UNIVERSITATEA POLITEHNICĂ BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Smart Presentation Feedback

Interfața cu utilizatorul

Coordonatori științifici :

Prof. dr. ing. Adina Magda Florea

As. dr. ing. Andrei Olaru

Absolvent:

Anca-Virginia Pîrvan

BUCUREŞTI

2012

Cuprins

Rezumat

Proiectul Smart Presentation Feedback constă în implementarea unei aplicaţii inteligente pentru trimiterea feedback-ului de la auditoriu la vorbitor, în timpul unei prezentări. Persoanele din sală pot trimite feedback prezentatorului sub formă de întrebări, aprecieri pozitive sau negative, folosind opţiunile disponibile. La finalul prezentării, vorbitorul are posibilitatea de a vedea feedback-ul primit într-o formă agregată; selecţiile făcute de auditoriu au culori diferite în funcţie de tipul de feedback ales. Observaţiile primite au rolul de a-l ajuta pe vorbitor să îşi îmbunătăţească prezentările viitoare. În această lucrare este descrisă o aplicaţie al cărei scop principal este eficientizarea comunicării între sală şi prezentator, dar şi între persoanele din sală. Aceasta se adresează în primul rând persoanelor care participă la o prezentare despre un anumit subiect şi doresc să urmărească cu uşurinţă opiniile fiecărui participant. Contribuția mea principală la acest proiect a fost implementarea interfeței, urmărind o interacțiune cât mai bună cu utilizatorul.

1. Introducere

1.1 Contextul proiectului

Dispozitivele mobile cu Android au cunoscut o foarte mare dezvoltare în ultimii ani, atingând o cotă de piaţă majoritară în anul 2012 (Figura 1). O dată cu evoluția telefoanelor mobile, s-au extins şi aplicațiile dezvoltate folosind sistemul de operare Android.

Ca urmare a acestei evoluții a sistemului de operare Android şi a aplicațiilor pe dispozitive mobile, proiectul meu se încadrează în aceeași zonă de interes şi constă în implementarea unei aplicații inteligente pentru trimiterea feedback-ului sub formă de întrebări, aprecieri pozitive sau negative, unui vorbitor care susține o prezentare, curs, lucrare sau prezintă un produs care trebuie lansat pe piaţă internă sau internațională. La finalul prezentării, vorbitorul are posibilitatea de a vizualiza într-o formă agregată feedback-ul primit de la auditoriu şi de a răspunde la întrebările şi observaţiile primite pe parcurs.

Proiectul Smart Presentation Feedback este parte a proiectului Google Android EDU şi a fost implementat în colaborare cu Stoica George-Cristian şi Dincă Dragoş, sub îndrumarea şi supravegherea As. dr. ing. Andrei Olaru şi Ing. Tudor Berariu.

1.2 Ideea şi scopul proiectului

Pe parcursul unei prezentări care conţine slide-uri cu informaţii complexe (ex: formule matematice, algoritmi, demonstraţii ale unor teoreme) ce solicită concentrarea persoanelor din auditoriu un timp îndelungat, poate interveni plictiseala şi lipsa de atenţie a publicului din diverse motive. Câteva din aceste motive ar putea fi: vorbitorul prezintă într-un ritm alert informaţiile, iar persoana din auditoriu nu are suficient timp pentru a înţelege ideea transmisă; până la finalul prezentării, când poate adresa o întrebare, este posibil ca persoana din sală să îşi piardă ideea.

Aplicaţia dezvoltată de noi are ca scop principal evitarea acestor probleme care pot apărea pe parcursul unei prezentări, sporirea atenţiei publicului şi eficientizarea comunicării dintre sală şi prezentator. Este dezvoltată pe dispozitive mobile cu Android, adresându-se în primul rând utilizatorilor, dar şi dezvoltatorilor de aplicații Android.

Folosind această aplicație, utilizatorii şi speakerul vor avea o mai bună comunicare. Persoanele din sală vor putea urmări prezentarea în propriul lor ritm, nefiind nevoite să ţină pasul cu vorbitorul şi putând aloca timpul dorit fiecărui slide, vor putea face adnotări pe parcurs, evitând astfel posibilitatea pierderii ideilor până la finalul prezentării/lecției.

Speakerul va putea răspunde, la finalul prezentării, la întrebările pe care le primește în feedback-ul agregat, în ordinea importanţei şi a numărului de întrebări de același tip.

Scenariul de funcționare al proiectului: Anca susține o prezentare în faţa colegilor săi de la cursul de IA. În timp ce ea parcurge prezentarea, colegii din audienţă pot să navigheze independent prin slide-uri, să facă adnotări, să scrie întrebări, să dea feedback pozitiv unui text/imagini folosind butonul `+1`. Cristi nu înțelege cum a aplicat Anca o formulă pe un anumit slide şi vrea să noteze o întrebare în legătură cu acest lucru, dar observă că un alt coleg a pus deja aceeaşi întrebare. Cristi poate doar să apese butonul +1 pentru a susține că are aceeași întrebare. În cazul în care Cristi vrea să primească o explicaţie detaliată legată de aplicarea unei formule sau a unui algoritm, el poate folosi unul din butoanele `ambigous` sau `citation or proof needed`. La finalul prezentării, Anca poate vedea feedback-ul agregat pe care l-a primit de la colegi şi poate răspunde la întrebări şi observaţii. Analizând feedback-ul primit, Anca își poate îmbunătăți prezentările viitoare. [1]

1.3 Structura proiectului

Proiectul este împărțit în patru module principale, fiecare din cei implicați în dezvoltare se ocupă de unul din aceste ele:

- Interfața cu utilizatorul
- Selecția din PDF
- Modulul client – server
- Clusterizarea întrebărilor

Eu m-am ocupat de partea de interfață cu utilizatorul care este împărțită în interfața audienţei şi interfața vorbitorului. Aceste interfețe au câteva funcționalități comune. Acest modul constă, în principal, în implementarea funcţionalităţilor şi design-ului butoanelor care compun cele două interfeţe.

1.4 Structura documentului

Documentul este împărțit în 8 capitole, fiecare capitol fiind structurat pe mai multe subcapitole. În cele ce urmează va fi prezentată modalitatea de realizare a aplicației, subliniind problemele întâlnite şi modalităţile de rezolvare a acestora.
În capitolul 2 voi prezenta câteva aspecte teoretice legate de interfaţa cu utilizatorul, ergonomie şi principiile ergonomice.
Capitolul 3 este o scurtă prezentare a sistemului de operare Android, la modul general, dar şi o prezentare a tehnologiilor Android, a tehnologiilor client – server şi a altor tehnologii pe care le-am utilizat pentru implementarea proiectului.
Capitolul 4 descrie arhitectura şi funcţionalităţile proiectului, modulele sistemului şi relaţiile dintre acestea.
Capitolul 5 este o prezentare a modului de implementare a intefeţelor clientului Android, cuprinzând detalii tehnice şi fragmente din codul propriu.
Capitolul 6 prezintă în detaliu aplicaţia dezvoltată şi cuprinde imagini sugestive ale funcţionalităţilor existente.

2. Aspecte teoretice

Interacțiunea dintre utilizatori şi calculatoare apare la nivelul interfeței cu utilizatorul, care include aspecte ergonomice, software şi hardware. La calculatoarele moderne există următoarele puncte centrale ale interfeței om-calculator: interfața grafică (pe un monitor sau ecran de calculator), interfața grafică mijlocită prin atingerea monitorului (touch screen) , interfața grafică prin comenzi verbale (recunoașterea vocii).

Interfața este acea parte a aplicației software prin intermediul căreia utilizatorul interacționează cu calculatorul, având posibilitatea de a-şi exprima intențiile de operare şi de a interpreta rezultatele efectuate de mașină. Interfața nu este concepută doar ca parte vizuală a software-ului, pentru majoritatea utilizatorilor reprezintă întregul sistem de calcul. Orice interfață poate fi utilă, utilizabilă şi utilizată.

Brazier considera că interfaţa defineşte modul în care utilizatorul trebuie să interacţioneze cu sistemul pentru a delega sistemului sarcinile unitate specifice. Utilizatorul traduce sarcinile unitate în sarcini de bază, ce sunt implementate în interfața utilizator.[2]

Un aspect deosebit de important în definirea interfeței utilizator este faptul că interfaţa reprezintă din sistemul complex utilizatorului, doar aspectele relevante pentru interacţiunea sa cu sistemul. Celelalte aspecte care există în sistem dar nu sunt accesibile utilizatorului, nu îi influențează acestuia interacțiunea cu sistemul. Van der Veer a numit interfaţa şi maşina virtuală a utilizatorului.[2]

O interfață - utilizator este bine scrisă atunci când programul se comportă exact așa cum se așteaptă utilizatorul.

Proiectarea interfețelor cu utilizatorul este rezultatul activităților de: înțelegere a nevoilor utilizatorilor, proiectare (design), evaluare/ testare, implementare finală, menținere.

Ergonomia este disciplina științifică ce studiază interacțiunea dintre oameni şi alte elemente ale unui sistem, precum şi profesia care aplică teorii, principii, informații şi metode de design pentru optimizarea activității omului şi performanţele sistemului din care acesta face parte [definiție adoptată în august 2000 de către consiliul director al Asociației Internaționale de Ergonomie]. Denumirea de ergonomie derivă din cuvintele grecești ergon (=muncă) şi nomos (=reguli) pentru a exprima știința muncii şi se aplică tuturor domeniilor de activitate. Ergonomia promovează o abordare care ia în considerare aspectele fizice, cognitive, sociale, organizaționale, de mediu şi alți factori importanți. Pentru aceasta, ergonomia integrează cunoștințe dintr-o varietate de discipline care includ: anatomia, fiziologia, științe tehnice, psihologie, sociologia, economie etc.

Ramurile principale ale ergonomiei sunt: ergonomia fizică, ergonomia cognitivă şi ergonomia organizațională.

Ergonomia fizică se referă la modul de raportare la activitatea fizică a caracteristicilor anatomice, fiziologice şi biomecanice ale omului. Domenii de studiu: posturi de lucru, manipularea obiectelor, mișcări repetitive, designul locului de muncă.

Ergonomia cognitivă se referă la modul în care procesele mentale, cum ar fi percepțiile, memoria, logica, răspunsurile motorii, influențează interacțiunile dintre oameni şi alte elemente ale unui sistem. Domenii de studiu: interacțiunea om-calculator, suprasolicitarea neuropsihică, luarea deciziilor, obținerea performanţei, stresul la locul de muncă, pregătirea.

Ergonomia organizațională este ramura ergonomiei care se preocupă de optimizarea sistemelor sociotehnice, incluzând structurile organizaționale, politicile şi procesele. Domenii de studiu: comunicarea, managementul resurselor, stabilirea orarului de muncă, munca în echipă, ergonomia comunităților, noi paradigme în muncă, organizații virtuale, teleactivitatea.

Ergonomia este definită şi ca ştiinţă şi tehnologie interdisciplinară având ca obiect adaptarea reciprocă între oameni, maşini şi mediu în cadrul interacţiunii lor ca sisteme [3]. Ergonomia apare deci ca un demers sistematic de achiziţie şi organizare a cunoştinţelor despre om în scopul utilizării lor în conceperea instrumentelor de lucru, în ameliorarea condiţiilor activităţii pentru reducerea efortului şi creşterea confortului, securităţii şi eficacităţii. Continuând tradiţia ergonomiei sistemelor om-maşină, **ergonomia cognitivă** se concentrează în mod particular asupra interacţiunii dintre om şi mediul activităţii sale cognitive, concentrându-se pe procesele de înţelegere, raţionament şi utilizare a cunoştinţelor.

Suportul fizic al interacţiunii om-calculator este reprezentat de **interfaţă**, adică totalitatea elementelor fizice reale sau virtuale şi a programelor de calculator implicate în acest proces. Alături de interfaţa propriu-zisă şi caracteristici specifice maşinii, interacţiunea om-calculator implică şi un context de utilizare (social, organizaţional), precum şi trăsături specifice omului (cunoştinţe, deprinderi, atitudini, aspecte afective şi motivaţionale etc).

Unul dintre **scopurile ergonomiei cognitive** este tocmai obţinerea unei tehnologii comprehensibile şi utilizabile. Interfaţa om-calculator este utilizabilă dacă consideră următoarele condiţii: utilizarea ei este uşor de învăţat şi reamintit, este eficientă, adică se pot efectua rapid acţiuni complexe, este consistentă, unitară, flexibilă, confortabilă. Proiectarea interfeţelor utilizabile, prietenoase şi personalizate impune considerarea mai multor **principii** de ergonomie cognitivă: principiul **coerenţei**, caracterul unitar al constituenţilor interfeţei, principiul conciziunii (reducerea efortului cognitiv), asigurarea unei economii cognitive, principiul conexiunii inverse (asigurarea feedback-ului), asigurarea unei reacţii la orice interacţiune pentru a furniza informaţii utilizatorului

asupra funcţionării sistemului în scopul înţelegerii cât mai uşoare a stării curente şi detectarea situaţiilor nedorite [4].

Criteriile ergonomice constituie un mijloc de încorporare a utilizabilității în proiectul unei interfețe, ajută la structurarea şi completarea unui set de indicatori şi măsuri pentru evaluare şi la elaborarea unor reguli de proiectare a interfețelor. Unui criteriu ergonomic îi corespund una sau mai multe reguli de proiectare. Mai jos voi analiza câteva dintre criteriile ergonomice:

- **Ghidarea utilizatorului** se referă la mijloacele de orientare, informare şi ghidare a utilizatorului pe parcursul interacţiunii om-calculator. Mijloacele de ghidare sunt diverse: etichete, mesaje, alarme etc.
  Ghidarea facilitează uşurinţa în învăţare şi utilizarea sistemului, având ca efect reciproc performanţe mai bune în execuţie şi reducerea numărului de erori.
- **Efortul utilizatorului** se referă la toate elementele interfeţei care au un rol în reducerea încărcării perceptuale sau cognitive a utilizatorului şi în creşterea eficienţei dialogului. Cu cât utilizatorul este mai încărcat, cu atât mai mare este probabilitatea de a comite erori. Nu este recomandabilă distragerea utilizatorului prin afişarea unei informaţii care nu este necesară, întrucât aceasta constituie o piedică în îndeplinirea cu eficienţă a sarcinii.
- **Controlul explicit** se referă atât la procesarea de către sistem a acţiunilor explicite ale utilizatorului cât şi la controlul pe care utilizatorul îl are asupra procesării acţiunilor sale de către sistem. Atunci când utilizatorul defineşte în mod explicit intrările şi când intrările se află sub controlul său, erorile şi ambiguităţile sunt limitate.
- **Adaptabilitatea** se referă la capacitatea unui sistem de a se comporta contextual şi în concordanţă cu nevoile şi preferinţele utilizatorului.
- **Tratarea erorilor** se referă la mijloacele disponibile pentru a preveni şi/sau reduce erorile şi de a permite recuperarea în caz de eroare. Erorile se referă la intrări de date invalide, erori de sintaxă etc. Întreruperea procesării ca urmare a erorilor utilizatorilor are efecte negative asupra activităţii utilizatorilor. Prin limitarea numărului de erori, numărul de întreruperi este de asemenea limitat.
- **Consistenţa** se referă la modul în care opţiunile de proiectare ale interfeţei (coduri, denumiri, formate, proceduri etc.) sunt menţinute în contexte similare şi sunt diferite atunci când sunt aplicate unor contexte diferite. Consistenţa este un criteriu elementar.
- **Semnificaţia** codurilor califică relaţia dintre un termen şi/sau un semn (simbol) şi referinţa acestuia. Codurile şi denumirile sunt semnificative pentru utilizator atunci când există o relaţie semantică puternică între aceste coduri şi articolele sau acţiunile la care se referă. Semnificaţia codurilor este un criteriu elementar.
- **Compatibilitatea** se referă la potrivirea dintre caracteristicile utilizatorului (memorie, percepţie, obiceiuri, vârstă, aşteptări) şi ale sarcinii, pe de o parte, şi organizarea intrărilor, ieşirilor şi dialogului pentru o aplicaţie dată, pe de altă parte. De asemenea, compatibilitatea se referă la coerenţa dintre medii şi dintre aplicaţii. [5]

Știința factorilor umani poate fi numită uneori **ergonomie** şi se referă la arta de a asigura aplicarea cu succes a ingineriei factorilor umani într-un program. În general, **un factor uman** este o proprietate fizică sau cognitivă a unui individ sau comportament social, care este specifică influenţei omului asupra funcționării sistemelor tehnologice şi echilibrului dintre om şi mediu. Ia în calcul factori legați de: vizibilitate, feedback, consistenţă (nu induce riscul unor confuzii cauzate de contradicții apărute între diferite aspecte interpretate), restricții şi indicații. Exemple: indicații pentru a clarifica modalitatea de folosire, restricții în aplicarea anumitor soluții, inconsistenta (tastatura numerică la calculator vs. telefon mobil).

