SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6678

VIZUALIZACIJA PODATAKA DOBIVENIH SEKVENCIRANJEM DNA

Nikola Osredek

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6678

VIZUALIZACIJA PODATAKA DOBIVENIH SEKVENCIRANJEM DNA

Nikola Osredek

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

ZAVRŠNI ZADATAK br. 6678

Pristupnik:	**Nikola Osredek (0036504054)**
Studij:	Računarstvo
Modul:	Programsko inženjerstvo i informacijski sustavi
Mentor:	doc. dr. sc. Krešimir Križanović

Zadatak: **Vizualizacija podataka dobivenih sekvenciranjem DNA**

Opis zadatka:

Današnji uređaji za sekvenciranje nisu u stanju sekvencirati cijele genome, već samo pročitati manje dijelove (očitanja) i pri tome unose određenu količinu pogreške. Jedan od prvih i osnovni zadataka prilikom analize podataka dobivenih sekvenciranjem jest mapiranje očitanja na referentni genom ili transkriptom. Zbog velike količine podataka, nije moguće ručno analizirati mapiranje svakog pojedinog očitanja, već je puno praktičnije koristiti prikladne alate za vizualizaciju. Potrebno je napisati aplikaciju za vizualizaciju mapiranja DNA očitanja na referentni genom. Aplikacija treba prikazati referencu, pokrivenost reference mapiranim očitanjima te mapiranja pojedinih očitanja. Aplikacija treba omogućiti zumiranje prikaza do razine pojedinog nukleotida te prikaz podataka prilagoditi razini zumiranja. Treba podržati učitavanje standardnih formata datoteka kao što su FASTA, FASTQ, SAM i BAM. Aplikacija treba moći efikasno upravljati većim količinama podataka. Rješenje treba biti napisano kao desktop aplikacija za operacijski sustav Linux. Programski kod je potrebno komentirati i pri pisanju pratiti neki od standardnih stilova. Napisati iscrpne upute za instalaciju i izvođenje. Kompletno programsko rješenje postaviti na Github pod jednom od OSI-odobrenih licenci.

Rok za predaju rada: 12. lipnja 2020.

Sadržaj

Uvod 1

1. Analiza problematike 2

1.1. Sekvenciranje 2

1.2. Mapiranje 2

1.2.1. FASTA format 2

1.2.2. FASTQ format 3

1.2.3. SAM format 3

1.3. Vizualizacija 4

2. Arhitektura sustava 5

2.1. Tehnologije 5

2.1.1. Vue.js 5

2.1.2. ASP.NET Core 6

2.1.3. Electron 6

2.2. Pozadinski servis 6

2.3. Korisničko sučelje 9

3. Funkcionalnost aplikacije 11

4. Moguća nadogradnja 16

Zaključak 17

Literatura 18

Sažetak 19

Summary 20

Privitak 21

Uvod

Sva genetska informacija je sadržana u lancu DNA i RNA te su koristi od određivanja tog slijeda ogromne, kao na primjer bolje razumijevanje nasljednih bolesti, infekcija, tumora i ostalih bioloških interakcija organizma. Sekvenciranje je proces određivanja poretka pojedinih nukleotida [1]. Današnji uređaji za sekvenciranje mogu pročitati samo ograničene količine nukleotida koji se onda mapiraju na referentni genom. To su izrazito velike količine podataka koje se ne mogu ručno analizirati.

Svrha ovog rada je izrada aplikacije za vizualizaciju rezultata sekvenciranja. Aplikacija za ulaz uzima već mapirana očitanja i prikazuje gdje se nalaze u odnosu na referentni genom. Korisnik može putovati po referenci i vidjeti koja sva očitanja su mapirana na nekoj određenoj poziciji. Iz mapiranih podataka se također generira histogram na kojem se može vidjeti pokrivenost referentnog genoma te ako referentni genom ima više kromosoma može se uspoređivati pokrivenost pojedinog kromosoma.

1. Analiza problematike

1.1. Sekvenciranje

Sekvenciranje je proces određivanja poretka adenina, citozina, gvanina i timina unutar DNA lanca (A, C, G i T). 1990. godine je pokrenut *Human genome project* kojemu je cilj bio odrediti sve parove baza ljudske DNA. Projekt je završio 2003. i pokazao da se ljudski genom sastoji od otprilike 20500 gena te da se gen prosječno sastoji od 3000 baza, a može i varirati do 2.4 milijuna baza.

Uređaji za sekvenciranje mogu pročitati od 50 nukleotida do nekoliko desetaka tisuća što nije dovoljno za duže gene. Zbog toga se koristi takozvano *shotgun* sekvenciranje gdje se DNA slučajno lomi na mnogo malih dijelova koji se onda posebno sekvenciraju i pomoću računala sastavljaju natrag.

1.2. Mapiranje

Mapiranje je postupak traženja preklapanja očitanja dobivena iz uređaja za sekvenciranje i nekog poznatog referentnog genoma. Mapiranje se radi pomoću programa kao što su *minimap2* koji za ulaz uzima referentni genom i očitanja, a izlaz su mapirana očitanja u SAM formatu.

1.2.1. FASTA format

FASTA format sadrži referentni genom na kojeg se mogu mapirati očitanja. FASTA format je tekstualan i služi za prikazivanje sljedova proteina i nukleotida. Svaki nukleotid je zapisan jednim slovom (A, C, G, T i U). Datoteka se može sastojati od više zapisa i svaki zapis se sastoji od zaglavlja koji sadrži identifikator te od slijeda koji se može protezati u više linija.

Primjer FASTA datoteke:

```
>IDENTIFIKATOR ime slijeda
ACGAACTCAGTCATCTACCCTAGATCATCATTCA
>IDENTIFIKATOR2 drugi slijed
AAAAAACGTTACTTTACTTACTCATCAT
```

1.2.2. FASTQ format

FASTQ format se koristi za zapis izlaznih podataka uređaja za sekvenciranje. FASTQ datoteka kao i FASTA može sadržavati više zapisa, svaki zapis se sastoji od 4 linije gdje je 1. linija identifikator očitanja i izborni opis, 2. linija je očitanje, 3. linija počinje sa znakom + te izborno nakon njega identifikator iz 1. linije s bilo kakvim opisom i 4. linija koja predstavlja kvalitetu očitanja i ima jednak broj znakova kao linija 2.

