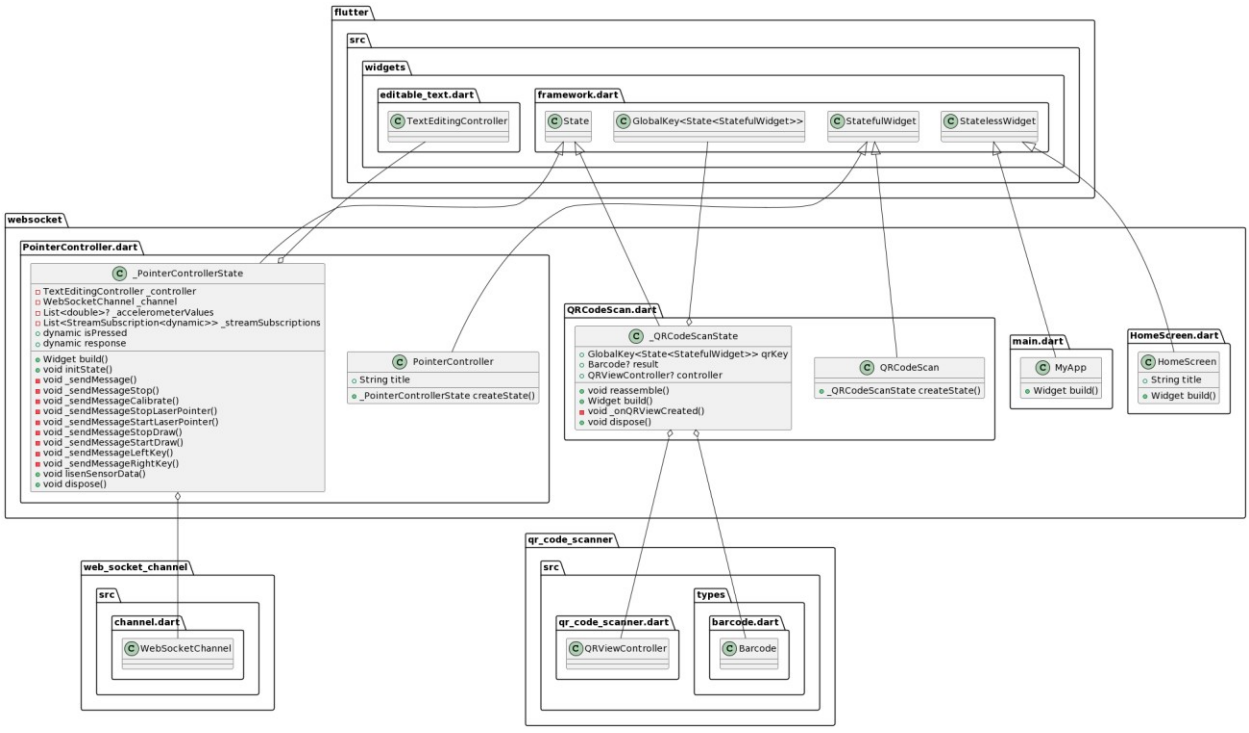
A telefonos alkalmazás Flutter keretrendszerben íródott. Azért választottuk a Flutter-t mert lehetőséget nyújt az alkalmazás más platformokra való megvalósításához egy kódbázis felhasználásával. Emellett a Flutter a Python-hoz hasonlóan rengetek csomagot támogat melyek segítségével a fejlesztők hatékonyan tudnak különböző problémákra megoldásokat kapni.

A telefonos alkalmazás főbb alkotó részeit az 19. ábra mutatja be egy osztálydiagram segítségével. Az osztály diagramon látható, hogy az alkalmazás négy főbb osztályból épül fel: a PointerController osztály, a QRCodeScan osztály, a MyApp osztály és a HomeScreen osztály. Ezek az osztályok mind a Flutter StatefulWidget vagy StatelessWidget osztályokból származnak. Az osztályok szerepe, hogy a nevükből is adódó állapotok (state) használatának megvalósítása az

alkalmazásban. A StatefulWidget olyan felhasználói elemeknek a használatát teszi lehetővé, amelyeknek az állapota változhat az alkalmazás futása alatt. Ez azt jelenti, hogy a felhasználói felületen az elemek képések újrarajzolni magukat, mint például, ha lenyomunk egy gombot, akkor azt jelezni tudja az alkalmazás a felhasználói felületen egy számláló növelésével.

A StatelessWidget olyan felhasználói felületi elemeket reprezentál, amelyeknek nincs belső állapotuk és nem lehet frissíteni az állapotukat az alkalmazás futtatása alatt. Ezt a widget típust statikus elemeknek a megvalósításához használják, mint például a szövegek vagy képek megjelenítéséhez.