În cadrul interacțiunii om-calculator, ramura care se ocupă cu aspectele psihologice cognitive ale interacțiunii om-calculator este ergonomia cognitivă. Menirea ergonomiei cognitive este de a adapta instrumentele cognitive, precum şi utilizarea lor în așa fel încât să îmbunătățească procesarea informației umane în termenii îmbunătățirii eficienţei interacțiunii, scăderii ratei de erori şi accidente şi crearea unui sentiment de confort.

Interfața trebuie să asigure o comunicare cât mai facilă. O interfață este utilizabilă dacă şi numai dacă folosirea ei este ușor de învățat şi reamintit, e eficientă (se execută rapid acțiuni complexe), e flexibilă, confortabilă, consistentă şi unitară.

Ergonomia interfețelor căută modalitățile cele mai bune de a realiza interfețe prietenoase, personalizabile şi utilizabile.

Ergonomia cognitivă se ocupă de aspectele mentale şi psihice ale ergonomiei interfețelor, nu de elementele fizice cum este în cazul ergonomiei clasice. Modelul mental indus de o interfață poate diferi de la utilizator la utilizator pentru aceeași interfață.

Ergonomia cognitivă e strâns legată de modelul utilizatorului (modelul procesorului uman), profilele cognitive şi starea de flux.

Modelul utilizatorului – interfața trebuie să preia opțiunile/scopurile/convingerile utilizatorului şi să le reprezinte prin acest model pentru a fi adaptabilă. În acest scop trebuie studiată psihologia umană şi semiotica (știința semnelor).

Interfața utilizator poate fi descrisă prin utilizarea unor **modele conceptuale**. Pentru a realiza un model conceptual al sistemului se utilizează diferite metode de modelare. Modelul conceptual poate fi abordat din trei perspective: lingvistică, psihologică şi a proiectării.

Modelele cognitive (perspectiva psihologică) sau arhitecturile cognitive sunt modele ale utilizatorilor dezvoltate în special de psihologii cognitiviști pentru a descrie procesele pe care le folosesc oamenii pentru a rezolva anumite sarcini, cum sunt rezolvarea de probleme sau utilizarea unui sistem complex.

3. Tehnologii folosite

3.1 Sistemul de operare Android. Tehnologiile Android folosite

Pentru dezvoltarea aplicaţiei am ales sistemul de operare Android, în defavoarea celorlalte tehnologii de pe piaţă ca Windows Mobile sau iPhone, mai ales datorită faptului că este cea mai răspândită tehnologie pe dispozitivele mobile actuale, fiind totodată o platformă open-source.

Android este în prezent cel mai popular sistem de operare dintre cele dedicate dispozitivelor portabile. A fost construit în jurul unui nucleu Linux, la fel ca şi Chrome OS sau MAC OS, de către o companie anonimă Android Inc. Şi achiziționat de Google în anul 2005. În anul 2007, s-a înființat consorțiul comercial Open Handset Alliance pentru a stimula evoluția în domeniul dispozitivelor portabile; în același an a fost anunțat sistemul de operare Android. La sfârșitul anului 2008, a apărut pe piaţă primul terminal cu Android 1.0, HTC Dream, care oferea printre facilități : Wi-Fi, integrarea serviciilor Google, Android Market (foarte diferit de varianta actuală). O dată cu apariția versiunii Android 1.1 a fost rezolvat un număr foarte mare de probleme descoperite la versiunea anterioară, piaţa Android fiind în continuă expansiune de atunci. În prezent, Android a ajuns la versiunea 4.0. – cea mai modernă versiune, apărută pe piaţă o dată cu telefoanele Galaxy Nexus. Ultima versiune de Android aduce cea mai mare varietate de schimbări în cadrul platformei. Printre schimbările majore se numără: butoanele virtuale, suport pentru widget-uri îmbunătățit, un font nou pentru reprezentarea caracterelor, şi nu în ultimul rând recunoașterea facială.[6]

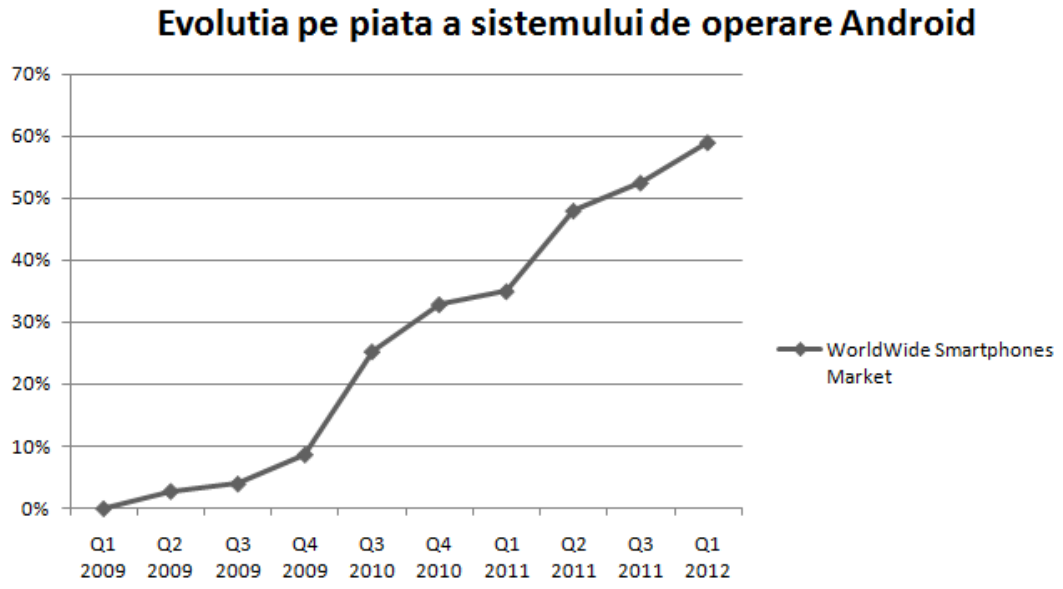

Figura 1: Evoluția pe piaţă a sistemului de operare Android (grafic realizat folosind date de la [6])

Linux oferă abstractizare a hardware-ului pentru Android, permițând să fie portat pe o mare varietate de platforme. Pe plan intern, Android folosește Linux pentru gestionarea memoriei, managementul proceselor, servicii de rețea şi alte servicii ale sistemului de operare.

Următorul nivel din arhitectura Android, conţine bibliotecile native. Aceste biblioteci partajate sunt scrise în C şi C++, compilate pentru arhitectura hardware folosită de telefoane şi preinstalate pe telefon. Câteva dintre cele mai importante biblioteci Android sunt: Surface Manager, Grafica 2D şi 3D, baza de date SQL, Browser-ul.

Diagrama din Figura 2 prezintă componentele majore ale arhitecturii sistemului de operare Android:

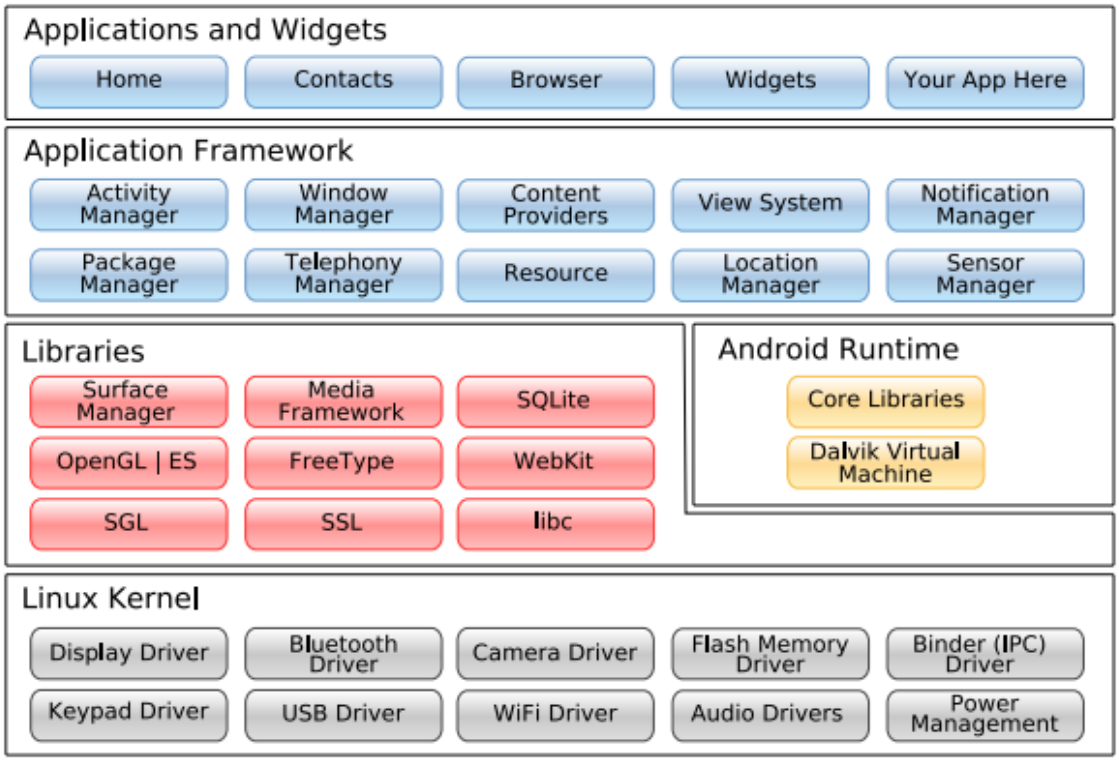

Figura 2: Arhitectura sistemului de operare Android [7]

O parte semnificativă a kernel-ului o reprezintă Android Runtime, care include maşina virtuală Dalvik şi bibliotecile de bază Java. Android Runtime trebuie să aibă următoarele facilități : viteza limitată a procesorului, memorie RAM limitată, baterie puternică, să nu aibă spațiu de swap şi să deţină un set divers de dispozitive. Având în vedere toate aceste cerințe, o mașină virtuală a părut a fi o alegere potrivită.

Dalvik este mașina virtuală (VM) din sistemul de operare Android. Este software-ul care rulează aplicațiile pe dispozitive Android. Dalvik este parte integrantă a Android, care este de obicei folosit pe dispozitive mobile cum ar fi telefoanele mobile şi tabletele. Programele sunt de obicei scrise în Java şi compilate într-o mașină de instrucțiuni independente numite bytecodes. Acestea sunt apoi convertite din fișiere `.class` sau `.jar` compatibile Java Virtual Machine în fișiere `.dex` (Dalvik Executable) compatibile Dalvik , înainte de a fi instalate pe dispozitive. Fișierele `.class` sunt citite de JVM la runtime. Fișierele `.dex` sunt mai compacte şi mai eficiente decât fișierele `.class`, un aspect important pentru limitarea de memorie şi baterie a dispozitivelor mobile. Un fișier `.class` conține o singură clasă, pe când un fișier `.dex` poate conține mai multe clase. Formatul Dalvik Executabil este special conceput pentru sisteme care sunt limitate în termeni de memorie şi viteză a procesorului. [8]

Diagrama din Figura 3 prezintă asemănările şi deosebirile între fișierele `.class` şi fișierele `.dex` care sunt executate de mașina virtuală Dalvik.

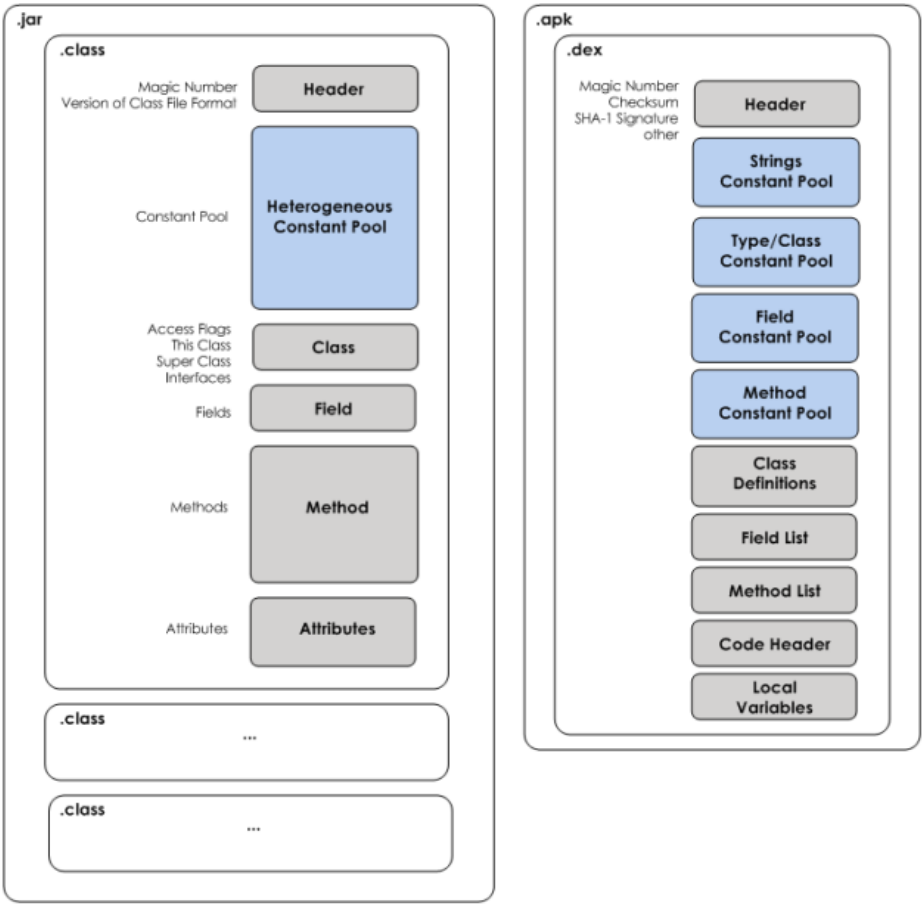

Figura 3: Comparaţie intre fişierele `.class` şi fişierele `.dex` [8]

O altă diferență dintre Dalvik şi mașinile tradiționale de Java sunt bibliotecile de bază Java care vin cu Android. Acestea sunt diferite de bibliotecile Java Standard Edition şi Java Mobile Edition.

Dalvik este un software open-source. Este în esență o mașina virtuală Java optimizată pentru consum redus de memorie. Permite rularea mai multor instanțe ale aceleiași mașini la un moment dat şi are avantajul de securitate al sistemului de operare Linux. Spre deosebire de mașinile virtuale Java care sunt mașini stivă, Dalvik este o arhitectură bazată pe regiştri. Pentru a converti anumite clase Java în format `.dex` este folosit un instrument numit DX. Java bytecode este de asemenea convertit în seturi alternative de instrucțiuni utilizate de Dalvik VM.

Un fișier necomprimat `.dex` este de obicei mai mic decât un fișier comprimat `.jar` (Java Archive) derivate din aceeași clasă. Fișierele executabile Dalvik pot fi modificate după ce au fost instalate pe dispozitivele mobile.

Standardul Java bytecode execută instrucţiuni de stivă pe 8 biți. Variabilele locale trebuie să fie copiate în sau din stivă de instrucţiuni separate. Dalvik foloseşte în schimb propriul set de instrucţiuni pe 16-biţi, care acționează în mod direct asupra variabilelor locale. Acest lucru reduce numărul de instrucțiuni şi măreşte viteza.

