OULUN SEUDUN
AMMATTIKORKEAKOULU

Simo Hiltunen

TIETOJÄRJESTELMÄN TOTEUTUS GRAILS-SOVELLUSKEHYKSELLÄ

TIETOJÄRJESTELMÄN TOTEUTUS GRAILS-SOVELLUSKEHYKSELLÄ

Simo Hiltunen
Opinnäytetyö
Kevät 2012
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistojen kehityksen suuntautumisvaihtoehto

Tekijä: Simo Hiltunen
Opinnäytetyön nimi: Tietojärjestelmän toteutus Grails-sovelluskehyksellä
Työn ohjaaja: Eero Nousiainen
Työn valmistumislukukausi ja -vuosi: Kevät 2012 Sivumäärä: 62 + 10 liitettä

Tässä opinnäytetyössä toteutettiin työn tilaajayrityksen CCC Corporation Oy:n henkilöstön käyttöön web-pohjainen mökkien arvontajärjestelmä, joka toimii lisäksi varausjärjestelmänä mökeille. Järjestelmän tarkoituksena oli korvata vanha Lotus Notes -pohjainen järjestelmä, jolla työntekijät olivat osallistuneet yrityksessä järjestettyihin mökkiarpajaisiin. Näissä arpajaisissa arvonnan kohteena on osallistujan haluama aikäväli, jolloin hän haluaa varata mökin käyttöönsä.

Järjestelmä toteutettiin Grails-sovelluskehyksellä ja järjestelmän käyttöliittymien toteutukseen käytettiin ExtJS JavaScript -sovelluskehystä. Järjestelmä kehitettiin NetBeans-kehitysympäristöllä, jolla pystytään toteuttamaan Grails-sovelluksia. Kehitystyö tehtiin Windows-käyttöjärjestelmäympäristössä, johon oli asennettu XAMPP-palvelinohjelmisto, jolloin voitiin käyttää ulkopuolista MySQL-tietokantaa. Kehitystyöprosessin hallintaan käytettiin Scrumia, jonka tavoista ja käytännöistä muodostettiin tähän projektiin sopiva tapa hallita kehitysprosessia. Scrumia ei voitu käyttää sellaisenaan, koska projektissa työskenteli pääsääntöisesti vain yksi henkilö. Scrum antoi tästä huolimatta järkevän tavan edetä ja hallita ohjelmistonkehitysprosessia.

Järjestelmä toteutettiin siihen suunniteltujen ominaisuuksien osalta, joille suunnitteluvaiheessa oli annettu korkein prioriteetti, jolloin järjestelmää pystytään käyttämään. Järjestelmän kohdeympäristönä oli TurnKey Linux LAMP stack -ympäristö, johon jouduttiin asentamaan Apache Tomcat -servlet container, mihin järjestelmän asennus tapahtui.

Asiasanat: Grails, Scrum, tietojärjestelmät

ABSTRACT

Oulu University of Applied Sciences
Information Technology and Telecommunications, option of Software developer

Author: Simo Hiltunen
Title of thesis: Development of an Information System Using the Grails Framework
Supervisor: Eero Nousiainen
Term and year when the thesis was submitted: Spring 2012 Pages: 62 + 10 appendices

The objective of the thesis is to deal with implementation of information system using the Grails framework. Grails framework is a web framework for the Groovy programming language. Groovy is dynamic programming language and it has its roots in Java. Groovy language is designed for Java Virtual Machine and because of that it can be integrated seamlessly with Java at the syntax level. Purpose of this thesis is also to give a picture for the reader of the features that Grails framework is about. This thesis has also a chapter, which describes features of Groovy language and shows differences between Groovy and Java.

The thesis was commissioned by CCC Ltd. The product is the information system, which is used to make raffle for the cottage reservations between those employees who have left a lottery ticket in the system. The thesis also deals with relational database, which is usually used to store the data that is used in web applications. It also deals with Scrum, which principles were adapted for managing the development process. Scrum principles are explained in two different chapters. First chapter is dedicated for explaining the Scrum roles and it's documents, and the second one explains by examples how to proceed the Scrum process.

The goal of the thesis is to give the reader a broad understanding of this type of information system development process using the Grails framework. A broad understanding is needed to realize the fact that development is not just programming. Development process is the combination of project management, planning, design and implementation.

Keywords: Grails, Scrum, Development of an Information System

SISÄLLYS

SANASTO

GDK (Groovy Development Kit). Laajennus Javan ohjelmointirajapintaan, jonka asennuksella saadaan Groovyn ominaisuudet käyttöön.

GORM (Grails Object Relational Mapping). Grailsin tietokantarajapinta.

GSP (Groovy Server Pages). näkymiä muodostava tekniikka, jota käytetään Grails-sovelluksissa.

JVM (Java Virtual Machine). Javan virtuaalikone, jota tarvitaan Javalla tai Groovylla kirjoitettujen ohjelmien suorittamiseen.

MVC-malli (Model View Controller). Ohjelmistoarkkitehtuuri malli, jolla erotetaan sovelluksen käyttöliittymä, toiminta ja käsiteltävä data toisistaan.

Tavukoodi (Bytecode). Javan virtuaalikoneen suoritettavissa oleva ohjelmointikielestä käännetty koodi.

URI (Uniform Resource Identifier). Merkkijono, jolla osoitetaan tiedon paikkaa.

Web-säiliö (Web Container / Servlet container). Verkkopalvelimen osa, jota käytetään Java-pohjaisten web-sovellusten ajamiseen.

1 JOHDANTO

Työn tavoite oli perehtyä kokonaisvaltaisesti tietojärjestelmän toteutusprosessiin Grails-sovelluskehyksellä, josta käytetään myös nimitystä Grails. Työn toinen tavoite oli toteuttaa mökkien arvontajärjestelmä Grails-sovelluskehyksellä CCC Corporation Oy:lle. Järjestelmän tarkoituksena oli korvata vanha Lotus Notes -pohjainen järjestelmä, jota työntekijät käyttivät osallistuakseen yrityksessä järjestettyihin mökkiarpajaisiin. Näissä arpajaisissa arvonnan kohteena on osallistujan haluama aikäväli, jolloin hän haluaa varata mökin käyttöönsä.

Koska perimmäisenä tarkoituksena oli perehtyä kokonaisvaltaisesti ohjelmistonkehitysprosessin kulkuun ja hallintaan, eikä pelkästään siihen, miten jokin sovellus toteutetaan Grailsilla, aikaa käytettiin myös Scrum-menetelmän tutkimiseen. Scrumia sovellettiinkin järjestelmän toteutusprosessin hallinnassa.

Grails-sovelluskehyksen käyttöön järjestelmän toteutuksessa päädyttiin, koska Grails oli entuudestaan tuttu. Grails on avoimeen lähdekoodiin perustuva sovelluskehys, jota käytetään web-pohjaisten sovellusten kehittämiseen. Grailsissa kehitystyö tehdään Groovy-ohjelmointikielellä, joka on dynaaminen ohjelmointikieli. Työhön liittyvät myös relaatiotietokannat, joita käytetään web-pohjaisten sovellusten tietovarastoina. Grails-sovellukset käyttävät relaatiotietokantaa GORM-tietokantarajapinnan kautta domain-luokkien säilömiseen, jolloin on tärkeää ymmärtää, miten relaatiotietokannat toimivat. Domain-luokkien ja tietokantojen suunnittelulla onkin hyvin läheinen yhteys toisiinsa.

2 GROOVY-OHJELMOINTIKIELI

Groovy on dynaaminen ohjelmointikieli, jota voidaan joko tulkita tai kääntää niin kuin perinteisiä ohjelmointikieliä kuten C++. Se on saanut vaikutteita Ruby-, Python- ja Smalltalk-ohjelmointikielistä, kuten myös Javasta. Groovy on suunniteltu erityisesti Java-alustalle toisin kuin Javalle muutetut kielet. Groovy on suunniteltu Javan virtuaalikonetta varten ja näin Groovy tukeutuu Javan ohjelmointirajapintaan. Koska Groovy on rakennettu Javan virtuaalikoneelle, tiukalla tavukooditason integroinnilla ne ovat näin ollen keskenään täysin yhteensopivia. (Judd – Nusairat – Shingler 2008, 30.)

Groovysta voidaan koostaa Java-luokkatiedosto, joka näin ollen voi olla osa suurempaa Javalla toteutettua kokonaisuutta (Rocher 2006, 26). Groovya voidaan oikeastaan pitää laajennuksena Javan ohjelmointirajapinnalle, koska Groovy Development Kit (GDK) lisää Javan jo olemassa olevien luokkien toiminnallisuuksia. Groovyssa on kuitenkin useita ominaisuuksia, joilla se eroaa Javasta. Java on kasvanut vuosien kuluessa tukemaan erilaisia alustoja. Sillä voidaan toteuttaa erilaisia sovelluksia sulautetuista työpöytäsovelluksiin. Näin ollen myös Groovy-ohjelmointikieltä voidaan käyttää samojen sovellusten toteuttamiseen. (Judd ym. 2008, 17, 30.)

Kaikki Groovylla toteutetut luokat ovat tavukooditasolla samanlaisia kuin Javalla toteutetut. Ne eivät siis eroa toisistaan Java Virtual Machine -tasolla ja ovat näin keskenään yhteensopivia. Luokat määritellään samalla tavoin kuin Javassa. Ne ovat myös rakenteeltaan ja ominaisuuksiltaan samanlaiset kuin Javassa. Groovy ja Java kuitenkin eroavat toisistaan syntaksiltaan. Groovyssa myös oletusarvoisesti määritellään kaikki näkyvyydeltään julkisiksi, ellei näkyvyyttä määritellä toisin. (Groovy.codehaus.org. 2011.)

2.1 Syntaksi

Optionaalinen tyypitys

Staattisesti tyypitetyissä ohjelmointikielissä, kuten esimerkiksi Javassa muuttujan tyyppi on määriteltävä, ennen kuin muuttujaan voidaan asettaa tietotyyppejä. Groovyssa voidaan muuttujat tyypittää staattisesti tai voidaan jättää määrittelemättä. Groovy tukee staattista ja dynaamista muuttujan tyypitystä, jolloin muuttujien tyypitystä voidaan pitää optionaalisena. Seuraavassa on esitetty kaksi esimerkkiä merkkijonon tyypityksistä:

1) String str1 = "I'm a String"
2) str2 = "I'm also a String".

Kumpikin yllä kuvatuista tavoista on Groovyssa hyväksyttyjä tapoja tyypittää merkkijono. Ensimmäisessä kuvataan staattista toteutusta, jolloin muuttujan tyyppi on määritelty merkkijonoksi. Toisessa kuvataan muuttujan dynaamista tyypittämistä, jolloin muuttujan tyyppiä ei tarvitse määritellä. (Dearle 2010, 18.)

Sulkeumat

Sulkeumilla tarkoitetaan sellaista ensimmäisen luokan funktiota, joka voidaan palauttaa paluuarvona funktiolta tai välittää parametrina funktiolle. Se muistaa myös viiteympäristön, jossa se on määritelty. (Enberg & Raunio 2007.) Erona normaaliin funktioon tai metodiin voidaan pitää sitä, että sulkeuma on objekti, kun taas normaalit funktiot tai metodit eivät itsessään ole (Judd ym. 2008, 24). Groovy tukee sulkeumia ja ne muodostetaan aaltosulkuihin, kuten esimerkissä (kuva 1).

```
def name = "Chris"
def printClosure = { println "Hello, ${name}" }

printClosure()

name = "Joseph"
printClosure()
```

```
Hello, Chris
Hello, Joseph
```

KUVA 1. Sulkeumat ja niiden käyttö (Judd ym. 2008, 24)

Kuvassa 1 sulkeuma käsittelee samassa viiteympäristössä määriteltyä muuttujaa ilman, että se olisi jotenkin välitetty sulkeumalle. Sulkeumille voidaan myös välittää parametreja, kuten kuvassa 2 on esitetty.

```
def printClosure = {name -> println "Hello, ${name}" }

printClosure("Chris")
printClosure("Joseph")
printClosure "Jim"
```

```
Hello, Chris
Hello, Joseph
Hello, Jim
```

KUVA 2. Parametrien välittäminen sulkeumille (Judd ym. 2008, 24–25)

Kuvassa 2 esitetty kolmas ja viimeinen kutsu, jossa ei ole käytetty sulkuja ei ole kirjoitusvirhe. Siinä osoitetaan Groovyssa käytettävän syntaksin optionalisuutta. Sulkeumille voidaan välittää myös monta parametria samanaikaisesti, kuten kuvassa 3 osoitetaan. (Judd ym. 2008, 25.)

```
def printClosure = {name1, name2, name3 -> println "Hello, ${name1},
    ${name2}, ${name3}" }

printClosure "Chris", "Joseph", "and Jim"
```

```
Hello, Chris, Joseph, and Jim
```

KUVA 3. Usean parametrin välittäminen sulkeumalle (Judd ym. 2008, 25)

2.2 Operaattorit

Kaikki Javan operaattoritaulukossa (liite 1) esitellyt operaattorit toimivat myös Groovyssa. Tämän lisäksi Groovyssa toimivat myös Groovyn operaattoritaulussa esitellyt operaattorit (liite 2). Groovyn ominaisuuksiin kuuluu kuitenkin operaattoreiden ylikuormittaminen. Tämän ominaisuuden ansiosta numeroiden, kokoelmien ja assosiaatiotaulujen kanssa työskentely helpottuu. (Groovy.codehaus.org. 2011). Operaattoreiden ylikuormittamisella tarkoitetaan sitä, että operaatio voidaan suorittaa myös muuttujille, jotka eivät ole numeerisia. Esimerkiksi yhteenlasku plus-operaattorilla voidaan suorittaa myös merkkijonolle, jolloin summana saadaan näiden kahden merkkijonon yhdistelmä. Operaattoreiden ylikuormittamisella muuttujat, jotka eivät ole numeerisia, saadaan käyttäytymään numeeristen muuttujien tavoin. Niille voidaan näin ollen suorittaa numeerisille muuttujille tarkoitettuja operaatioita. Groovyn operaattoreiden tulkinta objektien metodikutsuiksi esitellään liitteenä 3 olevassa taulukossa. (Dearle 2010, 15.)

2.3 Kokoelmat

Kaikki perustyyppiä monimutkaisemmat tietorakenteet on toteutettu olioiden avulla. Kaikkia sellaisia tietorakenteita, joihin säilötään olioita, kutsutaan kokoelmiksi. Kokoelmien tarkoitus on helpottaa ohjelmoijan työskentelyä tietorakenteiden parissa. Groovyssa on toteutettu monta eri luokkaa, joita voidaan käyttää tietorakenteita käsiteltäessä. Näiden kokoelmien implementoinnit ovat suurimmalta osalta peräisin Javasta, mutta niiden käsittelyssä on eroavaisuuksia Javan käsittelytapaan verrattuna. (Judd – Nusairat – Shingler 2008, 27–33.)

Lista (list)

Lista on järjestelmällinen kokoelma oliota, jonka tietoalkiot on varastoitu lineaariseen järjestykseen. Groovyssa listan luominen tapahtuu käyttämällä hakasulkeita. Groovyn lista on toteutus java.util.List-olion rajapinnasta. Kuvassa 4 on esimerkkejä listan luomisesta Groovylla. (Judd ym. 2008, 55.)

```
01 def emptyList = []
02 println emptyList.class.name //  java.util.ArrayList
03 println emptyList.size // 0
04
05 def list = ["Chris"]  // List with one item in it
06 // Add items to the list
07 list.add "Joseph"  // Notice the optional () missing
08 list << "Jim"  // Notice the overloaded left-shift operator
09 println list.size // 3
10
11 // Iterate over the list
12 list.each { println it } // Chris Joseph Jim
13
14 // Access items in the list
15 println list[1]  // Joseph  // Indexed access
16 list[0] = "Christopher"
17 println list.get(0)  // Christopher
18
19 list.set(0, "Chris") // Set the 0 item to Chris
20 println list.get(0)  // Chris
21
22 list.remove 2
23 list-= "Joseph"  // Overloaded - operator
24 list.each { println it } // Chris
25
26 list.add "Joseph"
27 list+="Jim" // Overloaded + operator
28 list.each { println it } // Chris Joseph Jim
29 println list[-1]  // Jim
```

KUVA 4. Listan luonti ja operaatiota (Judd ym. 2008, 55–56.)