Primjer FASTQ datoteke:

```
@identifikator
AGGTAGCCTGTTTTATGAGTGGGGATTACAAAGGATGGTCATCGGTCACGTGACAGTACGG
+
%%*+'..,*,-,3**)'((+*))),*,*-56+.,4/,*,-+-+,+221-*+*0/0.,.,+)
@identifikator2
ATCTTGATAGAGATTTCTAGGGGTATGCGTTAATGATCAGAAGCTCCCAATCAGGTTCAGT
+
%%(--..'&(()*67--&&+''&')+(**,)*,-)+.,217(*)+-.1..**(((--.).2
@identifikator3
GTTGCTTGTGCAGGTTACAGGATCCGATCTGCTCGATCCTGAAGAATTTGCCGAGCCCGAG
+
%%*+,++--..+(()())*(')'&(&(*),*+'(').(&'((*+--*,**,+**)(&(((' 
```

1.2.3. SAM format

SAM datoteka sadrži mapirana očitanja i informacije o njihovoj kvaliteti. SAM datoteka započinje zaglavljima koji imaju ispred sebe znak @ te nakon toga svaka linija je jedno očitanje s 11 obaveznih polja odvojenih *TAB* znakom [2].

Ta polja su redom ime očitanja, zastavica, ime reference, pozicija na referenci (počevši od 1), kvaliteta mapiranja, CIGAR niz, ime sljedećeg očitanja, pozicija sljedećeg očitanja, izračunata dužina reference, očitanje te kvaliteta očitanja iz FASTQ datoteke.

Za potrebe vizualizacije potrebni su samo zastavica, pozicija na referenci te CIGAR niz.

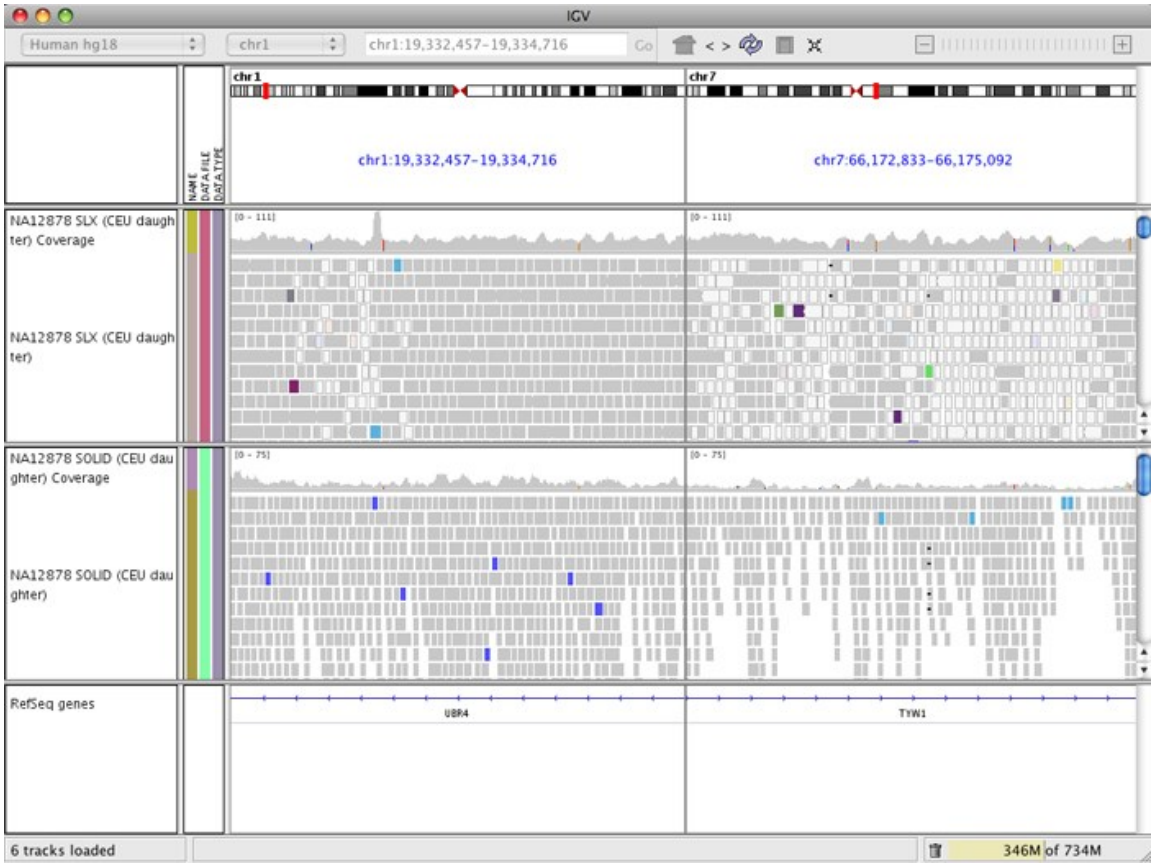
Zastavica je *bitwise* zastavica, a bitovi koji su nam važni su bit 4 koji kaže da segment nije mapiran te bit 16 koji nam govori da je segment reverzno komplementaran u odnosu na referencu.

CIGAR niz je niz znakova koji nam opisuje kako je očitanje mapirano na referencu. Sastoji se od brojka i znakova M,I,D,N,S,H,P,X i =. Svaka operacija se sastoji od brojke i jednog

od prije navedenih znakova. To označava da ta operacija vrijedi za taj broj nukleotida u segmentu. Znakovi redom označavaju operacije podudaranje (*match*), ubacivanje u referencu (*insert*), obrisano iz reference (*delete*), preskakanje reference (*intron*), nije dio reference ali je dio očitanja (*soft clipping*), nije dio reference i izbrisano je iz očitanja (*hard clipping*), podstava (*padding*), ne podudara se, podudara se.

1.3. Vizualizacija

Podaci dobiveni sekvenciranjem su često u rangu veličine GB ili čak nekoliko desetaka GB. To je prevelika količina podataka da se ručno analizira iz tekstualne datoteke. Zato postoje alati za vizualizaciju kao što je IGV (Integrative Genomics Viewer) u kojima se može jednostavno vidjeti poravnanja očitanja s referencom i pokrivenost cijele reference. Primjer vizualizacije mapiranja prikazan je slikom (Slika 1.1 IGV).

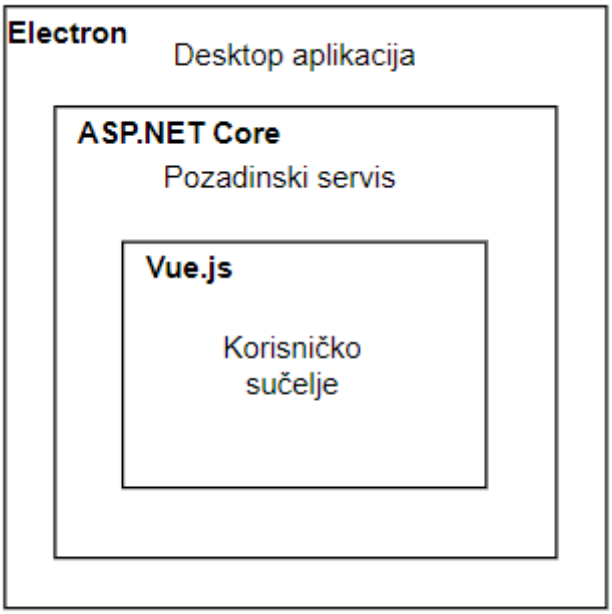


Slika 1.1 IGV

2. Arhitektura sustava

Kvalitetno osmišljena arhitektura je jedan od najbitnijih dijelova razvoja aplikacije. Ako je arhitektura dobro osmišljena aplikacija će se moći lako održavati i jednostavno nadograđivati.