Fiecare aplicaţie Android rulează propriul proces cu propria instanța a unei maşini virtuale Dalvik. Dalvik a fost scris, astfel că un dispozitiv poate rula mai multe instanțe ale unei mașini virtuale eficient. VM Dalvik se bazează pe kernel-ul Linux pentru funcţionalitatea de bază, cum ar fi gestionarea memoriei la nivel scăzut.

Android permite programatorilor să dezvolte aplicații scrise în limbajul de programare Java, folosind bibliotecile puse la dispoziție de SDK (Software Development Kit). **SDK**-ul Android cuprinde un set de instrumente de dezvoltare printre care şi un emulator de dispozitiv, biblioteci, documentație, program de depanare, etc.

Mediul de dezvoltare (IDE) suportat oficial este Eclipse, utilizând plug-in-ul Android Development Tools (ADT). Plug-in-ul ADT este proiectat pentru a oferi dezvoltatorilor Android un mediu puternic, integrat în care să creeze aplicații mai ușor. ADT extinde facilitățile Eclipse şi permite configurarea rapidă a proiectelor noi în Android, crearea mai ușoară a aplicațiilor UI, editarea facilă a fișierelor XML, depanarea aplicațiilor într-un mediu grafic.

O aplicaţie Android poate avea în componenta să oricâte surse `.java` şi fişiere `.xml`. Câteva dintre ele, însă, sunt predefinite. Orice aplicaţie Android are în componenţa sa fișierul Android Manifest. Acesta este un fișier de tip `.xml`, care conține informații despre proiect: iconița proiectului, numele autorului; tot aici se pot specifica versiunea şi numele pachetului. Simbolul @ din faţa anumitor atribute arătă că șirul care urmează este o referință la o resursă. Resursele se află în folderul `res` al proiectului. Folderul `values` (subfolder al folderului `res`) poate stoca diferite

tipuri de resurse printre care numere, culori, string-uri. Fişierul `strings.xml` este creat automat în acest folder şi conţine mesajul şi numele aplicaţiei care apar atunci când este rulat proiectul.

Programarea pe Android forţează dezvoltatorii să separe funcţionalităţile unui proiect de partea de interfaţă, pentru ca operaţiile să nu blocheze interfaţa. De aceea operaţiile de background trebuie să ruleze în thread-uri separate. Cea mai bună metodă de a face această separare în cadrul unui proiect Android este pattern-ul Model View Controller.

Model – View – Controller este o metodă de proiectare pentru software care separă reprezentarea informațiilor de interfața cu utilizatorul. Modelul cuprinde datele (informațiile) şi regulile de afaceri, controller-ul asigură comunicația între model şi view. View-ul conține elemente din interfața cu utilizatorul (texte, input-uri ale formularelor) sau date de ieșire (diagrame). MVC definește foarte bine interacțiunea dintre cele trei componente ale sale astfel: [9]

- **Controller**-ul primește datele introduse de utilizator şi inițiază un răspuns prin efectuarea apelurilor pe diferite obiecte model. Controller-ul poate trimite comenzi view-ului care îi este asociat pentru a schimba modul în care este reprezentat modelul de către view) şi comenzi către model pentru a-şi actualiza starea.

- **Modelul** anunță view-urile (perspectivele) şi controller-ele care îi sunt asociate în momentul în care apar schimbări în starea sa pentru ca acestea să poată reacționa. Modelul este cel care gestionează comportamentul şi datele aplicației, răspunde la cereri de informații despre starea acesteia şi răspunde la instrucțiuni pentru a schimba starea. Modelul trebuie implementat astfel încât să poată fi folosit fără a fi modificat pentru diferite interfețe.

- **View**-ul folosește informațiile primite de la model pentru a genera o reprezentare a ieșirii, de obicei o interfață pentru utilizator. Pot exista mai multe view-uri şi controllere pentru un singur model, având scopuri diferite.
  View-ul şi modelul interacționează numai cu controller-ul, niciodată nu interacționează între ele.

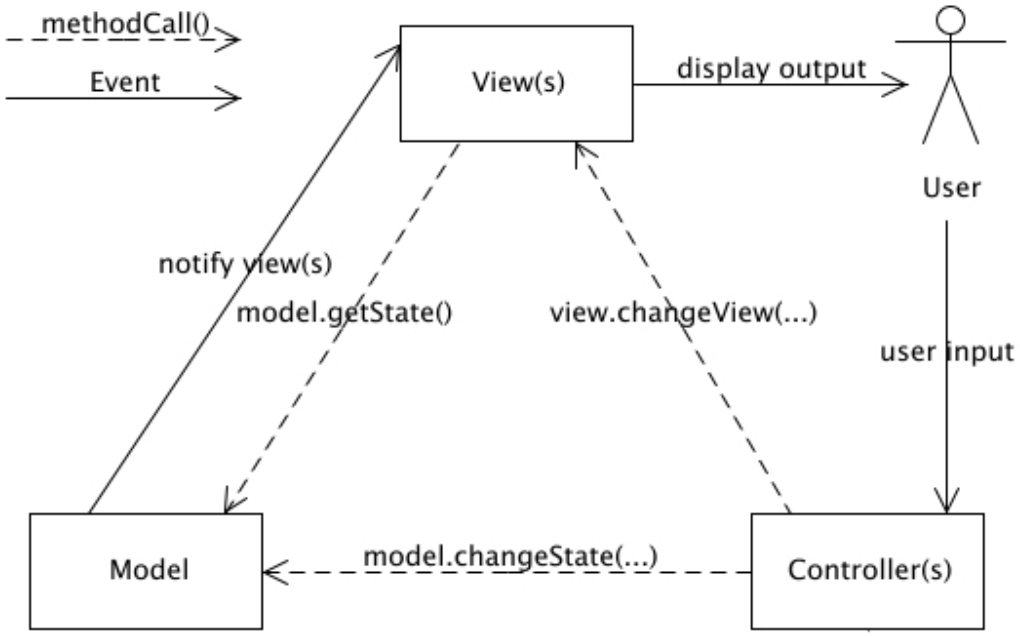

Figura 4: Model – View – Controller (inspirată de la [10])

În arhitectura Android, pattern-ul MVC este reprezentat astfel:

- Modelele identificate prin Content Providers.
  Content Providers este o clasă publică, abstractă din Android care gestionează accesul la un anumit set de date şi oferă mecanisme pentru securitatea datelor. Este interfaţa standard care conectează datele dintr-un proces care rulează cu datele din cadrul altui proces. Content Providers sunt un fel "manageri de date" şi este recomandat să fie folosiţi în schimbul de date între aplicaţii. Sunt folosiţi decât dacă este nevoie de schimb de date între mai multe aplicaţii.

- View-urile sunt Activităţile (Activity)
  Activity este principala componentă a interfeței cu utilizatorul, este derivată din clasa publică `Java Activity` (`android.app.Activity`) şi este container pentru `View` (`android.view.View`). Activity interacționează cu utilizatorul şi creează o fereastră în care se poate plasa UI (User Interface) folosind metoda `setContentView(View)`. În timp ce activităţile de multe ori sunt prezentate utilizatorului ca ferestre full-screen, ele pot fi folosite şi în alte moduri: ca ferestre plutitoare (într-o temă cu un `windowIsFloating` setat) sau încorporate în interiorul unei alte activităţi (folosind ActivityGroup).

- Controler-ele sunt reprezentate de Servicii (Services)
  Serviciile sunt componente de bază care se comportă că daemonii UNIX şi serviciile Windows. Ele rulează în fundal şi efectuează prelucrarea pe parcursul aplicației**.**
  Un serviciu este o componentă care reprezintă o cerere a aplicației de a efectua o operaţiune de mai lungă durată în timp ce nu interacționează cu utilizatorul sau de a furniza funcționalități pentru alte aplicații. Fiecare clasă de servicii trebuie să aibă o declaraţie corespunzătoare <service> în fișierul `AndroidManifest.xml` din pachetul său. Serviciile pot fi pornite cu `Context.startService()` şi `Context.bindService()`. [9]

3.2 Tehnologiile folosite pentru comunicaţia client-server. Alte tehnologii

Pentru serializarea/deserializarea diferitelor tipuri de mesaje de pe server (cel mai simplu tip de mesaj cuprinde o întrebare sub forma unui `string` şi o selecţie, tot sub forma `string)` este folosită în cadrul proiectului tehnologia Google Protocol Buffers. Un alt motiv pentru care am ales această tehnologie este consumul foarte mic de bandă.

Google Protocol Buffers este un Framework independent de platformă, un limbaj neutru, pentru serializarea/deserializarea datelor binare pentru a fi utilizate în protocoale de comunicaţie, de stocare a datelor, etc.

Protocol Buffers sunt flexibile, eficiente, mecanisme automate pentru serializarea datelor structurate, ca un XML, dar mai rapide şi mai simple. Cu ajutorul lor, se poate defini modul în care vor fi structurate datele o singură dată, apoi codul sursă generat special poate fi folosit pentru a scrie şi citi datele de la o varietate de fluxuri şi folosind o varietate de limbaje. [11]

Structurile de date (numite mesaje) şi serviciile sunt definite în fișierele Proto Definition (`.proto`) care sunt compilate cu `protoc`. Fiecare mesaj este o înregistrare logică de informații, conținând o serie de perechi nume-valoare. [11]
În secvenţa de cod 1 este un exemplu de astfel de mesaj (fişier .proto) :

```
message Persoana {
      required string nume = 1;
      required int32 id = 2;
      optional string email = 3;
      enum Telefon {
            MOBILE = 0;
            HOME = 1;
            WORK = 2;
      }
      message NumarTelefon {
```

```
            required string numar = 1;
            optional PhoneType tip = 2 [default = HOME] ;
      }
      Repeated NumarTelefon telefon = 4;
}
```

Secvenţa de cod 1: Exemplu de mesaj Protocol Buffers

Compilatorul Protocol Buffers creează o clasă care implementează codificarea automată şi parsarea datelor într-un format binar. Clasa generată oferă metode Getter şi Setter pentru câmpurile din interiorul unui Protocol buffer. Formatul Protocol Buffers suportă extinderea în timp, dar codul va fi compatibil şi cu formatele anterioare.

Pentru a folosi Protocol Buffers în primul rând trebuie dezvoltat un serviciu Web care convertește datele CSV în format Protocol Buffers, apoi trebuie construită o aplicație Android care ia date de la serviciul Web în acest format (Protocol Buffers) şi le analizează pentru a fi afișate utilizatorului.

Serverul folosit şi implementat în acest proiect este un server Java care foloseşte un serviciu web RESTful. Am ales tehnologia REST pentru că este standardizată (operaţiile HTTP sunt bine înţelese şi funcţionează în mod consecvent), este uşor de testat, este lizibilă, nu este nevoie de fişiere .xml

Representational state transfer (REST) este o arhitectură software pentru sisteme distribuite, un exemplu de proporții fiind World Wide Web. A fost dezvoltat în paralel cu HTTP/ 1.1, fiind bazat pe design-ul deja existent al HTTP/1.0 şi exemplifică modul în care interacționează cele patru componente ale Web-ului : Proxy-uri, servere, gateway şi clienți fără a impune limitări asupra participanților individuali. În ultimii ani, REST a devenit un model de design/proiectare predominant în lumea serviciilor Web, înlocuind modele ca SOAP şi WSDL, datorită stilului său simplu. [12]

REST este o abordare pentru a obține informații de pe un site web, folosind fișiere XML pentru a include conținutul dorit. Datorită simplității sale, REST a devenit o modalitate populară printre dezvoltatori de a accesa serviciile.

În arhitectura REST, informația de pe server este considerată o resursă pe care dezvoltatorii o pot accesa folosind HTTP (Hypertext Transfer Protocol) şi URI (Uniform Resource Identifiers). Deoarece REST utilizează HTTP ca protocol de comunicație este constrâns să fie o arhitectură de tip client-server. Clienții inițiază cereri către servere, serverele procesează cererile şi dau răspunsuri adecvate. Cererile şi răspunsurile sunt construite pe baza reprezentărilor resurselor. O resursă poate fi orice concept care poate fi abordat. Reprezentarea unei resurse este de obicei un document care surprinde starea actuală sau dorită a unei resurse. Clientul începe trimiterea de cereri când este pregătit pentru a face tranziția la o nouă stare. REST facilitează tranzacţia dintre servere de web,

permițând cuplarea între diferite servicii. Limbajul REST se bazează pe utilizarea de substantive şi verbe şi pune accent pe lizibilitate. [12]

Serviciile web RESTful (serviciile web care sunt create şi accesate prin utilizarea principiilor REST) folosesc protocolul HTTP pentru operațiile pe care le efectuează.

JAX-RS oferă un API pentru crearea serviciilor web RESTful în Java. API-ul JAX-RS folosește adnotări pentru a simplifica dezvoltarea serviciilor web RESTful. Adnotările, împreună cu clasele şi interfețele oferite de API-ul JAX-RS permit reprezentarea resurselor web ca POJO (Plain Old Java Object). [13]

Ca în orice altă aplicație web Java, aplicațiile JAX-RS sunt grupate ca un fișier `WAR` şi implementate apoi pe cu un container care suportă Servlets. (exemplu server Tomcat, server Glassfish). Servlets-urile oferite de container pot fi utilizate pentru a ruta cererile către cea mai apropiată resursă web. Un obiectiv al JAX-RS este acela de a permite portabilitatea pentru a implementa resurse web în diferite tipuri de containere.
Sun oferă o implementare de referinţă pentru JAX-RS numită **Jersey**.

Am ales să folosim o implementare a JAX-RS (Jersey) pentru că este extensibil, permite adăugarea de noi funcţionalităţi uşor şi foloseşte protocolul Google Procol Buffers. Ambele sunt proiectate pentru sisteme de înaltă performanţă şi pentru a reduce CPU utilizat în timpul serializării şi deserializării datelor.

Jersey foloseşte un server de web HTTP numit **Grizzly** şi servlet-ul Grizzly. Servlet-ul gestionează cererile către Grizzly. Pot fi dezvoltate aplicații JAX-RS folosind Jersey care implementează toate API-urile şi oferă adnotările necesare pentru crearea serviciilor web RESTful. Prin intermediul API-urilor proprii, Jersey oferă funcții suplimentare, cum ar fi AP-ul pentru clientul Jersey. [14]

Clasele şi interfețele care pot fi folosite pentru a crea servicii web RESTful cu JAX-RS se găsesc în următoarele pachete: javax.ws.rs, javax.ws.core, javax.ws.rs.ext. JAX-RS definește o resursă ca pe orice clasa Java (POJO) care folosește adnotări JAX-RS pentru a implementa resursa web. Notația @Path identifică o clasă Java ca pe o clasă de resurse. Exemplul din Secvenţa de cod 2 este o clasă Java:

```
import  javax.ws.rs.Path;
@Path("/stockquote")
public  class StockResource {
…
      public String getStockInfo() {
            return "This îs Stock Information";
      }
}
```

Secvenţa de cod 2: Clasa Java care foloseşte Jersey

Pentru fiecare cerere făcută la o resursă va fi creată o nouă instanță a clasei de resurse. După crearea obiectului, este invocat constructorul, iar când este primit răspunsul obiectul devine disponibil pentru garbage collection. Metodele resursei sunt metode publice ale clasei de resurse care pot fi recunoscute printr-un identificator al metodei. Identificatorii metodelor sunt adnotări folosite pentru a recunoaște metodele HTTP şi JAX-RS. JAX-RS definește adnotări pentru metode HTTP ca GET, PUT, POST, DELETE şi HEAD şi permite definirea de identificatori personalizați în funcție de metoda.

JAX-RS oferă o mapare clară între protocolul HTTP şi URI, având interfețe şi clase bine definite.