A 19. ábrán szereplő osztálydiagramon az is megfigyelhető, hogy a PointerController osztály és a QRCodeScan osztály két másik osztályt is felhasznál a funkcionalitásának a megvalósításához. A PointerController osztály a WebSocketChannel osztályt használja fel, amelynek a segítségével a WebSocket kapcsolatot hozz létre, amely segítségével a mutató irányításához szükséges adatokat tudja elküldeni a szerver alkalmazásnak. A QRCodeScan osztály pedig a QRViewController és a Barcode osztályt használja fel, hogy a QR kód beolvasását megvalósítja.



19. ábra A telefonos alkalmazás osztály diagramja

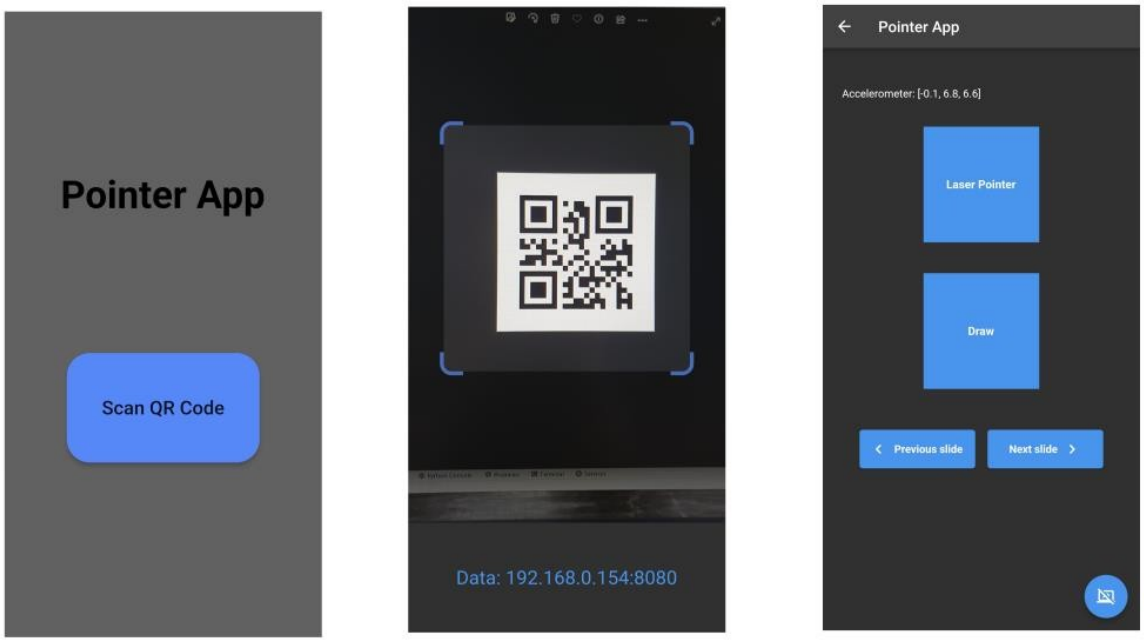
A main.dart fájlban lévő MyApp osztály szerepe az alkalmazás főbb tulajdonságainak a beállítása, mint például a témája és a HomeScreen osztály egy üdvözlő oldalt biztosít a felhasználó

számára. Az osztályok szerepének és függvényeinek a részletes bemutatása a későbbiekben kerül sorra.

4.3.3. A telefonos alkalmazás felhasználói felülete

A telefonos alkalmazás felhasználói felülete widgettek-ből épül fel, melyek fa struktúrába vannak elhelyezkedve. A widgetek lehetővé teszik az alkalmazás felhasználói felületének egyszerű kialakítását emelet, pedig könnyen szerkeszteni lehet a már meglévő elemeinket. Flutter keretrendszerben a widgetek jelképezik a felhasználó számára azokat az elemeket, amelyekkel kapcsolatba tud lépni. Minden widget hierarchiájának van gyökér-widgete, amely a mi alkalmazásunk esetében a MaterialApp. A MaterialApp arra szolgál, hogy implementálja a Material Design[15] stílusához kapcsolódó irányelveket és biztosítsa az alkalmazás számára az alapvető felhasználói felületi elemeket, mint például: a gombokat és a navigációs sémákat. Ezen felül az alkalmazásunk témáját és stílusának a beállítását is a MaterialApp használatával tudjuk módosítani.

Az alkalmazás felhasználói felülete, ahogy a 20. ábra is mutatja három oldalból épül fel: a HomeScreen-ből, QRCodeScan-ből és a PointerController-ból.



20. ábra Telefonos alkalmazás felhasználói felülete

[15] https://m3.material.io/get-started

Az első oldal a HomeScreen melyen látható az alkalmazásunk neve és egy gomb, mely segítségével átléphet a felhasználó a következő oldalra. Az alkalmazás felhasználói felületének tervezése során igyekeztünk arra törekedni, hogy a felület átlátható legyen emelet a felhasználónak intuitív legyen a használata magyarázatok nélkül is. Ezért az alkalmazás témájának is egy elég letisztult fekete, kék, fehér témát választottam.