Jedan od zahtjeva za ovu aplikaciju je bio da bude desktop aplikacija koja radi na operativnom sustavu *Linux*. Ova aplikacija je osmišljena kao web aplikacija kojoj je korisničko sučelje napravljeno pomoću *Vue.js*, pozadinski servis u *ASP.NET Core* te je sve „omotano“ pomoću *Electron* razvojnog okvira čime je rezultat desktop aplikacija za više platformi. Pozadinski servis vraća *Vue.js* aplikaciju, a *Electron* pokreće pozadinski servis (Slika 2.1 Shema aplikacije).



Slika 2.1 Shema aplikacije

2.1. Tehnologije

2.1.1. Vue.js

Vue.js je JavaScript okvir za stvaranje korisničkih sučelja i takozvanih *single page applications* (SPA). Jedna od najkorisnijih funkcionalnosti *Vue.js* je reaktivnost. Svaka

komponenta prati stanje svih svojih reaktivnih ovisnosti tako da sustav zna točno kada treba koju komponentu ponovno iscrtati. To nam omogućuje da korisniku olakšamo korištenje aplikacije bez da zakrčujemo ekran.

2.1.2. ASP.NET Core

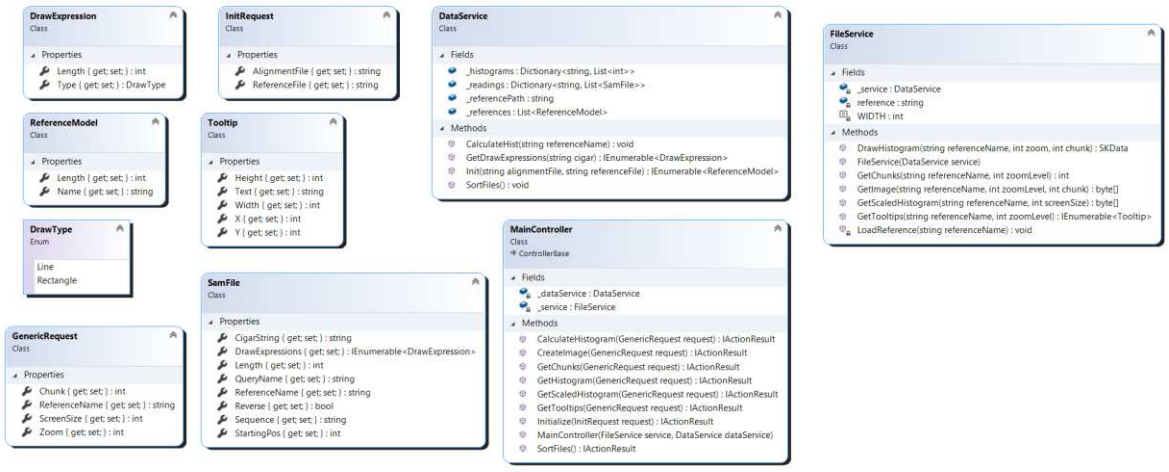
ASP.NET Core je web radni okvir razvijen i održavan od strane Microsofta te služi za izradu web aplikacija. Neke od važnijih funkcionalnosti su to da se *ASP.NET Core* aplikacije mogu pokretati na *macOS*, *Linux* i *Windows* sustavima, velik izbor službenih i korisničkih biblioteka koje se distribuiraju preko *Nuget* paketa i velike performanse.

2.1.3. Electron

Electron je razvojni okvir razvijen i održavan od strane Github-a koji služi za razvoj desktop aplikacija s grafičkim sučeljem pomoću web tehnologija. U pozadini koristi *Chromium* i *Node.js* i to omogućava da se aplikacija pokreće na raznim platformama. Neke od poznatijih aplikacija koje koriste *Electron* su *Slack, Visual Studio Code* i *Skype.*

2.2. Pozadinski servis

Pozadinski servis je pisan u C# pomoću ASP.NET Core radnog okvira. Sastoji se od jednog kontrolera na kojem se nalaze sve relevantne metode. Kontroler se sastoji od 8 metoda koje sadrže minimalnu količinu logike te većinom samo predaju parametre nekom servisu i vrate rezultate. Dijagram razreda priakzan je na sljedećoj slici (Slika 2.2 Dijagram razreda).



Slika 2.2 Dijagram razreda

Prva metoda koja se poziva je *Initialize* metoda koja prima put do SAM datoteke i put do FASTA datoteke te ih predaje servisu.

DataService je zamišljen kao *Singleton* koji početno obradi datoteke te u memoriji sačuva manje količine podataka. Sve započinje pozivom *Init*. U toj metodi datoteka s očitanjima se otvori kao datotečni tok i obrađuje se liniju po liniju. Ako linija počinje sa znakom „@“, to predstavlja zaglavlje i ako to zaglavlje ne počinje sa nizom „@SQ“ ta linija se preskače jer u tom zaglavlju nema bitnih informacija za vizualizaciju. Ako započinje s nizom „@SQ“ to označava da sadržava ime reference i dužinu reference te se ti podaci spreme u memoriju. Sve linije nakon zaglavlja su u formatu 11 ili više podataka odvojenih „\t“ znakom. Linija se obradi na način da se podijeli po „\t“ znaku te se prvo pogleda string na drugom mjestu koji označava zastavicu. Ako je zastavica „4“ to znači da očitanje nije mapirano i može se ignorirati. Ako nije onda se podaci o imenu očitanja, imenu reference, početna pozicija, CIGAR string i zastavica spreme u privremenu datoteku za tu referencu. Nakon što se obrade sve linije vraćaju se imena i dužine referentnih sekvenca. Na taj način se veličina datoteka drastično smanji i puno je lakše raditi s podacima.

Pozivom metode *GetDrawExpressions* dobivamo listu crta i pravokutnika koji označavaju ako se sekvenca poklapa s referencom. Ta metoda uzima za ulazni parametar CIGAR string. Ona radi na vrlo jednostavan način, u petlji se provjerava znak po znak iz CIGAR stringa. Za znakove „M“ i „=“ spremamo te podatke kao pravokutnik, a sve ostale kao liniju. Tu metodu zovemo više puta u kodu jer ju je isplativije opet pozvati nego spremiti podatke u memoriju ili u datoteku.