În serviciile web RESTful, metodele HTTP sunt mapate cu operațiile CRUD pe care le efectuează. În funcție de tipul cererii HTTP efectuate sunt apelați identificatorii metodei. Ex: pentru e cerere HTTP GET, serviciul va apela o metodă @GET şi va aștepta răspuns. O metodă a resursei poate întoarce o valoare void, un obiect sau un alt tip Java.

Pentru a construi URL-ul la care se găsește un PDF pe server şi pentru a face cache la întrebările de pe server folosim **SQLite**.

SQLite este un sistem open-source de gestiune de baze de date relaționale, conținut într-o bibliotecă de C. Este portabil, ușor de utilizat, compact şi eficient. Este o bază de date incorporata în aplicația pe care o servește, se comportă ca o parte a programului pe care îl găzduiește. Avantajul de a avea un server de baze de date în interiorul programului este acela că nu mai este necesară nicio configurare de rețea. Clientul şi serverul rulează același proces. [15]

SQLite are o arhitectură modularizata, modulele de bază fiind interfața, compilatorul şi mașina virtuală. Interfața este API-ul C SQLite, este modalitatea prin care programele şi limbajele de scripting interacționează cu SQLLite.

Procesul de compilare începe cu Tokenizerul şi parserul care lucrează împreună pentru a valida sintaxa şi pentru a o converti într-o structură cu care se poate lucra mai ușor. Parserul este generat de generatorul de parsere al SQLite numit Lemon.

Mașina virtuală, numită şi motorul de bază de date virtual (VDBE) funcționează pe cod octet , ca o mașina virtuală Java. Este concepută pentru prelucrarea datelor, fiecare instrucțiune realizând o operație specifică. Orice declarație scrisă în SQLite este mai întâi compilată în această mașina virtuală.

SQLite este disponibil pe orice dispozitiv Android, fără să aibă nevoie de configurare sau administrare. Este gestionat automat de platforma Android. Pentru accesul la baza de date SQLite se accesează sistemul de fișiere. Este recomandat ca operațiile pe bază de date să se efectueze asincron, prin intermediul unei clase **AsyncTask.**

Asynctask este o clasă abstractă Android care permite folosirea corectă şi uşoară a thread-ului UI, rularea operaţiilor în fundal şi afişarea lor în thread-ul UI.

O sarcină asincronă este definită printr-un calcul care rulează pe un thread din fundal, ale cărui rezultate sunt afişate pe thread-ul UI. Are trei tipuri generice: Params, Progress şi Result.

O bază de date SQLite este privată pentru aplicaţiile pe care le creează. Pentru a partaja date cu alte aplicații se poate folosi un **ContentProvider** .
ContentProvider poate fi folosit intern pentru a accesa date. Accesul la ContentProvider se face printr-un URI. URI este definit la declarația ContentProvider, în fișierul AndroidManifest.xml folosind atributul android: authorities. De obicei, clasele care implementează ContentProviders oferă constante publice pentru URI. ContentProvider poate fi definit în AndroidManifest.xml sau folosind o clasă care extinde `android.content.ContentProvider`.

Pentru vizualizarea fişierelor PDF pe dispozitivele mobile am folosit software-ul open-source MuPDF. Acestui proiect i-am adăugat noi funcţionalităţi pentru a îndeplini cerinţele aplicaţiei noastre.

MuPDF este un software gratuit, scris în C, care implementează parsarea şi randarea fișierelor PDF şi XPS. Randarea paginilor se face cu o precizie de fracțiuni de pixel pentru a reproduce perfect aspectul unei pagini pe ecran. [16]

Este folosit în primul rând pentru a converti paginile unui document în bitmap-uri, dar oferă suport şi pentru alte operații cum ar fi căutarea în document şi listarea conținutului, criptarea, transparenţa, adnotări şi hyperlink-uri.

Este o aplicație care permite vizualizarea fișierelor de tip PDF, XPS sau OpenXPS. Dezvoltatorii au pus accentul, în dezvoltarea aplicației, pe viteza, cod de dimensiuni reduse şi randare de înaltă calitate. Nu suportă funcții interactive, JavaScript şi tranziții. Este scris modular, astfel încât programatorii care îl folosesc să poată adăuga noi funcționalități.

Librăria vine cu un set de instrumente în linia de comandă: pentru randarea unui pdf (pdfdraw), pentru examinarea structurii fișierului (pdfshow) şi pentru rescrierea fișierelor (pdfclean).

MuPDF este disponibil ca pachet gratuit şi pentru Debian, Fedora, porturi FreeBSD. A fost portat pe multe platforme printre care Android, Amazon Kindle.

Proiectul pentru portarea MuPDF în sistemul de operare Android include librăria mudpdf, o interfața mupdf jni pentru Android şi aplicația Android pdfViewer (.apk).

4. Arhitectura sistemului

4.1 Descrierea arhitecturii şi a funcţionalităţii proiectului

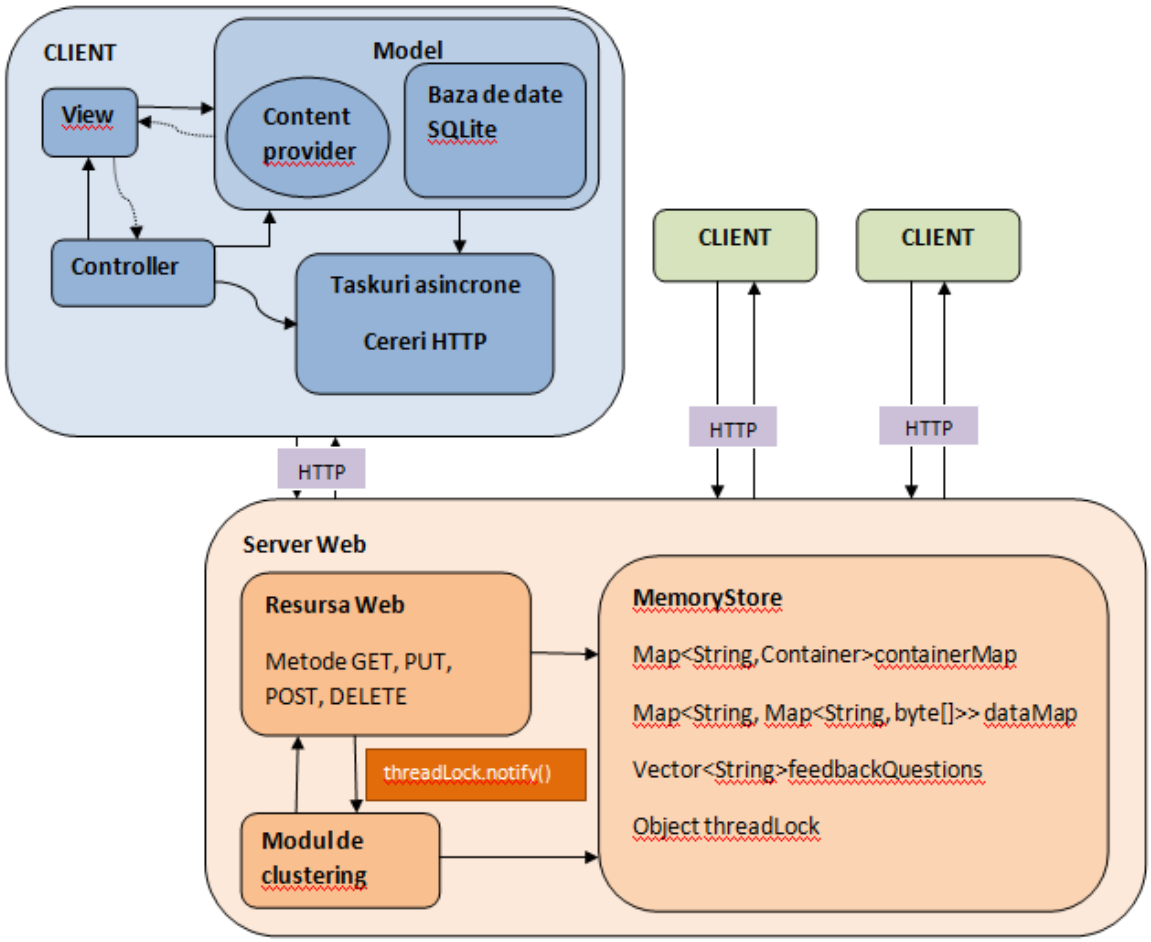

Figura 5 : Arhitectura proiectului

Pentru a implementa aplicaţia prezentată conceptual este necesară implementarea a două componente majore: modulul Client şi modulul Server.

Modulul Client se ocupă atat de partea de interacţiune cu utilizatorul: interfaţa prezentatorului şi interfaţa audienţei, cât şi de partea de comunicare cu serverul: trimite diverse tipuri de cereri HTTP asincrone la server (pentru a primi date legate de utilizatorii din baza de date –

SQLite, pentru a primi lista întrebărilor de pe un anumit slide/din toată prezentarea, pentru a primi numărul de +1, ambiguous sau citation or proof needed de pe un anumit slide).

Modulul Server se ocupă de partea de clusterizare a întrebărilor primite de la auditoriu, gruparea lor în funcţie de asemănările semantice, trimiterea întrebărilor către client, contorizarea feedback-ul de fiecare tip, primirea cererilor de la client.

Fiecare din aceste două module este împărţit la rândul sau în module mai mici, pe care le voi descrie în continuare.

4.2. Modulele funcţionale ale sistemului

Modulul Client este format din:

- Modulul View este reprezentat în aplicaţie de activităţile implementate (Activity). Acest modul este principala componentă a interfeţei cu utilizatorul. În cadrul lui sunt implementate funcţionalităţi ca: selecţia din PDF, funcţionalităţile butoanelor interfeţelor, sliding-ul paginilor.
- Modulul Controller comunică cu modulele View, Model şi cu modulul de taskuri asincrone şi cereri HTTP. Acest modul primeşte cererile de la utilizator şi le trimite modulului de cereri HTTP pentru a fi procesate şi trimise la server. Pe de altă parte, trimite cereri la ContentProvider, care la rândul lui trimite cererea la server. Acest modul cuprinde toate acţiunile care au loc la activarea interfeţei, este un fel de “handler de evenimente".
- Modulul Model este reprezentat de Content Provider şi de baza de date SQLite. Se ocupă cu stocarea şi securitatea datelor, conţine reprezentările datelor de intrare, ale datelor intermediare şi ale datelor de ieşire. Poate primi cereri de la controller pentru a interoga baza de date, returnează la cerere date din baza de date. Dacă este nevoie, actualizează baza de date cu cereri de la serverul web. Acest modul este independent de interfaţa cu utilizatorul, lucru deosebit de important deoarece dacă se doreşte schimbarea implementării prin adăugarea de noi funcţionalităţi, nu este necesară efectuarea de schimbări în modulul view.
- Modulul de taskuri asincrone şi cereri HTTP primeşte cereri asincrone de la Controller şi Model pe care le trimite prin HTTP serverului Web. Acest modul este format din clase de tip AsyncTask care rulează pe threaduri separate şi sunt instanţiate in Controller atunci când sunt apăsate butoanele. După ce se execută cererile HTTP, clasele modului actualizează interfaţa.

Modulul Server este format din:

- Modulul Resursa Web comunică cu modulul de clustering şi cu modulul Memory Store. În acest modul sunt clase care definesc metode HTTP (GET, POST, PUT, DELETE) care sunt folosite in cererile HTTP.
- Modulul de clustering comunică cu modulul de resurse web şi cu cel de memorie. În acest modul are loc clusterizarea întrebărilor în funcţie de asemănările semantice dintre acestea şi realizarea unui arbore cu acestea. Acest arbore este trimis apoi modulului Resursa Web prin metodele PUT, POST.
- Modulul Memory Store este o clasă cu membri şi metode statice unde scriu/citesc date toate modulele serverului. Primeşte date de la modulul Resursa Web şi de la modulul de clustering şi cuprinde toate structurile de date.

4.3. Arhitectura clientului Android

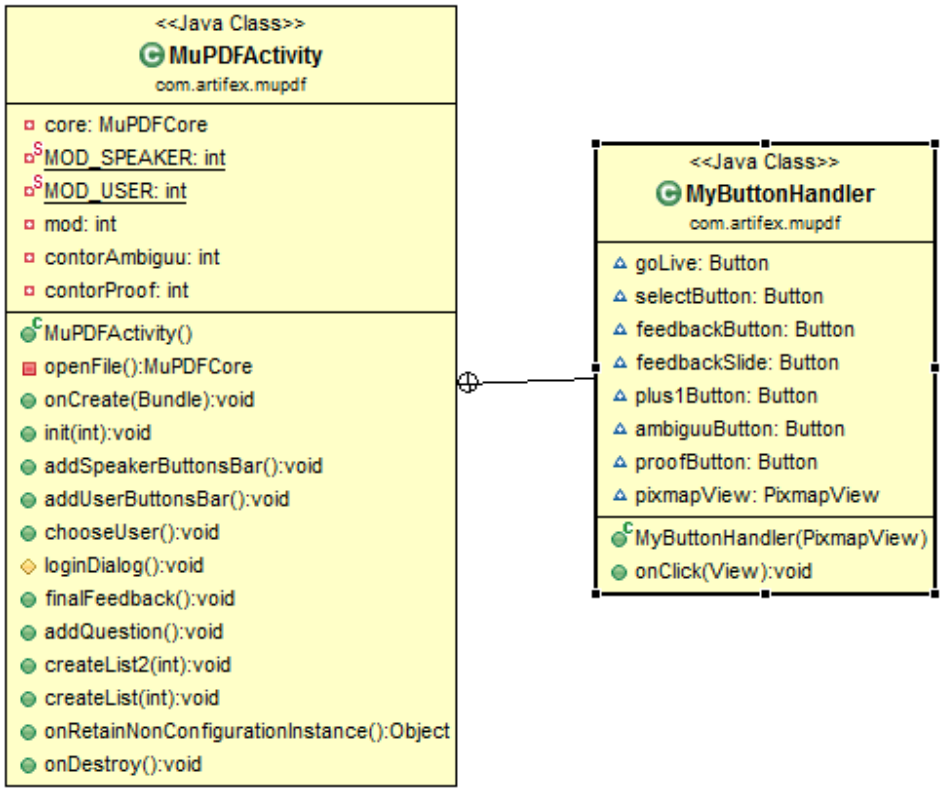

Figura 6: Structura clasei MuPDFActivity

Clientul Android are o clasă – MuPDFActivity care este strâns legată de implementarea interfeţei. Această clasă este împărţită în două module care comunică între ele.
Modulul principal - MuPDFActivity cuprinde numele şi valorile variabilelor, implementările metodelor pentru vizualizarea butoanelor şi a listelor agregate cu întrebări.
În modulul MyButtonHandler sunt declarate toate butoanele celor două interfeţe şi implementate metodele pentru click pe buton. Metodele acestui modul apelează metode din modulul MuPDFActivity.

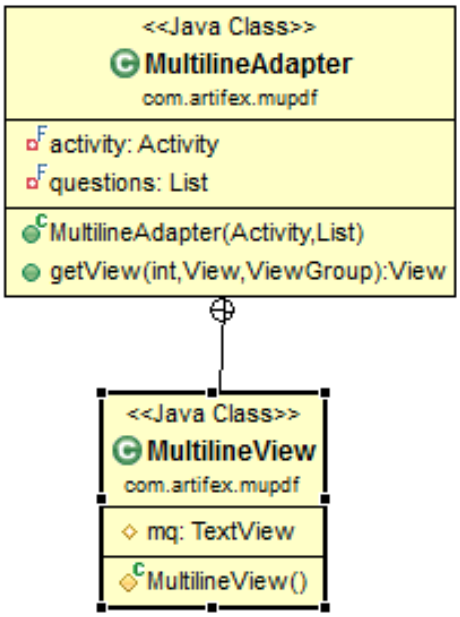

Figura 7 : Structura clasei MultilineAdapter

Pentru a implementa facilitatea de a vizualiza întrebările de dimensiuni mari pe mai multe rânduri am creat o nouă clasă MultilineAdapter care extinde clasa ArrayAdapter. Schema acestei clase este descrisă în Figura 7. Clasa Multiline Adapter are două module care comunică între ele. In modulul principal al acestei clase este suprascrisă metoda getView() a clasei ArrayAdapter . Această metodă întoarce un obiect de tip View care este trimis modului MultilineView.