Az alkalmazás témájának a beállítása a 21. ábrán is látható main nevezetű oldalon történik. Ez az oldal nem látható a felhasználó számára. A téma beállítása a MaterialApp ThemeData nevű osztályában történik meg, amelynek a color osztálya segítségével tudjuk beállítani az alkalmazás során használni kívánt színeket. Emellett itt tudjuk az alkalmazás nevét is beállítani a MaterialApp title osztályának segítségével.

```
const title = 'Pointer App';
return MaterialApp(
  title: title,
  theme: ThemeData(
      brightness: Brightness.dark),  // ThemeData
      color: Colors.blue,
```

21. ábra Telefonos alkalmazás témájának a beállítása

A második oldalon, amelyre a felhasználó a Scan QR Code gomb megnyomásával kerülhet az a QRCodeScan oldal. Ezen az oldalon a qr_code_scanner[16] csomag használatával példányosítsuk a QRView osztályt. A QRView osztály használatával, ahogy a 20. ábra középső

```
Expanded(
  flex: 5,
  child: QRView(
    key: qrKey,
    overlay: QrScannerOverlayShape(
      cutOutSize: MediaQuery.of(context).size.width * 0.8,
      borderWidth: 10,
      borderColor: Colors.blueAccent,
      borderLength: 20,
      borderRadius: 10,
    ),  // QrScannerOverlayShape
    onQRViewCreated: _onQRViewCreated,
  ),  // QRView
),  // Expanded
```

22. ábra QRView osztály

[16] https://pub.dev/packages/qr_code_scanner

oldalán is látható, az alkalmazás le tudja kérni a telefon kamerájának a képét, természetesen csak akkor, ha a felhasználó engedélyezi az alkalmazás számára a telefon kamerájának a használatát. A QRView osztály QrScannerOverlayShape osztályával, amely a 22. ábrán látható, az alkalmazás egy kis kockával jelezni tudja a felhasználó számára, hogy hova mutasson a kamerával ahhoz, hogy az alkalmazás felismerje a QR kódot. A QR kód felismerése után az alkalmazás megjeleníti a QR kód tartalmát egy tárolóban amire, ha rákattint a felhasználó a következő oldalra léphet.

Az utolsó oldal a PointerController, ezen az oldalon a felhasználónak lehetősége van irányítani a virtuális mutatót a Laser Pointer gomb hosszan tartásával és rajzolás funkciót is a lézer mutatóhoz hasonlóan a Draw gomb hosszan tartásával tudja aktiválni. A diák előre, illetve hátra mozgatása két gomb segítségével van megoldva a felhasználói felületen, melyeknek funkcióját egyszeri lenyomással lehet aktiválni. Az utolsó gomb pedig fellel a szerver leállításáért. Ami még látható a felhasználó számára az a gyorsulás merő adatai, emellett, ha a szerver visszaküld üzenetet tesztelés céljából, akkor az is ezen az oldalon jelenik meg a gyorsulásmérő értékei felett.

Az utolsó oldal felületének tervezése során, igyekeztünk minél nagyobb gombokat elhelyezni, könnyen elérhető pozícióba, hogy a felhasználók az alkalmazás használata során, minél kevesebbszer keljen lenézzenek a telefonjukra, evvel is hozzájárulva ahhoz, hogy a bemutató gördülékenyebb legyen. A nagy gombok használatának még egy hatalmas előnye, hogy sokkal kevesebb a mellé kattintások esélye és emellett hamarabb meg is tudja jegyezni a felhasználó a gombok helyzetét a felhasználói felületen.

4.3.4. Telefonos alkalmazás navigációjának megvalósítása

A telefonos alkalmazás navigációja a main dart fájlban van megvalósítva. A navigáció a MaterialApp initialRoute és routes tulajdonság felhasználásával valósul meg. Az initialRoute tulajdonság felel az alkalmazás kezdő képernyőjének a beállításáért. A routes tulajdonság egy map típusú objektum, amely az útvonalakat kulcsként és a hozzájuk tartozó widgeteket értékként menti le egy routes térképbe, amely a 23. ábrán is megfigyelhető, igy téve lehetővé az oldalak közötti navigációt.

```
initialRoute: '/',
routes: {
— '/': (context) => HomeScreen(key: key,title: title,),
— '/QRCodeScan' :(context) =>QRCodeScan(key: key,),
— '/PointerController' :(context) =>PointerController(key: key,title: title,),
```

