Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Žiga Banič

# Oddaljeno namizje s pomočjo tehnologije HTML5

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Dejan Lavbič

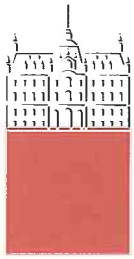
Ljubljana 2013

Rezultati diplomskega dela so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

*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: 00271/2012
Datum: 11.04.2012

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

Kandidat: **ŽIGA BANIČ**

Naslov: **ODDALJENO NAMIZJE S POMOČJO TEHNOLOGIJE HTML5**
**REMOTE DESKTOP WITH HTML5 TECHNOLOGY**

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Na področju oddaljenega dostopa do računalniškega namizja obstajajo številni pristopi (VNC, RDP idr.), ki omogočajo dostop do oddaljenih virov. Ključne razlike so predvsem v neodvisnosti na strani strežnika oz. odjemalca. V okviru diplomske naloge raziščite neodvisnost na strani odjemalca, kjer naj bo cilj podpreti čim širšo paleto naprav (namizni računalnik z različnimi platformami, pametni telefon, tablični računalnik itd.). V ta namen implementirajte prototip rešitve, ki temelji na HTML5, saj je omenjena tehnologija podprta na večini novejših brskalnikov.

Mentor:

doc. dr. Dejan Lavbič

Dekan:

prof. dr. Nikolaj Zimic

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Žiga Banič, z vpisno številko **63080097**, sem avtor diplomskega dela z naslovom:

*Oddaljeno namizje s pomočjo tehnologije HTML5*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  doc. dr. Dejana Lavbiča

- 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 15. januarja 2013 Podpis avtorja:

*Zahvala.*

*Zahvalil bi se svojemu mentorju doc. dr. Dejanu Lavbiču za potrpežljivost, strokovno usmerjanje, svetovanje in razpoložljivost pri nastajanju diplomskega dela.*

*Predvsem bi se zahvalil svojim staršem za nenehno pomoč, spodbujanje, ljubezen ter podporo. Zahvala gre tudi vsem, ki so prispevali vesele trenutke tekom študija.*

*Hvala!*

# Kazalo

**Povzetek**

**Abstract**

**1 Uvod** **1**

1.1 Oddaljeno namizje . . . . . 1

1.2 Opis problema . . . . . 2

1.3 Motivacija . . . . . 3

1.4 Rešitev . . . . . 3

**2 Oddaljeno namizje** **5**

2.1 Virtual Network Computing . . . . . 5

2.2 Remote Desktop Protocol . . . . . 7

2.3 Primerjava . . . . . 10

**3 Orodja** **11**

3.1 Obratni posredniški strežnik Nginx . . . . . 11

3.2 Python . . . . . 13

3.2.1 Spletni strežnik Tornado . . . . . 14

3.2.2 Zajem zaslona PySide . . . . . 15

3.2.3 Navidezna miška PyMouse . . . . . 16

3.2.4 Navidezna tipkovnica virtkey . . . . . 17

3.3 Spletno programiranje . . . . . 18

3.3.1 HTML5 . . . . . 19

3.3.2 Javascript . . . . . 22
3.3.3 Spletni brskalniki . . . . . 23

**4 Implementacija** **27**
4.1 Konfiguracija Nginx . . . . . 27
4.2 Programiranje v Python . . . . . 32
4.2.1 Zajem zaslona z uporabo PySide . . . . . 32
4.2.2 Uporaba navidezne miške PyMouse . . . . . 36
4.2.3 Uporaba navidezne tipkovnice virtkey . . . . . 38
4.2.4 Združitev in uporaba spletnega strežnika Tornado . . . 40
4.3 HTML5 in programiranje v Javascript . . . . . 47
4.3.1 Komunikacija WebSockets . . . . . 47
4.3.2 Element Canvas . . . . . 49

**5 Pregled in testiranje sistema** **55**
5.1 Pomanjkljivosti . . . . . 55
5.2 Ostali sistemi oddaljenega namizja . . . . . 57
5.3 Testiranje . . . . . 57

**6 Sklepne ugotovitve** **61**

# Povzetek

V diplomski nalogi smo implementirali oddaljeno namizje, s pomočjo novejše tehnologije HTML5. Razlog izbire le te je omogočiti dostop do oddaljenega sistema preko različnih računalniških platform. Današnji sodobni brskalniki poskrbijo za ustrezno podporo uporabe že prej omenjenih tehnologij. Prikaz namizja oddaljenega računalnika omogoča nov element canvas, kjer lahko manipuliramo nad izvedenimi dogodki miške, tipkovnice in dotika ter izvedemo ustrezen prikaz slike namizja. Način komuniciranja med strežnikom in odjemalcem je omogočila tehnologija WebSockets. To smo izkoristili za ustrezno pošiljanje podatkov in informacij za nadzor oddaljenega namizja. Primerno implementacijo ter združitev zgoraj opisanih tehnologij omogoča na strani odjemalca skriptni jezik Javascript. Možnosti pri izbiri strežniškega operacijskega sistema in temu primeren programski jezik za implementacijo WebSockets je bilo ogromno. Po kratkem pregledu uporabnosti smo se odločili za operacijski sistem Linux Ubuntu in odprto-kodni spletni strežnik Tornado, ki je napisan v programskem jeziku Python. Izvajanje navideznih dogodkov miške in tipkovnice ter zajem zaslona strežnika je primerno implementirano v tem programskem jeziku. Za testiranje celotne naloge je poskrbel obratni posredniški strežnik imenovan Nginx. Njegov namen je preusmeritev prometa do spletnega strežnika. V delu naloge smo usmerili pogled tudi na varnost izvajanja oddaljenega namizja. Na žalost se ni izšlo glede uporabe osebnih certifikatov, smo pa to nadomestili s prijavo v sistem preko varnostnega piškotka.

# Abstract

In our thesis we implemented a remote desktop with the newest technology HTML5. The main reason we choose this technology is to access the system through many different computing platforms. Present web browsers take care for the right support of using before mentioned technology. With the new element canvas we can manipulate and display the remote computer. To manipulate the system we just add event listeners for mouse, keyboard and touch events on the element. Technology WebSockets served as a communication between the server and client. Mainly it was used for sending proper data and information about the remote desktop. The implementation of both mentioned technologies was controlled with scripting language Javascript. There are many different operating systems to be used as a web server. In consideration, the server must have the proper support for implementing WebSockets with explicit programming language. After a brief review of system functionality, we made a decision for an operating system Linux Ubuntu and a open source web server Tornado, which is implemented with a programming language Python. With this programming language we imported a virtual mouse, keyboard and a screen capture technique. Proper system testing was cared with reverse proxy server named Nginx. His purpose is to balance and redirect traffic to the right web server. We also took a brief moment for the safety of using remote desktop. At first we wanted to add user certificates for authentication but there wasn't any support for mobile browsers. Therefore we replaced it with usage of safety cookie in login system.

# Seznam uporabljenih kratic

- Ajax (Asynchronous JavaScript and XML) - skupina tehnologij za ustvarjanje dinamičnih spletnih strani.
- ASCII (American Standard Code for Information Interchange) - 7-bitni nabor znakov za izmenjavo informacij.
- ATM (Asynchronous Transfer Mode) - asinhroni način prenosa; koncept telekomunikacije za prenos podatkov uporabnika.
- Canvas - element HTML5, ki omogoča izris objektov spletni strani.
- CSS (Cascade Style Sheet) - kaskadna stilska predloga; predstavi stil in način prikaza spletnih strani.
- DOM (Document Object Model) - je standard, ki je neodvisen od jezika in platforme ter opisuje objektno zgradbo spletne strani.
- HTML, HTML5 (HyperText Markup Language) - jezik za označevanje besedila spletnih strani.
- HTTP (Hypertext Transfer Protocol) - protokol na aplikacijski plasti, namenjen prenosu informacij.
- HTTPS (HyperText Transfer Protocol over SSL) - komunikacijski protokol za uporabo varne komunikacije preko omrežja. Uporabi plast protokola HTTP nad SSL.

- HMAC (Hash-based message authentication code) - koda za overitev sporočila, ki za svoje delovanje uporablja šifrirni ključ in razpršeno funkcijo (hash). Zagotavlja celovitost in overitev.
- LGPL (Lesser General Public License) - licenca za zastonjsko programsko opremo, ki jo je napisal Richard Stallman (projekt GNU).
- JPEG (Joint Photographic Experts Group) - vrsta kodiranja slik, kjer se del slikovnih informacij izgubi. Primeren za stiskanje fotografskih slik.
- JSON (JavaScript Object Notation) - format za prenos podatkov med spletnimi aplikacijami in spletnimi strežniki.
- RDP (Remote Desktop Protocol) - protokol za oddaljeno namizje.
- RFB (remote framebuffer) - preprost protokol, ki omogoča oddaljeni dostop do uporabniškega grafičnega vmesnika. Uporablja ga VNC za nadzor oddaljenega namizja.
- ORL (Olivetti & Oracle Research Laboratory) - raziskovalni laboratorij, kjer je bil razvit VNC.
- OS (Operating Sistem) - operacijski sistem; programska oprema, ki združi ostalo programsko in strojno opremo za vsakdanjo uporabo.
- VNC (Virtual Network Computing) - implementacija sistema za delitev grafičnega namizja, ki za svoje delovanje uporablja protokol RFB.
- SSL (Secure Sockets Layer) - kriptografski protokol za varno komunikacijo preko medmrežja.
- TCP (Transmission Control Protocol) - protokol, ki poskrbi za ustrezno povezavo dveh sistemov ter pravilno pošiljanje podatkov med njiju.
- XML (Extensible Markup Language) - razširljiv označevalni jezik; uporabimo ga pri označevanje podatkov, dokumentov.

# Poglavje 1

# Uvod

Uporaba oddaljenega namizja je trend, ki ga pri osebnem reševanju računalniških težav premalo uporabljamo. Drugače je seveda pri opravljanju določenih opravil, kjer je to potrebno. Svet današnje tehnologije omogoča nov način uporabe le-te.

## 1.1 Oddaljeno namizje

Oddaljeno namizje je poimenovanje za grafično delitev namizja. Poznamo več implementacij sistema. Med bolj znane spada Virtual Network Computing [8, 17], razvit v raziskovalnem laboratoriju Olivetti & Oracle ter Remote Desktop Protocol [13], razvoj podjetja Microsoft.

Pri obeh je za uspešno vzpostavitev oddaljenega namizja zahteva po uporabi programske opreme strežnika in odjemalca. Računalnika si med sabo preko omrežja pošiljata ustrezne dogodke miške in tipkovnice ter posodobitve zaslona. Za medsebojno pošiljanje informacij VNC uporablja protokol RFB [10], RDP pa je že lastniški protokol.

Splošna razlika med njima je ta, da je VNC neodvisen glede platforme, tako strežnik kot odjemalec, pri RDP pa zgolj odjemalec. Modificiranih in implementiranih je bilo nekaj protokolov RDP ostalih podjetij, ki podpirajo določene funkcionalnosti protokola, razvite s strani podjetja Microsoft.

Glavna sprememba je bila podpora strežnika na sistemih Unix in Linux.

Spremembe se opazijo še na določenih področjih. Ena izmed njih je način pošiljanja podatkov glede osveževanja zaslona. VNC z protokolom RFB pošilja podatke o barvah slikovnih pik določene pozicije. S pomočjo kompresije zagotovi učinkovitejši in hitrejši prenos teh podatkov preko omrežja. Strežnik RDP uporabi svoj grafični gonilnik za pretvorbo prikaza v informacijske pakete, ki te preko protokola RDP pošlje odjemalcu. Ta pakete prevede v ustrezen vmesnik grafične naprave OS Microsoft Windows. Vmesnik je aplikacijski programski vmesnik in komponenta jedra operacijskega sistema Microsoft Windows, ki skrbi za ustrezen prikaz na izhodni napravi, kot je monitor.

Na kratko smo opisali glavne podobnosti ter razlike v medsebojni primerjavi obeh tehnologij. Podroben opis posameznega sistema bomo podali v naslednjem poglavju.

## 1.2 Opis problema

Dandanes imamo na voljo veliko različnih platform, ki jih uporabljamo v vsakdanjem življenju. Vse od telefonov, tablic, prenosnih do namiznih računalnikov. Kljub nenehnemu razvoju, je še vedno mnogo različnih operacijskih sistemov, ki delujejo na različnih platformah in to vpliva tudi na sam razvoj programske opreme. V kolikor uporabljamo določeno programsko opremo na enem operacijskem sistemu, je majhna verjetnost, da bo to opremo mogoče uporabljati na drugem operacijskem sistemu. Edina možnost, ki nam preostane, je iskanje alternative, ki je v večini primerov neprimerljiva.

Pri uporabi različne programske opreme za oddaljeno namizje sistema na različnih platformah je malo drugače. Na voljo imamo kar nekaj profesionalnih orodij, ki jih lahko namestimo na strežniku in odjemalcu. Z ustrezno konfiguracijo orodij strežnika in odjemalca lahko dokaj hitro komuniciramo ter upravljamo z oddaljenim sistemom. Oddaljeno namizje nam lahko omogoči delo na daljavo, v primeru ko ni možen fizični dostop do tega sistema. Tak

način dela uporablja skoraj vsako podjetje (skrb za administracijo sistemov in reševanje težav uporabnikov) ali navadni uporabniki, ki želijo imeti dostop do svojega sistema. S tem prihranimo čas in denar.

Na mestu je vprašanje, ali bi bilo možno odjemalcu omogočiti dostop do sistema brez nadležnih namestitev, konfiguracij novih oziroma obstoječih orodij in uporabiti nam znano opremo, ki jo redno uporabljamo. Rešitev je lahko spletni brskalnik z novo različico označevalnega jezika spletnih vsebin, imenovan HTML5 [6] iz družine HTML. Na strani strežnika smo implementirali ustrezen spletni strežnik, ki lahko upravlja s komunikacijo ter skrbi za ustrezno upravljanje sistema.

## 1.3 Motivacija

Predlog izbire tehnologije HTML5 za prikaz ter upravljanje namizja je prišla s strani mentorja, saj nisem imel predhodnih znanj. Po branju literature ter prikazu dosedanjih dosežkov ustrezne uporabe te tehnologije je vzklila velika želja po učenju in novem znanju. Pozornost je pritegnilo dejstvo, da je mogoče upravljati namizje računalnika preko spletnega brskalnika, ki deluje kot odjemalec.

To omogoča neskončno možnosti nadaljnjega razvoja ter prikaže dejansko moč in konkurenčnost le-teh, saj jih redno dnevno uporabljamo.

## 1.4 Rešitev

Razvoj novih tehnologij strmo narašča in to tudi lahko vidimo pri nenehnem posodabljanju spletnih brskalnikov, kjer poskrbijo za kompatibilnost novo razvitih elementov.

HTML5 je prinesel mnogo novih funkcionalnosti, ki jih lahko izkoristimo in tako obogatimo našo spletno stran. V našem primeru smo uporabili nov način obojestranske komunikacije strežnika in odjemalca preko tehnike WebSockets [6, 11] ter prikaz slike strežnika v brskalniku, značka canvas [4]. Can-

vas ima podporo zaznave dogodkov miške in tipkovnice. S pomočjo teh dogodkov smo lahko prepoznali natančno pozicijo miške, uporabnikov klik na gumb ter tipkanje na tipkovnici. Te podatke smo potem naprej posredovali strežniku.

Na strani strežnika je za nadzor in komunikacijo skrbel spletni strežnik Tornado [1, 16]. Ker je spletni strežnik Tornado napisan v programskem jeziku Python [2], smo v tem jeziku tudi poskrbeli za nadzor sistema. Podatke odjemalca smo predelali v ustrezen ukaz in ga nato izvršili. Problem se je pojavil pri izvajanju ukazov tipkovnice, saj je tipkovnica na strežniku prepoznala zgolj alfa-numerične znake, zato je bilo potrebno posebne znake ustrezno definirati.

Posodobitev zaslona strežnika poteka tako, da odjemalec, po uspešnem prejetju in prikazu slike na elementu canvas, pošlje sporočilo, kjer zahteva posodobitev zaslona. Strežnik po prejetju tega sporočila naredi sliko namizja strežnika, jo zakodira v format, prepoznaven elementu canvas, ter te podatke pošlje odjemalcu. Odjemalec pripne podatke in tako vidimo sliko namizja strežnika. Celoten postopek se nato ponovi. S tem poskrbimo, da se podatki na strani odjemalca ne kopičijo, saj bi lahko konstantno pošiljanje novih slik pred obdelavo trenutne onemogočil tekoče delovanje.