Alue (range)

Alueita voidaan käyttää minkä tahansa Java-olion kanssa, joissa on toteutettu java.lang.Comparable-vertailutyökalu. Olioissa on oltava myös metodit seuraavaan ja edelliseen arvoon alueessa. Alue on lista peräkkäisistä arvoista, jotka voidaan ajatella ykkösestä kymmeneen tai kirjaimesta a kirjaimeen z. Näiden avulla voidaan luoda luettelo peräkkäisistä arvoista. Näitä arvoja voidaan käsitellä, kuten listoja. Kuvassa 5 on esitelty niiden käyttöä. (Groovy.codehaus.org. 2011.)

```
01 def numRange = 0..9
02 println numRange.size() // 10
03 numRange.each {print it}  // 0123456789
04 println ""
05 println numRange.contains(5)  // true
06
07 def alphaRange = 'a'..'z'
08 println alphaRange.size()   // 26
09 println alphaRange[1]    // b
10
11 def exclusiveRange = 1..<10
12 println exclusiveRange.size() // 9
13 exclusiveRange.each {print it}  // 123456789
14 println ""
15 println exclusiveRange.contains(10)  // false
16
17 def reverseRange = 9..0
18 reverseRange.each {print it} // 9876543210
```

KUVA 5. Alueen luonti ja operaatioita (Judd ym. 2008, 28-29)

Taulukko (Array)

Groovyn taulukot ovat samanlaisia, peräkkäisiä objekteja sisältäviä tauluja, kuten Javassa. Groovyssa niiden käsittely on hieman yksinkertaisempaa kuin Javassa. Käsittelyä on esitelty kuvassa 6. (Judd ym. 2008, 31.)

```
01 def stringArray = new String[3]
02 println stringArray.size()
03 stringArray[0] = "Chris"
04 println stringArray  // {"Chris", null, null}
05 stringArray[1] = "Joseph"
06 stringArray[2] = "Jim"
07 println stringArray // {"Chris", "Joseph", "Jim"}
08 println stringArray[1] // Joseph
09 stringArray.each { println it} // Chris, Joseph, Jim
10 println stringArray[-1..-3] // ["Jim", "Joseph", "Chris"]
```

KUVA 6. Taulukon luonti ja operaatioita (Judd ym. 2008, 31)

Joukko (set)

Joukolla tarkoitetaan sellaista tietorakennetta, joka on joukko joitain alkioita, joiden järjestyksellä ei ole merkitystä. Joukon tarkoituksena on sellaisen tietorakenteen toteuttaminen, jossa merkitys on ainoastaan sillä, kuuluuko alkio joukkoon. Groovyssa joukko on oletuksena HashSet-tyyppinen. Joukon käsittelyä on kuvattu kuvassa 7. (Judd ym. 2008, 30.)

```
01 def emptySet = [] as Set
02 println emptySet.class.name //  java.util.HashSet
03 println emptySet.size() // 0
04
05 def list = ["Chris", "Chris" ]
06 def set = ["Chris", "Chris" ] as Set
07 println "List Size: ${list.size()} Set Size: ${set.size()}" // List Size: 2 Set
Size: 1
08 set.add "Joseph"
09 set << "Jim"
10 println set.size() // 3
11 println set  // ["Chris", "Jim", "Joseph"]
12
13 // Iterate over the set
14 set.each { println it }
15 S
16 set.remove 2
17 set-= "Joseph"  // Overloaded - operator
18 set.each { println it } // Chris
19 set+= "Joseph"
20 set+= "Jim"
21 set.each { println it } // Chris Joseph Jim
22
23 // Convert a set to a list
24 List = set as List
```

KUVA 7. Joukon luonti ja sen käyttöä (Judd ym. 2008, 30)

Assosiaatiotaulu (map)

Groovyn assosiaatiotaulu on järjestelemätön kokoelma avain- ja arvopareja. Näistä kahdesta avain on uniikki kuten Javassa. Groovyssa assosiaatiotaulu on oletuksena LinkedHashMap-tyyppinen. Assosiaatiotaulun käyttöä kuvataan kuvassa 8. (Judd ym. 2008, 32.)

```
01 def emptyMap = [:]
02 // map.class returns null, use getClass()
03 println emptyMap.getClass().name  //java.util.LinkedHashMap
04 println emptyMap.size() // 0
05
06 def todos = ['a':'Write the map section', 'b':'Write the set section']
07 println todos.size() // 2
08 println todos["a"] // Write the map section
09 println todos."a"  // Write the map section
10 println todos.a    // Write the map section
11 println todos.getAt("b") // Write the set section
12 println todos.get("b")   // Write the set section
13 println todos.get("c", "unknown") //  unknown, Notice "c" wasn't defined
14                                    // and now it is
15 println todos // ["a":"Write the map section", "b":"Write the set section",
16               // "c":"unknown"]
17
18 todos.d = "Write the ranges section"
19 println todos.d // Write the ranges section
20 todos.put('e', 'Write the strings section')
21 println todos.e // Write the strings section
22 todos.putAt 'f', 'Write the closure section' // Notice () are optional
23 println todos.f // Write the closure section
24 todos[null] = 'Nothing Set'  // Using null as a key
```

KUVA 8. Assosiaatiotaulun luonti ja operaatioita (Judd ym. 2008, 32)

2.4 Kontrollirakenteet

Haaroitus ja silmukointi ovat ohjelmoinnissa käytettyjä kontrollirakenteita, jotka ovat tuttuja kaikista ohjelmointikielistä. Näiden rakenteiden tarkoitus on ohjata ohjelman suoritusta niissä esitettyjen ehtojen mukaan. Jokainen ohjelmointikieli sisältää samat kontrollirakenteet. Kontrollirakenteiden toteuttaminen voi erota toisistaan eri ohjelmointikielillä tai niiden toteutus voidaan suorittaa monella tavalla. Tässä luvussa esitellään Groovyssa käytetyn loogisen haaroituksen sekä silmukoinnin toteutustapoja. Groovy tukee Javan normaalia *if else* -syntaksia kuten myös sisäkkäisiä *if-, else if-, else*-rakenteita. Groovy tukee myös tenääristä eli kolmannen komponentin operaattoria, jolla voidaan muodostaa kolmiosainen toiminnallisuus lauseeseen. Tenäärisenä operaattorina Groovyssa on kysymysmerkki, jolla voidaan muodostaa eräänlainen ehtolause ohjelmakoodiin. Esimerkissä asetetaan kaksoispisteen oikean- tai vasemmanpuoleinen merkkijono muuttujaan sen mukaan, miten suluissa oleva ehto täyttyy (kuva 9). (Groovy.codehaus.org. 2011.)

```
def y = 5
def x = (y > 1) ? "worked" : "failed"
// x == worked

def y = 0
def x = (y > 1) ? "worked" : "failed"
// x == failed
```

KUVA 9. Tenäärinen operaattori (Groovy.codehaus.org. 2011)

Valintalause

Groovyssa käytetty valintalause pystyy käsittelemään minkä tahansa sille annetun arvon tyypistä riippumatta. Tällä tavoin arvolle voidaan suorittaa erilaisia vertailuja lauseen sisällä. Kuvassa 10 on esitelty valintalauseen käyttöä. (Groovy.codehaus.org. 2011.)

```
def x = 1.23
def result = ""

switch ( x ) {
    case "foo":
        result = "found foo"
        // lets fall through

    case "bar":
        result += "bar"

    case [4, 5, 6, 'inList']:
        result = "list"
        break

    case 12..30:
        result = "range"
        break

    case Integer:
        result = "integer"
        break

    case Number:
        result = "number"
        break

    default:
        result = "default"
}

assert result == "number"
```

KUVA 10. Valintalause (Groovy.codehaus.org. 2011)

Silmukointi

Groovyssa silmukointi voidaan toteuttaa samalla tavalla kuin muissa ohjelmointikielissä. Groovyssa silmukointia voidaan toteuttaa yksinkertaisemmin ja eri tavoin. Kuvassa 11 kuvataan eri tapoja siitä, millä tavoin silmukointia voi toteuttaa Groovyssa.

```
// for (int i = 0; i < 5; \++i) // not implemented by beta-10.

// iterate over a range
def x = 0
for ( i in 0..9 ) {
    x += i
}
assert x == 45

// iterate over a list
x = 0
for ( i in [0, 1, 2, 3, 4] ) {
    x += i
}
assert x == 10

// iterate over an array
array = (0..4).toArray()
x = 0
for ( i in array ) {
    x += i
}
assert x == 10

// iterate over a map
def map = ['abc':1, 'def':2, 'xyz':3]
x = 0
for ( e in map ) {
    x += e.value
}
assert x == 6

// iterate over values in a map
x = 0
for ( v in map.values() ) {
    x += v
}
assert x == 6

// iterate over the characters in a string
def text = "abc"
def list = []
for (c in text) {
    list.add(c)
}
assert list == ["a", "b", "c"]
```

KUVA 11. Silmukoinnin toteutuksia (Groovy.codehaus.org. 2011)

3 GRAILS-SOVELLUSKEHYS

Grails on avoimeen lähdekoodiin perustuva sovelluskehys, jota käytetään web-sovelluksien kehittämiseen. Ohjelmointikielenä siinä käytetään Groovya. Grailsissa toteutettavat sovellukset toteutetaan MVC-ohjelmistoarkkitehtuuria käyttäen. (Judd ym. 2008, 64.) Siinä käytettävät domain-luokat viittaavat MVC-arkkitehtuurin model-luokkiin. Controllers-luokat vastaanottavat sovellukselle lähetettyjä pyyntöjä. Views-luokat taas toteuttavat tarvittavat näkymät. Arkkitehtuurikuvassa (kuva 12) päällimmäisenä on kuvattuna Grailsilla toteutettu sovellus ja sen alla Grails. Alimmaisena kuvataan Grailsin perustana toimiva Java virtuaalikone.

KUVA 12. Grails-arkkitehtuuri (Judd ym. 2008, 68)

Grailsissa pyritään välttämään turhaa ja ylimääräistä konfigurointia. Se on korvattu käytännöillä, joiden tarkoitus on helpottaa ohjelmoijan työtä ja näin kasvattaa sovelluskehyksen tuottavuutta. Grailsin periaatteena on käyttää Convention Over Configuration -mallia, joka tarkoittaa sitä, että sovelluskehyksessä noudatetaan tiettyjä käytäntöjä. Käytännöillä pyritään vähentämään konfiguroinnin aiheuttamaa ylimääräistä työtä. (Judd ym. 2008, 66). Esimerkiksi tätä periaatetta käytetään siinä, miten Grails järjestää sovelluksen MVC-mallin mukaiset luokat. MVC-mallin mukaisen sovelluksen controller-luokat menevät Controllers-hakemistoon, näkymät Views-hakemistoon ja domain-luokat Domains-hakemistoon. Jokainen toteutettu domain-luokka eli malli olettaa kontrollerinsa muodostuvan nimensä ja Controller-sanan yhdistelmästä. Kun käytetään tätä periaatetta, esimerkiksi domain-luokan foo controller-luokka löytyy silloin Controller-hakemistosta nimellä fooController.

Grailsia voidaan käyttää komentoriviltä tai integroida ohjelmistojenkehitysympäristöön. Grails sisältää erilaisia komentoja, joilla voidaan esimerkiksi luoda projekti tai generoida domain-luokille controller-luokat ja näkymä-luokat. Listaus Grails-sovelluskehyksen käyttämistä komennoista on liitteessä 1.

3.1 Ominaisuuksia

Grails sisältää Junit-sovelluskehyksen joka on suunniteltu yksikkötestausta varten. Yksikkötestauksessa ohjelma pilkotaan pieniin testattaviin yksiköihin ja varmistetaan niiden oikeanlainen toiminta. Grails generoi sovellukseen omat tests-luokat yksikkötestejä varten. Näin jokaisen luodun controller-luokan metodien oikeanlainen toiminta voidaan testata yksikkötestauksen avulla. Grails tukee myös scaffolding-menetelmää, jolla voidaan ohjelman käännösvaiheessa luoda domain-oliolle CRUD-operaatiot (create read update delete) (Judd ym. 2008, 68, 47-49, 65).

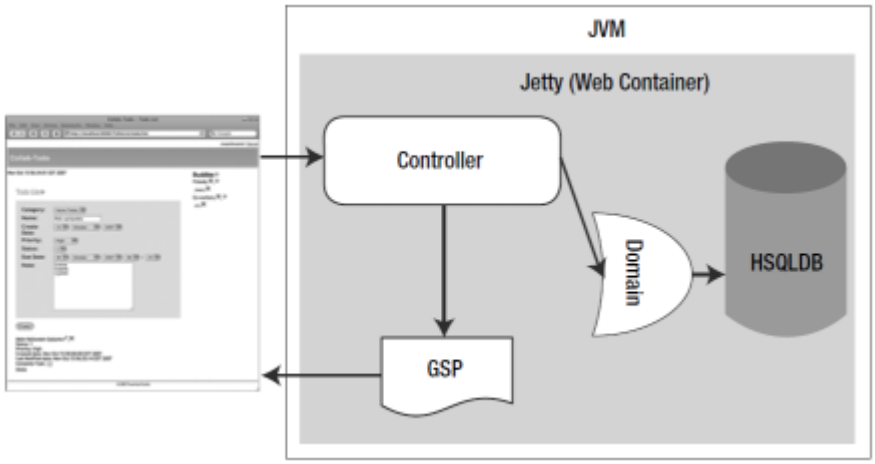
Grails hyödyntää SiteMesh-sovelluskehystä, jota käytetään HTML-sivun muodostamisessa. SiteMesh käyttää Decorator Pattern -mallia sivun muodostamisessa. Sivun muodostaminen tapahtuu yhdistelemällä eri layouteja ja templateja halutulla tavalla. (Rocher 2006, 192–193)

Grails sisältää myös kehitysvaiheessa käytettävän web-säiliön (web container). Kehitettävää sovellusta voidaan ajaa ja testata tässä ympäristössä. Grailsin 1.3.7 versiossa tähän tarkoitukseen on sulautettu Apache Tomcat -container. (Rocher – Ledbrook – Palmer – Brown – Daley – Beckwith 2011)

Grails sisältää myös HSQLDB-tietokannan. Sovelluskehykseen sulautettua HSQLDB-tietokantaa voi käyttää itsenäisenä tietokantana tai sulautettuna tietokantana. HSQLDB-tietokannan voi asettaa tallentamaan levylle tai toimimaan pelkän muistin varassa. Oletusasetuksena käytetään sulautettua ja muistinvaraista konfigurointia. (Judd ym. 2008, 68.)

3.2 Grails ja MVC-arkkitehtuurimalli

Grailsissa toiminnallisuus jaetaan kolmeen eri kategoriaan MVC-arkkitehtuurimallin periaatteen mukaan. Grailsin muodostama MVC-arkkitehtuuri kuvataan kuvassa 13. Grailsin MVC-arkkitehtuurimallin M eli model käsitetään Grailsissa domain-luokkana, jonka sisältämä data annetaan näkymän esitettäväksi. Controller-luokan tehtävänä on käsitellä WWW-selaimelta lähetettyjä pyyntöjä ja ohjata palveluita sekä muuta sovelluksen käytöstä. Näkymä-luokkina Grailsissa toimivat Groovy Server Pages (GSP). (Judd ym. 2008, 64.)



KUVA 13. Grails default runtime (Judd ym. 2008, 69)

MVC-arkkitehtuurimallin model

MVC-arkkitehtuurimallin model on Grailsissa domain-luoka. Domain-luokat sidotaan vastaamaan sovelluksen käyttämään tietokantaan. Ne ovat olioita, joiden tila säilyy ohjelman suorituskertojen välillä. Domain-luokat on kartoitettu tietokantaan siten, että luokan nimi muodostaa tietokantataulun nimen ja jokaisesta luokan sisältämästä attribuutista muodostuu tietokantataulun sisältämä sarake. (Rocher ym. 2011.)

MVC-arkkitehtuurimallin view

Groovy Server Pages (GSP) on teknologia, jota Grails käyttää näkymän muodostamisessa. GSP-luokka sisältää kuvauskieltä ja GSP-merkkejä, joista näkymä renderoidaan. Sillä on tyypillisesti myös malli, joka välitetään GSP-näkymään controller-luokan välityksellä. Mallin muuttujajoukkoa käytetään näkymän renderoinnissa WWW-selaimessa. (Rocher ym. 2011.)

MVC-arkkitehtuurimallin controller

Grailsissa jokainen controller-luokalle lähetetty pyyntö tarkastellaan erikseen. Tämä tarkoittaa sitä, että jokaista controller-luokan vastaanottamaa pyyntöä varten controller-luokasta luodaan uusi esiintymä, joka käsittelee vastaanotetun pyynnön. Controller-luokat voivat sisältävät useita sulkeuma-tyyppisiä metodeja, jotka Grailsissa käsitetään aktioksi (action). Jokainen aktio kartoitetaan URI-merkkijonoksi, josta aktio löytyy. Esimerkiksi my/list URI osoittaa myController-luokan sisältämään list-aktioon. (Rocher ym. 2011.)

3.3 Grails Object Relational Mapping (GORM)

GORM on Grailsin sisältämä Object Relational Mapping -teknologia, joka on rakennettu Hibernate 3 ORM -kirjaston päälle. Sitä käytetään domain-luokkien kuvaamiseen tietokannassa. Tällä teknologialla eri data-olioita voidaan tallentaa relaatiotietokantaan sekä hakea relaatiotietokannasta. Sen avulla Grails rakentaa tietokantataulut ja niiden sisältämät sarakkeet domain-olioille. GORM rakentaa myös tietokantataulujen väliset assosiaatiot tietokantaan. Näiden assosiaatioiden rakentamiseen käytetään domain-olioille määriteltyjä suhteita. (Rocher ym. 2011.)

Hibernate-istunto sidotaan Grailsissa automaattisesti sillä hetkellä suoritettavaan pyyntöön, jolloin kaikki GORMin sisältämät metodit ovat käytettävissä. Istunnot käyttävät välimuistia, jossa domain-olion käsittely tapahtuu. Tällä tavoin domain-oliolle suoritettavat toimenpiteet eivät välttämättä aiheuta SQL-operaatioita, vaan ne kerätään nippuun, josta ne suoritetaan istunnon päätteeksi. GORMin SQL-operaationippu voidaan kuitenkin tarvittaessa huuhtoa eli suo-

rittaa kesken istunnon, jolloin tietokanta ja istunnon välimuisti saadaan synkronoitua keskenään. (Rocher ym. 2011.)

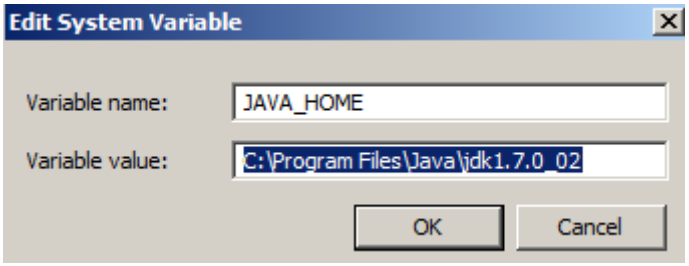
3.4 Liitännäiset

Grailsin liitännäisten tarkoitus on tarjota toiminnallisuuksia modulaarisina osina Grailsilla kehitettäviin sovelluksiin. Niitä liittämällä Grails-sovelluksiin saadaan toteutettua toiminnallisuuksia modulaarisesti tarvitsematta toteuttaa niitä itse. Ne tarjoavat ratkaisuja esimerkiksi autentikaation suorittamiseen. Tällaisten liitännäisten avulla toteutetaan modulaarinen ratkaisu järjestelmään kirjautumiselle. Tällaiset liitännäiset nopeuttavat sovellusten kehittämistä, koska niitä ei tarvitse suunnitella tai toteuttaa enää uudelleen. (Judd ym. 2008, 66.) Tätä kirjoitettaessa Grailsiin on saatavilla 748 liitännäistä, joista jokainen tarjoaa valmiin ratkaisun jollekin osa-alueelle (Grails 2012, linkit Plugins -> All). Liitännäisten tarjoamien modulaaristen ratkaisujen avulla saadaan nopeutettua sovelluksien kehittämistä Grailsissa (Grails 2012, linkit Documentation -> Reference -> 12 Plugins).

