Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Dušan Kalanj

# Razvoj kriptografske mobilne aplikacije za uporabo v zdravstvu

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: prof. dr. Aleksandar Jurišić

Ljubljana 2013

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

*Tržaška 25*
*1001 Ljubljana, Slovenija*
*telefon: 01 476 84 11*
*01 476 83 87*
*faks: 01 426 46 47*
*01 476 87 11*
*www.fri.uni-lj.si*
*e-mail: dekanat@fri.uni-lj.si*

Št. naloge: 00138/2013
Datum: 15.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **DUŠAN KALANJ**

Naslov: **RAZVOJ KRIPTOGRAFSKE MOBILNE APLIKACIJE ZA UPORABO V ZDRAVSTVU**

**DEVELOPEMENT OF CRYPTOGRAPHIC MOBILE APPLICATION IN HEALTH CARE**

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Osrednji cilj naj bo razvoj spletne in mobilne aplikacije (gsm telefoni in tablice z operacijskim sistemom Android), ki omogoča varno rokovanje z digitalno zdravstveno kartoteko.
Delo predstavi kriptografske osnove kot so npr.
simetrična kriptografija (bločne in tokovne šifre) ter asimetrična kriptogrfija (dogovor o ključu, digitalni podpis in digitalna potrdila), ki so potrebne za varno implementacijo.
Naredi naj tudi pregled uporabljenih orodij in tehnologije (kot so Android, Cordova, podatkovne baze Oracle ipd).
Sledi podroben opis same aplikacije in dokumentacija njenega testiranja.

Mentor:

prof. dr. Aleksandar Jurišić

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Nikolaj Zimic

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Dušan Kalanj, z vpisno številko **63100216**, sem avtor diplomskega dela z naslovom:

*Razvoj kriptografske mobilne aplikacije za uporabo v zdravstv*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Aleksandra Jurišića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 12. septembra 2013 Podpis avtorja:

# Kazalo

**Seznam uporabljenih kratic in simbolov**

**Povzetek**

**Abstract**

**1 Uvod** **1**

**2 Dosedanje rešitve** **3**

2.1 SAP EMR Unwired . . . . . 3

2.2 mMR (mobile Medical Records) . . . . . 4

2.3 Medscape . . . . . 5

2.4 Povzetek aplikacij . . . . . 5

**3 Osnovni kriptografski principi in metode** **7**

3.1 Klasična kriptografija . . . . . 7

3.2 Moderna kriptografija . . . . . 8

**4 Uporabljene tehnologije in orodja** **15**

4.1 Delovni pogoji . . . . . 15

4.2 Jeziki . . . . . 17

4.3 Orodja . . . . . 18

4.4 Operacijski sistem Android . . . . . 20

4.5 Cordova . . . . . 23

4.6 Podatkovna baza Oracle . . . . . 26

4.7 Spletna storitev . . . . . 26
4.8 JSON . . . . . 27
4.9 SSL/TLS . . . . . 27

**5 Aplikacija E-kartoteka** **29**
5.1 Načrtovanje . . . . . 29
5.2 Delovanje in uporabniška izkušnja . . . . . 37
5.3 Testiranje . . . . . 50
5.4 Možne izboljšave . . . . . 51

**6 Sklepne misli** **57**

**Literatura** **57**

# Seznam uporabljenih kratic in simbolov

**Android** je operacijski sistem, ki poganja predvsem mobilne naprave.

**ADT** (Android Developer Tools) je skupek orodij za razvoj mobilnih aplikacij za operacijski sistem android.

**CSS** (Cascading Style Sheets) je slogovni jezik za oblikovanje spletnih strani.

**Čistopis** je berljiv tekst.

**Digitalno potrdilo** je orodje za zagotavljanje varne komunikacije.

**Eclipse** je integrirano razvojno okolje.

**Hibridne aplikacije** so vrsta mobilnih aplikacij, ki jih je moč naložiti na več platform.

**HTML** (HyperText Markup Language) je označevalni jezik za razvoj spletnih strani.

**HTTP** (HyperText Transfer Protocol) je protokol, ki skrbi za ustrezen prenos informacij preko spleta.

**Java** je splošno-namenski programski jezik.

**JavaScript** je skriptni programski jezik za razvoj spletnih vsebin.

**JSON** (JavaScript Object Notation) je format za opisovanje podatkov, ki temelji na jeziku JavaScript.

**Ključ** je skrivnost, ki je potrebna za šifriranje in odšifriranje.

**Linux** je operacijski sistem, ki je zelo podoben sistemu Unix in nudi delo z ukazno vrstico.

**Microsoft Windows** je operacijski sistem.

**NetBeans** je integrirano razvojno okolje.

**OCI** (Oracle Call Interface) je knjižnica, ki se uporablja za poizvedovanje pri podatkovnih bazah Oracle.

**Odjemalec** je del programske opreme, ki dostopa do storitev, ki jih nudi strežnik.

**Odšifriranje** je postopek pretvarjanja tajnopisa v čistopis.

**OpenSSL** je knjižnica z implementiranim protokolom TLS.

**Oracle podatkovna baza** je objektno-relacijska podatkovna baza.

**Cordova** je ogrodje, ki se uporablja za razvoj hibridnih aplikacij.

**PHP** (PHP Hypertext Preprocessor) je skriptni jezik za razvoj spletnih strani.

**RC4** je šifrirni algoritem.

**REST** (Representational State Transfer) je množica arhitekturnih principov za razvoj spletnih storitev.

**RSA** je algoritem, ki se uporablja pri kriptografiji javnih ključev.

**SDK** (Software Development Kit) je paket za razvoj programske opreme.

**SQL** (Structured Query Language) je poizvedovalni jezik v relacijskih podatkovnih bazah.

**SSL** (Secure Socket Layer) je protokol, ki se uporablja za varno komunikacijo v internetu.

**Strežnik** je sistem, ki se odziva na zahtevke iz omrežja in s tem nudi določeno storitev.

**Šifriranje** je postopek pretvarjanja čistopisa v tajnopis.

**Tajnopis** je zašifriran, neberljiv tekst.

**TLS** (Transport Layer Security) je naslednik protokola SSL.

**URL** (Uniform Resource Locator) je enolični krajevnik vira, ki določa spletno stran v svetovnem spletu.

**UTF-8** (UCS Transformation Format-8bit) je eden izmed načinov kodiranja nabora znakov Unicode.

**Wireshark** je program, ki se uporablja za analizo prometa v omrežju.

**Xampp** je skupek orodij za razvoj spletnih vsebin.

**XML** (Extensible Markup Language) je razširljiv označevalni jezik, ki omogoča format za opisovanje strukturiranih podatkov.

**Vtičnik** je programska komponenta, ki doda določeno funkcijo obstoječi programski opremi.

# Povzetek

Kot je splošno znano, je v slovenskem zdravstvu še veliko prostora za napredek. Predvsem bi se dalo veliko narediti pri dolgih čakalnih vrstah, ki so posledica slabe komunikacije med posameznimi vejami medicinske stroke. Nemalokrat se zgodi, da mora pacient opravljati enak pregled večkrat, ker zdravniki nimajo podatkov, pridobljenih iz prejšnjih raziskav. Problem ni kompleksen in zanj obstaja več preprostih rešitev.

Diplomsko delo se osredotoča na eno izmed omenjenih rešitev in sicer na razvoj mobilne aplikacije za operacijski sistem Android, ki s kriptografijo omogoča varen prenos digitalne vsebine zdravniške kartoteke. Dotika se tudi razvoja spletne aplikacije, ki jo bodo uporabljali uporabniki omenjene mobilne aplikacije. Obe rešitvi uporabniku izgledata preprosti, njun razvoj pa je bil vse prej kot to.

Omeniti je potrebno, da opisana rešitev še ni dokončna in služi zgolj kot prototip za razvoj aplikacije, ki bi lahko relativno hitro prešla v širšo uporabo. Na to temo tudi že potekajo pogovori s predstavniki medicinske ter računalniške stroke.

**Ključne besede:** Android, kriptografija, Cordova, javni ključ, zasebni ključ.

# Abstract

It is widely known that there is a lot of room for improvement in the Slovenian health care system. A lot could be done about long queues, which are a consequence of poor communication between various branches of medical profession. It often occurs that patient must take the same test several times because doctors do not have the information obtained from the previous studies. For today's tools the problem is not complex and can be solved in a number of ways.

This thesis focuses on one of these solutions through the development of a mobile application for Android that enables a secure transfer of digital medical contents with cryptography. It also covers the development of a web application that will be used by users of the mobile application. The solution looks very simple to the user's eye, but as the reader will find, this is not the case.

It should be noted that the described solution is by far not final and merely serves as a prototipe for the development of a mobile application that may be relatively quickly passed into general use. This topic has already been discussed by the representatives of the medical and computer profession.

**Key words:** Android, cryptography, Cordova, public key, private key.

# Poglavje 1

# Uvod

Pametne naprave obstajajo že dobro desetletje, a takšnih, kot jih poznamo danes, ni bilo moč kupiti vse do leta 2007, ko je Apple svetu predstavil iPhone. Pred tem so bile tovrstne naprave sprejete kot nekaj, kar se uporablja le v poslovnem svetu, Apple pa je z iPhone-om ta koncept uspešno prenesel tudi na področje zabave. Že pred predstavitvijo iPhonea se je začel razvoj operacijskega sistema, ki bi ga bilo moč naložiti na veliko število naprav in ne le na lastne izdelke, kakor je to primer pri Applu. Android, odprto-kodni projekt, ki ga je sčasoma prevzel Google, je obljubljal zanesljiv in robusten operacijski sistem [1].

Glavni dejavnik uspeha pametnih naprav ni le njihova zmožnost hitrega in zanesljivega delovanja, pač pa tudi možnost dostopa do podaljškov zmogljivosti preko uporabniških aplikacij. Le-te nudijo veliko zabavnih in uporabnih vsebin, zato je skrajni čas, da se jih integrira v področja kot je zdravstvo. Do tega sklepa so prišli tudi nekateri zdravniki in podali pobudo za razvoj rešitve, opisane v diplomskem delu.

Mobilna aplikacija E-kartoteka je namenjena vsem vpletenim v procesu zdravljenja. Rešitev uporablja osnovne principe kriptografije za zagotavljanje varnega prenosa digitalne vsebine. Aplikacija je razvita s tehnologijo Cordova, ki omogoča razvoj mobilnih aplikacij s tehnologijami HTML, CSS ter Javascript, kar pomeni, da jih je v večini primerov moč naložiti na katerikoli

operacijski sistem. Pri aplikaciji E-kartoteka temu žal ni tako, saj uporablja še vtičnike, ki komunicirajo s platformo, značilno za določen operacijski sistem. Omenjeni vtičniki so razviti v programskem jeziku, ki je specifičen za posamezen operacijski sistem, zato bi jih bilo potrebno razviti še v jezikih, na katerih temeljijo drugi operacijski sistemi. To je glavni pogoj, da bi bila aplikacija prenosljiva tudi na ostale naprave.

Spletna aplikacija E-kartoteka, brez katere je mobilna aplikacija trenutno še neuporabna, je razvita s tehnologijami HTML, CSS, JavaScript in Php. Storitev nudi možnost varnega prenosa digitalne vsebine do podatkovne baze Oracle, od koder se nato ob uporabnikovem ukazu prenese na pametno napravo.

V tem besedilu bom predstavil svojo izkušnjo ob razvojo aplikacije E-kartoteka. V drugem poglavju lahko bralec prebere, katere rešitve se v zdravstvu že uporabljajo, v tretjem pa ima možnost pridobiti nekaj znanja o osnovnih kriptografskih principih in metodah. Omenjeno znanje je potrebno za dobro razumevanje poglavij, ki sledijo, v katerih so najprej predstavljene uporabljene tehnologije in orodja, za tem pa razvoj in analiza aplikacije E-kartoteka ter sklepne misli.

# Poglavje 2

# Dosedanje rešitve

V zdravstvu obstaja že veliko aplikacij s prav toliko nameni in načini uporabe, a med iskanjem nisem zasledil nobene, ki bi omogočala prenos vsebin na relaciji zdravnik - pacient ali zdravnik - zdravnik. Predvsem so to aplikacije, ki zdravnikom in medicinskemu osebju pomagajo pri delu v obliki svetovanja, izračunov ter podobnih storitev. Obstajajo tudi aplikacije, ki zdravnikom omogočajo pregled kartotek, a le od pacientov, ki so v oskrbi določene ustanove. V tem poglavju je opisanih nekaj primerov podobnih aplikacij. Potrebno je omeniti, da se v Sloveniji omenjene aplikacije še niso uveljavile in jih uporabljajo predvsem zdravstvene ustanove v državah, ki sodijo v sam vrh po kvaliteti zdravstva. To so predvsem države zahodne Evrope.

## 2.1 SAP EMR Unwired

Aplikacija SAP EMR Unwired stane 450 ameriških dolarjev in znatno olajša delo zdravnikom. Omogoča pregled kartotek njihovih pacientov, beleženje praktično vseh zapisov, ki jih najdemo v klasični kartoteki ter komuniciranje z drugimi zdravniki ter medicinskimi sestrami. Ima tudi odličen vmesnik za pregledovanje slik, ki omogoča pregledno primerjanje slik v določenem časovnem obdobju, kakor je prikazano na sliki 2.1. Omenjena storitev je zelo

uporabna predvsem pri spremljanju celjenja poškodb. Rešitev je razvita za vse razširjene platforme. Pomankljivost aplikacije je, da ne nudi možnosti posredovanja vsebine pacientom in zdravnikom drugih ustanov [2].


Slika 2.1: Primerjanje rentgenskih slik v aplikaciji EMR Unwired.

## 2.2 mMR (mobile Medical Records)

Odlično brezplačno aplikacijo so razvili pri podjetju DMS Health Technologies. Aplikacija nudi izvrsten vmesnik za pregled kartotek in komunikacijo s pacientom. Zdravnik ima tako možnost poklicati pacienta ali pa mu poslati sporočilo, preverjati poročila njihovih obiskov, kritične informacije, rezultate iz laboratorijev, diagnoze ter tudi njihove pritožbe. Kar se tiče pregledovanja kartotek, je aplikacija morda manj dovršena od EMR Unwired. Pomankljivost te aplikacije je, da ne nudi možnosti posredovanja vsebine drugim zdravnikom izven lastne ustanove [3].