V naslednjem poglavju bomo podali opis dveh sistemov, ki sta že dolgo priznana za uporabo kot sistem oddaljenega namizja. Dalje bomo pozornost namenili našemu izbranemu sistemu, kjer v poglavju 3 sledi podroben opis vseh orodij za implementacijo ter dalje v poglavju 4 dejanska implementacija sistema za uporabo oddaljenega namizja. Med testiranjem implementiranega sistema smo se srečali s določenimi nevšečnosti. Te so opisane v poglavju 5. Na koncu v poglavju 6 sledi zaključek, kjer podamo kratek opis celotne naloge ter končno oceno izvajanja le-te.

# Poglavje 2

# Oddaljeno namizje

## 2.1 Virtual Network Computing

Virtual Network Computing je implementacija sistema za delitev grafičnega namizja. Pri svojem delovanju uporablja protokol RFB ("remote framebuffer") za oddaljen nadzor drugega računalnika.

Sistem je bil razvit v raziskovalnem laboratoriju Olivetti & Oracle (ORL) [8, 17]. Namen razvoja je bilo omogočiti članom dostop do svojih računalnikov iz katerekoli lokacije (npr. druge pisarne, javnih terminalov, letališč).

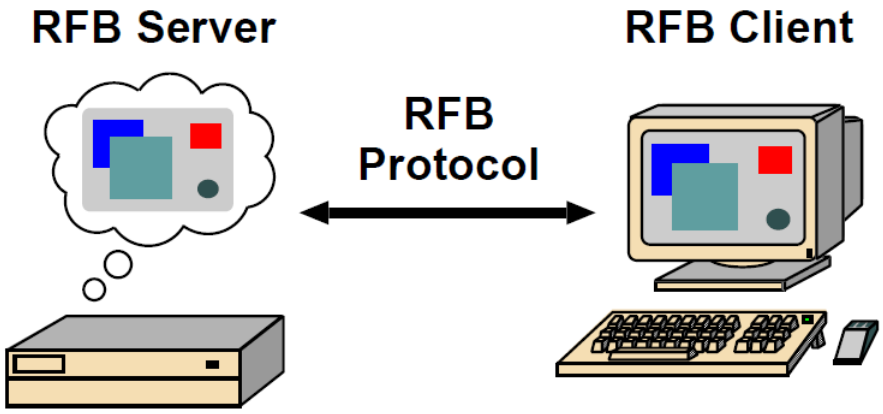
VNC je neodvisen od platforme. Tako se lahko odjemalec poveže na strežnik istega ali drugega operacijskega sistema, kot ga ima sam. Zaradi zgornjih dveh lastnosti, neodvisnost od platforme in možnost dostopa od oddaljenih lokacij, sistem omogoča mobilno računalništvo. Poleg tega imamo možnost dostopa do določenega namizja iz več različnih lokacij.

Virtual Network Computing je dobil svoje ime v laboratoriju ORL. Leta 1994 so v tem laboratoriju razvili "tanek" odjemalec imenovan Videotile. To je bila naprava z zaslonom LCD, pisalom in mrežno povezavo v načinu ATM (asinhron način prenosa podatkov). Ustvarjena je bila, da omogoča prikaz videa in interakcijo z aplikacijami. Sistem je iz oddaljenega računalnika pošiljal uporabniški vmesnik kot video na napravo LCD. Videotile je deloval dobro, ampak uporabil ogromno mrežnih kapacitet. Z dodatnim razvojem je

bil sistem zmožen pošiljati samo del zaslona uporabniškega vmesnika, ta, ki se je spreminjal. To je bila tudi ideja za nov protokol, torej VNC. Njegova zasnova omogoča, da se poenostavi uporabo novih odjemalcev na različni strojni opremi.

Moč sistema je predvsem v svoji lastni preprostosti, kjer za razliko od ostalih protokolov za oddaljeni nadzor omogoča neodvisnost od operacijskega sistema.

Tehnologija VNC je preprost sistem za oddaljen dostop do grafičnega uporabniškega vmesnika. Deluje na "framebuffer"načinu prenosa in se zaradi tega lahko uporablja v vseh aplikacijah in operacijskih sistemih. V bistvu se lahko aplicira v napravah, ki imajo vsaj določeno komunikacijsko povezavo. Protokol uporablja zanesljiv način prenosa podatkov, kot je na primer TCP/IP.




Slika 2.1: Shema koncepta protokola RFB. [10]

Pri VNC se uporabljajo naslednji pojmi:

- Odjemalec VNC je program, s katerim uporabnik komunicira (t.j. vhodna naprava, zaslon).
- Odjemalec prevzame kontrolo nad strežnikom.

- VNC protokol (RFB) omogoča oddaljen dostop in deluje po principu preprostega grafičnega primitiva od strežnika k odjemalcu ("Postavi pravokotnik točkovnih podatkov na določenem X, Y položaju.").

- VNC strežnik je program na napravi, ki deli svoj zaslon. Strežnik je med delovanjem odjemalca pasiven.

Za prikazovanje različnih komponent uporabniškega vmesnika imamo na voljo veliko shem kodiranja točkovnih podatkov. Pred pošiljanjem posameznega pravokotnika se lahko odločimo, kakšno kodiranje bomo izbrali. S tem se izognemo predvsem možnim neučinkovitostim protokola VNC in tako kar se da izkoristimo omrežne vire, hitrost procesiranja strežnika in hitrost izrisa odjemalca.

## 2.2 Remote Desktop Protocol

Remote Desktop Protocol je protokol za varno omrežno komunikacijo, ki ga je razvil Microsoft. Zasnovan je na tak način, da olajša interakcijo uporabnika z oddaljenim računalnikom. Protokol omogoča prenos grafičnih informacij oddaljenega računalnika k uporabniku in uporabnikove ukaze k oddaljenemu računalniku. RDP je bil razvit z namenom, da omogoča podporo različnim vrstam omrežnih topologij (npr. ISDN, POTS) in protokolom LAN (npr. NetBIOS, TCP/IP).

Za odjemalca RDP velja neodvisnost od platforme, kjer protokol podpira večino različic operacijskih sistemov, kot so Microsoft Windows, Linux, Unix, Macintosh OS X, Android in druge. Strežnik RDP (oddaljen računalnik) nima te lastnosti, toda vseeno obstajajo odprto kodni programi, ki delujejo na platformi Linux (na primer Xrdp [18]).

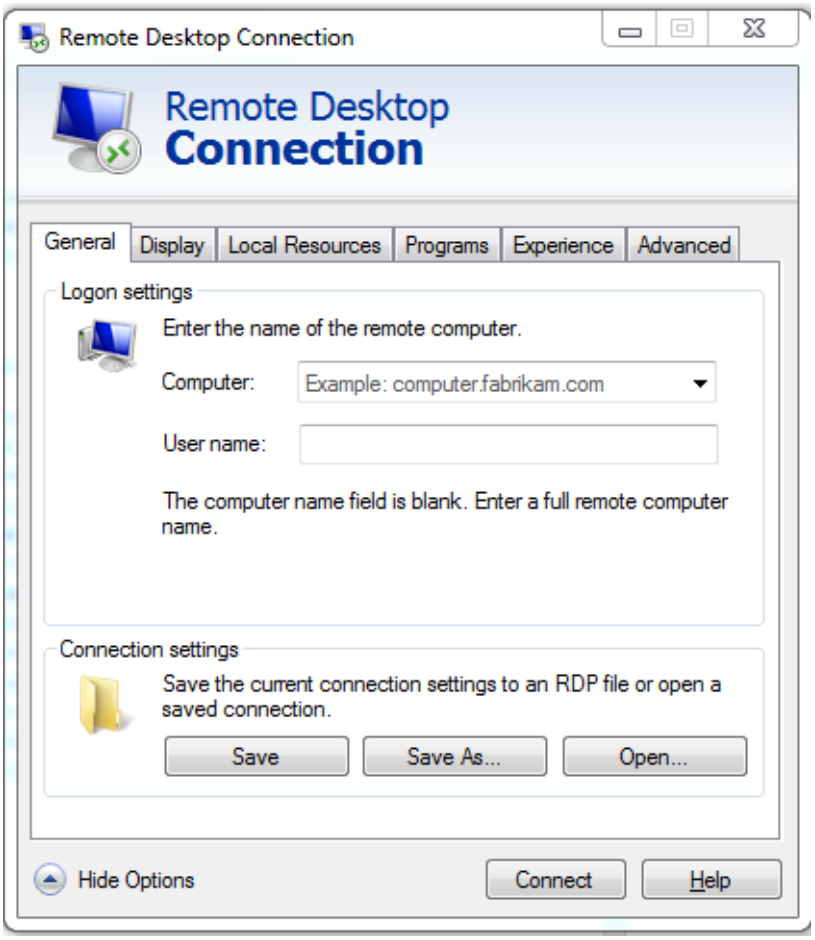
Protokol ima naslednje lastnosti:

- Šifriranje: protokol za varovanje podatkov uporablja kriptosistem RSA z RC4. Ta uporabi tekočo šifro (stream cipher) za šifriranje podatkov. RC4 se uporablja za varno komuniciranje preko omrežja.

- Zmanjševanje prometa preko omrežja: obstaja več mehanizmov, kot npr. kompresija podatkov, predpomnenje bitnih slik.
- Prekinitev gostovanja: uporabnik lahko brez odjave ročno prekine sejo s oddaljenim namizjem. Na to prekinjeno sejo se ponovno avtomatsko poveže pri prijavi v sistem. Če se odjemalčeva seja nepričakovano prekine (npr. napaka na omrežju), je uporabnik še vedno prijavljen v sistem, a ni več povezan.
- Odložišča: uporabnik lahko kopira, prilepi ter izbriše besedila in grafiko v aplikacijah na lokalnem računalniku in tistimi, ki delujejo na seji oddaljenega namizja.
- Preusmeritev tiskanja: aplikacije na seji oddaljenega namizja lahko tiskajo na tiskalnik, ki je priklopljen na napravo odjemalca.
- Navidezni kanali: omogočajo obogatitev obstoječih aplikacij in razvoj novih.
- Oddaljen nadzor: podporno osebje lahko nadzira sejo oddaljenega namizja. S tem ima osebje možnost diagnostike sistema in popravljanje napak.
- Uravnoteženje bremena omrežja.

### Poimenovanje programske opreme

Microsoft je uradno poimenoval programsko opremo strežnika RDP kot Remote Desktop Service (RDS) v preteklosti znan kot "Terminal Services" ter odjemalca kot Remote Desktop Connection (RDC) v preteklosti znan kot "Terminal Services Client". Od verzije OS Microsoft XP (razen edicija Home) je že nameščena programska oprema RDC. Dostop do njih je enostaven preko ukaza `mstsc.exe` . Z izvedbo ukaza se nam prikaže okno RDC za vpis imena strežnika ter po želji vpis naprednih podatkov za dostop do le tega. Slika 2.2 nam prikaže to napredno okno za vpis podatkov.




Slika 2.2: Prikaz okna po uporabi ukaza `mstsc`.

### Risanje elementov

RDP se vključi v grafični sistem samega OS Windows tako kot pravi grafični gonilnik, le da se predstavi kot navidezni gonilnik. Namesto pošiljanje ukazov fizični napravi uporabi inteligentno odločitev pri izbiri kodiranja ukazov za ustrezen RDP format. To so lahko od kodiranja delov slik do manjših ukazov kot je risanje preprostih linij ali izris teksta na določeni poziciji.

S takšnim pristopom se RDP pri zajemu izogne veliki potrošnji procesorskega časa, omogoči fleksibilnost pri kodiranju zaslona ter ponudi hitro delovanje samega sistema.

## 2.3 Primerjava

Sistema se razlikujeta predvsem pri načinu uporabe zajema zaslona ter uradni podpori sistema. VNC z preprosto in osnovno kompresijo pošilja bloke slike, RDP pa uporabi inteligentno izbiro glede kodiranja ukazov. Razlika je tudi v tem, da pri VNC ni na voljo podatkov o samih grafičnih plasteh zaslona. To omogoči enostavnost uporabe na vsaki napravi, saj se spreminja zgolj slika zaslona. RDP gre malo dalje, saj optimizira zajem na minimum potrebnih informacij, ki na strani odjemalca ustvarijo enako posodobitev. Veliko nam pove tudi podpora uporabnosti sistema na različnih OS. Same razlike glede podpore smo že omenili na začetku poglavij 2.1 in 2.2 . Glavni razlog pri izbiri določenega sistema sloni na našemu namenu uporabe ter potreba po specifičnih zahtevah.

# Poglavje 3

# Orodja

Sledi opis orodij na strani strežnika in odjemalca, ki služijo za uspešno izvajanja oddaljenega namizja. Pri izbiri operacijskega sistema za implementacijo strežnika glede naših potreb sta nekako kandidirala zelo znana Windows 7 ter Linux Ubuntu. Zaradi možnosti razširjanja, uporabnosti vseh orodij ter tudi prepoznavnosti je na koncu prevladala izbira operacijskega sistema Linux Ubuntu.

## 3.1 Obratni posredniški strežnik Nginx

V svetu spletnih strežnikov je že nekaj časa na vrhu zelo znani Apache. Poznan je po fleksibilnosti, razširljivosti in uporabnosti. Zadnje čase je odprtokodni spletni in obratni posredniški strežnik Nginx [7] pridobil na priljubljenosti, saj ga odlikujejo preprostost, zanesljivost, skoraj zanemarljiva obremenjenost in hitro delovanje. Prednost Nginx pred Apache je enostavna konfiguracija, saj lahko z nekaj vrstic v konfiguracijski datoteki na primer ustrezno priredimo preusmeritev do spletnih in poštnih strežnikov, uredimo varni dostop z SSL, serviramo statične datoteke spletnih strani, priredimo zapise napak strežnika v datoteke, in še bi lahko naštevali.

V našem primeru Nginx služi kot obratno posredniški ("reverse proxy") strežnik. Obratni posredniški strežnik je tip posredniškega strežnika, ki v

imenu odjemalca pridobi informacijo od spletnega strežnika in nato posreduje to informacijo odjemalcu. Skupaj s spletnim strežnikom sta povezana na internem omrežju. Po želji lahko implementiramo več spletnih strežnikov na interno omrežje, kjer bo obratni posredniški strežnik skrbel za ustrezno komunikacijo do vseh spletnih strežnikov.

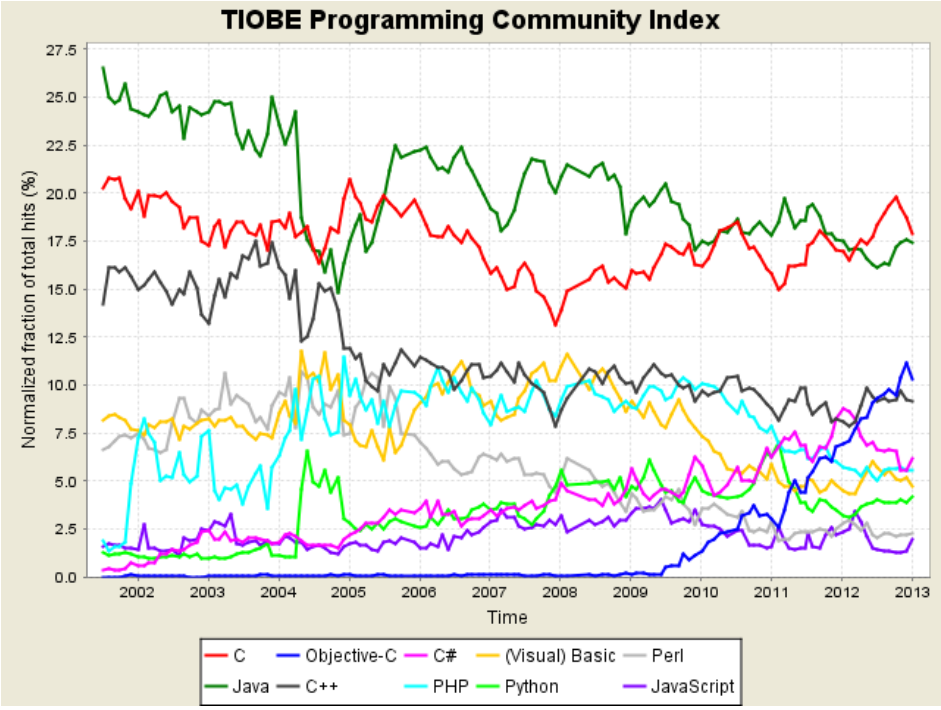
V poglavju 3.3.1 je napisano, da je za uspešen prenos sporočil in podatkov preko protokola WebSocket potrebna povezava TCP, ki Nginx uradno še nima podpore v zadnji stabilni različici. Pri iskanju rešitve lahko opazimo, koliko dodatnih razširitev ima Nginx, saj so programerji razvili ogromno modulov za uporabo. Izmed njih tudi posredniški modul TCP, ki reši zgoraj navedene težave. Negativna stran uporabe tega modula je, da ne moremo uporabiti hkrati ista vrata za sprejem zahteve na modulu TCP in HTTP. Za ustrezno uporabo moramo vključiti modul v namestitveno konfiguracijo. Hkrati lahko še dodamo dodatne module, kot sta na primer `perl` in `ssl`, saj niso avtomatsko vključeni. V kolikor lahko module tako dodajamo, jih lahko na ta način tudi onemogočimo. S tem zagotovimo, da se moduli, ki jih ne želimo uporabljati, ne namestijo. Dalje izvedemo korak namestitve in ustrezno konfiguracijo strežnikov.