Nakon što se datoteke obrade i podijele u manje privremene datoteke pozove se metoda *SortFiles* koja jednostavno učita jednu po jednu privremenu datoteku u memoriju i sortira ih uzlazno ovisno o početnoj poziciji.

Stvaranje histograma radi na način da se za neku referencu prvo pozove metoda *CalculateHist.* Histogram se računa tako da se čita linija po linija iz privremene datoteke i u listu veličine reference se sprema broj koliko pravokutnika je mapirano na to mjesto. Histogrami se ne računaju odmah kod inicijalizacije nego tek kad se pozove za neku referencu tako da bi se uštedjelo vrijeme ako korisnika možda ne zanimaju sve reference.

Kreiranje slika se vrši pomoću biblioteke *SkiaSharp* [4]. To je omotač oko *Skia* biblioteke za 2D crtanje razvijene od strane Google koja radi na različitim platformama.

Crtanje skaliranog histograma se radi na način da se za svaki piksel uzme prosjek nekog broja vrijednosti običnog histograma. Skaliranje histograma je ovisno o veličini korisničkog ekrana koji se predaje kao parametar te je svaka vrijednost prikazana kao pravokutnik širine jednog piksela, a visine su skalirane tako da je prosječna vrijednost na sredini slike. Slika se prije vraćanja natrag mora prvo kodirati u neki od formata za slike koje *Chromium* može prikazati, a na raspolaganju su nam PNG,WEBP i JPEG format. PNG i WEBP su formati koji se mogu kodirati bez gubitka pa se u aplikaciji koriste njih dvoje ovisno o potrebi.

Histogram pune veličine je najčešće prevelik da se pošalje odjednom pa je napravljeno da se šalje u više dijelova od 15000 piksela. 15000 piksela je odabrano kao granica zbog ograničenja WEBP formata. Ovaj histogram jednako kao i smanjeni histogram ima vrijednosti skalirane na takav način da srednja vrijednost cijelog histograma postiže svoj maksimum u sredini slike te se tamo također dodaje iscrtkana linija i vrijednost histograma na toj visini. Veličina zuma označava kolika će biti širina pravokutnika u histogramu gdje je početna širina jedan piksel, a maksimalna 10 piksela. Ako se odabere da je širina 10 piksela, ispod histograma će se dodati prikaz reference kao niz slova (A, C, G i T) u fontu *Courier New*. Odabran je taj font jer su svi znakovi jednake širine i dostupan je na većini računala. Prije ispisivanja teksta potrebno je izračunati veličinu teksta i skalirati da jedno slovo bude širine 10 piksela te se onda tekst može postaviti odmah ispod histograma.

Postupak vizualizacije poravnanja je glavna točka ovog rada i taj postupak je prošao kroz par iteracija. Prva iteracija je bila da se izračunaju lokacije pravokutnika i crta te da se onda samo pošalju na korisničko sučelje koje će to onda nacrtati. To je bilo najbrže rješenje, ali zbog nepoznatih i nedokumentiranih razloga nakon neke širine nije se više htjelo nacrtati. Druga iteracija je bio pokušaj kreiranja SVG datoteke. SVG datoteka sprema crtež kao vektorski zapis te bi se vrlo jednostavno mogla skalirati bez gubitka kvalitete kad bi se zumiralo. Problem kod tog pristupa je bila divovska količina informacije. Od privremene datoteke veličine 20-ak MB se stvorila SVG datoteka veličine 700 MB i time se onda gubi previše vremena da se učita i velika količina memorije. Zadnja iteracija je najbolje pronađeno rješenje da bi aplikacija mogla raditi na više platformi. U pozadinskom servisu se generiraju djelići slike koji se samo spajaju na korisničkom sučelju. Detaljnije je opisano u sljedećem odlomku.

Vizualiziranje poravnanja radi se u metodi *GetImage* koja kao i histogram vraća samo djelić slike ovisno koliki zum se želi. U metodi se učitavaju podaci očitanja liniju po liniju

iz privremene datoteke, ubaci se u prije opisanu metodu *GetDrawExpressions* i postavi se u prvi red na koji može kako bi se smanjila sveukupna visina slike i povećala preglednost. Traženje prvog reda u koji se očitanje može staviti je napravljeno na jednostavan način. Očitanja su već prije sortirana po početnoj poziciji, redom se prolazi po očitanjima i provjerava se ako u pomoćnoj listi ima element kojemu je krajnja pozicija veća za minimalno 1 od početne pozicije trenutnog očitanja. Ako je onda se samo element s tim indexom promijeni na novu krajnju poziciju, a u suprotnom se u listu doda novi element s krajnjom pozicijom trenutnog elementa. Vizualizacija očitanja se prikazuje kao niz crta i pravokutnika pa zbog ubrzavanja crtanja izračuna se ukupna dužina očitanja i nacrta se linija od početka do kraja čime smo smanjili broj elemenata za crtati na pola. Rezultantna slika se pokušava kodirati u WEBP formatu jer je algoritam za kodiranje brži, ali slika može biti viša od 15000 piksela što je izvan ograničenja formata i u tom slučaju slika se kodira u PNG formatu. Kodiranje slike je točka na kojoj aplikacija potroši najviše vremena i zbog tog razloga su rješenja koja koriste višeslojnu arhitekturu najčešće puno brža, ne moraju kodirati slike nego imaju direktan pristup površini za crtanje.

Kod pregledavanja mapiranja bitno je znati koje očitanje na slici je koje. Metoda *GetTooltips* to radi na način tako da učitava liniju po liniju iz privremene datoteke te za svako očitanje zna početnu poziciju i izračuna ukupnu dužinu. Te podatke spremi u listu kao opis pravokutnika zajedno s imenom očitanja te tu listu vraća korisničkom sučelju da prikaže.

2.3. Korisničko sučelje

Korisničko sučelje je pisano pomoću *Vue.js* okvira i sastoji se od 3 jednostavna pogleda (engl. *view*). Prvi pogled je jednostavna forma za unos 2 datoteke. Preglednici inače nemaju pristup cijelom ciljnom putu datoteke, ali *Electron* okvir dodaje te informacije. Nakon što su datoteke dodane poziva se metoda za inicijalizaciju te nakon toga za sortiranje podatka na pozadinskom servisu. Pozadinski servis vraća informacije o referencama koje se spremaju te se otvara drugi pogled.

