

Tallinna Ülikool  
Informaatika Instituut

**jQuery kasutamine reaalarakenduste loomisel**  
**Using jQuery in Development of Real-time**  
**Applications**

Bakalaureusetöö

Autor: Hannes Liiker

Juhendaja: Jaagup Kippar

Autor: ..... ,, ..... ,, 2011

Juhendaja: ..... ,, ..... ,, 2011

## **Autorideklaratsioon**

Kinnitan, et olen koostanud käesoleva bakalaureusetöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud.

Kõik töö koostamisel kasutatud teiste autorite tööd, põhimõttelised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....  
(kuupäev)

.....  
(autori allkiri)

# Sisukord

Sissejuhatus.....	4
1. Ülevaade teemaga seonduvast teabest.....	5
1.1 HTML DOM.....	5
1.2 JavaScript.....	6
1.3 JavaScripti teek.....	9
1.4 JSON.....	9
1.5 XMLHttpRequest.....	10
1.5.1 XHR käsitusviis.....	10
1.6 jQuery.....	10
1.6.1 jQuery teegi lisamine lehele.....	11
1.6.2 jQuery käsitusviis.....	12
1.7 Ajax.....	12
2. Näiteid jQuery Ajaxi moodulitest.....	14
2.1 Veebilehe sisu esitamine.....	14
2.1.1 jCarousel.....	14
2.1.2 jQuery LightBox.....	15
2.1.3 Dynamic News.....	17
2.1.4 MB.verticalSlider.....	18
2.2 Tekstivormidega manipuleerimine.....	19
2.2.1. Ajax Form Validation.....	19
2.2.2 FaceBook Autosuggest Like.....	19
2.3 Reaalaja otsingumootorid.....	20
2.3.1 Live Search 2.0.....	20
2.4 Varia.....	21
2.4.1 Ajax Upload.....	21
2.4.2 AjaxScroll.....	22
3. jQuery + Ajax rakendamine.....	23
3.1 Lühendatud Ajaxi andmevahetus.....	23
3.2 Laiendatud Ajaxi andmevahetus.....	24
3.3 Ajaxi kasutamine üle terve lehe.....	26
3.4 Lihtsustamata JavaScripti asünkroone päring.....	28
4. Testimine.....	29
4.1 jQuery teegifaili viide lehe päises versus lehe alaosas.....	30
4.2 JavaScripti päring versus jQuery päring.....	31
4.2.1 Jõudluse võrdlus.....	31
4.2.2 Andmemahu võrdlus.....	33
4.3 Ühilduvus eri lehitsejatega.....	35
Kokkuvõte.....	37
Summary.....	38
Kasutatud kirjanduse loetelu.....	39
Lisa 1.....	40

## Sissejuhatus

Võrreldes kümne aasta tagusega on tänapäeva veebimaastik teinud suure muutuse. Internetist on saanud platvorm rakendustele, mis oma funktsionaalsuse ja keerukuse poolest võrdväärseid lokaalsetele arvutiprogrammidega. Võib öelda, et veebirakendused on kasutajasõbralikud - nad ei nõua paigaldamist ja ei sõltu operatsioonisüsteemist. Lisaks on neid kerge juurutada, kuna kasutuselevõtuks piisab vaid ühest töötavast versioonist ja kasutajatel on sellele väga lihtne ligi pääseda.

Koos veebimaastikuga on muutunud ka vahendid selle arendamiseks. Algaja arendaja võib leida ennast olukorrast, kus konkreetse lahenduse loomiseks on lihtsalt üle mõistuse palju vahendeid ja alternatiive. Selle näiteks võib tuua JavaScripti, mille raamistike, teekide ja arenduskomplektide arv on lausa sadades. Samas laienduste funktsionaalsused tihtilugu kattuvad. Segaseks võib jääda ka, miks peaks eelistama laiendusi, kui emakeelega, mille baasil on laiendused üles ehitatud, on kõik tehtav ja samas isegi paremate tulemustega.

Käesoleva töö eesmärgiks on tutvustada jQuery teegi võimalusi Ajaxi reaalajarakenduste loomiseks. Seda siis juba valmis moodulite näol, kui ka demonstreerides konkreetseid meetodeid, mille baasil saab rakendusi ise luua.

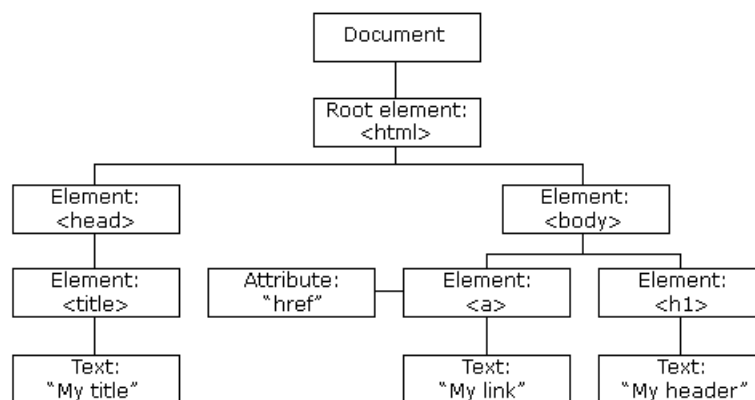
Teiseks eesmärgiks on välja selgitada, kas jQuery teegi kasutamine lihtsamate Ajaxi lahenduste loomisel on alati põhjendatud? jQuery on küll abivahend, mis peaks arendaja elu lihtsamaks tegema, kuid seejuures mis on miinused?

# 1. Ülevaade teemaga seonduvast teabest

## 1.1 HTML DOM (*HTML Document Object Model*)

HTML (ing. k. *HyperText Markup Language*) dokumendiobjektide mudel (ing. k. *Document Object Model*) on W3C konsortsiumi poolt kinnitatud eeskiri, mille põhjal veebilehitsejad esitavad ja käsitlevad HTML dokumente. Arendajate seisukohast vaadates on DOM kui standardiseeritud objektorienteeritud representatsioon veebilehest, mis lihtsustab elementide manipuleerimist erinevate tehnoloogiatega (nt. JavaScript-iga) ja tagab selle, et veebirakendused kuvatakse identselt igas veebilehitsejas. (W3Schools, 2011)

DOM määratleb HTML dokumendi objektid koos atribuutidega ning meetodid, kuidas neid saab rakendada. Kõik objektid on hierarhilise ülesehitusega ning vastavad reaalsele HTML elementidele, näiteks objektiks võib olla tekst, pilt, pealkiri jne. Alljärgnevalt näide HTML dokumendi elementidest (objektidest) (vt Illustratsioon 1).



*Illustratsioon 1: HTML dokumendi struktuur*

W3C (ing.k. *World Wide Web Consortium*) on rahvuvaheline interneti ja veebilahendustega tegelevate ettevõtete konsortsium, mille eesmärgiks on välja töötada avalikud eeskirjad, mis suunaksid interneti potentsiaalset arengut ning samas hoiaksid terviklikkust. (W3C, 2011)

## 1.2 JavaScript

JavaScript on funktsionaalne skriptimiskeel dünaamiliste objektide ja üldtuntud süntaksiga, mis tugineb ECMAScript standardile. (Crockford, 2010)

Skriptimiskeel (ing.k. *scripting language*) võimaldab siduda erinevate rakenduste erinevad komponendid omavahel. Tegemist ei ole "iseseisva" programmeerimiskeelega, mille abil saab luua nullist mahukaid rakendusi vaid vahendiga, mis võimaldab ühendada ja laiendada olemasolevad komponente. (Brown, 2009)

JavaScripti loojaks on Netscape programmeerija Brendon Eich. Esimene versioon kandis nime Live Script ja avalikustati 1995. aasta septembris ühe osana uuest Netscape Navigator 2.0 veebilehitsejast (ing. k. *browser*). Paar kuud pärast esmast avalikustamist muudeti nimi JavaScriptiks, sealjuures pole Javascriptil mingit seost Java programmeerimiskeelega, millele nimi viitab. (Bellis, 2010)

Skriptimiskeele eesmärgiks oli muuta veebirakenduste kasutajaliidesed dünaamilisemaks ja interaktiivsemaks - et nad oskaksid vahetumalt suhelda kasutajaga ja säästaksid üleliigsetest ühendustest serveriga. Näiteks, sobib JavaScript ideaalselt HTML vormide valideerimiseks, mis omal ajal oli suur arenguhüpe.

Hetke seisuga on JavaScript laialt levinud ja seda peetakse üheks kõige kasutatavamaks programmeerimiskeeleks. Peamiseks kasutuskeskkonnaks on veebilehitsejad, kuid näiteks ka erinevad arvutiprogrammid, operatsioonisüsteemid, andmebaasid, mobiilirakendused jne.

JavaScript on eriti efektiivne sümusepõhiste rakenduste skriptimisel.

JavaScript käivitatakse klientprogrammis (veebilehitsejas) ilma serveri poole pöördumata. Skriptid on kirjutatud HTML dokumendi sisse või paiknevad serveris omaette failidena. Viimasel juhul viidatakse neile HTML failis ja kaastakse lehe laadimisse. Koodi rakendamiseks ei pea eraldi kompilleerima, toimub vahetu tõlkimine veebilehitseja interpretaatori põhjal.

