OULUN SEUDUN
AMMATTIKORKEAKOULU

Teemu Manninen

HARJOITUSKIRJAAMO:
WEB-SOVELLUS AMPUMAHARRASTUSTOIMINTAAN

HARJOITUSKIRJAAMO:
WEB-SOVELLUS AMPUMAHARRASTUSTOIMINTAAN

Teemu Manninen
Opinnäytetyö
Kevät 2012
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehitys

Tekijä(t): Teemu Manninen
Opinnäytetyön nimi:
Harjoituskirjaamo: Web-sovellus ampumaharrastustoimintaan
Työn ohjaaja(t): Eero Nousiainen, Oulun seudun ammattikorkeakoulu
Työn tilaaja(t): Petteri Aro, T:mi Proguns P. Aro
Työn valmistumislukukausi ja -vuosi: Kevät 2012
Sivumäärä: 63 + 7 liitettä

Opinnäytetyönä toteutettiin ampumaharjoittelun seurantaan, dokumentointiin ja käyttäjähallintaan soveltuva kolmiportainen web-sovellus. Työn toimeksiantajana oli yritys T:mi Proguns P Aro., joka toimii aktiivisesti ampumaharrastuspiireissä ja mainosalalla.

Järjestelmän rakentamisen tavoitteena oli helpottaa ampumaharrastajien harrastustoimintaa tarjoamalla heille päiväkirja, johon voidaan kirjata kaikki omat harjoitukset sekä kilpailut, joihin on osallistunut. Järjestelmän tarkoitus on toimia myös ampumaharjoittelun seurantakirjana viranomaisen hyväksymälle ampuma-asekouluttajalle. Tavoitteena oli myös toteuttaa samaan sovellukseen käyttäjähallintajärjestelmä, joka toimisi ampumaseurojen edustajien sähköisenä työvälineenä.

Työssä käytettiin uusimpia Web 2.0 -tekniikoita, Grails -ohjelmistokehystä sekä Groovy-ohjelmistokieltä, ja se toimisi myös testiympäristönä uusien webprojektien toteutuksessa. Projektinhallinta toteutettiin ketterään ohjelmistokehitykseen kuuluvalla Scrum-menetelmällä. Opinnäytetyötä edelsi työharjoittelu, jolloin näitä tekniikoita ja menetelmiä opiskeltiin.

Lopputuloksena saatiin aikaan toimiva prototyyppi, jota voidaan esitellä viranomaisille, ampumaharrastajille ja -seuroille. Prototyyppi tullee toimimaan myös markkinointitarkoituksessa ja sen avulla haetaan uutta aluevaltausta liiketoiminnassa. Työn toteutus onnistui suunnitellusti ja kaikki prototyyppiin vaaditut ominaisuudet saatiin toteutettua.

Asiasanat: Grails, Groovy, HTML5, web-sovellus

ABSTRACT

Oulu University of Applied Sciences
Information Technology, software deveplopment

Author(s): Teemu Manninen
Title of thesis: Harjoituskirjaamo: Web Application To Shooting Activities
Supervisor(s): Eero Nousiainen, Oulu University of Applied Scienses
Commissioned by: Petteri Aro, Proguns P. Aro
Term and year: Spring 2012
Pages: 63 + 7 appendices

The purpose of this thesis was to implement Training registry web application. The project was commissioned by Proguns P. Aro which operates in the advertising industry. This system was aimed at the shooting hobby and all associated activities. The application is a prototype, which is used for marketing purposes only.

The system is so-called three-steps-application and its users are shooters, instructors and shooting-clubs. Shooters will use it as a practise-diary and instructors can monitor shooters activity within it. The clubs will use application to manage the user registry. A similar product is not on the market.

In the project was used a newest version of the Groovy-based Grails-framework and Scrum project management method. The project was divided into two phases, planning and implementation. In the planning phase was choosed the appropriate tools and set out the requirements.

As a result was accomplished a fully functional Web-application. The project was successful and it was carried out within the allotted time. Let's hope that the product will continue to develop in the future.

Keywords: Grails, Groovy, HTML5, web application

ALKULAUSE

Tämä projekti on tehty Oulun seudun ammattikorkeakoulun tietotekniikan koulutusohjelman opinnäytetyönä. Työ aloitettiin marraskuussa 2011 ja se toteutettiin T:mi Proguns P. Arolle. Työn valvojana toimi lehtori Eero Nousiainen Oulun seudun ammattikorkeakoulusta.

Haluan osoittaa kiitokseni T:mi Progunsin toimitusjohtajalle Petteri Arolle mielenkiintoisesta työn aiheesta sekä työn valvojalleni Eero Nousiaiselle, joka johdatti minut projektin pariin ja auttoi työn edistymisessä.

30.4.2012

Teemu Manninen

SISÄLLYS

KESKEISIÄ KÄSITTEITÄ

CSS

Cascading Style Sheets on yhtenäinen tyylisäännöstö (= määrittelykieli), jolla luodaan WWW-sivustojen ulkoasu ja hallinnoidaan sivujen eri elementtejä.

Grails

Grails on ilmainen ja avoimeen lähdekoodiin perustuva ohjelmistokehys Groovy- ja Java-kielille.

Groovy

Groovy on ketteriin ohjelmistomenetelmiin perustuva dynaaminen ohjelmointikieli Java-virtuaalikoneelle.

HTML

Hypertext Markup Language on avoimesti standardoitu kuvauskieli, jolla voidaan kuvata hyperlinkkejä sisältävää tekstiä eli hypertekstiä. HTML tunnetaan erityisesti kielenä, jolla www-sivut on koodattu.

Tomcat

Tomcat on palvelin, jota käytetään Java- ja Groovy-pohjaisten sovellusten suorittamiseen.

WWW

World Wide Web on Internet-verkossa toimiva hajautettu hypertekstijärjestelmä, jonka Internet-selaimet tulkkaavat luettaviksi WWW-sivuiksi.

XAMPP

XAMPP on avoimen lähdekoodin palvelinpaketti web-sovelluskehitykseen.

1 JOHDANTO

Opinnäytetyön aiheena oli suunnitella ja toteuttaa uudenlainen web-sovellus, joka tulisi toimimaan keskeisenä osana ampumaharrastusta ja sähköisenä työvälineenä ampumaseuroille ja -asekouluttajille. Web-sovelluksen nimeksi valittiin Harjoituskirjaamo ja se tulisi toimimaan WWW-ympäristössä alustariippumattomana järjestelmänä.

Harjoituskirjaamo on rakennettu kolmiportaiseksi web-sovellukseksi, jolla on kolme käyttäjätasoa: ampumaseurat, -asekouluttajat ja harrastajat. Ampumaseurat voivat sovelluksen avulla hallita käyttäjiä sekä ampumaratoja. Ampuma-asekouluttajat voivat puolestaan seurata harrastajien harjoitteluaktiivisuutta ja harrastajat käyttävät sovellusta harjoittelu- ja kilpailupäiväkirjana.

Web-sovellus oli tarkoitus rakentaa prototyypiksi, joka toimisi lähinnä markkinointivälineenä ampumaurheilupiireissä. Lainsäädännön muutokset ampuma-aseisiin liittyvässä harrastustoiminnassa mahdollistavat uuden liike-idean, jonka perustaksi Harjoituskirjaamo-sovellus on toteutettu.

Opinnäytetyöni taustalla on toimeksiantaja T:mi Proguns P.Aro, jonka pääasiallisena toimialueena on mainosala ja ampumaurheilu. Toimeksiantaja havaitsi uuden liikeidean ja Internetiä hyödyntävän palvelun. Toimeksiantajaan viitataan tässä dokumentissa jatkossa käyttämällä nimitystä Tilaaja.

Toimeksiantajan alkuperäinen suunnitelma oli tilata pelkkä mobiilisovellus, mutta yhdessä päätimme toteuttaa järjestelmän alustariippumattomana web-sovelluksena, jotta se pystyisi palvelemaan monipuolisesti koko ampumaurheilutoimintaa ja harrastajakuntaa.

2 TYÖN LÄHTÖKOHDAT

Harjoituskirjaamo-sovellusta alettiin kartoittaa marrraskuussa 2011 ja sen ensimmäinen toteutusversio 0.1 toteutettiin opinnäytetyönä keväällä 2012. Sovellus on tietokantapohjainen, WWW-ympäristössä toimiva alustariippumaton järjestelmä.

2.1 Tausta

Lainsäädännön muutokset ovat tuomassa tarkemman harrastusseurannan miltei kaikkeen aseisiin liittyvään harrastusaktiviteetteihin. Erityisesti ampumaurheilun perusteella haettavat hankkimis- ja hallussapitoluvat vaativat jatkossa uusilta harrastajilta enemmän näyttöjä todellisesta harrastamisesta. Aselain määrittelemä minimi todennetuille harrastuskerroille on 10 kertaa vuodessa. Suomessa on yli 600 000 harrastajaa, jotka pitävät hallussaan noin 1,6 miljoonaa ampuma-asetta. (1.)

Aselain toisen vaiheen uudistuksen luonnoksissa näkyy ajatus, että kaikki luvat tulevat olemaan maksimissaan viisi vuotta voimassa, ja ne tulee uusia luotettavaa harrastusnäyttöä vastaan. Tämä aiheuttaa painetta harrastuspiireissä ja onkin odotettavissa, että tietotekniikka ja Internet tulevat mukaan kuvioihin helpottamaan ampumaharrastusta ja siihen liittyvää dokumentointia. (1.)

2.2 Tavoite

Opinnäytetyön tavoitteena oli tuottaa valmis sovellus, jonka avulla ampumaseurat hallinnoivat harrastajien käyttäjätilejä, kouluttajat seuraavat harrastajien harjoitteluaktiivisuutta ja harrastajat syöttävät harrastus- sekä kilpailutietoja. Kyseessä on kolmiportainen käyttäjätilijärjestelmä, joka on jaettu erilaisiin käyttäjärooleihin. Sovelluksessa tulisi olla myös rekisteröintimahdollisuus uudelle harrastajalle.

Tuotteen varsinaisesta julkaisuversiosta tehtäisiin shareware eli maksullinen sovellus, jolloin yksittäinen ampumaharrastaja ja -seura maksaisivat vuosilisenssimaksua sovelluksen käytöstä.

2.3 Käyttötarkoitus ja -kohde

Harjoituskirjaamo-sovellus tulisi palvelemaan ampumaseuroja sähköisenä työvälineenä. Ampumaseura voisi rekisteröityä palveluun, jolloin se saisi kirjautumistunnukset. Seura huolehtisi omien harrastajien ja kouluttajien käyttäjätilihallinnasta sekä seuralle kuuluvien ampumaratojen tiedotuksesta. Kaikki uudet käyttäjät tallennettaisiin tietokantaan ja ampumaseura voisi halutessaan muokata käyttäjän tietoja tai poistaa käyttäjän.

Ampumaharrastajalle web-sovellus tulisi toimimaan sähköisenä harjoituspäiväkirjana. Käyttäjä voisi lisätä harjoituksia tietoineen ja seurata omia harjoituksia ja harjoitushistoriaa. Lisäksi käyttäjä voisi syöttää mahdolliseen ampumakilpailuun osallistumisen tiedot. Omaa käyttäjätiliä olisi myös mahdollisuus muokata.

Harjoitustietojen syöttö on aiemmin toteutettu kirjallisesti paperille, mutta tällainen dokumentointi on käyttäjälle hidas ja epäluotettava tapa. Tietojen syöttö sähköisesti mahdollistaa tietojen pitkäaikaisen seurannan, nopeuttaa harjoitusdokumentin etenemistä harjoitustilanteesta viranomaiselle ja, mikä tärkeintä, pitää harjoitustiedot aina tallessa varmuuskopioiden avulla. Vastaavaa tuotetta ei ole olemassa.

Viranomaisen hyväksymän ampuma-asekouluttajan yhtenä tehtävänä on seurata oman seuran ampumaharrastajien harrastusaktiivisuutta vuositasolla. Harjoituskirjaamo-sovellus antaa valmiudet tähän, ja lisäksi kouluttajalla on mahdollisuus kirjoittaa todistuksia viranomaisille, jotka tekevät päätöksiä aseiden hankkimis- ja hallussapitoluvista.

3 OHJELMISTOPROJEKTIN HALLINTA

Projektityölle on ominaista aina alku ja loppu. Projektityölle määritellään yleensä myös käytettävät resurssit, kuten raha, työvoima ja materiaali. Projektinhallinnan tarkoitus on hallita resursseja ja viedä projekti läpi sovitun aikataulun ja budjetin mukaisesti. (2.)

Projektinhallinta sopii hyvin ohjelmistokehitykseen ohjelmistojen rakenteen sekä useiden työntekijöiden vuoksi. Suunnitteluvaiheessa ohjelmisto voidaan helposti pilkkoa osiin ja jakaa tehtävät työntekijöiden kesken. Ohjelmistoprojektilla on myös aina alku ja loppu, jonka tuloksena on valmis tuote. Projektinhallintaa voidaan myös käyttää yhden työntekijän projekteihin ja sen avulla voidaan hyvin aikatauluttaa ja hallinnoida omaa työmäärää, kuten tässä opinnäytetyössä on tehty. (3.)

3.1 Ketterä ohjelmistokehitys

Ohjelmistoprojektit ovat kehittyneet yhä nopeammin muuttuvassa ympäristössä. On huomattu, että ennalta suunnittelun vaikeus on noussut esteeksi vanhemmille tuotantomenetelmille, kuten esimerkiksi vesiputousmalli. Iteratiivisessa ajattelumallissa sen sijaan prosessin jokaiseen vaiheeseen voidaan tarttua ja korjauksia voidaan tehdä jo varhaisessa vaiheessa. Muutosten hallinta on nopeaa ja prosessi etenee nopeammissa sykleissä. Tämä uusi ajattelumalli on johtanut ketteriin menetelmiin. (4.)

Ketterät eli Agile-menetelmät ovat vakiintunut termi ja niillä kuvataan ohjelmistokehitysprojekteissa käytettävien menetelmien joukkoa. Agile-menetelmiä on useita, mainittakoon mm. Extreme Programming (XP), DSDM (Dynamic System Deveploment Method) sekä Scrum. Kaikille Agile-menetelmille on tyypillistä resurssien suuntaaminen halutulle alueelle, ja koska ohjelmistoprojekteissa on aina käytössä rajalliset resurssit, päästään tällä tavoin nopeammin haluttuun lopputulokseen. Tunnetuimpia Agile-menetelmien käyttäjiä omissa ohjelmistokehityksissään ovat esimerkiksi Google, Facebook ja Microsoft. (4.)

Ketterissä menetelmissä käytännön työ tapahtuu yleensä itseohjautuvissa tiimeissä. Tiimit pyrkivät itse parantamaan tehokkuutta ja löytämään ongelmakohtia omassa työssään. Ohjelmiston kehittelytyöhön osallistuvat myös aina liiketoiminnan edustajat, jolloin saadaan jatkuvaa palautetta ja pystytään jatkuvan vuorovaikutuksen avulla suuntaamaan projekteja nopeammin haluttuihin suuntiin. Tähän perustuu Agile-menetelmien tehokkuus. (4.)