Za naše zahteve je Nginx opravil svoje delo več kot zadovoljivo. Branje literature nam daje občutek, da se še bolje izkaže pri obvladovanju večjega prometa.

Slika 3.1: Shema komunikacije med odjemalcem in spletnim strežnikom.

## 3.2 Python

Python je znan kot preprost visoko-nivojski programski jezik. Kljub temu v kategoriji popularnosti visoko-nivojskih jezikov Java in C še vedno kraljujeta na vrhu te, a vseeno lahko opazimo bogatost programskega jezika Python. Njegovo (ne)popularnost lahko opazimo na sliki 3.2.




Slika 3.2: Prikaz rasti popularnosti programskih jezikov. [14]

Za način procesiranja v nizko-nivojske jezike uporablja interpreter ali tolmač. Tolmačenje programske kode lahko izvedemo na dva načina. Pri pisanju daljših programov in metod, ki jih bomo še večkrat lahko izkoristili uporabimo skriptni način, drugače pa lahko za testiranje delcev kode uporabimo interaktivni način. Obstajajo razvojna okolja, ki z urejevalnikom ter ukazno vrstico izkoriščajo oba načina ter omogočijo lažje in preprostejše pisanje programov. Ti so nekoliko podrejeni uporabi operacijskega sistema, kjer je za MS Windows možno uporabiti PythonWin, za Linux pa Eric Python

IDE.

V vsakem primeru ali za pomoč uporabimo razvojno okolje ali kar navaden urejevalnik. Dokaj hitro spoznamo, da je programiranje enostavno. Vsiljuje nam dobro lastnost programiranja, ki jo marsikdo zanemari pri ostalih programskih jezikih in to je pisanje "lepe" kode. S tem omogočimo berljivost, lažje iskanje napak ter tudi tolmačenje kode v glavi, kjer lažje sledimo izvajanju poteka programa.

Z izbiro programskega jezika Python smo želeli dokazati, da lahko konkurira pri uporabi spletnega strežnika, simulira navidezno miško in tipkovnico ter izvede hiter in učinkovit zajem zaslona sistema. Vsi ti elementi so ustrezno implementirani v tem jeziku. Operacijskemu sistemu je prilagojena zgolj navidezna tipkovnica. Ostali elementi ponujajo podporo za različne operacijske sisteme.

### 3.2.1 Spletni strežnik Tornado

Tornado je odprto-kodna verzija spletnega strežnika, uporabljenega pri socialnem zbiralniku FriendFeed. Facebook je odkupil FriendFeed in tako je dejansko postal odprto-koden. Krasijo ga lastnosti, kot so razširljivost, hitrost, enostaven za implementacijo, uporaben za izvajanje različnih aplikacij in orodij v realnem času, podpira komunikacijo z WebSockets, možna je uporaba orodij za upravljanje z varnostjo, kot je integracija SSL in uporabniško prijavo, neskončno izvajanje funkcij po preteku časovnega intervala ter omogoča asinhrono interakcijo z raznimi bazami podatkov.

Tradicionalni spletni strežniki so se srečevali s problemi sočasne povezave že pri okoli 10.000 povezavah, kar je bilo glavno vodilo za nastanek visoko zmogljivega spletnega strežnika, ki bi rešil tegobe. Očitno je problem postal zelo znan, saj je tudi dobil svoje poimenovanje kot problem C10K.

Opisane pozitivne lastnosti in osebna mnenja različnih uporabnikov so tudi botrovali k izbiri spletnega strežnika Tornado, ki se je na koncu izkazal za pravilno. K temu moramo pripomniti, da je zelo dobro opisovanje primerov in razlag v obširni dokumentaciji rešilo marsikateri dvom. Pri spletnem

strežniku smo za način komuniciranja uporabili WebSockets, dodali prijavo uporabnika s shranjevanjem zaščitnega piškotka, servirali statične datoteke za kaskadno stilsko podlogo ter omogočili varno povezavo z digitalnim certifikatom strežnika.

Z implementacijo varne povezave želimo zavarovati komunikacijo, ki poteka med odjemalcem in strežnikom. Dodatno in nujno varovalo je prijava uporabnika z ustreznim uporabniškim imenom, geslom in ključem, saj ne želimo, da nam po strežniku brska neznanec in nam napravi škodo. V primeru pravilnega vnosa informacij se naredi zaščitena kombinacija z ustrezno kriptografsko funkcijo in shrani kot piškotek za določen čas. Spletni strežnik nato odjemalca preusmeri na vir, ki dovoli komunikacijo. Za lepši ogled spletne strani smo uporabili kaskadno stilsko podlogo, ki je statično servirana s strani strežnika.

Če naredimo pregled vseh funkcionalnosti, ki nam jih nudi spletni strežnik Tornado, lahko opazimo, da smo za naše potrebe izkoristili le majhen del tega. To nam da vedeti, koliko možnosti implementacije dejansko ponuja.

### 3.2.2 Zajem zaslona PySide

Ključnega pomena izvajanja oddaljenega namizja je bilo posodabljanje zaslona. V želji po hitri, preprosti in učinkoviti posodobitvi zaslona je odločal časovni test izvedbe zajema zaslona in način posodobitve le-tega.

Kratki pregled svetovnega spleta je zagotovil kar nekaj primerov, ki uspešno zajamejo trenutno prikazan zaslon. Zaradi kriterija preprostosti je bilo za testiranje izbrano orodje GTK+ GDK in PySide [12]. PySide je dejansko alternativa PyQt z LGPL licenco. Obe orodji po navadi lahko uporabimo pri programiranju grafičnega uporabniškega vmesnika. Testiranje je potekalo tako, da je bilo potrebno s ustreznimi funkcijami orodja uspešno zajeti zaslon in ga shraniti kot sliko za nadaljnjo uporabo. Tu se je izkazalo orodje PySide, saj je bil čas izvedbe zgoraj opisanega postopka več kot 50 % krajši. Hitrost zajema zaslona je dejansko botrovala pri končni izbiri uporabe orodja PySide, saj nam po testiranju končnega prototipa omogoča skoraj tekoče po-

sodabljanje zaslona.

PySide je eno izmed orodij za programiranje grafičnih uporabniških vmesnikov, ki je bil na začetku razvoja sponzoriran s strani podjetja Nokia. Zgoraj smo omenili, da je alternativa dokaj znanega orodja PyQt, saj je nastal nekako kot odgovor neuspešnosti licenciranja prej omenjenega PyQt pod LGPL licenco. Kljub temu da sta bili orodji razviti z različnimi tehnologijami, je prenos med njima enostaven. Če primerjamo med sabo obe orodji, je nekaj razlik pri samem klicu in uporabi določenih modulov, razredov. PySide ima podporo pri vseh platformah kot Qt, kjer lahko napisane programe dejansko brez modifikacij testiramo. Kot odprto-koden projekt je njegova prihodnost nekoliko negotova, saj večino novih del pride s strani nadobudnežev. Pozitivna stran vsega je definitivno licenciranje pod LGPL, ki podpira delo le teh.

### 3.2.3 Navidezna miška PyMouse

Vhodni podatki miške osebnih računalnikov ali dotik zaslona na naprednih napravah so osnova za kontrolo teh. Z njimi pridobimo informacije, kot so položaj, premik, pritisk in spust gumba ter določene poteze pri zaslonu na dotik. Ustrezno zaporedje teh informacij izvede vnaprej predvidevano akcijo kot na primer, če želimo premakniti mapo na namizju osebnega računalnika, se izvede naslednje zaporedje ukazov miške: pritisk levega gumba na določeni poziciji, premik (sprememba pozicije miške) in spust levega gumba na spremenjeni poziciji. Z opisanimi ukazi smo s simuliranjem navideznih ukazov miške poskrbeli za sočasno izvajanje le teh na strani odjemalca in strežnika, ki daje občutek popolne kontrole.

Delo nam je olajšal že implementiran sistem PyMouse. Podporo ima za več platform, kjer za izvajanje simulacije osnovnih ukazov miške (klik, pritisk, premik in spust) na sistemu Unix uporablja knjižico `Xlib` [5]. Za kontrolo miške smo testirali implementacijo dveh različnih načinov. Pri prvem načinu smo izkoristili zgolj klik različnih gumbov miške na določeni poziciji, pri drugem pa premik, pritisk in spust gumbov miške. Prikaz testa prvega načina

se je izkazal kot delno pravilen način delovanja implementiranega sistema. Z njim smo lahko dalje nadgradili sistem, ki se skuša približati prikazu uporabi oddaljenega nadzora v realnem času. Tako je drugi način postal končna oblika razvoja le-tega.

Eden od razlogov za uporabo PyMouse kot simulatorju navidezne miške je že naveden zgoraj, drugi izmed njih je preprosta in hitra uporaba kombinacij funkcij, ki te pripelje do želenega cilja.

### 3.2.4 Navidezna tipkovnica virtkey

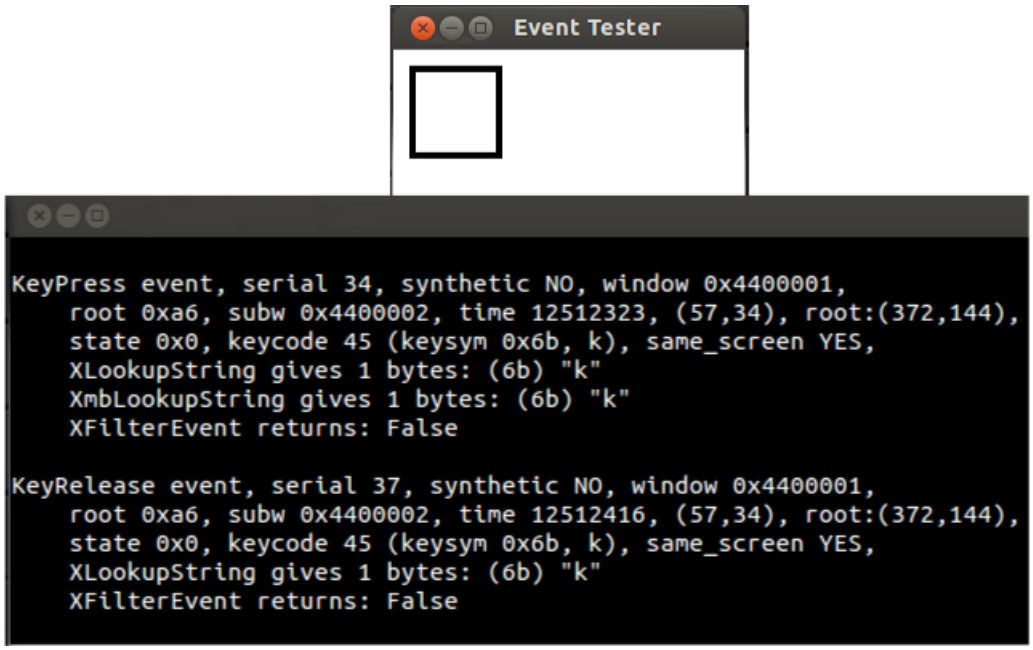
Občutek popolnega nadzora oddaljenega namizja doda tudi simulacija navidezne tipkovnice. V primerjavi s simulacijo miškinih ukazov je ta malo bolj zapletena, saj sta na miški zgolj dva ali trije gumbi. Tipkovnice imajo tipke, ki jo naredijo posebno, kot so na primer šumniki na slovenski tipkovnici. Zaradi kompatibilnosti odjemalca, problem je primerno opisan v poglavju 4.3.2, smo določeno podporo tipk izpustili. Ustrezno simulacijo tipk izvaja že implementiran sistem virtkey. S pomočjo ukaza `xev` [9] smo rešili problem izvajanja določenih tipk, saj jih drugače ni bilo možno uporabiti.

Simulacija navidezne tipkovnice virtkey rešuje z uporabo dodatka Xtest [3] za strežnik X. Virtkey dejansko ponudi poln nabor funkcij, kjer lahko izvedemo simulacijo pisanja alfa-numeričnih znakov, tipk za modifikacijo in sistemskih ukazov. Za naše potrebe izvajanja teh smo priredili skripto glede na vnos odjemalca. Manjši problem se je pojavil pri sistemskih ukazih, saj jih je bilo potrebno posebej definirati. Na našo srečo obstaja že prej omenjeni ukaz `xev`, kjer lahko z njegovo uporabo ujamemo dogodke miške in tipkovnice. Tako smo lahko prevzeli kodo, ki je privzeta za določene tipke tipkovnice in jih definirali za ustrezno uporabo.

Dodatno preurejanje skripte za simuliranje tipk za modifikacijo ni bilo potrebno, saj so ti že definirani v sistemski knjižici `X11/X.h` in v uporabljeni implementaciji. Za modifikacijo lahko uporabimo dvigalko (Shift), krmilko (Control) in izmenjalko (Alter). Virtkey za simulacijo alfa-numeričnih znakov ponuja uporabo že prej omenjenih privzetih sistemskih kod ali nam bolj

prijazno in znano kodiranje znakov Unicode. Izbrali smo slednjega, saj lahko enostavno prejete znake pretvorimo v znak Unicode in jih dalje uporabimo za izvedbo simulacije.

Za simulacijo navidezne tipkovnice je bila uporaba implementiranega sistema virtkey bolj kot ne edina enostavna izbira. Ostale rešitve za simulacijo so zahtevale dodatno nameščanje orodij, pri virtkey pa zgolj vpeljavo v Python. V kolikor je bila to pozitivna lastnost navidezne tipkovnice, je negativna povzročila manjše preglavice glede izredno pomanjkljivo opisane dokumentacije. Zgolj podrobno testiranje implementiranega sistema je rešilo zgoraj navedene težave.




Slika 3.3: Primer uporabe orodja xev pri pritisku tipke “k”.

## 3.3 Spletno programiranje

HTML5, Javascript [6] in spletni brskalniki. To so elementi prihodnosti, ki omogočajo neskončno možnost razvoja spletnega programiranja. Za našo

nalogo smo uporabili te elemente za implementacijo odjemalca oddaljenega namizja.

### 3.3.1 HTML5

HTML5 je peta generacija razvoja označevalnega jezika HTML, ki skrbi za prikaz spletnih vsebin že od leta 1993, ko so HTML standardizirali. Za prirejanje vsebine spletnih strani skrbijo značke, ki se pojavijo med lomljeni oklepaji ali drugače povedano med < >. Po navadi so značke v parih (začetni  in končni ), lahko pa so tudi prazni elementi (<br>). Nova generacija je prinesla nove značke in tehnologijo, ki popestrijo prikaz in uporabnost spletnih strani. S tem so približali možnost razvoja dokaj kompleksnih aplikacij in pokazali svetu programiranja, da je vse mogoče. Tu so predvsem povečali funkcionalnost skriptnega jezika Javascript z dodatki za lažjo ustvarjanje in manipulacijo, ki poveča možnost dinamičnosti spletnih strani.

Pri pregledu zgodovine lahko zasledimo popularnost kreiranja dinamičnih spletnih strani in željo povezave s spletnim strežnikom preko skupine tehnologij, imenovane Ajax (asinhroni Javascript in XML). Skupino sestavljajo elementi HTML in CSS za predstavitev strani, DOM za dinamični prikaz in interakcijo s podatki, XML za izmenjano podatkov, XSLT za manipulacijo podatkov, objekt XMLHttpRequest za asinhrono komunikacijo in Javascript, ki združi vse to skupaj. Od samega začetka nastanka skupine Ajax, je bilo ogromno dodatnih tehnologij uporabljenih v njenih aplikacijah. Kljub imenu lahko namesto XML uporabimo drug način izmenjave podatkov. Med njimi toplo priporočajo JSON, lahko uporabimo tudi navaden tekst ali formatiran HTML. XML in JSON imata enak namen uporabe za lažje razumevanje in predstavitev podatkov s strani računalnika ter človeka. JSON je v primerjavi z XML enostavnejši za branje in pisanje, preprostejši, bolje razumljiv, računalnik ga lažje procesira ter zapis je že vgrajen v določen programski jezik. Izbira med njima je odvisna od tega, kje ga rabimo. Uporaba XML je primernejša za izmenjavo dokumentov, JSON pa za izmenjavo podatkov.