Järgnevalt JavaScripti fundamentaalsed omadused ja terminoloogia, mis aitavad mõtestada ja seletada antud uurimustööd.

- **Tõstutundlik**

Tegemist on tõstutundliku keelega, mis tähendab et muutujate ja funktsioonide nimedes, samuti tekstiliste väärtuste võrdlemisel, mängivad rolli suured ja väikesed tähed. Näiteks alert() ei ole sama mis ALERT() ja samuti ei ole võrdsed tekstistringid „muusika“ ja „Muusika“.

- **Dünaamiliselt ja nõrgalt tüübitud**

Tüüpimine on muutujate klassifitseerimine selle järgi, mis liiki andmeid nad sisaldavad, näiteks string, täisarv jne. Dünaamiliselt tüübitud programmeerimiskeeles toimub tüübikontroll programmi täitmise ajal.

Nõrgalt tüübitud programmeerimiskeeles puuduvad eelnevalt määratud tingimused, kuidas informatsiooni saab rakenduse piires käsitleda. Informatsiooni ehk muutuja olemus koorub täielikult sellest, kuidas seda hakatakse käsitlema. Vastandina tugevalt tüübitud keeles on muutujad täielikult defineeritud. Näiteks on muutuja, mis programmi alguses on deklareeritud kui number, kuid hiljem kasutatakse seda stringina. See juures muudetakse väärtused vastaval nõuetele ehk number muudetakse tekstikujule.

- **Prototüübipõhine objektorienteeritud**

Objektorienteeritud programmeerimine (lüh. OOP) on paradigma, mis keskendub andmetele, mitte protsessidele. Programmid koosnevad omaette andmekogumikest ehk klassidest, kus on kirjas isendi omadused ja oskused. Klassiks võib olla näiteks "auto" ja iseloomulikuks omaduseks "toonitud klaasid" ja oskuseks "sõitmine". Omadused jaotuvad eraldi veel nimeks ja selle väärtuseks - antud juhul on "klaasid" nimi ja väärtuseks "toonitud". Klassi võib laias laastus nimetada kui šablooniks, mille põhjal

valmistatakse objektid.

Prototüübipõhine OOP on lähenemine, mis sarnaneb traditsioonilisele objektorienteeritud paradigmale, kuid erinevus on selles, et klasse ei eksisteeri. Kasutatakse ainult objekte ja kloonimine toimub prototüüpimise põhimõttel ehk objektist tehakse prototüüp, millel on samad omadused ja oskused objektiga, millest ta klooniti.

OOP-keeles kutsutakse objekti omadusi parameetriteks (ing.k. *properties*) ja oskusi meetoditeks (ing.k. *methods*).

- **Kapseldatavus (*Encapsulation*)**

Kõik parameetrid ja meetodid on objektide sisse kapseldatud ehk nende sisu on kaudselt varjatud, kuid nad on alati rakendatavad. Seega saab kiirelt ja tulemusele orienteeritult kasutada erinevaid objekte ilma lahkamata, kuidas nad täpsemalt töötavad. Antud põhimõttel töötavad JavaScripti teegid ja raamistikud. (Stefanon, 2008)

- **Pärandatavus (*Inheritance*)**

Objekt saab pärandada oma parameetrid ja meetodid teisele objektile, mis on määratud tema järglaseks. (Stefanon, 2008)

- **Polümorfism (*Polymorphism*)**

Objektid võivad omada samu meetodeid, kuid nende rakendamisel võivad tulemused olla erinevad. Põhjuseks on erinevad parameetrid ja andmetüübid. (Stefanon, 2008)

- **Koondatavus (*Aggregation*)**

Mitmete objektide koondamist ühtseks tervikuks (uueks objektiks) nimetatakse OOP-keeles koondamiseks. Vastupidiselt, kuid antud mõiste najal, võib ühe objekte jagada ka mitmeks väiksemaks objektiks. Näiteks objekt Raamat koosneb objektidest



Kirjastaja, Autor ja Leheküljed. Selline jaotatavus muudab vigade otsimise kergemaks, kuna aitab abstraktsel kujul paremini mõista programmi ülesehitust. (Stefanon, 2008)

### 1.3 JavaScripti teek (*JavaScript library*)

JavaScripti teek (ing.k. *JavaScript library*) on kogumik valmis kirjutatud JavaScripti alamprogramme. Antud alamprogrammide olemasolu muudab arendustöö kergemaks ja kiiremaks kuna arendaja ei pea enam igal sammul n.ö. "ratast uuesti leiutama" vaid saab rakendada juba eelnevalt loodud mooduleid.

### 1.4 JSON (*JavaScript Object Notation*)

JSON on lihtne ja universaalne andmete hoiustamise ja vahetamise formaat, mis tugineb JavaScripti süntaksil. Tegemist on populaarse alternatiiviga XML-ile. (JSON, 2011)  
Järgnevalt näide JSON andmestruktuurist (vt Koodinäide 1).

```
1.  {
2.    "nimi": "Toomas",
3.    "vanus": 32,
4.    "lapsed": [
5.      {
6.        "nimi": "Teele",
7.        "vanus": 13
8.      }
9.    ]
10. }
```

*Koodinäide 1: JSON andmestruktuur*

### 1.5 XMLHttpRequest

XMLHttpRequest (lüh. XHR) on W3C konsortsiumi poolt standardiseeritud JavaScripti objekt, mis võimaldab kliendi ja serveri vahelist andmevahetust ilme veebilehte täielikult uuendatama. Tegemist on Ajaxi ühe peamise komponendiga. Eksitavalt nimest võimaldab XHR objekti lugeda kõiksugu formaatides andmeid (mitte ainult XML) ja lubatud on ka teised protokollid peale HTTP (nt. FILE ja FTP). (Mozilla, 2011)

XHR objekt loodi Microsofti arendajate poolt 1999. aastal ja esimeseks platvormiks oli Microsoft Exchange Server 2000. Järgneva sammuna lisati see Internet Explorer 5.0 veebilehitsejale, mis nägi ilmavalgust 1999. aasta märtsis. Laiema populaarsus tekkis 2005 aastal, kui Google kasutas seda oma Google Suggest tekstisisestamise lahtrites (ingl. k. *input box*). (Hopmann, 2007)

### 1.5.1 Traditsiooniline XHR käsitusviis

1. XHR objekti loomine.
2. XHR objekti kasutamine, et edastada asünkroone päring serverile ning tagasikutse-funktsiooni (ing. k. *callback function*) defineerimine, mis käivitatakse juhul kui päring serverile on edastatud.
3. Serveri vastuse käsitlemine. (olenevalt *callback* funktsioonist)
4. Tagasi 2. sammu juurde või lõpetamine.

### 1.6 jQuery

jQuery on kompaktne JavaScripti teek, mis lihtsustab suhtlemist JavaScripti ja HTML dokumendiobjektide mudeli (lüh. DOM) vahel. jQuery üheks peamiseks plussiks on tema kergesti omandatav süntax. (jQuery, 2011)

Järgnevalt mõned faktid jQuery koduleheküljelt (url: [www.jquery.com](http://www.jquery.com)).

- jQuery on vabavaraline ja avatud lähtekoodiga - avalikustatud MIT ja GNU GPL (*General Public License*) litsentside alusel.
- jQuery on laialdaselt kasutusel ja omab suurt kogukonda - arendajaid ja eestvedajaid leidub igal pool üle maakera.
- Saadaval on suures koguses õppematerjale: e-raamatud, tutorialid, blogid ja artiklid. Lisaks keskkonnad, kust saab vajadusel abi küsida: maililistid, foorumid ja IRC kanalid.
- API (ing. k. *Application Programming Interface*) ehk rakendusliides on täielikult

dokumenteeritud ja kõigile kättesaadav.

- Suur ja järjest kasvav laienduste (ing.k. *plugins*) kogu.
- Andmemahult väike - pakitult 29KB ja pakkimata 207KB
- Kõik kaasaegsed veebilehitsejad toetava (IE6+, Firefox 2+, Safari 3+, Opera 9+, Chrome).

jQuery teek koosneb järgnevatest võimalustest:

- HTML elementide selekteerimine
- HTML elementidega manipuleerimine
- CSS laadilehtede reeglistikega manipuleerimine
- HTML sündmuste kasutamine
- JavaScript efektid ja animatsioonid
- HTML dokumendiobjektide mudeli struktureeritud käsitlemine ja modifitseerimine
- Ajaxi tehnikate rakendamine
- Uutiliidid

### 1.6.1 jQuery teegi lisamine lehele

jQuery teek koosneb ühest JavaScripti failist, mis sisaldab kõiki funktsioone ja meetodeid. Faili lisamiseks kasutatakse järgnevat märgistust (vt. Koodinäide 2).

```
1. <head>
2. <script type="text/javascript" src="jquery.js"></script>
3. </head>
```

*Koodinäide 2: jQuery teegifaili lisamine*

### 1.6.2 jQuery käsitusviis

jQuery töömehhanismi võib lihtsustatult kokku võtta järgnevalt: mingi elemendi valimine ja siis sellega millegi tegemine.