23. ábra Navigációs útvonalak

A navigációs használatához az útvonalak létrehozása után, az alkalmazás, a Navigator osztály pushNamed metódusa meghívásával végzi el a navigációt, ahogy azt a 24. ábrán látható gomb lenyomása is mutatja.

```
child: MaterialButton(
  onPressed: () {
    Navigator.pushNamed(context, '/QRCodeScan');
  },
```

24. ábra Navigáció használata

4.3.5. QR kód beolvasása

A QR kód beolvasása a már említett QRCodeScan oldalon történik a qr_code_scanner [17] csomag használatával. A QR kód beolvasásához először létrehozunk egy új állapotot (state). Az állapotok szerepe az, hogy a widgetek az alkalmazásában dinamikusak legyenek, így lehetővé téve, hogy reagáljanak a felhasználói interakciókra, és frissítsék a megjelenített adatokat.

A következő lépés egy globális kulcs (GlobalKey) létrehozása. A globális kulcs azért fontos a QR kód beolvasása szempontjából mivel, ennek a kulcsnak a segítségével éri el az alkalmazás a QRView widget-böl a QRViewController-t, amely a kamera vezérléséért és a beolvasott adatok kezeléséért felel. Miután létezik a globális kulcsunk, még szükség van egy Barcode típusú változóra, amibe az alkalmazás tárolja a beolvasott kódot. Az utolsó elem, amit inicializálnunk kell a konstruktorban az a QRViewController, hogy a widgetek-ben változtatni tudjuk a kamera beállításait.

A konstruktor után felül kell írjuk a reassemble függvényt. A reassemble függvény szerepe az alkalmazás újraépítése, ha a hot reload funkcióját szeretnénk használni a Flutter-nek tesztelés során. Azért kell felülírjuk a függvényt, mert Android estén csak akkor működik ez a funkció, ha megállítjuk a kamerát újraépítés után, míg IOS esetén el kell indítani a kamerát a resume függvény használatával.

[17] https://pub.dev/packages/qr_code_scanner

A QR kód beolvasása _onQRViewCreated függvényben történik, amint ezt a 25. ábra is mutatja, a controller.scannedDataStream.listen() függvénye figyeli a beolvasott QR kódokat és elmenti az eredményeket a scanData változóba. Ezt követően a setState() metódusba az alkalmazás átadja a beolvasott kódot a Barcode típusú result változónak és a resultUri-nak. A setState()

```
void _onQRViewCreated(QRViewController controller) {
  this.controller = controller;
  controller.scannedDataStream.listen((scanData) {
    setState(() {
      result = scanData;
      resultUri='${result!.code}';
    });
  });
  controller.pauseCamera();
  controller.resumeCamera();
}
```

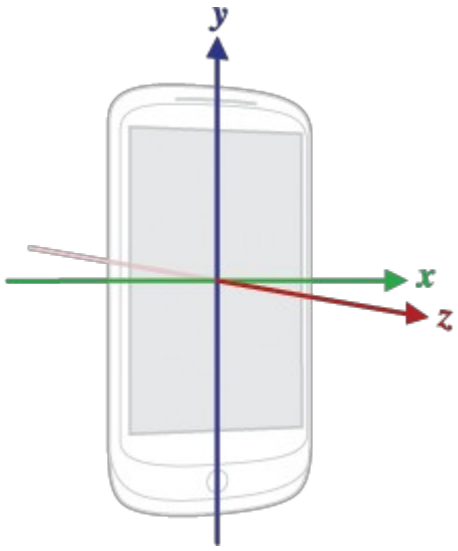
25. ábra QR kód beolvasása

metódusra azért van szükség, mert ez jelzi az alkalmazás számára, hogy frissítse a UI az új adatokkal, amit a controlller beolvasott a result változóba. A resultUri pedig egy globális változó, amely segítségével átadódik a QR kód értéke a következő oldalon létrejövő WebSocket kliensnek. A függvény utolsó feladata a camera újraindítása, hogy hibás érték esetén más QR kód beolvasására is lehetőség legyen.

A result változóban tárolt kód értéke egy konténerben jelenik meg szövegként melyre, ha a felhasználó rákattint átkerülhet az utolsó oldalra, amelyen irányítani lehet a virtuális mutatót. A kattintás érzékelését GestureDetector onTap() függvénye valósítsa meg, úgy hogy a konténert gyermekként adjuk meg a GestureDetector számára.

4.3.6. Gyorsulásmérő szenzor

A gyorsulásmérő szenzor[18] értékeit Android rendszerben egy standard három tengelyű koordináta rendszerben lehetne megjeleníteni. A telefon elhelyezkedése ezen a koordináta rendszeren a 26. ábra szemlélteti[19]. Ahogy az ábrán is megfigyelhető, az y koordináta mutat felfelé



26. ábra Telefon elhelyezkedése a koordináta rendszeren

az x koordináta a telefon jobb felére mutat, míg a z koordináta a telefon képernyőjétől felénk. A rendszerünk esetében mi a x, illetve y koordinátát fogjuk felhasználni. A x koordináta fogja szolgáltatni a mutató jobbra, illetve balra mozgatását, míg a y koordináta szerepe a mutató fel, illetve lefele történő mozgatása lesz.