## 2.3 Medscape

Ta aplikacija sicer ne omogoča pregleda zdravniških kartotek, a jo omenjamo, ker je najširše uporabljen zdravstveni priporočnik, saj ga uporablja preko dva miljona ljudi širom po svetu. Aplikacija je brezplačna in nudi pregled novic v svetu zdravstva, komunikacijo z drugimi uporabniki ter zelo široko podporo pri odločanju v obliki medicinskega kalkulatorja ter svetovanja pri diagnozah, posegih in predpisovanju zdravil [4].

## 2.4 Povzetek aplikacij

Mobilna aplikacija SAP EMR Unwired nudi pregled zdravniških kartotek, a je kvaliteti primerno tudi zelo draga. Brezplačna aplikacija MMR nudi uporabnikom podobne storitve kakor EMR Unwired, a je manj kvalitetna. Aplikacija Medscape je za razliko od mMR in EMR Unwired le priročnik, ki ga uporabljajo večinoma študentje, a je najbolj razširjena mobilna aplikacija za uporabo v zdravstvu. Vsem trem je skupno, da ne nudijo storitve prenosa datotek med uporabniki, kar pa nudi aplikacija, opisana v tem delu.

# Poglavje 3

# Osnovni kriptografski principi in metode

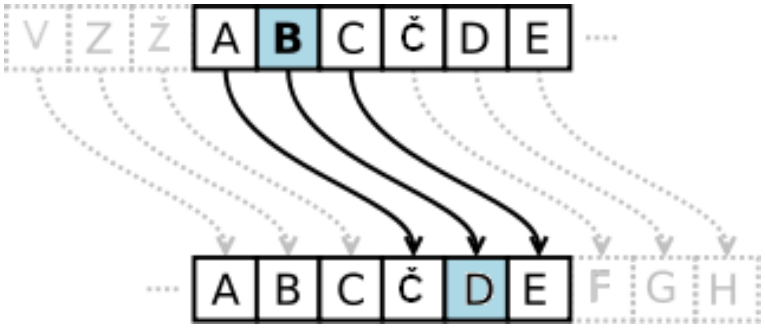
Za dobro razumevanje snovi, razložene v besedilu, je potrebno osnovno znanje kriptografije. V tem poglavju se lahko bralec poduči o nekaterih osnovnih principih kot so klasična kriptografija, moderna kriptografija, simetrična kriptografija, kriptografija javnih ključev, digitalni podpisi in potrdila ter algoritmi, ki se danes uporabljajo pri šifriranju.

## 3.1 Klasična kriptografija

Klasična kriptografija je veda o tehnikah pretvarjanja berljivega teksta (čistopisa) v neberljivi tekst (tajnopis) in obratno z namenom doseganja tajnosti oz. skrivanja vsebine pred nepooblaščenimi osebami. Z omenjenimi tehnikami se doseže, da je osebam, ki ne posedujejo šifrirnega ključa (razloženo v nadaljevanju), preprečen dostop do zašifriranih vsebin. Princip je možno zelo enostavno razložiti na Cezarjevi šifri, eni izmed najstareših in najpreprostejših šifrirnih tehnik.

Cezarjeva šifra je primer zamenjalne tehnike šifriranja, kjer se določen znak izvirnega teksta nadomesti z enim znakom šifrirnega teksta. Pri Cezarjevi šifri se tako vsaka črka izvirnega besedila nadomesti z drugo črko, ki je v

abecedi od nje oddaljena za fiksno število mest. To število pri Cezarjevi šifri predstavlja tudi t.i. ključ oz. skrivnost. Če vzamemo primer, da je Cezar hotel zašifrirati besedilo "ABC" in si je za ključ izbral število 3, potem je dobil zašifrirano besedilo "ČDE", saj je črka 'Č' od črke 'A' v slovenski abecedi oddaljena za tri mesta, enako pa velja tudi za preostale črke. Primer je grafično prikazan še na sliki 3.1. V primeru, da bi nepridiprav izvedel skrivnost, bi moral samo črke zamakniti za skrivnostno število v obratno smer in dobil bi izvirno besedilo. Tudi, če števila ne bi poznal, mu ne bi vzelo veliko časa, da bi preizkusil vseh 24 možnosti. To bi seveda lahko storil samo v primeru, da bi bil seznanjen z šifrirno tehniko, kar pa je bilo v tistih časih malo verjetno. Od takrat je kriptografija seveda zelo napredovala, zato danes poznamo veliko kompleksnejše tehnike šifriranja [5].




Slika 3.1: Cezarjeva šifra.

## 3.2 Moderna kriptografija

Moderna kriptografija se ukvarja predvsem z vprašanjem, kako varno prenesti digitalno vsebino iz ene lokacije na drugo ob prisotnosti nepooblaščenih oseb, ki sledijo komunikaciji preko javnega kanala. Veda vključuje znjanja iz področja matematike, računalništva, elektrotehnike, prava, financ in drugih..

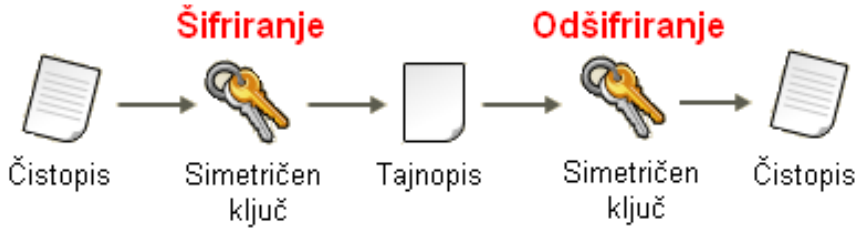
Danes ne gre več zato, da napadalec ne bi poznal algoritma šifriranja, pač pa velja predpostavka, da ga običajno pozna, manjka pa mu le ključ. Že sredi

devetnajstega stoletja je Auguste Kerckhoffs dejal, da mora biti kriptosistem varen tudi, če so o njem znane vse podrobnosti, razen ključa [6]. Iz tega razloga se danes v kriptografiji uporabljajo predvsem tehnike, pri katerih je ključe zelo težko ali skoraj nemogoče ugotoviti.

Tehnike moderne kriptografije se delijo na dve glavni veji in sicer na simetrično kriptografijo ter asimetrično kriptografijo oz. kriptografijo javnih ključev [5].

### 3.2.1 Simetrična kriptografija

Še 40 let nazaj smo poznali samo simetrično kriptografijo, pri kateri se uporablja isti ključ tako za šifriranje kot za odšifriranje, kakor je prikazano na sliki 3.2. Pri tem nastane težava upravljanja s ključi, saj zelo težko zagotovimo varno dostavo ključa le želenim prejemnikom. Simetrične šifre delimo na tokovne in bločne. Prve zašifrirajo vsak znak posebej, druge pa zašifrirajo več znakov skupaj.



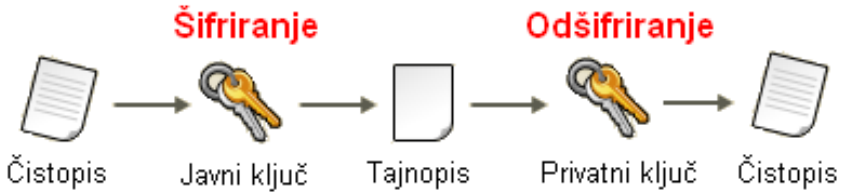

Slika 3.2: Šifriranje in odšifriranje s simetričnim ključem.

Verjetno najbolj znan algoritem, ki uporablja simetrično kriptografijo, je AES (napredni šifrirni standard, angl. *Advanced Encryption Standard*). Le-ta sporočilo zašifrira bločno s kombinacijo permutacij in substitucij, uporablja pa ključe dolžine 128, 196 ali 256 bitov. Šifra deluje tako, da celotno sporočilo zašifrira večkrat, število iteracij pa je odvisno od dolžine ključa. Za odšifriranje je potrebno iteracije izvesti v obratnem vrstnem redu [7].

### 3.2.2 Asimetrična kriptografija

Asimetrična kriptografija oz. kriptografija javnih ključev temelji na paru ključev: javni in zasebni. Praviloma ima vsak udeleženec v komunikaciji svoj par ključev. Če so podatki zašifrirani z enim ključem, jih je moč odšifrirati samo s pripadajočim komplementarnim ključem, kakor je prikazano na sliki 3.3. Tipičen primer algoritma iz te družine je RSA, ki je opisan v razdelku 3.2.5.

Če za primer vzamemo, da želi Jan poslati zasebno sporočilo Anji, bo to storil tako, da bo najprej pridobil njen javni ključ, nato pa z njim zašifriral sporočilo ter ji zašifrirano sporočilo poslal. Ko Anja prejme Janovo sporočilo, ga preprosto odšifrira z lastnim zasebnim ključem. Tu je treba omeniti, da čeprav lahko kdo prestreže Anjin javni ključ medtem, ko potuje do Jana, z njim ne more odšifrirati sporočila, lahko pa ga zamenja s svojim ter tako prestreza sporočila, namenjena Anji. Pojavi se problem identificiranja lastnika javnega ključa, ki ga rešimo z uporabo digitalnih potrdil, več o tem pa bomo izvedeli v razdelku 3.2.4.




Slika 3.3: Šifriranje in odšifriranje s parom ključev.

Sistem asimetrične kriptografije je zasnovan tako, da je izračun zasebnega ključa neizvedljiv iz informacije o javnem, pa čeprav sta povezana. Prednost asimetrične kriptografije pred simetrično je enostavnejše upravljanje s ključi, je pa proces šifriranja in odšifriranja veliko počasnejši, saj so simetrične šifre približno 1000-krat hitrejše od asimetričnih [8].

### 3.2.3 Digitalni podpis

Pri pošiljanju sporočila po internetu ni dovolj, da sporočilo zašifriramo. Pojavi se tudi problem identifikacije pošiljatelja. Zato se je razvilo podpisovanje sporočil, ki zagotavlja pristnost in integriteto poslanih podatkov. Princip je najlažje razložiti kar na primeru.

Če želi Jan poslati Anji podpisano zasebno sporočilo, ga najprej podpiše s svojim zasebnim ključem, nato si priskrbi Anjin javni ključ, z njim zašifrira že podpisano pismo in ji le-to pošlje. Anja prejeto sporočilo najprej odšifrira s svojim zasebnim ključem nato pa z Janovim javnim ključem preveri podpis in se tako prepriča, da ji ga je res poslal Jan.

### 3.2.4 Digitalna potrdila

Postopka, opisana v razdelkih 3.2.3 ter 3.2.2 imata skupno pomankljivost. Kako naj Jan ve, da je javni ključ, ki mu ga je poslala Anja, res njen? Napadalec bi lahko prestregel izmenjavo javnih ključev in se na vsaki strani predstavil bodisi kakor Anja ali Jan in tako prestrezal zaupna sporočila. Pojavi se problem identifikacije lastnika javnega ključa, ki se reši z uporabo digitalnih potrdil. To so elektronski dokumenti, ki poleg podatkov o ključu vsebujejo še podatke o lastniku, čas nastanka, rok uporabnosti in podobno.

Če so digitalna potrdila overjena s strani registriranih overiteljev, potem lahko prejemnik preveri identiteto pošiljatelja. Pri nas so v register vpisani štirje overitelji digitalnih potrdil, ki opravljajo storitev izdaje in upravljanja s kvalificiranimi digitalnimi potrdili za varno elektronsko podpisovanje [9]:

- Ministrstvo za javno upravo
- Nova Ljubljanska banka d.d.
- Pošta Slovenije d.o.o.
- Halcom d.d.

### 3.2.5 Hibridni kriptosistemi

Pri šifriranju velikih sporočil se danes uporabljajo t.i. hibridni kriptosistemi. Le-ti združujejo prednosti kriptografije javnih ključev z učinkovitostjo simetrične kriptografije tako, da sporočilo zašifrirajo s simetričnim ključem, le-tega pa nato zašifrirajo s javnim ključem. Naslovnik po prejetju podatkov simetrični ključ odšifrira s svojim zasebnim ključem in ga nato uporabi za odšifriranje sporočila [10]. Pri hibridnih kriptosistemih je zelo popularna kombinacija algoritmov *RSA* in *RC4*.

#### RSA

Algoritem, ki se uporablja pri kriptografiji javnih ključev, so leta 1977 zasnovali Ron Rivest, Adi Shamir ter Leonard Adleman, po njih pa nosi tudi ime.

Uporabnik algoritma izbere dve praštevili $p$ in $q$ in izračuna njun produkt $n$ ter vrednost $\phi = (p - 1)(q - 1)$. Nato izbere dve števili $e$ in $d$ tako, da velja $ed \equiv 1 \pmod{\phi}$. Par števil $(e, n)$ predstavlja njegov javni ključ, število $d$ pa zasebni ključ, ki se lahko včasih zapiše tudi s parom števil $(d, n)$.

Uporabnik objavi svoj javni ključ, zasebni pa mora ostati skrivnost. Samo nekdo, ki poseduje znanje o zasebnem ključu, lahko odšifrira sporočilo, zašifrirano s javnim ključem. Praviloma je to njun lastnik [11].

Obstaja vrsta možnih napadov na algoritem RSA. Primer napada je na primer t.i. napad z izbranim čistopisom, kjer napadalec v iteracijah šifrira besedilo, za katerega se mu zdi, da je podoben izvirnemu in testira, če je dobljen rezultat enak tajnopisu. Omenjenemu napadu se da izogniti tako, da se čistopisu doda določeno število varnostnih bitov. Le-ti omogočajo, da isti čistopis ob večkratnem šifriranju vsakič proizvede drugačen tajnopis. Algoritem je možno napasti še na veliko drugih načinov, vedno pa pomaga, če sta praštevili $p$ in $q$ zelo veliki, prav tako pa tudi njuna razlika.

Danes se smatra, da so ključi dovolj varni pri dolžini 1024 bitov, a se iz preventivnih razlogov uporabljajo tudi daljši ključi. Dolžina ključa je odvisna od tega, koliko dolgo mora ostati vsebina prikrita. Če mora vsebina ostati

zašifrirana le trenutek (kakor je to primer pri bančnih transakcijah), potem je dovolj, če so ključi krajši, saj napadalec nima veliko časa za izvedbo napada in ne more zlomiti niti kratkega ključa.