5. Detalii de implementare

5.1 Implementarea interfeţelor de utilizare ale clientului Android

5.1.1 Interfaţa clientului participant la prezentare

La deschiderea aplicației, clientul participant la prezentare trebuie să se autentifice folosind un nume. După autentificare, slide-urile se vor descărca automat de pe server pe telefonul său şi se vor sincroniza cu prezentatorul. Clientul intră astfel în view-ul principal al aplicaţiei pe care îl descriu în continuare. Acest view are o suprafaţă mare dedicată prezentării, dar şi o zonă dedicată butoanelor cu ajutorul cărora participantul va putea face adnotări pe parcurs.

Butonul `Go live` sincronizează prezentarea de pe telefonul participantului cu prezentarea de pe telefonul speaker-ului atunci când participantul apăsă pe el. În spatele acestei acțiuni, se trimite o cerere la server web prin metodele GET sau POST.

Butonul de selecție activează sau dezactivează posibilitatea utilizatorului de a selecta text din PDF. După ce a făcut o selecție, utilizatorul poate da feedback pe acea selecție. Ca modalități prin care se poate exprima feedback-ul sunt:

- Butonul `+1` folosit pentru feedback pozitiv sau pentru a susține o întrebare pe care a adresat-o un coleg mai devreme
- `Butonul ambiguous/unclear` folosit atunci când utilizatorul nu înțelege o anumită formulare şi ar vrea să primească o explicaţie mai concisă
- Butonul `citation or proof needed` folosit atunci când utilizatorul are nevoie de o demonstrație concretă a celor scrise pe slide

Butonul `Feedback` este o listă agregată cu întrebările care au fost puse pe fiecare slide (lista de tip `Alert dialog` din Android) pe care utilizatorul o poate vedea. În cazul în care 2 utilizatori vor să pună aceeași întrebare, pot folosi butonul +1 pentru a nu scrie de două ori textul întrebării. Tot în lista de feedback este şi butonul prin care utilizatorul poate adăuga o întrebare nouă – `Add new question.`

Întrebările sunt grupate după similaritate semantică, întrebările de același tip fiind într-o altă listă. Între lista principală de feedback şi listele cu întrebări similare se poate naviga.Toate butoanele din interfaţă au asociate imagini sugestive.

În aplicațiile Android, interfața este construită folosind obiecte de tip **`View`** şi **`ViewGroup`**. Există mai multe tipuri de `View` şi `ViewGroup`, fiecare dintre ele fiind un descendent al clasei `View`.

Obiectele de tip `View` sunt unitățile de bază ale interfeței cu utilizatorul pe platforma Android. Clasa `View` este considerată baza pentru subclase care oferă obiecte UI implementate, cum

ar fi butoane şi câmpuri de text. Clasa `ViewGroup` este clasa de bază pentru subclasele `layouts` care oferă diferite tipuri de structură cum ar fi liniară, tabel, relativă.

Un obiect de tip `View` este o structură de date ale cărui proprietăți stochează parametrii `layout`-ului şi conținutul pentru o anumită zonă, în general dreptunghiulară, a ecranului. Obiectul `View` este de asemenea un punct de interacțiune între utilizator şi receptor.

Pe platforma Android, pentru a defini interfața pentru o activitate (Activity) se folosește o ierarhie de View-uri şi ViewGroup, de forma celei din Figura 8:

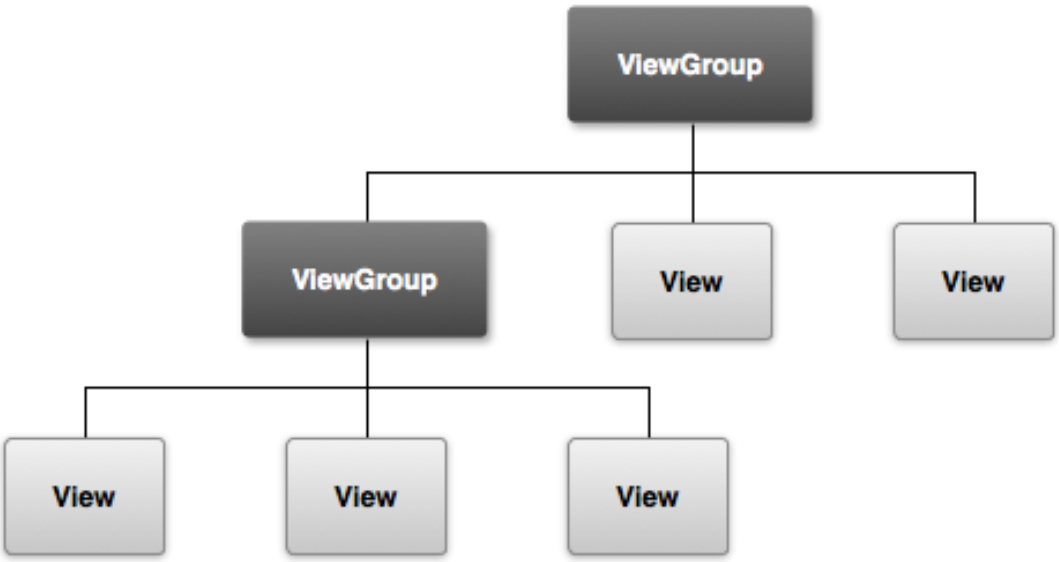

Figura 8 : Ierarhie View [17]

`Layout`: cel mai întâlnit mod de a defini un layout şi de a exprima ierarhia de View-uri este folosind un fișier `.XML`. Fiecare element din `.XML` este un View sau un obiect ViewGroup. Layout-ul este arhitectura pentru interfața cu utilizatorul dintr-o activitate (Activity).

Un layout poate fi declarat în 2 moduri:

- Declarând elemente UI într-un fișier `.XML`. Avantajul folosirii fișierelor `XML` este că ajută la o mai bună separare a parții de prezentare a aplicației, de partea de cod. Fiecare fișier `XML` trebuie să aibă un singur element rădăcină, care trebuie să fie un obiect de tip `View` sau `View Group`. După ce se definește elementul rădăcină se pot adăuga elementele `copil` în ierarhie pentru a defini layout-ul.
- Instanțiind elementele layout-ului la runtime. Aplicația poate crea obiecte de tip View sau ViewObject programatic.

Programatorul poate fi folosi oricare din aceste două metode sau pe amândouă în același timp pentru a construi interfața aplicației sale.

Layout-urile sunt de 4 tipuri: [18]

- **FrameLayout** este cel mai simplu tip de obiect layout. Este practic un spațiu gol pe ecran pe care îl poți umple mai târziu cu un singur obiect – de exemplu, o poză. Toate elementele `copil` ale FrameLayout sunt fixate în colţul stânga sus al ecranului şi nu pot fi mutate.
  Poate fi folosit pentru a afișa mai multe butoane/controale în același ecran.

- **LinearLayout** are toate elementele `copil` aliniate într-o singură direcție – vertical sau orizontal, în funcție de cum a fost definit atributul orientare (proprietatea `orientation`). Toate elementele apar în listă unul după altul, astfel că o listă verticală va avea un singur element pe rând , indiferent de cât de mare este şi o listă orizontală va avea un rând. Se respectă marginile între elemente.
  Cele mai importante proprietăți ale layout-ului linear sunt: `orientation`, `fill model`, `gravity`, `padding`.
  `Orientation`: această proprietate stabileşte cum vor fi afişate elementele, fie orizontal, stil rând sau vertical, gen coloană.
  `Fill model` : widget-urile din interiorul unui LinearLayout au înălţime şi lăţime. Aceste proprietăţi pot avea trei valori:
  - o valoare numerică în pixeli, dpi sau inchi. Valoarea default pentru fiecare din ele este 0.
  - `wrap_content` - widget-ul ocupă un spaţiu egal cu aria sa.
  - **f**`ill_parent` – widget-ul ocupă tot spaţiul existent.

  `Gravity` : iniţial toate controalele sunt poziţionate în partea stângă-sus. Acest lucru însă se poate modifica utilizând proprietatea `layout_gravity`.
  Proprietatea `padding` determină spaţiul între widget-uri padding_left, padding_top, padding_right, padding_buttom, padding.

- **TableLayout:** elementele sunt așezate pe linii şi coloane. Nu se afișează linii de frontieră pentru rânduri, coloane sau celule. Un TableLayout este alcătuit din mai multe componente `TableRow`, iar la rândul său un `TableRow` poate conţine una sau mai multe celule. Fiecare celulă poate conţine doar un singur `View`. Celulele unui rând pot fi formate dintr-o varietate de obiecte `View`, cum ar fi `ImageView` sau `TextView`. Coloanele care alcătuiesc structura tabelului sunt recunoscute de Android
  Tabelul va avea la fel de multe coloane ca rândul cu cele mai multe celule. Un tabel poate avea celule goale, dar celulele nu se pot întinde pe coloane, cum se întâmplă în HTML.

- **RelativeLayout** : acest tip de organizare presupune poziţionarea elementelor grafice relativ la un alt control sau chiar relativ la `părinte` (ţinând cont de structura ierarhică a ecranului). Acest tip de aşezare implică faptul că `un copil` de tip control, ce poate fi reprezentat printr-un `Button`, `TextView`, `EditText` sau oricare alt tip de control, poate fi plasat relativ la o altă componentă a interfeţei grafice fie deasupra, dedesubt, în dreapta sau în partea stângă a acesteia.
  `Copilul` de tip control poate fi poziţionat şi faţă de container-ul `părinte`: sus, jos, la dreapta sau la stânga. Această poziţionare trebuie să respecte nişte reguli stricte.

RelativeLayout asigură o organizare mai eficientă a componentelor interfeţei grafice fără a fi necesară utilizarea dimensiunii screen-ului sau a elementelor existente.

Butoanele sunt componentele de bază ale interfeței utilizatorului, cu ajutorul lor acesta poate efectua acțiunile precizate mai sus.

În SDK-ul de Android sunt două clase care pot fi folosite pentru a controla butoanele: `Button(android.widget.Button)` şi `ImageButton(android.widget.ImageButton)`. Diferența dintre cele două clase este mai mult vizuală, butoanele din clasa `Button` sunt, de obicei, butoane cu text (ex: butonul Save), iar butoanele din clasa `ImageButton` sunt asociate cu imagini (ex: butoane muzicale – play, stop).

Un buton este format din text şi/sau imagine care comunică ce acțiune se va produce atunci când utilizatorul îl atinge. Android suportă două tipuri diferite de butoane: butoane de bază şi butoane fără margini. Ambele tipuri de butoane pot conține text şi/sau imagini.

`Butoanele de bază` sunt butoane cu margini şi fundal. Pot fi de două tipuri: mari şi mici. Butoanele implicite au dimensiunea fontului mai mare şi sunt optimizate pentru afișare în afara conținutului formularului. Butoanele mici sunt concepute pentru afișare alături de alte tipuri de conținut, au un font mai mic.

`Butoanele fără margini` seamănă cu butoanele de bază, cu excepția faptului că nu au margini sau fundal. Pot fi folosite cu imagini şi text, se integrează mai frumos cu alte tipuri de conținut.

În proiect am folosit butoanele de baza.

Butoanele pot fi create în fişiere `.XML`, în interiorul unui layout sau în clasa activity a proiectului (`activity.java`).

În funcție de tipul de buton pe care vreau să îl creez folosesc una din metodele:

a) În cazul în care creez butoanele în fișiere .xml:

- Pentru butoane cu text , folosesc clasa Button:

```
<Button
      android:id = " @+id/buton"
      android:layout_width = "wrap_content"
      android: layout_height ="wrap_content"
      android:text ="@string/text_buton"
.. />
```

- Pentru butoane cu imagine, folosesc clasa ImageButton:

```
<ImageButton
      android:id = " @+id/buton"
      android: layout_width = "wrap_content"
      android: layout_height = "wrap_content"
      android: src = "@drawable/buton_imagine"
.. />
```

- Pentru butoane cu text şi imagine, folosesc clasa Button cu atributul android:drawableLeft:

```
<Button
      android:id = " @+id/buton"
      android:layout_width = "wrap_content"
      android:layout_height ="wrap_content"
      android_text = "@string/text_buton"
      android:drawableLeft = "@drawable/buton_imagine"
../>
```

  În codul Java, referirea la un buton se face folosind `findViewById(R.id.id_buton)`.
  Ex: `myBtn = (Button) findViewById(R.id.buton);`
  Fișierul R se găsește în interiorul folderului gen şi nu poate fi modificat.

b) În cazul în care creez butoanele în fișierul Java:

```
Button butonSelectie = new Button(this);
butonSelectie.setText("Select);
```

- Pentru a adăuga imagine pe un buton folosesc metoda:
  `setCompoundDrawablesWithIntrinsicBounds(int,int,int,int);`
  Exemplu utilizare:

```
butonSelectie.setCompoundDrawablesWithIntrinsicBounds(R.drawabl
e.nume_imagine, 0, 0, 0);
```

Funcționalitățile butoanelor se realizează adăugând evenimente pentru fiecare din ele si se pot adăuga în fișierele .XML sau în codul Java.

1. Folosind fișierul .XML

- pentru a defini evenimentul click pe un buton, în fișierul .XML, se adăugă atributul `android:onClick` elementului de tip `<Button>`. Valoarea pentru acest atribut trebuie să fie numele unei metode apelate ca răspuns la evenimentul click. Metoda trebuie să fie implementată în activity-ul corespunzător butonului.

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/butonSelectie"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/butonSelectie"
    android:onClick="SelecteazaText" />
```

În Activity se implementează metoda SelecteazaText:

```
public void SelecteazaText(View view) {
    // răspunsul la click
      butonSelectie.setText("Acum poţi selecta text");
}
```

2. Folosind fișierul `activity Java` care este în folderul src.

```
butonSelectie.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      butonSelectie.setText("Acum poţi selectă text");     }
});
```

Butonul `Feedback` al interfeței, este diferit de celelalte, el conține o listă agregată a întrebărilor participanților la prezentare. Pentru a realiza acest buton folosesc o succesiune de liste de tip **`alert dialog`** şi **`array adapter.`**

`Un dialog` este o fereastră care apare peste activitatea curentă. Activitatea de bază își pierde proprietățile şi fereastra dialog acceptă toate interacțiunile cu utilizatorul. Dialogurile sunt utilizate, de obicei, pentru notificările care ar trebui să întrerupă utilizatorul pentru a efectua sarcini scurte care se refera direct la aplicația în curs (bara de progres , solicitare de conectare).

Clasa Dialog este clasa de bază pentru crearea de dialoguri. Un dialog nu poate fi instanţiat direct, trebuie folosită una din subclasele clasei Dialog: `AlertDialog, ProgressDialog, DatePickerDialog, TimePickerDialog.`

Un dialog este creat şi afișat ca parte a unei activităţi. Trebuie creat folosind metoda de callback `onCreateDialog(int)`.Când este folosită această metodă, sistemul Android gestionează starea fiecărui dialog şi îl instanţiază. Dialogul moștenește anumite proprietăţi de la activitate. Pentru a afișa un dialog, se apelează metoda `showDialog(int)`, primește ca parametru o valoare întreagă care identifică în mod unic dialogul pe care vreau să îl afișez. Pentru a închide un dialog se apelează funcția `dismiss`() pe obiectul de tip Dialog. Se poate apela şi metoda `dismissDialog(int)` din activitate, metoda care aplează la rândul ei metoda `dismiss`().

Alert dialog sau fereastra de notificare poate fi creată în cadrul oricărei activităţi (Activity) şi asigură o mai bună comunicare între uşer şi aplicație. Aceasta poate gestiona zero, unu, două sau mai multe butoane şi/sau o listă de elemente selectabile care pot include casete cu text şi radio buttons, mesaje text, titluri. Este o extensie a clasei Dialog.

Pentru a crea un alert dialog se folosește subclasa `AlertDialog.Builder`. Se creează un constructor folosind `AlertDialog.Builder(Context)` şi apoi se utilizează metodele publice ale clasei pentru a defini toate proprietățile Alert Dialog-ului. După ce au fost scrise acțiunile corespunzătoare Builder-ului, dialogul se creează folosind `builder.create()`.

În exemplul din secventa de cod 3, creez un dialog cu ajutorul căruia utilizatorul poate alege dacă este speaker sau participant la prezentare. Alert Dialog-ul este format dintr-un mesaj şi două butoane .

Primul pas este adăugarea unui mesaj pentru Dialog folosind metoda `setMessage(CharSequence)`, apoi se stabilește dacă dialogul se poate închide cu butonul de

back al telefonului sau nu, folosind `setCancelable`(Boolean). Pentru fiecare buton se foloseşte una din metodele `set`(cum ar fi setPositiveButton()), care acceptă numele pentru buton şi un eveniment care defineşte acţiunea care va avea loc pe buton (`DialogInterface.OnClickListener`). Într-un dialog, se poate adăuga un singur buton de un anumit tip. Nu pot avea decât un buton `positive button`.

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("What are you?");
builder.setCancelable(true);
builder.setPositiveButton("Speaker", new
DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int id) {
            addSpeakerButtonsBar();
            dialog.cancel();
            }
      });
builder.setNeutralButton("Uşer", new
DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int id) {
            loginDialog();
      }
});
AlertDialog alert = builder.create();
alert.show();
```

Secvenţa de cod 3: Alert Dialog

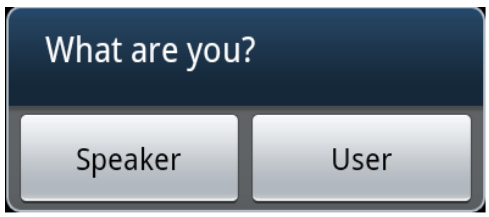

Figura 9: Alert dialog

În aplicație, folosesc AlertDialog şi pentru a crea o listă cu întrebări, din care utilizatorul poate selecta (secvenţa de cod 4).

