TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDU40LT

Sander Siniorg 134676IAPB

ANIMATSIOONID CAP TEOREEMI KOHTA HAJUSATE SQL-ANDMEBAASIDE NÄITEL

Bakalaureusetöö

Juhendaja: Erki Eessaar

Doktor

Dotsent

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Sander Siniorg

23.05.2016

Annotatsioon

Antud bakalaureusetöö eesmärk on luua õppematerjalina kasutatavad animatsioonid, mis selgitavad hajusate SQL-andmebaaside näitel CAP teoreemi, tutvustavad erinevaid süsteemi disaini võimalusi ja näitavad, kuidas need mõjutavad päringute ja andmemuudatuste tegemist. Animatsioonid on tehtud *JavaScripti* raamistiku *CreateJS* abil. Töö tulemusena valmib kolm animatsiooni.

Esmalt kirjeldatakse töös CAP teoreemi. Sellele järgneb ülevaade *CreateJS* raamistikust ja selle komponentidest ning ülejäänud kasutatud tehnoloogiatest. Töös sisaldub veel ülevaade animatsioonide veebilehe ülesehitusest, animatsioonide UML tegevusdiagrammi ning põhimõistete valdkonnamudelit. Lõpuks võrreldakse realiseeritud animatsioone olemasolevate sarnaste animatsioonidega.

Animatsioonid asuvad aadressil:

http://viktor.ld.ttu.ee/animatsioonid/animation_cap_theorem/

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 7 peatükki, 38 joonist, 1 tabel.

Abstract

Animations About the CAP Theorem in the Example of Distributed SQL Databases

The goal of this bachelor thesis is to create animations that one could use as a learning material that explains the CAP theorem in the example of distributed SQL-databases, gives an overview of various distributed system design options, and explain how it affects database queries and data changes. There will be three animations created with *JavaScript* framework *CreateJS*. The thesis is an example of implementing the design science methodology, which results with a new and essential artefact (in this case animations of CAP theorem) to solve a problem (how to get a better understanding of the theorem).

Firstly, we will give an overview of the CAP theorem. It is followed by a summary of CreateJS framework and its components along with other used technologies. In addition, we present a concept map to display relations between different technologies and animations. Thirdly, we write about how we designed and implemented the animations. This contains UML activity diagram of the animations and a concept map. We present an overview of the webpage structure and animation workflow with the questions that the animations will ask. Moreover, we present an overview of how one can easily modify the data and text in the animations. Finally, there will be a comparison of the implemented animations and other similar animations.

The animations are available on:

http://viktor.ld.ttu.ee/animatsioonid/animation_cap_theorem/

The thesis is in Estonian and contains 35 pages of text, 7 chapters, 38 figures, 1 table.

Lühendite ja mõistete sõnastik

ACID	*Atomicity, Consistency, Isolation, Durability* „**Atomaarsus, konsistentsus, isoleeritus, püsivus**. Andmebaasihaldurite puhul tehingutöötluse põhikarekteristikud. Ilma nendeta ei saa tagada andmebaasihalduri töökindlust“ [1].
API	*Application Programming Interface* „Arvuti operatsioonisüsteemiga või rakendusprogrammiga määratud reeglistik, mille alusel rakendusprogramm kasutab operatsioonisüsteemi või teise rakendusprogrammi teenuseid“ [1].
CAP	*Consistency, Availabilty, Partion Tolerance* Akronüüm, mille tähtedele vastavad sõnad: terviklikkus, käideldavus, jaotustaluvus; käesoleva töö poolt esitletavate animatsioonide teema.
CSS	*Cascadin Style Sheets* – keel, milles määratakse HTML-keelsete lehekülgede kujundust.
HTML	Keel, milles märgendatakse veebilehti [2].
Isend	*Instance* – programmi töö ajal loodud objekt, mis kuulub mingisse klassi ja on seega selle klassi isend [3].
Joonistuslõuend	*Canvas* HTML5 element, mis võimaldab renderdada dünaamilisi 2D kujundeid. Koosneb joonistamiseks mõeldud alast, millele *JavaScript* kood pääseb ligi läbi mitmete joonistamiseks mõeldud funktsioonide [4-5].
Kuvanimekiri	*Display list* – nimekiri vektorgraafilisi käske, millega määratletakse lõplik pilt väljundis.
Konteiner	Andmestruktuur, mille elementideks on kollektsioon objekte.
NoSQL	*Not only SQL* – 2010ndatel aastate alguses laialdaselt kasutusse tulnud üldnimi uutele andmebaasisüsteemidele, kus andmete haldamiseks ei kasutata SQL keelt ja andmete esitamiseks ei kasutata relatsioonilist mudelit.
Renderdamine	Renderdamine on vektorgraafika viimane protsess, kus luuakse ruumilisele stseenile vastav kahemõõtmeline pilt.
Sisuedastusvõrk	*Content Delivery Network (CDN)* Suur laiali jaotatud serverite võrk, mis on mõeldud kasutajatele tagama kiireid ja häid teenuse omadusi. See võimaldab

	kasutajatel kiiremini soovitud sisule ligi pääseda võrreldes sellega, kui sisu paikneks ühel serveril [1].
TTÜ	Tallinna Tehnikaülikool
Tween	*CreateJS* alamkomponendi *TweenJS* dokumentatsioonis kirjeldatud mõiste, mis tähendab mittelineaarse kiirusega animeerimist [6].
UML	*Unified Modeling Language* Unifitseeritud modelleerimiskeel. Üldotstarbeline (saab kasutada paljudeks ülesanneteks, sh animatsioonide kirjeldamiseks) visuaalne modelleerimiskeel tarkvara- ja infosüsteemide projekteerimiseks.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Bakalaureusetöö valikul lähtusin soovist luua midagi paljudele inimestele vajalikku. Selle eesmärgi täitmiseks tundus hea valik olema õppematerjalide loomine.

Mul on olnud palju kasu TTÜ õppeainete Andmebaasid I ja II läbimisel erinevatest selgitavatest animatsioonidest ja sellest lähtuvalt leidsin, et sarnase materjali loomine võiks olla hea valik. Valisin CAP teoreemi, sest mäletan, et selle selgitus teksti kujul oli esialgu üsna ebaselge ja ma oleksin väga tahtnud näha selgitavaid animatsioone. Seetõttu paistis selle teema kohta animatsioonide loomine olevat väga kasulik nii tulevastele üliõpilastele kui ka muidu andmebaaside huvilistele.

1.1 Taust ja probleem

Antud bakalaureusetöö on loodud õppematerjalina andmebaaside aineid õppivatele üliõpilastele ja teistele huvilistele. Valminud töö on kasulik kõigile hajutatud andmebaasidega kokku puutuvatele inimestele, sest CAP teoreemi mõistmine aitab paremini projekteerida hajutatud andmebaase.

Töö valmis autori õpingute ajal Tallinna Tehnikaülikoolis.

1.2 Ülesande püstitus

Antud töö käsitleb CAP teoreemi hajusate SQL-andmebaaside näitel. Töö käigus valmivad animatsioonid kolmest erinevast hajusate andmebaaside disainivalikust, mille puhul ootavad arendajad süsteemilt erinevaid omaduste kombinatsioone. Animatsioonide juures on selgitused teoreemi nõuete ja omaduste kohta ning küsimused kasutajale õpitavate teadmiste paremaks omandamiseks. Lisaks on näidatud päringute ja andmemuudatuste laused ning nendele vastavad andmebaasi vastused.

1.3 Metoodika

Käesolev töö on näiteks disainiteaduse (design science) metoodika rakendamisest, mille tulemuseks on uus ja vajalik artefakt (antud juhul animatsioonide komplekt) teatud probleemide lahendamiseks (antud juhul CAP teoreemi mõistmise parandamine) [7]. Loodud animatsioone testib ja neile annab tagasisidet andmebaasidega üldse mitte kokku puutunud inimene ja andmebaaside õppejõud, kes on ühtlasi ka töö juhendaja. Samuti võrdleb autor valminud animatsioone olemasolevate sarnaste animatsioonidega.

Animatsioonide loomiseks kasutatakse *Javascripti* raamistiku *CreateJS*. Raamistiku on integreeritud meetodid kujundite loomiseks ja nende animeerimiseks ning need on kasutatavad kõikides HTML5 protokolli toetavates veebilehitsejates.