### RC4

RC4 je najpogosteje uporabljena tokovna šifra, ki se uporablja pri popularnih protokolih kot sta TLS in WEP. Šifro je zasnoval Ron Rivest, eden izmed izumiteljev algoritma RSA. Je zelo hitra in preprosta, a posebej ranljiva, ko je razkrit del njenega ključa [12].

# Poglavje 4

# Uporabljene tehnologije in orodja

Po nasvetu mentorja sem se odločil, da aplikacijo razvijem za operacijski sistem Android s orodjem Cordova. Le-ta nudi ogrodje ter skupek knjižnic za razvoj hibridnih mobilnih aplikacij. V tem poglavju so opisani pogoji, ki sem jih imel pri delu, uporabljeni jeziki in orodja, na koncu pa še nekateri pojmi ter tehnologije, ki sem jih uporabil pri rešitvi.

## 4.1 Delovni pogoji

Spletno aplikacijo ter aplikacijski strežnik sem razvijal na prenosnem računalniku HP Elitebook 8730w, ki poganja operacijski sistem Windows XP Professional. Rešitev sem razvijal v jeziku PHP z orodjem NetBeans. Uporabljal sem podatkovno bazo Oracle na oddaljeni lokaciji, aplikacijo pa sem poganjal na strežniku Apache s pomočjo distribucije Xampp.

Mobilno aplikacijo sem razvijal na istem računalniku kakor spletno (HP Elitebook 8730w, Windows XP Professional). Rešitev je sprogramirana s orodji Cordova, s katerimi je možno spletne aplikacije izdelati v jezikih HTML, CSS ter JavaScript in ne v jeziku, ki je značilen za posamezen operacijski sistem, kakor je to običaj. Poleg tega aplikacija vsebuje še vtičnike, ki

so sprogramirani v Javi in nudijo storitve, ki se jih v jeziku Javascript ni dalo realizirati, saj le-ta nima potrebnih knjižnic. Aplikacijo sem razvijal v razvojnem okolju Eclipse z vgrajenim vtičnikom ADT (*Android Developer Tools*, slovensko razvojna orodja za Android). Omenjeno orodje je del paketa Android SDK (skupek programske opreme za razvijanje, angl. *Software Development Kit*), ki poleg tega vključuje še skupek knjižnic, vzorce kode, navodila, Android platformo ter Android emulatorje.

Čeprav so emulatorji uporabna stvar, pa je testiranje aplikacij veliko hitrejše in zanesljivejše na dejanskih napravah. Sam sem aplikacijo testiral na tablici Samsung Galaxy Tab 3, kakršno vidimo na sliki 4.1, ter na pametnem telefonu Samsung 2 mini. Prvo napravo poganja različica operacijskega sistema Android 4.2.2 (Jelly Bean), drugo pa različica 2.3.6 (Gingerbread).


Slika 4.1: Tablica Samsung Tab 3.

## 4.2 Jeziki

Spletno aplikacijo ter aplikacijski strežnik sem razvijal v jezikih PHP in HTML, pri razvoju mobilne aplikacije pa sem uporabil jezike Java, HTML, CSS ter JavaScript.

### 4.2.1 PHP

PHP je kratica za *Personal Home Page Tools* (slovensko orodja za osebno spletno stran) in predstavlja skriptni jezik, ki se uporablja za razvoj dinamičnih spletnih vsebin, a se uporablja tudi kot jezik za splošno programiranje. PHP danes poganja več kot 244 miljonov spletnih strani in 2,1 miljona spletnih strežnikov.

Programsko kodo PHP interpretira spletni strežnik in nato prikaže želeno spletno stran. Ukazi PHP so lahko integrirani neposredno v dokument, ki vsebuje značke HTML, bolj pogosto pa so vsebovani v svoji datoteki, iz koder se nato kličejo. PHP se na splošno smatra za enega najlažjih programskih jezikov [13]. Uporaba jezika, ko je koda vključena v datoteko HTML, je prikazana na primeru 4.1.

Listing 4.1: Primer uporabe jezika PHP neposredno v datoteki HTML.

```
1 <html>
2   <head>
3     <title>PHP Test</title>
4   </head>
5   <body>
6     <?php echo '<p>Hello World</p>'; ?>
7   </body>
8 </html>
```

### 4.2.2 Java

Java je objektni programski jezik, ki je zasnovan tako, da je izvajanje kode neodvisno od platforme. To pomeni, da programa, ki se izvaja na eni platformi, ni potrebno še enkrat prevesti, da se lahko izvaja na drugi platformi. Zaradi tega je eden najpopularnejših programskih jezikov in se uporablja tudi pri programiranju Android aplikacij [14].

### 4.2.3 HTML, CSS, JavaScript

HTML, CSS ter Javascript so jeziki, s katerimi se izdela prikaz, oblikovanje ter dinamičnost spletnih vsebin.

## 4.3 Orodja

Danes si razvoj programskih rešitev skoraj nemogoče predstavljamo brez orodij. V tem podpoglavju so na grobo opisana orodja, ki sem jih uporabil pri razvoju aplikacije.

### 4.3.1 NetBeans

Eno izmed najširše uporabljenih programskih orodij je NetBeans, ki so ga leta 1996 razvili študentje Fakultete za matematiko in fiziko na Karlovi univerzi v Pragi.

To je integrirano razvojno okolje (angl. IDE - Integrated Development Environment) za razvoj predvsem programskih rešitev v Javi, a nudi dobro podboro tudi za druge jezike kot so PHP, C, C++ in HTML. NetBeans je primarno urejevalnik kode, a nudi programerju tudi pomoč v obliki dokumentacije in samodokončevanja kode, ima pa tudi vgrajen razhroščevalnik [15].

### 4.3.2 Xampp

Xampp je brezplačen odprto-kodni paket rešitev za spletni strežnik, ki je podprt na več platformah. Sestavljajo ga predvsem Apache HTTP strežnik, podatkovna baza MySQL in prevajalniki za skripte, napisane v programskih jezikih PHP in Perl. Je zelo primerno orodje za razvoj spletnih rešitev in vključuje številne uporabne module kot sta OpenSSL in phpMyAdmin [16].

### 4.3.3 ADT

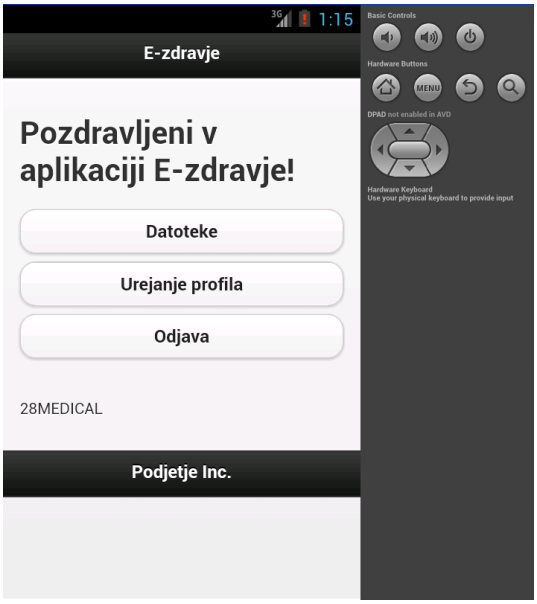
Android razvojna orodja (angl. *Android Development Tools*) so skupek programske opreme, ki novim razvijalcem močno olajša delo pri razvijanju lastnih aplikacij in med drugim vsebuje tudi razvojno orodje Eclipse z vtičnikom ADT ter Android SDK (orodja za razvoj programske opreme, angl. *Software Development Kit*) [17].

#### Eclipse in vtičnik ADT

Eclipse je integrirano razvojno okolje (angl. *IDE - Integrated Development Environment*) za razvoj programskih rešitev. Orodje razvijalcu nudi pomoč pri delu v obliki dokumentacije, samodokončevanja ukazov in razhroščevalnika, skupaj z vtičnikom ADT (angl. *Android Developer Tools*) pa omogoča tudi uporabo orodij za razvoj Android aplikacij kot so npr. emulatorji, knjižnice in podobno [18].

#### Android SDK

Android SDK je skupek orodij za razvoj aplikacij, ki med drugim vključuje razhroščevalnik, knjižnice, dokumentacijo, vzorce kode, navodila ter emulatorje, kakršen je prikazan na sliki 4.2 [19].




Slika 4.2: Primer emulatorja, ki ga nudi Android SDK.

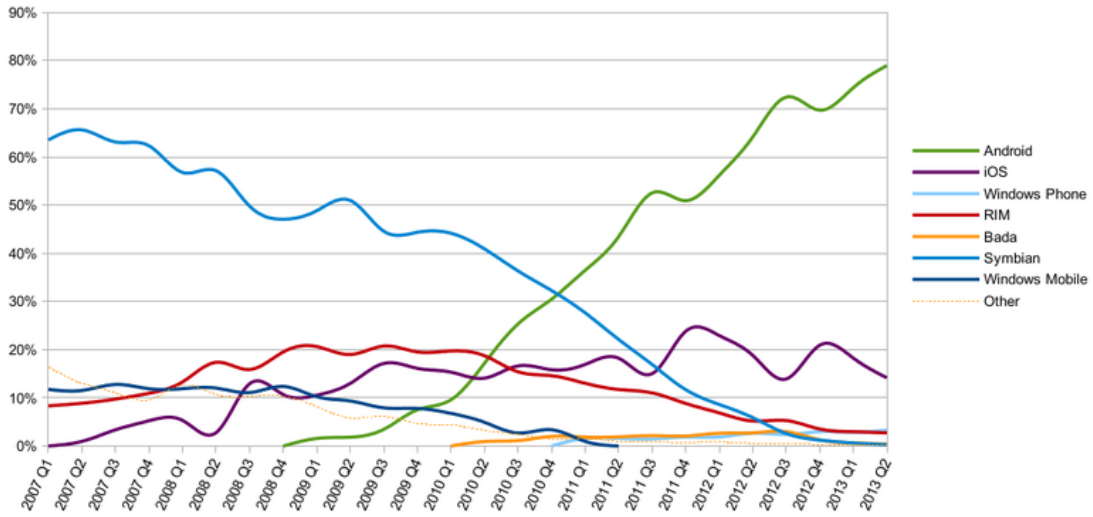
### 4.3.4 Wireshark

Wireshark je brezplačen odprto-kodni program, ki se uporablja za analizo prometa v omrežju.

## 4.4 Operacijski sistem Android

Android je trenutno najbolj popularen operacijski sistem za pametne naprave, predvsem za tiste, ki uporabljajo zaslon na dotik (primerjava z drugimi operacijskimi sistemi je prikazana na sliki 4.3). V to množico so všteti predvsem pametni telefoni ter tablični računalniki, kateri danes dosegajo zmogljivosti par let starih namiznih računalnikov. Do danes je bilo razvitih že 9 različic operacijskega sistema, vse pa bazirajo na operacijskem sistemu Linux. Najbolj priljubljena med njimi je verzija Jelly Bean iz leta 2012, ki je po zadnjih podatkih naložena na 36,6% vseh naprav, ki jih poganja Android.

Operacijski sistem je sprva razvijalo podjetje Android, Inc., a jih je po prvih znakih uspeha kupil Google. Android je bil javnosti prvič predstavljen




Slika 4.3: Deleži operacijskih sistemov na pametnih telefonih.

leta 2007, prvi pametni telefon, ki ga je podpiral, pa je bil prodan oktobra 2008 [20].

### 4.4.1 Arhitektura operacijskega sistema Android

Operacijski sistem Android je skupek programske opreme, ki je razdeljena na pet komponent preko štirih slojev, kakor predstavljeno na sliki 4.4 [21].

#### Jedro Linux

Na dnu slojev se nahaja jedro Linux, ki služi kot vmesnik med višjimi sloji ter strojno opremo. Je temelj celotne arhitekture in skrbi za osnovne funkcionalnosti, med katerimi so najpomembnejše upravljanje s procesi, upravljanje s pomnilnikom ter upravljanje z napravami kot so tipkovnica, kamera in zaslon.


Slika 4.4: Arhitektura operacijskega sistema Android.

### Knjižnice

Na naslednjem nivoju se nahaja skupek knjižnic, ki skrbijo za obdelavo različnih tipov podatkov. Tako na primer knjižnica SQLite skrbi za shranjevanje podatkov v bazo, SSL nudi internetno varnost, WebKit pa se uporablja za prikazovanje spletne vsebine. Knjižnice so napisane v programskih jezikih C in C++.

### Razvojno okolje

Komponenta razvojno okolje vsebuje navidezni stroj Dalvik, ki z upravljanjem virov skrbi za optimalno delovanje aplikacij. V tem sloju se nahajajo tudi nekatere knjižnice, napisane v programskem jeziku Java.

#### Aplikacijsko ogrodje

Četrta komponenta aplikacijam nudi višje-nivojske storitve v obliki razredov Java. Razvijalci omenjene storitve uporabljajo pri razvoju aplikacij. Primer take storitve je upravitelj virov (angl. *Resource Manager*), ki upravlja z različnimi tipi podatkov, ki se uporabljajo v aplikacijah.

#### Aplikacije

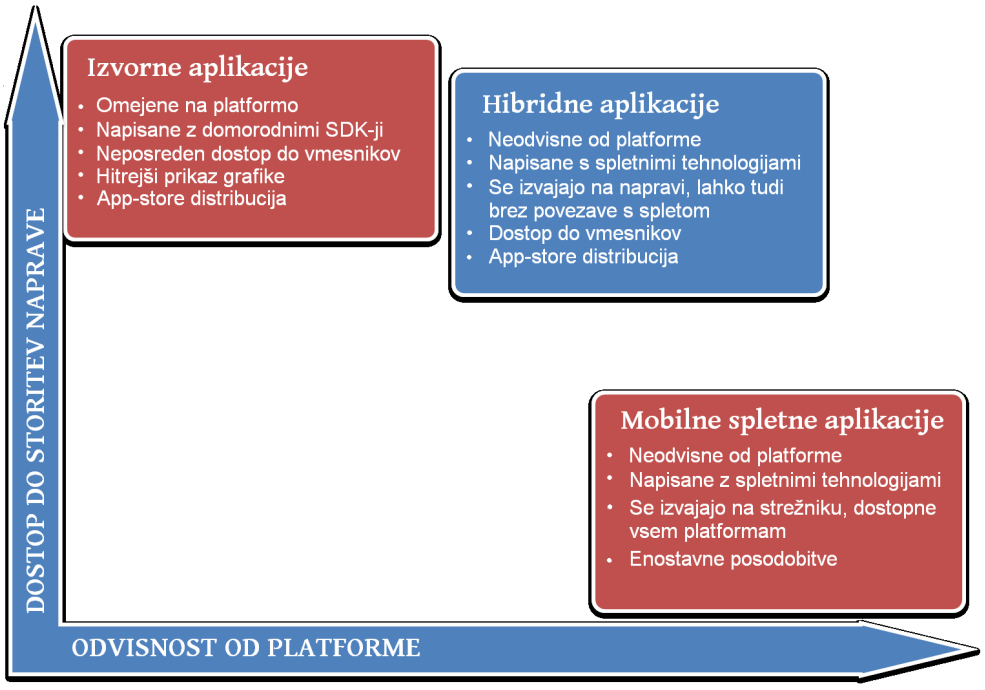
Aplikacije predstavljajo najvišje ležečo komponento v arhitekturi. Sem spadajo vse aplikacije, naložene na napravi. Primeri osnovnih aplikacij so imenik, spletni brskalnik.