U drugom pogledu se koristi paket *Konva* koji ima mnoštvo funkcija za upravljanje HTML5 Canvas elementom [3]. Pomoću *Konva* paketa se vrlo jednostavno postiže micanje po referenci te crtanje slika na ekran. U ovom pogledu sve počinje s *mounted* metodom koja se automatski poziva kad se otvori taj pogled. U *mounted* metodi

inicijaliziraju se sva platna za crtanje, pozove se metoda za inicijalizaciju histograma na pozadinskom servisu. Nakon toga se pozivaju metode za crtanje svih elemenata.

Svaka od metoda koje se pozivaju radi na sličan način. Znaju na koliko dijelova se slike dijele te koliko su te slike velike. U petlji se pozivaju metode s pozadinskog servisa dio po dio umjesto sve odjednom što usporava crtanje, ali omogućava uređajima s manje memorije da mogu koristiti aplikaciju. Za slike poravnanja se dodatno provjerava ako je slika koja je stigla veća od platna te izduži platno po potrebi.

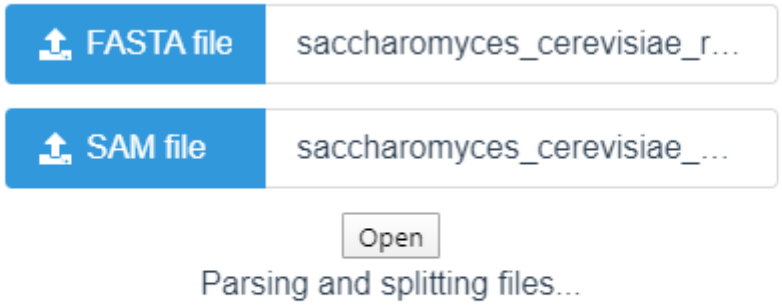
U aplikaciji postoji više načina micanja po referenci. Može se upisati točna lokacija nukleotida na koju se želi pomaknuti, može se vući histogram ili pravokutnik na skaliranom histogramu za brže micanje. *Konva* omogućuje da se neki element može vući te onda kad se aktivira event treba i sve ostale elemente pomaknuti na tu lokaciju. Koristeći te mogućnosti treba paziti za koliko se miču drugi elementi jer sva 3 načina miču referencu u različitim omjerima.

Bitno je još dodati informacije o očitanjima koje se radi u metodi *tooltips.* Svi podaci o očitanjima se zapišu kao pravokutnici s imenom očitanja na posebnom sloju. Na tom sloju se osluškuje ako je pokazivač ušao u koji od pravokutnika i pokaže se njegovo ime kraj pokazivača.

Posljednji pogled prikazuje smanjene histograme svih referenci jedan ispod drugog da se može jednostavno vidjeti pokrivenost svake reference odjednom. U svojoj *mounted* metodi pogled napravi platno dovoljno visoko za svaki histogram i ime reference. Nakon toga za svaku referencu koju ima spremljenu u memoriji poziva metodu za inicijalizaciju histograma za pozadinski servis te nakon toga metodu za dobivanje smanjenog histograma.

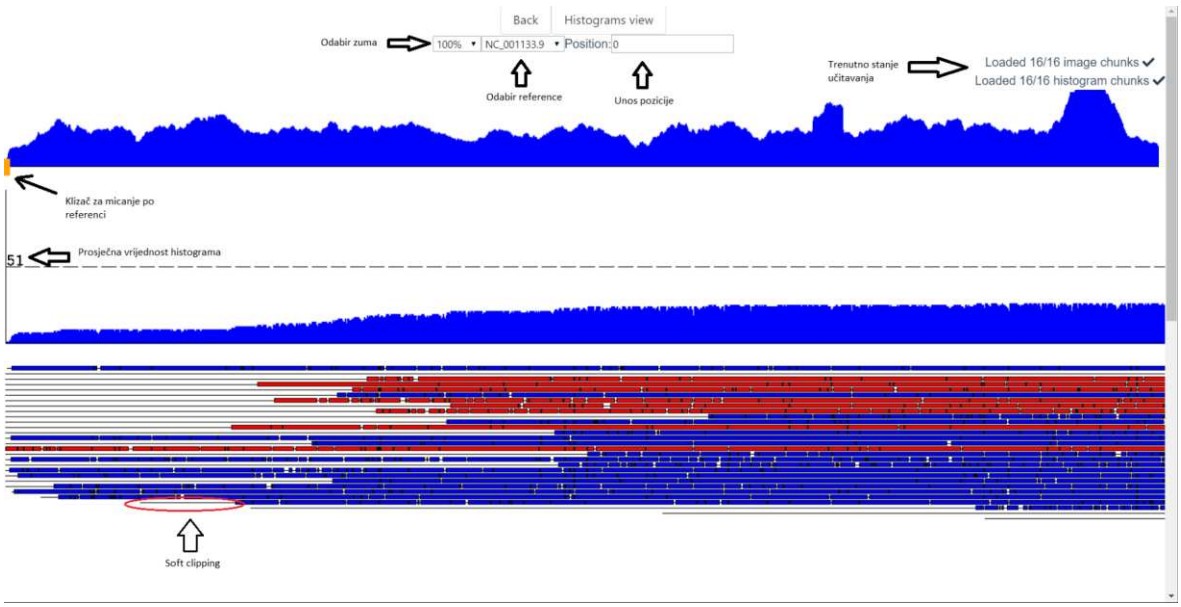
3. Funkcionalnost aplikacije

Početni pogled se sastoji od odabira datoteka i gumba za nastavak. Datoteke se odaberu te se klikne gumb za nastavak. Ako su datoteke malo veće i treba neko vrijeme da se obrade onda u opisu piše u kojoj je fazi inicijalizacije pozadinski servis trenutno (Slika 3.1 Početni ekran).