Popularnost uporabe Ajax je še vedno velika, saj novejše tehnologije še nimajo tolikšne podpore, kot jo ima ta, a počasi narašča. V kolikor bi bilo potrebno, bi za naše potrebe lahko uporabili tudi Ajax. S pomočjo JSON bi ustrezno priredili informacije o dogodkih in posodobitev zaslona. Uporaba nove tehnologije je zadovoljila naše zahteve, zato ni bilo potrebno dodatne implementacije prej omenjene skupine Ajax. Omenjena nova tehnologija je značka canvas in način komuniciranja med strežnikom in odjemalcem, imenovana WebSockets.

Slika 3.4: Primerjava zapisa istih podatkov z XML in JSON.

### Canvas

Element oziroma značka canvas je zanimiv pristop pri reševanju določenih ovir glede dodajanja vsebin, kot so slike ali s pomočjo skriptnih jezikov pri obdelavi objektov. Po pregledu določenih primerov, ki so uspešno uporabili kombinacijo te značke in skriptnega jezika Javascript, lahko opazimo dejanski napredek uporabnosti tehnologije. Vse skupaj so postavili na novo raven: od simulatorjev fizike, osnovne in zapletene animacije, prikaza in manipulacije slikovnih galerij, tudi možnost igranja iger do uporabe značke kot

risalne plošče. Razširitev je ogromno in vse to je navdih za nadaljnjo uporabo te kombinacije. Omenjeni skriptni jezik omogoča dodajanje poslušalca dogodkov miške, tipkovnice in za dotik na ta element. Tako lahko razširimo funkcionalnost tega elementa po naših želja.

Pri naši nalogi ni bilo potrebe po napredni uporabi elementa, smo pa s pomočjo prej omenjene kombinacije in v naslednjem odstavku opisane komunikacije uspešno implementirali prikaz posodobljenega zaslona oddaljenega namizja.

### WebSockets

Tehnologija za prenos strežniških informacij k odjemalcu je v uporabi že kar nekaj časa. Ta se po stilu komunikacije imenuje "push" (potisniti), saj strežnik odda zahtevo za prenos. Ajax uporablja eno izmed teh tehnologij imenovano "long polling" (pozivanje). Značilnost le-te je navidezna uporaba stila komunikacije "push". Navidezna je zaradi tega, ker odjemalec prične komunikacijo s strežnikom. Komunikacija ali HTTP povezava je odprta, dokler strežnik ne pošlje odgovora oziroma ustrezno pripravljenih podatkov. Vse tehnike pri stilu komunikacije "push" delujejo kar dobro, kot primer lahko vzamemo klepetalnico v Gmailu.

Pri komunikaciji "push" se pojavi problem, v kolikor želimo komunikacijo z minimalnim odzivnim časom. Ta se lahko reši z uporabo novega načina komuniciranja med strežnikom in odjemalcem, tehnologijo WebSockets. Uporaba te tehnologije odpira vrata razširitvam, ki jim HTTP in Ajax nista kos. Njen namen je omogočiti polno dvojno in konstantno povezavo z minimalnim odzivnim časom. Med sabo si strežnik in odjemalec lahko pošiljata podatke, kadarkoli želita. Ti podatki se pošiljajo preko protokola WebSockets, ki bazira na protokolu TCP. Protokol HTTP se uporabi zgolj pri rokovanju strežnika in odjemalca, kjer pride zahteva za nadgradnjo le-tega na protokol WebSockets. Sama tehnologija je dokaj nova in nima še popolne podpore. Kljub temu je razvoj svetovno znanih brskalnikov v koraku razvoja novih tehnologij in tako je poskrbljeno, da lahko to tehnologijo normalno uporabimo.

Dodatno motivacijo so nam dali dobro dokumentirani primeri implementacij.

Uporaba tehnologije WebSockets se je izkazala za dokaj močno in uporabno orodje. S strani odjemalca smo strežniku pošiljali informacije o dogodkih, ki so jih poslušalci zaznali nad elementom canvas. Ti so bili sprememba pozicije miške, ustrezen pritisk, spust gumba miške in izvajanje pritiskov tipk na tipkovnici. V nasprotno smer je strežnik odjemalcu pošiljal podatke o posodobljeni različici zajetega zaslona. Po prejetju podatkov je odjemalec te ustrezno pripel na element canvas, ki je odjemalcu prikazal posodobljeni zaslon. Uspešen prikaz zaslona je bil pogoj za ponovno zahtevo po posodobitvi zaslona. Tako smo nekako ustvarili zanko s pošiljanjem zahtev odjemalca in podatkov strežnika, ki poskrbi za nemoten, hiter, ustrezen in pristen občutek delovanja dogodkov v realnem času. To je bil tudi eden izmed ciljev te naloge.

### 3.3.2 Javascript

Javasript je standardiziran skriptni jezik, s katerim lahko spletne strani dinamično popestrimo. Pri samem pisanju sintakse lahko opazimo njegovo podobnost programskega jezika C, kjer programska struktura spominja na programski jezik Java. Z razliko od njiju so ukazi krajši in preprostejši.

Ustrezno izkoriščanje tehnologij WebSockets in canvas temelji na uporabi tega skriptnega jezika. Bitko za priljubljenost je bil od njegove pojavitve leta 1995 do 2005. To desetletje je pomenilo zanj polom, saj je ponudil slabo učinkovitost in nenavadno sintakso. To pa je pripeljalo do skrivnostnih napak. Rast priljubljenosti se je pričela pri nastanku skupine tehnologij Ajax leta 2005. Profesionalni razvijalci so tako dobili nov obraz tega jezika. Prav razvijalci so začeli delati na izboljšavi hitrosti tolmačenja in zagona kode ter posvetili pozornost na sam jezik. Razvoj le-tega je tudi prešel na stran strežnika, kjer na primer lahko s pomočjo `Node.js` implementiramo spletni strežnik.

V veliko pomoč so tudi orodja za sprotno pregledovanje izvajanja kode in razne knjižice. Med bolj znane spadata orodje Firebug in knjižica jQuery, ki olajšata delo razvijalcem. S Firebug lahko sledimo klicem komunikacij teh-

nologije Ajax, preverimo trenutno stanje določenih objektov DOM, izvajamo koračno preverjanje pravilnosti izvajanja kode in še mnogo ostalih stvari. Namen jQuery je hitrejše in preprostejše pisanje kode ter enostavnejša uporaba le-te v skriptnem jeziku Javascript. Glede na njegovo priljubljenost in uporabnost bi lahko rekli, da jo že skoraj priporočajo kot nujni dodatek jezika Javascript.

Prej smo omenili, da canvas in WebSockets temeljijo na uporabi Javascript. Seveda lahko tu tudi izkoristimo jQuery, a smo raje prilagodili kodo brez dodatkov. S tem skriptnim jezikom smo implementirali komunikacijo s strežnikom in dodali ustrezne poslušalce miške, tipkovnice in za dotik, ki izvršijo določeno zaporedje ukazov. Poslušalci so prevzeli vlogo privzetih nastavitev vhodnih naprav, ki se po navadi zgodijo nad elementom canvas. Vsak poslušalec je privzeto poimenovan glede na njegov način manipuliranja. Več o samih poslušalcih in njihova implementacija v poglavju 4.3.2. Tako smo implementirali stran odjemalca oddaljenega namizja.

### 3.3.3 Spletni brskalniki

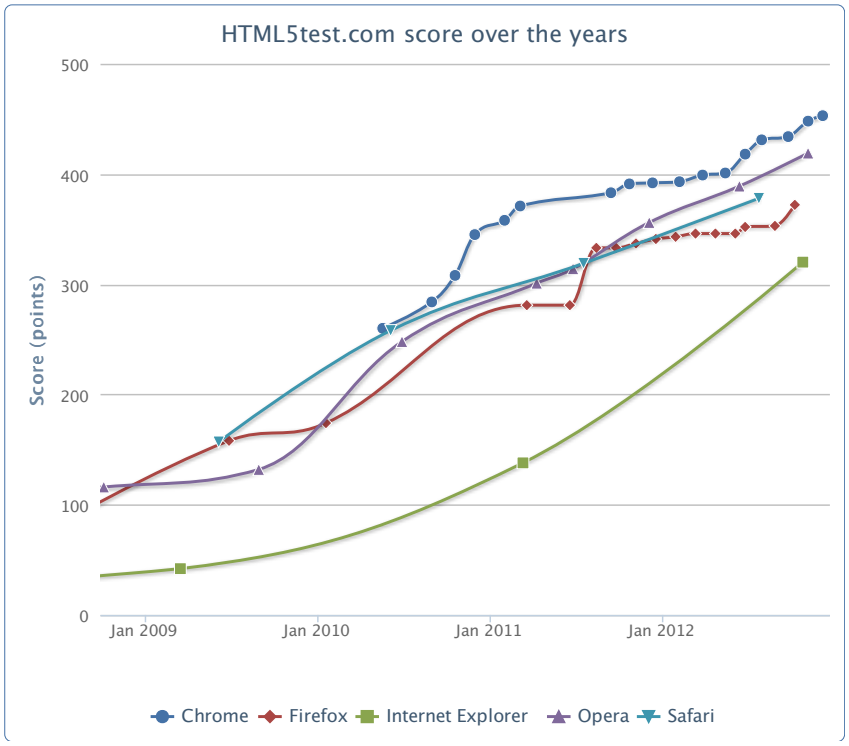
Spletni brskalnik je programska oprema, ki poskrbi za prenos, prikaz in navigacijo informacijskih virov s svetovnega spleta. Ti informacijski viri so predstavljeni kot niz, ki definira vir (definiranje preko URI). Viri so lahko spletne strani, video posnetki, slike ali ostala predstavljena vsebina. S pomočjo aktivne povezave se lahko povežemo do teh virov, ki jih nato spletni brskalnik ustrezno prikaže. To pa ni edina prednost spletnih brskalnikov. Kolikor jih lahko izkoristimo za dostop in prikaz vsebine svetovnega spleta, omogočajo tudi dostop do vsebin spletnih strežnikov ali datotečnega sistema.

Predpona na aktivni povezavi definira, na kakšen način se vir prenese. Najbolj znana predpona je `http`, zaščiten prenos je definiran preko `https`, prenos datotek s `ftp` in prenos lokalnih datotek s `file`. V primeru prenosa ustrezno označevalne vsebine (na primer HTML ali XML) in oblikovalnih podatkov (na primer CSS) se ti predelajo v ustrezno oblikovalno vsebino ter prikažejo na zaslonu uporabnika. Če želimo dodati vsebino (slike, video in

zvočne posnetke) kot del HTML, bo te spletni brskalnik ustrezno predstavil. Za Flash in Java aplikacije so ustrezni vtičniki, ki omogočajo prikaz le-te. V kolikor brskalnik nima podpore za prikaz vsebine, se prikaže vmesnik za shrambo te na lokalni disk. Malo je drugače pri podpori novih tehnologij. Ustrezno lahko uporabnika opozorimo glede nezmožnosti uporabe le-te.

Uporabnost novih tehnologij je lahko problematično, saj sistemi nimajo iste podpore pri uporabi le-teh. Zanimiv problem se pojavi pri mobilnih brskalnikih, saj določeni podpirajo možnost uporabe zgolj na novejših operacijskih sistemih. Drugače je pri osebnih računalnikih, kjer zastarele verzije spletnih brskalnikov ne podpirajo nove tehnologije. V primeru, da želimo omogočiti podporo le-te in biti v koraku s časom razvoja, je nuja posodabljanje brskalnikov in uporaba malenkost dražje in boljše prenosljive naprave.

Različni brskalniki omogočajo različno podporo tehnologije. To lahko opazimo na sliki 3.5, ki ponazarja rast podpore tehnologij HTML5 na svetovno znanih brskalnikih. Za ustrezno upravljanje oddaljenega namizja, je izbira spletnega brskalnika temeljila na njihovem obnašanju pri obremenitveni uporabi le-tega. Dobro se je izkazal brskalnik Google Chrome, tako na namiznem računalniku kot na mobilni napravi. Malo je bilo razočaranja glede uporabe Mozilla Firefox na mobilni napravi, saj je večkrat prišlo do sesutja le-tega še pred testiranjem naloge. Dokaj veliko razočaranje smo doživeli pri testiranju brskalnika Safari na napravah IOS ver. 5.1, kjer ima podporo zgolj za starejše različice komuniciranja preko protokola WebSockets. Z različico 6.0 so to popravili s polno podporo izbranih tehnologij.




Slika 3.5: Prikaz rasti podpore tehnologije HTML5 spletnih brskalnikov. [15]

# Poglavje 4

# Implementacija

V tem delu podamo podroben opis izvedbe implementacije sistema. Poglavje je tako kot poglavje 3 razdeljeno na stran strežnika in odjemalca. Opis implementacije orodij smo obogatili s primeri izvorne kode, konfiguracijo sistema ali slikami. S pomočjo njih podrobno opišemo pot do rešitev ključnih delov za uspešno testiranje končnega prototipa.

## 4.1 Konfiguracija Nginx

Ustrezna uporaba sistema Nginx potrebuje ustrezno konfiguracijo. Tako kot pri večina ostalih sistemov tudi ta poskrbi za pravilno delovanje sistema. Seveda vsako konfiguracijo sistema prilagodimo glede naših želja in potreb, v kolikor sistem to omogoča.

S primerjavo različnih datotek konfiguracije lahko opazimo različno sintakso in stil pisave potrebnih ukazov. Na žalost ni uveljavljenega standarda, ki bi definiral ustrezna pravila pisave. Za učenje in testiranje tako zapravimo kar nekaj časa. Problem se pojavi pri zahtevni in obsežni konfiguraciji, kjer je potrebno več truda. Kot smo omenili v poglavju 3.1, opis sistema Nginx, je konfiguracija dokaj kratka in enostavna. Sestavljena je iz logične strukture, kjer v ustrezne bloke vstavljamo potrebno množico ukazov za delovanje sistema po naših želja. V kolikor želimo, lahko za konfiguracijo strežnika upo-

rabimo ali samo glavno datoteko konfiguracije ali pa si izmislimo datoteko na ustrezni lokaciji datotečnega sistema. Izbira izmed teh dveh določa število vrstic potrebnih ukazov, predvsem osebni pregled nad glavno datoteko ter lokacijo shrambe, saj mogoče želimo imeti datoteko na posebni lokaciji.

Omenili smo logično strukturo, bloke in ukaze. Ukazi so definirani po pomenu in takšna je tudi njihova funkcionalnost v sistemu. Z gnezdenjem blokov poskrbimo za ustrezno logično strukturo.

```
1  http{
2    server{
3      listen 80;
4      server_name localhost;
5      access_log log/access.log;
6      error_log  log/error.log;
7      location / {
8        index index.html;
9    }
10   }
11 }
```

Listing 4.1: Primer logične strukture strežnika.

Zgornji primer 4.1 prikaže strukturo konfiguracije enega strežnika. Vidimo lahko, da se uporabi blok `http`, ki je tudi način povezave. V bloku `http` je lahko več blokov `server`, kar pomeni več konfiguracij strežnikov hkrati. Vsakemu strežniku ustrezno priredimo vrata za sprejem zahtev z `listen`, gostiteljevo ime s `server_name`, po želji izpisujemo podatke komunikacije `access_log` in napak `error_log` v datoteko ter podamo dejansko lokacijo statičnih datotek z `location`. V tem primeru se lokacija statične datoteke `index.html` nahaja tam, kjer se je Nginx namestil, natančneje dalje v mapi `html`. Ravno tako mapa, kjer hranimo izpis `log` se nahaja v namestitveni datoteki. Če želimo upravljati z več spletnih strani na enem strežniku, lahko dodamo datoteke v ustrezno mapo in podamo lokacijo teh. Na primer, da želimo zgornjem primeru dodati par statičnih datotek. Lokacija teh je shranjena na absolutni poti `/home/upo/test`, kjer je `test.html`

dejansko `index` datoteka. Pogoj je še dostop do teh datotek preko spletnega naslova `http://localhost/test/` . Vse, kar moramo dodati, je ukaz `root`. Ta poskrbi, da zahteva z danim naslovom servira `test.html` iz navedene mape. Ustrezen primer je podan spodaj, 4.2.

```
1  location /test/ {
2    root /home/upo;
3    index test.html;
4  }
```

Listing 4.2: Primer serviranja statičnih datotek.

Če ta primer podrobno pregledamo, hitro opazimo, da ni podane popolne absolutne poti. Posebnost `root` je, da sestavi podano pot in uporabljeno lokacijo spletnega naslova. To je uporabno, ko imamo naslov in mapo istega imena. V kolikor imamo različna imena, pa namesto `root` uporabimo ukaz `alias`, kjer podamo polno absolutno pot.