## 4.5 Cordova

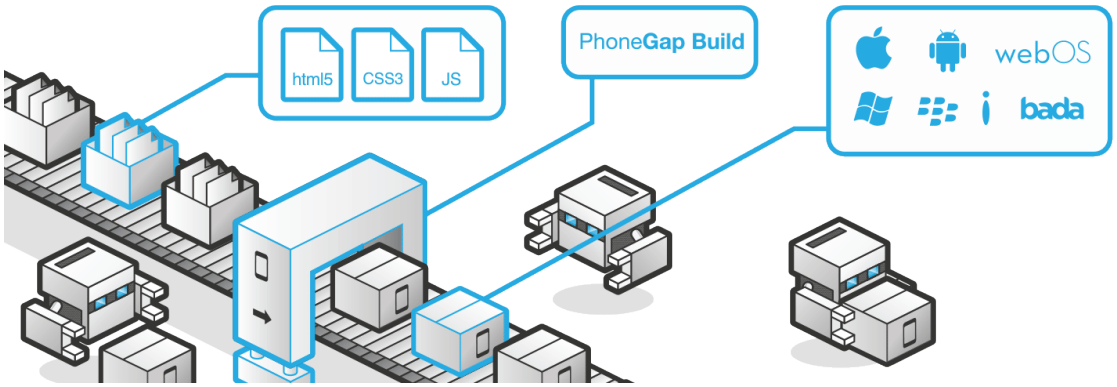
Cordova je brezplačno ogrodje, ki omogoča razvijanje mobilnih aplikacij s tehnologijami HTML5, CSS ter Javascript namesto z jeziki, ki so značilni za določene platforme. Aplikacije, ki temeljijo na tej tehnologiji, imenujemo hibridne aplikacije in jih je moč naložiti na več operacijskih sistemov. Razlog za to je dejstvo, da aplikacije niso izvorne (izdelane za točno določen operacijski sistem), pač pa so bolj podobne spletnim aplikacijam, s to razliko, da omogočajo dostop do izvornih vtičnikov. Primerjava hibridnih aplikacij z ostalimi je grafično prikazana na sliki 4.5. Ob prevajanju rešitve se datoteke HTML5, CSS ter Javascript zapakirajo v ovoj, ki omogoča aplikaciji komuniciranje s platformo. Omenjen proces je prikazan na sliki 4.6.

Ogrodje Cordova sestoji iz spletnega pogleda (ki je enak pogledom spletnih brskalnikov, le brez robov), množice funkcij Javascript, ki so vezane na funkcije izvornega okolja ter vmesnika, ki komunicira z izvornim okoljem [22]. Aplikacije v tehnologiji Cordvoa podpirajo sledeče platforme [23]:

- Apple iOS
- Android
- Blackberry OS
- WebOS
- Windows 7, 8
- Symbian
- Bada




Slika 4.5: Primerjava različnih vrst mobilnih aplikacij [10].




Slika 4.6: Proces prevajanja aplikacije v tehnologiji Cordova.

### Vtičniki

Vtičniki Cordova zagotavljajo aplikaciji komunikacijo z izvornim okoljem platforme, na kateri se aplikacija izvaja. Sestavljeni so iz vmesnika Javascript, ki je enak za vse platforme in iz izvornih implementacij, ki so značilne za določeno platformo.

Ko aplikacija potrebuje storitev, ki jo nudi vtičnik, to sporoči z vmesnikom Javascript. Ta domorodnemu okolju sporoči zahtevo, slednji pa po procesiranju zahtevka aplikaciji vrne odgovor. Le-tega nato sprocesira isti vmesnik, ki je zahtevo poslal in glede na vsebino sproži določen ukaz. Celoten proces bo bolj podrobno opisan v poglavju 5.2.3 v razdelku "Registracija" [24].

### JQuery Mobile

JQuery Mobile je ogrodje za uporabniški vmesnik (angl. *UI Framework*), optimizirano za platforme, ki uporabljajo zaslone na dotik. Ogrodje je zgrajeno na obstoječem jedru jQuery, kar omogoča enostavno uporabo izkušenim razvijalcem spletnih vsebin [25].

## 4.6 Podatkovna baza Oracle

Oracle podatkovna baza je sistem za upravljanje z objektno-relacijskimi bazami podatkov (angl. RDBMS - Relational Database Management System), ki ga je ustvarilo podjetje Oracle.

Pri relacijskih podatkovnih bazah so podatki predstavljeni v obliki tabel, pri objektnih podatkovnih bazah pa so podatki predstavljeni v obliki takšnih objektov, kot so uporabljeni pri objektnem programiranju. Objektno-relacijske podatkovne baze so podobne relacijskim, a uporabljajo objektno usmerjen podatkovni model: objekti, razredi in dedovanja so neposredno podprti v poizvedovalnem jeziku in podatkovnih shemah. Poleg omenjenega podpirajo tudi klasične relacijske poizvedbe [26, 27].

Za urejanje baze sem uporabljal program Toad (*Tool for Oracle Application Developers*).

## 4.7 Spletna storitev

Spletna storitev je način komunikacije med dvema elektronskima napravama preko svetovnega spleta. Je programska funkcija, ki je na voljo na določenem spletnem naslovu. Spletne storitve so načeloma vedno dostopne [28].

Spletna storitev, ki jo uporablja v tem besedilu opisana aplikacija, je razvita v stilu arhitekture REST, kjer odjemalec uporabi eno izmed metod HTTP (GET, POST, PUT, DELETE), da pošlje zahtevek na strežnik. Slednji po procesiranju zahtevka, vrne odgovor v formatu JSON, ki je opisan v naslednjem razdelku [29].

## 4.8 JSON

JSON (*JavaScript Object Notation*) je standard za prenos podatkov, ki bazira na tekstu in je posledično lahko berljiv. Izpeljan je iz jezika Javascript, a je od jezika neodvisen in se ga lahko uporablja v številnih drugih jezikih. JSON format je v prvi vrsti namenjen prenosu podatkov med strežnikom ter aplikacijo in predstavlja dobro alternativo formatu XML [30].

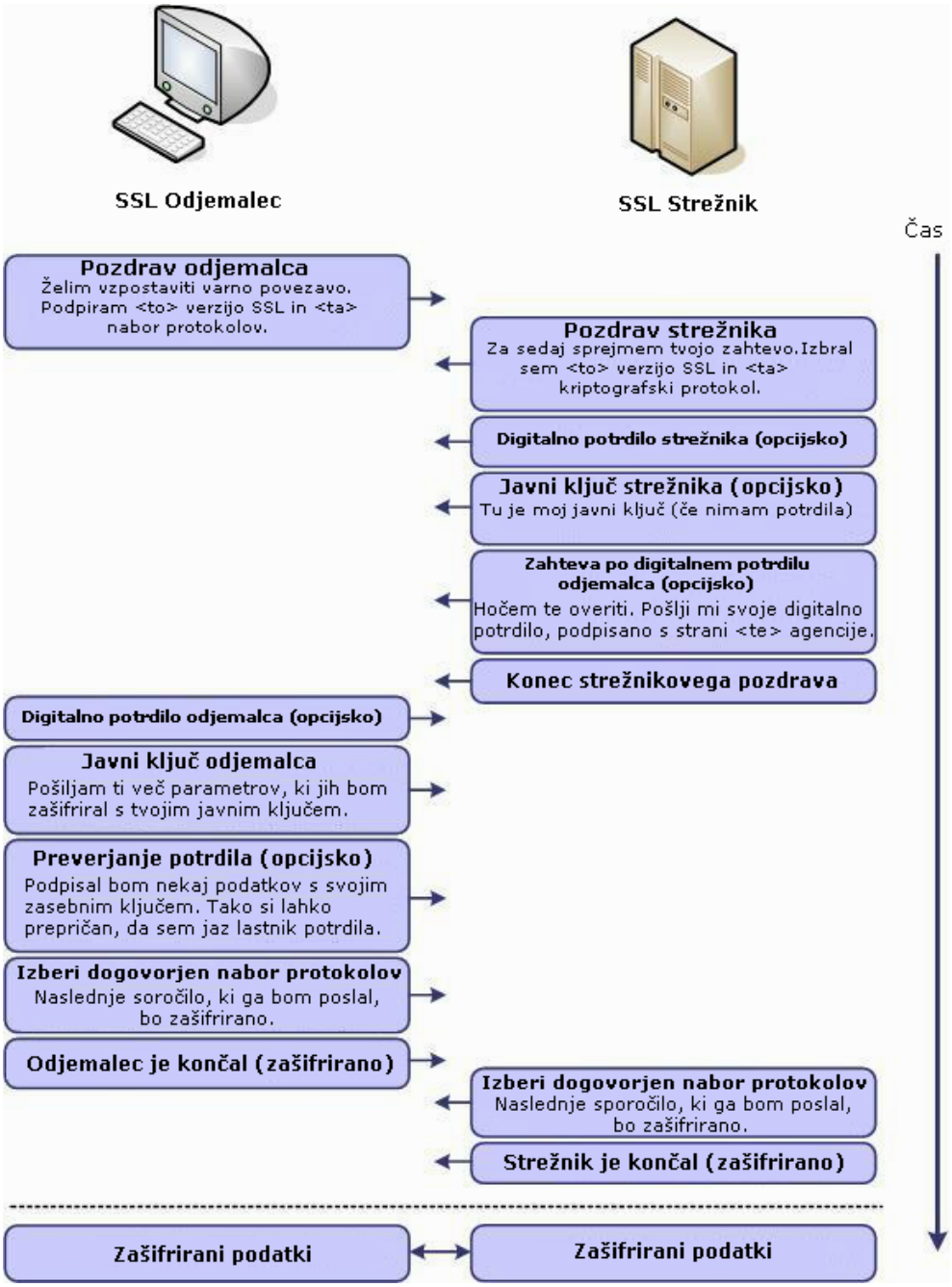
## 4.9 SSL/TLS

HTTP (protokol za prenos hiperteksta, angl. *Hypertext Transfer Protocol*) je protokol za prenos spletnih vsebin v obliki hipertekstain deluje po principu zahtevek - odgovor. Odjemalec (npr. spletni brskalnik) pošlje zahtevek po določeni vsebini na strežnik, le-ta pa odjemalcu, po procesiranju zahtevka, vrne odgovor. Ta vsebuje status o uspehu, lahko pa tudi določeno vsebino, ponavadi v obliki datoteke HTML. Komunikacija, ki poteka po protokolu HTTP ni zašifrirana in ji lahko prisluškuje vsakdo, ki ima osnovno znanje za rokovanje s programom za vohunjenje, kakršen je Wireshark [31].

Varnost dosežemo s protokolom SSL (sloj varnih vtičnic, angl. *Secure Socket Layer*) oz. z njegovim naslednikom TLS (zaščita transportnega sloja, angl. *Transport Layer Security*), ki skrbi za zaščiteno povezavo med brskalnikom in spletnim strežnikom tako, da jo zašifrira in se ji posledično ne da prisluškovati. Poleg tega omogoči uporabnikom, da preverijo, ali so se priključili na pravi strežnik, saj iz njegovega digitalnega potrdila izvedo potrebne podatke.

Protokol omogoča tudi overjanje uporabnikov preko digitalnih potrdil, ki jih vključimo v brskalnike. Tako ob priključitvi na nek strežnik ta zahteva, da se brskalnik predstavi s svojim potrdilom. Glede na vsebino potrdila potem strežnik dovoli priključitev ali pa prekine povezavo. Zaradi omenjenega je protokol zelo popularen pri bančnih spletnih aplikacijah [32].

Delovanje protokola je predstavljeno na sliki 4.7.




Slika 4.7: Potek vzpostavljanja komunikacije po protokolu SSL/TLS [33].

# Poglavje 5

# Aplikacija E-kartoteka

Mobilna aplikacija E-kartoteka je zaenkrat razvita le za operacijski sistem Android, implementacije za druge operacijske sisteme pa še sledijo. Aplikacija uporabniku omogoča varen prenos svoje digitalne zdravstvene kartoteke ter ogled njene vsebine. Je zelo preprosta, saj poleg vpisa ter registracije omogoča le ogled kartoteke, a bo v prihodnosti razširjena z dodatnimi vsebinami kot so npr. posredovanje datotek, komunikacija z zdravnikom, zapiski in podobno.

Spletna aplikacija E-kartoteka se uporablja za varen prenos vsebine zdravstvene kartoteke v podatkovno bazo, poleg tega pa omogoča le še vpis uporabnika. Registracija je zaenkrat mogoča le v mobilni aplikaciji, saj se ob tem v datotečnem sistemu naprave shrani še par ključev, ki je nujno potreben za storitev spletne aplikacije. Registracija pri spletni aplikaciji še ni mogoča, saj bi uporabnik ne sme biti zmožen uporabljati aplikacije, če še nima para ključev na mobilni napravi (razen, če gre za zdravniško osebje).