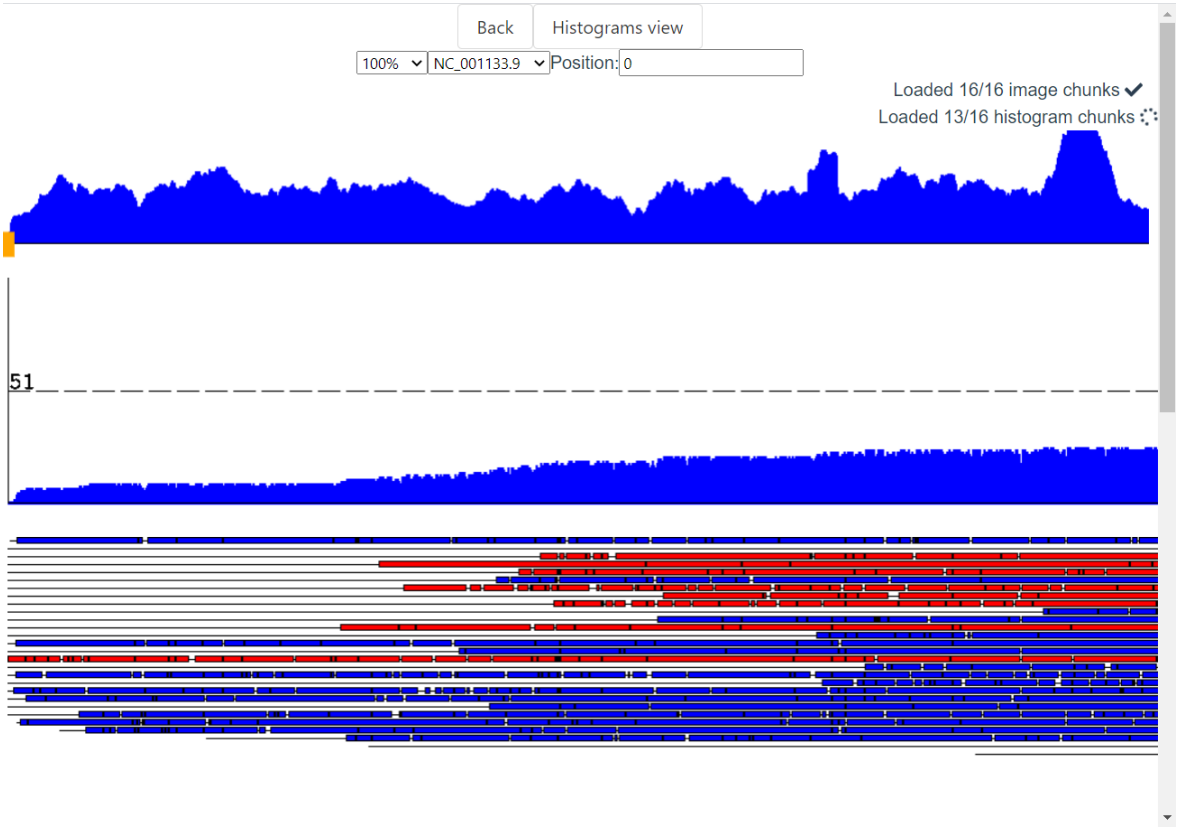
Slika 3.1 Početni ekran

Slijedeća slika prikazuje opis korisničkog sučelja (Slika 3.2 Opis korisničkog sučelja).



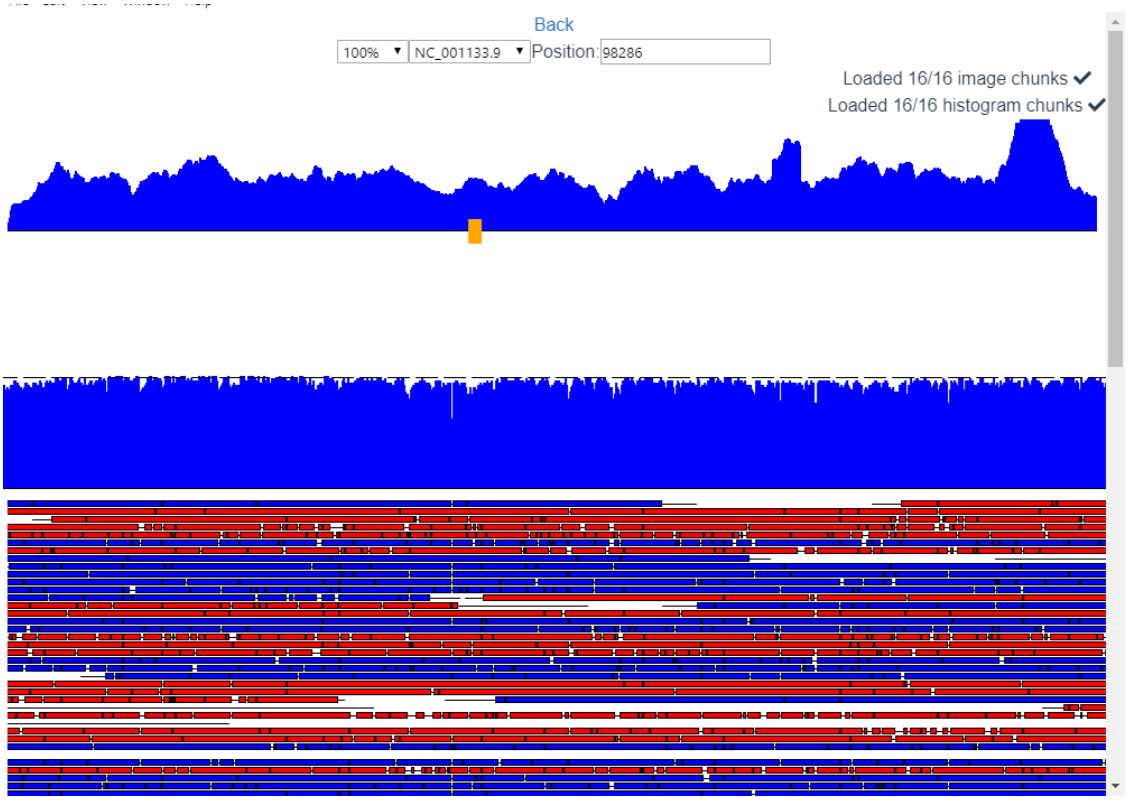
Slika 3.2 Opis korisničkog sučelja

Nakon što se sva inicijalizacija odradi otvara se vizualizacija te na ekranu možemo vidjeti smanjeni histogram tako da cijeli stane na ekran, početak velikog histograma i početak prikaza poravnanja. Na vrhu ekrana možemo vidjeti opcije za zum, odabir reference i odabir pozicije na referenci (Slika 3.3 Vizualizacija).