3.2 Scrum-ohjelmistonkehitysmenetelmä

Scrum on yksi ketteristä menetelmistä, jota käytetään yleensä ohjelmistoprojekteissa, mutta yhtä hyvin sitä voidaan käyttää lähes missä tahansa projektityössä. Nimi tulee rugbyn pelitaktiikkapalavereista ja ydinajatuksena on reagoida nopeasti ympäristön tarpeisiin. Scrum-menetelmä painottaa iteratiivista suunnittelua ja edistymisen seurantaa. (5.)

Scrumissa työtä tehdään itsenäisesti, mutta yhtenäisin tavoittein aina tiimeittäin. Scrum-tiimi on aina itseorganisoituva sekä asiantunteva ja sen sisällä on erilaisia rooleja. Scrum-tiimin ulkopuolelta voidaan ainoastaan havainnoida, mutta ei puuttua itse tiimin työskentelyyn. (5.)

Scrum-tiimissä yksi jäsen toimii tuotteen omistajana, joka huolehtii tuotteen arvon maksimoimisesta. Tuoteomistaja tuntee tuotteen liiketoimintaa ja edustaa asiakkaita ja käyttäjiä. Toinen rooli Scrum-tiimissä on kehitystiimi. Kehitystiimi vastaa tuotteen kehityksestä aina julkaisuversioon saakka. Kehitystiimin jäsenillä voi olla erilaisia ammattitaitoja, mutta vastuu on tiimillä yhdessä. Scrum-menetelmässä ei ole varsinaista projektipäällikköä, vaan Scrum-master, joka ohjaa prosessia ja kouluttaa tarpeen vaatiessa tiimin jäseniä. (4.)

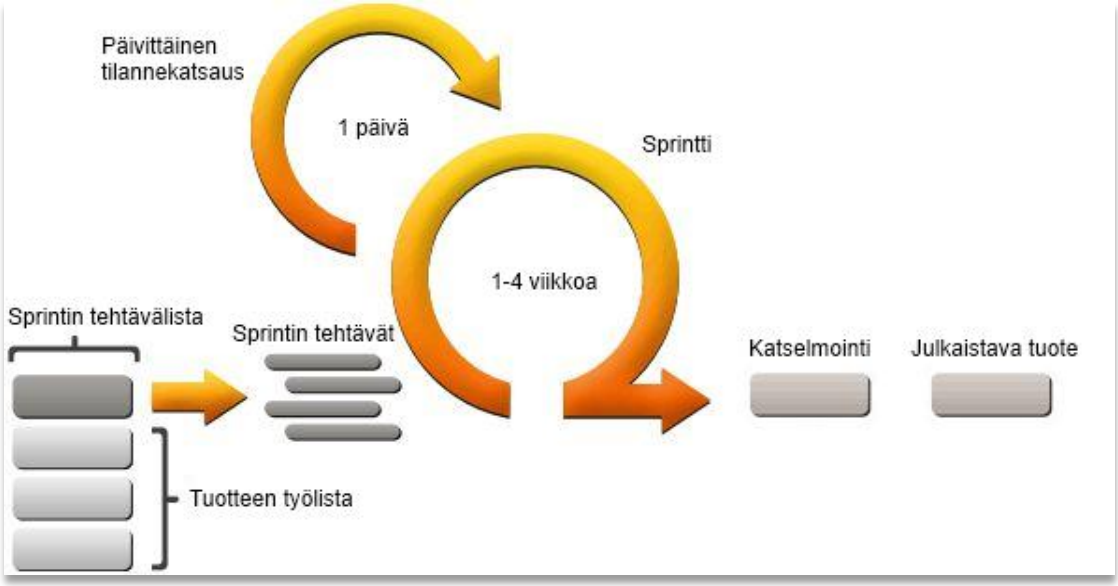
Kehitystyö Scrum-menetelmässä perustuu jaettuihin sprintteihin eli pyrähdyksiin. Sprintin pituus on maksimissaan neljä viikkoa tai 30 päivää. Päivittäin voidaan pitää kuitenkin tilannekatsaus, jossa katsotaan päivän työ ja seuraava työ. Jokaisen sprintin tuotoksena syntyy aina jokin uusi ominaisuus ja sprintin jälkeen suoritetaan aina sprintin katselmointi, ennen kuin siirrytään seuraavaan sprinttiin. (5.)

Scrumin tuotokset kuvaavat työn prosessia ja lisäävät läpinäkyvyyttä työn eri vaiheista. Lisäksi ne antavat mahdollisuuden tarkastella tuotekehitystä niin asiakkaan kuin kehittäjänkin näkökulmasta. (6.)

Tuotteen työlista (Product Backlog) on järjestetty lista kaikista tuotteen ominaisuuksista ja vaatimuksista. Tuotteen omistaja vastaa työlistan sisällöstä ja kehitystiimi sen toteuttamisesta. Työlista elää ja kehittyy tuotteen kehityksen mukana. (7.)

Sprintin tehtävälista (Sprint Backlog) koostuu sprintin sisäisistä tehtävistä, jotka suoritetaan aina yhden sprintin aikana. Sprintin tehtävälistan on aina oltava jokin valmis kokonaisuus, jonka mukaan tehtävät valitaan. Sprintin tehtävälistan ylläpito kuuluu kehitystiimille. (7.)

Scrumin tuotoksiin kuuluvat myös edistymiskäyrä (Burndown Chart), jolla kuvataan työmäärää ajan funktiona, sekä julkaisun työlista (Release Backlog). Julkaisun työlista on muuten kuin tuotteen työlista, mutta siihen on merkitty vain julkaistavan version ominaisuudet. Julkaisun työlistaa ei käytetä välttämättä kaikissa projekteissa (7). Scrum-prosessin elinkaari esitetään kuvassa 1.



KUVA 1. Scrum-prosessin elinkaari

3.3 Scrum-menetelmä osana opinnäytetyötä

Harjoituskirjaamo-sovellusta suunniteltaessa sovittiin, että projektinhallintaan käytetään Scrum-menetelmää. Vaikka yhden hengen projektissa en päässyt täysin hyödyntämään Scrumia, opin ymmärtämään sen periaatteen ja tavoitteen. Havaitsin nopeasti menetelmän hyödyllisyyden ohjelmoinnissa ja tarpeellisuuden projektissa, jossa on rajatut resurssit.

Projektin alussa pidettiin tilaajan kanssa suunnittelupalaveri, jossa selvitettiin, millaisen tuotteen tilaaja haluaa. Tämän perusteella laadittiin tuotteen työlista ja selvitettiin projektin resurssit. Resursseihin kuuluvat kaikki kehitysryhmän työntekijät sekä projektiin käytettävät työpäivät ja projektin kesto. Resursseista vähennetään tässä vaiheessa mahdolliset lomapäivät ja muihin projekteihin käytetty aika. Taulukosta on sen jälkeen helposti luettavissa kuinka paljon tiimillä on aikaa projektin toteuttamiseen. Tässä tapauksessa tunteja kertyi yhteensä 280 ja kehitystiimin kuului yksi henkilö.

OAMK
LIITE 2: Resurssien käytettävyys
Projekti: Harjoituskirjaamo
Scrum Master: Teemu Manninen
Aikataulu: 6.2. - 5.4.2012

Henkilö	ARKIPÄIVIÄ PROJEKTIN AIKANA	MUUHUN PROJEKTIIN	LOMAT	MUUT	PROJEKTITYÖN TEKEMISEEN PÄIVIÄ	PROJEKTIYÖN TEKEMISEEN TUNTEJA
Teemu Manninen	43	0	0	3	40	280
Yhteensä	**43**	**0**	**0**	**3**	**40**	**280**

KUVA 2. Resurssien käytettävyys

Tuotteen työlistaa laadittaessa mietittiin tuotteen prioriteetteja ja toteutusmahdollisuuksia resurssien rajoissa. Tässä vaiheessa jätettiin vielä muutamia tilaajan haluamia ominaisuuksia mietintään, koska niiden prioriteetti oli tässä vaiheessa tuotetta vielä varsin matala. Tuotteen työlista on kokonaisuudessaan liitteessä 1.

ID	Ominaisuuden kuvaus/käyttäjätarina/toiminnallisuus/laatuvaatimus	Sprintin numero	0	1	2	3	4		
		Päiviä jäljellä sprintissa	40	39	14	0	0		
		Päiviä käytetty	1	25	14	0	0	Huomioitavaa	Tila
1	SCRUM -opiskelu		1						
	SPRINTIN 0 MAALI: SCRUM dokumentaatio								
2	Grails projekti ja asetukset			4					
3	Suunnitellaan ja toteutetaan tietokanta			4					
4	Authentikointi ja kirjautuminen			4					
5	Toteutetaan roolitus(admin, seura, kouluttaja, käyttäjä)			4					
6	Toteutetaan kaikki sivut ja linkit			4					
11	Toteutetaan sähköpostipalvelu ja rekisteröinti			3					
8	Toteutetaan osoiterivin blokkaus ja testataan sovellusta			2					
	SPRINTIN 1 MAALI: OHJELMALLISET TOIMINNALLISUUDET								
12	Toteutetaan kouluttaja- sekä ratahaku				2				
10	Toteutetaan todistuksen luonti ja printtinäkymät				2				
7	Suunnitellaan käyttöliittymä				2				
9	Suunnitellaan ulkoasu				2				
13	Toteutetaan käyttöliittymä (napit, linkit, infotekstit ym)				3				
14	Toteutetaan ulkoasu(kuvat, tyylit ym)				3				

KUVA 3. Tuotteen ominaisuuslista

Varsinaisen kehitystyön käynnistyttyä päätettiin valita ensin muutamia ominaisuuksia, jotka toteutettiin ensimmäisessä sprintissä tärkeysjärjestyksen mukaisesti. Valitut ominaisuudet muodostivat täten sprintin tehtävälistan, joka jäi minun ylläpidettäväkseni. Sprintin tehtävälista on kehittäjän tärkein työkalu, koska siinä pilkotaan tuotteen ominaisuudet tehtäviksi ja resursoidaan käytettävä aika. Tässä työssä ominaisuudet vaativat useita aputehtäviä, koska kyseessä oli uusi ohjelmistokehys. Kuvassa 4 on näyte sprintin tehtävälistasta.

ID	Tehtävän ID	Ominaisuuden/tehtävän kuvaus	Tekijä(t)	Sprintin päivä	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				Päiväkohtainen resurssi tunneissa	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0
				Päivän aikana tehty tuntimäärä	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0
1		**GRAILS-Projekti**			Tehdyt tunnit sprintin aikana														
	1.1	Grails asetukset			7,0														
	1.2	Tietokantayhteydet				7,0													
	1.3	Bootstap. Tiedoston koodaus					7,0												
	1.4	Config. Tiedosto ja Url mappings asetukset						7,0											
2		**Tietokanta**																	
	2.1	Tietokannan suunnittelu							7,0										
	2.2	Tietokannan koodaus								4,0									
	2.3	Tietokantayhteydet								3,0	7,0								
3		**Authentikointi ja kirjautuminen**																	
	3.1	Haetaan spring security plugin ja asennetaan										4,0							
	3.2	testataan pluginia										3,0							
	3.3	Koodataan omat muutokset											7,0						

KUVA 4. Sprintin tehtävälista

Kaiken kaikkiaan Scrum-menetelmä osoittautui hyvin hyödylliseksi. Laadin sprintin tehtävälistan ennen varsinaisen ohjelmointityön alkua, jolloin pystyin keskittymään projektiin aina tehtävä kerrallaan. Liian suuria kokonaisuuksia ohjelmoidessa voidaan unohtaa osia sovelluksesta ja ajankäyttö on huomattavasti hankalampi resursoida. Lisäksi kokonaisuuden hallinta heikkenee ja ohjelmakoodi muuttuu yleensä sekavammaksi.

Täytyy kuitenkin muistaa, että Scrum-menetelmän täysi hyöty saavutetaan vasta usean henkilön projektilla, jossa voi olla useita eri alan ammattilaisia. Muutettaessa työlistaa sprintin tehtävälistaksi voidaan ominaisuudet pilkkoa juuri niin useaksi tehtäväksi kuin halutaan ja on tarpeellista. Mitä pienempiin tehtäviin projekti on jaettu, sitä helpompi sen resursseja on hallita. Samoin nähdään myös aikaisemmassa vaiheessa, riittääkö projektin käytössä olevat resurssit.

4 TEKNINEN TOTEUTUS

Opinnäytetyön yhtenä tavoitteena oli toteuttaa sovellus Web 2.0 -teknologiaan perustuvalla ohjelmistokehyksellä. Vaatimuksena oli ainoastaan, että kaikki ohjelmistotyökalut ovat avoimeen lähdekoodiin perustuvia ja että valmis prototyyppi toimii WWW-ympäristössä. Sovelluksen runkona päädyttiin käyttämään Grails-ohjelmistokehystä ja sovelluksen käyttöliittymä ja ulkoasu toteutettiin HTML5- sekä CSS3-tekniikoilla.

4.1 Web 2.0

Ennen kuin käsittelen syvemmin opinnäytetyön toteuttamiseen käytettyjä tekniikoita, on syytä mainita pari sanaa Web 2.0 -teknologiasta. Web 2.0 on perinteisen Webin uudempi, asiakaskeskeinen versio, jota pidetään WWW-sivustojen toisena vaiheena ja uutena liiketoimintamallina. Käsite on alun perin kehitetty markkinointitarkoitukseen vuonna 2004, mutta se on nykyään käytössä yleisenä terminä. (8.)

Ohjelmistotuotannossa Web 2.0 on tarkoittanut siirtymistä toiminnallisiin web-palveluihin, jotka nojaavat vahvasti sosiaaliseen verkostoitumiseen. Web 2.0 -sivustot ovat täysin WWW-pohjaisia ja sen sijaan, että ne olisivat staattisia ja suljettuja tietovarastoja, ne ovat enemmänkin avoimia käyttäjäkeskeisiä palveluita. Käyttäjä voi itse vaikuttaa järjestelmän sisältöön tallentamalla tietoa ja ottamalla siitä tietoa ulos. Tunnetuimpia Web 2.0 -teknologiaan perustuvia palveluita ovat mm. YouTube sekä Flickr, joilla on jo miljardeja käyttäjiä. (9.)