3.5 Asentaminen

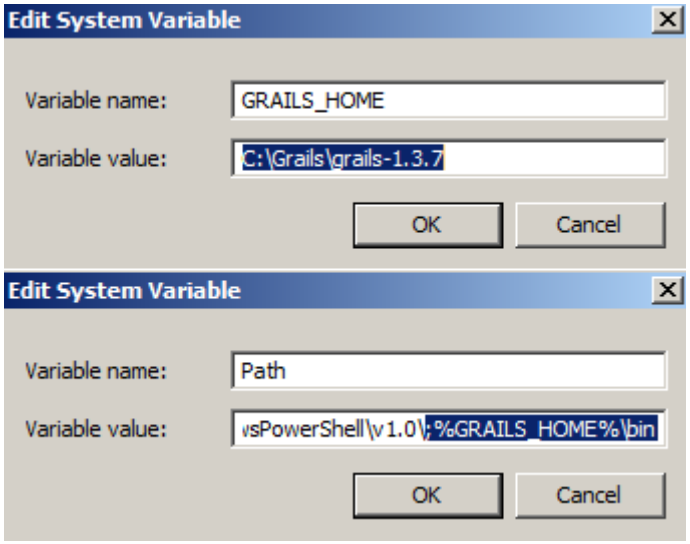
Tämä luku kertoo sen, miten Grails asennetaan Windows käyttöjärjestelmään. Grailsin asennus ei kuitenkaan poikkea kovinkaan paljon eri käyttöjärjestelmien välillä. Tarkemmin asennuksesta eri käyttöjärjestelmille on kerrottu Grailsin kotisivuilla. (Grails 2012.)

Ennen kuin Grailsia voi käyttää, täytyy järjestelmässä olla asennettuna Java Software Development Kit (SDK). Java SDK:lle täytyy myös asettaa JAVA_HOME-ympäristömuuttuja, jotta Grails pystyisi sitä käyttämään (kuva 14). Minimivaatimus käytettävälle Java SDK:lle riippuu siitä, mitä versiota Grailsista käytetään. (Grails 2012, linkit Documentation -> Installation.)



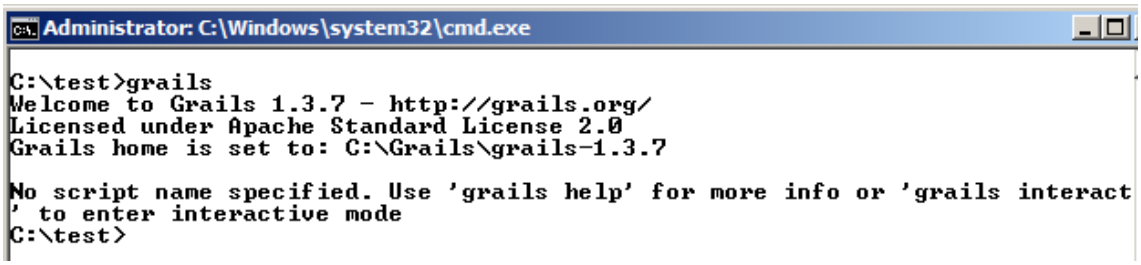
KUVA 14. JAVA_HOME-ympäristömuuttujat

Grails ladataan ja pakattu tiedosto puretaan haluttuun kansioon. Grailsille täytyy myös asettaa GRAILS_HOME-ympäristömuuttuja (kuva 15), joka osoittaa kansioon, johon tiedosto purettiin. Myös Path-ympäristömuuttujaan (kuva 15) joudutaan lisäämään viittaus bin-hakemistoon. (Grails.org. 2012, linkit Documentation -> Installation.)



KUVA 15. GRAILS_HOME- ja Path-ympäristömuuttujat

Komentoriville kirjoittetun komennon *grails* jälkeen pitäisi ilmestyä tervehdysviesti (kuva 16), joka kertoo onnistuneesta asennuksesta. Tämän jälkeen Grails-sovelluskehys on valmis käytettäväksi komentoriviltä ja NetBeans-kehitysympäristössä.



KUVA 16. Tervehdysviesti

4 RELAATIOTIETOKANNAT

Relaatiotietokantamalli on perustana relaatiotietokantaohjelmistoille kuten MySQL, Oracle ja Access. Relaatiotietokantamallin perustana toimivat matematiikan haarat, joukkoteoria ja ensimmäisen kertaluvun predikaattilogiikka. Relaatiotietokantamallia käyttäviä tietokantaohjelmistoja nimitetään yhteisesti relaatiotietokannoiksi. (Hernandez 2000, 12.)

Tietokantojen tarkoitus on toimia tietovarastona toteutettaville sovelluksille. Sinne on tarkoitus tallentaa kaikki sovelluksen käyttämä tieto, joka voidaan hakea myöhemmin sovelluksen käsiteltäväksi. Tietokantojen yhteydessä on myös muistettava datan ja informaation välinen ero. Tietokannan tarkoitus on sisältää kaikki tarvittava data, josta saadaan tuotettua mielekästä informaatiota. (Hernandez 2000, 3–17, 35.)

4.1 Tietokantojen tyypit

Tietokannat voidaan jakaa kahteen eri tyyppiin: käyttötietokantoihin ja analyyttisiin tietokantoihin. Käyttötietokannat sisältävät sellaista tietoa, jota kerätään, säilytetään ja muokataan paljon. Tämän tyyppiseen tietokantaan talletetaan dynaamista dataa, joka muuttuu jatkuvasti ja sen sisältämä data on aina ajan tasalla. Analyyttisellä tietokannalla tarkoitetaan sellaista tietokantaa, jota käytetään ajasta riippuvien ja historiallisten tietojen keräämiseen. Analyyttisen tietokannan sisältämä tieto on staattista, muuttumatonta dataa, jonka perusteella voidaan esimerkiksi luoda ennusteita tai seurata kehityksen suuntaa yrityksessä. (Hernandez 2000, 3–4.)

4.2 Relaatiotietokannan rakenne

Relaatiotietokanta koostuu relaatioista eli tauluista. Taulu on relaatiotietokannan perusrakenne, jonka tehtävänä on edustaa jotain konkreettista kohdetta kuten

esinettä, paikkaa tai henkilöä. Jokaisella taulun edustamalla kohteella on ominaisuuksia, joita voidaan tallentaa tauluun datana. (Hernandez 2000, 39.)

Tauluun tallennettava data muodostaa monikon eli tietueen. Tämä taulun sisäinen rakenne edustaa tiettyä uniikkia esiintymää jostain taulun edustamasta kohteesta. Tietue sisältää attribuutteja eli kenttiä. Kenttä on relaatiotietokannan pienin rakenne, joka edustaa jotain tiettyä kohteen ominaisuutta. Yksi näistä kentistä on tietueen pääavain, jonka avulla taulun sisältämä tietue tunnistetaan. Pääavain on siksi oltava ainutkertainen jokaisessa tietueessa. (Hernandez 2000, 42, 41, 44–45.)

Relaatiotietokantaan tallennetaan tieto myös siitä, mitkä tietokantataulut liittyvät toisiinsa, koska yhden tietueen ei ole tarkoitus sisältää kaikkea tarvittavaa tietoa. Tarkoitus on, että yksi reaalimaailman kohde ja siitä tallennettavat ominaisuudet ovat yhdessä taulussa. Näiden reaalimaailman kohteiden välillä voi olla jonkinlainen yhteys toisiinsa ja nämä yhteydet taas liittävät eri taulut toisiinsa. Relaatiotietokannassa taulut liitetään toisiinsa tiedolla, jotka muodostavat yhden samanlaisen kentän kummassakin taulussa. Tällä tavoin taulut eli relaatiot muodostavat yhteyden toistensa välille. (Hernandez 2000, 13, 39–41, 44–45.)

4.3 Yhteydet

Taulujen välille voidaan luoda yhteyksiä linkittämällä ne toisiinsa kolmannen taulun avulla tai yhteys voidaan muodostaa viiteavaimen ja pääavaimen avulla. Taulujen väliset yhteydet ovat aina tietyn tyyppisiä. Näitä mahdollisia yhteystyyppejä on kolme erilaista: yhdestä yhteen, yhdestä moneen ja monesta moneen. (Hernandez 2000, 46.)

Yhdestä yhteen

Taulujen välille muodostuu yhdestä yhteen -yhteys, kun jollain yksittäisen taulun tietueella on yhteys vain yhteen toisen taulun tietueeseen. Myös toisessa taulussa olevan tietueen tulee liittyä ainoastaan vain yhteen ja samaan ensimmäisessä taulussa olevaan tietueeseen. (Hernandez 2000, 47, 276.)

Yhdestä moneen

Yhdestä moneen -yhteydessä taulun yhdellä tietueella voi olla yhteys moneen eri tietueeseen toisessa taulussa tai taulussa on pelkästään yksi tietue, johon yhteys on muodostettu. Toisen taulun tietueen tulee kuitenkin muodostaa yhteys takaisin vain yhteen ja samaan tietueeseen, josta yhteys on muodostettu. (Hernandez 2000, 48, 277.)

Monesta moneen

Monesta moneen -tyyppinen yhteys muodostuu taulujen välille, kun yhdellä tietueella on yhteys yhteen tai useampaan toisen taulun tietueeseen. Monesta moneen -yhteydessä toisella taululla on myös yhteys useampaan tai pelkästään yhteen ensimmäisen taulun tietueeseen. Monesta moneen yhteys muodostetaan linkitystaulun avulla. Taulu sisältää kopiot yhteyden muodostavien tietueiden pääavaimista, joiden avulla tietueiden välinen yhteys osoitetaan. (Hernandez 2000, 49, 279, 293.)

4.4 Avaimet

Relaatiotietokannoissa käytetään kahdentyyppisiä avaimia: pääavaimia ja viiteavaimia. Ne ovat erikoiskenttiä, joiden tarkoitus taulussa on palvella jotain tiettyä tarkoitusta. Pääavaimen avulla tietue voidaan tunnistaa yksiselitteisesti. Viiteavainta käytetään kahden taulun välisen yhteyden muodostamisessa. (Hernandez 2000, 44.)

Pääavain

Pääavainta käytetään tietueen tunnistamiseen, joten sillä on oltava tiettyjä ominaisuuksia. Sillä on esimerkiksi pystyttävä määrittelemään tietue yksiselitteisesti. Pääavainkentän on silloin sisällettävä vain ainutlaatuisia arvoja, eikä sen sisältämä arvo voi olla koskaan tyhjä. (Hernandez 2000, 213–215.)

Viiteavain

Viiteavaimen tarkoituksena on muodostaa yhteys eri taulujen välille. Taulussa viiteavainkentän arvona käytetään toisen taulun pääavainta. Näin viiteavaimen avulla taulujen välinen yhteys saadaan muodostettua, kun viiteavaimessa ole-

valla arvolla viitataan toisen taulun tietueen pääavaimeen. (Hernandez 2000, 289–293.)

5 SCRUM-PROJEKTINHALLINTAMENETELMÄ

Scrum on projektinhallintamenetelmä, joka on yleisesti käytössä ketterässä ohjelmistokehityksessä. Scrumissa tuote kehitetään siihen haluttujen ominaisuuksien kirjosta muodostuvien tuotteen kehitysjonojen avulla. Näistä tuotteen omistajan priorisoimista kehitysjonoista tiimi muodostaa itselleen sprintin kehitysjonon, jonka tiimi sitoutuu toteuttamaan sprintin aikana. Sprintillä tarkoitetaan jotain tiettyä ajan määrää, esimerkiksi neljää viikkoa, jossa sprintin kehitysjono toteutetaan. Tämän jälkeen tiimi tekee itselleen uuden sprintin kehitysjonon ja aloittaa seuraavan sprintin. (Schwaber – Beedle 2001, 31–56.) Tässä osiossa käsitellään Scrumissa olevia rooleja ja niiden tuomia vastuita ja velvoitteita sekä käsitellään Scrumissa käytettäviä dokumentteja. Scrumiin liittyvää prosessia käsitellään tarkemmin luvussa 6 Scrum-prosessi.

Kirjassa Agile Software Development with Scrum kuvaillaan Scrumia vertaamalla sitä armeijan toimintaan: Konfliktin aikana armeija sijoittaa sotilaita tiettyihin pisteisiin toiminta-alueilla ja jokaiselle sotilaista muodostuvalle ryhmälle on määrätty jokin tietty tehtävä. Jokaisen ryhmän on itse organisoiduttava suoriutuakseen tehtävästään. Ryhmällä on kaikki heidän tarvitsemansa varusteet ja koulutus, joita he tarvitsevat tehtävän suorittamiseen. Koska ryhmän sijoituspaikka on keskellä kompleksista ja sekasortoista tilannetta, mistä ryhmällä ei ole ennestään mitään tietoa, siksi ryhmä ei voi etukäteen tietää myöskään sitä, mitä heidän tulee tehdä, jotta he saavuttavat tehtävänsä päämäärän. He voivat ainoastaan käyttää strategiaa apunaan päämäärän saavuttamiseksi. Ryhmän tarkoitus on improvisoida järjestelmällisesti saavuttaakseen päämääränsä. Ennalta määrätyn ajan päätyttyä tehtävä loppuu ja ryhmä haetaan pois. (Schwaber – Beedle 2001, 50.)

5.1 Scrum-mestari

Scrum-mestarin tehtävänä on huolehtia Scrumin onnistumisesta. Hän huolehtii siitä, että Scrumin arvoja, käytäntöjä sekä sääntöjä noudatetaan. Scrum-

mestarin tehtävänä on muodostaa johdon kanssa Scrum-tiimi. Tiimin muodostuksen jälkeen hänen tehtävänään on edustaa tiimiä johtokunnalle ja johtokuntaa tiimille. Scrum-mestari vertaa, mitä edistystä projektissa on tapahtunut suunnitellun sprintin tavoitteen sekä edellisen kokouksen ennusteisiin nähden. Hän myös yrittää mitata tiimin vauhtia, onko tiimi jumissa, takerteleeko kehitys vai edetäänkö projektissa. (Schwaber – Beedle 2001, 31–32.)

Scrum-mestari auttaa asiakastahoa yksilöimään heidän joukostaan henkilön, josta tulee tuotteen omistaja. Scrum-mestarin tehtävänä on muodostaa tiiminsä ja tuotteen omistajan kanssa tuotteen kehitysjono (product backlog). Scrum-mestari työskentelee tiiminsä kanssa, jotta sprintti saadaan aloitettua.

Sprintin aikana hän huolehtii kaikkien päivittäisten kokousten järjestämisestä ja johtamisesta. Päivittäisessä kokouksessa Scrum-mestarin tehtävänä on kuunnella tarkasti, mitä kukin tiiminjäsen raportoi hänelle. Scrum-mestari varmistaa myös sen, että kaikki projektiin vaikuttavat esteet poistetaan välittömästi. Hän huolehtii myös siitä, että kaikki projektia koskevat päätökset tehdään viipymättä. (Schwaber – Beedle 2001, 31–32.)

Scrum-mestarin tehtävänä on ylläpitää tiiminsä korkeinta mahdollista tuotantokykyä, ja siksi hänen työkuvaansa kuuluu pääasiassa päätösten tekeminen sekä projektia haittaavien esteiden poistaminen. Jos päivittäisessä kokouksessa on tehtävä päätöksiä, Scrum-mestari on vastuussa siitä, että päätökset tehdään välittömästi kokouksessa, vaikka asiasta ei olisi olemassa tarvittavaa tietoa oikean päätöksen tekemiseksi. Tällä tavoin toimittaessa tiimi voi jatkaa työskentelyä viipymättä. Jos tehty ratkaisu ei osoittaudu oikeaksi, se voidaan joka tapauksessa muuttaa myöhemmin toiseksi. Scrum-mestari pyrkii lisäksi poistamaan projektia haittaavat esteet välittömästi, mutta este voi olla sellainen, jota ei pysty poistamaan viipymättä. Silloin Scrum-mestari tekee ongelmasta näkyvän organisaatiolle. Tällainen ongelma voi koskea yrityksen menettelytapaa, politiikkaa, rakennetta tai tiloja, jotka vahingoittavat tiimin tuotantokykyä. (Schwaber – Beedle 2001, 31–32.)

5.2 Tuotteen omistaja

Ainoastaan tuotteen omistaja hallitsee ja kontrolloi tuotteen kehitysjonoa. Hänellä on myös virallinen vastuu projektista ja hänen tehtäviinsä kuuluu tuotteen kehitysjonon ylläpito sekä huolehtiminen siitä, että se on kaikkien saatavilla. Näin toimittaessa kaikkien tiedossa on myös korkeimmalla prioriteetilla olevat kohdat. (Schwaber – Beedle 2001, 35.)

Tuotteen omistajana toimii yleensä asiakkaan taholla työskentelevä henkilö, esimerkiksi tuotepäällikkö. Sisäisessä projektissa tuotteen omistajana voi toimia esimerkiksi projektipäällikkö. Tuotteen omistajana toimii aina vain yksi henkilö, eikä koskaan toimikunta. (Schwaber – Beedle 2001, 34.)

Toimikunta voi toimia tuotteenomistajan tukena ja neuvoa häntä, mutta ei koskaan saa suoranaisesti vaikuttaa tuotteen kehitysjonon prioriteetteihin. Tuotteen omistajan takana olevan toimikunnan on aina vakuutettava tuotteenomistaja, jos sen jäsenet haluavat muutoksia tuotteen vaatimuksiin tai sen prioriteetteihin. (Schwaber – Beedle 2001, 34.)

5.3 Tiimi

Scrum-työryhmän eli tiimin tehtävänä on sitoutua muuttamaan tuotteen kehitysjonosta valikoitu kokonaisuus toimivaksi tuotteeksi. Tähän sitoudutaan jokaisen sprintin alussa. Näin sitoudutaan myös saavuttamaan sprintin tavoite. Tiimillä on myös täysi valta tehdä päätöksiä saavuttaakseen päämääränsä. (Schwaber – Beedle 2001, 35–36.)

Tiimin tehokkuus perustuu sen dynamiikkaan. Tiimin jäsenillä voi olla hyvinkin erilaiset taustat. Heidän koulutuksensa voi olla hyvin erilainen ja siten myös työhistoriat voivat poiketa paljon toisistaan. Näin jäsenten erilaiset tiedot ja taidot lisäävät ryhmän tehokkuutta. Tiimiin tuleekin sisällyttää tarvittavilla taidoilla varustetut henkilöt, jotta sprintin tavoite saavutettaisiin. (Schwaber – Beedle 2001, 36, 37.)