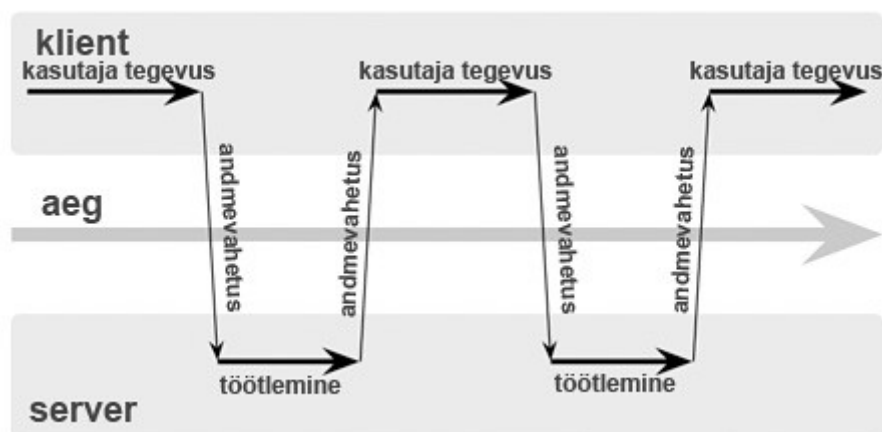
Traditsiooniline jQuery juhtlause koosneb neljast osast: jQuery funktsioon, selektor, toiming ja parameetrid. Alljärgnevalt ülevaade igast osast:

1. **jQuery funktsioon:** iga lause algab funktsiooniga **jQuery()**, mis käivitab teegi ja n.ö. avab ukse jQuery võimaluste maailma. Lühendina on kasutusel dollarimärk (\$) .
2. **Selektor:** element(mendid) millega hakatakse midagi tegema.
3. **Toiming:** tegevus ehk millesse element(mendid) kaasatakse.
4. **Parameetrid:** lisaandmed mis on vajalikud tegevuse täitmiseks.

## 1.7 Ajax

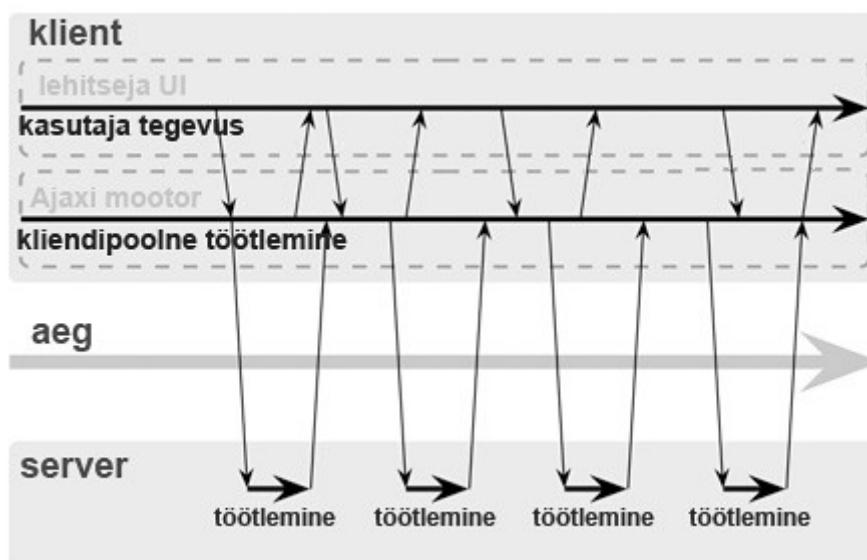
Ajax (ing.k. *Asynchronous JavaScript + XML*) ei ole eraldi tehnoloogia vaid kombinatsioon juba teadatuntud tehnikatest, mis võimaldavad muuta traditsioonilist arusaama kasutaja ja serveri vahelisest suhtlemisest. (Garrett, 2005)

Üldtuntud mudel järgi toimivad veebilehed sünkroonselt päring/vastuse põhimõttel - klient saadab päringu serverile ja server täidab vastavad muudatused ja saadab vastuse. Vastuse kättesaamiseks toimub lehekülje täielikult uuesti allalaadida (koos päise ja jalusega). Antud protsess võtab aega (olenevalt andmeedastuskiirusest) ja tekib viivitus, kus kasutaja on n.ö. "käed rüpes" (vt Illustratsioon 2).



Illustratsioon 2: Sünkroonne kliendi ja serveri vaheline suhtlemine

Ajax võimaldab andmevahetust serveri ja kliendi vahel vaadata kui mitme erineva asünkroonse tasandina, see tähendab, et ühe lehekülje raames toimub mitu erinevat päring/vastus protsessi. See võimaldab serveri ja kliendi vahel edastada ainult neid andmeid, mis on ka reaalselt vajalikud. Antud olukorras muutub interaktsiooni kasutaja ja veebirakenduse vahel vahetumaks ja produktiivsemaks, kuna vähenevad olukorrad, kus kasutaja on täielikult ootel (vt Illustratsioon 3).



Illustratsioon 3: Asünkroonne kliendi ja serveri vaheline suhtlemine

Tim O'Reilly poolt 2003. aastal kasutusele võetud terminit Web 2.0 võib mitmeti seostada Ajaxi meetodite levikuga. Võrreldes vana kulunud arusaamaga veebist kui staatilisest arhiivist, tähendas Web 2.0 uut suunda, mis hõlmab enda alla nii tehnoloogilised kui sotsiaalsed aspektid. Internetist on saanud platvorm, kus saavad kokku kliendid ja teenusepakkujad. Veebi graafilised kasutajaliidesed (ing. k. *Graphical User Interfaces*) suhtlevad kasutajaga vahetumalt ja painlikumalt. Populaarseks on saanud termin "rikkalik internetiliides" (ing. k. *Rich Internet Application*), mis tähendab netirakendust, mille kasutajasõbralikkus ja funktsionaalsus on võrdväärne lokaalsete arvutiprogrammidega (ing. k. *Desktop Applications*).

Ajax ühendab järgnevad veebitehnoloogiad:

- XHTML esitluskihi keel ja CSS laadilehed
- DOM - dokumendiobjektide mudel
- XML ja XSLT
- XMLHttpRequest objekt
- JavaScript

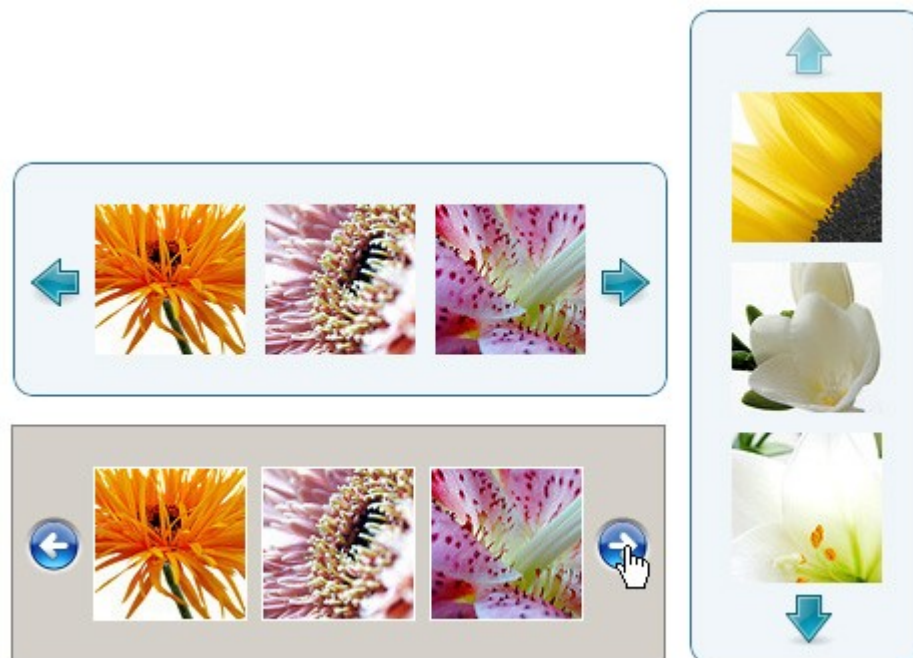
## 2. Näiteid jQuery Ajaxi moodulitest

Käesolevast peatükis on ülevaade jQuery moodulitest (ing.k. *plugins*), mis kasutavad Ajaxi tehnikaid. Peatüki eesmärgiks on tutvustada jQuery baasil loodud reaalajarakenduste kasutusvõimalusi ja välja tuua konkreetsed näited, kus ja millal antud mooduleid annaks rakendada.

Kõik moodulid on tasuta kättesaadavd ja järgus, kus nad on täielikult rakendatavad. Valiku tegemisel on lähtunud peamiselt mooduli praktilisest poolest, kuid arvesse läksid ka märksõnad nagu originaalsus ja innovaatus.