Primul pas în crearea unui alert dialog de tip listă este adăugarea titlului folosind metoda `setTitle(CharSequence)`. Apoi se adăugă lista de elemente selectabile folosind metoda `setItems`() care acceptă șiruri de elemente pentru a fi afișate şi în final un listener(DialogInterface.OnClickListener) care definește acțiunile care au loc atunci când utilizatorul selectează un anumit element din listă.

```
final CharSequence[] items_list = {"Question1", "Question2",
"Question3","Question4"};
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Questions");
builder.setItems(items_list, new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int item) {
            Toast.makeText(MuPDFActivity.this, "item "+item+"
      clicked", Toast.LENGTH_SHORT).show();
      }
});
```

Secvenţa de cod 4: Lista cu întrebări

Metoda exemplificată în secvenţa de cod 4 poate fi folosită în cazul întrebărilor de dimensiuni mici. În cazul în care utilizatorul adresează întrebări foarte lungi, metoda nu mai este utilă deoarece întrebările vor fi afișate incomplet (se va afișa doar prima parte a întrebării, urmată de puncte de suspensie). De aceea, soluția găsită de mine a fost extinderea funcționalității clasei **Array Adapter.**

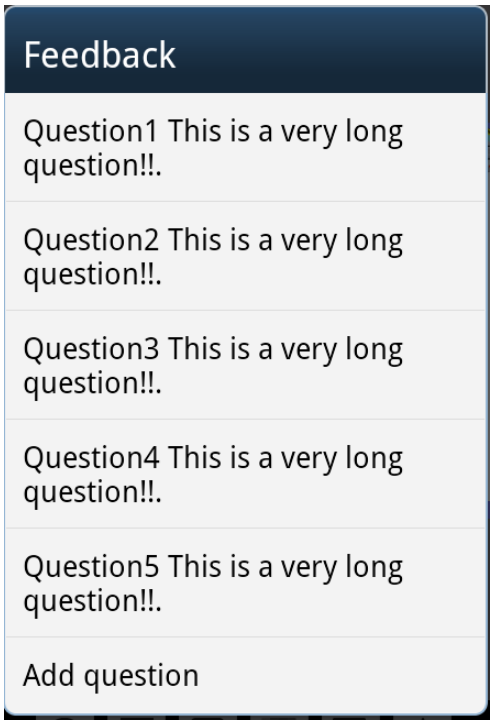

Figura 10: Lista cu întrebări

În Android, listele sunt implementate folosind modelul MVC (Model View Controller). Modelul este reprezentat de datele ce trebuie afișate, View-ul este reprezentat de lista propriu-zisă, Controller-ul este programul care controlează modul de afișare.

Un View de tip **ListView** poate fi implementat în orice tip de activitate (Activity). Android pune la dispoziție un tip special de activitate, numit **ListActivity**. [19]

Lista de tip ListView primește datele prin intermediul unui adaptor. Deci pentru a implementa o listă, trebuie scrisă componenta Adaptor a ei. Adaptorul definește cum va arăta fiecare rând din listă. Din punct de vedere al programării, acest lucru presupune crearea unui obiect care să implementeze interfața **ListAdapter**. Una din modalitățile prin care se poate implementa ListAdapter este ArrayAdapter.

Array Adapter (figura 11) este o clasă publică Android care poate fi folosită pentru a implementa liste simple, ce conțin elemente cu o singură linie de text. Poate primi la intrare obiecte Java de orice tip pe care le asociază unui View din layout. Acesta clasă presupune că toate elementele sunt stocate într-un şir (ex: Object[]) sau o listă (orice obiect ce implementează interfața List<type>). Pentru a folosi ArrayAdapter trebuie implementată metoda `getView`(), din clasa `BaseAdapter`. Această metodă ajută la determinarea aspectului rândului şi a felului cum sunt mapate datele pe View-uri în layout–ul curent. ArrayAdapter extinde clasa BaseAdapter. [19]

```
public View getView (int position, View convertView, ViewGroup list)
{
        // funcția trebuie să întoarcă view-ul de pe poziția position din lista
        // convertView este un element din lista ce nu mai este vizibil şi poate fi convertit
}
```

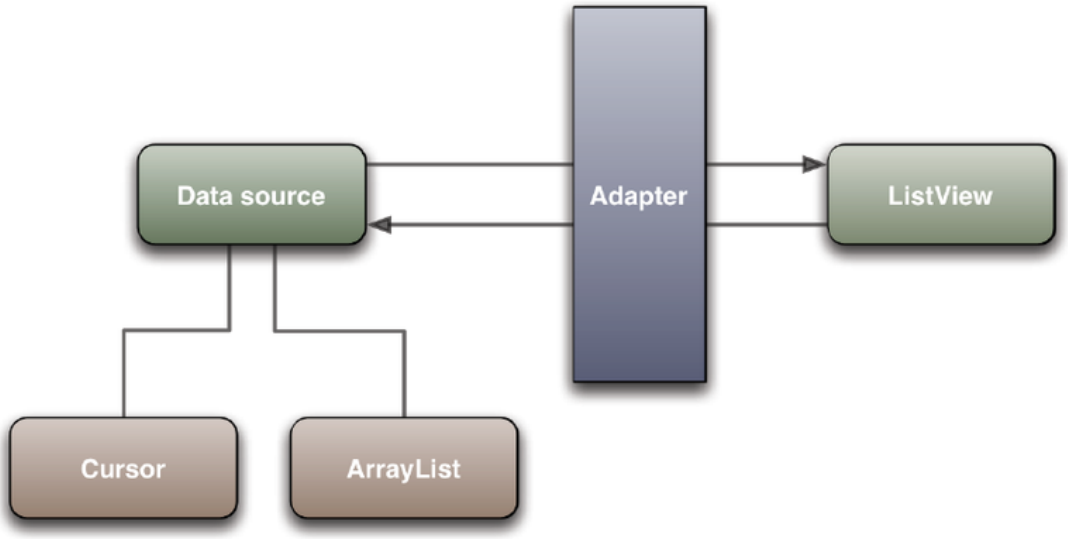

Figura 11 : List adapter [19]

În exemplul din secvenţa de cod 5 este implementată metoda getView a clasei MultilineAdapter care extinde Array Adapter, pentru a putea obține o listă cu întrebări pe mai multe rânduri.

```
public class MultilineAdapter extends ArrayAdapter {
//variabile …
public MultilineAdapter(Activity activity, List objects) {…}
@Override
 public View getView(int position, View convertView, ViewGroup
parent) {
      View rowView = convertView;
      MultilineView mView = null;
      if(rowView == null){
            LayoutInflater inflater = activity.getLayoutInflater();
             rowView    =    inflater.inflate(R.layout.multiline_item,
null);
              mView = new MultilineView();
              mView.mq = (TextView) rowView.findViewById(R.id.mq);
              rowView.setTag(mView);
      } else {   mView = (MultilineView) rowView.getTag(); }
            Question question = (Question) questions.get(position);
           mView.mq.setText(question.q);
      return rowView; }}
```

Secventa de cod 5: Implementarea metodei GetView

O implementare eficientă a ArrayAdapter presupune folosirea parametrului `convertView`, așa cum se întâmplă şi în cazul meu. Acesta este fie null, caz în care trebuie ignorat, fie un obiect întors anterior de către `getView()`, însă obiect care nu mai este vizibil. Ideea este că, în loc de a crea un nou View de fiecare dată, să se refolosească View-urile create anterior şi care nu mai sunt vizibile. În exemplul din secvenţa de cod 5, se folosește şi Layout Inflater pentru a parsa fișierul .xml.

Folosind Array Adapter şi BaseAdapter se pot construi şi liste mai complexe, care pot conține şi alte elemente (ex: imagini).

În realizarea interfeței clientului participant la prezentare, am folosit şi obiecte de tip **EditText** pentru câmpurile unde utilizatorul se autentifică şi pentru câmpul unde adăugă o întrebare.

EditText este o clasă publică Android care extinde clasa TextView, afișează textul introdus de utilizator şi permite editarea acestuia. Metodele `setWidht(int pixels)`, `setHeight(int pixels)`, `setHint(CharSequence hint)` sunt folosite pentru a seta dimensiunile ferestrei de text.

5.1.2 Interfaţa clientului prezentator

Prezentatorul, ca şi utilizatorul, primește slide-urile de pe server atunci când deschide aplicaţia. Acesta are un alt set de facilitați, având o interfața mai simplă decât a clientului participant.

La final, speakerul primește feedback-ul agregat al prezentării. Acesta are două butoane de feedback, unul din ele este o listă (de tip `alert dialog` cum este şi în cazul participantului) ce contine feedback-ul pe toată prezentarea, iar celălalt buton este o lista cu feedback-ul pe fiecare slide.

Atunci când participantul face o selecție pe o porțiune de text, secțiunea selectată va avea o anumită culoare, stabilită din aplicaţie. Speakerul va vedea în fereastra de feedback întrebările colorate în funcție de zonă pe care a fost făcută selecția. Exemplu: trei participanţi au selectat trei zone diferite dintr-un slide, selecțiile lor vor avea culori diferite şi întrebările care vor apărea în lista cu feedback a prezentatorului vor avea aceleași culori cu selecțiile făcute.

Şi în cazul butoanelor `+1`, `ambiguous/unclear` şi `citation or proof needed`, culorile selecțiilor făcute de participant corespund cu cele ale butoanelor. Butoanele `+1`, `ambiguous/unclear` şi `citation or proof needed` nu mai au aceleași funcționalități ca în cazul participantului; în acest caz sunt folosite doar pentru a vedea speakerul numărul de `+1`, `ambiguous/unclear` şi `citation or proof needed` care au fost marcate pe fiecare slide.

Din punct de vedere al implementării, având elemente de interfață asemănătoare cu participantul la prezentare, am folosit, în general, aceleași clase şi metode precizate şi exemplificate în subcapitolul precedent.

5.1.3 Folosirea MuPDF pentru vizualizarea PDF-urilor

MuPDF este un software open source care implementează randarea şi parsarea fişierelor PDF. În implementarea proiectului, am pornit de la varianta portată pentru Android a acestui software care include librăria mudpdf, o interfață mupdf jni pentru Android şi aplicația Android pdfViewer (.apk). Clasele proiectului Android la care am adăugat noi funcţionalităţi sunt: `MuPDFActivity.java`, `MuPDFCore.java şi PixmapView.java`. Randarea efectivă a `pdf`-ului, reprezentat printr-o imagine se face în cod nativ C.

Clasa MuPDFCore.java apelează funcţii native Java pentru deschiderea unui fişier, obţinerea dimensiunilor unei pagini, desenarea efectivă a unei pagini. Fiecare pagină a unui document `.pdf` este tratată ca o imagine (de tip `bitmap`).

Clasa PixmapView.java extinde clasa Android `SurfaceView` şi se ocupă de evenimentele şi acţiunile care au loc pe suprafaţa `pdf`-ului. Printre aceste evenimente sunt şi cele care au loc atunci

când este atins ecranul. În această clasă, au fost adăugate, în cadrul proiectului, funcţionalităţi pentru a realiza selecţia de text din document şi pentru a simula trecerea pe o altă pagină în `pdf`.

Clasa MuPDFActivity.java este clasa în care am adăugat butoanele din interfaţa utilizatorului şi funcţionalităţile lor

Comunicaţia dintre codul Android ce rulează în maşina virtuală de Android şi aplicaţiile native (programe specifice de la o platformă hardware şi sistemul de operare) sau biblioteca de C se fac prin intermediul framework-ului **Java Native Interface** (JNI).

JNI permite scrierea unor metode native pentru a gestiona situaţiile în care o aplicaţie nu poate fi scrisă în totalitate în Java. Poate fi de asemenea folosit pentru a modifica o aplicaţie existentă scrisă într-un alt program, pentru a fi accesibilă pentru aplicaţiile Java. Folosind JNI, o metodă nativă poate folosi obiecte Java în acelaşi mod în care le foloseşte şi codul Java. Altfel spus, o metodă nativă poate crea obiecte Java şi le poate folosi pentru a îndeplini propriile sarcini. Poate, de asemenea, folosi obiecte create în cod Java.

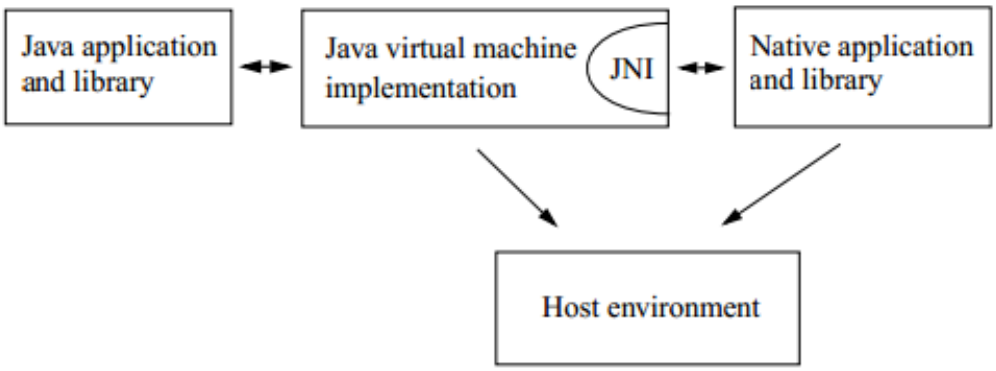

Figura 12: Rolul JNI [20]

6. Utilizarea aplicaţiei

Scenariul complet al aplicaţiei: Anca se pregăteşte să susţină o prezentare în faţa colegilor săi de la cursul de IA. Înainte de a începe prezentarea porneşte aplicaţia SmartPresentation de pe telefonul mobil. Acelaşi lucru îl fac şi colegii săi. Prezentarea Ancăi este descărcată automat de pe server pe dispozitivele mobile ale tuturor participanţilor. Pentru a vedea conţinutul prezentării utilizatorii trebuie să spună cine sunt (speaker sau utilizator din auditoriu) şi să se autentifice folosind un nume (figura 13).