## 5.1 Načrtovanje

Razvoj vsake aplikacije se praviloma začne z načrtovanjem. V tem podpoglavju je opisana arhitektura, v slogu katere se prenašajo podatki med strežnikom in aplikacijo, podatkovni model, arhitektura pogledov, način upo-

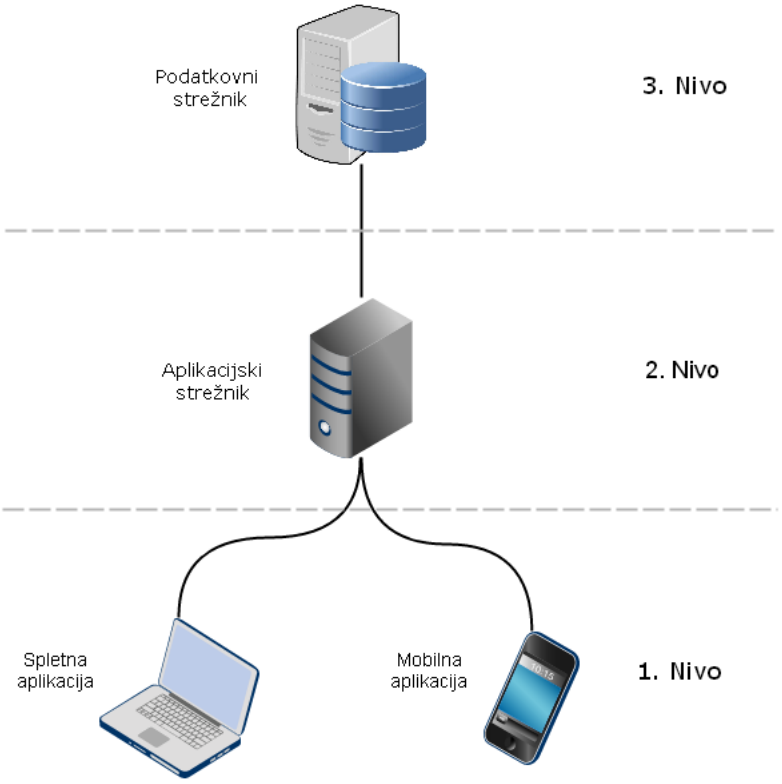
rabe kriptografije ter zaporedje izmenjav podatkov med odjemalci ter aplikacijskim in podatkovnim strežnikom.

### 5.1.1 Trinivojska arhitektura

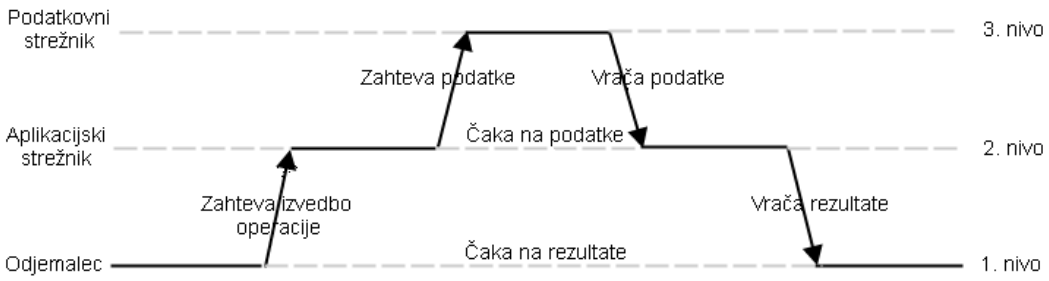
Z uporabo večnivojske arhitekture se aplikacija razdeli na več nivojev, ki so med seboj logično povezani. Posledica tega je, da je možno spreminjati samo določen nivo in ne celotne aplikacije. Zaradi omenjenega sem se pri načrtovanju mobilne in spletne aplikacije odločil za uporabo trinivojske arhitekture, kakršna je prikazana na sliki 5.1. To pomeni, da imamo poleg odjemalca in podatkovnega strežnika (lokacija, kjer so shranjeni podatki) še vmesni aplikacijski strežnik. Naloge vsakega nivoja so sledeče:

- **Odjemalec** (1. nivo): prikazuje uporabniški vmesnik in pošilja zahtevke na 2. nivo.
- **Aplikacijski strežnik** (2. nivo): procesira zahtevke, ki jih je poslal odjemalec ter glede na njihovo vsebino izvede določeno poslovno opravilo, ki ponavadi vključuje komunikacijo z tretjim nivojem. Po končanem procesiranju zahtevkov vrne odgovor na prvi nivo.
- **Podatkovni strežnik** (3. nivo): izvaja naloge, katere od njega zahteva drugi nivo in katere ponavadi vključujejo vračanje, vstavljanje, spreminjanje ali brisanje podatkov. Poleg tega tudi skrbi za integriteto in varnost podatkov.

Po vzoru opisane arhitektura mobilna in spletna aplikacija komunicirata z aplikacijskim strežnikom, ta pa jima po komunikaciji s podatkovnim strežnikom vrne odgovor v formatu JSON. Glede na prejet odgovor s strani aplikacijskega strežnika prikaže aplikacija določeno vsebino. Potek komunikacije je prikazan na sliki 5.2 [34].




Slika 5.1: Trinivojska arhitektura.



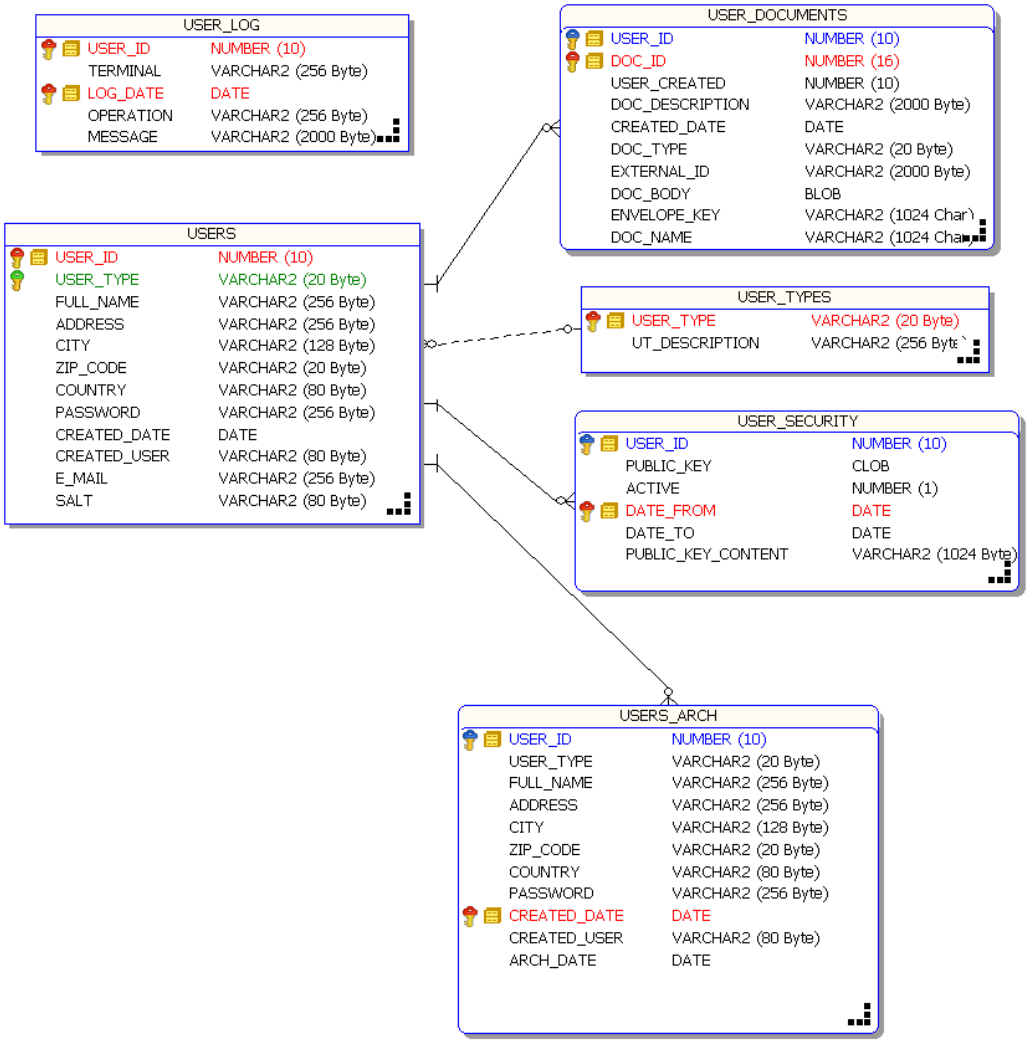

Slika 5.2: Potek komunikacije v trinivojski arhitekturi.

### 5.1.2 Podatkovni model

Podatkovni model sestavlja šest entitet, ki so grafično predstavljene na sliki 5.3, podrobneje pa v spodnjem seznamu.

- **Users**: entiteta hrani podatke o uporabnikih mobilne aplikacije in hrani povezave z vsemi preostalimi entitetami z izjemo tabele `user_log`.
- **User_types**: entiteta ima samo dve polji. `User_type` hrani vse možne tipe uporabnikov, `ut_description` pa njihov opis. Trenutno so tipi uporabnikov le trije:
  - Medical: tip opisuje zdravniško osebje. Uporabniki tega tipa lahko datoteke pošiljajo tako sebi kot drugim zdravnikom in pacientom.
  - Patient: tip opisuje paciente. Le-ti lahko datoteke pošiljajo le sebi in svojemu zdravniku.
  - Administrator: gre za administratorja aplikacije, ki skrbi za podatkovno bazo in reševanje napak v sistemu.

  Tabela hrani povezavo z entiteto `users`.
- **User_security**: entiteta trenutno hrani samo javni ključ uporabnika, a bo v prihodnosti hranila tudi status o veljavnosti ključa ter njegov rok uporabe. Entiteta hrani povezavo s tabelo `users`.
- **User_documents**: tabela je namenjena hranjenju zašifrirane vsebine uporabnikovih zdravniških dokumentov. Vsebuje tudi druga potrebna polja, ki hranijo opise dokumenta ter z javnim ključem uporabnika zašifriran simetrični ključ, ki je potreben za odšifriranje dokumenta. Entiteta hrani povezavo s tabelo `users`.
- **User_arch**: tabela hrani zgodovino sprememb uporabnikovih podatkov in hrani povezavo s tabelo `users`.
- **User_log**: tabela je namenjena nadzoru, tj. beleženju aktivnosti uporabnika, a je trenutno še nekoriščena.




Slika 5.3: Arhitektura podatkovnega modela.

### 5.1.3 Arhitektura pogledov v mobilni aplikaciji

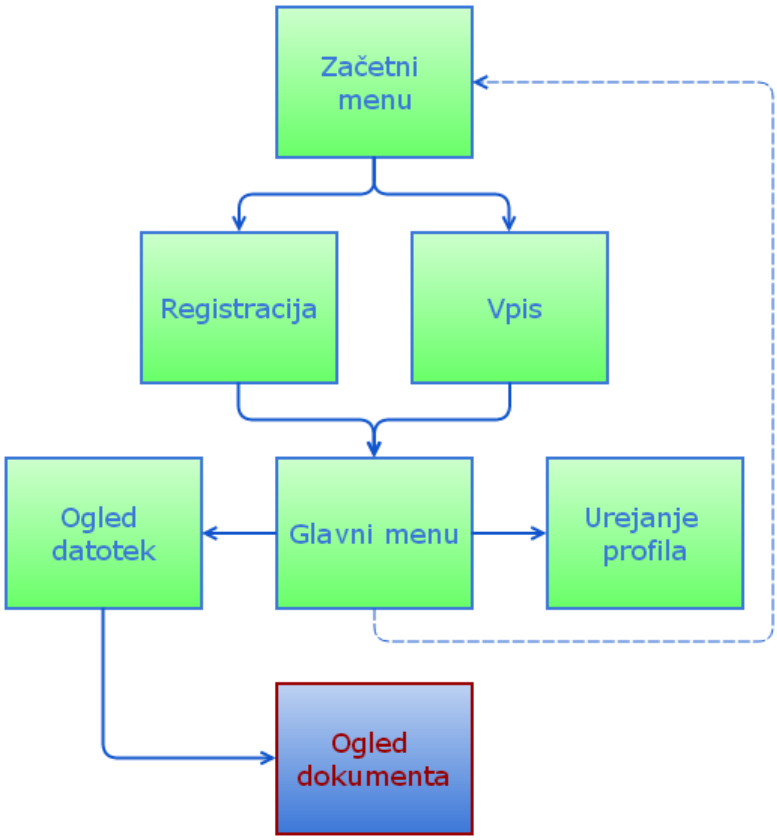
Ob zagonu aplikacije se uporabniku prikaže meni, v katerem je mogoče izbirati med vpisom in registracijo. Ob uspešnem vpisu ali registraciji se uporabniku prikaže glavni meni, kjer pa lahko uporabnik izbere eno izmed naslednjih treh opcij:

- Pregled datotek: če uporabnik pritisne na gumb za ogled datotek, se mu prikaže seznam datotek, ki jih ima shranjene v podatkovni bazi.
- Urejanje profila: če uporabnik pritisne na gumb za urejanje profila, lahko posodobi podatke, ki jih je bil prisiljen vpisati ob registraciji.
- Odjava: ob pritisku na gumb za odjavo se uporabnik vrne na začetni meni, kjer lahko izbira med registracijo in vpisom.

Če uporabnik izbere zadnjo izmed naštetih opcij, lahko pritisne na eno izmed datotek na seznamu. V tem primeru se uporabniku prikaže želena datoteka v privzetem ogledovalnem programu, ki ni del aplikacije. Potrebno je še omeniti, da se ob pritisku na tipko *nazaj* pogled vedno vrne na prejšnjega. Izjema pri tem pravilu je dogodek ob pritisku omenjene tipke v primeru, da se uporabnik nahaja v pogledu *glavni meni*. Takrat namreč uporabnik zapre aplikacijo. Organizacija je grafično prikazana na sliki 5.4.

### 5.1.4 Varnost

Komunikacija med odjemalcem ter aplikacijskim strežnikom je zašifrirana z uporabo SSL/TLS protokola, slednji pa za varno komunikacijo s podatkovnim strežnikom uporablja vmesnik OCI (*Oracle Call Interface*, slovensko ukazni vmesnik Oracle). Potrebno je omeniti, da uporaba SSL/TLS protokola pri mobilnih aplikacijah predstavlja problem, saj lahko pride do napada srednjega napadalca. Temu se izognemu s previdno implementacijo in z zaupanjem le izbranim digitalnim potrdilom (Android namreč privzeto zaupa veliko ali celo vsem potrdilom).




Slika 5.4: Arhitektura pogledov.