Modul TCP, ki smo ga vključili pred namestitvijo Nginx, omogoča ustrezno preusmeritev prometa povezave TCP na naslov spletnega strežnika. Odlično reši težave pri uporabi komunikacije WebSockets, a na žalost ne moremo ločiti prihajajoči promet istih vrat na HTTP in TCP. Za to je potrebna uporaba različnih vrat za sprejem zahtev. Z modulom dobimo tudi dodatne ukaze, kjer izmed teh določene izkoristimo za ustrezno testiranje. Kot smo že omenili, lahko za konfiguracijo TCP tudi uporabimo posebno datoteko s konfiguracijo le-tega. Za naše potrebe ni bilo potrebnih veliko vrstic in zaradi tega smo se odločili, da bomo združili vse skupaj v glavno datoteko.

Za ustrezno uporabo moramo, podobno kot pri primeru 4.1, dodati glavni blok. Tu namesto `http` uporabimo `tcp` blok. Vanj dodamo blok `upstream`, ki poskrbi za ustrezno in preprosto porazdelitev do spletnih strežnikov. Blok `upstream` je potrebno ustrezno poimenovati za nadaljnjo uporabo. Strežniki v tem bloku so napisani z naslovom in, v kolikor je potrebno, z ustreznimi vrati. V našem primeru je potrebno dodati vrata, ki smo jih uporabili pri zadnjem delu implementacije spletnega strežnika Tornado, kodiranje 4.14 na strani 46 . Dodatno smo želeli preverjati stanje teh strežnikov, zato je do-

dan modul `check` v bloku `upstream`. Modulu `check` smo dodali potrebne parametre za izvajanje preverjanja. Parametri so časovno obdobje preverjanja, števci povezljivosti in časovni zamik preverjanja. Števci povezljivosti določajo stanje strežnikov, kjer po številu preverjanj označijo stanje le-teh.

V blok `tcp` smo dodali še blok za strežnik, saj ga je potrebno definirati. Podobno kot v primeru 4.1, na strani 28 smo za sprejem zahtev uporabili standardna vrata `80` in ime strežnika `localhost`. Po želji bi lahko namesto imena strežnika uporabili IP naslov tega ter ga združili z vrati pod modul `listen`. V blok `server` dodamo še dve funkcionalnosti. Oba sta ukaza. Prvi je `tcp_nodelay`. Potrdilno vrednost tega ukaza podamo v primeru aplikacij, ki konstantno pošiljajo krajša sporočila. Drugi se imenuje `proxy_pass`, kjer podamo ime uporabljenega bloka `upstream`. Ta ukaz poskrbi za uspešno posredovanje informacij napisanih strežnikov.

```
1  tcp{
2    upstream websockets {
3      # streznika, prvi od Tornado, drugi testni
4      server 127.0.0.1:8888;
5      server 127.0.0.1:8887;
6      check interval=3000 rise=2 fall=5 timeout=1000;
7      }
8    server{
9      # listen in server_name kot pri primeru 4.1
10     tcp_nodelay on;
11     proxy_pass websockets;
12     }
13 }
```

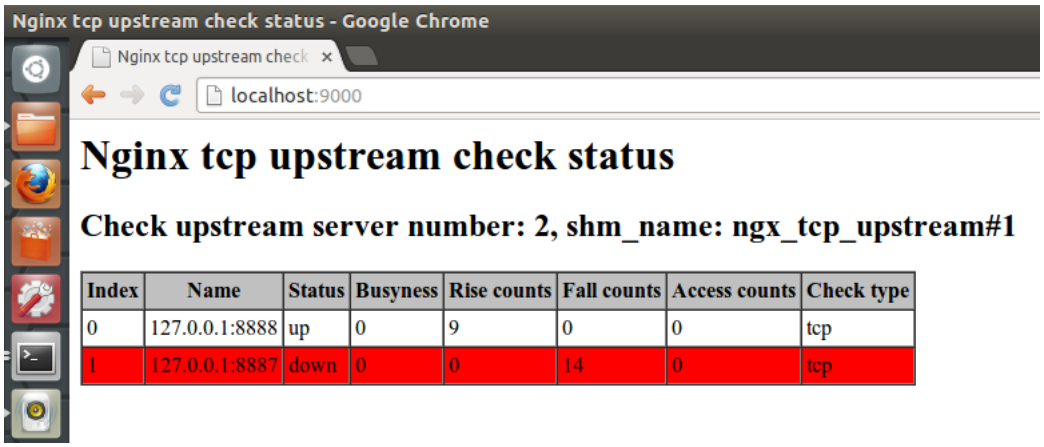
Listing 4.3: Konfiguracija Nginx.

Preverjanje statusa povezljivosti spletnih strežnikov smo uredili preko bloka `http`, kjer dalje v bloku `server` prejema zahteve preko vrat `9000`. Modul TCP omogoča preverjanje le-teh z ukazom `check_status`. Ta ukaz podamo v blok `location`, ki s pomočjo tega servira spletno stran za preverjanje.

```
1 http{
2   server{
3     listen 9000;
4     location / {
5       check_status;
6     }
7   }
8 }
```

Listing 4.4: Preverjanje povezljivosti v konfiguraciji.

V kolikor pogledamo prikaz konfiguracije Nginx, 4.3 lahko v bloku `server` vidimo dva strežnika. Za prikaz povezljivosti smo dodali še testni strežnik, ki ni implementiran in je tako nedosegljiv. Prikaz statusa prikaže spodnja slika 4.1, kjer lahko po atributu `Rise` oziroma `Fall` predvidevamo status strežnikov.




Slika 4.1: Preverjanje povezljivosti preko brskalnika.

Pri združitvi podanih konfiguracij Nginx 4.3 in 4.4 lahko kaj kmalu ugotovimo, da je konfiguracija sistema res enostavna in dokaj hitra. Opozoriti je potrebno, da smo prikazali zgolj tiste dele konfiguracije, ki je nuja za pravilno delovanje te naloge. Dodatni ukazi in tudi primeri so že podani v prvotni verziji glavne datoteke konfiguracije Nginx.

## 4.2 Programiranje v Python

Za naše potrebe smo implementirali vsak del funkcionalnosti strežnika posebej kot modul in nato te uvozili v implementacijo strežnika, saj je tako boljša preglednost le-tega. To lahko dokaj enostavno rešimo tako kot ostale potrebne funkcionalnosti modulov za prepoznavo njenih funkcij z ukazom `import <ime datoteke>`. Pogoj je, da shranimo datoteke s funkcijami na lokacijo implementiranega strežnika, saj tako lahko strežnik enostavneje dostopa do njih. V teh datotekah so posamezne funkcionalnosti napisane kot sekvenca ukazov v funkcijah, kjer te enostavno definiramo z ukazom `def <ime funkcije>`. Strežnik nato izvede to sekvenco ukazov preko kombinacije ukazov `<ime datoteke>.<ime funkcije>(argumenti)`. To imenujemo klic funkcije, kjer po potrebi pošljemo argumente za obdelavo.

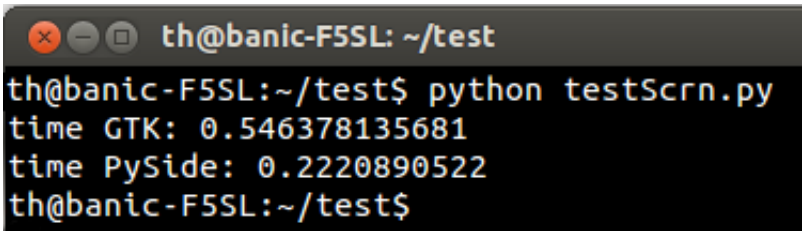
Razdelitev funkcionalnosti strežnika nam je pomagala pri razbitju programa na podprograme, lažjo implementacijo ter, kot že prej omenjeno, boljšo preglednost teh. Omeniti je potrebno še uporabo lovljenja izjem, saj se je pri testiranju večkrat pripetila nepričakovana napaka. Za ustrezno uporabo te moramo v blok `try` podati ukaze, ki lahko povzročijo napake. Izpis ali primerno rešitev podamo po podanih ukazih v blok `except`. V kolikor želimo, lahko lovimo prav specifične izjeme. Za primer vzamemo deljenje z `0`. Bloku `except` dodamo še ime napake, kjer v našem primeru to skupaj pride `except ZeroDivisionError`. Z uporabo blokov lovljenja izjem smo se rešili vseh problemov ter omogočili tekoče delovanje spletnega strežnika.

### 4.2.1 Zajem zaslona z uporabo PySide

V poglavju 3.2.2 smo omenili, da je ključnega pomena za izvajanje oddaljenega namizja v realnem času potrebna ustrezna implementacija sistema zajema zaslona. Opisali smo še način testiranja dveh različnih implementacij sistema GTK+ GDK ter PySide.

Testiranje je dejansko potekalo tako, da smo uvozili modula obeh programov ter nato posebej časovno izmerili izvedbo klica funkcije modulov. Naloga

obeh funkcij je bila ista, z ustreznimi funkcijami narediti zajem zaslona ter shraniti sliko v format JPEG. Tako smo lahko izvedli časovni test, kjer je bil po rezultatih PySide dokaj jasno primeren kandidat za nadaljnjo izvajanje zajema zaslona.




Slika 4.2: Časovni test izvajanja.

Izbira ustreznega formata je izboljšala hitrost zajema in pošiljanja kodirane slike odjemalcu, kjer je bila na strani odjemalca tudi zadovoljiva kvaliteta te slike.

### QtGui in QtCore

Za ustrezni zajem je bila potrebna pravilna kombinacija izvajanja funkcij PySide. Na voljo ima dva glavna modula `QtGui` in `QtCore`. Že poimenovanje dejansko definira njuno funkcionalnost. Modul `QtGui` poskrbi za gradnjo grafičnega dela uporabniškega vmesnika, modul `QtCore` se pa uporabi za ustrezen nadzor le-tega. Na primer, želimo implementirati osnovni grafični vmesnik, ki ga zaključimo s tipko "ESC" na tipkovnici. Z orodji modula `QtGui` ustrezno zgradimo vmesnik ter vanj implementiramo lovljenje dogodkov tipkovnice. Tu preverimo, ali je dogodek pritiska tipke na tipkovnici enak že definiranemu objektu tipke "ESC" v modulu `QtCore`. V primeru pozitivnega pogoja se izvede zaključitev aplikacije.

Na splošno se je modul `QtCore` uporabljal za povezovanje objektov grafičnega vmesnika in ustreznih akcij le-teh preko signalov. Pri novejših verzijah so to posplošili in poenostavili, saj so definirani v samih objektih s primernimi signali. Ti so ustrezno poimenovani, kot na primer pri gumbu najbolj upo-

raben signal `clicked`. Signale nato povežemo s funkcijo, ki izvede določeno akcijo. Kljub temu lahko še vedno uporabimo star način uporabe signalov.

### Izbira in opis funkcionalnosti

Začetek gradnje vseh vmesnikov se prične z inicializacijo instance modula. V primeru, da je to grafični vmesnik, se uporabi razred `QtGui.QApplication`, drugače `QtCore.QCoreApplication`. Ne glede na izbiro je to glavna Qt aplikacija in ker Qt lahko prejme argumente iz ukazne vrstice, je te potrebno posredovati izbranemu objektu. Kljub temu da ne gradimo grafičnega uporabniškega vmesnika, smo uporabili modul `QtGui`.

Njegova funkcionalnost dveh določenih razredov ter dalje njunih funkcij omogoča ustrezno izrabo sistema za zajem zaslona. Za pravilno delovanje sistema je potrebna kombinacija funkcij razreda `QPixmap`, `QApplication` in `QWidget`.

QPixmap je zasnovan in optimiziran za ustrezen prikaz slik na zaslonu, ki so lahko uporabne tudi za dodajanje na primerne objekte. Omogoča nekaj načinov ustreznega hranjenja slik, dva izmed njih sta na primer pri ustvarjenju objekta `QPixmap` ter preko funkcij za nalaganje (iz podatkov ali navadnih slikovnih datotek). QApplication vsebuje glavno dogodkovno zanko, ki skrbi za procesiranje dogodkov vhodnih naprav ter ostalih virov. Kot smo že napisali, upravlja z inicializacijo in zaključkom aplikacij ter njihovimi in sistemskimi nastavitvami. QWidget je osnovni razred za vse gradnike uporabniškega vmesnika. Gradniki so glavni elementi vmesnikov, saj sprejemajo dogodke vhodnih in drugih naprav ter jih ustrezno izvršijo na zaslonu. Razred `QWidget` ima ogromno funkcionalnosti, kjer pa jih veliko primerno lahko uporabimo zgolj v podrazredih.

### Zajem zaslona

Za zajem smo uporabili ustvarjenje objekta `QPixmap` preko statične funkcije `grabWindow()` . V to funkcijo je potrebno kot argument podati identifikator okna, saj tako “poslika” vsebino le-tega. Identifikator okna dobimo s

pomočjo sestave funkcij `desktop()`, razred `QApplication`, in `winId()`, razred `QWidget`. Funkcija `desktop()` dejansko vrne razred gradnika, to je `QDesktopWidget`. Ta gradnik lahko pridobi dostop do ustreznih podatkov namizja uporabnika. Ker je ta gradnik podrazred razreda `QWidget`, podeduje sistemsko funkcionalnost `winId()` . Tako lahko uporabimo funkcijo `winId()` za gradnik `QDesktopWidget` in dobimo potreben identifikator letega oziroma nekako preprosteje povedano na koncu pridobimo informacije namizja. Ustvarjen objekt `QPixmap` shranimo v željen JPEG format in tako je zaključen sistem zajema zaslona.

Testiranje sistema je opozorilo na nepravilno delovanje le-tega. Problem se je pojavil pri testiranju ciklične uporabe sistema, kjer je prišlo do napake `RuntimeError`. Napaka se pojavi zaradi že ustvarjene instance razreda `QApplication` in sistem ni dovolil ustvaritve nove, saj sme obstajati zgolj ena. Problem smo rešili z implementirano funkcijo `instance()` tega razreda, kjer lahko enostavno uporabimo že ustvarjeno instanco. Zagato glede uporabe te je rešil blok lovljenja izjem. Primer kodiranja je podan spodaj, 4.5.

```
1  # uvoz modulov
2  import sys
3  from PySide.Gui import QApplication, QPixmap
4
5  def zajem():
6    # zacetek instance aplikacije
7    try:
8      app = QApplication(sys.argv)
9    except RuntimeError:
10     app = QApplication.instance()
11
12   # izvedba zajema in shranitev slike v datoteko
13   pic = QPixmap.grabWindow( QApplication.desktop().winId() )
14   pic.save('zaslon.jpeg', 'jpeg')
```

Listing 4.5: Primer implementacije zajema zaslona.

Uporaba PySide se je izkazala za pravilno odločitev, saj je rešila enega izmed glavnih delov naše naloge, način zajema zaslona. Več o ustreznem

posodabljanju le-tega sledi v poglavju 4.3.2.

### 4.2.2 Uporaba navidezne miške PyMouse

Uporaba sistema PyMouse je pripomogla k hitremu in konkretnemu izvajanju kontrole navidezne miške. Podporo nudi za platforme Unix, Macintosh, Windows ter Java. Knjižice, uporabljene za ta namen, so: `Xlib` za Unix, `Quartz` in `AppKit` za Macintosh, `win32api` in `win32con` pri Windows ter `java.awt` za platformo Java. Zaradi uporabe operacijskega sistema Linux Ubuntu, ki bazira na sistemu Unix, bomo malo podrobneje pogledali način izrabe funkcij knjižice `Xlib`.

Xlib je programski aplikacijski vmesnik za sistem X. S pomočjo Xlib imamo grafično sposobnost za sistemsko okensko programiranje. Izkoristimo lahko tudi podporo naprav, kot sta miška in tipkovnica. Po navadi to knjižico ne uporabljamo direktno. Dostop do nje imajo predvsem razna orodja za gradnjo grafičnih vmesnikov. Primer takih orodij so Qt, GTK+ ali na primer Motif.

PyMouse uporabi elemente `desktop`, `fake_input` in `X` za primerno implementacijo navidezne miške. S pomočjo posameznih elementov lahko pridobimo pozicijo miške in informacije o zaslonu ter izvedemo premik, pritisk in spust gumbov miške. V primeru 4.6 lahko vidimo uporabnost opisanih modulov knjižice Xlib, kjer se izvede premik miške na ustrezno pozicijo ter klik gumba le-te.

```
1  # uvoz potrebnih modulov knjizice Xlib za test miske
2  from Xlib.display import Display
3  from Xlib import X
4  from Xlib.ext.xtest import fake_input
5
6  # pozicija in gumb miske = 1 leva, 2 kolesce, 3 desna
7  def miska(x, y, gumb):
8      zaslon = Display()
9      # premik
10     fake_input(zaslon, X.MotionNotify, x = x, y = y)
```

```
11      fake_input(zaslon, X.ButtonPress, gumb)    # pritisk
12      fake_input(zaslon, X.ButtonRelease, gumb) # spust
13      zaslon.sync() # posodobitev zaslona
```

Listing 4.6: Primer implementacije navidezne miške.