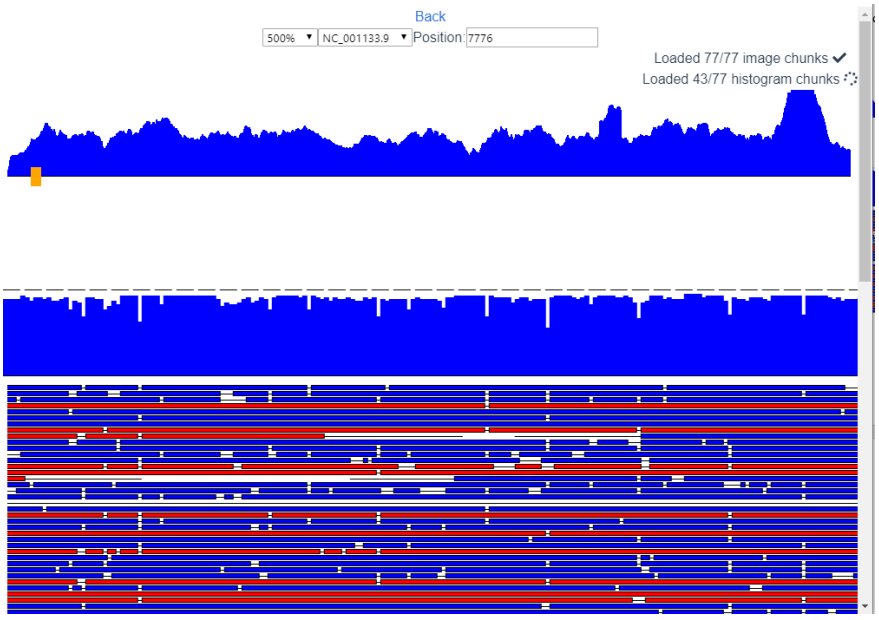
Slika 3.3 Vizualizacija

Klikom na narančasti pravokutnik se može promijeniti trenutna pozicija na bilo koji dio reference. To se također može postići upisivanjem u kućicu na vrhu ekrana ili se može kliknuti i vući na običnom histogramu za preciznije micanje (Slika 3.4 Micanje po referenci).



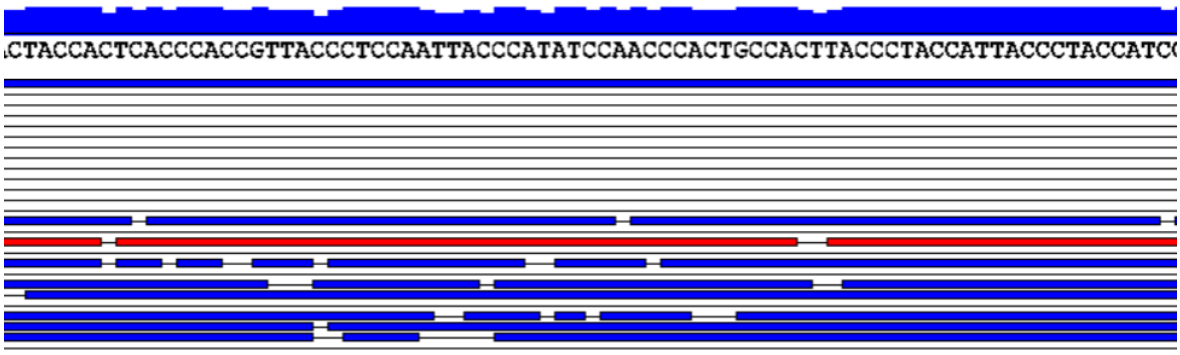
Slika 3.4 Micanje po referenci

Zumiranje se može obaviti u 4 razine: 100%, 200%, 500% i 1000%. Početno je 100% što označava da jedan nukleotid je širine 1 piksel, što znači da se može povećati do širine 1 nukleotida 10 piksela. Zumiranjem se slika proširuje samo po x osi čime dobivamo veću preglednost ali je slika toliko puta veća i treba više vremena da se učita (Slika 3.5 Zumirana vizualizacija).



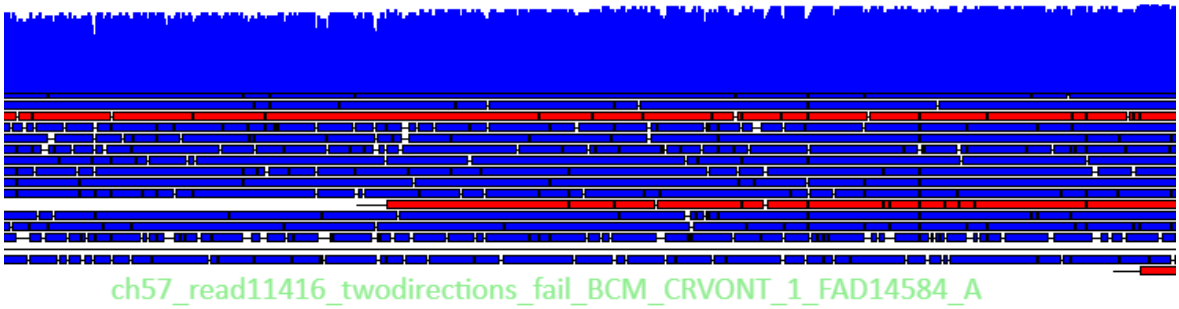
Slika 3.5 Zumirana vizualizacija

Ako se zumira do 1000% može se vidjeti pojedini nukleotid ispod histograma. Plava očitanja znače da je ta sekvenca jednaka kao i u referenci, a crvena znače da su reverzno komplementirana u odnosu na referencu (Slika 3.6 Prikaz pojedinog nukleotida).