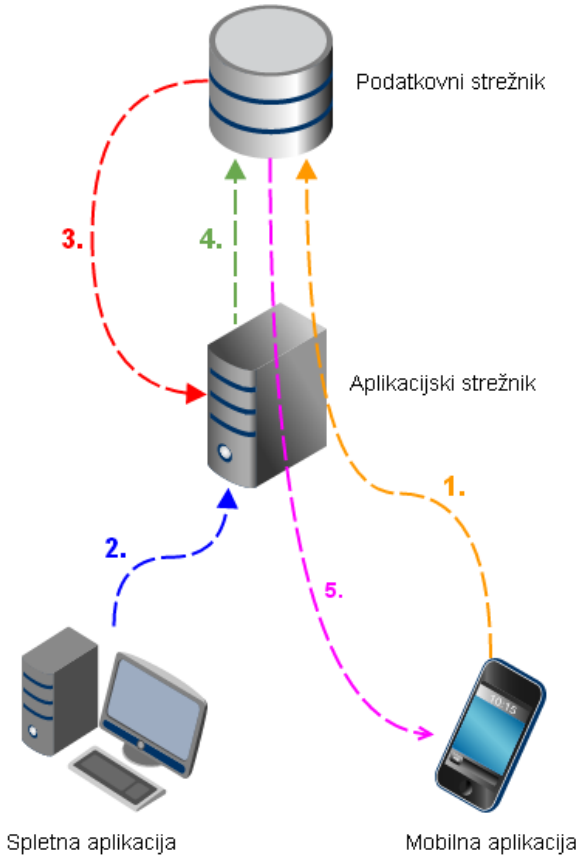
Šifriranje in odšifriranje dokumentov poteka s kombinacijo simetrične ter asimetrične kriptografije, vse pa bo bolj podrobno razloženo v poglavjih 5.2.2 in 5.2.4.

### 5.1.5 Zaporedje izmenjav podatkov

V tem razdelku je opisano zaporedje prenosov podatkov, ki je potrebno za ogled vsebine določene datoteke ter operacije, ki se med tem izvajajo. Za boljšo predstavo si lahko bralec pomaga s primerom na sliki 5.5.

- **1:** ob uporabnikovi registraciji, ki je mogoča le pri mobilni aplikaciji, se na pametni napravi ustvari par ključev: javni in zasebni. Javni ključ se preko aplikacijskega strežnika shrani v podatkovno bazo.
- **2, 3, 4:** ko želi uporabnik zašifrirati določeno datoteko, jo po varnem kanalu pošlje do aplikacijskega strežnika (2). Le-ta nato naredi naslednje:
  - Izvede poizvedbo po javnem ključu uporabnika (3).
  - Ustvari simetričen ključ ter z njim zašifrira vsebino datoteke.
  - Simetričen ključ zašifrira s pridobljenim javnim ključem.
  - Zašifriran simetričen ključ ter zašifrirano vsebino datoteke shrani v podatkovno bazo (4).
- **5:** ko si želi uporabnik ogledati datoteko, to sporoči aplikacijskemu strežniku. Le-ta nato naredi poizvedbo po zašifrirani vsebini datoteke in zašifriranem simetričnem ključu, s katerim je bila zašifrirana datoteka ter pridobljene podatke pošlje pametni napravi. Le-ta z zasebnim ključem odšifrira simetričen ključ, katerega nato uporabi za odšifriranje vsebine datoteke, ki jo na koncu prikaže uporabniku. Potrebno je omeniti, da se datoteka v berljivi obliki izbriše še preden uporabnik zaključi z ogledovanjem. Če si je uporabnik datoteko že ogledoval, je njena zašifrirana vsebina shranjena na napravi. V tem primeru aplikacijski strežnik naredi poizvedbo le po zašifriranem simetričnem ključu.

Morebitne nejasnosti bodo razložene v poglavju 5.2, kjer so vse omenjene operacije podrobneje opisane.




Slika 5.5: Prenos podatkov po arhitekturi.

## 5.2 Delovanje in uporabniška izkušnja

V tem razdelku je opisana uporabniška izkušnja aplikacije, ob vsaki operaciji, ki jo ima uporabnik moč sprožiti, pa je površinsko opisano tudi kaj se dogaja v ozadju.

### 5.2.1 Aplikacijski strežnik

Aplikacijski strežnik je razdeljen na del, ki služi spletni aplikaciji in del, ki podpira mobilno aplikacijo. Oba sta napisana v jeziku PHP, a se rahlo razlikujeta. Medtem, ko ima vsaka operacija, ki jo izvaja spletna aplikacija, na strežniku lasten dokument PHP, pa vse operacije mobilne aplikacije ko-

municirajo z strežnikom preko skupneg dokumenta *index.php*, od koder se nato kličejo funkcije, definirane v dokumentu *DB_Functions.php*. Operacije se nato ločijo glede na vsebino parametra *tag*. Organizacija je na primeru opisana v izseku 5.1.

Listing 5.1: Strežniški del mobilne aplikacije

```
<?php
require_once 'include/DB_Functions.php';
// Ustvari instanco razreda DB_Functions,
// definiranega v datoteki DB_Functions.php.
$db = new DB_Functions();

// Ustvari tabelo, ki se bo v obliki JSON
// poslala odjemalcu kot odgovor na zahtevo.
$response = array("tag" => $tag, "success" => 0, "error" => 0);

if (isset($_POST['tag']) && $_POST['tag'] != '') {
  // Pridobi vsebino parametra tag
  $tag = $_POST['tag'];

  // Ce je odjemalec poslal zahtevo po overitvi uporabnika...
  if ($tag == 'login') {
    // Uporabi funkcijo razreda DB_Functions
    // za overitev uporabnika.
    $user = $db->getUserByEmailAndPassword($email, $password);
    if ($user != false) {
      $response["success"] = 1;
    } else {
      $response["error"] = 1;
    }
    // Vrni odgovor odjemalcu v obliki JSON.
    echo json_encode($response);
  } else if ($tag == 'decrypt'){
    ...
  } ...
}
?>
```

### 5.2.2 Spletna aplikaija

Spletna aplikacija nudi uporabniku le dva pogleda in sicer pogled za prijavo ter pogled, ki vsebuje formo za vnos podatkov o datoteki, ki jo želi uporabnik shraniti.

#### Prijava

Ob vstopu v aplikacijo se uporabniku prikaže le pogled za vpis v aplikacijo. Registracija v spletni aplikaciji zaenkrat še ni mogoča, razlog za to pa je opisan v uvodu poglavja 5. Ko uporabnik potrdi svoj vpis prijavnih podatkov, se le-ti po zašifrirani povezavi pošljejo do aplikacijskega strežnika, ta pa z dostopom do podatkovne baze preveri njegovo identiteto. To stori tako, da iz baze pridobi zgoščeno geslo ter t.i. sol, nato pa z omenjeno soljo zgosti uporabnikovo geslo z uporabo eno-smerne zgoščevalne funkcije ter ga primerja s tistim iz baze. Ob vpisu napačnih podatkov se uporabniku prikaže obvestilo o napačnem vnosu, v nasprotnem primeru pa brskalnik prikaže formo za shranjevanje datotek.

#### Forma za shranjevanje datotek

V tem pogledu ima uporabnik možnost izbrati datoteko ter vnesti njen opis, s pritiskom na gumb *Zašifrirajj* pa se datoteka zašifrira in shrani v bazo. Šifriranje opravi ukaz *openssl_seal*, ki ga aplikacija pridobi iz knjižnice OpenSSL. Ukaz sprejme štiri parametre in sicer:

- `$data`: vsebina, ki se bo zašifrirala.
- `$encrypted`: v to spremenljivko se bo shranila zašifrirana vsebina.
- `$a_envelope`: tabela, v katero se bo shranil simetričen ključ, s katerim bo zašifrirana vsebina.
- `$a_key`: tabela, ki hrani uporabnikov javni ključ.

Ukaz datoteko zašifrira tako, da ustvari 128 bitov dolg simetričen ključ, z njim zašifrira vsebino, nato pa z javnim ključem uporabnika zašifrira simetričen ključ. Ukaz je na primeru opisan še v izseku 5.2.

Listing 5.2: Šifriranje datoteke

```
1
2 // Pridobi vsebino datoteke.
3 $data = file_get_contents($_FILES["file"]["tmp_name"]);
4 // Ustvari tabelo, kamor se bo shranil simetricen kljuc.
5 $a_envelope = array();
6 // Pridobi javni kljuc,
7 $publicKey = openssl_pkey_get_public($publicKeyContents);
8 // in ga shrani v tabelo.
9 $a_key = array($publicKey);
10 if (openssl_seal($data, $encrypted, $a_envelope, $a_key) ===
     FALSE)
11   // V primeru neuspeha prekini izvajanje skripte.
12   die('Failed to encrypt data');
```

### 5.2.3 Mobilna aplikacija

Tudi tukaj bom sledil vzoru iz prejšnjega razdelka ter delovanje aplikacije razložil preko uporabniške izkušnje. Še enkrat moram omeniti, da aplikacija nudi le storitev ogleda datotek in služi bolj kot osnutek za nadaljni razvoj.

#### Začetni meni

Ko uporabnik požene aplikacijo, se v pomnilnik naloži dokument HTML, ki vsebuje celotno aplikacijo. To je tudi princip, ki se uporablja pri razvijanju aplikacij z orodjem Cordova. Za boljšo predstavo je organizacija predstavljena na primeru 5.3.

Listing 5.3: Struktura aplikacije

```
<!DOCTYPE HTML>
<html>
  <head>...</head>
  <body>
    <div id="container">
      <div data-role="page" id="homepage">
        ...
      </div>
      <div data-role="page" id="registerPage">
        ...
      </div>
    </div>
  </body>
</html>
```

V glavnem meniju lahko uporabnik izbere med registracijo ter vpisom. Spremembo pogleda opravi ukaz `$.mobile.changePage`, ki je definiran v knjižnici jQuery Mobile.

### Registracija

Uporabniku se prikaže obrazec za vnos podatkov, kakor je prikazano na sliki 5.6. Uporabniku je za uspešno registracijo potrebno vnesti vse podatke. Vnosi se preverjajo tako na strani odjemalca kot na strežniku, preverja pa se tudi, če uporabnik z vpisanim e-naslovom morda že obstaja.

Ob uspešni registraciji uporabnika se prvič v naši aplikaciji uporabi storitev, ki jo nudi vtičnik. Takrat se namreč ustvari par ključev, ki se nato shrani na datotečni sistem, a še pred tem se javni ključ shrani v podatkovno bazo, zasebni pa se zašifrira z uporabniškim geslom. To je tudi idealna priložnost, da na primeru razložim, kako hibridna mobilna aplikacija komunicira s platformo. Za uspešno komunikacijo so potrebne sledeče komponente [24]:

- Datoteka JavaScript, ki vsebuje razred, preko katerega se zgodi klic metode `execute`, ki je implementirana v javanskem razredu. Razred


Slika 5.6: Posnetek pogleda, ki prikazuje obrazec za registracijo.

JavaScript vsebuje definicijo funkcije, ki kot parametre sprejme funkcijo, ki se izvede ob uspehu metode `execute`, funkcijo, ki se zgodi ob neuspehu ter dodatne parametre, ki jih želimo sporočiti vtičniku. Primer je prikazan v izseku 5.4.

- Javanski razred (glej izsek 5.5), v katerem je definirana metoda `execute`. V omenjeni metodi lahko hibridna aplikacija komunicira z napravo na isti način, kakor to počne navadna mobilna aplikacija. Ob izteku metode le-ta vrne odgovor na enega izmed dveh načinov:
  - Ob uspešni operaciji se pokliče metoda `success` javanskega razreda `callbackContext`, ki prejme parameter, ki ga želimo posredovati nazaj v aplikacijo. V našem primeru se aplikaciji vrne sporočilo o uspešni registraciji.
  - Ob neuspešni operaciji se pokliče metoda `error` javanskega razreda `callbackContext`, katera prav tako sprejme parameter, ki ga želimo posredovati nazaj v aplikacijo. V našem primeru se aplikaciji vrne sporočilo o neuspešni registraciji.
- Datoteka XML, ki vsebuje potrebne podakte za uspešno komunikacijo aplikacije z napravo.

Listing 5.4: Datoteka Javascript

```
var CreateKeys = {
  callCreateKeys: function (success, fail, user_id, password) {
    return cordova.exec(  // Klic execute funkcije
    success,       // Funkcija, ki se izvede ob uspehu
    fail,        // Funkcija, ki se izvede ob neuspehu
    "com.example.e_zdravje.CreateKeys",  // Lokacija javanskega
                // razreda CreateKeys
    "keys",        // Ime akcije
    [user_id, password] // Dodatni parametri
    );
  }
};
```

Listing 5.5: Javanski razred

```
public class CreateKeys extends CordovaPlugin {
  public boolean execute(JSONArray data) {
    // Ustvari par kljucev.
    KeyPair keyPair = genKeyPair();
    // Shrani javni kljuc v podatkovno bazo.
    storePublicToBase(keyPair.getPublic());
    // Zasifriraj zasebni kljuc ter shrani par
    // na datotecni sistem.
    saveKeyPair(keyPair);
    if (...) {
      callbackContext.success("Registracija uspesna");
      return true;
    } else {
      callbackContext.error("Registracija neuspesna.");
      return false;
    }
  }
}
```

### Prijava

Če uporabnik v začetnem meniju izbere možnost za prijavo, potem se mu prikaže pogled z vnosnimi polji za e-naslov ter geslo. Po potrditvi vnosa se podatki prenesejo na aplikacijski strežnik, kjer se primerjajo s podatki iz podatkovne baze. Ob uspešni prijavi se uporabnikova ID številka shrani v sejno spremenljivko in pogled se prestavi na glavni meni, v nasprotnem primeru pa se uporabniku prikaže sporočilo o napaki.