V samem primeru 4.6 lahko opazimo, da je izvedba navideznega premika in klika miške dokaj enostavna. Na primer, da želimo na poziciji `x=10` in `y=10` pritisniti na levi gumb miške. To lahko sedaj storimo s preprostim ukazom `miska(10,10,1)`. S kodo smo opisali prejem informacij zaslona ter tri glavne funkcije miške. Tu načeloma ni pravilna uporaba posodobitve zaslona. Posodobitev je potrebna po vsakem koraku `fake_input`, ker edino tako poskrbimo za pravilen prikaz treh izvedenih akcij miške na zaslonu.

PyMouse posploši uporabo Xlib s preprostimi ukazi. Ti ukazi izvedejo kombinacijo premika miške in pritiska ali spusta gumba. Premik je že definiran v pritisku in spustu gumba miške. Izvedemo lahko tudi klik gumba miške, kjer je kombinacija pritiska in spusta le-tega. Spodnji primer 4.7 prikaže te preproste ukaze.

```
1 from pymouse import PyMouse
2 def miska(x, y, gumb):
3   m = PyMouse()           # inicializacija miske
4   m.press(x, y, gumb)     # pritisk gumba miske
5   m.release(x, y, gumb) # spust gumba miske
6 # ali kar kombinacija vseh ukazov
7   m.click(x, y, gumb)
8 # sam premik miske
9   m.move(x, y)
```

Listing 4.7: Primer ukazov PyMouse.

Za našo uporabo smo implementirali tri glavne ukaze navidezne miške (premik, pritisk in spust gumba). Kombinacije vseh (ukaz `click`) ni bilo mogoče uporabiti, saj tako ni bilo podpore določenih funkcionalnosti pri kontroli miške na oddaljenem namizju. Tu je predvsem mišljen premik objektov na drugo lokacijo. Zato smo z ustrezno implementacijo poskrbeli za dokaj pristen občutek izvajanja oddaljenega namizja v realnem času.

### 4.2.3 Uporaba navidezne tipkovnice virtkey

Tako kot miško je bilo potrebno implementirati simulacijo izvajanja navidezne tipkovnice. Uporabimo lahko že implementiran sistem navidezne tipkovnice virtkey. Z razliko od PyMouse je ta napisan v programskem jeziku C. Za ustrezno uporabo si sistem, tako kot pri miški, pomaga s knjižico `Xlib`. Podobnost uporabe kode Xlib med primerom miške 4.6 in tipkovnice 4.8 je dokaj očitna.

```
1  # uvoz istih modulov kot v primeru kode miske 4.6
2  def klikTipke(koda):
3    zaslon = Display()
4    fake_input(zaslon, X.KeyPress, koda)     # pritisk ''a''
5    fake_input(zaslon, X.KeyRelease, koda)   # spust ''a''
6    zaslon.sync() # osvezevanje zaslona
7
8  if __name__ == '__main__':
9    # koda 38, definirana kot tipka ''a''
10   klikTipke(38)
```

Listing 4.8: Primer implementacije navidezne tipkovnice.

Velika razlika med njima je kompleksnost simuliranja navidezne naprave. Simulacija miške zahteva zgolj poznavanje vrstnega reda treh tipk. Pri simulaciji tipkovnice pa je malo drugače. To lahko opazimo že v vrstici 9 pri zgornjem primeru 4.8, kjer smo zahtevali simulacijo tipke "a" na navidezni tipkovnici. Kolikor lahko vidimo, je potrebna nekakšna koda te tipke. V poglavju 4.3.2 smo tudi omenili pridobivanje ustreznih kod tipk preko dogodkov tipkovnice. Se pravi, da na strani odjemalca dobimo ustrezno kodo tipke ter na strani strežnika izvedemo simulacijo tipk preko njene kode. Tu se pojavi problem, saj sta kodi različnih standardov. Na strani odjemalca je standard ASCII oziroma prirejen brskalniku ter na strani strežnika pa Xlib. Tako lahko vidimo, da ni možna direktna uporaba le-teh. Problem se lahko delno reši tako, da spremenimo način pošiljanja dogodkov tipkovnice odjemalca ter na strežniku dodamo funkcije, ki te dogodke ustrezno predelajo.

Pri odjemalcu spremenimo kodo tipk v ustrezen znak. Se pravi, pošiljamo dejansko vrednost tipk. Strežnik tako sprejme te vrednosti ter jih s pomočjo dveh dodatnih funkcij predela v ustrezno kodo za izvršitev simulacije. Prva funkcija spremeni znak v število, ki predstavlja simbol tipke na tipkovnici, druga pa to število simbola predela v kodo, ki predstavlja fizični naslov tipke. Simbol in naslov smo že spoznali pri opisu orodja `xev`, stran 18, kjer se prvi imenuje `keysym`, drugi pa `keycode`. Ustrezna sprememba uporabe primera implementacije 4.8 je opisana spodaj, 4.9.

```
1  # uvoz istih modulov kot v primeru kode miske 4.6
2  from Xlib import XK # dodamo se en modul za pretvorbo znakov
3
4  def klikTipke(niz):
5    zaslon = Display()
6  # niz spremenimo v simbol keysym
7    koda = XK.string_to_keysym(niz)
8  # nato v ustrezen naslov kode za fake_input
9    koda = zaslon.keysym_to_keycode(koda)
10   ... # nadaljujemo s fake_input, kot pri primeru 4.8
11
12 if __name__ == '__main__':
13   klikTipke('a')  # primer izvedbe simuliranja znaka
```

Listing 4.9: Spremenjen primer implementacije navidezne tipkovnice.

Opisali smo kako lahko dokaj enostavno simuliramo tipkanje znakov. Implementiran sistem virtkey je nadgradnja teh primerov, saj uporabi njeno funkcionalnost in ponuja še več. Zaradi tega smo tudi uporabili ta sistem.

Za našo uporabo smo prevzeli funkcije pritiska tipk ter zaklepa in odklepa tipk za modifikacijo znakov. Rahel problem se je pojavil pri prenosu ukaznih tipk do strežnika. Za te tipke smo strežniku pošiljali ASCII kode, kjer smo jim preko orodja `xev` že prej priredili vrednosti `keycode`. Za izvedbo pritiska in spusta alfa-numeričnih ter funkcijskih tipk sta poskrbeli funkciji `press_unicode`, `release_unicode` in `press_keycode`, `release_keycode`. Prvi način smo uporabili pri pošiljanju predelanih ASCII kod v razumljive vrednosti, drugi pa, tako kot že prej omenjeno, pri pošiljanju dejanskih ASCII

kod. Pri tipkah za modifikacijo je možnost uporabe funkcije `lock_mod` in `unlock_mod` . Te so definirane s posebnimi vrednostmi v sistemski knjižici `X11/X.h` : `1<<0` kot dvigalka, `1<<2` kot krmilke in `1<<3` kot izmenjalka. Spodnji primer 4.10 prikaže enostaven primer kombinirane uporabe ukazov. Funkcija `ord` je že vgrajena v Python in spremeni znak v število Unicode. Spodaj lahko vidimo primer 4.10 kako opisane funkcije uporabimo.

```
1 # uvozimo modul virtkey
2 import virtkey
3 if __name__ == '__main__':
4   v=virtkey.virtkey()
5 # zelimo izpis crke ''a'' preko njenega naslova
6   v.press_keycode(38)
7   v.release_keycode(38)
8 # zelimo izpis z znakom crke ''A''
9   v.lock_mod(1<<0)     # pritisnemo dvigalko
10  v.press_unicode(ord('a'))
11  v.release_unicode(ord('a'))
12  v.unlock_mod(1<<0)  # spustimo dvigalko
```

Listing 4.10: Primer uporabe navidezne tipkovnice virtkey.

Opazimo lahko, da smo uporabili dokaj enostaven sistem navidezne tipkovnice. Poskrbeti moramo za ustrezen spust tipk, saj se lahko kaj hitro pripeti kakšna težava. S sistemom virtkey smo tako poskrbeli za vse tipke, ki smo jih želeli simulirati.

### 4.2.4 Združitev in uporaba spletnega strežnika Tornado

Glavni del naloge je bila ustrezna implementacija spletnega strežnika ter vpeljava vseh potrebnih modulov za pravilno uporabo oddaljenega nadzora. Za naše potrebe spletni strežnik Tornado poskrbi glede komunikacije preko WebSockets, uporabe varne povezave ter ustrezne prijave odjemalca.

### Rokovalnik zahtev

Implementacija spletnega strežnika zahteva od programerja gradnjo ustreznih rokovalnikov. Rokovalnik primerno poskrbi glede odjemalčeve zahteve ter mu po potrebi posreduje ustrezen odgovor. Primer uporabe rokovalnika je iskalnik z obrazcem. Tu uporabnik vpiše vse potrebne kriterije, po katerih želi iskati. Kriterije nato strežnik obdela, uredi odgovor ter ga posreduje uporabniku. Glede na obdelavo in urejanja je lahko odgovor pozitiven ali negativen, vse je odvisno od kriterijev. Podobno opisan primer je prikazan spodaj, 4.11, kjer je enostavna implementacija rokovalnika s vnosno formo ter odziv glede posredovanje vsebine le-te.

```
1  # uvozimo potrebne module streznika
2  import tornado.ioloop
3  import tornado.web
4  import tornado.httpserver
5
6  # rokovalnik, ki poskrbi za zahteve odjemalca
7  class obrazec(tornado.web.RequestHandler):
8    # zacetek komunikacije se pricne s prikazom spletnega obrazca
9    def get(self):
10     self.write('<form action="/" method="post">'
11                'Ime: <input type="text" name="ime"><br />'
12                'Priimek: <input type="text" name="pri"><br />'
13                '<input type="submit" value="Test">'
14               '</form>')
15
16   # obdelava podatkov odjemalca in posiljanje odgovora
17   def post(self):
18     self.write('Zdravo '+ self.get_argument("ime") +' '
19                         + self.get_argument("pri") )
20
21 if __name__ == "__main__":
22 # zagon aplikacije
23   app = tornado.web.Application(handlers=[(r"/", obrazec)])
24 # posredovanje aplikacije strezniku
25   http_server = tornado.httpserver.HTTPServer(app)
```

```
26    http_server.listen(8888)
27    tornado.ioloop.IOLoop.instance().start()
```

Listing 4.11: Primer implementacije spletnega strežnika Tornado.

Pri pregledu primera 4.11 lahko opazimo razdelitev strežnika na dva glavna dela. Prvi del je rokovalnik zahtev odjemalca (`RequestHandler`). V njem preuredimo HTTP funkciji "pridobi" (`GET`) in "objavi" (`POST`). Funkcija "pridobi" poskrbi za začetek komunikacije med strežnikom in odjemalcem. Odjemalcu prikaže začetno stran z možnostjo vnosa osebnih podatkov. S klikom na gumb "Test" odjemalec pošlje vpisane podatke strežniku in tako se izvede druga funkcija "objavi". Tu strežnik obdela pridobljene podatke ter jih izkoristi za sestavo stavka "Zdravo <ime> <priimek>" . Strežnik pošlje sestavljen stavek odjemalcu, kjer se izpiše na strani odjemalca kot novo spletno mesto. V drugem delu ustvarimo instanco aplikacije ter jo posredujemo objektu strežnika. Aplikaciji je potrebno kot argument dodati razred implementiranega rokovalnika. Tako strežnik ve, kateri rokovalnik uporabiti za določeno zahtevo odjemalca. Temu primerno je potrebno objektu strežnika dodeliti poslušalca vrat. Na koncu ustvarimo še instanco neskončne zanke strežnika. Ta poskrbi za ustrezen zagon strežnika. Za testiranje celotnega sistema je potreben zagon spletnega strežnika ter v brskalniku vpisati spletni naslov `http://localhost:8888` .

### Rokovalniki

Za naše potrebe smo uporabili dve vrsti rokovalnikov: rokovalnik zahtev za preverjanje in prijavo odjemalca ter rokovalnik komunikacije protokola WebSockets za ustrezno komuniciranje (`WebSocketHandler`).

Preverjanje odjemalca zahteva preureditev funkcije `get_curret_user` v enem izmed rokovalnikov zahtev. V tej funkciji podamo zahtevo po pridobitvi določenega zaščitenega piškotka. Dalje smo uporabili še dva rokovalnika zahtev. Eden od njiju poskrbi za prikaz prijavnega obrazca in preverjanje ustreznosti vnesenih podatkov, drugi pa deluje kot nekakšen usmerjevalnik.

Ta glede na pogoje ustrezno preusmeri odjemalca na prijavno stran ali oddaljeno namizje. Pred preusmeritvijo na namizje strežnik shrani piškotek. Spodaj, 4.12, je opisan prvi del implementacije.

```
1  # uvozimo vse module iz primera 4.11 in
2  # se za preverjanje ustreznosti vnesenih podatkov
3  import lock
4
5  # rokovalnik za pridobitev piskotka
6  class BaseHandler(tornado.web.RequestHandler):
7      def get_current_user(self):
8          return self.get_secure_cookie("id")
9
10 # rokovalnik preusmerja glede na shranjen piskotek
11 class MainHandler(BaseHandler):
12     def get(self):
13         if not self.current_user:
14             self.redirect("/login")
15             return
16
17         self.render("clientSockets.html")
18
19 # rokovalnik za prikaz obrazca in preverjanje podatkov
20 class LoginHandler(BaseHandler):
21     def get(self):
22         self.write('<form action="/login" method="post">'
23                 'Username: <input type="text" name="upo">'
24                 'Password: <input type="password" name="passwd">'
25                 'Key: <input type="password" name="key">'
26                 '<input type="submit" value="Sign in">'
27                     '</form>')
28     def post(self):
29         try:
30           if lock.logIn(self.get_argument("upo")+self.
                 get_argument("passwd")):
31               self.set_secure_cookie("id", self.get_argument("upo
                     "), 2)
32         except:
```

```
33              print "Forget something?"
34          self.redirect("/")
```

Listing 4.12: Prvi del implementacije spletnega strežnika Tornado.

Glavni rokovalnik ravno ne preusmeri odjemalca do oddaljenega namizja, ampak z ukazom `render` prikaže odjemalcu vsebino spletne datoteke. V tej datoteki smo implementirali sistem za oddaljen nadzor. Celoten postopek primerne in varne povezave do oddaljenega namizja temelji na zaščitenem piškotku. Ta piškotek shranimo pod pogojem pravilno vpisanih podatkov v prijavnem obrazcu. Podatke preverimo z uporabo funkcije `logIn` modula `lock`, kjer posredujemo podatke obrazca. V tej funkciji smo uporabili kombinacijo teh podatkov in kriptografsko funkcijo `SHA256`. Tu preverimo enakost med ustvarjeno kombinacijo in že prej definirano spremenljivko. Spremenljivka vsebuje podatek, ki je prav tako zaščiten z isto kriptografsko funkcijo. V primeru vpisa pravilnih podatkov v prijavni obrazec, rokovalnik "preusmeri" odjemalca na oddaljeno namizje oziroma na rokovalnik komunikacije.

Rokovalnik komunikacije je poskrbel za ustrezno upravljanje z oddaljenim sistemom ter komunikacijo z odjemalcem. Za pravilno upravljanje smo vanj priredili funkcije za uspešen pričetek komuniciranja z odjemalcem (`open`), prejemanje sporočil o dogodkih tipkovnice, miške in o posodobitvi zaslona (`on_message`), izvedbo posodobitev zaslona (`updateScreen`) ter zaključek komunikacije (`on_close`). V funkcijah smo uporabili implementirane sisteme navideznih naprav ter zajem zaslona. Spodaj, 4.13, je opisan drugi del implementacije.

```
1 import tornado.websocket
2 import shotscr, mouseClick, keyPress, time
3 class eWebSocket(tornado.websocket.WebSocketHandler):
4   def open(self):
5       print "WebSocket opened"
6       self.updateScreen()
7
8   def updateScreen(self):
9       shotscr.shot()
```

```
10          data_uri=open("zajem.jpeg", "rb").read().encode("base64")
11          self.write_message(""+data_uri,
                False)
12
13      def on_message(self, message):
14          dogodek=""+message
15          if "mouse" in dogodek:
16              mouseClick.clik(dogodek)
17
18          if "key" in dogodek:
19              keyPress.press(dogodek)
20
21          if "send me a pic" in dogodek:
22              time.sleep(0.1)
23              self.updateScreen()
24
25      def on_close(self):
26            print "WebSocket closed"
```

Listing 4.13: Drugi del implementacije spletnega strežnika Tornado.

Imena funkcij prikažejo njihovo dejansko funkcionalnost v rokovalniku. Funkcija `open` oznani uspešno povezavo med odjemalcem in strežnikom ter takoj prične z osveževanjem zaslona.