Animatsioonid on ülesehitatud nii, et neis kuvatavaid andmeid oleks lihtne muuta. Kuvatavaid tekste on võimalik muuta eraldi failist. See tähendab, et andmeid on võimalik muuta ilma lehekülje enda programmikoodi muutmata.

1.4 Ülevaade tööst

Kõigepealt selgitatakse lahti CAP teoreemi põhimõtted. Sellele järgneb animatsioonide loomiseks kasutatud raamistiku *CreateJS* ja teiste kasutatud tehnoloogiate kirjeldus. Välja on toodud ka mõistekaart, mis näitab, kuidas iga kasutatud tehnoloogia on seotud animatsioonidega. Kolmandaks näidatakse animatsioonide skeeme ja ülesehitust, samuti veebilehe üldkuju ning kuidas on võimalik vajadusel animatsioonides kuvatavaid andmeid kerge vaevaga muuta. Viimasena on võrreldud olemasolevaid sarnaseid animatsioone antud töö käigus loodud animatsioonidega.

2 CAP Teoreem

Eric Brewer sõnastas aastal 2000 teoreemi, mida tuntakse nimede all CAP teoreem ja Breweri teoreem – hajusa andmebaasi korral on võimatu korraga anda järgnevat kolme garantiid: terviklikkus, käideldavus ja jaotustaluvus. CAP teoreemi kohaselt saab hajussüsteemis, mille osaks on kindlasti ka hajus andmebaas, olla samaaegselt tagatud maksimaalselt kaks omadust kolmest. S. Gilbert ja N. Lynch tõestasid selle teoreemi aastal 2002 [8].

Kuna arvutivõrgu tõrked on suure tõenäosusega paratamatud , siis hajus andmebaas peab kindlasti omama jaotustaluvuse omadust. Seega CAP teoreemi kohaselt tuleb hajusate andmebaaside kavandamise korral valida terviklikkuse ja käideldavuse vahel.

12 aastat hiljem, aastal 2012, on Brewer oma teoreemi leidnud liialt lihtsustava. Nimelt võib ühes süsteemis toimuda valik käideldavuse ja terviklikkuse vahel palju kordi ja erinevate allsüsteemide, operatsioonide, andmete või kasutajate korral võib valik käideldavuse ja terviklikkuse vahel olla erinev. See tähendab, et need omadused ei ole binaarsed, need võivad olla osaliselt täidetud [9].

2.1 Hajusate andmebaaside süsteemsed nõuded

Järgnevalt kirjeldatakse lühidalt CAP teoreemis nimetatavaid süsteemi omadusi.

2.1.1 Terviklikkus (*Consistency*)

Terviklikkus tähendab, et kõikides sõlmedes (komponent-andmebaasides) on klientidel lugemiseks alati samad andmed. Kui ühes sõlmes muudetakse andmeid, tuleb see muudatus teha ka kõigis teistes sõlmedes, et tagada kõikidele klientidele asukohast sõltumata lugemiseks samad andmed [10].

2.1.2 Käideldavus (*Availability*)

Iga töökorras oleva sõlme poole pöördumine peab õnnestuma. Sellest järeldub, et kõik kliendid peavad saama igal ajahetkel andmeid lugeda ja kirjutada, sõltumata sellest,

millise töökorras sõlme poole nad pöörduvad. Kuna võrgukatkestus võib kesta kuitahes kaua, ei või sõlmed vastamisega viivitada kuni katkestus kõrvaldatakse [10].

2.1.3 Jaotustaluvus (*Partion Tolerance*)

Jaotustaluvus tähendab, et ükski viga, peale täieliku võrgukatkestuse, ei tohi tingida vigu andmete lugemises või muutmises. Süsteem peab töötama korrektselt ka siis, kui võrguühendus osade sõlmede vahel ei tööta [10].

2.2 Hajusa andmebaasi disainivalikud

Järgnevalt kirjeldatakse CAP teoreemi poolt lubatud süsteemi omaduste paare. Neist võib mõelda kui disainivalikutest.

2.2.1 Terviklikkus + käideldavus

Kui andmebaasis soovitakse tagada andmete terviklikkust ja andmebaasi käideldavust, siis ei talu hajus andmebaas jaotuste teket (võrgukatkestusi erinevate sõlmede vahel). Selleks, et arvutivõrgu vead ei mõjutaks andmebaasi kasutamist, võib loobuda andmete hajutatud paigutamisest, kuid siis ei ole tegemist enam hajusa andmebaasiga [10].

2.2.2 Terviklikkus + jaotustaluvus

Kui andmebaasis soovitakse tagada andmete terviklikkust ning jaotustaluvust, siis ei saa tagada andmebaasi käideldavust, sest võrgukatkestuse korral tuleb oodata katkestuse parandamist, et andmemuudatused teistesse sõlmedesse üle kanda või loobuda üldse sellise muudatuse tegemisest. Esimesel juhul ei saa terviklikkuse saavutamiseni andmebaasi kasutada, sest andmed ei ole terviklikud. Teisel juhul jääb tegemata konkreetne andmemuudatus. Mõlemal juhul pole käideldavus tagatud [10].

2.2.3 Käideldavus + jaotustaluvus

Kui andmebaasis soovitakse tagada käideldavust ning jaotustaluvust, siis ei saa tagada andmebaasi terviklikkust. Andmebaas peab olema alati lugemiseks ning muutmiseks kasutatav (käideldavuse nõue) ning seda isegi siis, kui tekivad katkestused serverite vahelises võrguühenduses (jaotustaluvuse nõue). Eelnev on võimalik vaid juhul, kui süsteem ei pea tagama andmete terviklikkust igal ajahetkel, vaid võib seda teha viivitusega. Andmemuudatus ühes sõlmes kantakse üle teistesse sõlmedesse siis, kui selleks on võimalus (võrgukatkestused puudavad) [10].

3 Animatsioonide loomiseks kasutatud raamistikud

Kuigi programmeerimiskeel *JavaScript* sisaldab ka ise mõningaid animeerimiseks mõeldud meetodeid, sai antud töö käigus valminud animatsioonide loomiseks valitud spetsiaalselt animatsioonide jaoks mõeldud *JavaScripti* raamistik *CreateJS*. See raamistik osutus valituks, sest see võimaldab nii HTML5 joonistuslõuendile joonistamist kui ka kõikide elementide animeerimist. Samuti on seda autori hinnangul lihtne ja mugav kasutada.

3.1 CreateJS

CreateJS on üks HTML5 joonistuslõuendiga ja animatsioonidega töötamiseks mõeldud võimalusterohkeim *JavaScripti* raamistik. Selle alamkomponentidega on võimalik nii HTML5 joonistuslõuendile joonistada (*EaselJS*) kui ka neid objekte hiljem animeerida (*TweenJS*). Lisaks on veel komponendid heli lisamiseks lehele ja eellaadimiseks, kuid neid kahte antud töö raames ei kasutatud.

Kuna see raamistik põhineb *JavaScriptil*, siis ei pea kasutaja veebilehe vaatamiseks midagi oma arvutisse eraldi paigaldama. Kõik veebilehitsejad, mis toetavad *JavaScripti* ja HTML5, on võimelised antud lehte välja kuvama, kriteeriumitele mittevastavaid veebilehitsejaid on aga väga vähe [11].

Antud bakalaureusetöö tegemisel kasutati *EaselJS* versiooni 0.8.2 ja *TweenJS* versiooni 0.6.2.

3.1.1 EaselJS

JavaScripti teek, mis on mõeldud HTML5 joonistuslõuendiga töötamise hõlbustamiseks. Selleks on raamistikus loodud abistavad klassid, hierarhiline objektide sidumine ning veel näiteks kursori lugemise lihtsustamine [12].

3.1.2 TweenJS

JavaScripti teek, mis on mõeldud HTML5 ja *JavaScripti* elementide animeerimiseks. Seda saab kasutada iseseisvalt või integreerituna *EaselJS*-ga. See võimaldab muuta nii numbrilisi kui mittenumbrilisi parameetreid ja animatsioone omavahel siduda [6].