### 2.1 Veebilehe sisu esitamine

#### 2.1.1 jCarousel



*Illustratsioon 4: jCarousel*

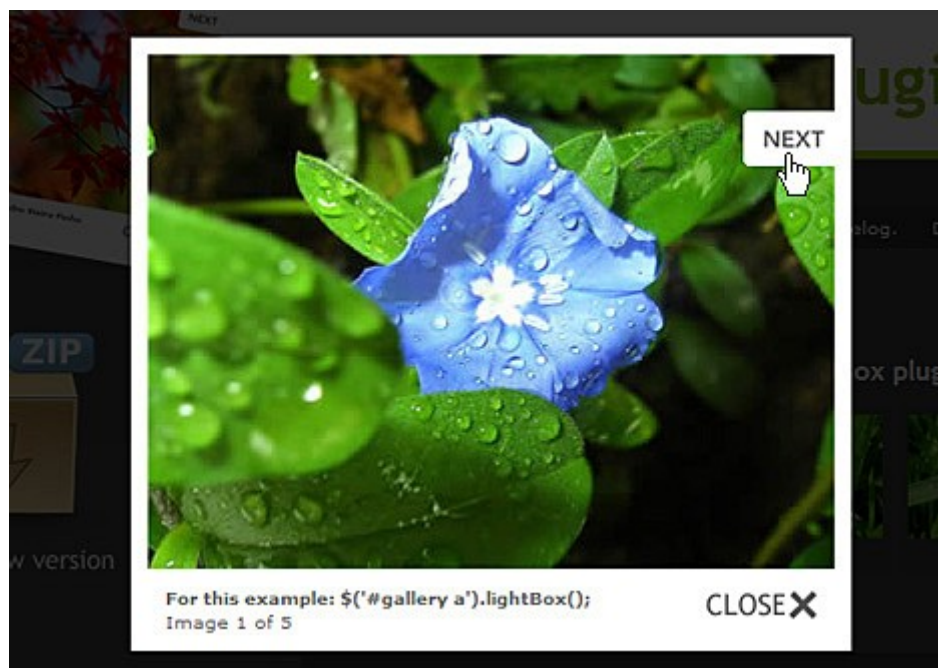
Arendaja kodulehel on järgnev tutvustus: jCarousel on mõeldud elementide kuvamiseks nii

horisontaalselt kui vertikaalselt. Elementide laadimine toimub nii staatiliselt (kõik andmed loetakse lehe avamisel korraga) kui dünaamiliselt, kasutades Ajaxi tehnikaid. (url: <http://sorgalla.com/jcarousel/>)

jCarousel sobib suuremas mahus väiksemate piltide kuvamiseks ja seejuures ei võta ta palju ruumi, näiteks oleks ta sobilik ettevõtte kodulehele erinevate toodete tutvustamiseks, kus külastaja saaks kiirelt otsida vastavalt oma soovile. Miinusena võib öelda, et on puudu liides, mis avaks pildid ka täissuuruses.

Dünaamiline sisu kuvamine toimub läbi eraldiseisva tekstidokumendi (*dynamic\_ajax.txt*), kus on kirjas elementide viited. Lehe avamisel laaditakse alla ainult need pildid, mis on ka reaalselt näitamisel. Nupule vajutades laetakse automaatselt tekstidokumendi viidete põhjal järgmised pildid jne (vt Illustratsioon 4).

### 2.1.2 jQuery LightBox



*Illustratsioon 5: jQuery LightBox*

Arendaja kodulehel on järgnev tutvustus: jQuery Lightbox on lihtne, elegantne, tagasihoidlik



moodul, mis on mõeldud täismõõdus piltide kuvamiseks ilma pealehelt lahkumata. Lightbox ei vaja eraldi koodipõhist märgistust - see on võimlik tänu paindlikele jQuery selektoritele. (url: <http://leandrovieira.com/projects/jquery/lightbox/>)

Tegemist on küllaltki levinud viisiga täissuuruses piltide kuvamiseks. Kui kasutaja klõpsib hiirega pispildil (ing. k. *thumbnail*) siis avaneb samal lehel täismõõdus versioon sellest. Seejuures saab taustaks olev veebileht tumeda varjundi, mis aitab pilti esile tuua. Nupud *Next* ja *Prev* navigeerivad järgmiste ja eelnevate piltide vahel (vt Illustratsioon 5).

Mooduli paigaldamine koosneb kolmest sammust:

1. jQuery teegi ja LightBox mooduli lisamine lehele:

```
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript" src="js/jquery.lightbox0.4.js">
</script>
```

2. CSS laadilehe lisamine:

```
<link rel="stylesheet" type="text/css" href="css/jquery.lightbox-
0.4.css" media="screen" />
```

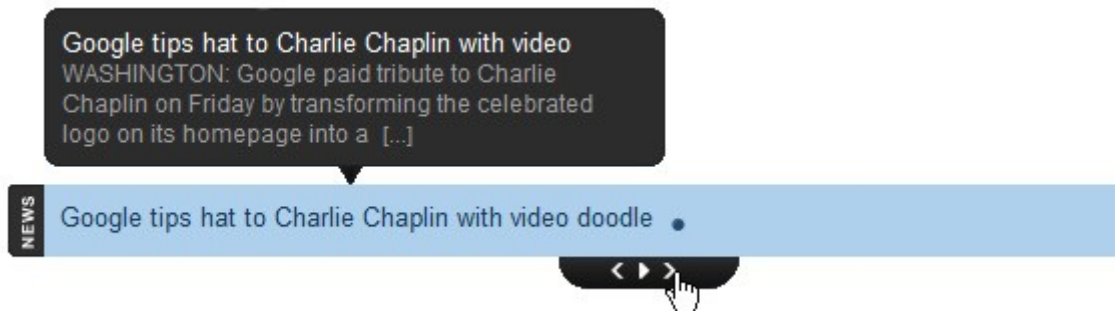
3. jQuery funktsiooni välja kutsumine:

```
<script type="text/javascript">
  $(function() {
    $('#gallery a').lightbox({fixedNavigation:true});
  });
</script>
```

Antud näites on jQuery selektoriks **#gallery a**. Kui CSS elemendi **#gallery** piires, vajutatakse mõnele lingile (**a** märgend), siis käivitatakse **.lightbox** meetod. Kaasatud parameeter **fixedNavigation:true** määrab, et lisatakse ka navigeerimisnupud.

Nendest kolmest sammust piisab, et lisada moodul oma lehele. Märkimisväärne on, et piltide viidetele ei ole vaja lisada mõnda täpsustavat koodi - lightBox meetodi käivitamiseks on vaja määratleda ainult CSS selektor (ing. k. *CSS selector*).

### 2.1.3 Dynamic News



*Illustratsioon 6: Dynamic News*

Arendaja kodulehel on järgnev tutvustus: Dynamic News on jQuery moodul, mis dünaamiliselt kuvab ja vahetab uudiseid. Uudiseid saab lisada kahel viisil: läbi staatilise HTML faili või kasutades RSS uudistevoogusid (ing. k. Really Simple Syndication). (url: <http://www.egrappler.com/rss-driven-dynamic-jquery-news-slider-plugin-dynamic-news/>)

Käesolev moodul sobib uudiste lisamiseks lehele juhul, kui selleks on vähe ruumi. DynamicNews kuvab ühe uudise korraga ja vahetab vaikumisi iga viie sekundi tagant. Täiendavalt, hiire lohistamisel rakendusele, avaneb pikem versioon ja juhtpaneel, mille abil saab uudiseid vahetada (vt Illustratsioon 6).

### 2.1.4 MB.verticalSlider



*Illustratsioon 7:  
MB.verticalSlider*

Arendaja kodulehel on järgnev tutvustus: MB.verticalSlider kuvab suures koguses infot väikese ala piires. Sisu võib olla juba laetud (ehk juba dokumendiobjektide mudeli sees) või laetakse eraldi, kasutades Ajaxi tehnikaid.

(url: <http://pupunzi.com/#mb.components/mb.verticalSlider/verticalSlider.html>)

MB.verticalSlider koosneb veel kahest moodulist: jquery.easing.1.3 ja jquery.mousewheel (vt Illustratsioon 8). Esimest kasutatakse, et animeerida elementide vahetumist. Vaikimisi kasutatakse *easeOutExpo* animatsiooni ehk üleminek ühelt nimekirjalt teisele toimub n.ö. sujuvalt. Teise mooduliga saab registreerida hiire rulliku kasutamist, mis antud juhul on mõeldud elementide kerimiseks hiirega. Testimisel Chrome veebilehitsejaga see ei töötanud. Firefox lehitsejas saab küll mingil määral kerida, kuid ainult ühe suunaliselt ja väga suure viivitusega.