Tiimi on itseorganisoituva ja näin ollen tiimin jäsenillä ei ole määriteltyä titteliä. Jokainen tiimin jäsen toimii omalla parhaiten osaamallaan alueella tehden myös osaamisalueensa ulkopuolisia tehtäviä. Tällä tavoin toimimalla tiimit ovat joustavia ja pystyvät käsittelemään mitä tahansa työtä, mitä projektissa eteneminen vaatii. Tiimin tehtävänä on myös valita tuotteen kehitysjonosta sellainen määrä työtä, joka on mahdollista tehdä seuraavan sprintin aikana. (Schwaber – Beedle 2001, 38, 37.)

Tiimin pitäisi koostua noin seitsemästä henkilöstä kahden hengen marginaalilla. Pienemmällä ryhmäkoolla voi olla etunsa, mutta siitä voi olla myös haittaa ryhmän sisäiseen vuorovaikutukseen ja näin ollen se voi vähentää tuotantokykyä. Ylisuuret tiimit eivät myöskään toimi Scrumissa, sillä ryhmän tuotteliaisuus laskee ja Scrumin hallintamekanismi muodostuu hankalaksi. (Schwaber – Beedle 2001, 36–37.)

Ylisuuret tiimit tulisi jakaa pienemmiksi tiimeiksi. Silloin, kun tiimissä on enemmän kuin kahdeksan henkeä, on suositeltavaa jakaa se kahteen pienempään tiimiin. Jokaisella suuremmasta kokonaisuudesta jaetulla tiimillä on oma Scrum-mestari, joka hoitaa tiiminsä kanssa päivittäiset kokoukset. Jokainen jaetun tiimin Scrum-mestari taas osallistuu alkuperäisen Scrum-mestarin järjestämään päivittäiseen kokoukseen. Tällä tavoin toimittaessa Scrum palvelee hyvin myös suurta tiimiä. (Schwaber – Beedle 2001, 37.)

5.4 Tuotteen kehitysjono

Tuotteen kehitysjono (product backlog) on kehittyvä priorisoitu jono teknillistä ja liiketoiminnallista toiminnallisuutta, josta kehitettävä järjestelmä muodostuu. Kaikki järjestelmän tarvitsemat vaatimukset on listattu tuotteen kehitysjonoon. Siihen voidaan myös listata tuotteen kannalta hyviä ideoita. Se on lista tuotteen kaikista ominaisuuksista, toiminnoista, parannuksista, teknologioista sekä virheistä, joita tuote sisältää. Kaikki, mikä ilmaisee jollain tavalla työtä, joka pitää suorittaa, kuuluu tuotteen kehitysjonoon. (Schwaber – Beedle 2001, 32–33.)

Tuotteen kehitysjono on aluksi puutteellinen alustava lista asioista, joita tuote tai järjestelmä vaatii. Ensimmäinen versio tuotteen kehitysjonosta voi olla pelkkä lista tuotteen vaatimuksista, jotka on kerätty jostain muusta dokumentista. Tuotteen kehitysjonon sisältämät tehtävät lajitellaan niiden prioriteetin mukaan. Mitä korkeammalla prioriteetilla tehtävä on, sitä kiireellisemmin se toteutetaan. Korkeampi prioriteetti määrittää myös sen, miten tarkkaa ja selkeää määrittely on. (Schwaber – Beedle 2001, 33.)

Tuotteen kehitysjonoon voidaan listata myös ongelmia, joita prosessissa edettäessä muodostuu. Tällaiset ongelmat vaativat yleensä ratkaisun ennen kuin yksi tai useampi tuotteen ominaisuuksista voidaan toteuttaa. Esimerkiksi jonkin käytetyn teknologian ominaisuus voi muodostaa tällaisen ongelman. Myös ongelmat priorisoidaan samalla tavalla, kuten tavalliset tehtävät priorisoitaisiin. (Schwaber – Beedle 2001, 33–34.)

5.5 Sprintin tehtävälista

Sprintin tehtävälista (sprint backlog) sisältää kaikki ne tehtävät, jotka tiimi aikoo toteuttaa kyseisen sprintin aikana. Tehtävälista muodostetaan tuotteen kehitysjonosta valituista tehtävistä sprintin suunnittelupalaverissa. Jokainen sprintin tehtävälistan sisältämä tehtävä jaetaan pienempiin tarkemmin määriteltyihin osiin. Näille osille annetaan arvio siitä, kuinka paljon työtä sen toteuttaminen vaatii. Kun tiimin jäsen työskentelee sprintin tehtävälistassa olevan tehtävän parissa, on jäsenen päivittäisenä tehtävänä arvioida jäljellä olevan työn määrää. Tämä arvio voi kasvaa, jos tehtävä osoittautuu luultua kompleksisemmaksi. Työn edetessä voidaan myös havaita joitain ennakoimattomia tehtäviä, jotka joudutaan toteuttamaan kuluvan sprintin puitteissa. Tiimi on silloin velvollinen lisäämään nämä tehtävät sprintin tehtävälistaan ja tekemään työmääräarviot tehtäville. (Schwaber – Beedle 2001, 71.)

5.6 Scrum-palaveri

Scrum-palaverissa (daily scrum), joka on yksi Scrumin käytäntöjä, on tarkoitus vastata kolmeen Scrum-mestarin esittämään kysymykseen:

1) Mitä olet tehnyt edellisen päivän aikana?
2) Mitä olet ajatellut tehdä seuraavan päivän aikana?
3) Mitkä tekijät estävät tai hidastavat sinua tekemisissäsi?

Scrum-palaverin kesto on noin 15 minuuttia tai vähemmän. Scrum-palaverissa ei ole tarkoitus raportoida johdolle, vaan tarkoitus on organisoida, optimoida ja koordinoida sprintin etenemistä. Tiimi myös jakaa toisilleen tietoa tekemisistään ja siitä, mitä he aikovat tehdä seuraavaksi. (Ketterät käytännöt.fi. 2008.)

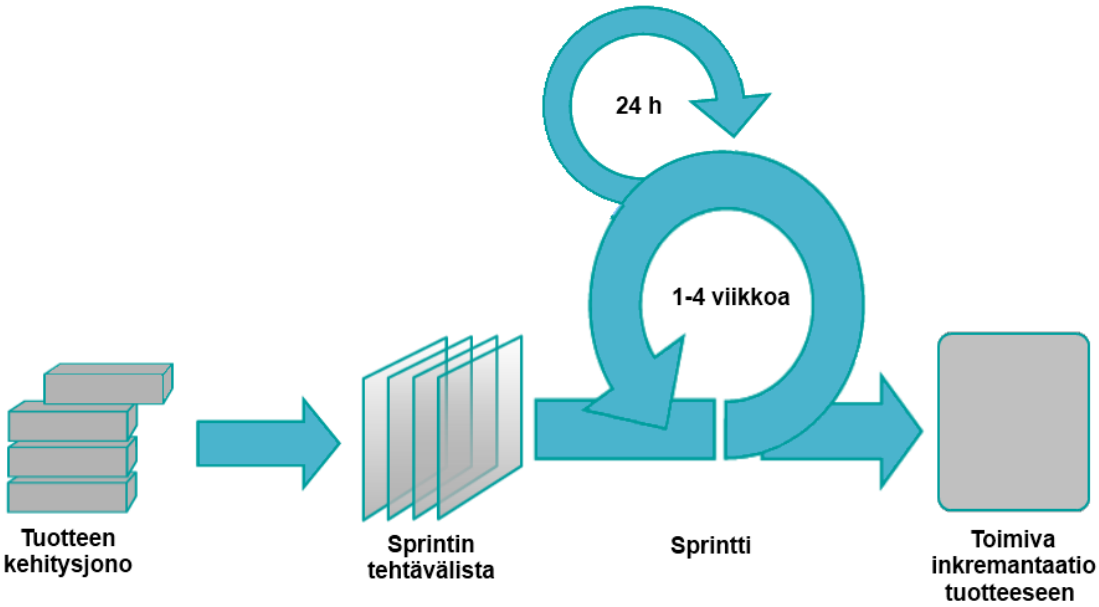
Scrum-palaveriin osallistuvat aina kaikki tiimin jäsenet ja Scrum-mestari. Se on myös avoin kaikille projektista kiinnostuneille, mutta vain tiimin jäsenillä on oikeus puhua kokouksessa (Ketterät käytännöt.fi. 2008).

Scrum-palaverissa ei ole tarkoitus keskustella, vaan vastataan ainoastaan kolmeen Scrum-mestarin esittämään kysymykseen. Jos jokin asia on sellainen tai ilmenee, että jokin asia kaipaa tarkempaa pureutumista ja keskustelua, tämä tehdään heti kokouksen jälkeen erillisessä palaverissa. (Sutherland 2010, 24.)

6 SCRUM-PROSESSI

Scrumissa työskentely tapahtuu tiimissä, jonka tarkoitus on valmistaa jokin toimiva tuote. Työskentely tapahtuu toistamalla samoja työvaiheita eli iteroimalla. Jokaisen iteraation alussa muodostetaan sprintin kehitysjono, jonka tarkoituksena on selvittää se, mitä tässä iteraatiossa tulee tehdä. (Schwaber – Beedle 2001, 31–56; Sutherland 2010, 5–34.)

Iteraation tarkoitus on inkrementaalisesti, eli koko ajan lopullista muotoaan kohti kasvaen, lisätä jotain tuotteeseen. Jokaisen iteraation tuottaman inkrementoinnin lopputuloksena pitäisi olla toimiva tuote, joka on jollain tapaa uudistunut, muuttunut tai kasvanut. (Schwaber – Beedle 2001, 31–56; Sutherland 2010, 5–34.) Tämä Scrum-prosessi on kuvattu kuvassa 17. Tässä luvussa käsitellään esimerkin avulla sitä, miten Scrum-prosessi etenee.



KUVA17. Scrum-prosessi

6.1 Tuotteen kehitysjonon muodostaminen

Ensimmäinen askel Scrumissa on muodostaa tuotteen kehitysjono (product backlog). Se muodostetaan jostain ideasta tai innovaatiosta, josta tuotteen omistaja muodostaa priorisoidun listan tuotteen vaatimuksista. (Schwaber – Beedle 2001, 18.)

Kehitysjonon priorisointi tapahtuu siten, että kehitysryhmä arvioi sitä, kuinka hankalaa mikäkin tuotteelle asetettu vaatimus on toteuttaa. Tuotteen omistaja on taas vastuussa arvion tekemisestä sille, kuinka paljon liiketoiminnallista arvoa milläkin vaatimuksella on yritykselle, jota hän edustaa. Pääasiassa näiden kahden arvion perusteella tuotteen omistaja priorisoi tuotteen kehitysjonon. Näiden arvioiden lisäksi voidaan vielä käyttää apuna arviota riskeistä. Tuotteen omistajan tarkoituksena on kuitenkin priorisoida kehitysjono maksimoiden investoinnin tuottoprosentti. (Schwaber – Beedle 2001, 19.)

Kehitysjonon muodostamiseen voidaan käyttää myös käyttäjätarinoita. Käyttäjätarinat ovat ketterissä käytännöissä käytetty tapa, jolla tuotteen vaatimusmäärittely toteutetaan. Ne eivät kuitenkaan sisällä tarkkaa määrittelyä, kuten perinteisissä menetelmissä on tapana, vaan ovat ennemminkin toiveita siitä, millainen tuotteen tulisi olla. Tällainen käytäntö tukeekin sitä tosiasiaa, että asiakas tai käyttäjä ei välttämättä tiedä tarkalleen, mitä haluaa, vaan hänen on ensin nähtävä jotain sinnepäin olevaa, ennen kuin hän voi määritellä tarkemmin sen, mitä hän haluaa. Näin tuotteen vaatimukset tarkentuvat ja muotoutuvat projektin edetessä. Käyttäjätarinoissa arvioidaan myös niiden toteutuksen hankaluutta ja liiketoiminnallista arvoa. (Ketterät käytännöt.fi 2008.)

Hypoteettiseksi esimerkiksi voidaan ottaa esimerkiksi jokin verkkokauppa, jolle tuotteenomistaja asettaa vaatimuksiksi seuraavaa:

1) Asiakkaana haluan laittaa tuotteita ostoskoriin.
2) Asiakkaana haluan poistaa tuotteita ostoskorista.
3) Asiakkaana haluan maksaa ostoskorin verkkopankissa.
4) Asiakkaana haluan maksaa ostoskorin luottokortilla.
5) Asiakkaana haluan aina nähdä tuotteet jotka ovat ostoskorissa.
6) Asiakkaana haluan antaa palautetta kaupasta.

7) Kaikki palvelimet tulee päivittää uusimpaan versioon.

Näistä vaatimuksista tai käyttäjätarinoista voidaan muodostaa tuotteen kehitysjono (kuva 18), josta tuotteen omistaja arvioi tuotteen ominaisuuden tai tehtävän liiketaloudellista arvoa. Arviointia tehdessä ei ole tapana käyttää mitään absoluuttisia yksiköitä, vaan ennemmin käytetään verrannollista pisteytystä, jossa kehitysjonossa olevia asioita verrataan keskenään. (Sutherland 2010, 19.)

Item (Tehtävä)	Details (Yksityiskohdat)	Priority (Prioriteetti)	Estimate of Value (Arvio Arvosta)	Inital Estimate of Effort (Arvio Hankaluudesta)
Asiakkaana haluan laittaa tuotteita ostoskoriin			7	
Asiakkaana haluan poistaa tuotteita ostoskorista			6	
Asiakkaana haluan maksaa ostoskorin verkkopankissa			6	
Asiakkaanaa haluan maksaa ostoskorin luottokortilla			6	
Asiakkaana haluan aina nähdä tuotteet jotka ovat ostoskorissa			4	
Asiakkaana haluan antaa palautetta kaupasta			5	
Kaikki palvelimet tulee päivittää uusimpaan versioon			3	
Asiakkaana haluan…			2	

KUVA 18. Tuotteen kehitysjono, jossa liiketoiminnallisen arvon arviointi

Seuraavaksi tiimi vuorostaan arvioi sitä, kuinka paljon työtä vaaditaan työjonossa olevien tehtävien tai ominaisuuksien toteuttamiseksi. Tarkoitus on, että tiimi vertaa työjonon tehtäviä ja ominaisuuksia keskenään ja pisteyttää tehtävä-jonossa olevat asiat hankalimmasta helpoimpaan (kuva 19). Tämä on yleensä helpoin tapa arvioida työstä aiheutuvaa kuormaa. (Versionone.com. 2011.)

				Total Amount of Points 50	Points left 0
Employee (Työntekijä)	Points (Pisteet)	Item (Tehtävä)	Estimate of Value (Arvio Arvosta)	Inital Estimate of Effort (Arvio Työn Määrästä)	
John	10	Asiakkaana haluan laittaa tuotteita ostoskoriin	7	6	
Jane	10	Asiakkaana haluan poistaa tuotteita ostoskorista	6	5	
Alice	10	Asiakkaana haluan maksaa ostoskorin verkkopankissa	6	4	
Jack	10	Asiakkaana haluan maksaa ostoskorin luottokortilla	6	4	
Mike	10	Asiakkaana haluan aina nähdä tuotteet, jotka ovat ostoskorissa	4	3	
		Asiakkaana haluan antaa palautetta kaupasta	5	7	
		Kaikki palvelimet tulee päivittää uusimpaan versioon	3	5	
		Asiakkaana haluan...	2	6	
		Asiakkaana haluan...	3	7	
		Asiakkaana haluan...	2	7	

KUVA 19. Työmäärän arviot

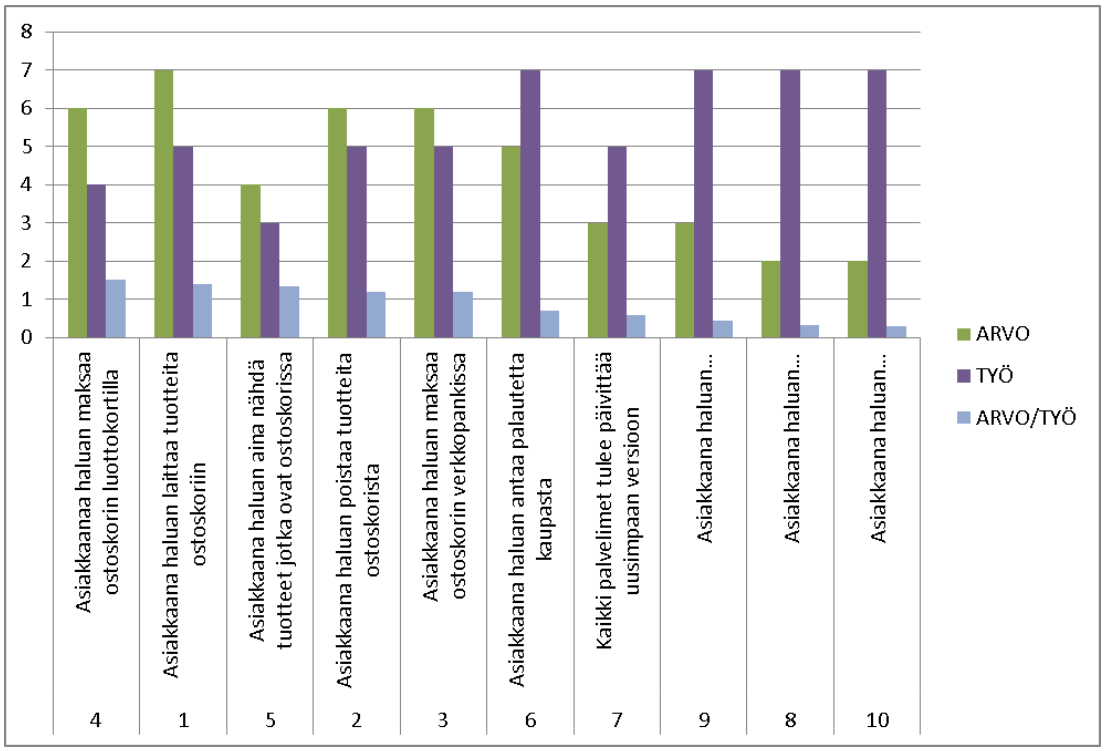
Tässä esimerkissä oletetaan tiimin koostuvan viidestä henkilöstä, joilla jokaisella on käytettävänään kymmenen pistettä (kuva 20). Tiimin tehtävänä on jakaa kaikki nämä pisteet tuotteen kehitysjonossa oleville tehtäville. Jakaminen tapahtuu tiimin arvion perusteella. Tiimi arvioi sitä, kuinka vaikea mikäkin tuotteen kehitysjonossa eritelty tehtävä on heidän mielestään toteuttaa. (Sutherland 2010, 19.)