3.1.3 Kasutamine

CreateJS komponentide lisamine oma veebilehele on lihtne. Selleks tuleb veebilehe HTML programmikoodi päisesse lisada viide vastavat komponenti sisaldavale failile (Joonis 1). Vajaminevad failid võib laadida oma arvutisse või serverisse ja seejärel viidata neile või siis viidata otse sisuedastusvõrgule.

```
<script src="https://code.createjs.com/easeljs-0.8.2.min.js"></script>
<script src="https://code.createjs.com/tweenjs-0.6.2.min.js"></script>
```

Joonis 1.CreateJS lisamine.

Lisatud teekide uuendamine on samuti väga lihtne. Tuleb olemasolevas *script* märgendi sees asendada *src* parameetri väärtust. Teise variandina võib ka vana faili uuega üle kirjutada.

3.1.4 Meetodid

CreateJS komponendid sisaldavad mitmeid meetodeid, millega on võimalik luua animatsioone. Järgnevalt on välja toodud animatsioonide loomiseks kasutatud meetodid.

enableMouseOver võimaldab sisse ja välja lülitada kursori liikumise ja vajutuste lugemise CreateJS teegi poolt (Joonis 2) [12]. Kasutati, et lugeda kursori vajutusi vastusevariantide vajutamisel.

```
.enableMouseOver([sagedus])
```

Joonis 2. Meetod *enableMouseOver*.

addChild lisab elemendi kuvanimekirja (Joonis 3) [12]. Kasutati, et lisada elemente juurkonteinerisse *Stage* ja seeläbi kasutajale nähtavaks teha.

```
.addChild([element])
```

Joonis 3. Meetod *addChild*.

addEventListener lisab määratud tüübiga sündmuse kuulaja (Joonis 4) [12]. Kasutati, et lugeda kursori vajutusi vastusevariantide vajutamisel.

```
.addEventListener([tüüp], [kuulaja funktsioon])
```

Joonis 4. Meetod *addEventListener*.

beginFill alustab graafikaobjekti täitmist valitud värviga (Joonis 5) [12]. Kasutati, et joonistada kujundeid, mis on seest värvitud.

```
.beginFill([värv])
```

Joonis 5. Meetod *beginFill.*

drawRect joonistab ristküliku määratud asukohta etteantud laiuse ja kõrgusega (Joonis 6) [12]. Kasutati ristkülikute joonistamiseks animatsioonides.

```
.drawRect([x koordinaat], [y koordinaat], [laius], [kõrgus])
```

Joonis 6. Meetod *drawRect.*

beginStroke alustab kriipsu tõmbamist valitud värviga (Joonis 7) [12]. Kasutati joonte joonistamiseks.

```
.beginStroke([värv])
```

Joonis 7. Meetod *beginStroke.*

setStrokeStyle määrab joone paksuse (Joonis 8) [12]. Kasutati joonte joonistamiseks.

```
.setStrokeStyle([paksus])
```

Joonis 8. Meetod *setStrokeStyle*.

moveTo liigutab joonistuspunkti määratud asukohta (Joonis 9) [12]. Kasutati joonte alguspunkti määramiseks.

```
.moveTo([x koordinaat], [y koordinaat])
```

Joonis 9. Meetod *moveTo*.

lineTo tõmbab joone *moveTo* meetodiga määratud joonistuspunktist määratud asukohani (Joonis 10). Eelnevalt peab olema välja kutsutud meetod *moveTo* [12]. Kasutati joonte joonistamiseks määratud sihtpunkti.

```
.lineTo([x koordinaat], [y koordinaat])
```

Joonis 10. Meetod *lineTo.*

drawCircle joonistab määratud raadiusega ringi (Joonis 11) [12]. Kasutati ringide joonistamiseks.

```
.drawCircle([x koordinaat], [y koordinaat], [radius])
```

Joonis 11. Meetod *drawCircle.*

setStrokeDash määrab või eemaldab joone mustri (Joonis 12) [12]. Kasutati punktiirjoone joonistamiseks, et märkida võrgukatkestuskoht.

```
.setStrokeDash([segmente kirjeldav massiiv])
```

Joonis 12. Meetod *setStrokeDash*.

removeChild eemaldab määratud elemendi kuvanimekirjast (Joonis 13) [12]. Kasutati elementide eemaldamiseks joonistuslõuendilt.

```
.removeChild([element])
```

Joonis 13. Meetod *removeChild.*

Tween.get tagastab uue *tween* isendi määratud objektist (Joonis 14) [6]. Kasutati, et animeerida elementi. Kõigepealt tuleb kasutada *get* meetodit, et element kätte saada ja siis saab seda *to* meetodiga animeerida.

```
.get([object], [omadused])
```

Joonis 14. Meetod *Tween.get.*

Tween.wait võimaldab animatsioonil lasta määratud aja oodata enne käivitumist või jätkamist (Joonis 15) [6]. Kasutati, et lasta animatsioonil oodata, enne jätkumist.

```
.wait([kestvus millisekundites)])
```

Joonis 15. Meetod *Tween.wait.*

Tween.to võimaldab *tween* objekti omadusi muuta soovitud väärtusteni määratud aja jooksul. Numbrilised väärtused muudetakse (*tweenitakse*) sujuvalt aja jooksul soovitud väärtuseni, mittenumbrilised väärtused muudetakse ära määratud aja lõppedes (Joonis 16) [6].

```
.to([massiiv omaduste ja nende soovitud väärtustega], [kestvus
            millisekundites], [tween funktsioon])
```

Joonis 16. Meetod *Tween.to.*

Ease.getPowInOut kohendatav eksponentsiaalne funktsioon, näiteks sisend 3 tagastab kuupfunktsiooni (Joonis 17) [6].

```
.getPowInOut([pow])
```

Joonis 17. Meetod *Ease.getPowInOut.*

Ticker.on võimaldab mõne objekti panna kuulama tsentraliseeritud *tick* sündmust, mille vaikimisi sagedus on 30 sündmust sekundis (Joonis 18). Tavaliselt pannakse juurkonteiner *Stage* kuulama *tick* sündmust, et kuvada joonistuslõuendil kogu aeg ajakohane kuvanimekiri [12].

```
Ticker.on([tüüp], [objekt])
```

Joonis 18. Meetod *Ticker.on.*

3.1.5 Abistavad klassid

Järgnevalt on välja toodud animatsioonide loomisel kasutatud *CreateJS* teegi abistavad klassid.

Stage on juurtaseme konteiner kuvanimekirja elementide jaoks. Iga kord kui *tick* meetod välja kutsutakse, renderdatakse kogu temas sisalduv kuvanimekiri ettenähtud joonistuslõuendile (Joonis 19) [12].

```
createjs.Stage([canvas])
```

Joonis 19. Abistav klass *Stage*.

Ticker pakub tsentraliseeritud *tick* või muud määratud intervalliga sündmuse edastamist. Kuulajad saavad hakata jälgima neid sündmusi, et kindla intervalliga mingit tegevust korrata [12].

```
createjs.Ticker.on([tüüp], [objekt])
```

Joonis 20. Abistav klass *Ticker*.

Graphics klass võimaldab ligipääsu lihtsasti kasutatavale API-le, millega on võimalik luua vektorgraafikat (Joonis 21). Seda on võimalik kasutada eraldiseisvana või *EaselJS* raamistiku osana, et lisada vektorgraafikat *EaselJS* kuvanimekirja [12].

```
createjs.Graphics()
```

Joonis 21. Abistav klass *Graphics*.

Shape klass võimaldab lisada vektorgraafikat kuvanimekirja. See kasutab *Graphics* isendi API-t võimaldamaks ligipääsu vektorgraafika joonistamiseks vajalikele meetoditele (Joonis 22). Ühte *Graphics* isendit on võimalik jagada mitme *Shape* isendi vahel kuvamaks sama vektorgraafikat erinevatel positsioonidel ja kujudel [12].

```
createjs.Shape([graphics isend])
```

Joonis 22. Abistav klass *Graphics*.

Ease klass pakub hulga erinevaid funktsioone, mida kasutada *TweenJS* raamistikuga. Enamasti antakse *Ease* meetodid otse *TweenJS* meetoditele sisendiks [6].

Container on hierarhiline kuvanimekiri, mis võimaldab mitu isendid liita kokku ühe konteineri alla (Joonis 23). Nii on võimalik kõiki konteineris olevaid isendeid koos animeerida ja nende omadusi muuta, samas jääb võimalus ka neid isendeid teistest sõltumatult muuta [12].

```
createjs.Container()
```

Joonis 23. Abistav klass *Container*.