Name	Type	Size
jquery.easing.1.3	JScript Script File	7 KB
jquery.mousewheel.min	JScript Script File	2 KB
mbVerticalSlider	JScript Script File	8 KB

*Illustratsioon 8: MB.verticalSlider failistruktuur*

## 2.2 Tekstivormidega manipuleerimine

### 2.2.1. Ajax Form Validation

Register

Username, valid: a-z.-\_

This username is free

Avatar URL

*Illustratsioon 9: Ajax Form Validation*

Arendaja kodulehel on järgnev tutvustus: Tekstivormid (ing. k. *forms*) on muutunud nii laialt kasutatavateks ja igapäevaseks, et me tihtilugu ei pööra enam tähelepanu sellele, mis me kirjutame. Aga kui lehele on lisatud mõned täiendused siis muudab see teksti sisestamise mugavamaks ja kiiremaks. (url: <http://jqueryfordesigners.com/using-ajax-to-validate-forms/>)

Käesolev moodul testib reaalajas (kasutaja trükkimise ajal) tekstivormide sisu. Kontrollimine toimub kahel tasandil: esimesel juhul toimub lihtne serverist sõltumatu formaadi kontroll, näiteks kas emaili aadress on korrektne ja kas salasõna kinnitus on identne. Teisel juhul toimub asünkroone suhtlemine serveriga - kontrollitakse, kas kasutaja poolt kirjutatud kasutajanimi on juba kasutusel või mitte. Kui nimi on võetud siis teavitatakse koheselt (kasutaja ei pea lehte uuesti laadima) (vt Illustratsioon 9). Avatari pildi viite lisamisel püüab moodul automaatselt pilti kuvada ja kui see luhtub siis teavitatakse jällegi.

## 2.2.2 FaceBook Autosuggest Like



*Illustratsioon 10: FaceBook Autosuggest Like*

Moodul on valmistatud BSN Autosuggest rakenduse baasil, mis kasutab ainult traditsioonilist JavaScripti. Tegemist on abivahendiga, mis teksti sisestamisel pakub automaatselt soovitusi. Kasutaja saab hiirega klõpsida otse soovitusel või kasutada üles ja alla nooleklahve. Soovitused on salvestatud kas XML või JSON formaadis. Antud abivahend muudab tekstisisestamise kiiremaks ning lollikindlamaks, kuna kasutaja näeb täpsemalt, missugust infot temalt eeldatakse saada. (url: <http://www.web2ajax.fr/2008/02/03/facebook-like-jquery-and-autosuggest-search-engine/>)

## 2.3 Reaalaja otsingumootorid

### 2.3.1 Live Search 2.0



*Illustratsioon 11: Live Search 2.0*

Arendaja kodulehel on järgnev tutvustus: Live Search 2.0 muudab tavalise tekstivormi reaalaaja otsingumootoriks. Otsingu tulemused kuvatakse kohe teksti sisestamise ajal meelepärases HTML vormingus. (url: <http://andreaslagerkvist.com/jquery/live-search/>)

Moodul on põhjalikult dokumenteeritud koodiga. Paigaldamine on tehud võimalikult kergeks. Lehele tuleb lisada viited moodulile, stiililehele, jQuery teegile ning rakendamiseks piisab ainult järgnevast koodist:

```
<script type="text/javascript">  
    jQuery('#q').liveSearch({url: '/ajax/search.php?q='});  
</script>
```

Elemendile identifikatsiooniga **#q** lisatakse külge **liveSearch** meetod, millele on kaasatud parameeter **url**, mis viitab, kus kohast andmeid otsitakse ja kuvatakse.

Käesolev lahendus muudab otsimise märgatavalt kiiremaks ja vahetumaks. Tulemused on nähtavad reaajas ja vahetult teksti sisestamise ajal (vt Illustratsioon 11).

## 2.4 Varia

### 2.4.1 Ajax Upload

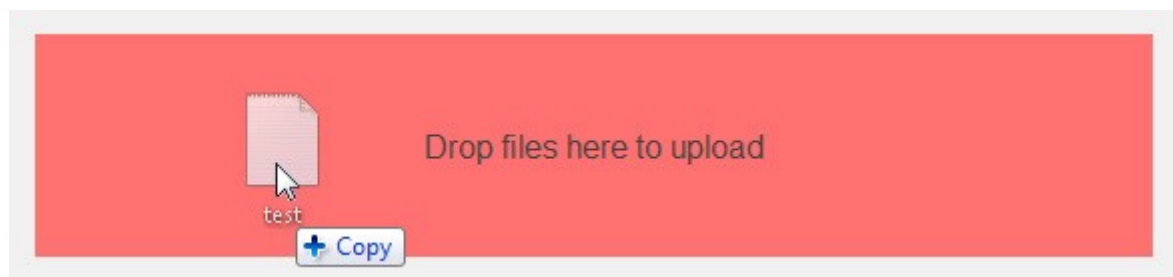


*Illustratsioon 12: Ajax Upload*

Arendaja kodulehel on järgnev tutvustus: Ajax Upload kasutab failide üleslaadimiseks vaikumisi XHR objekti. Juhul kui see ei ole lubatud siis kasutatakse automaatselt *iframe*-põhist lahendust. Toetatud on edenemisenäitur (ing. k. *progress bar*) ja mitme faili korraga üles laadimine. (url: <http://valums.com/ajax-upload/>)

Ajax Upload võimaldab lokaalsest masinast faile serverisse laadida ja seda kõike ühe veebilehe raamides (vt. Illustratsioon 12). Vaikumisi kasutatakse XHR objekti, mis on alternatiiviks *iframe* meetodil kasutatavatele lahendustele. XHR objekti kasutamine on turvanõuetega suuresti piiratud, olenevalt lehitsejate tõlgendamisest.

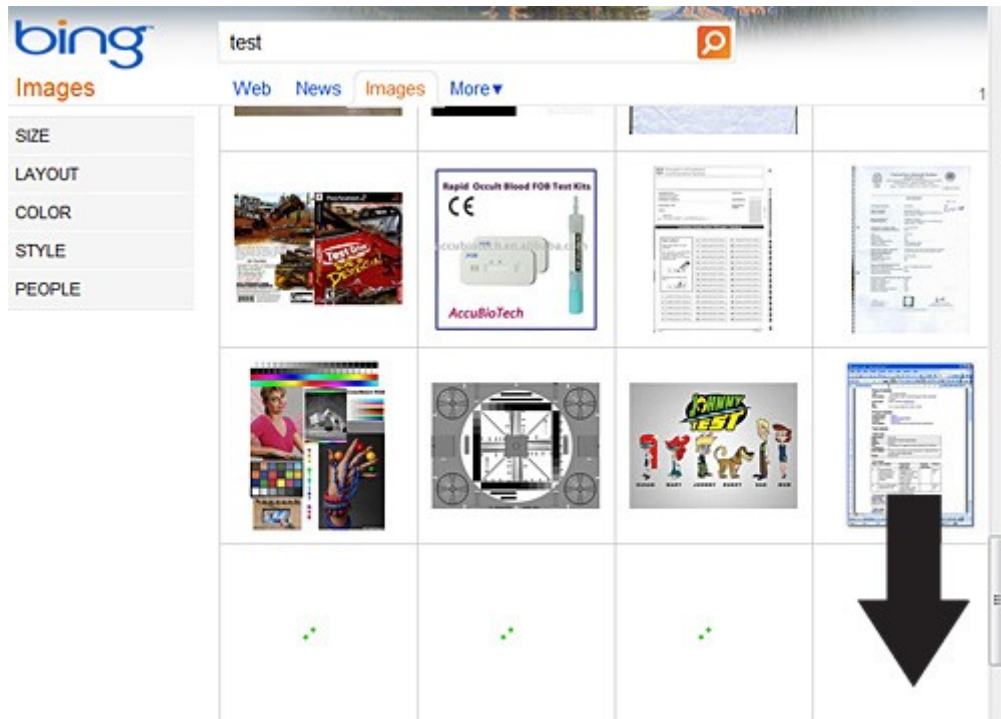
Lisaks on toetatud (ainult Firefox ja Chrome lehitsejaga) failide hiirega lohistamine (ing.k. *drag-and-drob*) (vt. Illustratsioon 13).



*Illustratsioon 13: Ajax Upload hiirega lohistamise võimalus*

## 2.4.2 AjaxScroll (url: <http://blog.yctin.com/archives/jquery-plugins-ajaxscroll/>)

AjaxScroll on lihtne väike moodul, mis lisab kerimisriba lõppu jõudes automaatselt lehel uusi elemente juurde. Analoogne tehnika on kasutusel näiteks Bing otsingumootoris, kus pildiotsingu korral puuduvad viited järgmisele ja eelnevale lehele. Lehe allaossa jõudes kuvatakse asünkroonselt uued pildid ja seda nii kaua kuni pilte jätkub (vt. Illustratsioon 14).



Illustratsioon 14: Bing otsingumootori reaajas piltide kuvamine



### 3. jQuery + Ajax rakendamine

jQuery on varustatud mitmekülgsete vahenditega, mille abil on võimalik luua Ajaxi rakendusi. Käesoleva peatüki eesmärgiks on tutvustada neid võimalusi täpsemalt. Iga alapeatükk on jagatud kolmeks osaks: eesmärk, lahendus eesmärgi saavutamiseks ja lahenduse lahkamine. Antud näited peaksid andma praktilise ülevaate jQuery baasil Ajaxi tehnikate rakendamisest.