### Urejanje profila

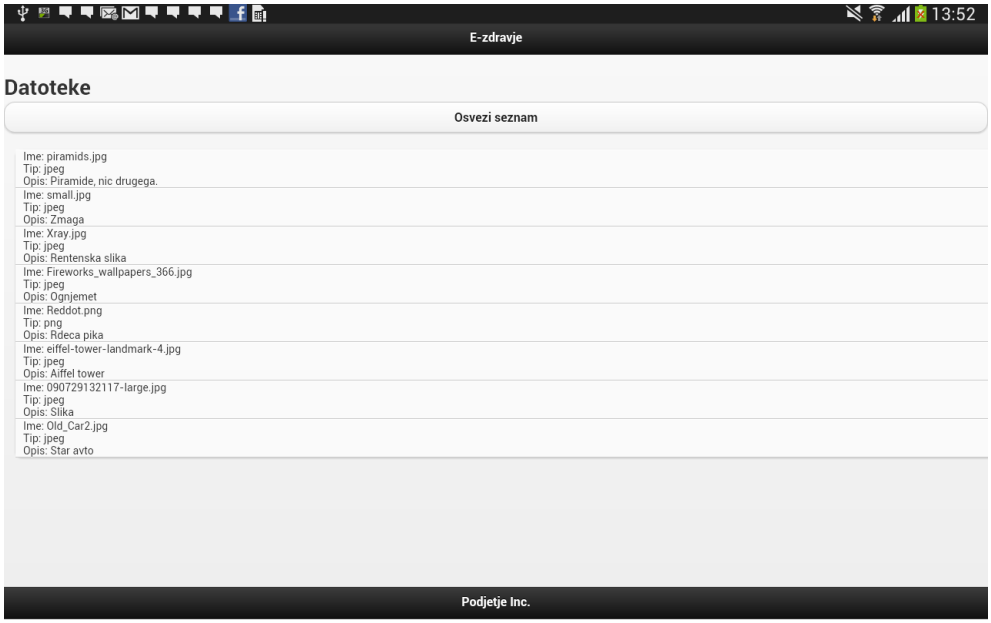
V glavnem meniju ima uporabnik možnost izbire urejanja profila. V primeru, da jo izbere, se pogled prestavi na obrazec, ki je zelo podoben obrazcu za registracijo na sliki 5.5, s to razliko, da je dodano še polje za vnos starega gesla. Razlikuje se tudi v tem, da so, razen gesel, vsa polja predhodno zapolnjena z obstoječimi podatki o uporabniku. Še pred prikazom strani se namreč naredi poizvedba po uporabnikovih obstoječih podatkih, katere se nato vpiše v vnosna polja.

Po potrditvi vnešenih podatkov se, v primeru ustreznosti e-naslova in starega gesla, posodobijo uporabnikovi podatki v podatkovni bazi. Opravi se tudi ponovno šifriranje zasebnega ključa z novim uporabniškim geslom.

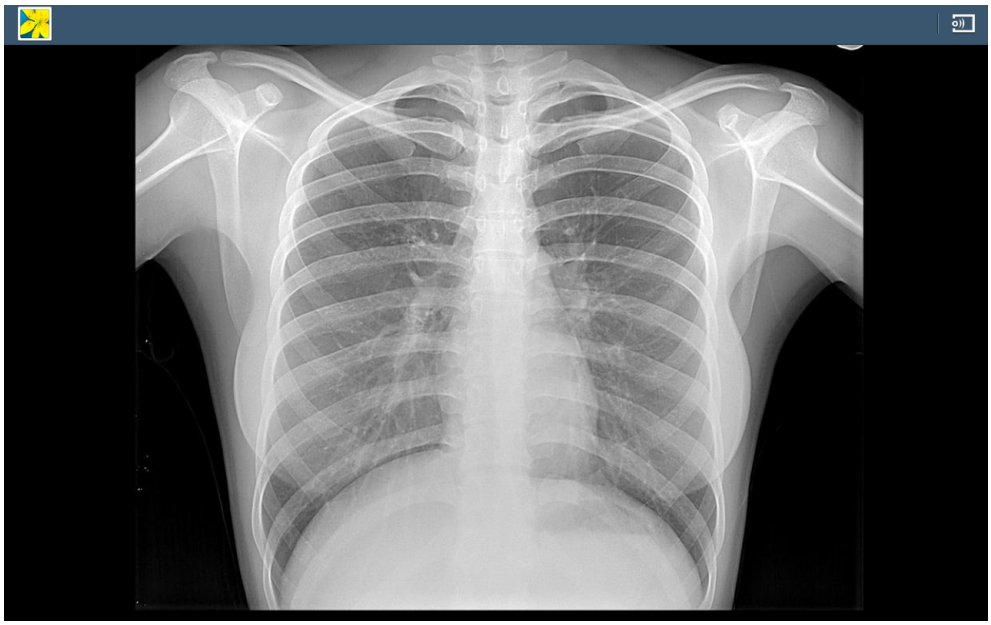
### Ogled kartoteke

Ob vsakem pritisku na gumb za ogled kartoteke se pogled preusmeri na stran, ki prikazuje vsebino kartoteke, kakor je prikazano na sliki 5.7. Ob tem se seznam, ki se na nahaja na omenjeni strani, napolni s podatki o datotekah, ki jih ima pacient shranjene v podatkovni bazi.

Ob pritisku na izbrano datoteko se uporabniku prikaže okence za vnos gesla, potrebnega za odšefriranje zasebnega ključa. Če je geslo pravilno, se zasebni ključ uporabi pri odšifriranju datoteke. Po končanem odšifriranju se datoteka prikaže s privzetim programom za ogledovanje (primer na sliki 5.8), kar je programsko rešeno z uporabo javanskega razreda `Intent`. Med tem




Slika 5.7: Posnetek pogleda, ki prikazuje seznam datotek.



Slika 5.8: Posnetek pogleda, ki prikazuje želeno sliko.

aplikacija stoji in se ponovno aktivira, ko uporabnik konča z ogledovanjem datoteke. Omenjeno programsko rešitev lahko bralec preveri v izseku 5.6. V pogledu ima uporabnik tudi možnost pritisniti na gumb za osvežitev seznama.

Listing 5.6: Koda za zagon privzetega programa za ogled datoteke

```
1 // Dolocitev tipa datoteke. Glede na tip datoteke
2 // se bo izbral privzeti program.
3 if (type.equals("pdf"))
4   finalMimeType = "application/pdf";
5 else if (type.equals("jpg"))
6   finalMimeType = "image/jpeg";
7 ...
8 // Ustvari instanco tipa Intent.
9 Intent intent = new Intent(Intent.ACTION_VIEW);
10 // Doloci pot do datoteke in tip datoteke.
11 intent.setDataAndType(path, finalMimeType);
12 // Zazeni privzeti program.
13 this.cordova.getActivity().startActivity(intent);
```

#### Odjava

Če uporabnik v glavnem meniju izbere možnost za odjavo, se sprostijo vse sejne spremenljivke, pogled pa se vrne na začetni meni.

### 5.2.4 Odšifriranje

Ker je odšifriranje najpomembnejši člen mobilne aplikacije, si zasluži svoje poglavje. Kar je opisano v naslednjih odstavkih, se zgodi med uporabnikovim pritiskom na izbrano datoteko ter njenim prikazom.

Kakor že opisano, se uporabniku po pritisku na datoteko prikaže dialog za vpis gesla. Po potrditvi vpisa se omenjeno geslo, skupaj z identifikacijsko številko uporabnika ter identifikacijsko številko datoteke pošlje do vtičnika, ki se nahaja v izvornem okolju. V omenjenem vtičniku se geslo uporabi za odšifriranje zasebnega ključa, katerega se nato shrani v spremenljivko tipa

`PrivateKey`. Če je bilo odšifriranje zasebnega ključa neuspešno, vtičnik že na tem mestu zaključi z izvajanjem ter vrne sporočilo o nepravilnem geslu.

Če si je uporabnik že pred tem ogledoval datoteko, ima njeno zašifrirano vsebino shranjeno na napravi (razen, če je datoteko ročno izbrisal). V tem primeru se vsebina prebere iz datotečnega sistema. Program nato naredi poizvedbo po zašifrirani vsebini simetričnega ključa, s katerim je bila zašifrirana datoteka, ter v primeru, da si uporabnik datoteke še ni ogledoval, tudi po zašifrirani vsebini datoteke.

Sedaj ima program zasebni ključ, z zasebnim ključem zašifriran simetrični ključ, ter s simetričnim ključem zašifrirano vsebino datoteke. Za pridobitev odšifrirane vsebine datoteke je odšifriranje potrebno izvesti v obratnem vrstnem redu, kakor je bilo izvedeno šifriranje. Tako je pri odšifriranju potrebno izvesti dva koraka, ki si sledita v naslednjem vrstnem redu:

- Odšifriranje simetričnega ključa z uporabo zasebnega ključa.
- Odšifriranje vsebine datoteke z dobljenim simetričnim ključem.

Proces je bolj podrobno opisan v naslednjih odstavkih, ki se sklicujejo na izseke 5.7, 5.8 ter 5.9.

Listing 5.7: Postopek odšifriranja vsebine datoteke

```
1 public static String decrypt(String cipherText, String cipherKey)
2   PrivateKey privateKey) {
3
4   byte[] plainKey = decryptRSA(cipherKey, privateKey);
5   byte[] plaintext = decryptRC4(plainKey, cipherText);
6   return new String(plaintext, "UTF-8");
7
8 }
```

Metoda `decrypt` prejme naslednje tri parametre:

- `cipherText`: spremenljivka vsebuje zašifrirano vrednost datoteke.
- `cihperKey`: spremenljivka vsebuje zašifrirano vrednost simetričnega ključa.
- `privateKey`: spremenljivka vsebuje zasebni ključ.

V telesu metode se izvedeta klica metod `decryptRSA` ter `decryptRC4`.

### Odšifriranje simetričnega ključa

Metoda `decryptRSA` prejme dva parametra in sicer tabelo tipa `byte`, ki vsebuje podatke o zašifriranem simetričnem ključu ter objekt tipa `PrivateKey`, ki vsebuje podatke o zasebnem ključu.

V metodi se najprej ustvari objekt razreda `Cipher`, ki nudi funkcije za šifriranje in odšifriranje. V našem primeru se ustvari objekt, ki opravlja transformacije po algoritmu RSA. Osnovni opis algoritma je podan v poglavju 3.2.5.

Listing 5.8: Podrobnosti metode decryptRSA

```
1 private static byte[] decryptRSA(byte[] cipherKey,
2   PrivateKey privateKey) {
3
4   Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
5   cipher.init(Cipher.DECRYPT_MODE, privateKey);
6   return cipher.doFinal(cipherKey);
7
8 }
```

Transformacija poteka po načinu ECB (`Electronic Code Book`), ki določa, da se enaki bloki čistopisa zašifrirajo v enake bloke tajnopisa [35]. S parametrom `PKCS1Padding` izberemo standard, ki določa preference varnostnih bitov, ki se dodajo čistopisu preden je zašifriran. Število varnostih bitov je enako številu bitov, ki jih je potrebno dodati, da je dolžina podatkov, ki se

bodo zašifrirali, enaka večkratniku velikosti bloka. Varnostni bit je vedno vsaj en [36, 37]. Zadnji parameter je v našem primeru nujen, saj omenjeni standard uporablja funkcija `openssl_seal`, s katero zašifriramo datoteko. Če parametra ne bi določili, bi objekt uporabil privzeti standard, ki pa je različen od `PKCS1`.

Naslednji ukaz določa, da bo naloga objekta Cipher odšifriranje ter zasebni ključ, ki ga bo pri tem uporabil. Metoda se konča z vrnitvijo odšifrirane vsebine simetričnega ključa, ki jo proizvede ukaz `doFinal()`.

### Odšifriranje vsebine datoteke

Listing 5.9: Podrobnosti metode decryptRC4

```
1 private static byte[] decryptRC4(byte[] plainKey,
2   byte[] cipherText) {
3
4   SecretKey skeySpec = new SecretKeySpec(plainKey, "RC4");
5   Cipher cipher = Cipher.getInstance("RC4");
6   cipher.init(Cipher.DECRYPT_MODE, skeySpec);
7   return cipher.doFinal(cipherText);
8 }
```

Metoda `decryptRC4` prejme dva parametra in sicer dve tabeli tipa `byte`, od katerih prva vsebuje podatke o simetričnem ključu, druga pa podatke o zašifrirani vsebini datoteke.

V telesu metode se najprej ustvari objekt tipa `SecretKey`, v katerega se shrani simetričen ključ. Ta bo uporabljen za odšifriranje vsebine, zašifrirane po algoritmu RC4.

V naslednj vrstici se objektu tipa `Cipher` določi, da bo njegova naloga odšifriranje. Prav tako se mu pripiše ključ, ki bo pri tem uporabljen. Telo metode se konča z vrnitvijo podatkov o odšifrirani vsebini datoteke.

Celoten proces odšifriranja se zaključi, ko prva izmed opisanih metod (metoda `decrypt`) vrne objekt tipa `String`, v katerega se shrani vsebina odšifrirane datoteke.

## 5.3 Testiranje

Vsako programsko rešitev je potrebno testirati in aplikacija E-kartoteka pri tem ni izjema. V tem poglavju je opisano testiranje spletne aplikacje, mobilne aplikacije ter analiza prometa med odjemalcem ter strežnikom.

### 5.3.1 Testiranje spletne aplikacije

Spletno aplikacijo sem testiral na računalniku HP Elitebook 8730w, ki ga poganja operacijski sistem Windows XP Professional, testiranje pa je podalo zadovoljive rezultate. Čas, ki je potreben za šifriranje različno velikih datotek, je zabeležen v tabeli 5.1.

### 5.3.2 Testiranje mobilne aplikacije

Kot že omenjeno, sem mobilno aplikacijo testiral na tablici Samsung Tab 3 ter na pametnem telefonu Samsung 2 mini. Na obeh napravah aplikacija teče pravilno in daje zadovoljive rezultate, a so razlike med napravama velike.

Tablica izvaja odšifriranje veliko hitreje kakor mobilni telefon, kar smo zaradi razlike v hitrosti procesorjev tudi predvideli. Časi odšifriranja različno velikih datotek so opisani v tabeli 5.1, kjer oznaka A pomeni, da si je uporabnik datoteko že ogledoval, oznaka B pa obratno.

Opazimo, da se v primeru, ko si uporabnik datoteke še ni ogledoval, celoten proces izvaja veliko dlje, to pa je posledica dejstva, da je zašifrirano vsebino datoteke potrebno še prenesti na napravo, kar v nasprotnem primeru ni potrebno.