Võimaldab luua ühe või mitu rida dünaamilist teksti kuvanimekirja (Joonis 24) [12]. Seda kasutati animatsioonides selgitavate tekstide lisamiseks. Seda klassi kasutades oli lihtsam tekste joonistuslõuendile paigutada ja animeerida, kui oleks olnud HTML elemente.

```
createjs.Text([tekst], [font], [värv])
```

Joonis 24. Abistav klass *Text.*

4 Muud kasutatud raamistikud ja tehnoloogiad

Lisaks *CreateJS* raamistikule, millega sai joonistatud kogu graafika ja tehtud animatsioonid, kasutatakse antud bakalaureusetöö tulemusena loodava animatsioonide veebilehe loomisel ka *Bootstrapi* raamistikku.

4.1 Bootstrap

Bootstrap on üks enimlevinumaid HTML, CSS ja *JavaScripti* raamistikke, mis võimaldab luua kerge vaevaga hea väljanägemisega kohalduvaid veebilehti. See sisaldab palju valmis komponente, mida on väga mugav ja lihtne kasutada.

Bootstrap sisaldab ka ise mõningaid animeerimiseks mõeldud meetodeid, kuid antud bakalaureusetöös neid ei kasutatud, sest *CreateJS* raamistik pakub animatsioonide loomiseks palju rohkem võimalusi.

Antud bakalaureusetöö koostamisel kasutati *Bootstrapi* versiooni 3.3.6.

4.1.1 Kasutamine

Bootstrapi lisamiseks veebilehele tuleb HTML koodi päisesse lisada viited *Bootstrapi* failidele. *Bootstrap* vajab korrektseks toimimiseks ka seda, et veebilehele oleks lisatud *jQuery* raamistik, mis tuleb lisada HTML koodi päisesse enne *Bootstrapi JavaScripti* faili. *Bootstrapi* jaoks on vajalik kaks faili, üks CSS-i ja teine *JavaScripti* jaoks (Joonis 25).

```
<link href="css/bootstrap.min.css" rel="stylesheet">
<script src="js/jquery-2.2.0.min.js"></script>
<script src="js/bootstrap.min.js"></script>
```

Joonis 25. Bootstrapi lisamine.

4.2 jQuery

jQuery on *JavaScriptiga* töötamise hõlbustamiseks mõeldud teek. Seda on väga lihtne kasutada ja see teeb palju lihtsamaks mitmete *JavaScript* funktsioonide rakendamise.

jQueryt kasutava veebilehe kuvamiseks ei pea kasutaja oma arvutisse eraldi midagi paigaldama. Kõik veebilehitsejad, mis toetavad *JavaScripti*, toetavad ka *jQueryt*. Selliseid veebilehitsejaid, mis seda ei teeks, on väga vähe [11].

Antud bakalaureusetöös kasutati *jQuery* versiooni 2.2.0

4.2.1 Kasutamine

jQuery raamistiku lisamine oma veebilehele on lihtne. Selleks tuleb veebilehe HTML programmikoodi päisesse lisada viide vastavale failile (Joonis 26). Vajamineva faili võib laadida oma arvutisse või serverisse ja seejärel viidata sellele. Samuti võib viidata ka otse sisuedastusvõrgule.

```
<script src="js/jquery-2.2.0.min.js"></script>
```

Joonis 26. jQuery lisamine.

4.2.2 Meetodid

jQuery raamistik sisaldab palju meetodeid, kuid antud töös kasutati neist ainult ühte, JSON andmefaili lugemiseks. *getJSON* meetod laadib JSON kodeeringus andmed *HTTP GET* päringuga (Joonis 27) [13].

```
.getJSON([url], [funktsioon])
```

Joonis 27. getJSON meetod.

4.3 JSON

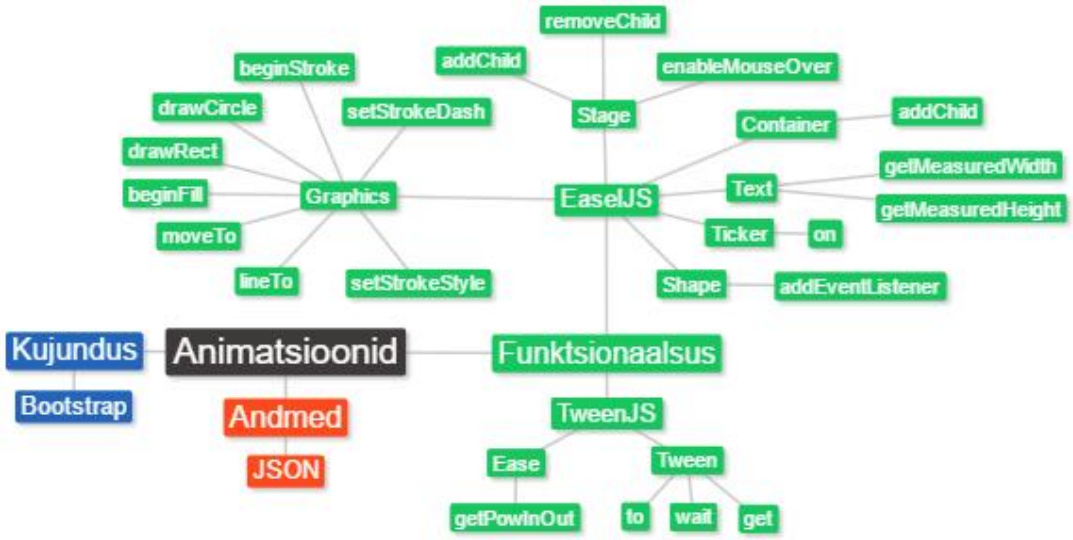
JSON on lihtsustatud andmevahetusvorming, mis põhineb *JavaScripti* programmeerimiskeele alamhulgal, kuid on sellest sõltumatu. Seda kasutatakse eelkõige *Ajax* päringutes XML-i asemel [14].

4.3.1 Kasutamine

Antud töö raames kasutati JSON-t animatsioonide andmete hoidmiseks eraldi failis, et neid oleks hiljem lihtne muuta. Et JSON andmed oleksid teiste programmeerimiskeelte (antud töö puhul *Javascripti)* poolt kergesti loetavad, tuleb failis esitada süntaktiliselt korrektne JSON-i sõne. JSON koosneb nimi-väärtuste paaride kogumitest ning järjestatud väärtuste jadadest [14].

4.4 Seosed kasutatud raamistike ja tehnoloogiate ning animatsiooni elementide vahel

Järgnevalt on esitatud kasutatud raamistike, tehnoloogiate ning animatsiooni elementide vahelised seosed. Mõistekaardil on kasutatud eri värve erinevate animatsioonide osade grupeerimiseks: rohelisega on märgitud funktsionaalsus, punasega andmed ja sinisega kujundus (Joonis 28).



Joonis 28. Seosed raamistike, tehnoloogiate ja animatsiooni elementide vahel.

5 Animatsioonide projekteerimine ja realiseerimine

Selles peatükis on kirjeldatud animatsioonide projekteerimist ja realiseerimist. Animatsioonid on projekteeritud autori kogemuste põhjal ainet „Andmebaasid II“ kuulates. Autoril on meeles, et CAP teoreemi loengus jäi kergelt arusaamatuks, kuidas erinevad hajusa andmebaasi disainivalikud mõjutavad kliendi ja andmebaasi vahelist andmevahetust, millistel juhtudel toimuvad päringud ja millised andmed tagastatakse. Sellest lähtuvalt on animatsioonid loodud konkreetsetele näidetele.