Egy fontos tulajdonsága gyorsulásmérő szenzor használatának Android-os készülékek esetében, hogy a szenzor mintavételi sebessége[20] limitálva van 200 Hz. Ezt a limitálást biztonsági okokból vezették be, hogy a felhasználók potenciálisan érzékeny információikat megvédhessék.

A gyorsulásmérő szenzor nem csak a telefonunk elmozdulásai sebességét méri, hanem a mért adat tartalmazza a gravitációs erőt. Ahogy ezt a 27. ábra is mutatja[21] a gyorsulásmérő, úgy

$$A_D = -g - (\frac{1}{mass}) \sum F_S$$

27. ábra Telefonra ható erő

---

[18] https://developer.android.com/guide/topics/sensors/sensors_overview#sensors-coords

[19] https://developer.android.com/static/images/axis_device.png

[20] https://developer.android.com/guide/topics/sensors/sensors_overview#sensors-rate-limiting

[21] https://developer.android.com/static/images/guide/topics/sensors/acceleration_with_grav.png

határozza meg a telefonra (Ad) ható gyorsulást, hogy megméri a szenzorra (Fs) ható erőt, viszont a mért erőt mindig befolyásolja a gravitációs erő is[22].

Viszont ezt a tulajdonságot az előnyünkre is tudjuk fordítani, mivel, ha nem vennénk figyelembe a gravitációs erőt, akkor nem tudnánk meghatározni a telefon dőlve tartásakor, hogy mi történik pontosan, mivel a szenzor, a kezdetleges mozdulat észlelése után, nem tudná megállapítani a telefon helyzetét. Viszont, ha figyelembe vesszük a gravitációs erőt, akkor a szenzorra történő gravitációs erő eloszlásából a három tengelyen figyelni tudjuk a telefon elhelyezkedését és így tudva azt például, hogy jobbra van dőlve a telefon, mivel az x koordinátán mért gravitációs erő értéke közelítené a $-9.8\ m/s^2$,ugyanígy balra döntés esetén, pedig közelítené a $9.8\ m/s^2$. Hasónlóképen a telefon felfele, illetve lefele mutatását is mérni tudjuk, úgy, ha y koordináta értéke közelíti a $9.8\ m/s^2$ értéket és a lefele mutatás esetén közelíti a $-9.8\ m/s^2$.

Az így mért értékek felhasználásával tudjuk mozgatni a virtuális mutatónkat, mivel x és y értékeket meg tudjuk feleltetni a mutató pixelben lévő helyzetéhez, így akár milyen mozgásra képes a mutató a diákon a két koordináta segítségével.

[22] https://developer.android.com/guide/topics/sensors/sensors_motion#sensors-motion-accel

4.3.7. Mutató irányítása és a funkciók megvalósítása

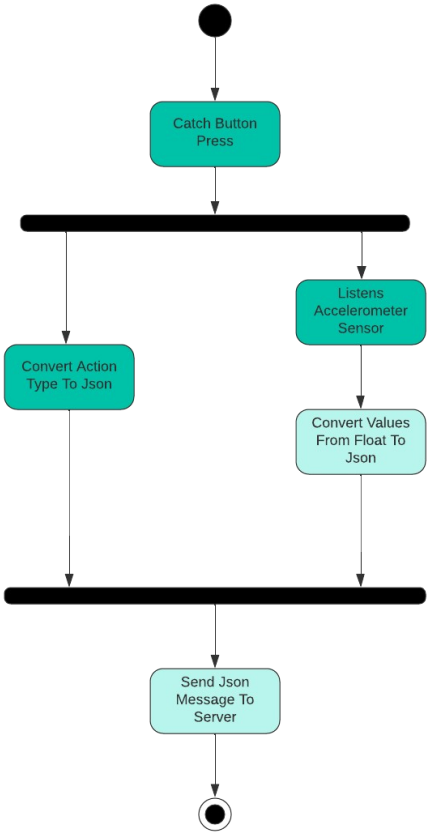
A mutató irányításához először a rendszernek kapcsolódnia kell a számítógépen futó WebSocket szerverhez. A szerverhez való csatlakozás 28. ábrán is megfigyelhető

```
class _PointerControllerState extends State<PointerController> {
  final TextEditingController _controller = TextEditingController();
  final _channel = WebSocketChannel.connect(
    Uri.parse('ws://$resultUri'),
  );
```

28. ábra WebSocket szerverhez csatlakozás

_PointerControllerState konstruktorában történik meg WebSocketChannel.connect() függvénnyel, mely a resultUri-ban tárolt érték felhasználásával és annak a megfelelő formázásával csatlakozik a szerverhez. A resultUri egy globális változó, amely a QR kód beolvasás után kap értéket.