Varsinaisia ohjelmistotekniikoita, jotka ovat tunnusomaista Web 2.0 -sivustolle, ovat esimerkiksi Ajax, CSS-taitto ja selkeät URL-osoitteet. Ajax (= Asynchronous JavaScript and XML) on nimitys joukolle tuttuja web-teknologioita ja standardisoitunut työkalu vuorovaikutteisemman käyttöliittymän luontiin. CSS-taitolla määritellään sivustojen tyylimuotoilut erillisiin tiedostoihin. Kasvavien sivumäärien myötä URL-osoitteista pyritään tekemään entistä kuvaavampia ja selkeämpiä. Alla on yksi teoreettinen esimerkki mahdollisesta Web 2.0-osoiterivistä. (9.)

http://www.myprograms.net/photos/show/5

4.2 Grails-ohjelmistokehys

Grails on avoimeen lähdekoodiin perustuva, web-sovellusten toteuttamiseen tarkoitettu uuden ajan ohjelmistokehys. Se perustuu MVC-arkkitehtuuriin ja Java-virtuaalikoneessa ajettavaan sovellukseen. Grails käyttää ohjelmointikielenä dynaamista, Javaan perustuvaa Groovy-kieltä. Sitä on kehitetty Ruby on Railsin ja muiden vastaavien kehysten innoittamana vuodesta 2005 saakka. Nykyinen versio 2.03 on julkaistu 3.4.2012. Grailsin alkuperäinen kehittäjä G2One on siirtynyt sittemmin VMwaren omistamaan SpringSouce-kehitysosastoon. (10.)

Grails kehitettiin vastaamaan uudempien ja yhä monipuolisempien WWW-sivustojen tarpeisiin. Päämääränä oli toteuttaa korkean tuottavuuden ja nopean sovelluskehityksen ohjelmistokehys, joka automatisoi osan ohjelmakoodista. Tällöin ohjelmistokehittäjä voi paneutua haastavampiin ohjelmistoratkaisuihin ja resursseja ei kulu jatkuvasti perusasioihin. Grails-ohjelmistokehyksen ydinajatukset voidaan kiteyttää seuraavasti:

- tiedon toistamisen vähentäminen (Don´t Repeat Yourself)
- ohjelmistokehittäjän keskittyminen ainoastaan sovelluksen yksilöitäviin tekijöihin (Convention over Configuration)
- dynaamisen web-ohjelmointikielen, Groovyn tarjoamat mahdollisuudet
- oliotallennusrajapinnan hyödyntäminen ohjelmistokehityksessä. (Object Relational Mapping)
- tietylle toimintoalueelle ominainen ohjelmointi (Domain specific Language). (11.)

Grails sisältää myös valmiin HSQLDB-tietokannan, joka toimii välimuistissa. Sitä hyödyntämällä voidaan jättää varsinaisen ulkoisen tietokannan rakentaminen myöhemmäksi. Täytyy kuitenkin muistaa, että sisäänrakennettu tietokanta tyhjenee aina projektin uudelleenkäynnistyksen yhteydessä, joten se ei sovellu kovinkaan hyvin testaukseen. Tässä työssä käytin alusta asti erillistä tietokantaratkaisua, koska se kuului olennaisena osana sovellukseen. (11.)

4.2.1 Groovy-ohjelmointikieli

Aiemmin mainittu Groovy on dynaaminen olio-ohjelmointikieli, johon Grails-ohjelmistokehys pohjautuu. Se on saanut vaikutteita monista ohjelmointikielistä, kuten Ruby, Pearl, Python ja Java. Groovy toimii Java-alustalla ja pohjautuu Java-kielen syntaksiin, mutta sen muodollisuutta on kevennetty merkittävästi ja uusia ominaisuuksia on lisätty syntaksiin parantamaan tehokkuutta. Java-sovellukseen verrattuna kirjoitetun koodin määrä vähenee dramaattisesti. Seuraavasta esimerkistä nähdään, miten Groovyn edut tulevat hyvin esille kevennetyn syntaksin käytössä. Kumpikin koodirivi tuottaa saman lopputuloksen:

1) *List booksILike = new ArrayList();* (Java)
2) *def booksLike = []* (Groovy)

Groovyn vahvuuksiin kuuluu myös sen yhteensopivuus Java-virtuaalikoneen kanssa ja oma kääntäjä, joka tulkkaa ohjelmakoodin luokat JVM-tavukoodiksi. Tällöin Java-ohjelmointikielellä kirjoitetut luokat ovat integroitavissa Groovyyn ja päinvastoin. Kielet ovat keskenään jopa niin yhteensopivia, että esimerkiksi .java-tiedosto voidaan nimetä .groovy-tiedostoksi ohjelman suorituksen siitä häiriintymättä. (12, s. 1–24.)

Groovy-ohjelmointikielessä on myös useita muita etuja, kuten sen ohjelmointi-ominaisuuksiin kuuluvat sulkeumat ja optionaalinen tyypitys. Sulkeuma (engl. closure) on ensimmäisen luokan funktio, joka muistaa viiteympäristön, jossa se on määritetty. Tämä funktio voidaan palauttaa paluuarvona funktiolta tai välittää parametrina funktiolle. Seuraavassa on esimerkki sulkeuman käytöstä. (13, s. 1–7.)

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

Optionaalisessa tyypityksessä voidaan valita, käytetäänkö staattista vai dynaamista tyypitystä. Staattisessa mallissa muuttujan tyyppi on määriteltävä, ennen kuin siihen voidaan asettaa tietotyyppejä. Dynaamisessa mallissa muuttujat voidaan määritellä aivan kuin staattisessa tyypityksessä, mutta ne voidaan myös jättää määrittelemättä. Havainnollistetaan tätä esimerkin avulla:

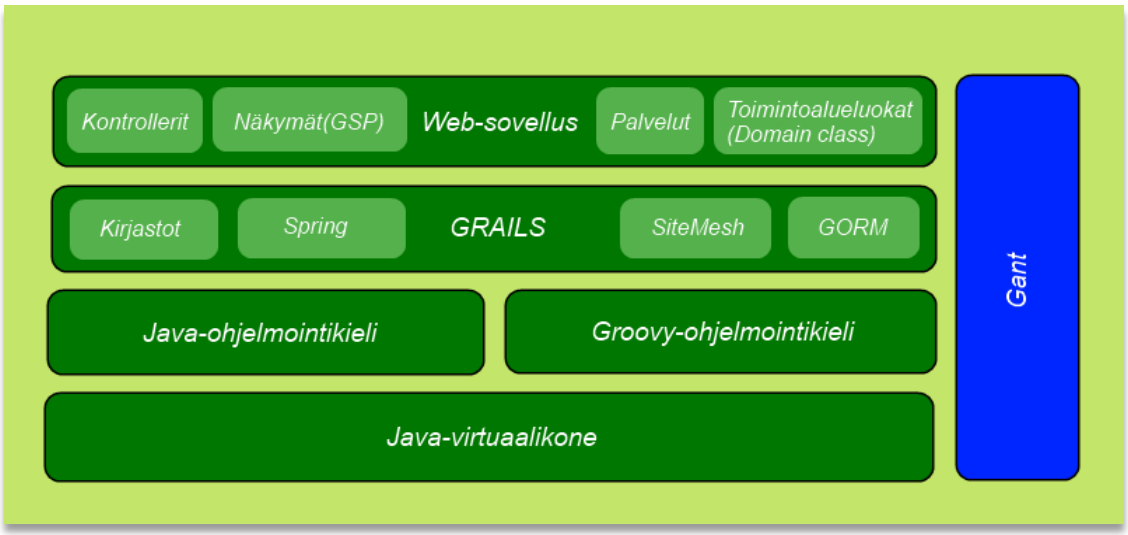
1) *String merkkijono = ”abcde”*
2) *merkkijono2 = ”fghij”*

Ensimmäinen esimerkkilause on staattisesti ja jälkimmäinen dynaamisesti tyypitetty merkkijono. Dynaamisessa tavassa muuttujan tyyppiä ei ole määritetty. Molemmat mallit ovat hyväksyttyjä Groovyssa. (12, s. 1–24.)

Yhtenä Groovy-ohjelmointikielen tärkeimpänä ominaisuutena pidetään metaolioprotokollaa (MOP), jonka avulla voidaan laajentaa valmiita luokkia ja lisätä niihin uusia toiminnallisuuksia. Tämä antaa mahdollisuuden taivuttaa ohjelmointikieltä ohjelmoijan tarpeita vastaavaksi. Metaolioprotokolla on myös keskeisessä asemassa Grails-ohjelmistokehyksessä, sillä se mahdollistaa Grailsin toimintoalueluokkien ominaisuudet. (12, s. 1–24.)

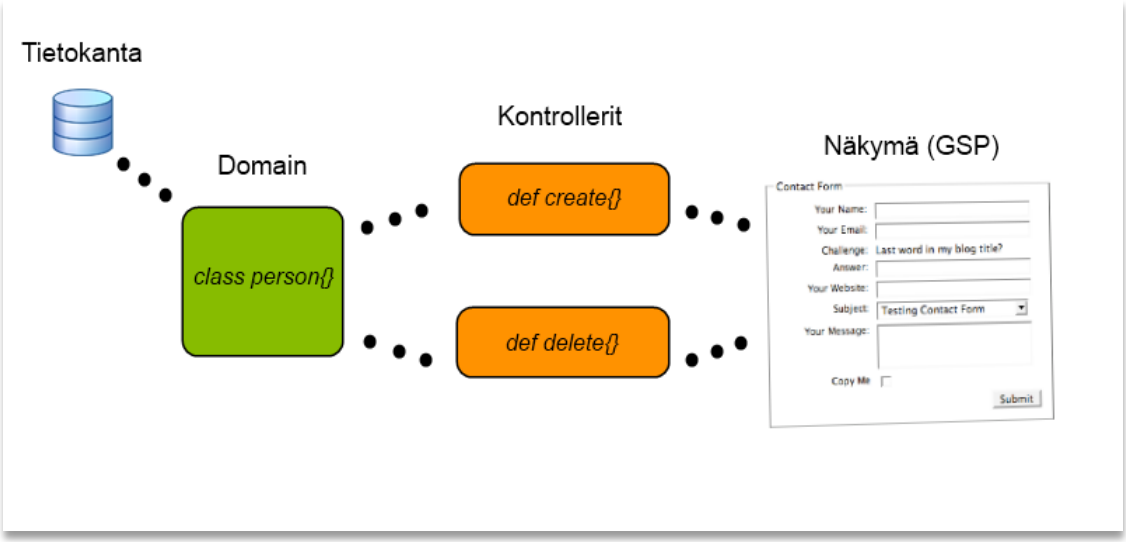
4.2.2 Arkkitehtuuri ja sovellusrakenne

Grails-ohjelmistokehys perustuu monikerrosarkkitehtuuriin, jossa alimpana tasona on Java-virtuaalikone (JVM). Seuraavassa kerroksessa ovat Grailsin ohjelmointikielet, Java ja Groovy. Näiden kerrosten yläpuolella ovat varsinainen ohjelmistokehyskerros sekä ylimpänä web-sovelluskerros, joka muodostaa rajapinnan WWW-ympäristöön. Näiden lisäksi Grails käyttää Gant-teknologiaa, joka automatisoi osan projektien tehtävistä. Arkkitehtuurin toteutus on esitetty osittain kuvassa 5. ja kokonaisuudessaan liitteessä 2. (12, s. 63–103.)



KUVA 5. Grails-ohjelmistokehyksen arkkitehtuuri.

Grails-sovellukset rakentuvat MVC-arkkitehtuurimallin mukaisesti. Malli perustuu kolmeen eri luokkaan, jossa M eli model edustaa domain-luokkaa. Domain-luokka on olio, joka pitää sisällään sovelluksen datan ja antaa sen kontrollerin käyttöön. Kontrollerit (= controllers) ovat Grails-sovelluksen selkäranka. Niiden avulla käsitellään dataa, ohjataan sitä domainista näkymille ja päinvastoin. Kontrollereilla on myös tärkeä tietoturvamerkitys, koska mahdollinen arkaluontoinen data käsitellään niissä eikä suoraan näkymissä. Näkymät (= views) toimivat sovelluksen käyttäjä-rajapinnassa. Ne näyttävät kaiken annetun informaation ja mahdollistavat datan syötön, jota ohjataan eteenpäin kontrollereissa. Kuvassa 6 nähdään Grails-sovelluksen MVC-arkkitehtuurimallin mukainen ajonaikainen suoritus. (12, s. 63–103.)



KUVA 6. Grails-sovelluksen ajonaikainen suoritus

Grails käyttää näkymien muodostamisessa erityisiä GSP-tiedostoja (Groovy Server Pages) (11). GSP-luokka voidaan joko renderöidä automaattisesti tai manuaalisesti kontrollerin puolelta:

render(view: "index")

GSP-teknologia on samankaltainen kuin Javassa käytettävä JSP (Java Server Pages). Teknologian periaatteena on tuoda HTML-pohjaisiin sivuihin logiikkaa ja tehdä niistä dynaamisia omien tag-kenttien avulla. (12, s. 63–103.)

4.2.3 GORM

Domain-luokat ovat minkä tahansa kaupallisen web-sovelluksen sydän. GORM on Grails-ohjelmistokehyksessä toteutettu ORM eli Object Relational Mapping-teknologia. Se on yksi tärkeimmistä Grailsin ominaisuuksista, koska sen avulla voidaan kuvata domain-luokkia tietokannassa. Se käyttää hyvin joustavaa ja suosittua Hibernate3-relaatiotyökalua, joka on suunniteltu käytettäväksi Java-pohjaisilla alustoilla (14).

GORMissa tuodaan relaatiotietokantojen taulut suoraan ohjelman käyttöön ja niitä mallinnetaan dataolioilla. Toisin sanoen Grails osaa rakentaa tietokannan taulut sekä niiden sarakkeet dataolioiden avulla. Ohjelmistosuunnittelija voi luoda tietokannan sisällön ja tietokantayhteydet suoraan olioilla koskematta varsinaiseen tietokantaan mitenkään (14).

Hibernate on istunto, jossa on sidottu Grailsin tekemät pyynnöt dataolioiden käsittelyyn liittyen. Jokainen istunto on aina välimuistissa ja kaikki oliolle pyydetyt toimenpiteet suoritetaan istunnon päätteeksi. Tällöin vältytään jatkuvalta liikenteeltä tietokantaan päin. (12, s. 63–103.)

Grails-komentokehote on yksi tapa luoda kätevästi uusia domain-luokkia. Domain-luokka on aina mallinnus relaatiotietokannan taulusta ja se voidaan luoda seuraavalla tavalla:

```
grails create-domain-class org.bookstore.Book
```

Tietokantaan on nyt luotu taulu Book ja Grails-projektiin domain mallintuu .groovy-tiedostoon seuraavasti:

```
package org.bookstore

class Book {

}
```

Tämän jälkeen taulun ominaisuuksia voidaan määritellä Java-kielen tyyppisesti antamalla haluttuja arvoja esimerkiksi näin:

```
class Book {
```

```
    String title
    Date releaseDate
}
```

Jokainen annettu ominaisuus on tietokannan taulun yksi tietue. SQL-tyypit tunnistetaan automaattisesti Java-ohjelmointikielen tyypeistä, mutta niitä voidaan muokata DSL:n (Domain Specific Language) avulla. Tietokantayhteydet voidaan niin ikään rakentaa suoraan GORMia hyödyntämällä. Esimerkissä luodaan taulut Face ja Nose ja niihin tietokantayhteyden seuraavasti:

```
class Face {
    Nose nose
}
class Nose {
    static belongsTo = [face:Face]
}
```

Kuten esimerkistä voidaan helposti päätellä, Face on ”äititaulu” ja Nose ”lapsitaulu” ja yhteytenä käytetään yksi-moneen-yhteyttä. Yhteydessä yhden taulukon tietue on yhteydessä toisen taulukon useaan tietueeseen, mutta toisen taulukon tietueet ovat yhteydessä vain ensimmäisen taulukon yhteen tietueeseen.