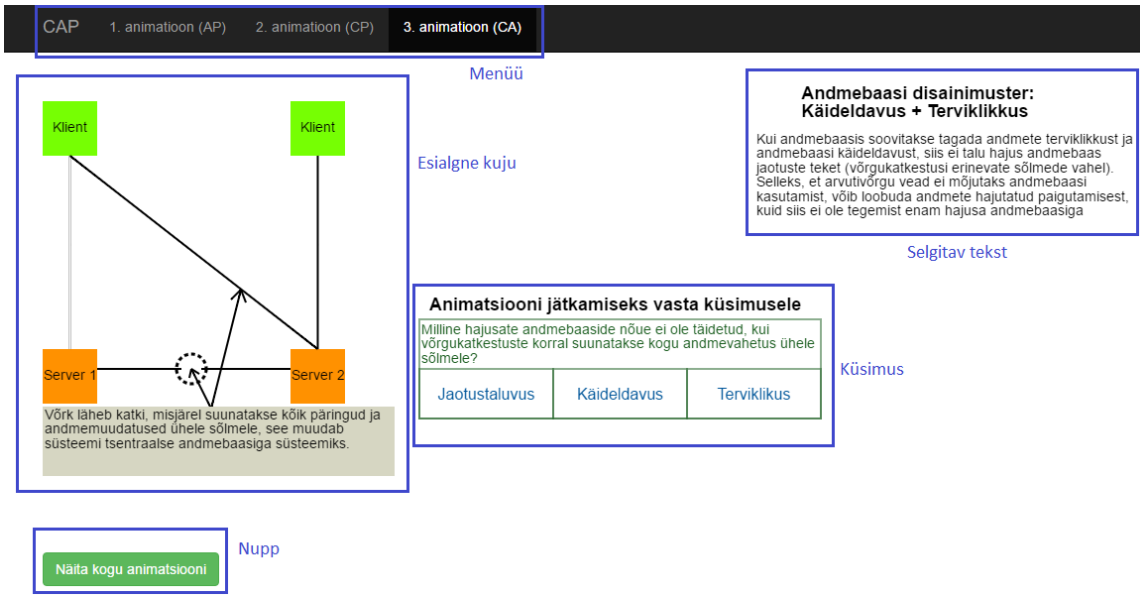
Animatsioonides kasutatavad näitetabelid, SQL laused ja kogu animatsioonide ülesehitus on käesoleva bakalaureusetöö autori looming.

5.1 Animatsioonide seesmine ülesehitus käideldavuse ja terviklikkuse animatsiooni näitel

Kõik animatsioonid sai loodud ainult *CreateJS* raamistiku kasutades. Kõik animatsioonides nähtavad elemendid on joonistatud kasutades eelnimetatud raamistiku komponenti *EaselJS* ja animeeritud komponenti *TweenJS* kasutades. Seda on tehtud põhjusel, et nii osutus lihtsamaks kõikide elementide omavaheline paigutamine.

5.1.1 Üldvaade

Kõikide loodud animatsioonide veebilehekülje on ülesehitusel väga sarnased. Lehekülje paremas servas on kirjeldatud andmebaasi disainivalik, mille kohta konkreetne animatsioon käib. Juurde on lisatud veel lühike selgitav tekst, mida selline disainivalik endast kujutab. Kui animatsiooni ülespanija soovib teksti muuta, siis tuleb JSON andmefailis muuta *„sideText“* atribuudi väärtust (Joonis 35). Animatsioon käivitub leheküljele minnes automaatselt ja alustatakse võrgu ülesehituse joonistamisega ja esimese küsimuse küsimisega (Joonis 29).



Joonis 29. Lehekülje esialgne kuju.

Järgnevalt on kujutatud, kuidas üks klient soovib andmebaasi tabelisse lisada ühte rida. Seejärel pärivad mõlemad kliendid kogu eelnimetatud tabeli sisu ja näidatakse, millised andmed neile tagastatakse. Iga tegevusega, mis animatsioonis toimub, kaasneb lühike selgitav tekst ja tegevuse kohta käiv küsimus. Animatsioon jätkub iga kord kui kasutaja on küsimusele õigesti vastanud. Kasutaja saab vastuse varianti mitu korda, kuni on andnud õige vastuse. Kui animatsiooni ülespanija soovib küsimuse teksti muuta, siis tuleb JSON andmefailis muuta *„questions“* atribuudi vastava elemendi väärtust (Joonis 35). Elemendid on jadas samas järjekorras, milles esituvad küsimused. Õige vastusevariandi muutmiseks tuleb JSON andmefailis muuta *„answers“* atribuudi vastava elemendi väärtust (Joonis 35). Ka selles jadas on elemendid järjestatud selles järjekorras, milles animatsioonides kuvatakse küsimused. Küsimuste eesmärk on kasutajal kinnistada teadmisi CAP teoreemist ning samuti aitab see igal kasutajal valida omale sobiv tempo animatsiooni läbimiseks.

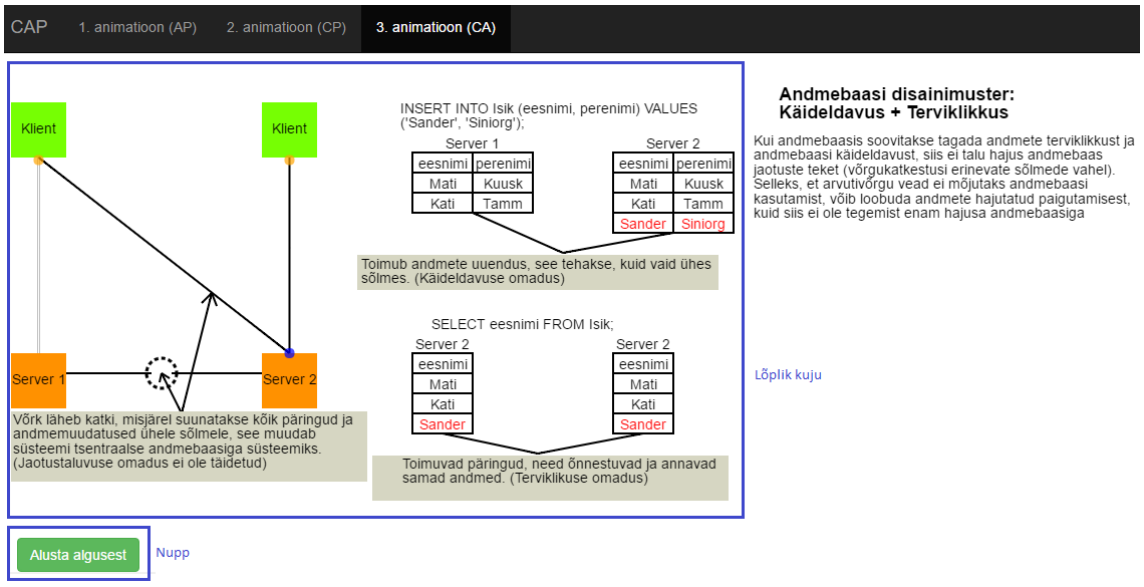
Tabel 1 esitab animatsioonides esitatavad küsimused ja nende õiged vastused.

Tabel 1. Animatsioonides esitatavad küsimused, koos vastustega.

Küsimus	**Vastus**
Milline hajusate andmebaaside nõue ei ole täidetud, kui kõikidele klientidele vastab ainult üks sõlm (server)?	Jaotustaluvus
Millise hajusate andmebaaside nõudega on tegemist, kui muudatus tehakse koheselt?	Käideldavus

Küsimus	Vastus
Millise hajusate andmebaaside nõudega on tegemist, kui kõikidele klientidele on lugemiseks samad andmed?	Terviklikkus
Milline hajusate andmebaaside nõue on täidetud, kui võrgukatkestuse korral jätkavad kõik sõlmed klientidele vastamist?	Jaotustaluvus
Milline hajusate andmebaaside nõue ei ole täidetud, kui kõikidele klientidele ei ole lugemiseks samad andmed?	Terviklikkus
Milline hajusate andmebaaside nõue on täidetud kui võrgukatkestuse korral oodatakse muudatuse tegemisega andmebaasis kuni võrgukatkestus on parandatud?	Terviklikkus
Milline hajusate andmebaaside nõue ei ole täidetud, kui päring andmebaasi ei õnnestu igal ajahetkel koheselt?	Käideldavus
Milline hajusate andmebaaside nõue ei ole täidetud, kui andmemuudatusi pole igal ajahetkel võimalik sooritada?	Käideldavus

Animatsioon on võimalik käivitada ka kohe algusest lõpuni vajutades nuppu „Näita kogu animatsiooni“. Siis kasutajale küsimusi ei esitata, vaid animatsioon jookseb algusest lõpuni. Animatsiooni lõppedes asendub „Näita kogu animatsiooni“ nupp „Alusta algusest“ nupuga (Joonis 30).