Neljas näide koosneb traditsioonilise lihtsustamata JavaScripti baasil loodud reaalarakendusest. Antud näide on sisse toodud selleks, et võrrelda seda jQuery teegi-põhise lahendusega. Pärast, neljandas peatükis, toimib kahe antud lahenduse testimine.

#### 3.1 Lühendatud Ajaxi andmevahetus

Eesmärgiks on teha päring serverile ja küsida andmeid ilma lehte uuesti laadimata, kasutades selleks jQuery lühendatud meetodit.

Lahenduseks on alljärgnev lihtne koodjupp:

```
1. <script type="text/javascript">
2.   $(document).ready(function() {
3.     $('#sisu').load('test.html');
4.   });
5. </script>
```

*Koodinäide 3: Lühendatud jQuery Ajaxi päring*

Funktsioon käivitatakse kohe lehe avamisel kui DOM on allalaetud.

```
$('#sisu').load('test.html');
```

Antud näites on selektoriks element **#sisu**, mille sees kuvatakse faili **test.html** andmestik. Meetod **.load** kuulub kõrgema astme (ing. k. *high level*) käsitlusviisi alla. Antud meetodit on kerge rakendada, kuid tal puuduvad seadistusvõimalused. jQuery süntaks omab mitmeid lühendatud rakendusviise, mis lihtsamate arendustööde korral kiirendavad märgatavalt

tulemuse saavutamist. Antud võimalust illustreerib ka jQuery üks peamised müügilauseid: "Write Less, Do More" - lühikese koodijupiga annab palju korda saata. (jQuery, 2011)

### 3.2 Laiendatud Ajaxi andmevahetus

Eesmärgiks on teha päring serverile ja küsida andmeid, mis kuvatakse ilma lehte uuesti laadimata. Antud näites kasutame laiendatud jQuery meetodit.

Lahenduseks on järgnev kood:

```
1. <script type="text/javascript">
2.     $(document).ready(function() {
3.         $.ajax({
4.             type: 'GET',
5.             url: 'test.html',
6.             dataType: 'html',
7.             error: function(xhr, textStatus) {
8.                 alert('Tekkis viga! ' + (textStatus));
9.             },
10.            success: function(html, textStatus) {
11.                $('body').append(html);
12.            },
13.        });
14.    });
15. </script>
```

Koodinäide 4: Laiendatud jQuery Ajaxi päring

Peale DOM-i allalaadimist käivitatakse **\$.ajax** meetod (vt. Koodinäide 4, rida 3), mis on Ajaxi tehnikate rakendamise südameks. Tegemist on peamise komponendiga Ajaxi päring ja vastuse (ing. k. *request and response*) protsesside loomiseks kliendi ja serveri vahel. **.ajax()** liigitub madala astme (ing. k. *low level*) käsitusviisi alla - antud meetodit on keerulisem rakendada, kuid samas ta omab suuremat funktsionaalsust.

```
var options = {
    type: 'GET'
};
```

Meetodiga **jQuery.ajax()** kaasatakse objekt **options**, kus on kirjas seaded, mille põhjal protsess läbi viiakse. Sätete arv võib varieeruda mitmeti. Harilikult, nagu ka antud näites, on

kasutusel järgnevad parameetrid: **type**, **url**, **dataType**, **error**, ja **success**. Esimese sammuna defineeritakse, mis tüüpi HTTP päringut kasutatakse. Üldjuhul on selleks kas GET või POST.

POST ja GET on kaks HTML meetodit, mille baasil esitab lehitseja andmeinfo serverile. Nende kahe erinevus seisneb peamiselt andmete kodeerimises. GET meetodiga saadetakse andmed läbi internetiaadressi ning üldjuhul kasutatakse seda juhul, kui andmete hulk on väiksem. (Korpela, 2001)

```
var options = {
  type: 'GET',
  url: 'test.html',
  dataType: 'html'
};
```

Järgnevalt on defineeritud **URL** ja **datatype** parameetrid: esimene on viide serveris asuvale failile, mille andmeid soovitakse kuvada, parameeter **dataType** määratleb faili formaadi. Antud juhul eeldatakse saada *.html* formaadis faili, millele osutab ka parameeter **URL**.

```
var options = {
  type: 'GET',
  url: 'luuletus.txt',
  dataType: 'html',
  error: function(xhr, textStatus) {
    alert('Tekkis viga: ' + textStatus);
  },
  success: function(data, textStatus) {
    $('body').append( data );
  }
};
```

Järgmised kaks seadet määravad tagasikutsefunktsioonid (ing. k. *callback functions*): esimene nimega **error** ja teine **success**. Nagu nimedki vihjavad käivitatakse need juhul, kui protsess ei õnnestunud edukalt lõpuni viia või siis vastupidi. Esimesel juhul avaneb hüpikaken tekstiga "**Tekkis viga:**" koos vea tüübiga. Juhul kui päring õnnestus, kuvatakse uued andmed **body** siltide vahel.

Nagu lahenduse koodis on näha (vt Koodinäide 4) siis puudub seal **options** objekt. Tegemist

on lühendiga, kus jQuery lisab määratud seaded automaatselt **options** objekti alla.

### 3.3 Ajaxi kasutamine üle terve lehe, kus tehakse mitmeid päringuid.

Eesmärgiks on luua vaikimisi seaded veebilehele, kus ühe lehe ulatuses tehakse mitmeid asünkroonseid päringuid serveriga.

Lahenduseks on alljärgnev kood:

```
1. <script type="text/javascript">
2.     $(document).ready(function() {
3.         $('#laadimisIndikaator')
4.             .bind('ajaxStart', function() {
5.                 $(this).show();
6.             })
7.             .bind('ajaxComplete', function() {
8.                 $(this).hide();
9.             });
10.        $.ajaxSetup({
11.            cache: true,
12.            dataType: 'json',
13.            error: function(xhr, status, error) {
14.                alert('Tekkis viga: ' + error);
15.            },
16.            timeout: 60000,
17.            type: 'POST',
18.            url: ''
19.        });
20.    });
21. </script>
```

*Koodinäide 5: jQuery Ajaxi vaikimisi seaded*

Töötades suuremate veebirakendustega on mugavam defineerida Ajaxi päringuteks vaikimisi seaded. Tehes nii, lähevad kõik päringud läbi ühe värava, kus juhitakse kõiki sinna saabuvaid protsesse. Juhul kui individuaalse päringuga on defineeritud eraldi seaded siis toimub ühendus nende baasil - vaikimis seaded kirjutatakse lihtsalt üle.

```
$('#laadimisIndikaator')
    .bind('ajaxStart', function() {
        $(this).show();
    })
    .bind('ajaxComplete', function() {
```

```
$(this).hide();  
});
```

**ajaxStart** näol on tegemist funktsiooniga, mille käivitamisel tuuakse nähtavale element **#laadimisIndikaator**. Antud funktsioon käivitub juhul kui lehel toimub mõni Ajaxi päring. Tegemist on siis lihtsa märguandega, mis teavitab kasutajat, et leheküljel toimub andmete uuendamine. Kui kõik Ajaxi päringud on lõpetatud siis peidetakse ka indikaator, kasutades selleks **ajaxComplete** funktsiooni.

```
$.ajaxSetup({  
  cache: true,  
  dataType: 'json',  
  error: function(xhr, status, error) {  
    alert('Tekkis viga: ' + error);  
  },  
});
```

Funktsiooniga **\$.ajaxSetup** seadistatakse siis vaikimisi seaded. Tegemist on võtme-väärtuste paaride kogumikuga, mille parameetrite hulk võib varieeruda vastavalt kasutamissoovile. Parameeter **cache** määrab, et veebilehitseja võib antud rakenduse puhul kasutada puhvermälu ning **dataType** defineerib failitüübi, mida eeldatakse serverist saada vastuseks. Tagasikutsefunktsioon **error** käivitatakse juhul kui päring ebaõnnestus.

```
timeout: 60000,  
type: 'POST',  
url: 'sisu.js'  
});
```

Parameeter **timeout** määrab aja, mille ületamisel loetakse antud päring ebaõnnestunuks. Antud näites on selleks kuus sekundit. Sättega **url** viidatakse failile, kus kohast hakatakse andmeid kuvama.

Kui vaikimisi seaded on määratud siis tulevaste päringute vormistamine muutub igati kergemaks. Järgnevalt näide päringu vormistamiseks, kui vaikimisi seaded on määratud (vt Koodinäide 6).

```

1. $.ajax({
2.     data: {
3.         // küsitavad andmed
4.     }
5. });

```

*Koodinäide 6: Päringu vormistamine vaikimisi seadetega*

### 3.4 Lihtsustamata JavaScripti asünkroone päring

Eesmärgiks on luua asünkroone ühendus serveriga, kasutades selleks traditsioonilist lihtsustamata JavaScripti.