Employee (Työntekijä)	Points (Pisteet)
John	10
Jane	10
Alice	10
Jack	10
Mike	10

KUVA 20. Työn kuorma henkilöä kohti

Arviointitavassa, jossa käytetään työn yksiköihin pisteyttämistä, ei yritetä suhteuttaa niitä mitenkään aikaan, vaan pelkästään siihen, miten vaikeita kehitysjonossa olevat asiat ovat toteuttavan kehitysryhmän toteuttaa toisiinsa verrattuna. Vertailussa mittarina käytetään varsinaisen ohjelmoinnin aiheuttamaa työtaakkaa, joka merkataan työnyksikköinä. Työnyksikkö sisältää ainoastaan tähän käytettävää ohjelmointityötä, eikä se saa sisältää mitään muuta. Muutaman iteraation jälkeen näitä yksiköitä voidaan ryhtyä suhteuttamaan aikaan, jolloin muodostetaan tiimin nopeus. Nämä työnyksiköt muodostavat työn määrän, minkä tiimi pystyy jossain ajan suureessa suorittamaan. Työnyksiköiden kanssa toimiessa ei pitäisi kiirehtiä suhteuttamaan niitä ajan suureisiin. (Versionone.com 2011.)

Seuraavassa vaiheessa priorisoidaan tuotteen kehitysjonon sisältö. Tämän priorisoinnin suorittaa tuotteen omistaja. Tuotteen omistaja valitsee näistä korkeimmalle prioriteetille ne kohdat, joiden arvo on korkea ja työn määrä mahdollisimman alhainen. Tällä tavoin sijoitetun pääoman tuottoprosentti saadaan mahdollisimman korkeaksi. (Sutherland 2010, 19.) Kaikkia ei tietenkään voida aivan näin suoraan priorisoida. Esimerkiksi ostoskorin maksamista ei voi toteuttaa ennen kuin ostoskori on toteutettu. (Kuvat 21–22.)



KUVA 21. Järjestys arvon ja työn suhteen

Item (Tehtävä)	Details (Yksityiskohdat)	Priority (Prioriteetti)	Estimate of Value (Arvio Arvosta)	Inital Estimate of Effort (Arvio Työn Määrästä)
Asiakkaana haluan laittaa tuotteita ostoskoriin		1	7	5
Asiakkaanaa haluan maksaa ostoskorin luottokortilla		2	6	4
Asiakkaana haluan aina nähdä tuotteet jotka ovat ostoskorissa		3	4	3
Asiakkaana haluan poistaa tuotteita ostoskorista		4	6	5
Asiakkaana haluan maksaa ostoskorin verkkopankissa		5	6	5
Asiakkaana haluan antaa palautetta kaupasta		6	5	7
Kaikki palvelimet tulee päivittää uusimpaan versioon		7	3	5
Asiakkaana haluan…		8	3	7
Asiakkaana haluan…		9	2	7
Asiakkaana haluan…		10	2	7

KUVA 22. Tuotteen kehitysjono

6.2 Sprintin suunnittelu

Jokaisen sprintin alussa järjestetään sprintin suunnittelukokous, joka jakaantuu kahteen eri kokoukseen. Ensimmäisessä kokouksessa tuotteen omistaja ja tiimi tarkastelevat korkeimman prioriteetin saaneita kohtia, jotka tuotteen omistaja

haluaa toteuttaa sprintissä. He keskustelevat sprintin tavoitteesta ja toteutettavien kohtien taustasta valaisten tiimille tuotteen omistajan ajatuksia ja tarkoitusperiä, joita kohtien takana on. He myös määrittelevät ”tehdyn” merkityksen tässä osassa kokousta, koska tehdyllä voi olla eroja riippuen siitä, mistä näkökulmasta asiaa katsotaan. Tehdyllä voidaan tarkoittaa sitä, että kaikki mahdollinen dokumentointi, testaus, integrointi sekä muu kohtaan liittyvä on toteutettu, tai se sisältää pelkästään kohdan toteutuksen. Tämä takia on tärkeää määritellä, mitä sanalla ”tehty” tarkoitetaan. Kokouksen tämän osion on tarkoitus luoda ymmärtämys tiimille siitä, mitä tuotteen omistaja haluaa. (Sutherland 2010, 20.)

Toisessa osiossa keskitytään jokaisen kohdan yksityiskohtaiseen suunnitteluun. Tarkoitus on jakaa jokainen toteutettavaksi valittu kohta pienempiin osiin ja näin suunnitella valittujen kohtien toteutuksen vaatimat tehtävät. Tiimi valitsee siis korkeimmalla prioriteetilla olevat kohdat tuotteen kehitysjonosta ja sitoutuu toteuttamaan ne seuraavan sprintin aikana. Tuotteenomistaja ei kuitenkaan tiedä, kuinka moneen kohtaan tiimi sitoutuu, mutta hän tietää kuitenkin sen, että tiimi valitsee työn alle korkeimmalla prioriteetilla olevia kohtia tuotteen kehitysjonosta. Tiimillä on kuitenkin valta valita myös matalammalla prioriteetilla olevia kohtia tuotteen kehitysjonosta. Näin yleensä tapahtuu silloin, kun huomataan jonkin tällaisen kohdan sopivan yhteen korkeamman prioriteetin tehtävän kanssa. Tiimin täytyy kuitenkin aina neuvotella asiasta tuotteen omistajan kanssa. (Sutherland 2010, 20.)

Esimerkissä ensimmäiseen 14 päivän sprinttiin valitaan toteutettavaksi ostoskori ja sen maksaminen. Nämä toteutettavat ominaisuudet jaetaan pienempiin osiin, joille arvioidaan, minkä verran työpäiviä mikäkin osa vaatii (kuva 24). Tiimin nopeus arvioidaan työpäivissä, jotka heillä on käytössään Sprintin aikana. Sprintin aikaiseksi nopeudeksi saadaan 49 työpäivää, jotka jaetaan sprintin sisältämien tehtävien kesken. (Kuva 23.)

Daily working hours: 7,5		Hours in Sprint: 367,5		Days in Sprint: 49	
Employee (Työntekijä)	**Days (Päivät)**	**Vacations (Lomat)**	**Working Days (Työpäivät)**	**Working Hours (Työtunnit)**	
John	14	4	10	75	
Jane	14	5	9	67,5	
Alice	14	4	10	75	
Jack	14	4	10	75	
Mike	14	4	10	75	

KUVA 23. Tiimin nopeuden määrittäminen

Jokainen tiimin jäsen valitsee haluamansa tehtävän tai tehtävät. Jotkin tehtävät voidaan katsoa sen verran suuriksi, että niiden toteutus voidaan päättää toteuttaa pareittain. (Kuva 24.)

Spirnt Lenght in days: 14		0	49		
ITEM#	**Tekijä(t)**	**Effort**	**Not Started (Aloittamatta)**	**On Going (Tekeillä)**	**Done (Valmis)**
Asiakkaana haluan laittaa tuotteita ostoskoriin					
Luo ostokorille yhteys tietokantaan (Grails domain)	John	6	X		
Luo näkymä ostoskorille (Grails view)	Mike/Jane	16	X		
Luo toiminta logiikka (Grails Controller)	Jack/Alice	20	X		
			X		
			X		
Asiakkaanaa haluan maksaa ostoskorin luottokortilla					
Luo toiminnallisuus Controlleriin	John	6	X		
Luo ostokri näkymään painike maksamiselle	Jane	1	X		
.....					
.....					
.....					

KUVA 24. Sprintin tehtävälista

6.3 Sprintti

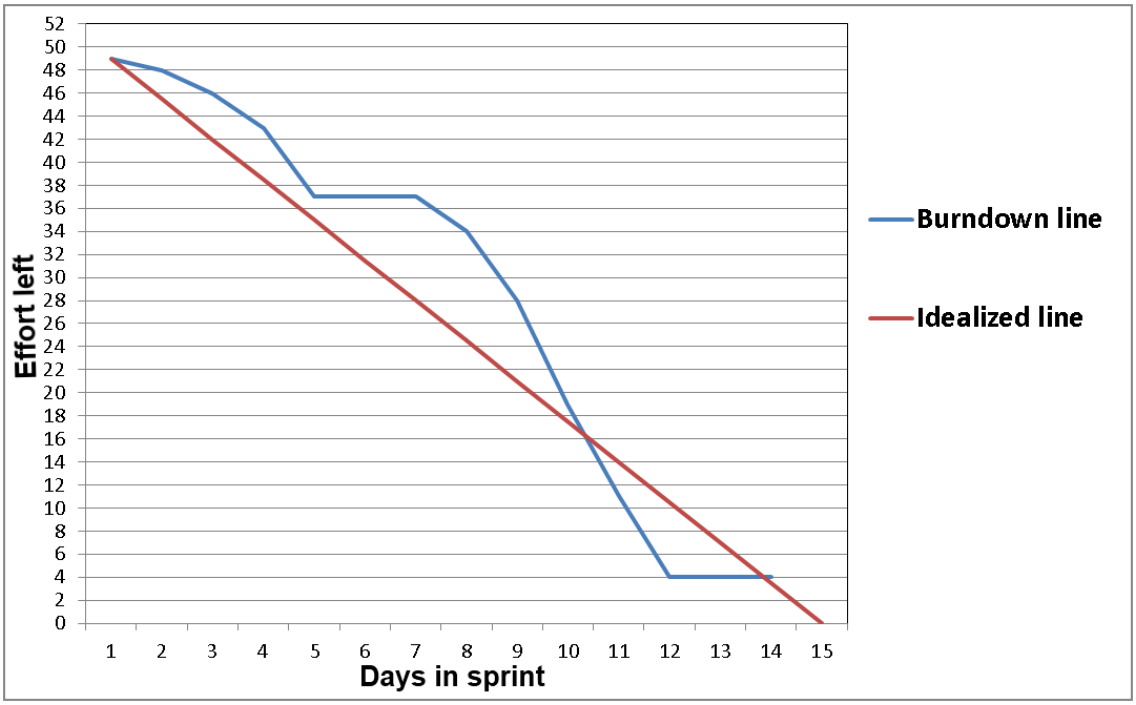
Tiimi työskentelee kiinteän ennalta sovitun ajan, jota kutsutaan sprintiksi eli pyrähdykseksi. Tiimin on tarkoitus tehdä sprintti, jonka tarkoituksena on saavuttaa sprintin tavoite. (Schwaber – Beedle 2001, 50.) Esimerkissä sprintin tavoitteeksi asetetaan kahden korkeimman prioriteetin tehtävän suorittaminen, joista muodostettiin sprintin tehtävälista (kuva 21). Sprintti voi sisältää suunnittelua, kokouksia ja konsultointia sekä kaikkea, mitä tiimi katsoo tarvitsevansa päämäärän-

sä saavuttamiseksi. Tiimillä on täysi valta valita, mitä sprintti sisältää, koska yritys antaa sille sprintin ajaksi vapauden tähän. Tiimillä on ainoastaan kaksi vastuualuetta sprintin aikana: päivittäinen Scrum-kokous ja sprintin tehtävälista. (Schwaber – Beedle 2001, 52–53.)

Tiimi seuraa päivittäin sprintin etenemistä ja myös arvioi, kuinka paljon tehtävää työtä on vielä jäljellä (kuva 25). Työmäärää arvioidaan samoilla sovituilla yksiköillä. Näistä arvioista muodostetaan sprintin edistymiskäyrä (sprint burndown chart) (kuva 26). Tällä käyrällä mitataan jäljellä olevan työn määrää. Työmäärän on tarkoitus laskea kohti nollaa sitä mukaa, kun edetään kohti sprintin viimeistä päivää. Käyrän tarkoituksena on mitata tiimin edistymistä sprintin tavoitteeseen pääsemiseksi. (Sutherland 2010, 26.)

Spirnt Lenght in days: 14		0	49			49	48	46	43	37
ITEM#	Tekijä(t)	Effort	Not Started (Aloittamatta)	On Going (Tekeillä)	Done (Valmis)	14	13	12	11	10
Asiakkaana haluan laittaa tuotteita ostoskoriin						New Estimates of				
Luo ostokorille yhteys tietokantaan (Grails domain)	John	6		X		6	5	4	3	2
Luo näkymä ostoskorille (Grails view)	Mike/Jane	16		X		16	14	14	13	10
Luo toiminta logiikka (Grails Controller)	Jack/Alice	20		X		20	22	21	20	18
Asiakkaanaa haluan maksaa ostoskorin luottokortilla										
Luo toiminnallisuus Controlleriin	John	6	X			6	6	6	6	6
Luo ostokri näkymään painike maksamiselle	Jane	1	X			1	1	1	1	1

KUVA 25. Esimerkkiprojektin sprintin kehitysjono, jossa työmäärän tehtäväkohtaiset sekä kokonaismääräiset arviot viikon ajalta. Yllä punaisella ympyröity kokonaismääräinen ja alla tehtäväkohtainen työmäärä päivissä.



KUVA 26. Esimerkkiprojektin ensimmäisen sprintin edistymiskäyrä

6.4 Sprintin katselmointi

Sprintin päätyttyä tiimi esittelee sen, mitä he saivat aikaiseksi sprintin aikana. Tässä katselmoinnissa ovat mukana tuotteen omistaja sekä kaikki projektista kiinnostuneet henkilöt. Tämä tilaisuus tarjoaa kaikille sen kohdan projektista, missä he voivat tutkia ja mukauttaa sprintissä tapahtuneita asioita. Yrityksen johtoa voi esimerkiksi kiinnostaa se, mitä tiimille annetuilla resursseilla pystyttiin toteuttamaan, tai asiakkaat ovat kiinnostuneita siitä, pitävätkö he tiimin aikaansaannoksesta. (Schwaber – Beedle 2001, 55.) Pääasiallinen tarkoitus tällä on kuitenkin perusteellinen keskustelu ja yhteistyö tiimin sekä tuotteenomistajan välillä. Tilaisuudessa tarkastellaan retroperspektiivisesti, mikä toimi ja mikä ei toiminut kuluneessa sprintissä. Tällä tavoin tulevat sprintit voidaan mukauttaa toimivammiksi. (Sutherland 2010, 28.)

7 TIETOJÄRJESTELMÄN TOTEUTUS

Järjestelmä toteutettiin Grails-sovelluskehyksellä, joka on tarkoitettu web-sovellusten kehittämiseen. Grailsissa ohjelmoinitkielenä käytetään Groovya, jonka juuret on Javassa. Järjestelmän ohjelmointityö tehtiin NetBeans-kehitysympäristössä, jota voidaan käyttää Grails-sovellusten kehittämiseen. Tässä luvussa kuvataan ja esitellään toteutettu järjestelmä sekä kerrotaan, miten Scrumia käytettiin tässä projektissa. Luvussa esitellään myös järjestelmään toteutetut käyttäjärajapinnat ja niiden toimintalogiikka.

7.1 Kuvaus ja esittely

Toteutetun sovelluksen tarkoituksena on toimia työntekijöiden mökkien WWW-pohjaisena arvontajärjestelmänä. Arvontajärjestelmään voidaan jättää arpa, johon voidaan merkitä kolme aikaväliä. Näillä aikaväleillä arvan jättänyt henkilö osoittaa, mille aikavälille hän haluaisi mökin varauksen muodostuvan. Henkilö voi ilmoittaa aikavälien yhteydessä myös sen, voiko aikaväli olla lyhyempi. Arpaan voidaan kirjoittaa myös viesti. Viestissä henkilö voi kertoa, miksi juuri hänen pitäisi saada varaus haluamalleen aikavälille. Jokainen arvonta muodostuu arvontajärjestelmällä luodusta kaudesta, johon arvalla voidaan osallistua. Kausi sisältää aikavälin siitä, milloin arvontaan voi osallistua sekä millä aikavälillä mökit ovat varattavissa.

Järjestelmä toimii myös mökkien varauksia käsittelevänä sovelluksena. Mökeille voidaan tehdä varauksia ja varauksia voidaan muuttaa sekä poistaa. Järjestelmä muuttaa arvontakauteen osallistuneen henkilön arvan voittaneeksi, kun tälle henkilölle tehdään varaus johonkin kauden mökeistä. Arpa poistuu silloin listalta, jossa ovat kaikkien arvontaan osallistuneiden arvat, joita ei vielä ole käsitelty.

Järjestelmään toteutettiin kaksi käyttäjärajapintaa: toinen arvontaan osallistujille ja toinen arvonnan valvojalle, joka myös hallinnoi järjestelmää. Osallistujien rajapinta on rajattu käsittämään vain näkymä, josta arpa jätetään. Valvojan raja-

pinnasta voidaan hallinnoida käyttäjien oikeuksia ja selata arvontaan osallistuneita arpoja sekä hallinnoida varauksia.

Järjestelmäarkkitehtuuriltaan järjestelmä koostuu tietokannasta, Mökkiarpa-sovelluksesta ja WWW-palvelimesta. Tietokanta sisältää kaiken sovellukselta tulevan tiedon, jota tarvitaan arvonnassa ja mökkien varauksissa. Sovelluksen tarkoitus on helpottaa ja automatisoida arvontaa sekä arvoista muodostettavien mökkivarausten luomista.

7.2 Scrum projektissa

Projektin hallinnassa käytettiin Scrumia, jota mukautettiin tähän projektiin sopivaksi. Kehitystyön etenemiseksi arvontajärjestelmästä laadittiin käyttäjätarinoiden pohjalta priorisoitu tuotteen kehitysjono. Tuotteen kehitysjonosta muodostettiin sprintin tehtävälistat jokaiselle sprintille. Tällä tavoin hallittiin tapahtuvaa kehitystyötä.

Muut Scrumin käytännöt kuten Scrum-palaverit katsoin tarpeettomiksi siksi, että työskentelin pääasiassa yksin kehitettävän järjestelmän parissa. Scrumista saatiin kuitenkin erinomainen pohja kehitystyön prosessille, jonka dokumentteihin nojautuen arvontajärjestelmä voitiin toteuttaa.