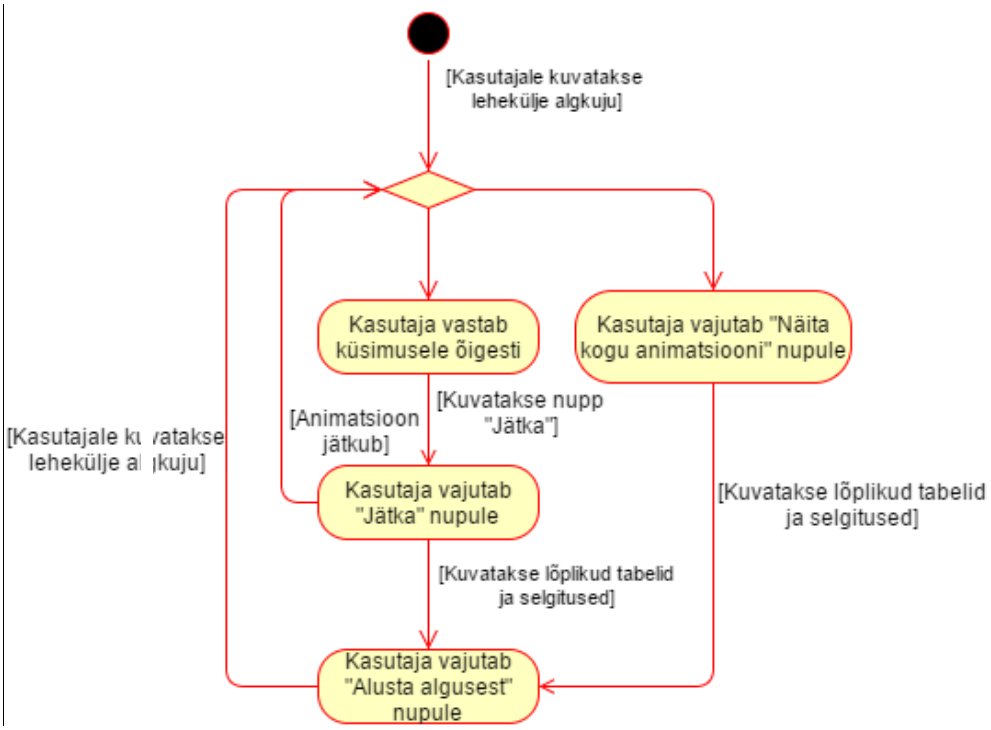


Joonis 30. Lehekülje lõplik kuju.

Animatsioone on testitud järgnevates veebilehitsejates: Google Chrome (versioon 50.0.2661.102), Internet Explorer (versioon 11), Mozilla Firefox (versioon 46.0) ja Safari (versioon 9.1). Kõigis nendes veebilehitsejates töötasid animatsioonid korrektselt.

5.1.2 Tegevusdiagramm

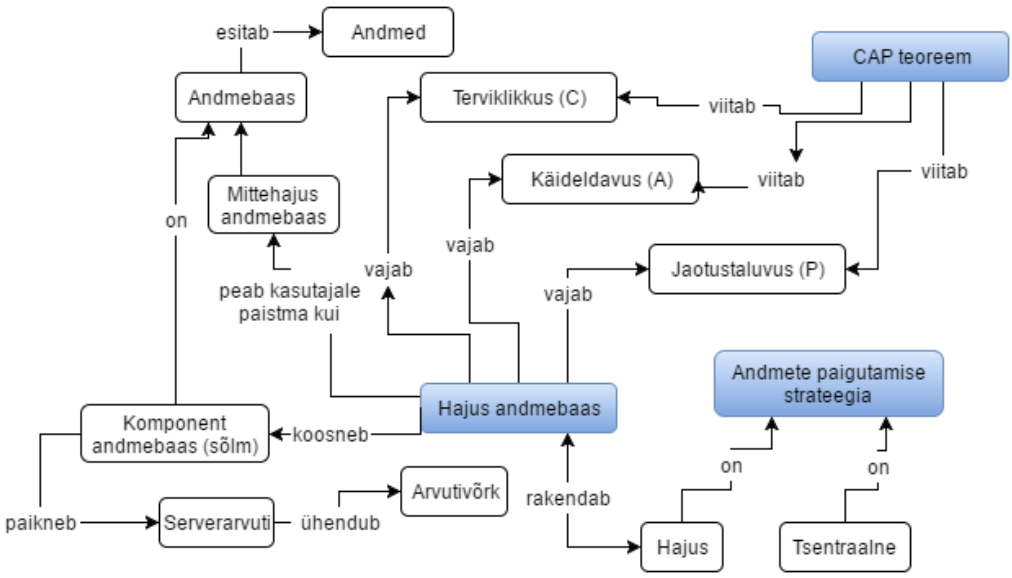
Järgnevalt on näidatud terviklikkuse ja jaotustaluvuse animatsiooni UML tegevusdiagramm (Joonis 31). Sama skeemi järgivad üldiselt ka teised animatsioonid.



Joonis 31. Käesoleva töö animatsioonide üldine tegevusdiagramm.

5.1.3 Valdkonnamudel

Järgnevalt esitatakse animatsioonides esinevad põhilised hajusate andmebaasidega seotud mõisted valdkonnamudelina (Joonis 32).



Joonis 32. Animatsioonides esinevate mõistete valdkonnamudel.

5.1.4 Animatsiooni häälestamine

Animatsioonides kuvatavad tabelid ja SQL-laused on veebilehele loetud eraldi failist, kus andmed on JSON vormingus. Selle lähenemise idee on saadud Metsvahi bakalaureusetööst [15]. Andmefaili nimi on vastavalt animatsioonile, kas ca.json, ap.json või cp.json. Andmefaili kasutades on võimalik muuta veebilehel kuvatavaid SQL lauseid, esitatavaid küsimusi ja vastuseid, selgitavaid tekste ning tabelites esitatavaid andmeid ilma lehekülje programmikoodi muutmata. Järgnevalt on toodud näide, kuidas andmefaili muutmine mõjutab animatsioonis esinevas tabelis kuvatavaid andmeid. Andmefailis muudeti *„rows“* atribuudi väärtust. Joonisel 33 on esitatud kuvatõmmis animatsioonist enne muudatuse tegemist. Joonisel 34 on esitatud kuvatõmmis animatsioonist peale muudatuse tegemist. Joonisel 35 on esitatud andmefailis olevad andmed enne muudatuse tegemist ja joonisel 36 peale muudatuse tegemist.

Server 1

eesnimi	perenimi
Mati	Kuusk
Kati	Tamm

Joonis 33. Kuvatõmmis animatsioonist enne muudatuse tegemist.

Server 1

eesnimi	perenimi
Juhan	Kadak
Mari	Mets

Joonis 34. Kuvatõmmis animatsioonist peale muudatuse tegemist.

```
{"sqls": ["INSERT INTO Isik (eesnimi, perenimi) VALUES ('Sander', 'Siniorg');",
          "SELECT eesnimi FROM Isik;"],
"headers" : ["Klient", "Server"],
"columns" : ["eesnimi", "perenimi"],
"rows" : [["Mati", "Kuusk"], ["Kati", "Tamm"]],
"extraRow" : ["Sander", "Siniorg"],
"explanations" : ["Kõik sõnumid ühest võrgusõlmede rühmast teise lähevad kaotsi.",
                  "Toimub andmete uuendus, oodatakse, et teha see kõikides sõlmedes.",
                  "Toimub päring, see ei õnnestu, sest andmed ei ole terviklikud.",
                  "Jääb tegemata konkreetne andmemuudatus, sest seda pole võimalik kanda e
"questions" : ["Milline hajusate andmebaaside nõue ei ole täidetud, kui võrgukatkestuste k
               "Milline hajusate andmebaaside nõue on täidetud, kui võrgukatkestuse korral
               võrgukatkestus on parandatud?",
               "Milline hajusate andmebaaside nõue ei ole täidetud, kui päring andmebaasi
               "Milline hajusate andmebaaside nõue ei ole täidetud, kui andmemuudatusi pol
"answers" : [1, 3, 2, 2],
"sideText" : "Kui andmebaasis soovitakse tagada andmete terviklikkust ning jaotustaluvust,
võrgukatkestuse korral tuleb oodata katkestuse parandamist, et andmemuudatused teistesse s
tegemisest. Esimesel juhul ei saa terviklikkuse saavutamiseni andmebaasi kasutada, sest an
konkreetne andmemuudatus. Mõlemal juhul pole käideldavus tagatud."}
```