Grails-ohjelmistokehykseen on lisätty myös muita GORMiin liittyviä toimintoja helpottamaan suunnittelijoiden työtä. Yksi näistä on CRUD-teknologia, jossa on valmiit perusfunktiot dataolioiden luontiin, lukemiseen, päivittämiseen ja poistamiseen. Näitä funktioita voidaan käyttää suoraan tai niitä voidaan myös muokata itselle sopivaksi. Grails renderöi automaattisesti nämä funktiot myös näkymiin, jolloin tietokannan käsittelyn perustoimintoihin ei vaadita enää suunnittelijan työaikaa. CRUD-teknologia on kuvattu tarkemmin liitteessä 7. (14.)

4.2.4 Liitännäiset

Grailsissa on web-kehittäjille lukuisia valmiiksi toteutettuja ja testattuja liitännäisiä. Liitännäisten avulla sovellukset ovat helposti ja nopeasti laajennettavissa eikä työaikaa kulu testaukseen. Ne tarjoavat myös valmiita ratkaisuja moniin ongelmiin. Liitännäisiä käytetään irrallisina osina sovellusta, jolloin ne voidaan

yksitellen asentaa tai poistaa sovelluksesta. Tällä hetkellä Grailsiin on saatavana yli 700 liitännäistä tai modulaarista osaa. (11.)

Liitännäisiä voidaan asentaa Grailsiin joko komentokehotteella automaattisesti tai päivittämällä Plugin-lista Grails-sivustolta ja asentamalla ne manuaalisesti. Komentokehotteella asennettaessa tarvitaan tietää vain liitännäisen nimi ja Grails huolehtii lopusta:

grails install-plugin auto-test

Liitännäistä ei koskaan asenneta Grails-projektin tiedostopuuhun vaan sille luodaan automaattisesti kansio erityiseen plugin-tiedostopuuhun. Liitännäisten tiedostopuu sijaitsee eri paikassa kuin itse pääprojekti. Liitännäisiä voidaan myös luoda itse.

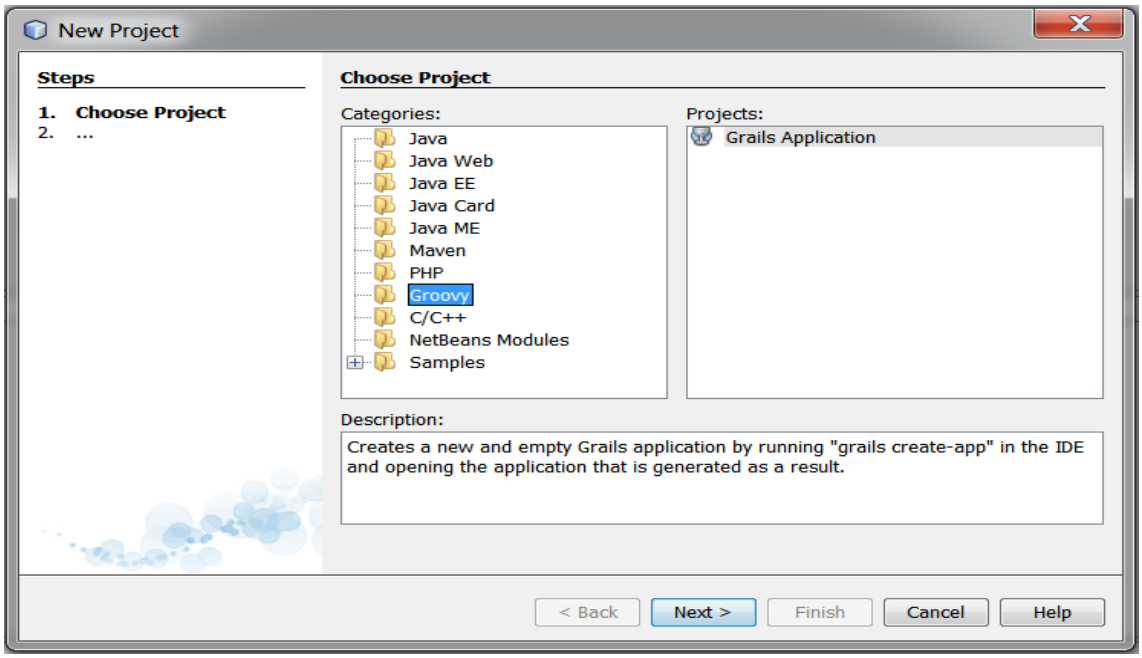
4.3 Netbeans-kehitysalusta ja Grails

Netbeans IDE on avoimen lähdekoodin ohjelmointialusta PHP-, C-, C++-, Java-, Ruby-, Python-, JavaScript- ja Groovy-ohjelmointikielille. Netbeansin avulla voidaan luoda ammattimaisia sovelluksia mobiilialustoille, WWW-ympäristöön sekä työpöytäkäyttöön. Ohjelmointialusta on toteutettu Java-ohjelmointikielellä ja se tarvitsee toimiakseen Javan ajoympäristön (JRE). Viimeisin vakaa versio ohjelmasta on 7.1.1 ja se on julkaistu 2012 (15).

Netbeans tukee valmiina useita ohjelmistokehyksiä, kuten Zend ja Symphony PHP-ohjelmointikielelle ja Grails Groovy-ohjelmointikielelle. Netbeans ymmärtää Grails-projektin rakenteen ja elinkaaren erittäin hyvin ja soveltuu siksi hyvin sille kehitysalustaksi (15). Olen käyttänyt Netbeansia myös aikaisemmissa web-projekteissa, joten se oli luonnollinen valinta myös opinnäytetyöhön

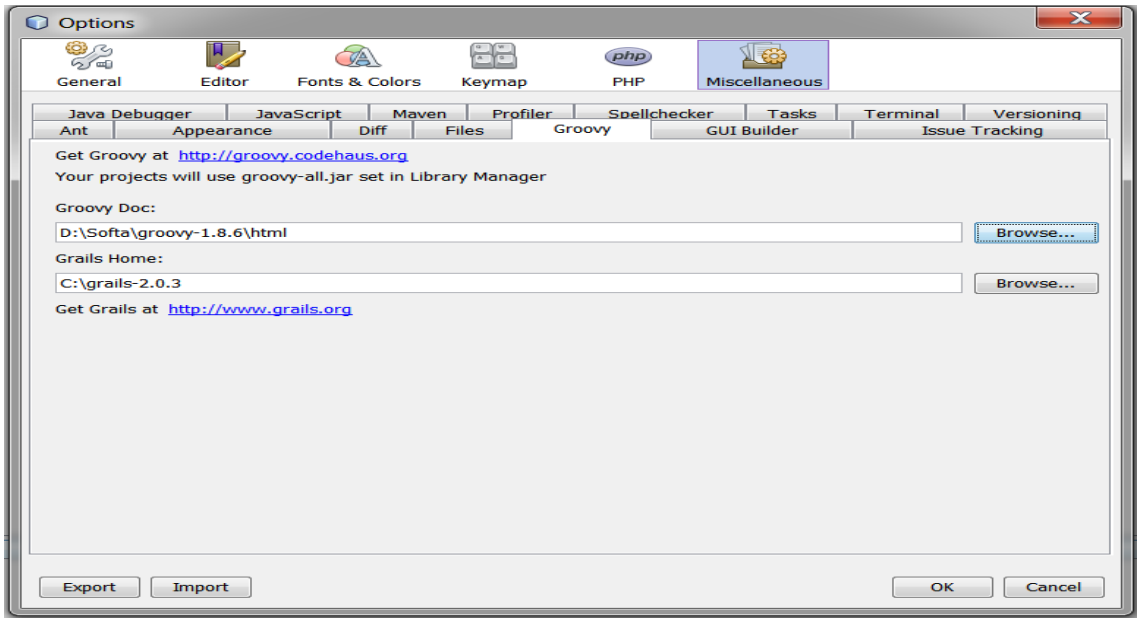
4.3.1 Asennus ja käyttöönotto

Grails-projektin perustaminen Netbeansissa on helppoa. Kun ohjelmointikieleksi valitaan Groovy, Netbeans antaa valmiiksi vaihtoehdoksi Grails-projektin. Kuvassa 7 on havainnollistettu projektin perustaminen ja alkuasetukset kehitysympäristössä.



KUVA 7. Grails-projektin perustaminen Netbeans-kehitysalustaan

Grails-ohjelmistokehyspaketti ladataan Grailsin verkkosivuilta grails.org ja pakattu tiedosto puretaan haluttuun paikkaan. Groovy-ohjelmointikielen kirjastot saa ladattua osoitteesta http://groovy.codehaus.org. Tämän jälkeen nämä osat otetaan käyttöön Netbeans-ympäristössä kuvan 8 tapaan.

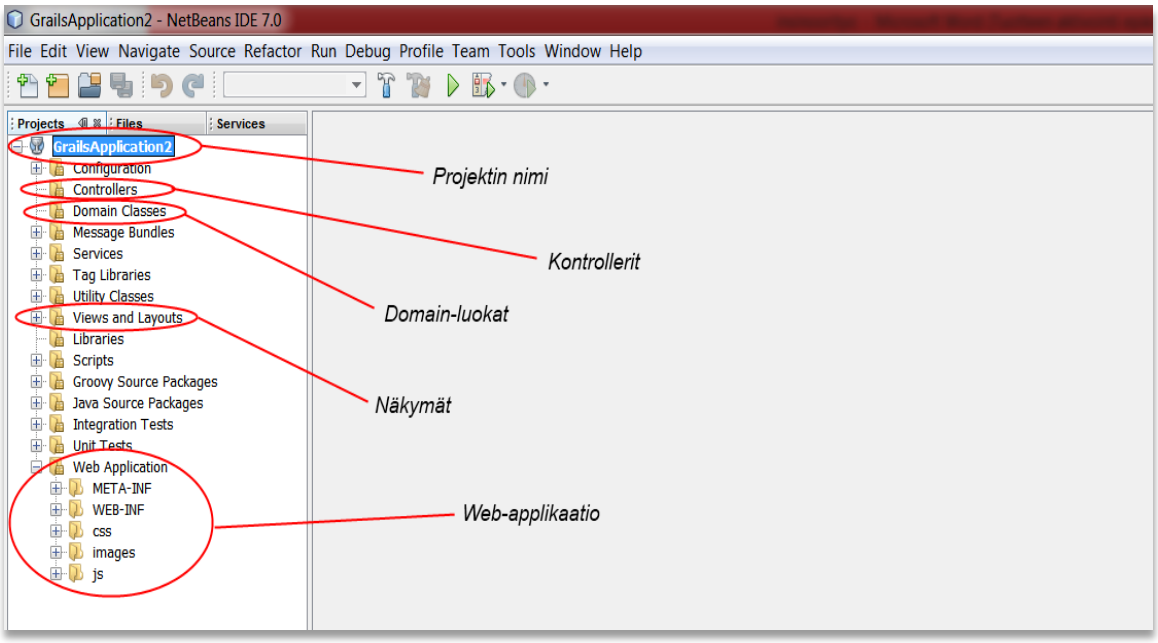


KUVA 8. Grails-kehyksen ja Groovyn käyttöönotto

Grails on tämän jälkeen integroitu Netbeans-kehitysympäristöön ja se on käyttövalmis. Uudempien versioiden käyttöönotto tapahtuu samalla tavoin. Ladataan ja puretaan Grails-kehityspaketti sekä Groovy-dokumentti haluttuun paikkaan ja määritetään tiedostopolut Netbeans-valikossa.

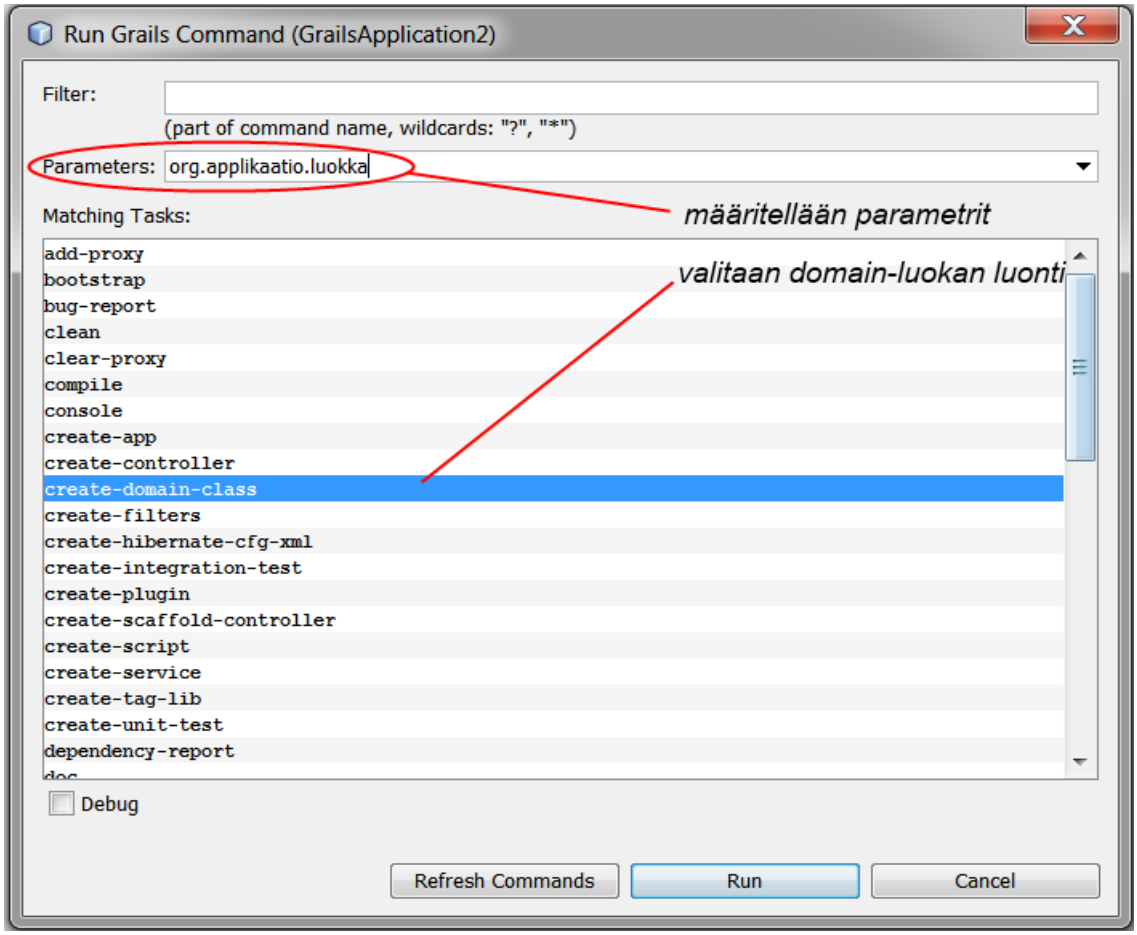
4.3.2 Hallinta

Grails-projekti on helposti hallittavissa Netbeans-tiedostonäkymien avulla. Koko projektin tiedostopuu tuodaan näkyville ja sitä voidaan helposti käsitellä kansioittain. Projekti jakaantuu varsinaiseen aplikaatio-osaan, johon kuuluvat kaikki ohjelmalliset toiminnot sekä web-applikaatioon, joka sisältää kaikki projektin XML-, CSS- ja Javascript-tiedostot sekä mahdolliset kuvat (kuva 9).