A virtuális mutató irányításához szükséges üzenet létrehozásának a lépéseit a 29. ábra szemlélteti. Az ábrán megfigyelhető, hogy a kommunikációs kapcsolat létrejötte után, az első



29. ábra Virtuális mutató irányításához használt üzenet létrehozása

állapot, ahova kerül a telefonos alkalmazás az a felhasználó által lenyomott gomb lenyomásának a kezelése. Ezt követően kezdi el a szenzor adatainak feldolgozását és a parancsok átalakítását a szervernek való elküldés céljából.

A mutató irányításához szüksége van az alkalmazásnak a telefon szenzorjaihoz való hozzáféréshez. Ezekhez a szenzorokhoz a sensors_plus [23] csomag felhasználásával fér hozzá az alkalmazás. A szenzor adatainak lekérése előtt, ami az alkalmazás esetében a telefon gyorsulásmerőjének az értékei, szükség van egy double típusú listára, melynek a neve _accelerometerValues, amelyben tárolja a rendszer az adatokat. Egy elengedhetetlen lépés az adatok lekérése előtt egy StreamSubscription lista létrehozása. A Stream segítségével tudja figyelni az alkalmazás aszinkron módon a szenzor adatait, így frissítve az adatokat a felhasználói felületen és a szerver által küldött értékek is a stream használatával frissülnek.

Az alkalmazás a gyorsulásmérő értékeit a lisenSensorData() függvényben kapja meg, melynek a felépítése a 30. ábrán is látható. A függvényt az alkalmazás automatikusan meghívja, amikor a felhasználó a PointerController oldalra lép a initState() függvény felülírásával.

```
void lisenSensorData() {
  _streamSubscriptions.add(
    accelerometerEvents.listen(
      (AccelerometerEvent event) {
        setState(() {
          _accelerometerValues = <double>[event.x, event.y, event.z];
          response = json.encode(
              {'type': 'points', 'x': event.x, 'y': event.y, 'z': event.z});
        });
      },
    ),
  );
}
```

30. ábra Gyorsulásmérő adatainak lekérése

A lisenSensorData() függvényben először az alkalmazás feliratkozik a folyamra a _streamSubscriptions.add() függvénnyel, melyben a gyorsulásmérő adatait figyeli az accelerometerEvents.listen() függvényen keresztül. Az event osztályba tárolódnak a gyorsulásmérő értékei három koordinátaként. Ezeket az értékeket egy double típusú listába elmenti

[23] https://pub.dev/packages/sensors_plus

az alkalmazás, hogy kiírható legyen a felhasználói felületen, emelet pedig egy response változóba elmenti Json formátumban az értékeket a szervernek való elküldés céljából. Az értékek Json formátumba való átalakítás azért fontos, mert, így sokkal könnyebben feldolgozhatóak az adatok a szerver oldalon és emellett nagyban növeli az adatok átláthatóságát is.

A gyorsulásmérő adatainak lekérése után az alkalmazás következő lépése az adatok elküldése a szervernek a felhasználó által választott akciókkal együtt. Mindegyik akció egy

```
void _sendMessage() {
  _channel.sink.add(response);
}

void _sendMessageStop() {
  var responseAction = json.encode({'type': 'actions', 'action': 'stop'});
  _channel.sink.add(responseAction);
}
```

31. ábra Üzenetek küldése a telefonos alkalmazásból

függvénynek felel meg az alkalmazásban, ahogy ezt a 31. ábrán látható két akció is mutatja. Az akciók elküldése a _channel.sink.add() függvényhívással történik. A _channel amint az a 28. ábrán is látható tárolja a WebSocket kapcsolatot, amelyen keresztül az alkalmazás kommunikál a szerverrel.

A felhasználói felületen a függvényeket, melyek a különböző akciók, illetve a pontok elküldésért felelnek gesztusérzékelők (GestureDetector) és a hozzájuk tartozó konténerek (Container), valamint gombok felhasználásával vannak meghívva.