Joonis 35. Andmed enne muudatuse tegemist.

```
{"sqls": ["INSERT INTO Isik (eesnimi, perenimi) VALUES ('Sander', 'Siniorg');",
          "SELECT eesnimi FROM Isik;"],
"headers" : ["Klient", "Server"],
"columns" : ["eesnimi", "perenimi"],
"rows" : [["Juhan", "Viidik"], ["Mari", "Mets"]],
"extraRow" : ["Sander", "Siniorg"],
"explanations" : ["Kõik sõnumid ühest võrgusõlmede rühmast teise lähevad kaotsi.",
                  "Toimub andmete uuendus, oodatakse, et teha see kõikides sõlmedes.",
                  "Toimub päring, see ei õnnestu, sest andmed ei ole terviklikud.",
                  "Jääb tegemata konkreetne andmemuudatus, sest seda pole võimalik kanda e
"questions" : ["Milline hajusate andmebaaside nõue ei ole täidetud, kui võrgukatkestuste k
               "Milline hajusate andmebaaside nõue on täidetud, kui võrgukatkestuse korral
               võrgukatkestus on parandatud?",
               "Milline hajusate andmebaaside nõue ei ole täidetud, kui päring andmebaasi
               "Milline hajusate andmebaaside nõue ei ole täidetud, kui andmemuudatusi pol
"answers" : [1, 3, 2, 2],
"sideText" : "Kui andmebaasis soovitakse tagada andmete terviklikkust ning jaotustaluvust,
võrgukatkestuse korral tuleb oodata katkestuse parandamist, et andmemuudatused teistesse s
tegemisest. Esimesel juhul ei saa terviklikkuse saavutamiseni andmebaasi kasutada, sest ar
konkreetne andmemuudatus. Mõlemal juhul pole käideldavus tagatud."}
```


Joonis 36. Andmed peale muudatuse tegemist.

Soovi korral on võimalik muuta ka SQL lauseid, selleks tuleb teha JSON andmefailis muudatused „*sqls*“ atribuudi väärtuses. Soovides ise lisada reavahetus tuleb sõnesse lisada märkide kombinatsioon „*\n*“, muidu poolitakse read vastavalt leheküljе kujundusele. Samuti on võimalik muuta animatsiooniga kaasnevaid selgitavaid lauseid muutes andmefailis „*explanations*“ atribuudi väärtust, tabelisse lisanduva rea andmeid muutes atribuudi *„extraRow“* väärtust ning küsimusi ja õigeid vastuseid muutes vastavalt *„questions“* või *„answers“* atribuudi väärtust. Õige vastus esitatakse numbriliselt, number 1 vastab esimesele vastusevariandile jne.

6 Võrdlus teiste animatsioonidega

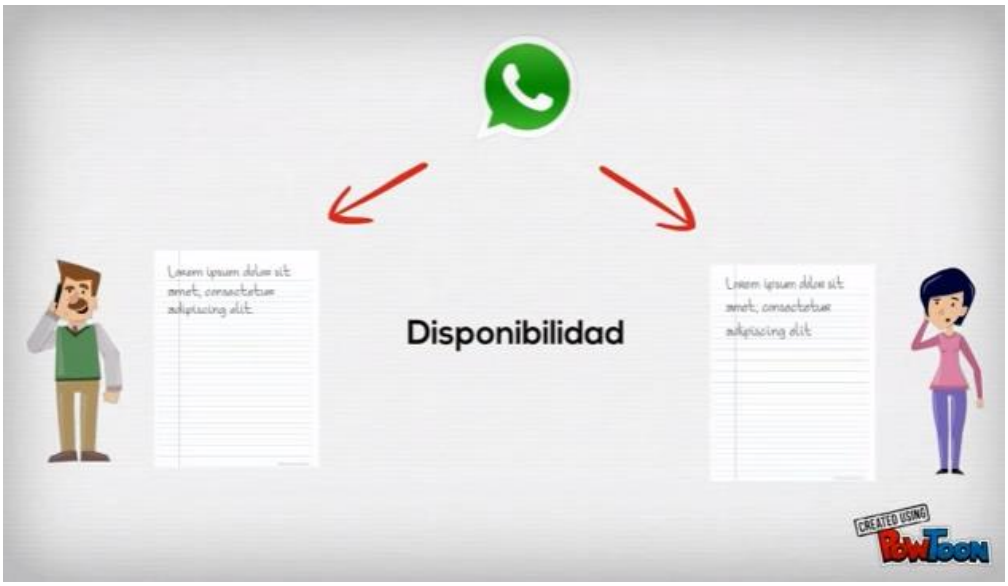
CAP teoreemi kohta käivaid olemasolevaid animatsioone on väga vähe. Mõned üksikud, mis leiti, olid küllaltki pealiskaudsed ja selgitasid ainult, mida iga CAP teoreemis nimetatud hajusate andmebaaside nõudega mõeldud on. Väga keeruline oli neist mõista, miks saab kolmest omadusest kehtida maksimaalselt kaks ja kuidas täpselt toimub erinevate andmebaasi disainivalikute korral andmevahetus kliendi ja andmebaasi vahel. Antud töö käigus realiseeritud animatsioonide loomisel lähtuti just sellest, et tuua välja väga selgelt, miks kaks omadust kehtivad ja üks ei kehti ning millal täpselt lubatakse klientidel andmetes muudatusi teha ja milliseid andmeid neil on võimalik andmebaasist küsida.

6.1 Eestikeelsed animatsioonid

Kasutades Google otsingumootoris otsingusõnu „CAP teoreem animatsioon“ või „Brewer teoreem animatsioon“ ei leitud ühtegi CAP teoreemi kohta käivat eestikeelset animatsiooni. See tõestab veelgi, miks antud bakalaureusetöö raames valminud animatsioonidest võiks Eesti tudengitele palju kasu olla, sest emakeelseid animatsioone vaadates on seda olulist andmebaaside teoreemi kindlasti kergem omandada.

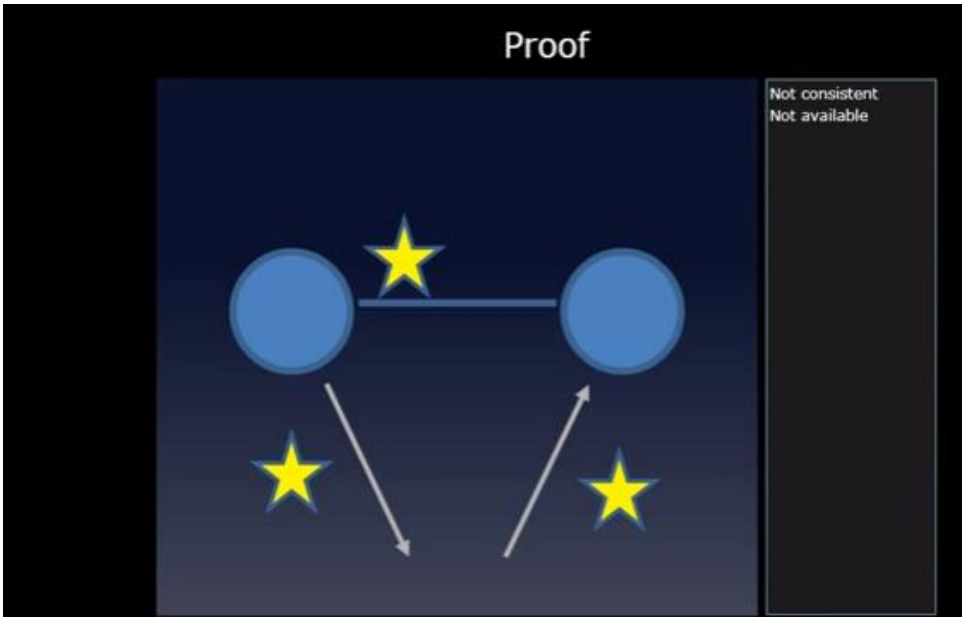
6.2 Inglisekeelsed animatsioonid

Kasutades Google otsingumootoris otsingusõnu „Cap theorem animation“ või „Brewer theorem animation“ leiti üks animatsiooni video kujul ja õpetavad videod, mida võib mingil määral animatsioonideks lugeda. Animatsioon on liikumise simulatsioon, mis saavutatakse hulga piltide või kaadrite näitamisega [1]. Esimene animatsiooni video on leitav aadressil https://www.youtube.com/watch?v=elWq2-RdfRk (Joonis 37). Video on üsna lõbusa ülesehitusega ja eluliste näidetega, aga hispaania keeles, mis teeb sellest arusaamise keskmisele Eesti tudengile küllalt raskeks. Nii palju kui autor pildist välja luges, püüti selgitada, kas ja kui palju kõnesid üks uudistekontor suudab vastu võtta ja kuidas see mõjutab uudisloo andmeid. Siiski jääb selgusetuks, kuidas täpselt on iga CAP teoreemi omadus seotud konkreetselt andmebaasi päringute ja andmetega.