KUVA 9. Grails-projektin tiedostopuu

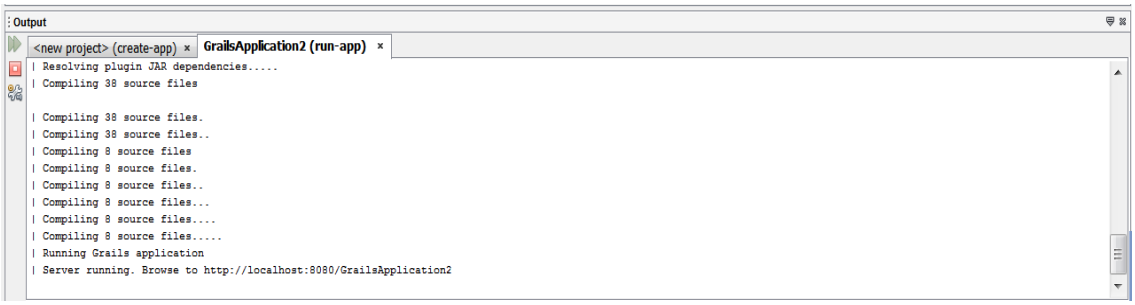
Kuten jo aikaisemmin kävi ilmi, Grails sisältää useita hyödyllisiä ominaisuuksia ja nämä ominaisuudet tulevat mukana myös Netbeans-kehitysympäristöön. Ohjelmoijalla on mahdollisuus valita, käyttääkö hän graafista tapaa vai komentokehotetta. Opinnäytetyössäni suosin komentokehotteen käyttöä, koska sitä suositeltiin myös Grails-sivustolla ja se tuntui toimivalta sekä helpolta käyttää (kuva 10).



KUVA 10. Domain-luokan luonti komentokehotteen avulla

Komentorivi voidaan aina käynnistää suoraan Grails-projektin hakemistopuusta ja siinä on mukana kaikki valmiit Grails-komennot. Liitännäiset voidaan myös asentaa suoraan komentoriviltä, jos tiedetään asennettavan liitännäisen nimi. Nämä kaikki ominaisuudet ovat toteutettavissa myös graafisen käyttöliittymän avulla.

Grails-sovellus käynnistetään myös valinnanvaraisesti joko komentokehotteesta tai Netbeans-ympäristöstä. Ajosuoritus ja kääntäjän viestit näytetään omassa ikkunassa (kuva 11) ja jos virheitä ei tule, ohjelma käynnistyy Internet-selaimeen. Grailsin sisälle rakennettu Tomcat-palvelin huolehtii sovelluksen suorittamisesta ja SiteMesh-sovelluskehys rakentaa projektin HTML-muotoon, jotta selaimet voivat tulkita sitä. (14.)



KUVA 11. Grails-sovelluksen käännös ja ajo

4.4 WWW-palvelinympäristö

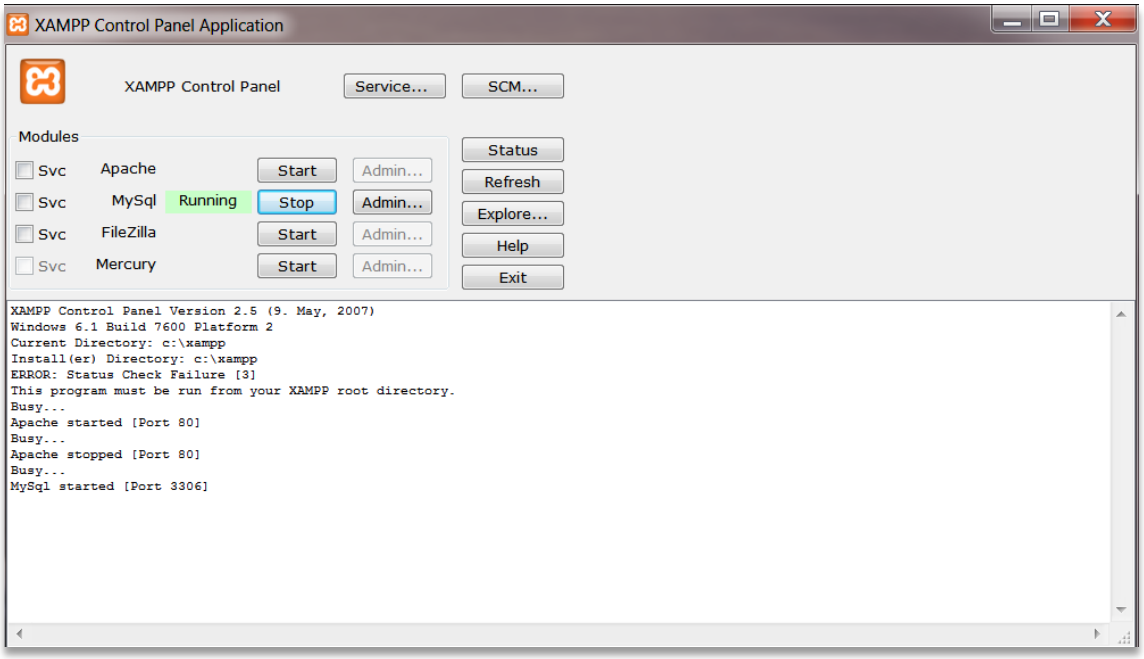
WWW-palvelinympäristö koostuu tiivistettynä web- ja tietokantapalvelimesta sekä ohjelmointikielestä. Palvelinympäristö voidaan toteuttaa sekä paikallisena että etänä. Grails-ohjelmistokehyspaketti sisältää sisäisen valmiin palvelinympäristön ja Tomcat-palvelimen, jossa sovellusta voidaan suorittaa. Sisäinen tietokanta huolehtii datan väliaikaisesta säilytyksestä.

Opinnäytetyössäni päädyin käyttämään erillistä palvelinympäristöä lähinnä tietokantaratkaisun vuoksi. Grails-sovellusta ajettaisiin paikallisena ja sen taustalla pyörisi erillinen MySQL-palvelin. Lisäksi oli tärkeää, että valmis sovellus voidaan paketoida ja ajaa Tilaajan koneella omalla palvelimella. Varsinaiseen WWW-ympäristöön sovellus siirrettäisiin mahdollisessa julkaisuversiossa.

4.4.1 XAMPP-palvelinpaketti

XAMPP on avoimen lähdekoodin ilmainen palvelinpaketti, joka sisältää Apachen HTTP-palvelimen, MySQL-tietokantapalvelimen sekä PHP- ja Perl-ohjelmointikielten tuen. XAMPP on suunniteltu pääasiassa web-ohjelmoijille testialustaksi ja se on saatavana kaikkiin yleisimpiin käyttöjärjestelmiin. (16.)

XAMPP-asennuspaketin mukana tulee kehittäjälle tärkeä ohjauspaneeli, jonka avulla voidaan käynnistää haluttuja palveluita. Tässä työssä tarvitsin ainoastaan MySQL-palvelinta, joten muut palvelut saivat olla suljettuna. Kuvassa 12 on havainnollistettu XAMPP-ohjauspaneelin toimintaa.



KUVA 12. XAMPP-ohjauspaneeli ja käynnistetty MySQL-palvelin

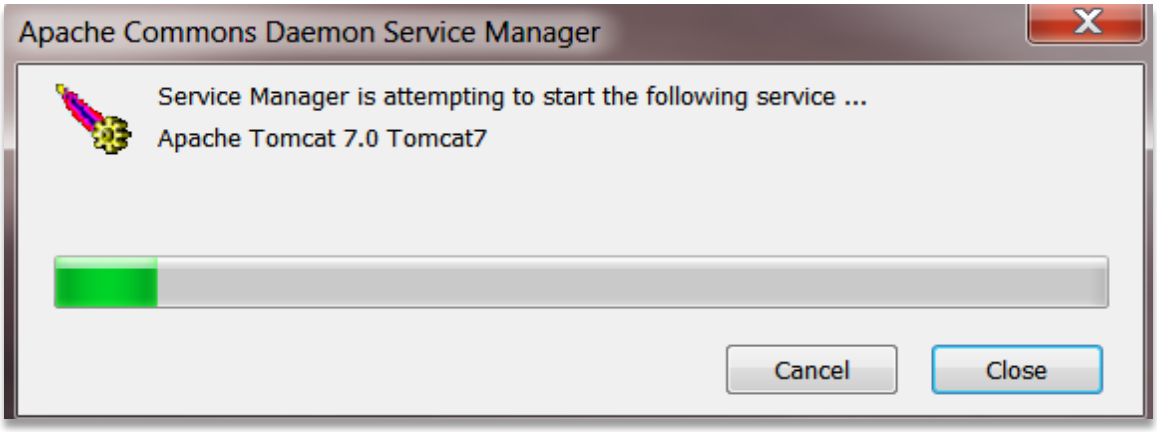
XAMPPin käyttö on suositeltavaa monestakin syystä. Ehkä suurin syy on, että kannattaa tehdä virheet omalla koneella, ennen kuin varsinaiset sivustot ovat todellisessa WWW-ympäristössä kaikkien nähtävillä. Lisäksi päästään eroon FTP:n käytöstä. Kaikkia pieniä muutoksia ei tarvitse siirtää erikseen FTP:llä ja testata vaan ne voidaan testata jo valmiiksi ennen siirtoa. (17.)

4.4.2 Tomcat-palvelin

Tomcat-palvelinohjelmisto on avoimen lähdekoodin JavaServer Pages ja Servlet-säiliö. Tomcat kuuluu Apachen tuoteperheeseen ja sen avulla voidaan ajaa Java- ja Groovy-pohjaisia sovelluksia. Tomcat vaatii toimiakseen Javan ajoympäristön (JRE). (18.)

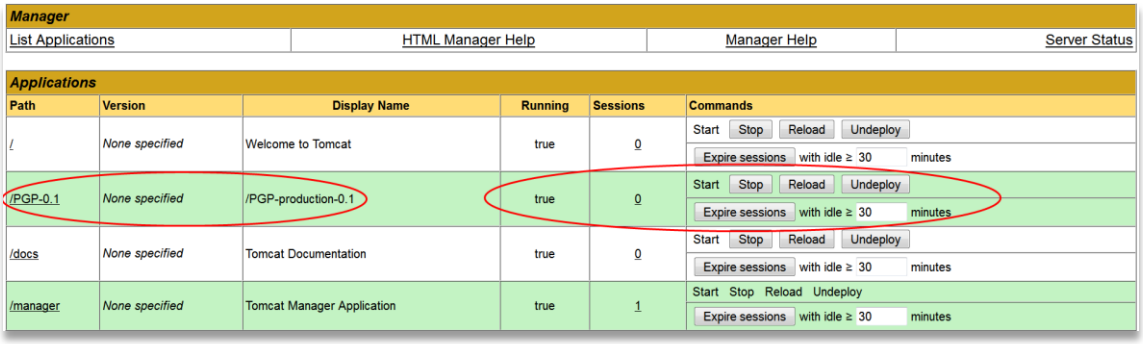
Tämän työn yhtenä osana oli kokeilla valmiin Grails-sovelluksen ajamista Tomcat-palvelimella. Päädyin käyttämään Tomcat 7.0 -versiota, joka on viimeisin julkaistu versio. Tomcatin asennus Windows-käyttöjärjestelmään onnistuu samoin kuin mikä tahansa ohjelma: ladataan valmis asennuspaketti Tomcatin WWW-sivuilta ja asennetaan ohjelma. Asennuksen keskellä määritellään mahdolliset käyttäjäoikeudet palvelimelle sekä Java-ajoympäristön tiedostopolku. Asennuksen jälkeen Tomcat on käyttövalmis. Tomcat-palvelinta hallitaan oman

ohjauspaneelin avulla. Kuvassa 13 on esitetty Tomcatin käynnistäminen ohjauspaneelista.



KUVA 13 Tomcat-palvelin käynnistyy

Tomcatin käynnistyttyä sitä voidaan hallita Web-käyttöliittymän avulla. Käyttöliittymän etusivulla on useita hallintaominaisuuksia, mutta tämän työn kannalta olennaisin tekijä oli päästä Tomcat Web Application -manageriin. Palvelu löytyy linkin takaa ja siihen vaaditaan mahdollinen asennusvaiheessa syötetty käyttäjätunnus ja salasana. Tällä sivulla hallitaan kaikkia palvelimen web-sovelluksia. Valmis Grails-sovellus paketoidaan .war-tiedostoksi (= Web Application Archive) ja lähetetään Web Application -managerin avulla palvelimelle. Tomcat tunnistaa tiedostomuodon ja kääntää sen Internet-selaimen ymmärtämäksi WWW-sivustoksi. Tämän jälkeen sovellusta voidaan hallita managerin avulla. Se voidaan käynnistää, sammuttaa, poistaa tai uudelleenkäynnistää. Kuvassa 14 sovellus on käynnistetty ja sitä ajetaan selaimessa paikallisena.



KUVA 14. Web-sovelluksen ajo Tomcat-palvelimella

4.5 Merkintäkielet ja toteutustekniikat

Ohjelmoitaessa mitä tahansa Web-sovellusta ei voida välttyä WWW-sivujen toteuttamiseen tarvittavista merkintäkielistä ja erilaisista toteutustekniikoista. Merkintäkielen avulla tieto esitetään loogisesti ja helposti käsiteltävässä muodossa. Merkkauksessa esitettävään tietoon lisätään erilaisia merkkejä. Internet-selaimet tulkitsevat näitä merkkejä ja näyttävät tiedon halutulla tavalla. (20.)

Tunnetuin WWW-sivuilla käytetty merkintäkieli on HTML. Kieli sisältää erilaisia elementtejä ja ominaisuuksia, joiden avulla sivujen ulkoasu määritellään. Monesti kuullaan puhuttavan myös XML-kielestä, joka on sen sijaan metakieli. Siitä johdetaan uusia merkintäkieliä, kuten XHTML (= laajennettu HTML). XHTML-kielessä on standardisoidut, tiukat säännöt, joita tulee noudattaa. Tällöin kaikki selaimet näyttävät WWW-sivut samalla tavalla eikä tulkinnassa synny ongelmia. (19.)