Figura 13: Autentificare utilizator

După autentificare, utilizatorii pot vedea slide-urile prezentării şi butoanele cu funcţionalităţi corespunzătoare tipului de utilizator ales. În figura 14 se pot vedea butoanele din cadrul celor două tipuri de interfaţă, speaker şi persoana din auditoriu.

Figura 14: Butoane interfaţa speaker vs. Butoane interfaţa utilizator din sală

Andrei poate urmări prezentarea în propriul său ritm, alocând timpul pe care îl consideră necesar fiecărui slide. Dacă vrea să vadă la ce slide a ajuns Anca poate apăsă butonul `Go live`. Acest buton îl va poziţiona pe slide-ul la care este în acel moment prezentatorul.

Bogdan are o întrebare în legătură cu un subiect abordat de Anca. Pentru a scrie întrebarea trebuie să activeze întâi selecţia, folosind butonul de selecţie, apoi să adauge întrebarea în lista de întrebări. Poate selecta o zonă de text sau o poză de pe slide. După ce urmează aceşti paşi, observă că Dragoş a pus deja o întrebare pe acelaşi slide, aşa că foloseşte butonul +1 pentru a susţine întrebarea colegului său. În figura 15 este prezentat scenariul de selecţie a unei imagini din pdf şi adăugarea unei întrebări noi:

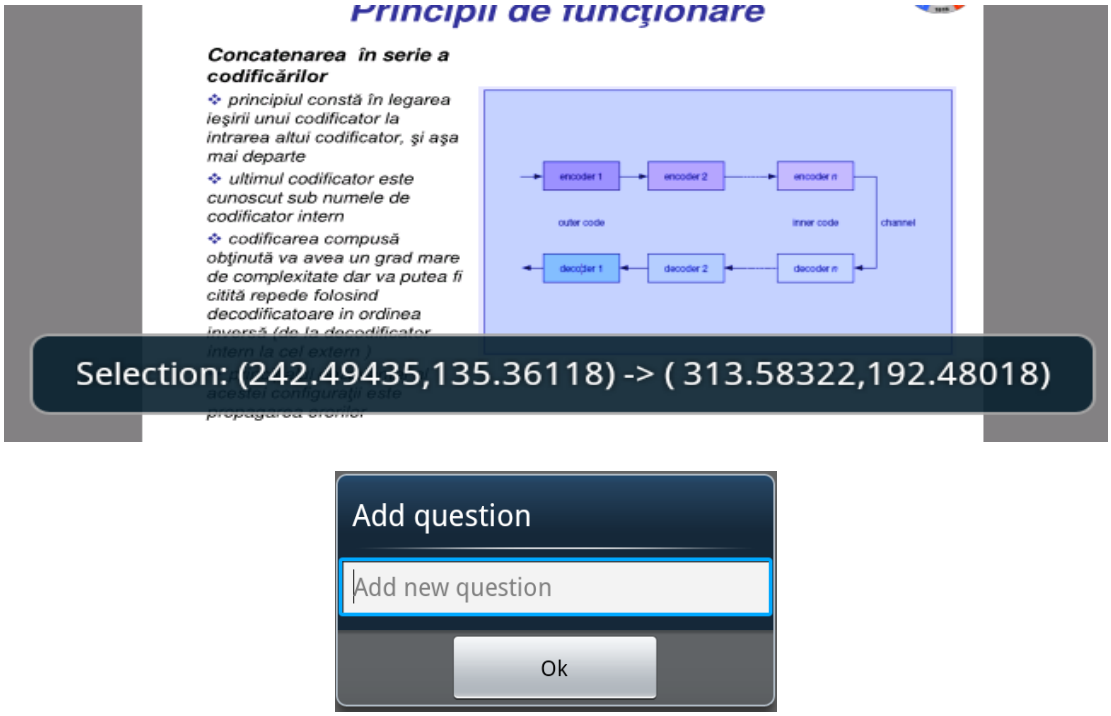

Figura 15: Scenariu selecţie din document şi adăugare întrebare

Butonul +1 poate fi folosit şi individual de întrebări, pentru a da feedback pozitiv unei porţiuni de text sau unei imagini.

Bogdan nu înţelege cum a fost aplicată o anumită formulă şi vrea să îi ceară Ancăi o demonstraţie detaliată. Pentru aceasta poate folosi butoanele `ambigous` şi `proof needed` din interfaţă.

Fiecare utilizator vede feedback-ul sub forma unei succesiuni de mai multe liste cu întrebări. Lista principală conţine întrebările cele mai generale adresate de auditoriu, întrebările rădăcină ale arborelui de întrebări. Presupun că Bogdan a adresat întrebarea “Ce înseamnă E?”. În cazul în care mai există şi alte întrebări asemănătoare ca semantica cu întrebarea lui Bogdan, le va putea vedea

prin simpla apăsare pe întrebarea sa din lista de feedback. Astfel se va deschide o nouă listă care poate cuprinde întrebările : “Ce înseamnă E în formula ta?”, “Care este rolul lui E în formula ta?”. Acelaşi lucru este exemplificat în figura 16.

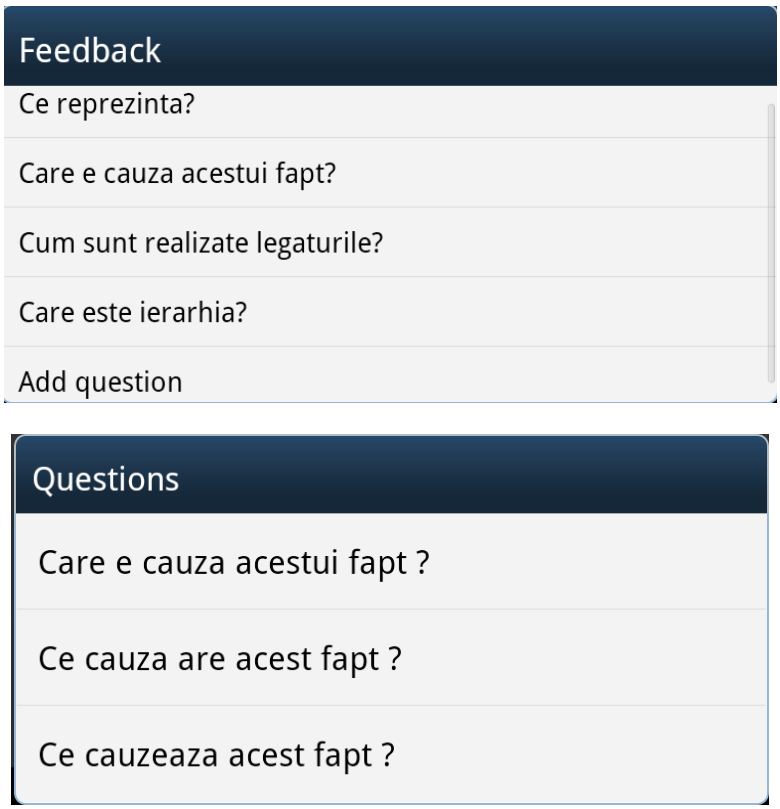

Figura 16: Liste de feedback

Orice utilizator poate şterge o întrebare proprie din listă şi poate da +1 unei întrebări puse de altcineva.

În cazul în care utilizatorul a selectat o zonă de text pe care a dat feedback ambigous sau proof needed, a doua oară când apăsă butonul, aplicaţia consideră că este posibil să dorească să îşi retragă feedback-ul. De aceea apare o fereastră în care este întrebat dacă vrea să renunţe la feedback-ul ales anterior.

La finalul prezentării, Anca poate vedea feedback-ul agregat primit de la toate persoanele din sală. Pentru a vizualiza întrebările adăugate pe parcursul prezentării, Anca are 2 butoane. Unul dintre ele îi arată feedback-ul pentru toată prezentarea, iar celălalt feedback-ul pentru un singur slide. Butoanele din interfaţa ei au grafica diferită de butoanele audienţei, unele din ele fiind colorate. În funcţie de culoarea butonului din interfaţă, Anca vede feedback-ul primit. Astfel, pentru că butonul +1 este galben, atunci când apăsă pe el va putea vedea toate selecţiile pe care s-a dat ca feedback +1

în culoarea galben. La fel şi în cazul butoanelor `ambiguous` şi `proof needed` care sunt colorate. În cazul întrebărilor, fiecare zonă selectată pe care au fost adăugate întrebări va avea o anumită culoare. Atunci când apăsă butonul de Feeback, Anca vede textul întrebărilor în aceeaşi culoare cu zona selectată de utilizator din text. În figura 17 este exemplificat feedback-ul din perspectiva prezentatorului pentru butoanele +1, ambiguous şi proof needed.

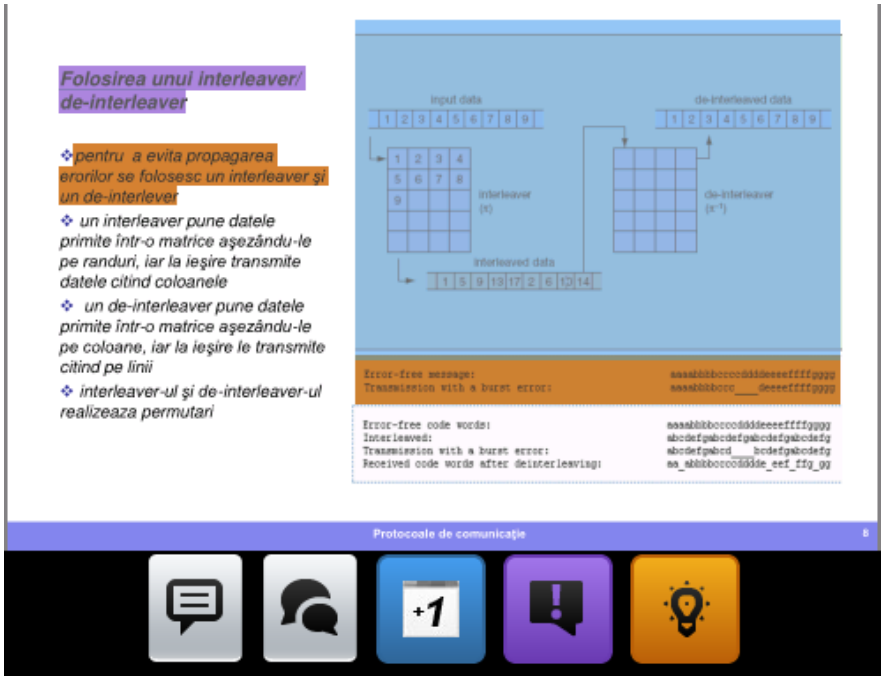

Figura 17: Feedback din perspectiva prezentatorului

7. Concluzii şi dezvoltări ulterioare

Aplicaţia dezvoltată este destinată grupurilor de persoane care doresc să participe la o prezentare în care să se poată implica şi să poată interacţiona cu prezentatorul, exprimându-şi acordul sau dezacordul cu privire la anumite părţi ale lucrării/prezentării.

Scopul principal al proiectului este îmbunătăţirea relaţiei dintre prezentator şi auditoriu, dar şi dintre persoanele din sală pe parcursul unei prezentări. Pentru a atinge acest obiectiv am ales să dezvoltăm o aplicaţie pe dispozitve mobile cu Android care să permită utilizatorilor urmărirea prezentării atât în timp real, cât şi în propriul ritm şi adăugarea, pe parcurs, de feedback pozitiv sau negativ pe o anumită zonă selectată de text/imagine. La finalul prezentării, prezentatorul va putea analiza şi răspunde la întrebările şi adnotările primite.

Folosirea aplicaţiei are numeroase avantaje atât pentru utilizatori, cât şi pentru prezentator. Avantajul principal al utilizatorilor este interacţiunea, atât cu cei din sală pentru că pot vedea întrebările notate de fiecare şi pot interacţiona cu persoanele care au interese comune cu ei, dar şi cu prezentatorul căruia îi pot adresa întrebări şi cere explicaţii. Avantajul prezentatorului este în primul rând faptul că poate obţine feedback relevant de la un număr destul de mare de persoane, poate răspunde feeback-ului primit şi îşi poate îmbunătăţi pe viitor prezentările luând în considerare observaţiile primite.

Am preferat sistemul de operare Android pentru dezvoltarea aplicaţiei, în defavoarea celorlalte, mai ales datorită faptului că este cea mai răspândită tehnologie pe dispozitivele mobile actuale, este bine documentat şi intuitiv din punct de vedere al modului de utilizare. Pentru a implementa partea de server am folosit tehnologii ca SQLite, Google Protocol Buffers, Jersey şi Grizzly.

Principalul obiectiv viitor al aplicaţiei este îmbunătăţirea părţii de feeback din perspectiva prezentatorului: în lista de feebback vor fi afişate primele 10 întrebări în ordinea relevanţei şi a numărului de apariţii, în dreptul fiecărei întrebări din interfaţa prezentatorului va apărea un timp estimat pentru a răspunde la acea întrebare, raportat la timpul rămas. Astfel, prezentatorul îşi va putea gestiona mai bine timpul alocat răspunsurilor şi explicaţiilor. Atât utilizatorii,cât şi prezentatorul vor avea afişat,în aplicaţie, timpul rămas până la finalul prezentării,

Un alt obiectiv viitor al aplicaţiei este adăugarea de scenarii noi, care ar putea dovedi necesitatea acesteia pe scară largă (ex: prezentări şi conferinţe pe Internet). Pentru realizarea acestui obiectiv, o idee ar fi ca aplicaţia să deţină o funcţionalitate ce personalizează o prezentare pe baza unor profiluri, asociate unui tip de scenariu, în măsură să codifice anumite caracteristici pentru o experienţă de utilizare mai bună. Un exemplu de acest tip este implementarea streaming-ului audio şi video pentru prezentări pe Internet.

Bibliografie

[1] *Google AI-MAS Group*. Preluat de pe http://aimas.cs.pub.ro/androidEDU

[2] *Interactiunea om- calculator.* Preluat de pe

http://www.scribd.com/doc/92389473/Interactiunea-Om-Calculator

[3] Iosif, G. *Activitatea cognitivă a operatorului uman.* Bucuresti: Editura Academiei.

[4] Trăuşan-Matu, Ş. (2000). *Interfaţarea evoluată om-calculator.* Bucuresti: Editura MatrixRom.

[5] *Proiectarea intefeţelor utilizator in conducerea avansata – Criterii ergonomice.*Preluat de pe http://www.aie.ugal.ro/sica/curs/Curs5.pdf

[6] Gargenta, M. (2011). *Learning Android.*

[7] Burnette, E. (2010). Pragmatic Hello Android, Third Edition.

[8] Ehringer, D. (2010). The Dalvik virtual machine architecture.