Osveževanje zaslona se izvede v treh korakih. Prvi korak je zajem zaslona ter shranjevanje kot sliko v format JPEG. Korak je podrobneje opisan v poglavju 4.2.1 od strani 34 dalje. Drugi korak izvede odprtje te slike ter jo primerno zakodira kot podatek. Ta je primeren za prikaz v HTML značkah kot na primer <`img`> ali <`canvas`> . Zaradi tega ni potrebno dodatno delo s pripravo slike na strani odjemalca. Slaba lastnost kodiranja je dodatna obremenitev omrežja, saj se poslani podatki v tem formatu do 33 % povečajo. Tretji korak je pošiljanje kodiranih podatkov slike odjemalcu kot sporočilo. Podatkom slike dodamo še lastnost o njenem ter kodiranem formatu. Sporočila lahko, preko protokola WebSockets, pošiljamo tudi po bajtih, zato dodamo še negativno vrednost "False" kot argument.

S strani odjemalca smo pošiljali tri kategorije sporočil. Ti so dogodki

miške, tipkovnice ter zahteva po posodobitvi zaslona. Poslani dogodki miške in tipkovnice so označeni glede na njihovo akcijo oziroma njihovo vrednost. Akcije miške so premik, pritisk in spust. Vrednost tipk tipkovnice so razdeljeni na znake `Unicode`, modifikacije in posebne tipke. Primer 4.10 na strani 40 prikaže in opiše način uporabe vseh treh. Pred klicem funkcije za zajem zaslona smo dodali pavzo vrednosti 0.1 sekunde. Tako smo poskrbeli, da ni prišlo do zatikanja ter preobremenitve pošiljanja podatkov slik.

Za zaključek rokovalnika poskrbi funkcija `close`, ki opozori na konec komunikacije z odjemalcem preko protokola WebSockets.

Zadnji del implementacije strežnika je podoben primeru 4.11, le da smo tu dodali še tri lastnosti strežnika. Te funkcije so razhroščevanje, skrivnost in dodajanje piškotka ter uporabe varne povezave HTTPS s certifikati strežnika. Spodaj, 4.14, je opisan zaključek implementacije spletnega strežnika.

```
1  import os
2  if __name__ == '__main__':
3    application = tornado.web.Application([
4      (r"/", MainHandler),
5      (r"/echo",eWebSocket),
6      (r"/login", LoginHandler),
7      ],debug=True,
8      cookie_secret="<kodirana vrednost>")
9
10   data_path = "/home/th/openssl/"
11   http_server = tornado.httpserver.HTTPServer(application,
         ssl_options={
12                 "certfile":os.path.join(data_path,"web.crt"),
13                 "keyfile":os.path.join(data_path,"web.key"),})
14
15   http_server.listen(8888)
16   tornado.ioloop.IOLoop.instance().start()
```

Listing 4.14: Zadnji del implementacije spletnega strežnika Tornado.

Razhroščevanje omogoča lažje testiranje spletnega strežnika. Poskrbi, da nam za vsak popravek ni potrebno ročno ponovno zagnati le-tega. Določene

sintaktične napake implementacije še vedno ustavijo strežnik. Prepoznava in hranjenje identitete odjemalcev potekata s pomočjo piškotka ter njegove skrivnosti. Funkciji, podprti v strežniku Tornado, sta `set_secure_cookie` in `get_secure_cookie`. Piškotki so podpisani z definirano skrivnostjo. Za funkcijo podpisa Tornado uporabi kriptografski način HMAC. Poleg podpisa je potrebno dodati tudi časovni interval zapadlosti piškotka. V primeru zapadlosti ali napačnega podpisa funkcija za pridobitev piškotka vrne negativno vrednost. Dodatna varnost oddaljenega namizja omogoči uporabo načina povezave s protokolom HTTPS oziroma vpeljave certifikatov strežnika. Tako serviramo promet s kriptografskim protokolom SSL. Certifikat smo ustvarili s pomočjo orodja `OpenSSL`.

Skozi dane primere lahko opazimo, da smo izvedli implementacijo spletnega strežnika Tornado za rokovalnike zahtev in komunikacije WebSockets le v nekaj korakih. Največ časa smo porabili pri logičnem razmišljanju izvajanja in povezovanju vseh potrebnih funkcionalnosti oddaljenega namizja s spletnim strežnikom.

## 4.3 HTML5 in programiranje v Javascript

Prikazali smo implementacijo sistema na strani strežnika, sedaj je na vrsti odjemalec. Zanj smo, kot že večkrat omenjeno, izbrali novo tehnologijo prikaza elementov, imenovano canvas ter način komuniciranja s strežnikom WebSockets. Za primerno implementacijo sistema skrbi skriptni jezik Javascript. V njem združimo vse potrebne dele za prikaz in občutek nadzora oddaljenega namizja.

### 4.3.1 Komunikacija WebSockets

Začetek uporabe komunikacije se prične z definiranjem spremenljivke, v kateri kličemo konstruktor `WebSockets(<povezava>)` . Atribut `<povezava>` je v konstruktorju zapis aktivne povezave oziroma naslov strežnika. Tu v predpono namesto `http` podamo novo shemo `ws` ali `wss` za uporabo varne HTTP

povezave. Dalje še dodamo ustrezne rokovalnike dogodkov, ki poskrbijo za potrdilo odprte povezave (`onopen`) in prejemanje sporočil (`onmessage`). Tem dogodkom priredimo določeno funkcijo, ki se izvede ob primeru pojava letega. Spodaj je opisan primer kode 4.15 kjer se s pomočjo komunikacije WebSocket povežemo do lokalnega strežnika. Glede na njegov naslov lahko razberemo, da je strežnik lokalen, posluša vrata `8888` ter ima definiran rokovalnik komunikacije na naslovu "/poz".

```
1 var povezava = new WebSocket('ws://localhost:8888/poz');
2
3 povezava.onopen = function() {
4   povezava.send('Pozdravljen streznik!');
5 };
6
7 povezava.onmessage = function(dogodek) {
8   alert('Streznik: ' + dogodek.data);
9 };
```

Listing 4.15: Primer uporabe komunikacije WebSockets na strani odjemalca.

Odjemalec kot znak uspešne povezave s strežnikom pošlje sporočilo. Dodali smo podporo možnosti prikaza sporočila, poslanega s strani strežnika. Do tega sporočila pridemo s funkcijo sporočila `dogodek.data` .

Zaradi želje po varni povezavi med strežnikom in odjemalcem smo za našo implementacijo uporabili shemo `wss`. Naslov strežnika je podobno sestavljen kot v primeru 4.15, le da uporabimo njegov IP naslov ter spremenimo pot do rokovalnika komunikacije. Pot do ustreznega rokovalnika je podana na strani 46 v primeru 4.14 kot "`/eWebSocket`".

Posebnost komunikacije je bila kombinacija njenih dogodkov s funkcionalnostjo elementa za prikaz in nadzor oddaljenega namizja. Za pravilen nadzor tega elementa je potrebno vse poslušalce dogodkov definirati pred prikazom spletne vsebine. Tako smo priredili vse poslušalce v funkcijo prvega koraka komunikacije, potrdilo odprte povezave (`onopen`). Več o samih poslušalcev dogodkov v poglavju 4.3.2.

Druga zanimivost je bila posodobitev in prikaz zaslona oddaljenega namizja na tem elementu. Potreben je bil pravilen pristop k rešitvi, saj bi drugače lahko prišlo do večjih težav glede uporabnosti oddaljenega namizja. Zagato smo rešili s pomočjo funkcije dogodka za sprejem sporočil. To smo uredili za pripravo, prikaz in posodobitev podatkov zaslona. Zanimivo dejstvo je to, da s strani strežnika ni bilo potrebe po pošiljanju drugih sporočil razen podatkov o zaslonu. Postopek prikaza slike s podatki je opisan v poglavju 4.3.2. Zahtevo po posodobitvi zaslona smo izvedli po opisanem koraku o prikazu slike. Tako smo zopet dobili podatke o sliki ter ponovili postopek priprave, prikaza in zahteve o posodobitvi zaslona. S tem smo ustvarili zanko med strežnikom in odjemalcem.

### 4.3.2 Element Canvas

Za prikaz in navidezni nadzor oddaljenega namizja je poskrbel element canvas. Ključen element uspešnega nadzora je bila pravilna implementacija vseh poslušalcev dogodkov.

#### Prikaz

Priprava in prikaz slike je sestavljena iz kombinacije ukazov za nadzor elementa canvas. Kot prvo smo želeli pripraviti objekt elementa za risanje slike. To naredimo s kombinacijo ukazov `document.getElementById()` za pridobitev elementa canvas in `getContext("2d")` za pridobitev vsebine tega elementa. Dalje z uporabo konstruktorja `Image` ustvarimo novo sliko ter nanj pripnemo prejete podatke. Z rokovalnikom dogodka ustvarjene slike `onload` narišemo sliko v element. Tako naredimo prikaz prejete slike.

```
1 povezava.onmessage = function (dogodek) {
2   var c=document.getElementById("mojCanvas");
3   var ctx=c.getContext("2d");
4   var img = new Image();
5   img.src = dogodek.data;
6   img.onload = function () {
```

```
7       ctx.drawImage(img,0,0);
8       };
9     povezava.send("send me a pic");
10 };
```

Listing 4.16: Primer posodobitve slike zaslona.

### Nadzor

Nadzor elementa canvas so prevzeli implementirani poslušalci dogodkov. Ti zamenjajo privzete lastnosti vhodnih naprav ter omogočijo izvajanje oddaljenega nadzora sistema. Za naše potrebe smo dodali poslušalca miške, tipkovnice ter za dotik. Tako poskrbimo za vse tipe naprav. Poslušalci so privzeto poimenovani glede na njihovo vlogo.

Pri miški smo uporabili poslušalce za premik (`mousemove`), pritisk in spust gumba (`mousedown` in `mouseup`) ter prepoznavo, ali je v samem elementu (`mouseover` in `mouseout`). S prepoznavo poskrbimo za ustrezno dodajanje in odstranjevanje fokusa na element, saj drugače poslušalci tipkovnice ne delujejo pravilno. Glavne tri funkcije uporabljajo isti način pridobitve informacij o poziciji miške. Med sabo se razlikujejo le po dodatni informaciji o pritisku ali spustu gumba miške. Poslušalec miške je podan spodaj, 4.18.

```
1 c.addEventListener("mousedown",function(evt){
2   var rect = c.getBoundingClientRect();
3   var msX = Math.round(evt.clientX - rect.left + c.scrollLeft);
4   var msY = Math.round(evt.clientY - rect.top + c.scrollTop);
5   if(evt.which==null)
6       mouseClicked=(evt.button<2)?"1":((evt.button==4)?"3":"2");
7   else
8       mouseClicked=(evt.which<2)?"1":((evt.which==2)?"3":"2");
9   povezava.send("mouse down;"+msX+";"+msY+";"+mouseClicked);
10 }
```

Listing 4.17: Primer poslušalca dogodka pritisk miške.

Pozicijo miške dobimo s seštevki koordinat treh različnih objektov. Prvi (`getBoundingClientRect`) opisuje odmik celotnega elementa canvas, drugi

(`evt.clientX/Y`) poda pozicijo pritiska gumba miške ter tretji (`scroll`) doda odmik v primeru premika prevelikega elementa. Prepoznavo posameznega gumba miške smo izvedli glede na uporabo primernega brskalnika, saj ni definiranega standarda zanj. Lastnost dogodka `button` uporabimo za prepoznavo gumbov pri brskalniku Internet Explorer, za preostale pa je lastnost `which`. Opazimo lahko, da so tudi različno definirani. Pri `button` so definirano "1" kot levi gumb, "4" je kolesce ter "2" kot desni gumb. Lastnost `which` uporabi "1" kot levi gumb, "2" je kolesce ter "3" kot desni gumb. Tu smo preuredili definirane lastnosti za potrebe uporabe navidezne miške na strani strežnika.

Poslušalcev dogodkov tipkovnice je manj kot pri miški, a opaziti je večjo kompleksnost pri nadzoru. Tu uporabimo poslušalca za pritisk in spust tipk (`keydown` in `keyup`). Lovili smo tri vrste tipk. Te so alfa-numerične, funkcijske in tipke za modifikacijo. Kakor pri miški tudi tu ni definiranega enotnega standarda za prepoznavo znakov tipk, zato smo posebej definirali tipke pri pošiljanju, saj je strežnik tako lažje prepoznal njen namen. Vsaka tipka je opisana s posebno kodo, značilno za brskalnik ali pa zaradi večje prepoznavnosti v standardu ASCII. Alfa-numeričnim tipkam smo kodo predelali v prepoznaven znak za pisanje. Ostale smo pošiljali kar s kodo, saj so bili že definirani na strani strežnika. Problem se je pojavil pri prepoznavi posebnih tipk na slovenski tipkovnici, saj pri večini brskalnikov ne prepozna definirane kode ali znaka. Zaradi tega smo opustili podporo pri uporabi le-teh. Primer implementacije poslušalca dogodkov tipkovnice je opisan spodaj, 4.18.

```
1  c.addEventListener('keyup',function(evt){
2    evt.preventDefault();
3    var code=(evt.keyCode) ? evt.keyCode : evt.which;
4    var mess="";
5    if( (code >= 65 && code <= 90) || (code >= 48 && code <= 57) )
6      mess="key pressed;"+String.fromCharCode(code).toLowerCase();
7    else
8      mess="key extra;"+code;
9  //koda za dvigalko=16, ostali sta krmilka=17, izmenjalka=18
10   if(code == 16)
```

```
11      mess="key special;shift up";
12
13    if(mess.length != 0)
14      povezava.send(mess);
15  }
```

Listing 4.18: Primer poslušalca dogodka spust tipke.

Z uporabo tipk ali kombinacijo teh smo preprečili privzete funkcije tipkovnice s funkcijo `preventDefault`. Omenili smo že predelavo alfa-numeričnih kod v prepoznavne znake. To omogoča funkcija `String.fromCharCode` . Funkcijo dogodka `keyup`, zgornji primer 4.18, smo uporabili za klik tipke in ne za spust. Razlog je uporaba tipk za modifikacijo. Te smo v funkciji dogodka `keydown` definirali kot pritisk ter tu kot spust. Tako smo lahko dejansko uporabili te tipke na strežniku.

Zadnja implementacija pri odjemalcu je bila podpora uporabe oddaljenega namizja na naprednih mobilnih napravah. Za te naprave smo dodali orodno vrstico z gumbi za vklop miške, pošiljanje in brisanje besedila ter polje za vnos besedila. Vklop, izklop miške omogoči privzete lastnosti zaslona na dotik ali uporabo dotika kot prenosno miško. Pošiljanje in brisanje besedila se doda oziroma izbriše na poziciji utripajočega kurzorja. Orodna vrstica je prikazana na spodnji sliki 4.3.

Slika 4.3: Orodna vrstica na mobilnih napravah.

Za te naprave smo dodali poslušalce za začetek `touchstart`, premik `touchmove` in konec dotika `touchend`. Dodatna funkcionalnost je držanje dotika, ki se izvede kot klik desnega gumba miške. Tu smo na začetek in konec dotika definirali trenuten čas ter podali ustrezen pogoj. Njuna razlika in pogoj sta izvedla pravilno akcijo klika na levi ali desni gumb miške. Zaznava premika dotika je spremenila ta pogoj v negativno vrednost. Funk-

cije poslušalcev dotika so podobno definirane kot funkcije poslušalcev miške. Z razliko od miške, je tu poenostavljena funkcija za pridobitev pozicije in pošiljanje sporočil o dotiku, ki jih posredujemo zgolj pri premiku in koncu dotika. S preverjanjem števila dotikov in omejevanjem pošiljanja sporočil smo spletnemu strežniku preprečili nekontrolirano pritiskanje gumba ter omogočili lažji nadzor izvajanja navidezne miške po naših želja.

Pogoj za pridobitev informacij o dotiku je število le-teh. Za pravilno uporabo je pogoj imeti samo en dotik. To preverimo s seznamom dotikov "`targetTouches`", kjer njihovo število dobimo z uporabo `length`. V kolikor je pogoj pozitiven, si v spremenljivko shranimo ta dotik iz seznama. Dotik dobimo z dostopom do prvega elementa v seznamu, `targetTouches[0]`. Pozicija dotika je sestavljena, kot pri vseh, s koordinatama `x` in `y`. Do teh vrednosti pridemo preko prej shranjenega dotika in uporabe `pageX` in `pageY`. Za dokončno pošiljanje informacij je prisotnih še par potrebnih pogojev, ki pripomorejo k pravilni uporabi navidezne miške. Primer prirejanja pozicije dotika je podan spodaj, 4.19 .

```
1  //  spremenljivke oneX/Y in startTime so predhodno definirane
2  c.addEventListener("touchstart",function(evt){
3  //  preverimo ali smo vklopili misko v orodni vrstici
4    if( window.touchEvent ) {
5      evt.preventDefault();
6  //    izvedba hranjenja pozicije dotika
7      if ( evt.targetTouches.length == 1 ) {
8        var touch = evt.targetTouches[0];
9        oneX = touch.pageX;
10       oneY = touch.pageY;
11 //    shranimo cas zacetka pritiska
12       startTime = new Date().getTime();
13     }
14   }
15 //  izvajanje dotika v privzetem nacinu
16   else
17     return true;
18 },false);
```

Listing 4.19: Primer poslušalca dogodka pritisk dotika.