Lahenduseks on järgnev kood:

```

1. <script type="text/javascript">
2.   window.onload = function loadXMLDoc() {
3.     var xmlhttp;
4.     if (window.XMLHttpRequest) {
5.       xmlhttp=new XMLHttpRequest();
6.     }
7.     else {
8.       xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
9.     }
10.    xmlhttp.onreadystatechange=function() {
11.      if (xmlhttp.readyState==4 && xmlhttp.status==200) {
12.        document.getElementById("content").innerHTML+xmlhtt
13.        p.responseText;
14.      }
15.      xmlhttp.open("GET","test.html",true);
16.      xmlhttp.send();
17.    }
18.  </script>

```

*Koodinäide 6: Lihtsustamata JavaScripti Ajaxi päring*

Kui võrrelda traditsioonilise JavaScripti ja jQuery baasil tehtud päringut siis JavaScripti oma on kindalt mahukam ning koosneb enamatest sammudest, kuid samas ei saa neid siiski üks ühele võrrelda. jQuery käsustik on küll lihtsam ja kiiremini õpitavam, kuid samas on reaalne loogika ja sammude hulk peidetud - näha on vaid jäämäe tipp.

JavaScripti päringu tegemiseks tuleb esmalt luua **XMLHttpRequest** objekt, mis on uuemates lehitsejates kenasti teostatav läbi samanimelise funktsiooni (vt Koodinäide 6, rida 5). Vanemad Internet Explorer lehitsejad seda funktsiooni paraku ei tunnista. Selle jaoks tuleb luua eraldi kontroll, mis selgitaks välja **XMLHttpRequest** funktsiooni olemasolu. Juhul, kui see puudub, siis kasutatakse **ActiveX** elemendil põhinevat lahendust, mis kokkuvõttes teeb sama välja (vt Koodinäide 6 rida 8).

Kui **XMLHttpRequest** objekt on loodud siis järgnevateks sammudeks on:

- **XMLHttpRequest** objekti avamine **open** meetodiga (vt Koodinäide 6, rida 15).
- Päringu teele saatmine meetodiga **send** (vt Koodinäide 6, rida 16).
- Sündmuse kuulamine **onreadystatechange** (vt Koodinäide 6, rida 10) abil ja kui **readyState** väärtus on **4** siis kuvatakse andmed.

## 4. Testimine

Käesolev peatükk koosneb neljast testist, mis viiakse läbi eelmise peatüki rakenduste põhjal.

Lisainfo:

- Jõudluse (ing. k. *perfomance*) all on mõeldud, kui kaua läheb lehitsejal aega rakenduse käsustiku täitmiseks. Tulemused on esitatud millisekundites [1 sekundis (s) = 1000 millisekundit (ms)].
- Igas testis on kasutatud jQuery versiooni 1.5.2.

### 4.1 jQuery teegifaili viide lehe päises versus lehe alaosas

Esimese testi eesmärgiks on võrrelda, kuidas on seotud teegifaili viite paigutamine ja reaalajarakenduse jõudlus. Selle väljaselgitamiseks tuleb teha kaks testi: esimesel juhul asub viide lehe päises, teisel korral üks rida ennem rakendust.

#### Keskkond ja vahendid:

- Testimine toimub lokaalses tööjaamas, kasutades selleks WAMP serverit.
- Testimiseks kasutatakse Mozilla Firefox lehitsejat.
- Jõudluse mõõtmiseks kasutatakse Firebugi-i JavaScripti konsooli, mis mõõdab, kui kaua kulub lehitsejal aega (ms) skripti rakendamiseks.

#### Kriteeriumid:

- Mõlemal juhul kasutatakse identset eelmiseses peatükis (vt alapeatükk 3.2) tutvustatud *Ajax()* meetodil põhinevat päringut, kuid ainuke vahe on teegifaili viite asukohas. Esimeses testis on viide lehe päises (*head* märgendite vahel). Teises testis asub üks rida ennem skripti (*body* märgendite vahel).

#### Testimine:

- Viis kordust mõlema lahendusega, mille järel saab arvutada keskmise.



**Tulemused:**

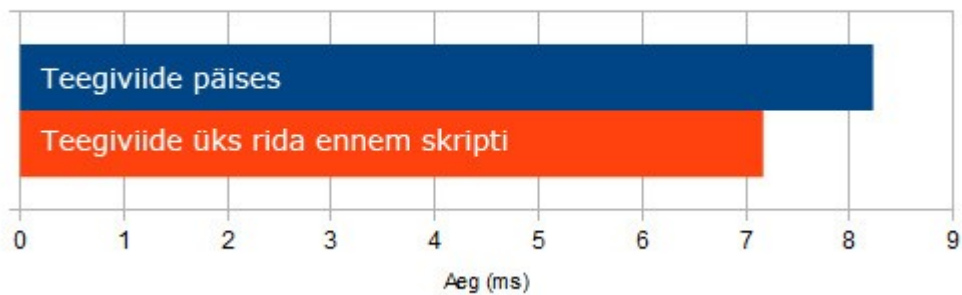
Test	Aeg (ms)
#1	8,378
#2	8,043
#3	8,143
#4	8,264
#5	8,297

*Tabel 1: Test 1 tulemused: Teegiviide lehe päises*

Test	Aeg (ms)
#1	7,034
#2	7,113
#3	7,151
#4	7,236
#5	7,288

*Tabel 2: Test 2 tulemused: Teegiviide üks rida ennem skripti*

**Keskmine skripti rakendamiseks kulunud aeg:**



*Joonistus 1*

## 4.2 JavaScripti päring versus jQuery päring

### 4.2.1 Jõudluse võrdlus

Järgneva testi eemärgiks on võrrelda lihtsustamata JavaScripti (vt alapeatükk 3.4) ja jQuery *Ajax()* päringu (vt alapeatükk 3.2) omavahelist jõudlust.

#### Keskkond ja vahendid:

- Testimine toimub lokaalses tööjaamas, kasutades selleks WAMP serverit.
- Testimiseks kasutatakse Mozilla Firefox lehitsejat.
- Jõudluse mõõtmiseks kasutatakse Firebug-i JavaScripti konsooli.

#### Kriteeriumid mõlemale rakendusele:

- Andmed kuvatakse koheleht lehe laadimisel.
- Andmeteks on fail *test.html*, mis koosneb tekstist ja ühest pildist (vt Lisa 1).
- Veateated on koodist eemaldatud.
- jQuery teegifaili viide asub skriptist eelneval real (kehtib siis ainult jQuery rakendusele).

#### Testimine:

- Viis kordust mõlema rakendusega, mille järel saab arvutada keskmise.

#### Tulemused:

Test	Aeg (ms)
#1	0,575
#2	0,662
#3	0,682
#4	0,678
#5	0,564

Tabel 3: Test 3 tulemused: Lihtsustamata JavaScript



Joonistus 2

#### 4.2.2 Andmemahu võrdlus

Eemärgiks on välja selgitada kahe lahenduse (alapeatükk 3.2 versus 3.4) andmemahulised erinevused. Läbivaks küsimuseks on, kui kaua võtab lehitsejal aega, et kuvada terve veebileht? Eelnevas testis võrreldi, kui kaua võtab lehitsejal aega skriptikoodi lugemine. Antud olukorras lisanduvad veel järgmised sammud:

- Serveriga ühenduse loomine
- Failide ületamine
- Andmete liigendamine algelementideks - DOM-i loomine ja selle kuvamine.
- Piltide visualiseerimine.

#### Keskkond ja vahendid:

- Mõlemad rakendused on laetud serverisse ja on läbi internetiühenduse kättesaadavad.
- Testimiseks kasutatakse Mozilla Firefox lehitsejat.
- Andmeedastuse mõõtmiseks kasutatakse Firebugi-i Net liidest.

#### **Kriteeriumid mõlemale rakendusele:**

- Andmed kuvatakse koheselt lehe laadimisel.
- Andmeteks on fail *test.html* (vt Lisa 1), mis koosneb tekstist (922 tähemärki) ja ühest pildist (andmemahut kokku on 47,5 KB). Fail koos pildiga on samas kataloogis kus rakenduski.
- HTTP päringu meetodiks on GET.
- Puhvermälu (ing. k. *cache*) kasutamine on lubatud.
- Veateated on koodist eemaldatud.

#### **Kriteeriumid jQuery rakendusele:**

- jQuery teegifaili viide asub skriptist eelneval real.
- Teegifail loetakse Google Labs serverist ja on kokkupakitud (ing. k. *gzipped*) (url: <https://ajax.googleapis.com/ajax/libs/jquery/1.5.2/jquery.min.js>). Kokkupakitud teegifaili suuruseks on 29KB.

#### **Testimine:**

- Mõlema lahenduse andmemahu välja selgitamine.
- Mõlema lahenduse internetist kuvamiseks kulunud aja mõõtmine (viis kordust).