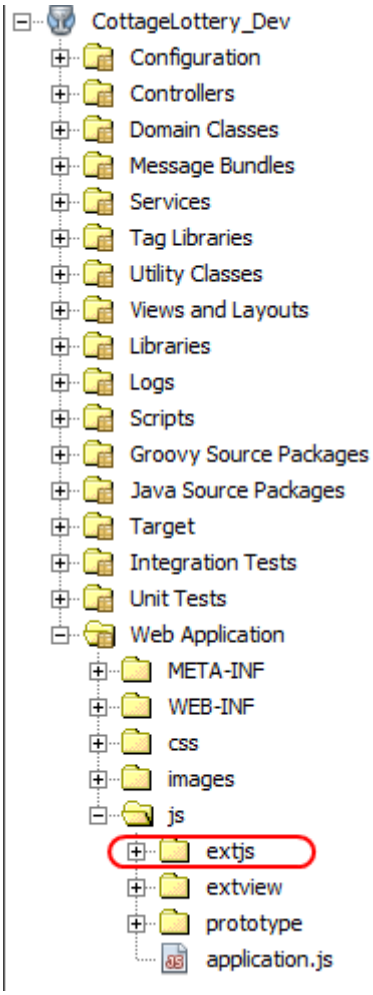
7.3 Työkalut ja kirjastot

Järjestelmä toteutettiin NetBeans-ohjelmointiympäristössä, johon Grails integroituu. NetBeans valittiin ohjelmointiympäristöksi, koska se oli ennestään tuttu. Integroinnin jälkeen NetBeans pystyy luomaan Grails-projektin ja sen läpi voidaan suorittaa Grailsissa käytettäviä komentoja (liite 4).

ExtJS on JavaScript-ohjelmistokehys, jolla luodaan erilaisia widgettejä eli käyttöliittymäkontrolleja. Se tarjoaa myös erilaisia metodeja näiden hallintaan. ExtJS:llä voidaan luoda ja hallita erilaisia ikkunoita, paneeleja sekä syöttökenttiä. ExtJS on lisensioitu kahdella eri lisenssillä: avoimen lähdekoodin lisenssillä sekä kaupallisella lisenssillä. ExtJS:n avoimen lähdekoodin lisenssillä toteutet-

tujen sovellusten lähdekoodi on annettava sovelluksen käyttäjille GNU GPLv3-lisenssiehtojen mukaan. (Sencha.com. 2012.)

ExtJS saadaan käyttöön lataamalla se osoitteesta http://www.sencha.com/products/extjs3/download/ext-js-3.4.0/203. Osoitteesta ladattu paketti puretaan ja siirretään Grails sovelluksen tiedostopuussa Web Application -kansiossa sijaitsevaan js-kansioon (kuva 27).



KUVA 27. Sovelluksen tiedostopuu.

ExtJS saadaan käyttöön GSP-sivuilla tekemällä linkit tarvittaviin resursseihin (kuva 28). Tämän jälkeen ExtJS:n widgetit ovat käytössä. Näitä widget-komponentteja voidaan sekoittaa GSP-sivuille tai JavaScriptillä voidaan tehdä täysin ExtJS:än pohjautuvia näkymiä.

```
<!-- Ext relies on its default css so include it here. -->
<!-- This must come BEFORE javascript includes! -->
<link rel="stylesheet"
      type="text/css"
      href="${resource(dir:'js/extjs/resources/css/ext-all.css')}"  />
<!-- Include here your own css files if you have them. -->

<!-- First of javascript includes must be an adapter... -->
<g:javascript library="prototype" />
<script type="text/javascript"
        src="${createLinkTo(dir:'js/extjs',file:'adapter/prototype/ext-prototype-adapter-debug.js')}">
</script>
<g:javascript src="extjs/adapter/ext/ext-base.js"/>

<!-- ...then you need the Ext itself, either debug or production version. -->
<!--<script type="text/javascript" src="extjs/ext-all-debug.js"></script> -->
<g:javascript src="extjs/ext-all-debug.js"/>
```

KUVA 28. GSP-sivulle sisällytettävät ExtJS-linkitykset

7.4 Vaatimusmäärittely

Toteutusprosessi aloitettiin määrittelemällä toteutettavan järjestelmän vaatimukset. Vaatimusmäärittely toteutettiin Scrumin tapaan muodostamalla lista, johon kerättiin käyttäjätarinoita. Käyttäjätarinoihin ideoitiin kaikkia mahdollisia järjestelmän ominaisuuksia, joista muodostettiin tietojärjestelmän vaatimukset (kuva 29).

Item#	UserStory	Prio#
1	Arpojan eli adminin autentikointi sisäänkirjautuessa	1
2	Jokainen voi jättää arvan arvonta kaudelle, eli osallistujalle ei autentikointia	1
3	Arvassa olatava yksi - kolme ajanjaksoa	1
4	Vain yksi arpa henkilöä kohden per arvonta kausi	1
5	Arvassa kenttä "lyhyempi käy" joka tarkoittaa, että arpoja voi lyhentää kolmea ajanjaksoa	1
6	Näkymä arvalle, josta arpalipuke jätetään	1
7	Näkymä arpojalle	1
8	Arpojan näkymästä näytetään arvan jättäneet	1
9	Arpojan näkymässä kalenteri johon arvottavat ajanjaksot sijoitetaan	2
10	Järjestelmä sijoittaa ei päällekkäiset ajanjaksot varauksiksi	2
11	Järjestelmä arpoo automaattisesti voittoarvat	3
12	Näkymä josta valitaan joko jätä arpa tai järjestelmän kirjautuminen	1
13	Järjestelmään luodaan tili arvontaan osallistuttaessa	3
14	Järjestelmä lähettää Tunnarit s-postiin arvan jätön jälkeen	3
15	Järjestelmä ilmoittaa s-postin välityksellä voittiko arpa	2
16	Arvontaan voi osallistua vain enen määräajan päättymistä	1
17	Arvassa selitekenttä jolla voidaan koittaa vaikuttaa arpojaan	1
18	Varauskalenteri johon voi lisätä myös arvonnan ulkopuolisia varauksia	1
19	Active Directorysta "AD" nimi lyhenteellä sähköpostiosoitteen kysely	2
20	Nimi lyhenteellä voittojen kysely järjestelmästä	3
21	Syötetyn arvan yhteydessä aiempien arpojen voittojen näyttäminen	2

KUVA 29. Arvontajärjestelmän käyttäjätarinat

Seuraavassa vaiheessa käyttäjätarinoista muodostettiin tuotteen kehitysjono, joka priorisoitiin kolmeen eri ryhmään. Tällöin tuotteelle muodostui kolme kehitysjonoa. Priorisointi tapahtui järjestämällä kaikki käyttäjätarinoihin kasatutut asiat ja ideat siten, että ensimmäisen kehitysjonon toteuttamisen jälkeen sovellusta voitaisiin käyttää ja testata. Seuraavaksi jäljelle jääneistä muodostettiin kaksi kehitysjonoa, joiden on tarkoitus automatisoida ja helpottaa arpojan toimintaa. (Kuva 30.)

Product Backlog				
	Item#	Description	Prio#	
Very High				
	1	Arpojan eli adminin autentikointi sisäänkirjautuessa	1	PaP
	2	Jokainen voi jättää arvan arvonta kaudelle, eli osallistujalle ei autentikointia	1	PaP
	3	Arvassa olatava yksi - kolme ajanjaksoa	1	PaP
	4	Vain yksi arpa henkilöä kohden per arvonta kausi	1	PaP
	5	Arvassa kenttä "lyhyempi käy" joka tarkoittaa, että arpoja voi lyhentää kolmea ajanjaksoa	1	PaP
	6	Näkymä arvalle, josta arpalipuke jätetään	1	PaP
	7	Näkymä arpojalle	1	PaP
	8	Arpojan näkymästä näytetään arvan jättäneet	1	PaP
	12	Näkymä josta valitaan joko jätä arpa tai järjestelmän kirjautuminen	1	PaP
	16	Arvontaan voi osallistua vain enen määräajan päättymistä	1	PaP
	17	Arvassa selitekenttä jolla voidaan koittaa vaikuttaa arpojaan	1	PaP
	18	Varauskalenteri johon voi lisätä myös arvonnan ulkopuolisia varauksia	1	PaP
High				
	9	Arpojan näkymässä kalenteri johon arvottavat ajanjaksot sijoitetaan	2	PaP
	10	Järjestelmä sijoittaa ei päällekkäiset ajanjaksot varauksiksi	2	PaP
	15	Järjestelmä ilmoittaa s-postin välityksellä voittiko arpa	2	PaP
	19	Active Directorysta "AD" nimi lyhenteellä sähköpostiosoitteen kysely	2	PaP
	21	Syötetyn arvan yhteydessä aiempien arpojen voittojen näyttäminen	2	PaP
Medium				
	11	Järjestelmä arpoo automaattisesti voittoarvat	3	PaP
	13	Järjestelmään luodaan tili arvontaan osallistuttaessa	3	PaP
	14	Järjestelmä lähettää Tunnarit s-postiin arvan jätön jälkeen	3	PaP
	20	Nimi lyhenteellä voittojen kysely järjestelmästä	3	PaP

KUVA 30. Arvontajärjestelmän kehitysjono

7.5 Suunnittelu

Suunnitteluvaiheessa tehtiin suunnitelma siitä, miten tietojärjestelmä toteutettaisiin Grailsilla. Suunnittelun kohteina olivat domain-luokat, controller-luokat ja näkymät. Suunnitteluvaiheessa suunniteltiin myös se, miten tietojärjestelmän kehitysjono toteutettaisiin jakamalla se Scrumin mukaisesti sprintteihin.

Tehtävälistojen suunnittelu

Tuotteen kehitysjonon korkeimman prioriteetin ominaisuuksista muodostettiin viisi sprintin tehtävälistaa (kuva 31). Tehtävälistat suunniteltiin siten, että jokaisen tehtävälistan toteuttaminen tukee seuraavan tehtävälistan toteuttamista. Jokainen tehtävälista muodostaa myös yhden jollain tapaa testattavan kokonaisuuden. Esimerkiksi domain-luokat voitiin testata helposti siten, että Grails-sovelluskehyksellä generoitiin controller-luokat ja näkymät.

Sprint 1	Status
Grailsin asennus	Completed
XAMPP asennus	Completed
SQLyog asennus	Completed
NetBeans asennus	Completed
Muut kehitysympäristön pystyttkseen liittyvät	Completed
Sprint 2	**Status**
Domain Luokkien suunnittelu	Completed
Domain luokkien toteutus	Completed
Autentikoinnin toteutus	Completed
Testaus näiltä osin	Completed
Sprint 3	**Status**
Käyttöliittymien Suunnittelu arpalippu/arpojan näkymä	Completed
Controller luokkien suunnittelu	Completed
Arpalippu näkymän toteutus	Completed
Controller luokka arpalipulle	Completed
Arpalipun testaus	Completed
Sprint 4	**Status**
Arpalippunäkymän muutos	Completed
Arpojan käyttöliitymän totetus	Completed
gridi arvoille	Completed
Kalenteri näkymä	Completed
gridin yhteys tietokantaan	Completed
Sprint 5	
Arpojan käyttöliitymän totetus	Completed
arpojen näyttö kalenterissa	Canceled
Arvonta ominaisuuden toteutus	Completed
Kantayhteys ulkopuliseen kantaan	Completed
	Not Started

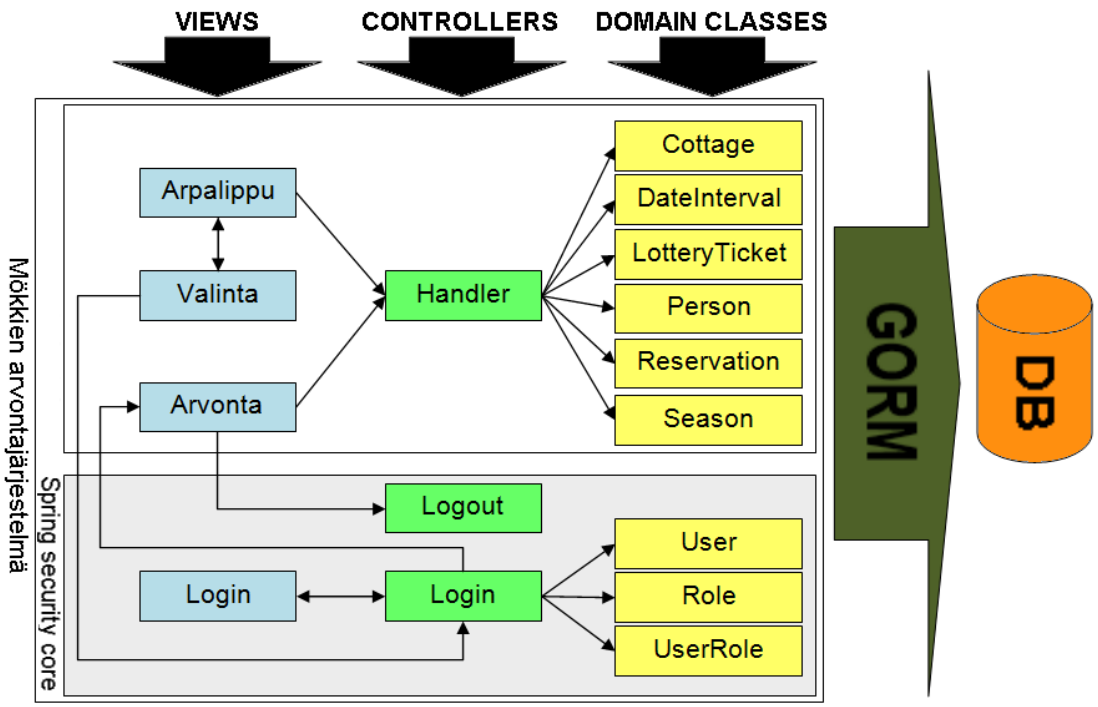
KUVA 31. Tehtävälistat

Arkkitehtuuri

Arkkitehtuurikuvassa (kuva 32) kuvataan järjestelmän keskeiset osat sekä niiden välistä kommunikointia. Kuvan vasemmassa laidassa sinisellä pohjalla olevat osat ovat järjestelmän käyttöliittymiä eli näkymiä. Toisena vasemmasta laidasta vihreällä pohjalla ovat järjestelmän controller-luokat. Kolmantena vasemmalta keltaisella pohjalla ovat domain-luokat. Toisena kuvan oikeasta laidasta oleva vihreä nuoli kuvaa sitä, miten domain-luokat ovat sidoksissa tietokantaan. Kuvassa esitetään myös Grailsin käyttämä GORM-teknologia, jota käytetään tietokantayhteyden muodostamiseen.

Arkkitehtuurikuvan (kuva 32) alareunassa harmaalla pohjalla olevat osat kuuluvat liitännäiseen nimeltä Spring security core. Tällä liitännäisellä järjestelmään

toteutettiin kirjautuminen. Sillä myös rajoitetaan käyttäjien käyttöoikeuksia järjestelmässä.



KUVA 32. Mökkien arvontajärjestelmän arkkitehtuuri

Domain-luokat

Domain-luokkien suunnittelu aloitettiin analysoimalla tuotteen kehitysjonoa tarvittavien domain-luokkien näkökulmasta, jotka ovat pohjana relaatiotietokannalle. Tämän analysoinnin lopputuloksena oli kuusi domain-luokkaa, joista jokainen kuvasi yhtä relaatiotietokannan taulua eli relaatiota, jotka muodostivat toisiinsa nähden riippuvuuksia.

Season on luokka, joka sisältää kuvauksen kaudesta. Kautta kuvataan neljällä attribuutilla. Ensimmäiset kaksi sisältävät kauden alkamis- ja loppumispäivämäärän. Seuraavat kaksi attribuuttia sisältävät alkamis- ja loppumispäivämäärän aikavälille, jolloin arvontaan voidaan osallistua. Mökeistä muodostettiin Cottage-luokka. Tämä luokka sisältää kuvauksen mökistä ja tiedon siitä, mille kaudelle mökki kuuluu. Arpalipuista muodostettiin LotteryTicket-luokka, joka sisältää instansit kolmesta DateInterval-luokasta, viestin ja boolean-arvon siitä, onko arpa voittanut. Arpalippu kuuluu lisäksi jollekin henkilölle sekä kaudelle, joten LotteryTicet-luokasta on yhteys Person- ja Season-luokkaan.

Aikavälistä muodostui arpalipulle kuuluva DateInterval-luokka, joka sisältää aikavälin alkamis- ja loppumispäivämäärän sekä tiedon, voiko tämä aikaväli olla lyhyempi. Henkilöstä muodostettiin Person-luokka, joka sisälsi tiedon henkilön käyttäjätunnuksesta ja sähköpostiosoitteesta. Reservations-luokka sisältää päivämäärät varauksen alkamisesta ja loppumisesta. Se sisältää myös tiedot siitä, mille mökille varaus on tarkoitettu, mikä kausi on kysymyksessä ja kenelle varaus suoritetaan.

Näkymät

Arpalipukenäkymä suunniteltiin lomakkeeksi, johon tulevat henkilön käyttäjänimi ja sähköpostiosoite. Lomakkeeseen tulee myös yhdestä kolmeen aikaväliä. Aikaväleille voidaan myös merkata se, voiko aikaväli olla haluttua lyhyempi. Lomakkeessa on myös kenttä, johon voidaan halutessa kirjoittaa viesti arvontaa suorittavalle henkilölle. Kaikkien henkilöiden on myös pystyttävä osallistumaan arvontaan, joten järjestelmään kirjautumista ei tarvita tässä näkymässä.

Arvonnan suorittavalle henkilölle suunniteltiin näkymä, jossa voidaan tarkastella tietoja jätetyistä arpalipukkeista ja toteutuneista varauksista. Näkymässä pystytään myös suorittamaan arvontaa. Tämä tarkoittaa sitä, että jonkun henkilön jättämästä arpalipukkeen aikavälistä muodostetaan varaus mökistä arpalipukkeen jättäneelle henkilölle. Mökkeihin kohdistuneet varaukset näytetään jonkin tyyppisessä kalenterissa, josta varauksia voidaan muokata ja poistaa. Tähän näkymään tarvitaan kirjautuminen järjestelmään.

Näiden edellä kuvattujen näkymien lisäksi suunniteltiin myös näkymä, josta voidaan valita arvontaan osallistuminen tai järjestelmään kirjautuminen. Arvontaan osallistuminen avaa näkymän arpalipukkeen jättämistä varten, kun taas järjestelmään kirjautuminen ohjaa käyttäjän näkymään, jossa autentikointi suoritetaan.