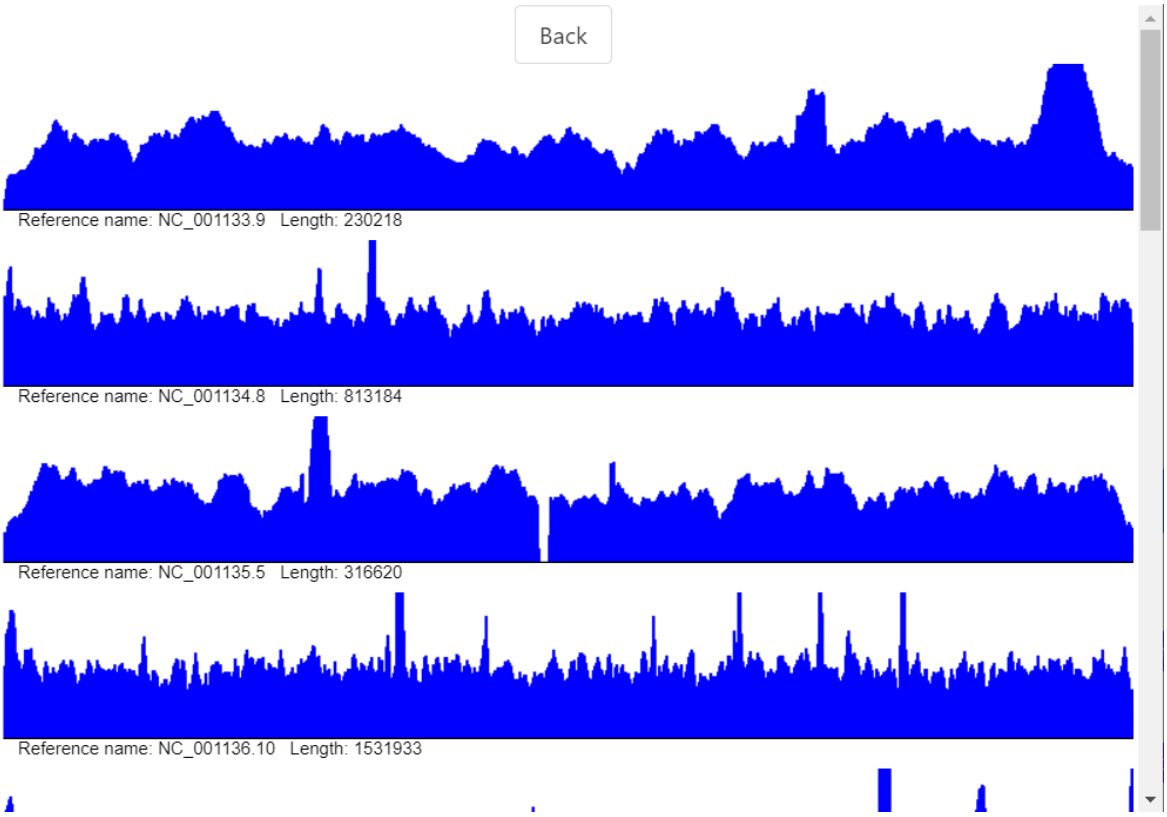
Slika 3.6 Prikaz pojedinog nukleotida

Ako očitanje ima liniju prije prvog pravokutnika ili poslije zadnjeg pravokutnika te linije označavaju *„soft clipping“*. Ako se pokazivačem prođe preko bilo kojeg očitanja pojavit će u blizini pokazivača svijetlo zeleni tekst s nazivom očitanja (Slika 3.7 Tooltip očitanja).



Slika 3.7 Tooltip očitanja

Klikom na gumb „Histograms view“ dolazi se na posljednji pogled. Počnu se učitavati reference redom i crtaju se histogrami skalirani na širinu ekrana jedan ispod drugoga. Platno je visine tako da su svi histogrami uvijek nacrtani pa ako ima više referenca pomoću klizača se može spustiti dolje da se vide ostale reference. Reference mogu biti različite dužine i samim time ne prikazuju iste lokacije pokrivenosti (Slika 3.8 Pregled histograma).



Slika 3.8 Pregled histograma

4. Moguća nadogradnja

Aplikacija ima mogućnosti za nadogradnju. Jedna od tih mogućnosti je učitavanje BAM datoteka. BAM datoteke su binarni ekvivalent SAM datoteka koje su kompresirane i time zauzimaju manje mjesta.

Slijedeća moguća nadogradnja je upravljanje velikim podacima. Aplikacija trenutno dijeli datoteku s očitanjima na manje datoteke grupirane po referencama, ali to ne mora uvijek značiti da će se dobiti datoteke koje su manje od radne memorije te bi se tu moglo dodatno optimizirati.

Korisničko sučelje zove pozadinski servis na način da pričeka da dođe prijašnji djelić slike prije nego pozove za sljedeći. Sustav bi se možda mogao ubrzati tako da poziva za nekoliko djelića odjednom. Kodiranje slika koristi veliku količinu radne memorije pa bi trebalo naći koliko se može slika kodirati u isto vrijeme da se ne iskoristi cijela radna memorija.

Slaba točka ove aplikacije je kodiranje slika na pozadinskom servisu. Dolaskom novih tehnologija aplikacija bi se mogla napraviti bez potrebe za kodiranjem slika. Ako postoji neko više platformsko rješenje u kojem se može crtati direktno bez kodiranja slika aplikacija bi bila ubrzana nekoliko puta.

Zaključak

Bioinformatika je područje gdje se pojavljuju ogromne količine podataka. Kako bi se ti podaci mogli obraditi trebaju se koristiti različiti alati da ih ljudi mogu shvatiti. Ova aplikacija je jedan od tih alata.

Koristeći ovu aplikaciju korisnik može učitati neka očitanja i gledati koliko dobro su reference mapirane, ako ima dijelova referenca koje nisu uopće mapirane i na koje mjesto reference je koje očitanje mapirano.

Aplikacija nije zamjena za profesionalne alate kao *IGV*, ali se može koristiti za neke jednostavne preglede podataka. Može se nastaviti nadograđivati novim vizualizacijama ili se dolaskom novih tehnologija mogu ubrzati performanse.

Literatura

[1] Šikić , M., Domazet-Lošo, M. *Skripta iz bioinformatike*. Zagreb, 2013.

[2] SAM FORMAT DOKUMENTACIJA, s Interneta, https://samtools.github.io/hts-specs/SAMv1.pdf

[3] KONVA DOKUMENTACIJA, s Interneta, https://konvajs.org/docs/index.html

[4] SKIASHARP DOKUMENTACIJA, s Interneta, https://docs.microsoft.com/en-us/dotnet/api/skiasharp?view=skiasharp-1.68.1

Sažetak

Vizualizacija podataka dobivenih sekvenciranjem DNA

Ovaj rad opisuje izradu aplikacije za vizualizaciju mapiranih očitanja dobivenih sekvenciranjem DNA. Na početku je objašnjeno što je sekvenciranje i mapiranje te primjer alata koji se koristi za vizualizaciju. U radu je objašnjena potpuna arhitektura aplikacije i kako su komponente aplikacije povezane. Opisana je funkcionalnost aplikacije te je objašnjeno kako se ona koristi. Opis arhitekture i funkcionalnosti aplikacije popraćen je objašnjenjem programskog koda i slikama korisničkog sučelja.

Ključne riječi: DNA sekvenciranje, mapiranje, C#, *ASP.NET Core*, vizualizacija

Summary

DNA Sequencing Data Visualisation

This paper describes the development process of an application for visualising mapped reading from DNA sequencing. The beginning of the paper explains what is sequencing and mapping and an example of a tool used for visualisation is given. The paper describes the complete architecture of the application and how its components are connected. The functionality of the application and how to use it is also described. Description of application architecture and functionality is accompanied by examples of source code and screenshots of user interface.

Keywords: DNA sequencing, mapping, C#, *ASP.NET Core*, visualisation

Privitak

Instalacija programske podrške

Instalacija aplikacije se radi u par koraka. Na računalu je potrebno imati instalirani Node.js, npm i .NET Core SDK. Također treba instalirati ElectronNET.CLI pomoću ove naredbe:

```
dotnet tool install ElectronNET.CLI -g
```

Prvo se treba klonirati git repozitorij pomoću naredbe:

```
git clone https://github.com/nosredek/ZavrsniRad.git
```

Nakon toga se pozicionira u mapu „ZavRad“ i tamo se izvrši naredba:

```
electronize build /target win
electronize build /target osx
electronize build /target linux
```

ovisno o operativom sustavu za koji želimo instalacijski paket. Nakon te naredbe stvori se instalacijski paket koji se pokrene i aplikacija je instalirana na sustav.