Mobilni telefon trenutno ni zmožen odšifrirati datoteke, nekaj večje kakor 3Mb, saj nima dovolj delovnega spomina. Pri tablici je ta meja nekje pri 4,5Mb. Razlike so tudi v izgledu, a se s tem pri razvoju skorajda nisem ukvarjal.

| | Šifriranje | Odšifriranje | | | |
|---|---|---|---|---|---|
| Datoteka (kb) | Računalnik (s) | Tablica A (s) | Tablica B (s) | Telefon A (s) | Telefon B (s) |
| 71 | 1,5 | 2 | 6 | 12 | 13 |
| 661 | 7 | 3 | 7 | 14 | 17 |
| 1357 | 14 | 4 | 11 | 16 | 23 |
| 3094 | 33 | 5,5 | 24 | / | / |
| 4922 | 53 | / | / | / | / |

Tabela 5.1: Primerjava hitrosti šifriranja in odšifriranja na različnih napravah

### 5.3.3 Analiza prometa

Promet med odjemalci ter strežnikom sem analiziral s pomočjo programa *Wireshark*. Namen analize je bilo le preverjanje pravilnega delovanja protokola SSL/TLS. Ugotovljeno je bilo ustrezno delovanje protokola, kakor je to prikazano na sliki 5.9.

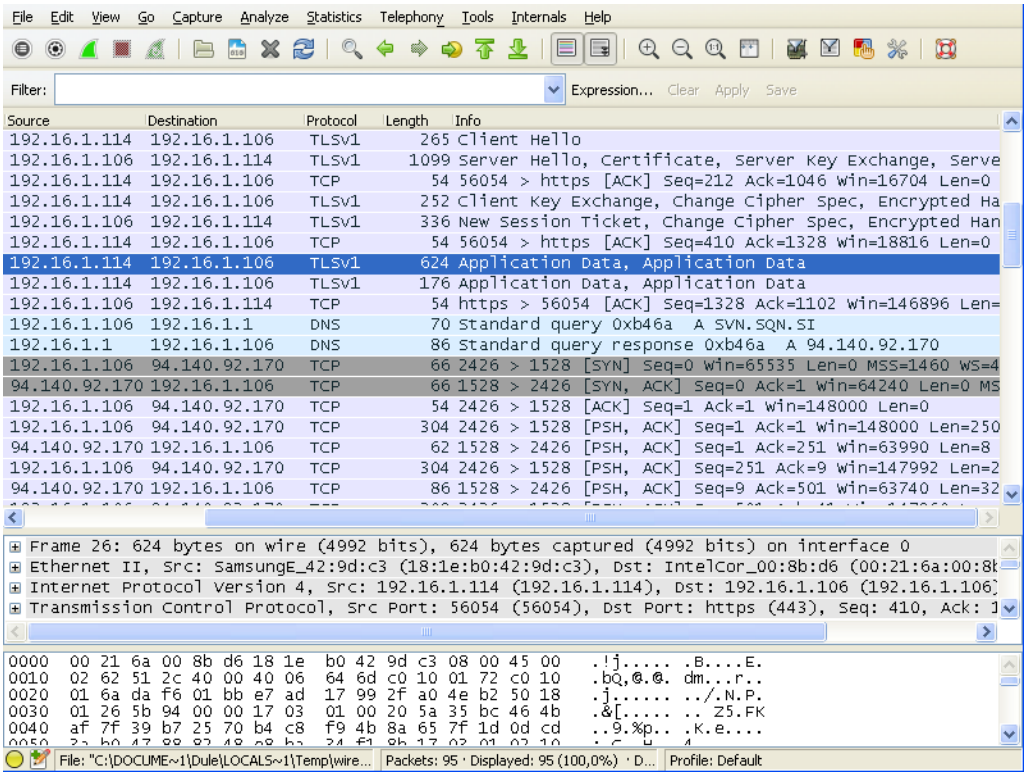
## 5.4 Možne izboljšave

O pomankljivostih in možnih (oz. potrebnih) izboljšavah bi se dalo napisati še eno diplomsko delo, v sklopu tega besedila pa se bom osredotočil na tistih nekaj najbolj pomembnih.

### 5.4.1 Izboljšave pri spletni aplikaciji

Spletna aplikacija trenutno obstaja le za namen testiranja mobilne aplikacije. Ne vsebuje nobenega stilskega pravila ali definicije v jeziku JavaScript.

Prva izboljšava, ki jo aplikacija potrebuje, je kompresija datotek pred šifriranjem, kar bi občutno pohitrilo tudi odšifriranje na strani odjemalca. Aplikacija bi poleg omenjenega lahko vsebovala še številne uporabne vsebine kot so npr. ogled datotek, možnost komuniciranja z zdravniškim osebjem, kontaktni podatki in podobno. Ker je spletna aplikacija trenutno namenjena




Slika 5.9: Analiza prometa.

le testiranju mobilne aplikacije je več pomankljivosti ali izboljšav na tem mestu nesmiselno iskati.

### 5.4.2 Izboljšave pri aplikacijskem strežniku

Aplikacijski strežnik bi zaradi boljše varnosti bilo potrebno razširiti z vmesnikom PDO ter z uporabo digitalnih potrdil, primerno pa bi bilo tudi, da bi se arhitekturo strežnika organiziralo tako, da bi se popolnoma držala principa RESTful.

#### Vmesnik PDO

Pri trenutni implementaciji strežnika so možni t.i. napadi z vrinjenjem, kjer se zlonamerna koda vstavi v ukazni niz SQL, ki se posreduje podatkovni

bazi kot legitimna poizvedba. Rešitev omenjenega problema je preprosta, tiči pa v uporabi PDO vmesnika. Le-ta nudi ukaze za varno komuniciranje z podatkovnimi bazami in prepreči tovrstne napade [38].

### Digitalna potrdila

Strežnik pri komunikaciji z odjemalci sicer že uporablja svoje digitalno potrdilo, a je le-to neoverjeno s strani registriranega overitelja, kar pomeni, da odjemalec strežniku ne more popolnoma zaupati.

Če je digitalno potrdilo neoverjeno s strani registriranega overitelja, lahko pride do napada, kjer se napadalec nepooblaščeno predstavi z identiteto strežnika, v nasprotnem primeru pa ima registrirani overitelj moč preklicati potrdilo in tako preprečiti njegovo nadaljno uporabo [39].

### Arhitektura RESTful

Arhitektura aplikacijskega strežnika se trenutno le delno drži principa REST, saj se pri komuniciranju uporablja le metoda HTTP POST. Strežnik bi bilo potrebno organizirati tako, da bi odjemalec pri komunikaciji s strežnikom uporabljal vse metode HTTP. Metode in njihov način uporabe so opisane v spodnjem seznamu.

- **GET**: uporaba pri pridobivanju podatkov.
- **POST**: uporaba pri vstavljanju podatkov.
- **PUT**: uporaba pri spreminjanju podatkov.
- **DELETE**: uporaba pri brisanju podatkov.

Poleg uporabe opisanih metod je za RESTful arhitekturo značilno tudi naslavljanje virov z naslovi URI, kakor je prikazano na spodnjem primeru [40].

```
https://E-kartoteka.com/Uporabniki/Pacienti/9
```

### 5.4.3 Izboljšave pri mobilni aplikaciji

Za namen boljše varnosti bi bilo tudi mobilno aplikacijo potrebno razširiti z uporabo digitalnih potrdil, veliko pa bi se dalo narediti tudi pri izgledu, dodatnih vsebinah in odzivnosti.

#### Izgled

Podobno kakor pri spletni aplikaciji je bil pri razvoju mobilne aplikacije izgled stranskega pomena. Rešitev uporabniku sicer nudi opozorila o napačnih vnosih ter o izvajanju nekega opravila, a bi bilo aplikacijo potrebno bolje oblikovati, predvsem pa poskrbeti za konsistenten izgled na vseh napravah.

#### Digitalna potrdila

Naprava s strežnikom sicer komunicira po varni povezavi, a slednji ne more zagotovo vedeti, da komunicira z uporabnikom, za katerega se le-ta predstavlja. Za možnost overitve uporabnika bi bilo tako skoraj nujno uvesti uporabo overjenih digitalnih potrdil, ki bi jih sicer lahko podpisal tudi zdravnik in ne agencija za overjanje, saj je z pacientom v osebnem odnosu.

#### Večnitnost

Čeprav je malo verjetno, da bo napravi potrebno odšifrirati datoteke, velike več Mb, pa se lahko dolgemu čakanju izognemo z implementacijo večnitnosti. Uporabnik bi tako lahko nemoteno uporabljal aplikacijo, med tem ko bi se v ozadju odvijala prenos ter odšifriranje datoteke.

#### Dodatne vsebine

Edini funkcionalnosti, ki jih aplikacija trenutno omogoča, sta poleg vpisa in registracije le še sprememba uporabnikovega profila ter ogled digitalne kartoteke. Dodati bi bilo potrebno še druge uporabne vsebine kot so brisanje datotek, urejanje datotek, komunikacija z zdravnikom, posredovanje datotek

in še veliko več. Brez dodatnih vsebin je tudi implementacija večnitnosti nesmiselna.

# Poglavje 6

# Sklepne misli

Diplomsko delo opisuje celoten proces razvoja aplikacije E-kartoteka, ki se je začel s študijem kriptografije, nadaljeval z izborom potrebnih tehnologij in metod, končal pa z načrtovanjem ter implementacijo rešitve.

Žal je aplikacija precej pomankljiva, saj sem se v razvoj podal z zelo osnovnim znanjem kriptografije ter brez znanja o razvijanju mobilnih aplikacij. Največ sivih las zagotovo povzroča dejstvo, da bi bilo rešitev za zagotovitev boljše varnosti potrebno skoraj nujno razširiti z uporabo digitalnih certifikatov. To seveda ni najbolj prijazno do uporabnikov, še posebno, če vzamemo v zakup dejstvo, da je večina bolnih ljudi starejših, ki pa zagotovo ne vedo, kako z njimi rokovati.

Če bi moral aplikacijo razviti še enkrat, bi se verjetno odločil za razvoj v izvornem okolju Android in ne z orodji Cordova. Aplikacija namreč veliko komunicira z napravo, kar pa aplikacije, razvite s tehnologijo Cordova navadno ne počnejo oz. to počno v manjšem obsegu.

Na samem koncu so opisane smernice za potencialen nadaljni razvoj aplikacije, do katerega pa bo skoraj zagotovo tudi prišlo. Rešitev trenutno omogoča le delno varno rokovanje z digitalno zdravstveno kartoteko, a predstavlja dober osnutek za prihodnji razvoj.

# Literatura


[1] B. McCarty, "The history of the smartphone," December 2011. Dostopno na: http://thenextweb.com/mobile/2011/12/06/the-history-of-the-smartphone/

[2] "Sap emr unwired," September 2013. Dostopno na: http://www.youtube.com/watch?v=ZqRZQ4IxMjE

[3] "mmr (mobile medical records)," May 2013. Dostopno na: https://play.google.com/store/apps/details?id=com.klouddata.mmr.main

[4] "Medscape," Januar 2013. Dostopno na: https://play.google.com/store/apps/details?id=com.medscape.android

[5] "Kriptografija," Julij 2013. Dostopno na: http://sl.wikipedia.org/wiki/Kriptografija

[6] "Kerckhoffs's principle," Avgust 2013. Dostopno na: http://en.wikipedia.org/wiki/Kerckhoffs's_principle

[7] "Aes," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

[8] "Encryption and cryptography," 2013. Dostopno na: http://www.cryptoforge.com/security.htm

[9] "Kako do ustreznega digitalnega potrdila," September 2012. Dostopno na: http://e-uprava.gov.si/storitve/cms/page/kje_dobiti_cert

[10] D. Seven, "What is a hybrid mobile app?" Junij 2013. Dostopno na: http://www.icenium.com/blog/icenium-team-blog/2012/06/14/what-is-a-hybrid-mobile-app-?

[11] M. Ouwehand, "Rsa: the algorithm," Julij 2001. Dostopno na: http://certauth.epfl.ch/rsa/node3.html

[12] "Rc4," August 2013. Dostopno na: http://en.wikipedia.org/wiki/RC4

[13] "Php," September 2013. Dostopno na: http://en.wikipedia.org/wiki/PHP

[14] "Java (programming language)," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Java_(programming_language)

[15] "Netbeans," September 2013. Dostopno na: http://en.wikipedia.org/wiki/NetBeans

[16] "Xampp," September 2013. Dostopno na: http://en.wikipedia.org/wiki/XAMPP

[17] "Adt plugin," September 2013. Dostopno na: http://developer.android.com/tools/sdk/eclipse-adt.html

[18] "Eclipse," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Eclipse_(software)

[19] "Sdk," September 2013. Dostopno na: http://developer.android.com/sdk/index.html

[20] "Android," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Android_(operating_system)

[21] "Android architecture," September 2013. Dostopno na: http://www.tutorialspoint.com/android/android_architecture.htm

[22] "Phonegap," September 2013. Dostopno na: http://en.wikipedia.org/wiki/PhoneGap

[23] "Phonegap supported features," September 2013. Dostopno na: http://phonegap.com/about/feature/

[24] "Developing a plugin on android," September 2013. Dostopno na: http://docs.phonegap.com/en/2.2.0/guide_plugin-development_index.md.html

[25] "jquery mobile," September 2013. Dostopno na: http://en.wikipedia.org/wiki/JQuery_Mobile

[26] "Object database," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Object_database

[27] "Relational database," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Relational_database

[28] "Web service," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Web_service

[29] "Rest," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Representational_state_transfer

[30] "Json," September 2013. Dostopno na: http://en.wikipedia.org/wiki/JSON

[31] "Hypertext transfer protocol," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[32] "Ssl/tls," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Transport_Layer_Security

[33] A. Maj, "Ssl/tls handshake," Februar 2005. Dostopno na: http://beefchunk.com/documentation/bin/apache/apache2_with_ssl_-tls/part1.htm

[34] "Multitier architecture," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Multitier_architecture

[35] "Ecb," Avgust 2013. Dostopno na: http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

[36] "Pkcs #1," Maj 2013. Dostopno na: http://en.wikipedia.org/wiki/PKCS1

[37] "Pkcs padding method," 2010. Dostopno na: http://pic.dhe.ibm.com/infocenter/zos/v1r11/index.jsp?topic=/com.-ibm.zos.r11.csfb400/pkcspad.htm

[38] "Orale functions (pdo_oci)," September 2013. Dostopno na: http://php.net/manual/en/ref.pdo-oci.php

[39] "Self-signed certificate," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Self-signed_certificate

[40] "Representational state transfer," September 2013. Dostopno na: http://en.wikipedia.org/wiki/Representational_state_transfer