Kontrolleriluokat

Controller-luokkien tehtävänä on vastata mökkiarpajärjestelmän näkymistä lähetettyihin pyyntöihin. HandlerController-luokan tehtävänä on kontrolloida näkymien Arpalippu ja Arvonta pyyntöjä. LoginController-luokan tehtävänä on käsitellä kirjautumispyyntöjä Login-näkymältä. LogoutController-luokan tehtävänä

on huolehtia kaikista pyynnöistä, jotka koskevat järjestelmästä uloskirjautumista.

7.6 Toteutus

Järjestelmään toteutettiin suunnitellut domain-luokat, joiden lisäksi jouduttiin toteuttamaan yksi ennalta suunnittelematon domain-luokka. Luokan tehtävänä on avustaa mökkien varauksien näyttämistä arpojan näkymässä olevassa kalenterissa.

Mökeille kohdistuvia varauksia oli tarkoitus hallita RichUI-liitännäisellä toteutettujen näkymien kautta. Liitännäinen sisälsi kalenterin sekä muita rikkaita AJAX-komponentteja, joilla oli tarkoitus rakentaa järjestelmän näkymät.

Kalenteria ei kuitenkaan onnistuttu liittämään järjestelmään halutulla tavalla, joten näkymät toteutettiin ExtJS:llä. ExtJS:n valinta perustui siihen, että aikaisemmissa harjoitteluprojekteissa oli käytetty samaa JavaScript-kirjastoa.

Näkymien toteutuksessa käytetty ExtJS:n versio on 3.2. ExtJS on esitelty luvussa 7.3 Työkalut ja kirjastot. Järjestelmään toteutettiin myös kontrolleriluokka, joka käsittelee näkymiltä lähtevät pyynnöt. Järjestelmään toteutettiin ExtJS:n avulla kolme erillistä käyttäjärajapintaa, jotka esitellään seuraavassa luvussa.

7.7 Järjestelmän käyttäjärajapintojen toteutus

Liitteessä 2 on valintanäkymä, jossa käyttäjä valitsee joko järjestelmään kirjautumisen tai arvontaan osallistumisen. Valintanäkymä toteutettiin kahdella painikkeella, jotka on sijoitettu paneeliwidgettiin. Toinen painike ohjaa käyttäjän arvontanäkymään järjestelmään kirjautumisen kautta. Toinen painike ohjaa käyttäjän arpanäkymään, josta arvontaan voi osallistua. Toimintaperiaate on piirretty nuolien avulla arkkitehtuurikuvassa (kuva 32). Arpanäkymän avautuminen kuitenkin edellyttää, että kauden arvonta-aika on meneillään, jolloin arvontaan voi osallistua. Kuitenkin, jos arpanäkymää yritetään avata muulloin, järjestelmä ilmoittaa, millä aikavälillä seuraava arvonta on avoinna.

Näkymä, josta järjestelmään kirjautuminen tapahtuu, on Spring security core -liitännäisen generoima näkymä (liite 5). Näkymä generoidaan, kun Spring security core liitetään järjestelmään ja suoritetaan liittämisen edellyttämät toimenpiteet. Näkymässä on kaksi syöttökenttää, joihin käyttäjä kirjoittaa käyttäjätunnuksen ja salasanan.

Arpalippunäkymästä (liite 6) tapahtuu arvontaan osallistuminen. Arpalippunäkymä toteutettiin siten, että se koostuu neljästä eri kentästä, joista yksi on osallistujan tietoja varten. Osallistuja kirjoittaa tähän kenttään nimensä, sähköpostiosoitteensa ja halutessaan viestin. HandlerController-luokka muodostaa näistä tiedoista uuden henkilön järjestelmään, tai jos järjestelmästä löytyy sama henkilö, arpa lisätään silloin henkilön meneillään olevan kauden arvaksi. Näkymään on toteutettu kolme aikavälikenttää, joihin osallistuja syöttää haluamansa aikavälit ja ilmoittaa myös sen, voiko se olla haluttua aikaväliä lyhyempi.

Arpojan näkymä (liite 7) on toteutettu jakamalla se kahteen widgetpaneeliin: läntiseen ja itäiseen. Läntisen paneelin avulla voidaan selata mökkien varauksia ja jätettyjä arpoja. Itäisestä paneelista voidaan luoda kausia ja suorittaa arvontaa. Läntinen paneeli itsessään sisältää kaksi taulukkowidgettiä (liite 8), joista ensimmäiseen taulukoon listataan arvontaan osallistuneet arvat. Toiseen taulukkoon listataan kaikki toteutuneet varaukset.

Itäinen paneeli on jaettu kahteen paneeliwidgettiin (liite 7), joista toinen sisältää taulukon kausista ja toisen paneelin sisältö vaihtelee kolmen widgetin välillä (liite 9). Ensimmäinen näistä kolmesta widgetistä on dynaaminen lomake, jolla muodostetaan arvontakaudet. Lomake sisältää arvonta- sekä varauskauden alkamis- ja loppumisajankohdat. Lomakkeessa voidaan dynaamisesti vaihdella mökki-kenttien lukumäärää yhden ja kolmen välillä. Mökki-kentät sisältävät syöttökentän, johon syötetään kuvaus mökistä.

Keskimmäinen widgetti liitteessä 6 on mökkien varaamista varten, josta tarkempi kuva liitteessä seitsemän. Tämän widgetin pohjoinen osio sisältää taulukon, jossa on rivi jokaista kauden alkamis- ja päättymispäivämäärän välillä olevaa päivää kohden. Se sisältää myös sarakkeen jokaista kauteen osallistuvaa

mökkiä kohti. Taulukkoon merkitään henkilön nimi sarakkeeseen, joka vastaa hänelle varattavaa mökkiä. Nimen merkintä suoritetaan jokaiselle varauksen alkamis- ja päättymispäivämäärän välissä sijaitsevalle päivämäärää vastaavalle riville. Taulukossa indikoidaan täysin varauksettomat päivät vihreällä värillä ja punaisella päivät, jolloin kaikki mökit ovat varattuina. (Liite 10.)

Eteläinen osa sisältää lomakewidgetin, jolla varaus suoritetaan. Varauksen alkamispäivämäärä saadaan lomakkeeseen valitsemalla päivämäärää vastaava rivi yläpuolen taulukosta. Rivin valinta vaikuttaa myös varaukselle valittavissa oleviin mökkeihin. Kentässä, jossa mökin valinta suoritetaan, rajoitetaan valittavissa olevat mökit vain vapaisiin (liite 10). Varauksen tekeminen muuttaa henkilön arvan voittaneeksi ja poistaa henkilön lomakkeen henkilövalinnan pudotusvalikosta. Arpojan näkymän toimintalogiikalla pyritään estämään päällekkäisten varauksien syntyminen ja helpottamaan arvontatehtävää.

8 POHDINTA

Työssä saatiin valmiiksi web-pohjainen mökkienarvonta- ja varausjärjestelmä, joka on toteutettu Grails-sovelluskehyksellä. Järjestelmän kayttöliittymät on suunniteltusta poiketen toteutettu ExtJS:llä, vaika tähän oli tarkoitus käyttää RichUI-liitännäistä. Järjestelmään toteutettiin kaikki korkeimman prioriteetin ominaisuudet, joiden toteuttaminen oli asetettu tavoitteeksi järjestelmän toteutukseen varatussa ajassa. Kaikkien korkeimman prioriteetin ominaisuuksien toteuttaminen mahdollisti järjestelmän käyttöönotton. Matalamman prioriteetin ominaisuuksista jäi hyvä pohja järjestelmän jatkokehitykselle.

Tästä prosessista muodostui selkeä kuva siitä, mitä kaikkea on otettava huomioon suunniteltaessa ja toteuttaessa tietojärjestelmää. Prosessin aikana muodostui myös kuva siitä, että pienen tietojärjestelmän toteuttaminen voi vaatia kokonaisuudessa suuren työn, jotta järjestelmä saadaan toimimaan halutulla tavalla. Aikaa kannattaa todella käyttää tietokannan suunnitteluun. Tässä yhteydessä tarkoitan Grailsin käyttämien domain-luokkien suunnittelua. Niitä voidaan käyttää hyväksi esimerkiksi tietokantojen suunnitteluprosessia, joka on esitelty Michael J. Hernandezin kirjassa Tietokannat suunnittelu käytännössä.

Tarkastellessani jälkeenpäin arvontajärjestelmää toteuttaisin sen hieman eri tavalla kuin se on nyt toteutettu. Esimerkiksi arvontajärjestelmän domain-luokat suunnittelisin erilaisiksi, jolloin tallennetusta tiedosta muodostettava informaatio olisi helpompi esittää. Ongelmat informaation esittämisestä tulivat esille erityisesti arpojan näkymän toteutuksen aikana, jolloin arpojalla on oltava yhtä aikaa paljon informaatiota käytettävissä.

Myös Scrum kehitysmenetelmänä tuli tutuksi ja osoittautui yhdeksi varsin mielenkiintoiseksi osaksi tätä prosessia. Menetelmästä ja sen prosessista kehittyi selkeä kuva. Uutena asiana koin sen, kuinka Scrumissa voidaan mitata sijoitetun pääoman tuottoprosenttia ja sillä saavutetaan todellista liiketaloudellisesta hyötyä. Scrum osoittautui kehitysmenetelmänä myös joustavaksi ja mukailemalla siitä saatiin hyvät puitteet tähän kehitysprosessiin.

Grails-sovelluskehyksen ja samantyyppisten sovelluskehyksien toiminta selkeytyi prosessin aikana. Järjestelmän ongelmallinen asentaminen palvelimelle toi taas esille sen, että kaikki ei välttämättä ole niin yksinkertaista kuin ohjeissa annetaan ymmärtää. Asennusvaiheesta muodostui kaksiviikkoinen ongelmalähtöiseen oppimiseen keskittynyt kokonaisuus. Asennusvaiheen aikana Linux-pohjaiset palvelimet tulivat varsin tutuiksi.

LÄHTEET

Dearle, Fergal 2010. Groovy for Domain-Specific Languages. Birmingham, UK: Packt publishing Ltd.

Docs.oracle.com. 2011. The Java™ Tutorials. Saatavissa: http://docs.oracle.com/javase/tutorial/index.html

Enberg, Pekka – Raunio, Mika 2007. Sulkeumat. Saatavissa: http://www.cs.helsinki.fi/u/wikla/OKP/ArtikkelitK07/sulkeumat.pdf. Hakupäivä 24.11.2011.

Feature Estimation. 2011. Saatavissa: http://www.versionone.com/Agile101/Feature_Estimation.asp. Hakupäivä 17.11.2011.

Grails.org. 2012. Saatavissa: http://grails.org/ Hakupäivä 20.1.2012.

Groovy.codehaus.org. 2011. Saatavissa: http://groovy.codehaus.org/. Hakupäivä 23.11.2011.

Hernandez, Michael J 2000. Tietokannat, suunnittelu ja toteutus. Suom. Tomi Kajala. Edita, IT Press

Judd, Christopher M. - Nusairat, Joseph Faisal – Shingler, James 2008. Beginning Groovy and Grails: From Novice to Professional. New York, Springer-Verlag, Inc.

Ketterät käytännöt.fi 2008. Scrum. Saatavissa: http://www.ketteratkaytannot.fi/fi-FI/Kaytannot/. Hakupäivä 17.11.2011.

Rocher, Graeme Keith 2006. The Definitive Guide to Grails. New York, Springer-Verlag, Inc.

Rocher, G – Ledbrook, P – Palmer ,M – Brown, J – Daley, L – Beckwith, B. The Grails Framework - Reference Documentation, version 1.3.7. Saatavissa: http://grails.org/doc/1.3.7/guide/index.html. Hakupäivä 22.2.2012.

Sencha.com 2011. ExtJS. Saatavissa: http://www.sencha.com/products/extjs/. Hakupäivä 24.2.212

Schwaber, Ken – Beedle, Mike 2001. Agile Software Development with Scrum. Upper Saddle River, NJ: Prentice-Hall, Inc.

Sutherland, Jeff 2010. Scrum Handbook. Saatavissa: http://jeffsutherland.com/scrumhandbook.pdf. Hakupäivä 16.11.2011.

LIITTEET

Operators	**Precedence**
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

(Docs.oracle.com. 2011, hakusana operators.)

Operator Name	Symbol	Description
Spaceship	<=>	Useful in comparisons, returns -1 if left is smaller 0 if == to right or 1 if greater than the right
Regex find	=~	Find with a regular expresion? See Regular Expressions
Regex match	==~	Get a match via a regex? See Regular Expressions
Java Field Override	.@	Can be used to override generated properties to provide access to a field
Spread	*.	Used to invoke an action on all items of an aggregate object
Spread Java Field	*.@	Amalgamation of the above two
Method Reference	.&	Get a reference to a method, can be useful for creating closures from methods
asType Operator	as	Used for groovy casting, coercing one type to another.
Membership Operator	in	Can be used as replacement for collection.contains()
Identity Operator	is	Identity check. Since == is overridden in Groovy with the meaning of equality we need some fallback to check for object identity.
Safe Navigation	?.	returns nulls instead of throwing NullPointerExceptions
Elvis Operator	?:	Shorter ternary operator

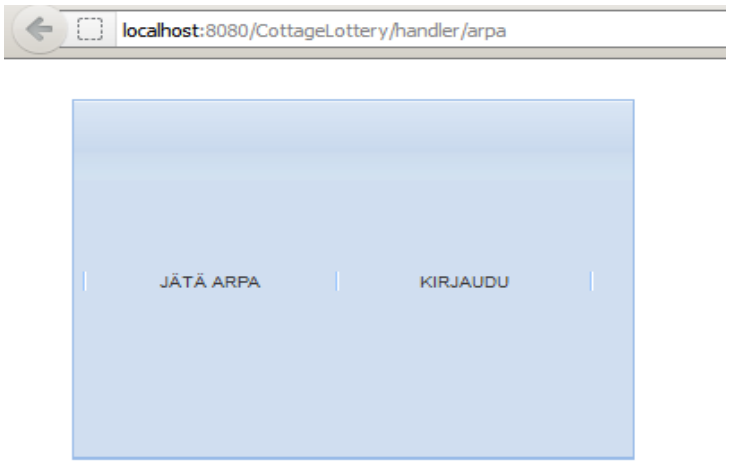
(Groovy.codehaus.org. 2011, hakusana operators.)

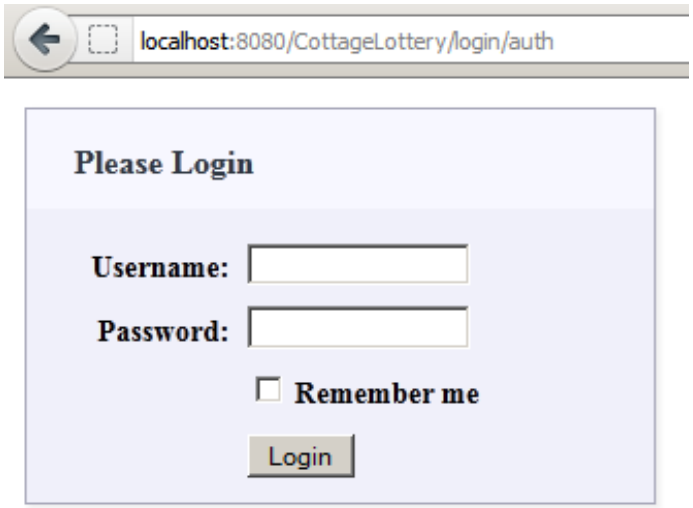
Operator	Method
a + b	a.plus(b)
a - b	a.minus(b)
a * b	a.multiply(b)
a ** b	a.power(b)
a / b	a.div(b)
a % b	a.mod(b)
a | b	a.or(b)
a & b	a.and(b)
a ^ b	a.xor(b)
a++ or ++a	a.next()
a-- or --a	a.previous()
a[b]	a.getAt(b)
a[b] = c	a.putAt(b, c)
a << b	a.leftShift(b)
a >> b	a.rightShift(b)
switch(a) { case(b) : }	b.isCase(a)
~a	a.bitwiseNegate()
-a	a.negative()
+a	a.positive()

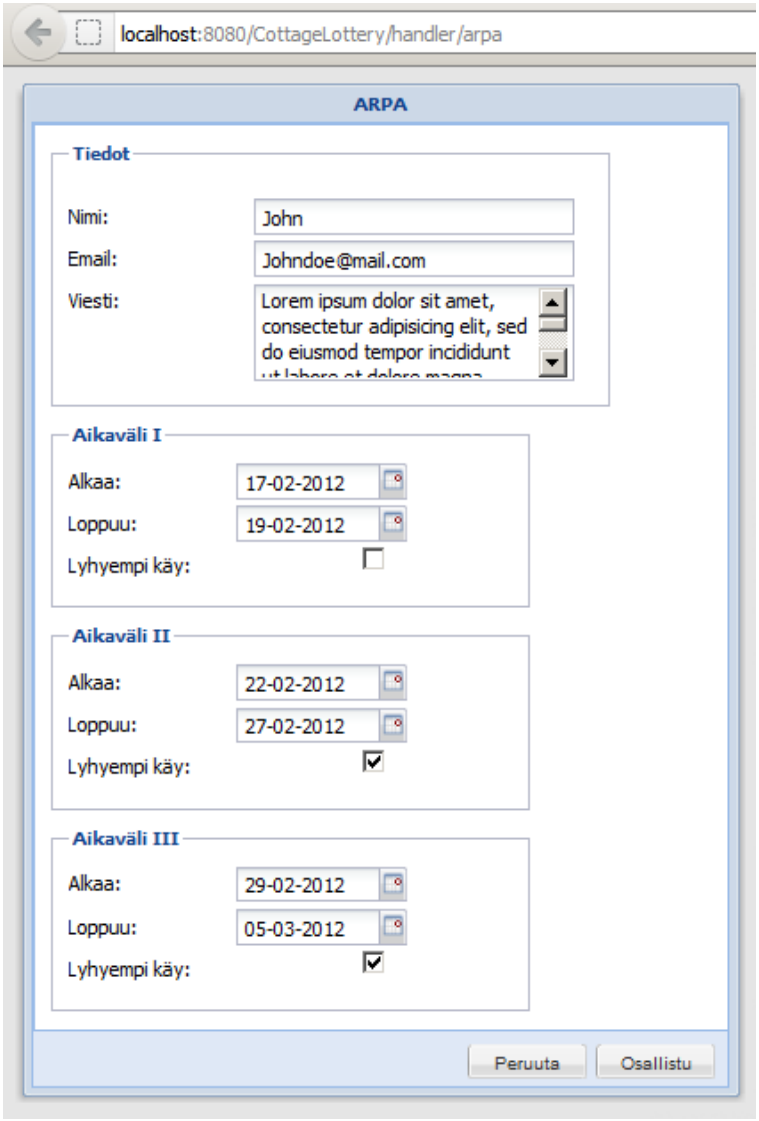
(Groovy.codehaus.org. 2011, hakusana operator overloading.)