Joonis 37. Teorema CAP, 2016 YouTube.

Teises videos näidatakse lihtsalt erinevaid pilte ja seletatakse taustaks, mida on nendel piltidel mõeldud. Käesolev video on leitav aadressilt https://www.youtube.com/watch?v=Jw1iFr4v58M (Joonis 38). Videos on seletatud iga nõude definitsiooni ja seejärel tõestatud piltlike joonistuste abil, miks need kolm ei saa korraga garanteeritud olla. Videos on hästi ja lihtsalt seletatud, mida iga omadus tähendab ja ka teoreemi tõestus on lihtne ja lühike. Samas jällegi sooviks näha konkreetseid näiteid, millal ja kuidas andmed erinevates andmebaasi sõlmedes muutuvad. Veel on selle animatsiooni puuduseks see, et enamus edastatavast infost antakse edasi heli kaudu ehk ta on üsna väärtusetu, kui pole võimalik animatsiooni heliga vaadata.



Joonis 38. CAP Theorem, 2016 YouTube.

7 Kokkuvõte

Antud bakalaureusetöö eesmärgiks oli luua animatsioonid hajusate andmebaaside CAP teoreemi kohta. Animatsioonide loomiseks kasutati CreateJS teeki. Antud töös suudeti katta kõik kolm võimaliku CAP teoreemi kaudu määratud hajusate andmebaaside disainivalikut.

Loodud CAP teoreemi animatsioonid põhinevad SQL-andmebaasidel.

Töö tegemise käigus valmisid järgnevad vahetulemused, mis on esitatud antud töö peatükkidena:

- Lühiülevaade CAP teoreemist
- Ülevaade CreateJS teegi kasutatud komponentidest ja nende kasutusvõimalustest, samuti muudest kasutatud raamistikest ja tehnoloogiatest.
- Animatsioonide põhimõtte kirjeldamine ja animatsioonide realiseerimine. Loodud animatsioonide sisu saab failis andmete muutmisega muuta ja seega on animatsioonid häälestatavad.
- Võrdlus teiste samateemaliste animatsioonidega.

CAP teoreemi kohaselt saab hajusate andmebaaside korral olla kolmest akronüümiga tähistatud nõudest maksimaalselt garanteeritud kaks. Ühest nõudest loobumine ei tähenda, et teised kaks oleksid tagatud, selleks tuleb vaeva näha. Üks hajusate andmebaaside süsteem võib erinevate andmete ja olukordade puhul tagada kolmest nõudest erinevaid kahekaupa variante.

CreateJS on JavaScripti teek, mis on mõeldud HTML5 joonistuslõuendile vektorgraafika joonistamiseks ja elementide animeerimiseks. Antud animatsioonide loomiseks kasutati alamkomponente EaselJS ja TweenJS.

Testisikuks olnud mitte IT-spetsialist arvas, et animatsioonid täidavad oma eesmärki. Isegi teemat üldse mitte tundes, tekkis tal teoreemist mingi arusaam ja kui esimesest animatsiooni vaadates kõigile küsimustele kohe õiget vastust ei osanud valida, siis teist ja kolmandat animatsiooni vaadates oli see juba lihtsam ja enamasti leiti vastus esimesel katsel.

Teisi samateemalisi animatsioone õnnestus leida kahelt aadressilt: https://www.youtube.com/watch?v=elWq2-RdfRk ja https://www.youtube.com/watch?v=Jw1iFr4v58M. Selgus, et mõlemad on tehtud video kujul ja laetud YouTube'i. See tähendab, et seadmest, mis Flashi ei toeta, ei ole võimalik neid videoid vaadata. CreateJS töötab aga kõigi seadmete peal, kus on JavaScripti toetav veebilehitseja. Enamus veebilehitsejaid seda ka teevad.

Kõik püstitatud eesmärgid said täidetud. Töö kirjutamise käigus sain rakendada oma olemasolevaid teadmisi ning ühtlasi õppisin ka palju uut. Valmisid animatsioonid, mida saab kasutada CAP teoreemi õppimiseks ja õpetamiseks.

Kindlasti võib antud tööd veel edasi arendada. CAP teoreemi jaotustaluvuse nõue on väga lai ja erinevaid situatsioone on palju. Antud töös kasutati selle kõige lihtsamat kuju ehk võrgukatkestust. Lõputöö sai üles ehitatud nii, et animatsioonides kasutatud andmeid oleks kerge muuta ning täiendada, kuid ka selles osas on võimalik edasi minna, muutes veel suurema osa animatsioonist andmetega juhitavaks. Töö edasiarenduseks võiksid olla ka animatsioonid, kus on näidatud, millises olukorras millistele hajusate andmebaaside nõuetele tuleks rõhuda ning kuidas vale disaini valimine võib põhjustada probleeme. Lisaks võiks muudetavad andmed olla võetud otse andmebaasist, kus kasutajad neid saaksid veelgi mugavamalt läbi mõne rakenduse vajadusel muuta.

Valminud animatsioonid leiab aadressilt:

http://viktor.ld.ttu.ee/animatsioonid/animation_cap_theorem/

Kasutatud kirjandus

[1] Vallaste, H., *e-Teatmik: IT ja sidetehnika seletav sõnaraamat*. [Online] http://www.vallaste.ee/index.asp (17.05.2016)

[2] *HTML*. [WWW]. http://et.wikipedia.org/wiki/HTML (20.05.2016)

[3] Pöial, J., *Objektorienteeritud programmeerimise põhimõisted*. [Online] http://enos.itcollege.ee/~jpoial/oop/loeng/oop_alused.html (17.05 2016)

[4] *HTML5 - Joonistuslõuend*. [WWW] http://metshein.com/index.php/veeb/html5/522-19-html5-joonistuslouend (20.05.2016)

[5] *Canvas element*. [WWW] https://en.wikipedia.org/wiki/Canvas_element (17.05.2016)

[6] *TweenJS v0.6.6 API Documentation: TweenJS*. [WWW] http://www.createjs.com/docs/tweenjs/modules/TweenJS.html (17.05.2016)

[7] *Design science*. [WWW] https://en.wikipedia.org/wiki/Design_science (17.05.2016)

[8] Gilbert, S., Lynch, N., *Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web*. [Online] https://pdfs.semanticscholar.org/24ce/ce61e2128780072bc58f90b8ba47f624bc27.pdf (17.05.2016)

[9] Brewer, E., *CAP Twelve Years Later: How the "Rules" Have Changed*, 2012. [Online] https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed (17.05.2016)

[10] Eessaar, E., *Hajus Andmebaas*, 2015. [Online] http://maurus.ttu.ee/ained/IDU0220_IDU0230/doc/21/Teema_IDU0230_12_2015_ver3.pdf (17.05.2016)

[11] *Comparison of web browsers.* [WWW] https://en.wikipedia.org/wiki/Comparison_of_web_browsers#JavaScript_support. (17.05.2016)

[12] *EaselJS v0.8.2 API Documentation: EaselJS*. [WWW] http://www.createjs.com/docs/easeljs/modules/EaselJS.html (17.05.2016)

[13] *jQuery.getJSON() | jQuery API Documentation*. [WWW] http://api.jquery.com/jquery.getjson/ (18.05.2016)

[14] Crockford, D., *JSON: The Fat-Free Alternative to XML, 2006*. [Online] http://www.json.org/fatfree.html (17.05.2016)

[15] Metsvahi, K., *Animatsioonid andmebaasi normaliseerimise kohta SQL-andmebaaside näitel. Animations About Database Normalization in the example of SQL Databases - TTÜR DIGIKOGU,* 2015 [Online] http://digi.lib.ttu.ee/i/?3431 (20.05.2016)