[9] *Design Patterns - Model View Controller (MVC) Pattern* . Preluat, de pe http://www.bogotobogo.com/DesignPatterns/mvc_model_view_controller_pattern.php

[10] *Model View Controller Pattern*. Preluat de pe http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/mvc.html

[11] *Google Protocol Buffers Java documentation*. Preluat de pe http://code.google.com/intl/ro-RO/apis/protocolbuffers/docs/javatutorial.html

[12] *REST* . Preluat de pe http://searchsoa.techtarget.com/definition/REST

[13] *JAX-RS: Developing RESTful Web Services in Java*. Preluat de pe http://www.devx.com/Java/Article/42873/1954

[14] *Jersey Documentation* . Preluat de pe http://jersey.java.net/nonav/documentation/latest/user-guide.html#d4e1972

[15] Owens, M. *The Definitive Guide to SQLite* . Preluat de pe http://www.campusnewsjbp.com/campus_news/placement_papers/Sqlite.pdf

[16] *MuPDF* . Preluat de pe http://www.mupdf.com/

[17] *Building an Android application*. Preluat de pe

http://www.skill-guru.com/blog/2011/01/10/building-an-android-application/

[18] Badita, M. *Tutoriale Android*. Preluat de pe

http://magdabadita.wordpress.com/tag/android-tutorials/

[19] *Liste in Android*. Preluat de pe http://pdm.ipworkshop.ro/mediawiki/index.php/Lab3

[20] Liang, S. *The Java Native Interface, Programmer's Guide and Specification.* Preluat de pe http://java.sun.com/docs/books/jni/download/jni.pdf

[21] Tufiş, D. C. (2008). *DIAC+: A Professional Diacritics Recovering System.* Preluat de pe http://www.racai.ro/~tufis/papers/Tufis-Ceausu-LREC2008.pdf

Anexe

Clasa MuPDFActivity.java

```
package com.artifex.mupdf;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnCancelListener;
import android.os.Bundle;
import android.os.Environment;
import android.view.*;
import android.view.View.OnClickListener;
import android.widget.*;
import java.io.File;
import java.util.ArrayList;
import com.artifex.mupdf.PixmapView;

public class MuPDFActivity extends Activity
{
	/* The core rendering instance */
	private MuPDFCore core;
	private static int MOD_SPEAKER = 1;
	private static int MOD_USER = 2;
	private int mod;
	private int contorAmbiguu = 0;
	private int contorProof = 0;

	private MuPDFCore openFile()
	{
		String storageState = Environment.getExternalStorageState();
		File path, file;
		MuPDFCore core;
		if (Environment.MEDIA_MOUNTED.equals(storageState))
		{
			System.out.println("Media mounted read/write");
		}
		else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(storageState))
		{
			System.out.println("Media mounted read only");
		}
		else
		{
			System.out.println("No media at all! Bale!\n");
			return null;
		}
		path = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);
		file = new File(path, "test.pdf");
		System.out.println("Trying to open "+file.toString());
		try
		{
			core = new MuPDFCore(file.toString());
		}
		catch (Exception e)
		{
```

```
                        System.out.println(e);
                        return null;
                }
                return core;
        }
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState)
        {
                super.onCreate(savedInstanceState);

                chooseUser();
        }
        public void init(int mod) {
                PixmapView pixmapView;
                if (core == null) {
                        core = (MuPDFCore)getLastNonConfigurationInstance();
                }
                if (core == null) {
                        core = openFile();
                }
                if (core == null)
                {
                        /* FIXME: Error handling here! */
                        return;
                }
                pixmapView = new PixmapView(this, core);
                /* Now create the UI */
                RelativeLayout layout;
                LinearLayout bar;
                MyButtonHandler bh = new MyButtonHandler(pixmapView);
                bar = new LinearLayout(this);
                bar.setOrientation(LinearLayout.HORIZONTAL);
                LinearLayout.LayoutParams butMargin = new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.WRAP_CONTENT, LinearLayout.LayoutParams.WRAP_CONTENT);
                butMargin.setMargins(5, 0, 5, 0);
                bh.goLive = new Button(this);
                bh.goLive.setOnClickListener(bh);
                bh.goLive.setCompoundDrawablesWithIntrinsicBounds(R.drawable.live, 0,0,0);
                if (mod == MOD_USER) bar.addView(bh.goLive);
                bh.selectButton = new Button(this);
                bh.selectButton.setOnClickListener(bh);
                bh.selectButton.setCompoundDrawablesWithIntrinsicBounds(R.drawable.select, 0,0,0);
                if (mod == MOD_USER) bar.addView(bh.selectButton);
                bh.feedbackButton = new Button(this);
                bh.feedbackButton.setOnClickListener(bh);
                bh.feedbackButton.setCompoundDrawablesWithIntrinsicBounds(R.drawable.feedback, 0,0,0);
                bar.addView(bh.feedbackButton);
                bh.feedbackSlide = new Button(this);
                bh.feedbackSlide.setOnClickListener(bh);
                bh.feedbackSlide.setCompoundDrawablesWithIntrinsicBounds(R.drawable.feedback_slide, 0,0,0);
                if(mod == MOD_SPEAKER)
                        bar.addView(bh.feedbackSlide);
                bh.plus1Button = new Button(this);
                bh.plus1Button.setOnClickListener(bh);
```

```
		bh.plus1Button.setCompoundDrawablesWithIntrinsicBounds(R.drawable.plus, 0,0,0);
		if (mod == MOD_SPEAKER) {
			bh.plus1Button.setBackgroundResource(R.layout.blue_button);
			bh.plus1Button.setLayoutParams(butMargin);
		}
		bar.addView(bh.plus1Button);
		bh.ambiguuButton = new Button(this);
		bh.ambiguuButton.setOnClickListener(bh);
		bh.ambiguuButton.setCompoundDrawablesWithIntrinsicBounds(R.drawable.ambiguu, 0,0,0);
		if (mod == MOD_SPEAKER) {
			bh.ambiguuButton.setBackgroundResource(R.layout.purple_button);
			bh.ambiguuButton.setLayoutParams(butMargin);
		}
		bar.addView(bh.ambiguuButton);
		bh.proofButton = new Button(this);
		bh.proofButton.setOnClickListener(bh);
		bh.proofButton.setCompoundDrawablesWithIntrinsicBounds(R.drawable.proof, 0,0,0);
		if (mod == MOD_SPEAKER){
			bh.proofButton.setBackgroundResource(R.layout.yellow_button);
			bh.proofButton.setLayoutParams(butMargin);
		}
		bar.addView(bh.proofButton);

		layout = new RelativeLayout(this);
		layout.setLayoutParams(new RelativeLayout.LayoutParams(
					RelativeLayout.LayoutParams.FILL_PARENT,
					RelativeLayout.LayoutParams.FILL_PARENT));
		layout.setGravity(Gravity.FILL);

		RelativeLayout.LayoutParams barParams =
			new RelativeLayout.LayoutParams(
					RelativeLayout.LayoutParams.WRAP_CONTENT,
					RelativeLayout.LayoutParams.WRAP_CONTENT);
		barParams.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
		barParams.addRule(RelativeLayout.CENTER_HORIZONTAL);
		bar.setId(100);
		layout.addView(bar, barParams);
		RelativeLayout.LayoutParams pixmapParams =
			new RelativeLayout.LayoutParams(
					RelativeLayout.LayoutParams.FILL_PARENT,
					RelativeLayout.LayoutParams.FILL_PARENT);
		pixmapParams.addRule(RelativeLayout.ABOVE,100);
		layout.addView(pixmapView, pixmapParams);
		setContentView(layout);
}
public void addSpeakerButtonsBar(){
		Toast.makeText(MuPDFActivity.this, "speaker", Toast.LENGTH_SHORT).show();
		init(MOD_SPEAKER);
		mod = MOD_SPEAKER;
}
public void addUserButtonsBar(){
		Toast.makeText(MuPDFActivity.this, "user", Toast.LENGTH_SHORT).show();
		init(MOD_USER);
		mod = MOD_USER;
}
```

```
public void chooseUser(){
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setMessage("What are you?");
        builder.setCancelable(true);
        builder.setPositiveButton("Speaker", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                addSpeakerButtonsBar();
              dialog.cancel();
            }
          });
        builder.setNeutralButton("User", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
              loginDialog();
            }
          });
        AlertDialog alert = builder.create();

        alert.show();
}
protected void loginDialog(){
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setMessage("Login to continue");
        LinearLayout ll = new LinearLayout(this);
        ll.setOrientation(LinearLayout.VERTICAL);
        EditText username = new EditText(this);
        username.setHint("Username");
        username.setWidth(400);
        ll.addView(username);
        builder.setView(ll);

        builder.setPositiveButton("Ok", new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int id) {
          addUserButtonsBar();
          dialog.cancel();
      }
    });
        AlertDialog alert = builder.create();
        alert.show();
}
public void finalFeedback(){
        final AlertDialog alert;
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(true);
        LinearLayout ll = new LinearLayout(this);
        ll.setOrientation(LinearLayout.HORIZONTAL);
        Button plus1 = new Button(this);
        Button delete = new Button(this);
        plus1.setCompoundDrawablesWithIntrinsicBounds(R.drawable.plus, 0,0,0);
        delete.setCompoundDrawablesWithIntrinsicBounds(R.drawable.delete, 0,0,0);
        ll.addView(plus1);
        ll.addView(delete);
        builder.setView(ll);
        alert = builder.create();
        delete.setOnClickListener(new OnClickListener() {
                public void onClick(View v) {
```

```
                        Toast.makeText(MuPDFActivity.this, "delete my question!", Toast.LENGTH_SHORT).show();
                                alert.dismiss();
                        }
                });
plus1.setOnClickListener(new OnClickListener() {
                        public void onClick(View v) {
                                Toast.makeText(MuPDFActivity.this, "+1!", Toast.LENGTH_SHORT).show();
                                alert.dismiss();
                        }});
                alert.setOnCancelListener(new OnCancelListener() {
                        public void onCancel(DialogInterface dialog) {
                                createList2(mod);
                        }});
                alert.show();
        }
        public void addQuestion(){
                final AlertDialog alert;
                AlertDialog.Builder builder = new AlertDialog.Builder(this);
                builder.setTitle("Add question");
                LinearLayout ll = new LinearLayout(this);
                ll.setOrientation(LinearLayout.VERTICAL);

                EditText et = new EditText(this);
                et.setHint("Add new question");
                et.setWidth(400);
                ll.addView(et);
                builder.setView(ll);
                builder.setPositiveButton("Ok", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                  dialog.cancel();
            } });
                alert = builder.create();
                alert.show();
        }
        public void createList2(final int mod){

                final CharSequence[] items_list2 = {"Question1 Question Question11 Question1 Question1"};
                AlertDialog.Builder builder = new AlertDialog.Builder(this);
                builder.setTitle("Questions");
                builder.setItems(items_list2, new DialogInterface.OnClickListener() {
                  public void onClick(DialogInterface dialog, int item) {
                        Toast.makeText(MuPDFActivity.this, "item "+item+" clicked", Toast.LENGTH_SHORT).show();
                                if(mod == MOD_USER){
                                        finalFeedback();
                                }}});
                builder.setOnCancelListener(new OnCancelListener() {
                        public void onCancel(DialogInterface dialog) {
                                createList(mod);
                        }
                });
                AlertDialog alert = builder.create();
                alert.show();
        }
        public void createList(final int mod) {
                final ArrayList<Question> items = new ArrayList<Question>();
```

```
		for (int i = 1; i<6; i++)
			items.add(new Question("Question"+i+" This is a very long question!!."));
		if (mod == MOD_USER)
			items.add(new Question("Add question"));
		AlertDialog.Builder builder = new AlertDialog.Builder(this);
		builder.setTitle("Feedback");
		builder.setCancelable(true);
		builder.setAdapter(new MultilineAdapter(this, items), new DialogInterface.OnClickListener() {
		  public void onClick(DialogInterface dialog, int item) {
		Toast.makeText(MuPDFActivity.this, "question "+item+" clicked", Toast.LENGTH_SHORT).show();
				if (item == (items.size() - 1) && mod == MOD_USER) {
				addQuestion();
			} else {
				createList2(mod);
			}
		  }
		});
		AlertDialog alert = builder.create();
		alert.show();
}
public Object onRetainNonConfigurationInstance()
{
		MuPDFCore mycore = core;
		core = null;
		return mycore;
}
public void onDestroy()
{
		if (core != null)
			core.onDestroy();
		core = null;
		super.onDestroy();
}
private class MyButtonHandler implements OnClickListener
{
		Button goLive;
		Button selectButton;
		Button feedbackButton;
		Button feedbackSlide;
		Button plus1Button;
		Button ambiguuButton;
		Button proofButton;
		PixmapView pixmapView;
		public MyButtonHandler(PixmapView pixmapView)
		{
			this.pixmapView = pixmapView;
		}
		public void onClick(View v)
		{
			if (this.pixmapView.selectActiv == true && v != selectButton)
			{
				pixmapView.getThread().resetSelect();
				if (this.pixmapView.doneSelection == false)
				{
```

```
                                        Toast.makeText(MuPDFActivity.this, "First make a selection",
Toast.LENGTH_SHORT).show();
                                }
                                else
                                {
                                if (pixmapView.selectionString.charAt(pixmapView.selectionString.length() - 1) == '.')
                                        {
                                                this.pixmapView.doneSelection = false;
                                                Toast.makeText(MuPDFActivity.this, "Nothing was selected",
Toast.LENGTH_SHORT).show();
                                        }
                                        else
                                        {
                                                Toast.makeText(MuPDFActivity.this, "Selection: " +
this.pixmapView.selectionString, Toast.LENGTH_SHORT).show();
                                        }}}
                        if(v == goLive)
                                Toast.makeText(MuPDFActivity.this, "Go live!", Toast.LENGTH_SHORT).show();
                        else if(v == selectButton)
                        {
                                if (this.pixmapView.selectActiv == false)
                                {
                                        this.pixmapView.selectActiv = true;
                                        Toast.makeText(MuPDFActivity.this, "Make selection!",
Toast.LENGTH_SHORT).show();
                                }
                                else
                                {
                                        this.pixmapView.selectActiv = false;
                                        this.pixmapView.doneSelection = false;
                                        pixmapView.getThread().resetSelect();
                                        this.pixmapView.getThread().forceRedraw();
                                        Toast.makeText(MuPDFActivity.this, "Stopped selection!",
Toast.LENGTH_SHORT).show();
                                }
                        }
                        else if(v == feedbackButton)
                        createList(mod);
                        else if(v == feedbackSlide)
                                createList(mod);
                        else if(v == plus1Button && mod == MOD_USER)
                                Toast.makeText(MuPDFActivity.this, "+1!", Toast.LENGTH_SHORT).show();
                        else if(v == ambiguuButton && mod == MOD_USER){
                                contorAmbiguu++;
                                if(this.pixmapView.selectActiv == true && contorAmbiguu == 1){
                                        Toast.makeText(MuPDFActivity.this, "Ambiguu",
Toast.LENGTH_SHORT).show();
                                }
                                else if(this.pixmapView.selectActiv == true && contorAmbiguu == 2) {

                                AlertDialog.Builder builder = new AlertDialog.Builder(MuPDFActivity.this);
                                        builder.setMessage("Do you want to delete ambiguu selection?");
                                builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                                    public void onClick(DialogInterface dialog, int id) {
                                        pixmapView.selectActiv = false;
```

```
                                        pixmapView.doneSelection = false;
                                        pixmapView.getThread().resetSelect();
                                        pixmapView.getThread().forceRedraw();
                                        dialog.cancel();
                                    }
                                });
                            builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
                                    public void onClick(DialogInterface dialog, int id) {
                                        dialog.cancel();
                                    }
                                });
                                    AlertDialog alert = builder.create();
                                    alert.show();
                            }
                            else {
                                    contorAmbiguu = 0;
                                    this.pixmapView.selectActiv = false;
                                    this.pixmapView.doneSelection = false;
                                    pixmapView.getThread().resetSelect();
                                    this.pixmapView.getThread().forceRedraw();
                                    Toast.makeText(MuPDFActivity.this, "Make selection!",
Toast.LENGTH_SHORT).show();
                            }
                        }
                        else if(v == proofButton && mod == MOD_USER){
                            contorProof++;
                            if(this.pixmapView.selectActiv == true && contorProof == 1){
                                    Toast.makeText(MuPDFActivity.this, "Proof",
Toast.LENGTH_SHORT).show();
                            }
                            else if(this.pixmapView.selectActiv == true && contorProof == 2) {

                            AlertDialog.Builder builder = new AlertDialog.Builder(MuPDFActivity.this);
                                    builder.setMessage("Do you want to delete PROOF selection?");
                            builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                                    public void onClick(DialogInterface dialog, int id) {
                                        pixmapView.selectActiv = false;
                                        pixmapView.doneSelection = false;
                                        pixmapView.getThread().resetSelect();
                                        pixmapView.getThread().forceRedraw();
                                        dialog.cancel();
                                    } });
                            builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
                                    public void onClick(DialogInterface dialog, int id) {
                                        dialog.cancel();
                                    }});
                                    AlertDialog alert = builder.create();
                                    alert.show();}
                            else {
                                    contorProof = 0;
                                    this.pixmapView.selectActiv = false;
                                    this.pixmapView.doneSelection = false;
                                    pixmapView.getThread().resetSelect();
                                    this.pixmapView.getThread().forceRedraw();
                        Toast.makeText(MuPDFActivity.this, "Make selection!", Toast.LENGTH_SHORT).show();}}}}}
```