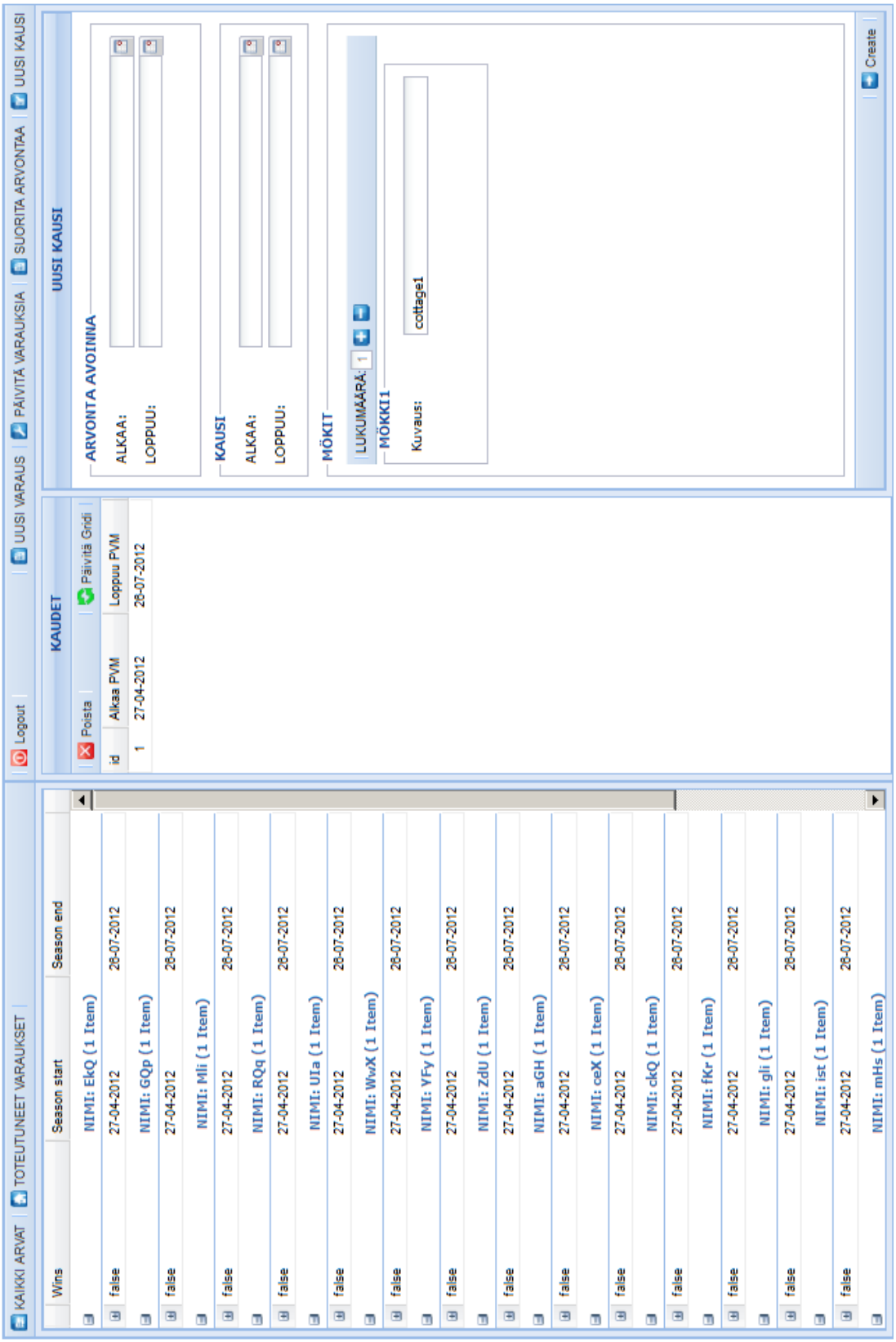
```
Examples:
grails dev run-app
grails create-app books

Available Targets (type grails help 'target-name' for more info):
grails add-proxy
grails bootstrap
grails bug-report
grails clean
grails clear-proxy
grails compile
grails console
grails create-app
grails create-controller
grails create-domain-class
grails create-filters
grails create-hibernate-cfg-xml
grails create-integration-test
grails create-plugin
grails create-script
grails create-service
grails create-tag-lib
grails create-unit-test
grails dependency-report
grails doc
grails generate-all
grails generate-controller
grails generate-views
grails help
grails init
grails install-dependency
grails install-plugin
grails install-templates
grails integrate-with
grails interactive
grails list-plugin-updates
grails list-plugins
grails package
grails package-plugin
grails plugin-info
grails release-plugin
grails remove-proxy
grails run-app
grails run-script
grails run-war
grails schema-export
grails set-proxy
grails set-version
grails shell
grails stats
grails test-app
grails uninstall-plugin
grails upgrade
grails war
C:\test>_
```

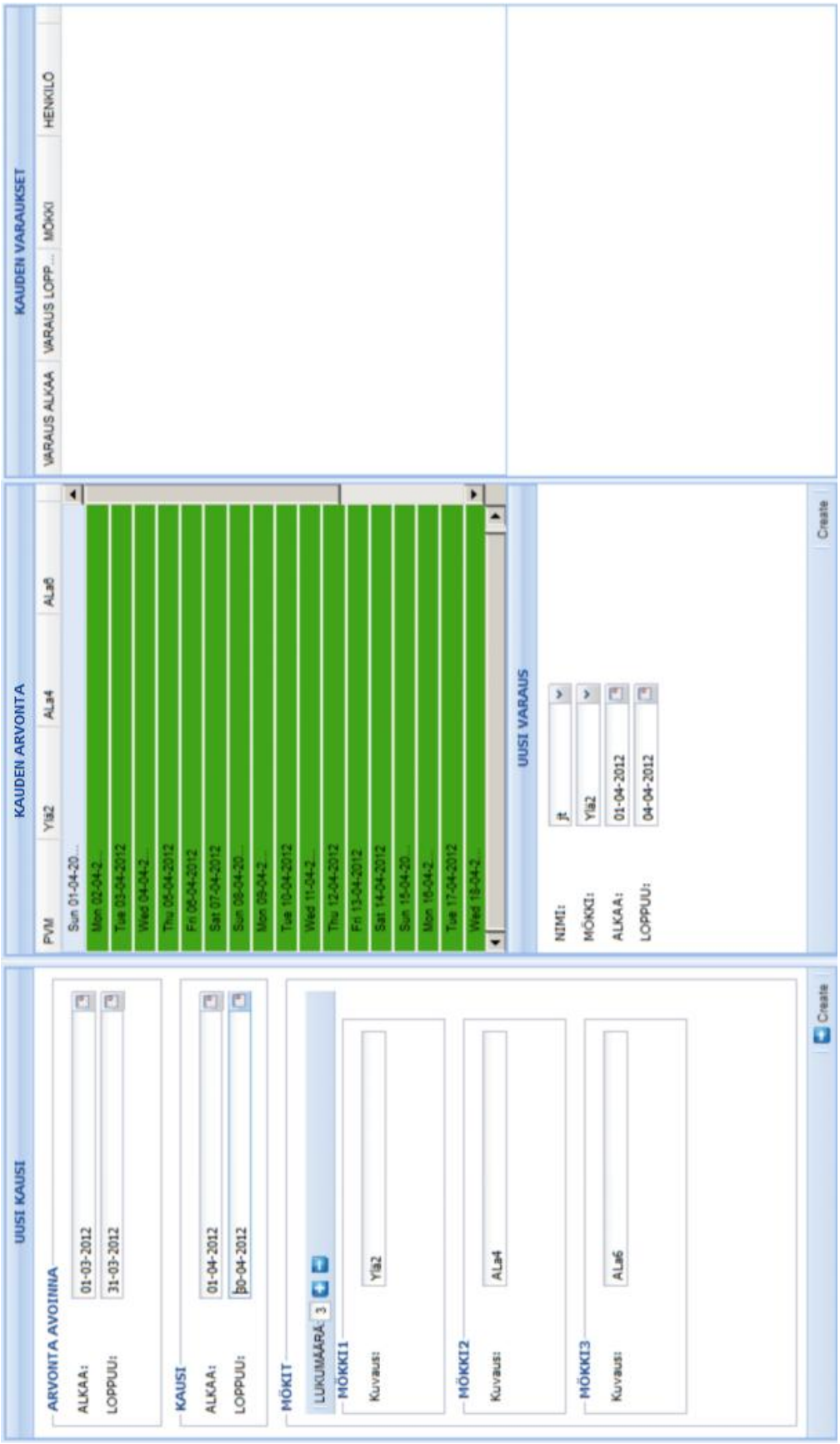

localhost:8080/CottageLottery/handler/arpa
JÄTÄ ARPA
KIRJAUDU

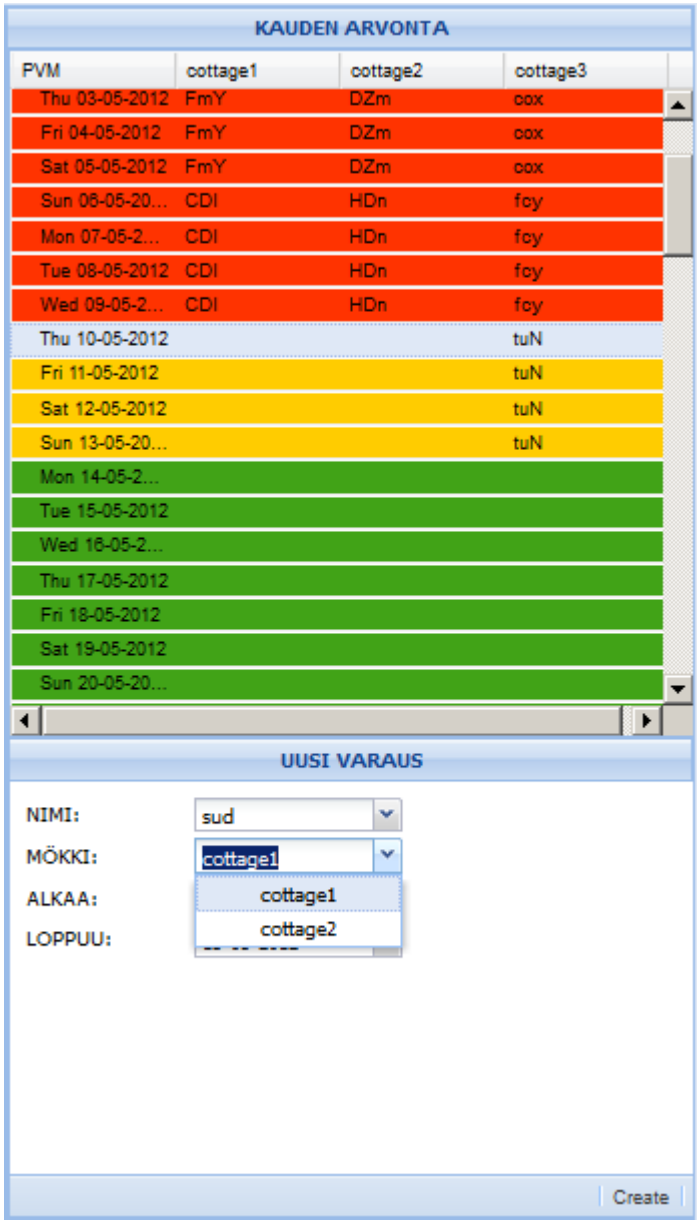

localhost:8080/CottageLottery/login/auth
Please Login
Username:
Password:
Remember me
Login


localhost:8080/CottageLottery/handler/arpa
ARPA
Tiedot
Nimi:
John
Email:
Johndoe@mail.com
Viesti:
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt
Aikaväli I
Alkaa:
17-02-2012
Loppuu:
19-02-2012
Lyhyempi käy:
Aikaväli II
Alkaa:
22-02-2012
Loppuu:
27-02-2012
Lyhyempi käy:
Aikaväli III
Alkaa:
29-02-2012
Loppuu:
05-03-2012
Lyhyempi käy:
Peruuta
Osallistu


KAIKKI ARVAT
TOTEUTUNEET VARAUKSET
Logout
UUSI VARAUS
PÄIVITÄ VARAUKSIA
SUORITA ARVONTAA
UUSI KAUSI
KAUDET
Poista
Päivitä Gridi
id
Alkaa PVM
Loppuu PVM
1
27-04-2012
28-07-2012
UUSI KAUSI
ARVONTA AVOINNA
ALKAA:
LOPPUU:
KAUSI
ALKAA:
LOPPUU:
MÖKIT
LUKUMÄÄRÄ:
1
MÖKKI1
Kuvaus:
cottage1
Create
Wins
Season start
Season end
NIMI: EkQ (1 Item)
false
27-04-2012
28-07-2012
NIMI: GQp (1 Item)
false
27-04-2012
28-07-2012
NIMI: Mli (1 Item)
false
27-04-2012
28-07-2012
NIMI: RQq (1 Item)
false
27-04-2012
28-07-2012
NIMI: UIa (1 Item)
false
27-04-2012
28-07-2012
NIMI: WwX (1 Item)
false
27-04-2012
28-07-2012
NIMI: YFy (1 Item)
false
27-04-2012
28-07-2012
NIMI: ZdU (1 Item)
false
27-04-2012
28-07-2012
NIMI: aGH (1 Item)
false
27-04-2012
28-07-2012
NIMI: ceX (1 Item)
false
27-04-2012
28-07-2012
NIMI: ckQ (1 Item)
false
27-04-2012
28-07-2012
NIMI: fKr (1 Item)
false
27-04-2012
28-07-2012
NIMI: gli (1 Item)
false
27-04-2012
28-07-2012
NIMI: ist (1 Item)
false
27-04-2012
28-07-2012
NIMI: mHs (1 Item)


KAIKKI ARVAT
TOTEUTUNEET VARAUKSET
KAIKKI VARAUKSET
Season start
Season end
NIMI: Jane (1 Item)
Ulkopuolinen
Varaus:
Aikaväli: 01.03.2012 - 05.03.2012
Mökki: Ala2
NIMI: Jaska (1 Item)
Ulkopuolinen
KAIKKI ARVAT
TOTEUTUNEET VARAUKSET
Wins
Season start
Season end
NIMI: EkQ (1 Item)
false
27-04-2012
26-07-2012
NIMI: GQp (1 Item)
false
27-04-2012
26-07-2012
NIMI: Mli (1 Item)
NIMI: RQq (1 Item)
false
27-04-2012
26-07-2012
NIMI: Ula (1 Item)
NIMI: WwX (1 Item)
false
27-04-2012
26-07-2012
NIMI: YFy (1 Item)
false
27-04-2012
26-07-2012
Toivotut aikavälit:
Toive 1: 21-08-2012 - 25-08-2012
Toive 2: 17-01-2013 - 21-01-2013
Toive 2: 15-08-2012 - 20-08-2012
Viesti:
Lorem ipsum dolor sit amet, consectetur
NIMI: ZdU (1 Item)
false
27-04-2012
26-07-2012
Toivotut aikavälit:
Toive 1: 22-08-2012 - 29-08-2012
Toive 2: 03-02-2013 - 09-02-2013
Toive 2: 01-05-2012 - 05-05-2012
Viesti:
Lorem ipsum dolor sit amet, consectetur
NIMI: aGH (1 Item)
false
27-04-2012
26-07-2012
NIMI: ceX (1 Item)
false
27-04-2012
26-07-2012
NIMI: ckQ (1 Item)


KAUDEN VARAUKSET
VARAUS ALKAA
VARAUS LOPP...
MÖKKI
HENKILÖ
KAUDEN ARVONTA
PVM
Ylä2
ALa4
ALa6
Sun 01-04-20...
Mon 02-04-2...
Tue 03-04-2012
Wed 04-04-2...
Thu 05-04-2012
Fri 06-04-2012
Sat 07-04-2012
Sun 08-04-20...
Mon 09-04-2...
Tue 10-04-2012
Wed 11-04-2...
Thu 12-04-2012
Fri 13-04-2012
Sat 14-04-2012
Sun 15-04-20...
Mon 16-04-2...
Tue 17-04-2012
Wed 18-04-2...
UUSI VARAUS
NIMI:
jk
MÖKKI:
Ylä2
ALKAA:
01-04-2012
LOPPUU:
04-04-2012
Create
UUSI KAUSI
ARVONTA AVOINNA
ALKAA:
01-03-2012
LOPPUU:
31-03-2012
KAUSI
ALKAA:
01-04-2012
LOPPUU:
30-04-2012
MÖKIT
LUKUMÄÄRÄ:
3
MÖKKI1
Kuvaus:
Ylä2
MÖKKI2
Kuvaus:
ALa4
MÖKKI3
Kuvaus:
ALa6
Create


KAUDEN ARVONTA
PVM
cottage1
cottage2
cottage3
Thu 03-05-2012 FmY DZm oox
Fri 04-05-2012 FmY DZm oox
Sat 05-05-2012 FmY DZm oox
Sun 06-05-20... CDI HDn foy
Mon 07-05-2... CDI HDn foy
Tue 08-05-2012 CDI HDn foy
Wed 09-05-2... CDI HDn foy
Thu 10-05-2012 tuN
Fri 11-05-2012 tuN
Sat 12-05-2012 tuN
Sun 13-05-20... tuN
Mon 14-05-2...
Tue 15-05-2012
Wed 16-05-2...
Thu 17-05-2012
Fri 18-05-2012
Sat 19-05-2012
Sun 20-05-20...
UUSI VARAUS
NIMI:
sud
MÖKKI:
cottage1
ALKAA:
LOPPUU:
cottage1
cottage2
Create