4.5.1 HTML5

HTML5 on uusin versio perinteisestä HTML-kielestä ja siitä odotetaan standardia kieltä kaikille WWW-sivuille vielä tämän vuosikymmenen loppupuolella. Yhä kasvava erilaisten mobiilialustojen tuotanto ohjelmistopuolella on johtanut HTML-kielen kehitykseen. HTML5 poistaa tarpeen useiden eri sovellusten kehitykselle ja ylläpidolle. Tavoitteena on tehdä merkintäkielestä myös yhä selkeämpää ja lisätä siihen valmiita elementtejä WWW-sivujen rikastuttamiseksi. Nykyään käytössä olevasta Flash-tekniikasta ollaan luopumassa ja samat ominaisuudet halutaan kiinteäksi osaksi Internet-selaimia. (20.)

Sovelluskehittäjät ja Web-suunnittelijat huomaavat ensimmäisen uudistuksen hypertekstidokumentin määrittelyssä, joka hoidetaan HTML5-kielessä kahdella sanalla aikaisemman kahden rivin sijaan.

Vanha tapa:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
                "http://www.w3.org/TR/html4/strict.dtd">
```

Uusi tapa:

<!doctype html>

HTML5-kielessä voidaan myös nimetä sivujen alueita ilman -tageja esimerkiksi tähän tapaan:

Vanha tapa:

* *
* *
* *

Uusi tapa:

<section>Sisältöä...</section>
<article>Artikkeliosa</article>
<footer>Alaosa, vaikka copyright</footer>

HTML5-tiedostoon täytyy liittää Javascript-tiedosto ja tyylitiedostoon täytyy kirjoittaa kyseisiä tageja vastaavat määritykset, jotta sivut näkyvät selaimessa oikein. HTML5-kielessä on myös lukuisia muita ominaisuuksia, mutta kaikkia uusia tapoja ei ole pakollista käyttää. WWW-sivu luokitellaan HTML5-kieltä käyttäväksi, jos se käyttää edes dokumentin määrittelyä uudella merkintätavalla. Ylen uutiset (http://yle.fi/uutiset/) on yksi useista WWW-sivustoista, joka käyttää HTML5-tekniikkaa. (21.)

4.5.2 CSS3

CSS ei ole varsinaisesti ohjelmointikieli vaan mieluummin WWW-sivuille kehitetty tyyliohjeiden laji. Aiemmin tyyliohjeet on määritelty suoraan HTML-tiedostossa, mutta kasvava datamäärä ja sivujen rikkaampi sisältö ovat johtaneet erilliseen tyylitiedoston kehitykseen. CSS-tiedostolla annetut säännöt ehdottavat selaimelle, kuinka sivu tulisi esittää. Ongelmaksi on muodostunut kuitenkin selainten rajoittuneisuus tyylitiedostojen suhteen. (22.)

CSS:n syntaksi on hyvin selkeä. Annetaan vain HTML-tiedoston valitsin ja määritellään sille haluttu arvo ja ominaisuus. Esimerkissä määritellään HTML-sivun taustaväri mustaksi:

body {backround-color: #000000;}

Toisin sanoen HTML-tiedostossa määritellään halutun alueen nimi ja tämän alueen tyylit määritellään puolestaan CSS-tiedostossa.

CSS3 on uusin kehitysversio tyylilajeista ja suosituimmat Internet-selaimet, kuten esimerkiksi Mozilla Firefox, Google Chrome ja Apple Safari osaavat hyödyntää jo sen ominaisuuksia. CSS3-versioon on kytketty lisää ominaisuuksia ja tärkeimmät uutuudet voidaan luetella seuraavasti:

- ominaisuudet voi jakaa moduleihin
- pyöreät kulmat sekä varjostukset
- erilaiset esitystasot (WWW-sivu vs. kalvo)
- uusia valitsimia
- lisäominaisuuksia tekstin ja ulkoasujen muotoiluun
- tapahtumankäsittely
- Speech-media tyypin uusinta ja ääniefektit. (23.)

Tässä opinnäytetyössä päätin raapaista hieman pintaa HTML5- ja CSS3-tekniikoista. Seuraavassa on esimerkki, miten määrittelin ajan mukaan vaihtuvan värin kun kohdistetaan hiiri haluttuun linkkiin.