Veliko časa smo porabili pri implementaciji in testiranju pravilnega nadzora oddaljenega sistema. Uporaba kombinacije tehnologije WebSockets in elementa canvas se je izkazala kot velik izziv. Na našo srečo je že ogromno implementiranih funkcionalnosti, ki so nas rešile iz zagate uporabe le teh. Kljub temu smo se srečali z marsikatero nevšečnostjo. Te so opisane v naslednjem poglavju. Hkrati je prikazano tudi testiranje na različnih napravah.

# Poglavje 5

# Pregled in testiranje sistema

Testiranje sistema je prineslo zelo zadovoljiv nadzor oddaljenega namizja. Na žalost pa je sistem na določenih delih nekoliko pomanjkljiv. V tem poglavju bomo predstavili te težave. Prikazali bomo tudi testiranje nadzora oddaljenega namizja preko različnih naprav.

## 5.1 Pomanjkljivosti

Kot prvo velja omeniti, da je bil namen naloge implementirati ogrodje oddaljenega nadzora, se pravi nekakšno spoznavanje z novimi tehnologijami ter predstaviti njihove zmožnosti. V kolikor bo želja ter potreba po nadaljnjih posodobitvah, bo to dokaj enostavno, saj je sistem razširljiv in omogoča preprosto implementacijo. Trenutno največji problem predstavlja prenosljivost tehnologije obratnega posredniškega strežnika in navidezne tipkovnice na ostale operacijske sisteme, uporabnost oddaljenega namizja v določenih brskalnikih ter omejitev pri uporabi sistem v širokopasovni povezavi.

Pri opisu sistema Nginx, poglavje 3.1, smo omenili trenutno pomanjkanje uradne podpore usmerjanja prometa TCP na ustrezen strežnik. Na srečo že obstaja modul, ki nekako zaobide to težavo. Možnost razširjanja sistema Nginx je bil tudi eden izmed razlogov pri izbiri operacijskega sistema Linux Ubuntu. Z razliko od sistema Nginx, je nadzor navidezne tipkovnice virtkey

prilagojen zgolj za sisteme, ki uporabljajo protokol Xlib. V kolikor bi želeli omogočiti podporo prenosljivosti celotnega sistema, bi bilo potrebno urediti sistem virtkey ali ponuditi sistem, ki zopet nudi podporo samo določenega operacijskega sistema.

Testiranje je prikazalo tudi pomanjkljivost uporabe sistema v določenih brskalnikih. To je bilo opazno pri brskalniku Safari, verzija 5.1, ter mobilnih brskalnikov, ki so vgrajeni ali pa jih je potrebno namestiti v OS Android. Omenjena verzija Safari ponuja podporo uporabe komunikacije WebSockets zgolj za starejšo različico. Z razliko od nekako delne podpore te komunikacije v brskalniku Safari, je v vgrajenem mobilnem brskalniku na OS Android drugače. Tu je potrebna namestitev določenega mobilnega brskalnika, saj vgrajeni nimajo podpore te komunikacije. Posebno pozornost je treba nameniti pri nameščanju in uporabi brskalnikov, saj določeni presenetijo z nezanesljivim delovanjem ali manjkajočo podporo uporabe v določenih sistemih. Kot smo omenili v podpoglavju 5.3, je brskalnik Google Chrome dobro opravil svoje delo v podprtih sistemih.

Problem se pojavi pri uporabi sistema v brezžičnem načinu povezave naprave z usmerjevalnikom ter ponudnikih nizkih hitrosti širokopasovne povezave. Razlog za to je način implementacije sistema za posodabljanje zaslona, saj se posodobljena slika pošilja kar v celoti. To dokaj obremeni delovanje omrežja, a hkrati omogoči zadovoljivo izvajanje posodobitve zaslona. Rešitev je možna s spremenjenim načinom posodobitve le tega ali uporaba ustrezne kompresije slik.

Poleg omenjenih problemov velja omeniti spremembo pri prijavi uporabnika. Želeli smo uporabiti osebne certifikate za verodostojnost odjemalca. Na žalost imajo zgolj vgrajeni mobilni brskalniki podpore uporabe le teh, ostali naj bi to pridobili v bližnji prihodnosti. Zaradi tega smo spremenili način prijave uporabnika. Ohranili smo certifikat spletnega strežnika, saj smo s tem zagotovili uporabo varne povezave preko HTTPS.

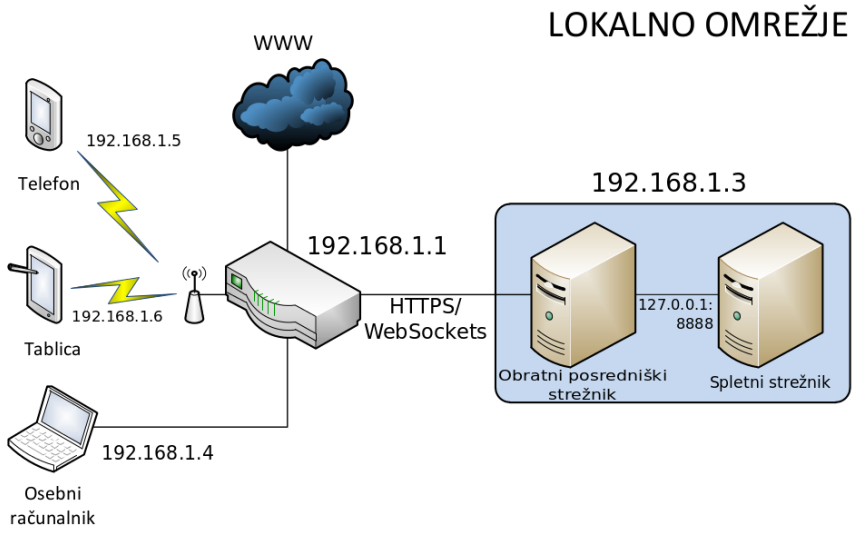
V kolikor bo želja po nadgradnji sistema, bo potrebno vse te pomanjkljivosti rešiti na najbolj preprost in učinkovit način.

## 5.2 Ostali sistemi oddaljenega namizja

Kratki pregled svetovnega spleta razkrije uporabo istih ali vsaj podobnih tehnologij v sistemih za oddaljeno namizje. Določeni imajo podporo za implementacijo strežnika na več OS, drugi zgolj za nekatere. Opazimo lahko, da so nekateri za alternativo ali tudi poleg komunikacije WebSockets uporabili tehnologijo Ajax. To omogoča prilagoditev uporabe sistema, kjer ni polne podpore za komunikacijo preko WebSockets. Nekaj primerov implementiranih sistemov: ThinVNC/RDP, Guacamole, RemoteSpark, TightVNC in noVNC. Vsak je po svoje implementiral sistem oddaljenega namizja ter ponudil podporo uporabe kolikor je le mogoče.

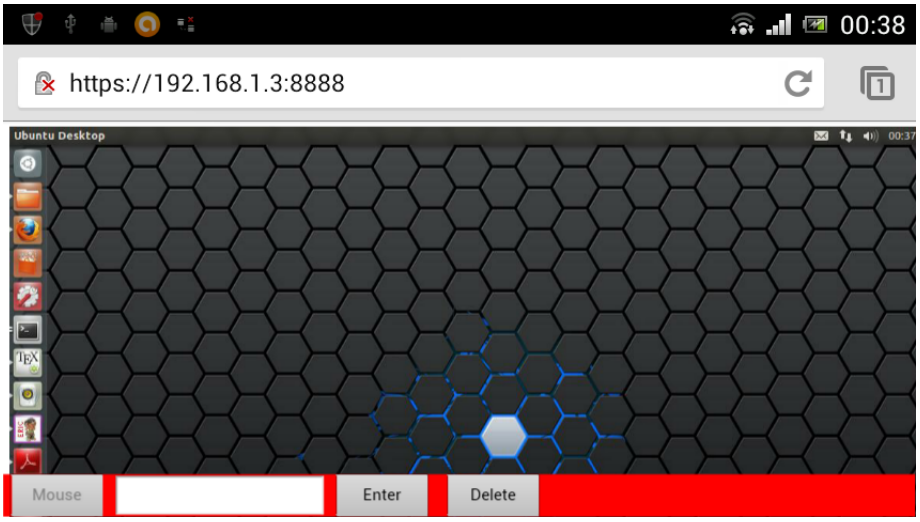
## 5.3 Testiranje

Celotno testiranje je potekalo preko lokalnega omrežja z uporabo določenih brskalnikov. Spodnja slika prikazuje shemo povezljivosti različnih odjemalcev do spletnega strežnika.

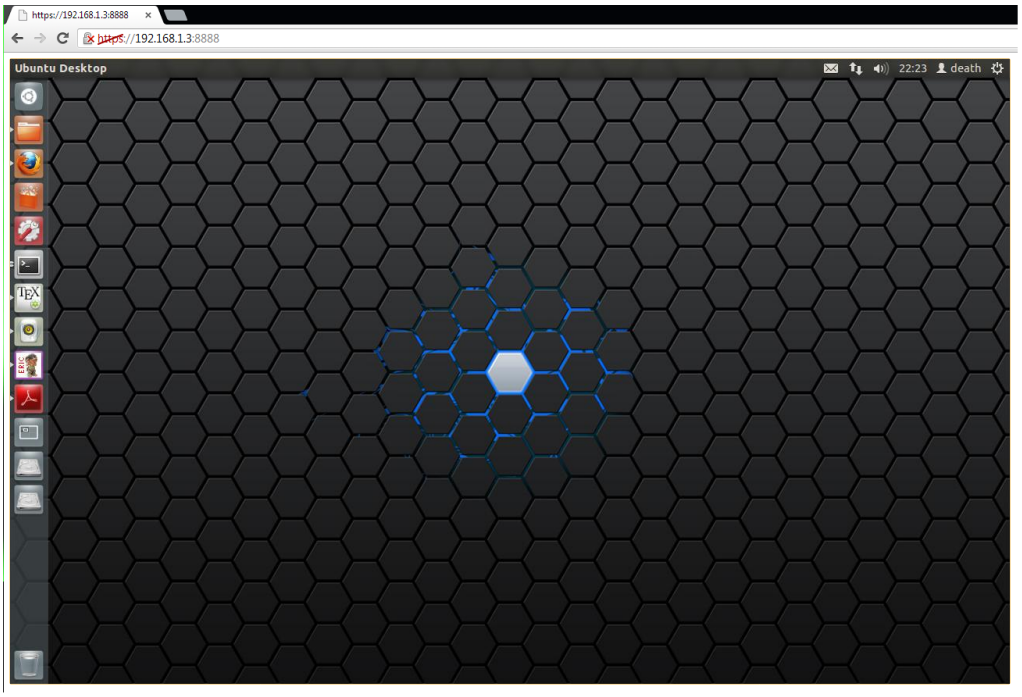



Slika 5.1: Prikaz povezave odjemalcev s spletnim strežnikom.

Najbolje se je odrezal brskalnik Google Chrome, tako na mobilni kot namizni platformi. Sliki 5.3 in 5.2 prikažeta nadzor oddaljenega namizja, na različnih platformah, preko tega brskalnika.




Slika 5.2: Prikaz oddaljenega namizja na sodobnem naprednem telefonu.




Slika 5.3: Prikaz oddaljenega namizja na osebnem računalniku.

# Poglavje 6

# Sklepne ugotovitve

Glavni cilj diplomske naloge je implementacija oddaljenega namizja z uporabo tehnologije HTML5, podrobneje način komunikacije preko WebSockets ter prikaz in nadzor namizja z elementom canvas. Skozi nalogo smo na strani strežnika in odjemalca podrobno opisali in prikazali vse potrebne elemente za uspešno izvajanje oddaljenega namizja. Skoraj vse elemente smo okrepili s podrobnimi primeri izvorne kode in ustrezne konfiguracije sistema. Med elementi strežnika velja omeniti uporabo sistema za zajem zaslona PySide, navidezne tipkovnice virtkey in miške PyMouse, preusmerjanje prometa z obratnim posredniškim strežnikom Nginx ter spletnim strežnikom Tornado, ki je poskrbel za uspešno komunikacijo z odjemalcem in povezal vse elemente za nadzor sistema. Prikaz delovanja navidezne tipkovnice in miške smo dodelali še s primeri uporabe sistema Xlib, ki je prisoten v vseh sistemih X. Na strani odjemalca smo s kombinacijo izbranih tehnologij poskrbeli za ustrezen prikaz in pristen občutek nadzora oddaljenega namizja v realnem času. K temu je tudi pripomoglo nenehno testiranje funkcij. Prišli smo do spoznanja, da je sistem že ogromno napredoval in nam dejansko lahko veliko ponudi. Nekako nas je to tudi presenetilo, saj je tehnologija še dokaj nova.

S temo diplomske naloge smo skušali prikazati enostavnost ter preprostost implementacije in uporabe sistema za oddaljeno namizje. Velik del raziskovanja je bil usmerjen v uporabnost novih tehnologij, podprtih predvsem s

strani spletnih brskalnikov. S tem smo poskušali prikazati nove zmožnosti le teh, saj jih uporabljamo vsak dan in se verjetno ne zavedamo njihove moči. Nalogo smo z našega vidika rešili zelo zadovoljivo. Možne razširitve k večji in boljši uporabnosti sistema vidimo predvsem v načinu zajema in posodobitvi zaslona, lastni implementaciji navideznih naprav in izboljšanju varnosti pri prijavi odjemalca v sistem. Na samem začetku implementacije je bil namen uporabe zasebnih certifikatov za prijavo odjemalcev. To se je kmalu izjalovilo, saj mobilni spletni brskalniki še nimajo podpore za uporabo le-teh. Tako smo certifikate omejili ter spremenili način prijave.

Na svetovnem spletu lahko opazimo nekaj sistemov, ki uporabijo tehnologijo HTML5 pri implementaciji oddaljenega namizja. Par primerov implementiranih sistemov: ThinVNC/RDP, noVNC in TightVNC. Nekateri imajo večjo podporo uporabnikov, drugi manjšo, vsi pa se borijo za prevlado v svetu sistemov oddaljenega namizja.

# Literatura


[1] M. Dory, A. Parrish and B. Berg. *Introduction to Tornado*, O'Reilly Media, 2012.

[2] A. Downey, *Think Python: How to Think Like a Computer Scientist*. Green Tea Press, 2012.

[3] K. Drake, *XTEST Extension Protocol*, UniSoft Ltd., verzija 2.2 , 2002.

[4] S. Fulton, J. Fulton, *HTML5 Canvas: Native Interactivity and Animation for the Web*. O'Reilly Media, 2011.

[5] J. Gettys, R. W. Scheifler, *Xlib − C Language X Interface*, The Open Group, poglavje 1 in 2, zadnja različica l. 2002.

[6] Z. Kessin, *Programming HTML5 Applications*, O'Reilly Media, 2008.

[7] C. Nedelcu, *Nginx HTTP Server*, Packt Publishing Ltd, 2010.

[8] T. Richardson, Q. Stafford-Fraser, K. R. Wood and A. Hopper. *Virtual Network Computing*, IEEE Internet Computing vol. 2, num. 1, 1998.

[9] J. Fulton, *Xev*. MIT X Consortium. Dostopno na: http://www.xfree86.org/current/xev.1.html

[10] T. Richardson, J. Weatherall, A. Harter, K. Wood. *The RFB Protocol*, RealVNC Ltd (formerly of Olivetti Research Ltd / AT & T Labs Cambridge), verzija 38., 2010. Dostopno na: http://www.realvnc.com/docs/rfbproto.pdf

[11] M. Ubl, E. Kitamura, *Introducing WebSockets: Bringing Sockets to the Web*. HTML5 Rocks. Dostopno na:
http://www.html5rocks.com/en/tutorials/websockets/basics/

[12] *PySide, Python for Qt*. Dostopno na:
http://qt-project.org/wiki/PySide

[13] *Remote Desktop Protocol*, Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Remote_Desktop_Protocol

[14] *TIOBE Programming Community Index*. Dostopno na :
http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[15] *The HTML5 test – how well does your browser support HTML5*. Dostopno na:
http://html5test.com

[16] *Tornado Documentation*. Dostopno na:
http://www.tornadoweb.org/documentation/index.html

[17] *Virtual Network Computing*, Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Virtual_Network_Computing

[18] *Xrdp*. Dostopno na:
http://www.xrdp.org/