A mutató irányításához gesztusérzékelő onLongPressStart és onLongPressEnd funkcióját használja az alkalmazás, amelyek lehetővé teszik, az alkalmazás számára, hogy figyelje a felhasználó mozdulatait, így tudva azt, hogy mikor van a gomb hosszan lenyomva tartva és mikor engedi fel a gombot a felhasználó. A gomb lenyomva tartásakor, a lézermutató esetében az alkalmazás, meghívja a _sendMessageCalibrate() függvényt, amely elküld egy kalibrálási akciót a szervernek Json formátumban. Ahogy azt már a számítógépes alkalmazásnál is említettük a kalibrálás során a mutató kezdő pontja a képernyő közepére kerül. A mutató kalibrálása azért szükséges, hogy a felhasználó számára biztosítsunk egy kezdő pontot, mivel az egér mutató mozgását emulálja az alkalmazás, így az egér mutató előző helyzete lenne az indulási pont, amely nem minden szituációban lenne átlátható a felhasználó számára. A következő függvény ami

meghívódig a _sendMessageStartLaserPointer() függvény, amely segítségével jelzi a telefonos alkalmazás a számítógépes alkalmazásnak, hogy a felhasználó a lézer mutatót szeretné használni. Ezt követően egy feltételes while ciklusban aszinkron módon a _sendMessage() függvénnyel folyamatosan küldi a gyorsulásmérő értékeit a szervernek. Azért szükséges aszinkron módon küldeni az adatokat, hogy a felhasználói felület reszponzív maradjon. Az adatok küldése akkor ér véget, ha a felhasználó felengedi a gombot, így jelezve egy változón keresztül a ciklusnak, hogy érjen véget. Minden ciklusban az alkalmazás a gyorsulásmérő adatainak az elküldése után vár száz ezredmásodperc, hogy elkerüljük a szerver túlterhelését.

A lézer mutató leállítása a gomb felengedésekor hajtódik végre, amely gesztusérzékelővel és konténer használatával van megvalósítva. A felengedéskor a _sendMessageStopLaserPointer() függvény egy utolsó üzenettel jelzi a szervernek, hogy kapcsolja ki a lézer mutató funkciót és a setState() függvényt használva az alkalmazás az isPressed változó értékét hamisra állítja, aminek a hatására a feltételes while ciklus megáll, így a gyorsulásmérő adatainak az elküldése is megáll.

A rajzolás funkciója is hasonló módon működik, mint a már említett lézer mutató, ahogy ezt a 32. ábrán lévő kód részeleten is látható. Az egyik nagy különbség a két funkció között, hogy a rajzolás esetén nem küldünk kalibrálási kérést a szervernek. Azért döntöttünk a mellet, hogy ne küldjünk kalibrálást rajz esetén, hogy amikor a felhasználó rajzolni szeretne könnyen odatudja vinni a lézer mutató segítségével a kurzort a kívánt helyre és utána rajzolás gomb lenyomásával akár ki is tudja jelölni azt a diákon.

A diák előre, illetve hátra vitele emelt gomboknak (ElevatedButton) a lenyomásakor _sendMessageLeftKey() függvény vagy a _sendMessageRightKey() függvény meghívásával történik meg, amelyek _channel.sink.add() függvénnyel elküldenek egy üzenetet, amely jelzi a szervernek, hogy melyik parancsot hajtsa végre.

Az utolsó funkció a szerver leállításának lehetősége a telefonos alkalmazásból. Ennek a funkciónak a megvalósítása lebegő gomb (FloatingActionButton) lenyomásával történik. A gomb lenyomásakor _sendMessageStop függvény hívódik meg, amelynek a tartalma a 31. ábrán látható. Ez a függvény is egy Json stringet küld a szervernek, így adva át a kívánt parancsot, amelyet a szerver az üzenet feldolgozásakor végrehajt.

A WebSocket kliens leállítása az alkalmazás bezárásakor vagy az oldal elhagyásakor történik meg a dispose() függvény felülírásával. A függvényben a _channel.sink.close() függvénnyel lezárjuk a kapcsolatot a szerver és a kliens között, emellett ebben a függvényben kell lezárjuk a stream-t subscription.cancel() függvény használatával, amely a gyorsulásmérő adatainak a lekérdezéséért felelnek.

```
child: GestureDetector(
    onLongPressStart: (_) async {
      _sendMessageStartDraw();
      isPressed = true;
      do {
        _sendMessage();
        await Future.delayed(const Duration(milliseconds: 100));
      } while (isPressed);
    },
    child: Container(
      width: 150,
      height: 150,
      color: Colors.blue,
      padding: const EdgeInsets.all(12),
      child: const Center(
        child: Text(
        "Draw",
        style: TextStyle(
          fontWeight: FontWeight.bold,
          fontSize: 15,
        ),  // TextStyle
      )),  // Text, Center
    ),  // Container
    onLongPressEnd: (_) {
      _sendMessageStopDraw();
      setState(() => isPressed = false);
    }),  // GestureDetector
```

32. ábra Rajzolás funkció meghívása

5. Összefoglaló

A dolgozatom során sikerült egy olyan rendszert kivitelezni, amely használatával a tanárok vagy más emberek, akiknek a feladatkörének kulcsfontosságú része a prezentációk bemutatása, egy olyan rendszert biztosítani, amely nagyban megkönnyíti a prezentációk bemutatását és egyúttal egy szabadabb bemutatási módot is biztosít a hagyományos módszerekhez képest.