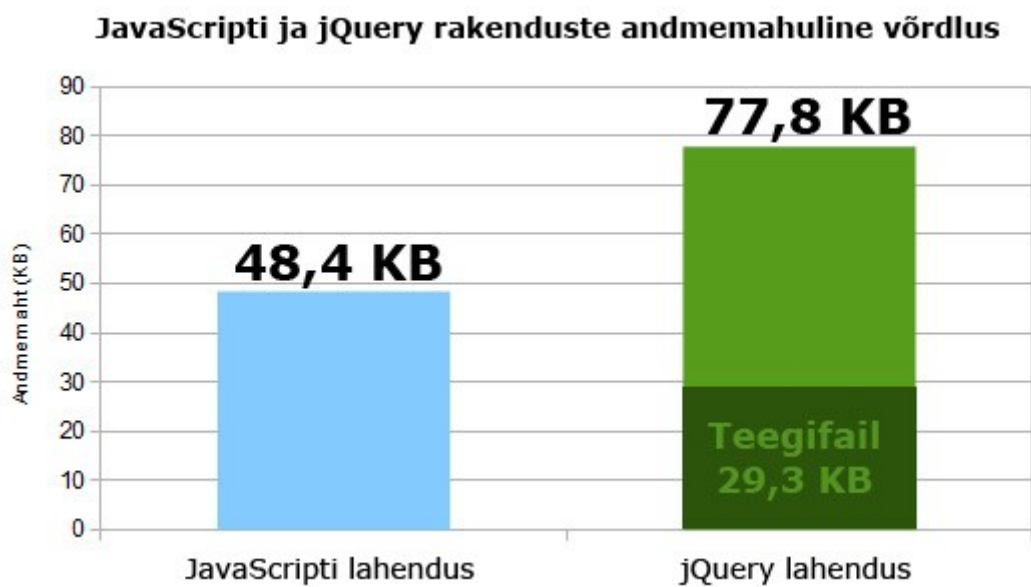
## Tulemused:

Test	Aeg (ms)
#1	374
#2	387
#3	376
#4	374
#5	368

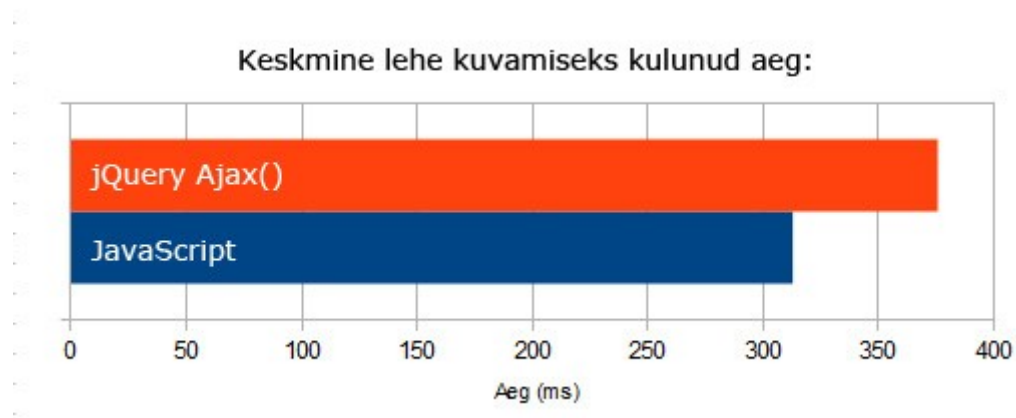
Tabel 4: Test 4 tulemused: jQuery rakenduse kuvamiseks kulunud aeg

Test	Aeg (ms)
#1	303
#2	304
#3	333
#4	316
#5	309

Tabel 5: Test 5 tulemused: JavaScripti rakenduse kuvamiseks kulunud aeg



Joonistus 3



Joonistus 4

### 4.3 Ühilduvus eri lehitsejatega

Eesmärgiks on testida jQuery *Ajax()* meetodit erinevate vanema põlvkonna veebilehitsejatega.

#### Keskkond ja vahendid:

- Windows 7 operatsioonisüsteem.
- Lisaks Browsershots veebilehitsejate kaugtestimiskeskond (ing. k. *remote testing service*) (url: <http://browsershots.org/>).

#### Kriteeriumid:

- jQuery *Ajax()* meetodiga andmete kuvamine lehe esmasel laadimisel.
- Andmeteks on fail *test.html*, mis koosneb tekstist ja ühest pildist (vt Lisa 1).
- HTTP päringu meetodiks on GET.

#### Testimine:

- Veebilehe testimine erinevate lehitsejatega, kas lokaalses tööjaamas või Browsershots keskkonnas. Kui andmeid ei kuvata siis läheb kirja negatiivne tulemus.

#### Tulemused:

<b>Lehitseja väljalaskeaasta</b>	<b>Lehitseja</b>	<b>Tulemus</b>
2000	Opera 5.0	-
2001	Internet Explorer 6.0	+
2002	Netscape Navigator 7.2	+
2004	Mozilla Firefox 1.0	+
2005	Seamonkey 1.1	+
2007	Safari 3.1	+

*Tabel 6: jQuery Ajax() ühildumine vanemate veebilehitsejatega*

## Kokkuvõte

Käesolevas töös valmis ülevaade jQuery populaarsematest moodulitest, mille baasil on võimalik luua interaktiivseid ja praktilisi reaalajarakendusi. Kokkuvõtteks võib öelda, et olemasolevate lahenduste hulk on küllaltki suur. Esindatud on kõik levinumad Ajaxi lahendused ning seda väga kergesti ja kiiresti rakendataval kujul.

Kui võrrelda jQuery ja lihtsustamata JavaScripti süntaksit ja loogikat siis võib kindlalt öelda, et jQuery on kiiremini omandatav. jQuery käsitusviisi võib illustreerivalt vaadata kui klotside ladumist üksteise peale. Antud lähenemine võimaldab luua väga kiireloomulisi arendustöid - lühikese kuid funktsionaalse koodijupiga annab palju korda saata.

Testide tulemused näitasid, et Ajaxi päringute tegemisel on tavalise JavaScripti jõudlus parem. Lisaks kehvemale jõudlusele on jQuery ka andmemahult suurem, kuna lehega kaasneb alati ka teegifail. Seega ei ole jQuery kasutamine lihtsamate Ajaxi lahenduste loomiseks alati põhjendatud.

Lisaks said testimisel kinnitust järgnevad seisukohad:

- Skriptide paigutamine lehe alaosasse vähendab nende töötlemiseks kuluvat aega.
- jQuery Ajaxi lahendused töötavad ka enamustes vanema põlvkonna veebilehitsejates.



## **Summary**

The aim of this work is to introduce jQuery solutions for making web-based real-time applications.

### **This work is divided into four major sections:**

1. Theoretical information about the subject
2. Examples of jQuery plugins that are based on Ajax solutions
3. Overview of jQuery methods for making real-time applications
4. Practical tests

### **Conclusional statements based on this work:**

- There are many well-written Ajax plugins for jQuery.
- jQuery is equipped with flexible and rich toolset for making Ajax applications.
- jQuery is easy to learn and supports rapid web development.
- Traditional JavaScript provides better performance and loadtime compared to jQuery-based solutions.
- Scripts in bottom of the page increase overall performance.
- jQuery supports Ajax methods in older browsers (in most cases).

## Kasutatud kirjanduse loetelu

- Crockford, D. (2010). *Really, JavaScript?*. Vaadatud 20.02.2011 aadressil [http://blip.tv/file/3755495?utm\\_source=player\\_embedded](http://blip.tv/file/3755495?utm_source=player_embedded)
- Brown, V. (2009). *Scripting Languages*. Loetud 20.02.2011 aadressil <http://www.mactech.com/articles/mactech/Vol.15/15.09/ScriptingLanguages/index.html>
- Bellis, M. (2010). *The History of JavaScript*. Loetud 20.02.2011 aadressil <http://inventors.about.com/od/jstartinventions/a/JavaScript.htm>
- Stefanon, S. (2008). *Object-Oriented JavaScript*. Birmingham, UK: Packt Publishing.
- JSON. (2011). *Introducing JSON*. Loetud 29.02.2011 aadressil <http://www.json.org/>
- Mozilla Developer Network. (2011). *XMLHttpRequest*. Loetud 5.03.2011 aadressil <https://developer.mozilla.org/en/XMLHttpRequest>
- Hopmann, A. (2007). *Story of XMLHTTP*. Loetud 6.03.2011 aadressil <http://www.alexhopmann.com/story-of-xmlhttp/>
- jQuery. (2011). *jQuery is a new kind of JavaScript Library*. Loetud 15.03.2011 aadressil <http://jquery.com/>
- W3Schools. (2011). *HTML DOM Tutorial*. Loetud 18.03.2011 aadressil <http://www.w3schools.com/html/dom/default.asp>
- W3C Homepage. (2011). *About W3C*. Loetud 27.03.2011 aadressil <http://www.w3.org/Consortium/>
- Garrett, J. (2005). *Ajax: A New Approach to Web Applications*. Loetud 27.03.2011 aadressil <http://www.adaptivepath.com/ideas/e000385>
- Korpela, J. (2001). *Methods GET and POST in HTML forms - what's the difference?*. Loetud 19.04.2011 aadressil <http://www.cs.tut.fi/~jkorpela/forms/methods.htm>

## Lisa 1

Testimiseks kasutatud fail *test.html*, mis koosneb tekstist (922 tähemärki) ja ühest pildist (*sun.jpg*). Andmemaht kokku on 47,5 KB.