A rendszer két komponensből épül fel: az első komponens a telefonos alkalmazás, amely a felhasználó kézmozdulatait rögzíti, illetve küldi el a számítógépes alkalmazásnak és a számítógépes alkalmazás, amely a küldött adatokat dolgozza fel és irányítja a virtuális mutatót.

A rendszer fő céljául kitűzött virtuális mutató mellett, sikerült megvalósítani a diákon történő rajzolás funkció implementálását és a diák előre, illetve hátra való mozgatásának a lehetőségét. Emellett még sikerült egy olyan felhasználói felületet létrehozni, amelynek használata rövid idő alatt is könnyen megtanulható.

A rendszer verziózására is sor került a GitHub rendszer használatával, így a következő GitHub oldalon[24] megnézhető a rendszer fejlesztésének lépései, illetve le is tölthető a rendszer használatra.

Összességében sikerült egy olyan rendszert lefejleszteni, amely egy könnyen használható alternatívát biztosíthat az embereknek a prezentációk bemutatására a telefonjuk segítségével.

5.1. Továbbfejlesztési lehetőségek

A rendszer továbbfejlesztésére elég sok lehetőség van. Az egyik legfontosabb továbbfejlesztési lehetőség az IOS-es alkalmazás véglegesítése és tesztelése. Ez a folyamat elég könnyen megvalósítható a továbbiakban, mivel a telefonos alkalmazásához a Flutter keretrendszert választottuk, így pár tesztelés során működőképes is lehet az alkalmazás.

A rendszer funkcionalitásait is lehet bővíteni, például egy radír funkció implementálásával vagy a módosított diák mentésének a lehetőségének a megoldásával. A radír funkció azért lenne hasznos, hogy a diákra rajzolt elemeket eltüntessük a telefonunk segítségével a helyet, hogy a számítógéphez kellene menjen a felhasználó.

Egy hasznos funkció lehetne még a diák alatt lévő jegyzet megjelenítése a felhasználó számára a telefonon, hogy a kulcs fontosságú pontokat, amiket át akar adni a hallgatóknak láthassa a bemutatás során.

[24] https://github.com/SzaszNimrod12/PointerApp

6. Irodalomjegyzék

[1] Katt, J., Murdock, J., Butler, J., & Pryor, B. (2008). Establishing best practices for the use of PowerPoint™ as a presentation aid. Human Communication, 11(1), 193-200.

[2] Szabo, A., & Hastings, N. (2000). Using IT in the undergraduate classroom: should we replace the blackboard with PowerPoint?. Computers & education, 35(3), 175-187.

[3] Alkash, K. A. M., & Al-Dersi, Z. E. M. (2017). Advantages of using PowerPoint presentation in EFL classroom & the status of its use in Sebha University. Tersedia http://eltsjournal. org/upload/2014-05-13.

[4] Katai, Z., & Iclanzan, D. (2022). Impact of instructor on-slide presence in synchronous e-learning. Education and Information Technologies, 1-27.

[5] Tkachuk, V., Yechkalo, Y., Semerikov, S., Kislova, M., & Khotskina, V. (2020). Exploring student uses of mobile technologies in university classrooms: Audience response systems and development of multimedia. CEUR Workshop Proceedings.

[6] Goh, J. E. E., Goh, M. L. I., Estrada, J. S., Lindog, N. C., Tabulog, J. C. M., & Talavera, N. E. C. (2018). Presentation-aid Armband with IMU, EMG sensor and bluetooth for free-hand writing and hand gesture recognition. International Journal of Computing Sciences Research, 1(3), 65-77.

[7] Budiman, A., & Triono, J. (2016). Smart and Simple Classroom Presentation Tools. International Journal of Advanced Research in Computer Science, 7(7).

[8] Fette, I., & Melnikov, A. (2011). The websocket protocol (No. rfc6455).

[9] Srinath, K. R. (2017). Python–the fastest growing programming language. International Research Journal of Engineering and Technology, 4(12), 354-357.

[10] Sanner, M. F. (1999). Python: a programming language for software integration and development. J Mol Graph Model, 17(1), 57-61.

[11] Zammetti, F. (2019). Practical Flutter. Berkeley, CA: Apress.

[12] Windmill, E. (2020). Flutter in action. Simon and Schuster.

[13] Bollini, L. (2017). Beautiful interfaces. From user experience to user interface design. The Design Journal, 20(sup1), S89-S101.

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA

FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE, TÎRGU-MUREȘ

SPECIALIZAREA CALCULATOARE

Vizat decan

Conf. dr. ing. Domokos József

Vizat director departament

Ș.l. dr. ing Szabó László Zsolt