```
#wrapper1 {
        -webkit-transition: background-color 0.5s linear;
        -moz-transition: background-color 0.5s linear;
        -o-transition: background-color 0.5s linear;
        transition: background-color 0.5s linear;
}
#wrapper1:hover {
        background-color: #009B00;
}
```

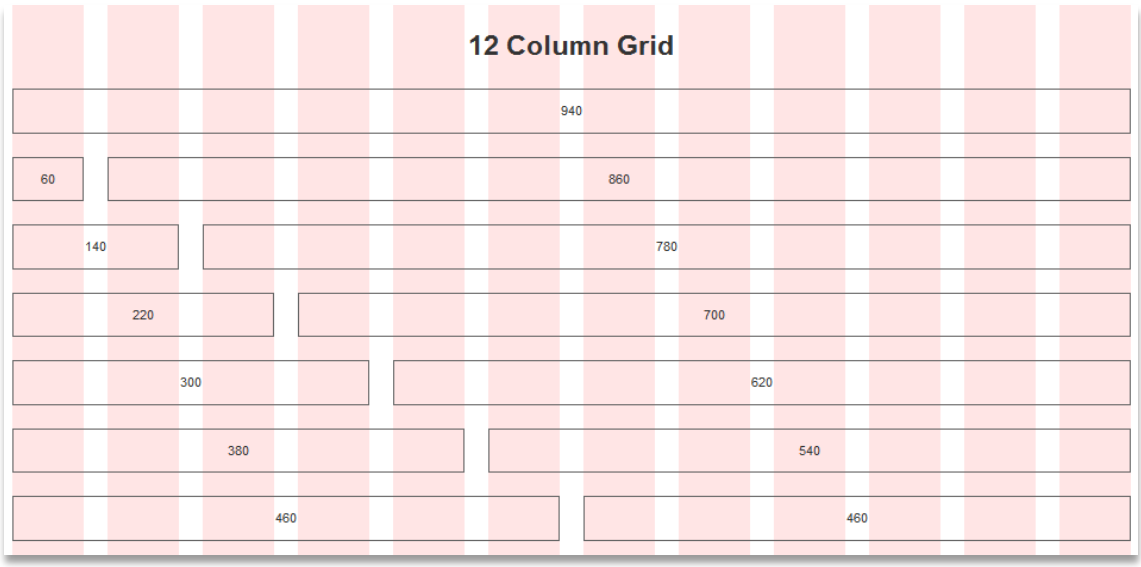
On huomattava, että eri selaimille pitää välittää omat tyylipaketit, koska selaimet on ohjelmoitu hieman eri tavoin. CSS3:n ja HTML5-tekniikoiden avulla pyritään keventämään WWW-sivujen toteutustapoja ja vähentämällä liitännäisiä kuten erilliset mediasoittimet.

4.5.3 960 Grid System

960 Grid System on CSS-kehys, joka on tarkoitettu helpottamaan WWW-sivujen asettelua. Se on saavuttanut suurta suosiota web-suunnittelijoiden keskuudessa ja se soveltuu hyvin sivuprototyyppien kehitykseen tai valmiiseen tuotantoympäristöön. (24.)

Monipalstaisten Web-sivujen rakentaminen on tuottanut aikaisemmin päänvaivaa suunnittelijoille. Palstanleveyksiä on käytännössä joutunut laskemaan käsin, mikä on ollut erittäin työläs työvaihe varsinkin monimutkaisten sivulayoutien suunnittelussa. Nimensä mukaisesti kehys perustuu 960 pikselin sivuleveyteen, jota voidaan käyttää 12-, 16-, tai 24-palstaisena versiona (kuva 15). Pikselimäärä perustuu helppoon jaollisuuteen kokonaisluvuilla sekä nykyajan näyttöjen resoluutiokokoon. Epätarkimmassakin näytössä sivualue pysyy näytön reunojen sisällä. (24.)

Työkalun huonona puolena voidaan pitää WWW-sivujen staattisuutta, jolloin sivurakenne pysyy samana ja sivuista voi tulla liian kaavamaisia. Toinen hyvin selkeä heikkous on sidonnaisuus 960 pikseliin. Jatkuva resoluutioiden kasvu ja näyttöjen suuret erot hankaloittavat oikean sivuleveyden määrittämistä. Kaiken kaikkiaan työkalu on kuitenkin oiva apuväline web-suunnitteluun ja lisäksi sen käyttö on vapaata. (24.)



KUVA 15. 960 Grid System-kehyksen 12-palstainen layout (25.)

5 SUUNNITTELU JA TOTEUTUS

Harjoituskirjaamo-projekti käynnistyi alkupalaverin jälkeen marraskuussa 2011. Projekti jaettiin karkeasti suunnittelu- ja toteutusvaiheeseen. Suunnitteluvaiheeseen kuului ohjelmistotyökalujen valinta, ohjelmiston vaatimusmäärittely, esitutkimus ja kaikki projektin käynnistykseen liittyvä dokumentointi. Toteutusvaiheeseen kuului arkkitehtuurisuunnittelu, projektin hallinta sekä varsinainen ohjelmointityö.

Suunnitteluvaiheessa pidettiin useita välipalavereita, joissa haettiin yhteistä linjaa Web-sovelluksen suhteen. Tilaajan tuli selvittää, mitä ominaisuuksia sovelluksessa tulisi olla ja mihin tarkoitukseen se tulisi varsinaisesti olemaan. Ohjelmistosuunnittelijan tehtävänä oli tehdä taustatyötä ja tutustua hieman harrastustoimintaan, johon sovellusta suunniteltiin. Suunnitteluvaiheen tuloksena saatiin aikaan esitutkimusdokumentti, projektisopimus ja -suunnitelma sekä vaatimusmäärittely.

Varsinainen sovelluksen toteutusvaihe aloitettiin helmikuussa 2012. Toteutusvaiheen ensimmäisenä tehtävänä oli suunnitella sovelluksen arkkitehtuuri, joka hyväksyttäisiin yhteisessä palaverissa ennen seuraavaa vaihetta. Varsinainen ohjelmointityö jaettiin kahteen sprinttiin, joista ensimmäisessä ohjelmoitiin lähes koko sovelluksen toiminta ja toisessa käyttöliittymä.

5.1 Vaatimusmäärittely

Vaatimusmäärittely on ohjelmistosuunnittelun kivijalka, johon koko ohjelmisto pohjautuu. Vaatimusmäärittelyssä kuvataan ohjelmistoprojektin tavoitteita ja vaatimuksia. Siinä määritellään, miten ohjelmiston tulisi toimia sekä millä keinoilla toiminnallisuudet saavutetaan. Vaatimusmäärittely koostuu toiminnallisista ja ei-toiminnallisista vaatimuksista. Toiminnalliset vaatimukset kuvaavat ohjelmiston ominaisuuksia ja sen toimintaa käyttäjän kannalta. Ei-toiminnalliset vaatimukset voidaan jaotella vielä tarkemmin resurssi- ja laatuvaatimuksiin. Laadun määritelmiä voivat olla esimerkiksi käytettävyys, suorituskyky, turvallisuus, siirrettävyys ja ylläpidettävyys. Resurssit kuvastavat sen sijaan aikaa ja rahaa. Vaatimusmäärittely toteutetaan yhdessä asiakkaan kanssa. (26.)

5.1.1 Toiminnalliset vaatimukset

Harjoituskirjaamo-sovellus haluttiin tilata WWW-pohjaisena sen laajan käyttäjäkunnan ja käyttötarkoituksen vuoksi. Ohjelman varsinaisia käyttäjiä ja asiakkaita ovat ampumaseurat, ampuma-asekouluttajat sekä ampumaharrastajat eli peruskäyttäjät. Käyttäjät on jaettu ohjelmassa samannimisiin käyttäjärooleihin.

Ohjelman avulla voidaan suorittaa seuraavia toimintoja:

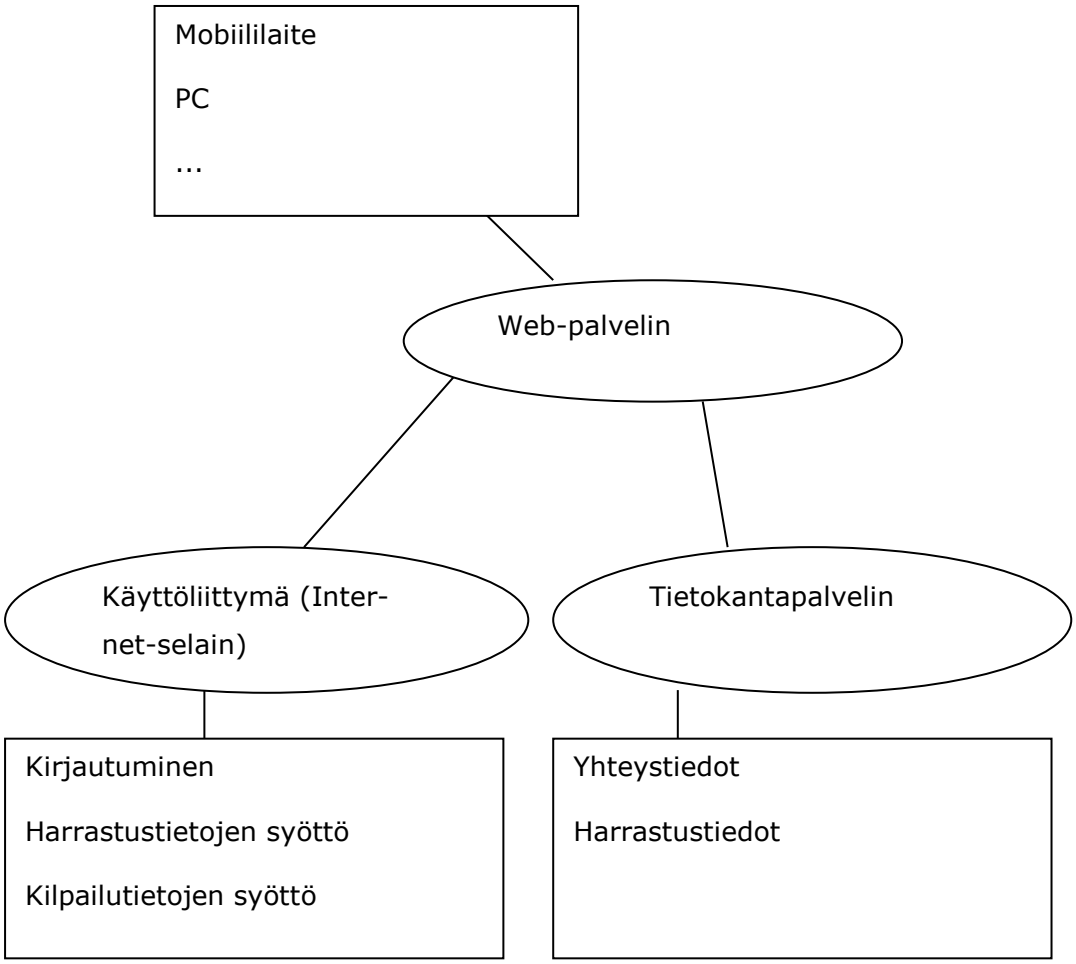
- Järjestelmässä on oltava neljä käyttäjätasotiliä (järjestelmähallitsija, ampumaseura, kouluttaja ja peruskäyttäjä).
- Järjestelmä on ns. suljettu järjestelmä, jolloin kaikki toiminnot ja käyttäjätilit ovat suojattu käyttäjätunnuksella ja salasanalla.
- Järjestelmänhallitsija pystyy luomaan ampumaseuratilejä ja ampumaseurat voivat luoda kouluttaja- sekä käyttäjätilejä
- Uudella käyttäjällä on myös mahdollisuus itse rekisteröityä palveluun.
- Käyttäjätilin salasanan tulee olla kryptattu.
- Peruskäyttäjä voi syöttää ohjelmaan omat harrastus- ja kilpailutiedot jokaisesta ammunnasta.
- Kouluttaja voi seurata harrastajan aktiivisuutta lukemalla hänen harrastus- ja kilpailutietoja.
- Kouluttaja voi kirjoittaa peruskäyttäjälle todistuksen harrastusaktiivisuudesta.
- Ampumaseurat voivat kirjata omia ampumaratoja tietoineen, jotka ovat käyttäjän valittavissa harjoitustapahtumassa.
- Kaikilla käyttäjätileillä on mahdollisuus muokata omia tietojaan, jotka päivittyvät automaattisesti myös muiden nähtäväksi.

Toiminnallisiin vaatimuksiin kuului yhtenä asiakasvaatimuksena myös harrastajan sijainnin paikannus ampumaharjoitustilanteessa. Tällöin harrastaja varmistaisi esimerkiksi matkapuhelimen avulla sijaintinsa ja voitaisiin todentaa, että harrastaja todellakin on käynyt radalla, jonka hän on harjoitustietoihin merkinnyt. Toteutustapaa ja tekniikkaa pohdittiin ja päätettiin, että asia jätetään toistaiseksi auki, kunnes asiasta saataisiin kerättyä tarpeeksi palautetta käyttäjäkunnalta ja viranomaisilta.

Harjoituskirjaamo-sovellus toteutettiin prototyyppinä, jolloin projektiin ei kuulunut varsinaista ylläpitovelvollisuutta. Sovellus asennettiin Tilaajan työasemalle paikalliseksi järjestelmäksi, joten se ei edellytä kehittäjältä minkäänlaisia webhotellin asennus- ja asetustoimintoja.

5.1.2 Ei-toiminnalliset vaatimukset

Ohjelmistokehitysympäristöksi sovittiin Netbeans-ohjelmistokehitysalusta sekä Grails-ohjelmistokehys ja koko sovellus päätettiin toteuttaa Groovy-ohjelmointikielellä. Käyttöliittymä toteutettiin HTML5- ja CSS3-ohjelmointikielillä. Tietosisältöratkaisuna päädyttiin MySQL-tietokantaan, joka asennettiin paikallisena työasemalle. Peruskäyttäjän näkökulmasta sovellus toimii teknisesti kuvan 16 esittämällä tavalla.



KUVA 16. Peruskäyttäjän ajonaikainen suoritus sovelluksessa

Web-sovelluksen käytettävyys on tarkoin määritelty. Käyttäjäkunta on hyvin laaja ja tietotekninen osaaminen vaihtelevaa, jolloin käyttöliittymän täytyy olla tarpeeksi yksinkertainen ja helppo oppia. Käyttöliittymässä päätettiin käyttää mahdollisimman vähän linkkejä ja päätoiminnot kuvattaisiin ikoneilla. Käyttöliittymän tuli olla myös nopea käyttää, koska se antaisi mahdollisuuden käyttää ohjelmaa jo harjoitustilanteessa tai välittömästi sen jälkeen. Sovelluksen vikasietoisuuden täytyisi olla hyvä, jonka vuoksi käyttäjätilien toimintoja rajoitettiin ja tiedonsyöttö toteutettiin suurimmaksi osaksi monivalinnoilla ja rajatuilla tekstikentillä.

Tietoturvaratkaisuna päädyttiin Grails-ohjelmistokehyksen omaan Spring Security -liitännäiseen. Liitännäisen avulla kaikkien käyttäjätilien salasanat kryptataan tietokantaan tilien luonnin tai muokkauksen yhteydessä. Kryptausmenetelmänä käytetään SHA-512 algoritmia, joka on NSA:n suunnittelema ja Yhdysvaltain hallituksen standardisoitu salausmenetelmä. Sovelluksen istunnot ovat käyttäjäkohtaisia ja määräaikaisia. Jos jostain syystä selain sulkeutuu tai yhteys katkeaa, sovellus lopettaa istunnon. Oikea tapa on käyttää kuitenkin sovelluksen uloskirjautumislinkkiä. Sisäänkirjautuminen on suojattu luonnollisesti salasanan peitolla, jolloin ulkopuoliset eivät voi nähdä sitä näytöllä. Lisäksi eri käyttäjätilitasot suojattiin niille kuuluvilla rooleilla, jolloin esimerkiksi peruskäyttäjätasolla ei ole koskaan oikeuksia muille tasoille. Rekisteröintisivulle toteutettiin erillinen Captcha-kuvanvarmennusmenetelmä, jolla voidaan varmistaa, että käyttäjä on ihminen.

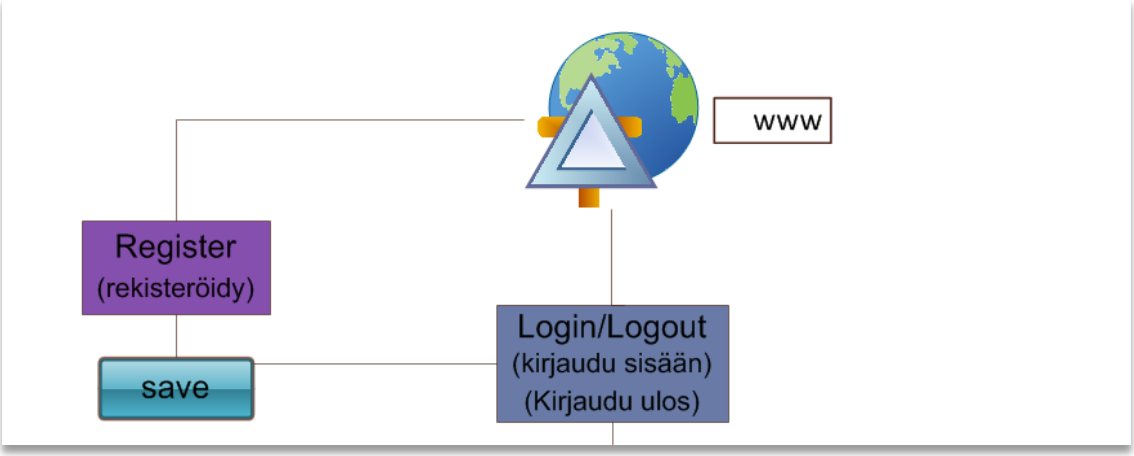
Harjoituskirjaamo-sovelluksen toimintavarmuus perustuu ainoastaan selainympäristöön ja sen mahdollisiin muuttujiin. Laiterajoituksia sovelluksella ei käytännössä ole. Tarvitaan ainoastaan Internet-yhteys. Sovellus testattiin kaikilla yleisimmällä Internet-selaimilla ja todettiin toimivaksi.

On selvää, että projektin luonteen vuoksi ei keskitytty sovelluksen luotettavuuteen ja ylläpitoon juuri lainkaan. Sovellus on prototyyppi ja julkaisuversio tulisi rakentaa alusta asti, jolloin ylläpidolla olisi suurempi merkitys. Sovelluksen siirrettävyydelle ei sinänsä ole muita rajoituksia kuin Java-pohjainen Web-palvelin sekä MySQL-tietokantapalvelin.

5.2 Arkkitehtuurisuunnittelu

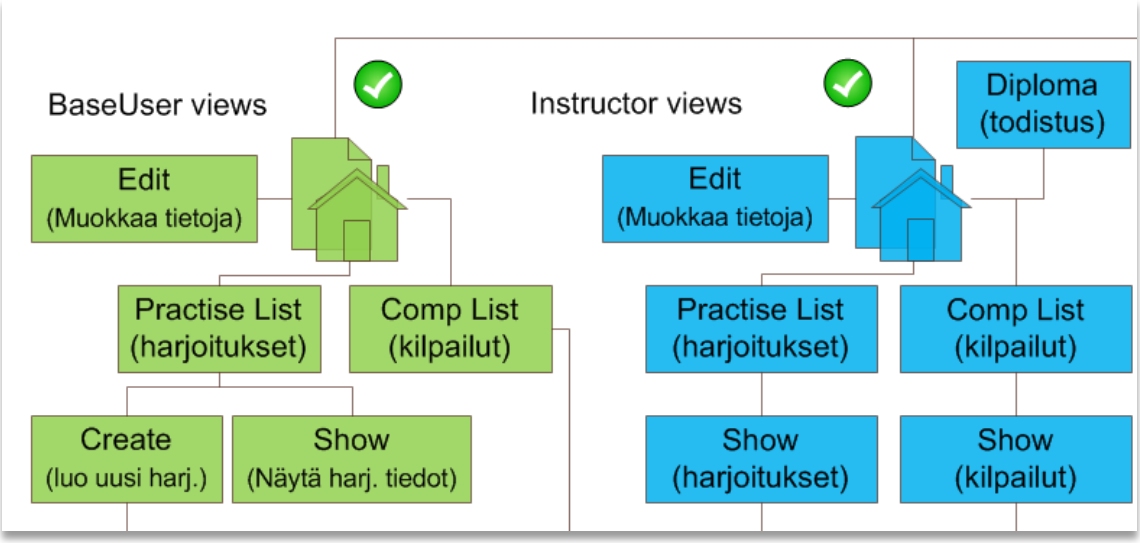
Arkkitehtuuria voidaan pitää järjestelmän yleisnäkymänä. Siinä esitellään ohjelmakomponentit ja niiden keskinäinen vuorovaikutus sekä toiminta järjestelmässä. Arkkitehtuurin tehtävänä on toimia siltana asiakasvaatimusten ja ohjelmiston välillä. Arkkitehtuurisuunnittelu tapahtuu vaatimusmäärittelyn ja varsinaisen ohjelmointityön välissä.

Harjoituskirjaamo-sovelluksen arkkitehtuuri rakennettiin osittain. Ensimmäisenä vaiheena piirrettiin rekisteröinti- ja kirjautumiskomponentit (kuva 17). Kuvasta voidaan jo päätellä sovelluksen rakennetta. Sovelluksella on yksi etusivu, josta kirjaudutaan sisään käyttäjätason mukaan. Kaikki käyttäjätasot käyttävät samaa kirjautumiskomponenttia. Rekisteröinti tapahtuu niin ikään etusivun kautta ja rekisteröinnin jälkeen voidaan kirjautua suoraan sisään. Uloskirjautuminen tapahtuu käännetyssä järjestyksessä.



KUVA 17. Arkkitehtuurin kirjautumisosa

Kirjautumisen jälkeen piirrettiin jokaisen käyttäjätason arkkitehtuurirakenne. Jokainen käyttäjätaso eroaa toisistaan sisällön perusteella, mutta jokaiseen pystyttiin käyttämään samaa rakennetta. Kuvassa 18 on havainnollistettu peruskäyttäjän sekä kouluttajan näkymien rakenteet.



KUVA 18. Peruskäyttäjän ja kouluttajan näkymät arkkitehtuurissa

Kuvasta voidaan havaita, että molemmissa käyttäjätasoissa on omat kotisivut. Peruskäyttäjällä on mahdollisuus valita kolme polkua: harjoitukset, kilpailut tai omien tietojen muokkaus. Kouluttajalla on näiden lisäksi myös todistusten kirjoittamismahdollisuus.

Arkkitehtuuri piirrettiin siis käyttäjätaso kerrallaan ja suunniteltiin jokaisen tason kaikki mahdolliset näkymät. Arkkitehtuurin pohjalla on kuvattuna tietokanta, josta on yhteydet tallennuslomakkeisiin. Koko arkkitehtuurinäkymä on kuvattu liitteessä 3.

Kuten voidaan huomata, arkkitehtuuri toimii kehittäjälle ikään kuin sovelluksen karttana, jossa eri komponentit ovat rakennuksia ja niiden väliset suhteet liikenneyhteyksiä. Suunnittelemalla riittävän havainnollinen kartta sovelluksen kokonaiskuva hahmottuu paremmin ja kehittäjä voi paneutua ohjelmakomponenttien toteutukseen. On myös muistettava, että isommissa ohjelmistoprojekteissa arkkitehtuurisuunnittelija on yleensä eri henkilö ja ohjelmistosuunnittelijoita voi olla useita, joten silloin arkkitehtuurin merkitys kasvaa entisestään.

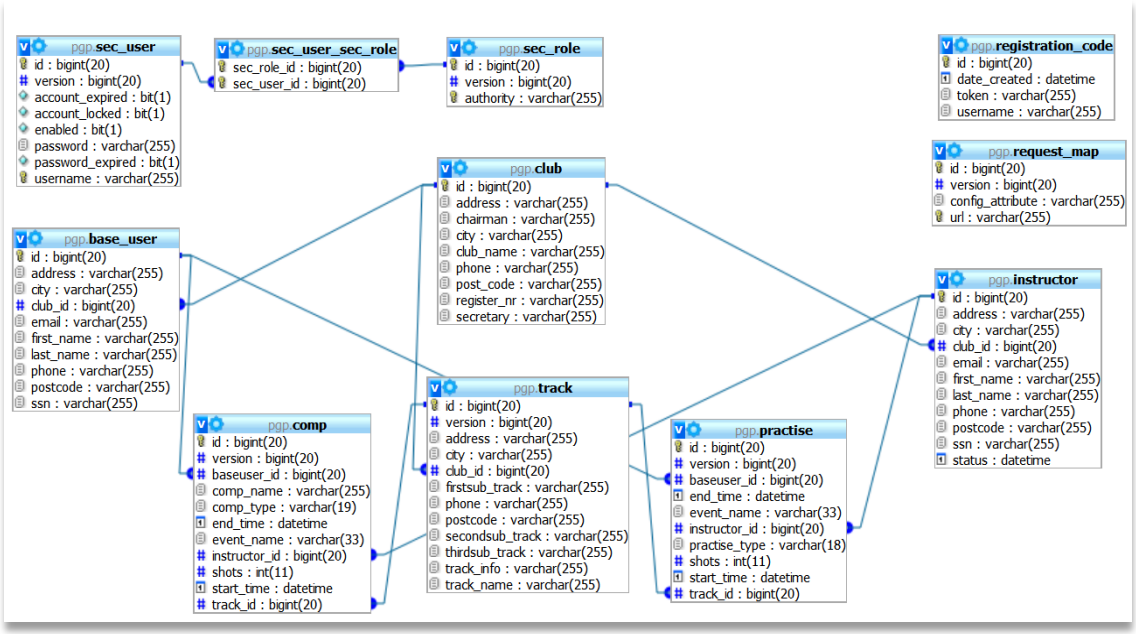
5.3 Tietokanta

Yhä useammat Web-sivustot ja järjestelmät käyttävät hyväkseen tietokantaa ja sitä voidaankin pitää ohjelmiston sydämenä. Tietokanta ei ole kuitenkaan itsestäänselvyys ja sitä kannattaa käyttää ainoastaan silloin, kun se on pakollista.

Tämä kannattaa tehdä sen vuoksi, koska tietokanta aiheuttaa aina enemmän suunnittelutyötä ja mahdollisesti ylläpitotoimia kehittäjälle.

On syytä tässä yhteydessä mainita pari sanaa tietokannoista. Tietokanta on määrittelemätön joukko tietoja, jotka ovat yhteydessä toisiinsa. Tietotekniikassa sitä käytetään ohjelmiston tietovarastona. Tietokannan rakentaminen perustuu siihen tallennettavan tiedon mallintamiseen ja yleisin tekniikka on relaatiotietokanta, joka perustuu relaatiotietomalliin. Relaatiotietokantaa voidaan hallita siihen soveltuvan SQL-kielen avulla, ja vaikka ohjelmistorajapinnassa käytettäisiin jotain muuta ohjelmointikieltä, lopulliset tietokannan hallintakomennot muutetaan SQL-kielisiksi. Tietokanta koostuu tauluista ja tietueista, joihin voidaan tallentaa eri formaateissa olevaa tietoa. Tietokannan koot voivat vaihdella yhden taulun tietokannoista hyvin suuriin tietokantoihin, joissa voi olla miljoonia tietueita. (27.)

Harjoituskirjaamo-sovelluksen tietokanta rakennettiin hieman normaalista poikkeavalla tavalla Grails-ohjelmistokehyksen vuoksi. Suunnittelun jälkeen tietokanta luotiin ja sille määriteltiin käyttöoikeudet MySQL-palvelimella (kuva 19). Kaikki tietokannan taulut ja tietueet määriteltiin dataolioina ohjelmistokehyksessä, joka käänsi komennot SQL-kieliseksi tietokannan rajapinnassa.



KUVA 19. Tietokantasuunnitelma Harjoituskirjaamo-sovellukseen

Seuraavassa esimerkissä kuvataan kahden taulun toteuttaminen Harjoituskirjaamo-sovellukseen. Aluksi Grails-ohjelmistokehykseen määriteltiin tietokantayhteydet paikalliseen tietokantaan, jonka nimeksi sovittiin pgp:

```
dataSource {
   pooled = true
   driverClassName = "com.MySQL.jdbc.Driver"
   username = "*****"
   password = "*****"
}

environments {
   development {
      dataSource {
         dbCreate = "update" // one of 'create', 'create-drop','update'
         url = "jdbc:MySQL://localhost:3306/pgp"
      }
   }
}
```

Tämän jälkeen luotiin Grailsin komentokehotteella kaksi taulua: Club (= ampumaseura) ja BaseUser (= peruskäyttäjä/harrastaja). Grailsissa tietokannan taulut mallinnetaan aina domain class- luokiksi. Komentojen jälkeen tauluille määriteltiin tarvittavat tietueet. Ampumaseurataulu toteutettiin seuraavasti:

```
class Club extends SecUser {

   String clubName
   String registerNr
   String address
   String postCode
   String city
   String phone
   String chairman
   String secretary


   static constraints = {

      clubName(blank: false)
      registerNr(blank: false)
      address(blank: false)
      postCode(blank: false)
      city(blank: false)
      phone(blank: false)
      chairman(blank: false)
      secretary(blank: false)
   }

}
```

Peruskäyttäjätaulu toteutettiin seuraavasti:

```
class BaseUser extends SecUser {

    String firstName
    String lastName
    String ssn
    String address
    String postcode
    String city
    String phone
    String email


    static constraints = {

        firstName(blank: false)
        lastName(blank: false)
        ssn(blank: false)
        address(blank:false)
        postcode(blank: false)
        city(blank: false)
        phone(blank: false)
        email(blank: false)
    }

}
```

Operaation jälkeen pgp-tietokantaan muodostui kaksi taulua ja niille tietueet. Kuvassa 20 on nähtävissä ampumaseurataulun rakenne.

KUVA 20. Ampumaseurataulun rakenne

Tässä vaiheessa tietokannan taulut olivat vielä yksinään toimivia ja niille piti määritellä vielä keskinäinen yhteys. Kuten jo ehkä taulujen nimistä voidaan

päätellä, yksi-moneen-yhteys tuottaa halutun lopputuloksen. Ampumaseuroilla voi olla useita peruskäyttäjiä, mutta peruskäyttäjä voi kuulua vain yhteen seuraan kerrallaan. Tästä johtuen tauluihin määriteltiin ehdot.

Ampumaseuratauluun määriteltiin seuraava ehto:

```
static hasMany = [baseusers: BaseUser]
```

Peruskäyttäjätauluun määriteltiin vastaavasti ehto:

```
static belongsTo = [club: Club]
```

Taulut ovat nyt yhteydessä keskenään ja toimivat ns. johdannaispäivitysperiaatteella, jolloin esimerkiksi poistettaessa ampumaseura tietokannasta myös siihen kuuluvat käyttäjät poistetaan automaattisesti. Tämän jälkeen tehtäväksi jäi ainoastaan taulun esittäminen halutun tietueen avulla. Esimerkiksi peruskäyttäjää haluttiin mallintaa sukunimen ja etunimen perusteella, jolloin domain-luokkaan määriteltiin vielä palautuslause:

```
String toString(){
      "${lastName}, ${firstName}"
   }
```

Näin taulusta palautetaan kätevästi peruskäyttäjä seuraavanlaiseen muotoon: etunimi, sukunimi. Grailsin domain-luokkien käsittelyssä on myös paljon muita etuja, kuten esimerkiksi constraints-lohko, johon voidaan määritellä muuttujien halutut raja-arvot. Sen avulla voidaan esimerkiksi määritellä näkymässä syötettävien merkkien enimmäismäärä ja päättää voiko syöttää tyhjää arvoa. Harjoituskirjaamo-prototyypissä ei ollut tarpeellista toteuttaa muuttujien validointeja.

5.4 Toiminnallisuus

Tietokannan jälkeen päästiin ohjelmoimaan Web-sovelluksen toiminnallisuus. Grails tarjoaa tukun valmiita työvälineitä ja ominaisuuksia dataolioiden käsittelyyn ja näkymien rakentamiseen.

5.4.1 Kontrollerit

Grails perustuu MVC-arkkitehtuuriin, jonka perusideana on kuljettaa tietoa domain-luokkien ja näkymien välillä. Tiedon kuljetus tarvitsee kuitenkin ohjausta ja siihen tarkoitukseen käytetään kontrollereita. Kontrolleri käsittelee kaikki pyynnöt ja luo tai valmistelee vastauksen. Se voi itse tuottaa vastauksen tai ohjaa valmistellun pyynnön näkymälle. Itse näkymät rakennetaan myös kontrollereissa funktioiden avulla.

Harjoituskirjaamo-sovelluksessa käytettiin domain-luokkia vastaavia kontrollereita sekä luotiin myös omia kontrollereita ohjelman rakenteen selkeyttämiseksi. Seuraavassa esimerkissä kuvataan peruskäyttäjän tiedot -sivun toteutus kahteen näkymään.

Aikaisemmin nähtiin, miten peruskäyttäjän tiedoille rakennettiin domain-luokka BaseUser. Tämän luokan avulla saatiin automaattisesti myös samanniminen kontrolleri. Domain-luokkien omat kontrollerit huolehtivat tietokantapyynnöistä, mutta ongelmaksi muodostui sivustojen rakentaminen. Ampumaseuran edustajan ja peruskäyttäjän itse tulisi pystyä tarkastelemaan ja muokkaamaan käyttäjän omia tietoja. Tarvittiin kopiot varsinaisista kontrollereista sekä roolitukset, joilla kukin kontrolleri jaettiin omaa käyttäjätasoa vastaavaksi. Peruskäyttäjän tietoja varten saatiin näin BaseUser-kontrolleri, joka oli ampumaseuran käytössä, ja BaseUserPage-kontrolleri, joka oli peruskäyttäjän käytössä. Molemmissa kontrollereissa luodaan omat näkymät, mutta kummatkin käyttävät samaa domain-luokkaa. Selvennetään tätä hieman esimerkin avulla.

Ampumaseuran BaseUser-kontrollerin rakenne on seuraava:

```
def index {
   ... (tähän käyttäjä-etusivu, ohjaus suoraan käyttäjälistaan)
}
def list {
  ... (tähän käyttäjälista)
}
def create {
  ...(tähän käyttäjän luonti)
}
def save {
  ...(tähän tallennuspyyntö)
}
def show {
```

```
  ...(tähän tietojen näyttö)
}
def edit {
  ...(tähän muokkausnäkymä)
}
def update {
  ...(tähän päivityspyyntö)
}
def delete {
  ...(tähän poistopyyntö)
}
```

Peruskäyttäjän puolella BaseUserPage-kontrollerin rakenne näyttää puolestaan seuraavanlaiselta:

```
def index {
   ... (tähän käyttäjä-etusivu)
}
def show {
  ...(tähän tietojen näyttö)
}
def edit {
  ...(tähän muokkausnäkymä)
}
def update {
  ...(tähän päivityspyyntö)
}
```

Tällä tavoin pystyttiin hyödyntämään valmiin kontrollerin funktioita myös toiseen kontrolleriin, jolloin säästyttiin turhalta ohjelmointityöltä. Tämä on yksi Grailsin vahvoja ominaisuuksia: käytetään hyödyksi valmiiksi toteutettuja ja testattuja funktioita muussa ohjelmakoodissa.

5.4.2 Erikoistoiminnot

Grails-ohjelmistokehyksen avulla voidaan suorittaa ohjelmistokoodiin tiettyjä erikoistoimintoja, jotka kuuluvat ketteriin ohjelmistokehitysmenetelmiin. Harjoituskirjaamo-sovelluksessa hyödynnettiin näistä muutamia.

Grails-projektin tärkein asetuksiin liittyvä kansio on nimeltään Conf. Kansio sisältää projektin tietokanta-, konfiguraatio- ja käynnistysasetukset. Lisäksi kansiossa on URL-osoitteistoon liittyvät ohjausasetukset.

Kun Grails-projekti käynnistetään, ladataan ensimmäisenä Bootstrap.groovy-tiedosto. Tässä tiedostossa sijaitsevat kaikki sovelluksen käynnistykseen liittyvät ohjelmakoodit. Tiedosto voi olla myös tyhjä, mutta Harjoituskirjaamo-

sovelluksessa siihen toteutettiin järjestelmänhallitsija-tilin luonti sekä sovelluksen käyttäjäroolit seuraavasti:

```
class BootStrap {

    def init = { servletContext ->

 def baseuserRole = SecRole.findByAuthority('ROLE_BASEUSER')?:new SecRole(authority: 'ROLE_BASEUSER').save(failOnError: true)
…
        def adminUser = SecUser.findByUsername('admin') ?: new SecUser(
            username: '*****',
            password: '*****',
            enabled: true).save(failOnError: true)
           …
        }
    }

    def destroy = {
    }
}
```

Config.groovy on Grails-projektin kokoonpanotiedosto, jossa sijaitsevat kaikki sovelluksen hallintatiedot. Harjoituskirjaamo-sovelluksessa sitä käytettiin lisäksi sivuston käyttäjäoikeuksiin ja roolitukseen. Seuraavassa esimerkissä on Spring Security-liitännäisen avulla määritetyt käyttäjäoikeudet järjestelmänhallitsija-sivustolle.

```
grails.plugins.springsecurity.securityConfigType = SecurityConfigType = 'InterceptUrlMap'
grails.plugins.springsecurity.interceptUrlMap = [

'/adminPage/**': ['ROLE_ADMIN'],
'/admin/**': ['ROLE_ADMIN']

]
```

Config-kansiossa sijaitsevaa Urlmappings.groovy-tiedostoa käytetään nimensä mukaisesti WWW-sivuston URL-osoitteiden ohjaukseen ja muokkaukseen. Se on erittäin hyödyllinen määriteltäessä sivustolle haluttu URL-osoitteisto. Harjoituskirjaamo-sovelluksessa sivustolle toteutettiin kotisivu nimeltä home. Normaalisti kotisivu kääntyisi muotoon *www.harjoituskirjaamo.fi/home,* mutta UrlMapping-tiedoston avulla se muutettiin muotoon *www.harjoituskirjaamo.fi/.* Tämä toteutettiin seuraavasti:

```
"/"(view:"/home")
```

Grails-projektin URL-osoitteet määräytyvät kontrollereissa määriteltyjen funktioiden mukaan. Osoitteita voidaan kuitenkin muokata URLMapping-tiedoston avulla muuttamatta funktioiden nimiä. Esimerkissä nähdään, miten sovelluksen järjestelmänvalvoja-sivuston URL-osoitteisto muokattiin paremmin kuvaavaksi.

Alkuperäiset URL-osoitteet olivat seuraavat:

www.harjoituskirjaamo.fi/club/list
www.harjoituskirjaamo.fi/club/create
www.harjoituskirjaamo.fi/club/show
www.harjoituskirjaamo.fi/club/edit

UrlMapping.groovy -tiedoston osoitteisto muutettiin seuraavasti:

"/admin/seurat/lista" (controller="club", action: "list")
"/admin/seurat/luonti" (controller="club", action: "create")
"/admin/seurat/nayta" (controller="club", action: "show")
"/admin/seurat/muokkaa" (controller="club", action: "edit")

Uudet URL-osoitteet kääntyivät muotoon:

www.harjoituskirjaamo.fi/admin/seurat/lista
www.harjoituskirjaamo.fi/admin/seurat/luonti
www.harjoituskirjaamo.fi/admin/seurat/näyta
www.harjoituskirjaamo.fi/admin/seurat/muokkaa

Grails-ohjelmistokehys sisältää paljon valmiita erikoistoimintoja. Useiden liitännäisten avulla niitä voidaan hyödyntää paremmin.

5.4 Käyttöliittymä ja ulkoasu

Nimensä mukaisesti käyttöliittymä on ohjelmiston osa, joka toimii käyttäjärajapinnassa. Käyttöliittymän avulla käytetään ohjelmistoa ja se toimii syötteenä varsinaiselle ohjelmistolle. Käyttöliittymän merkitys on olennainen myös liiketoiminnan kannalta, koska mahdollinen asiakas näkee ainoastaan sen osan ohjelmistosta. Asiakkaan kannalta epäolennaisia ominaisuuksia ovat toiminnallisuus ja tekniikka. Käyttöliittymä ja toiminnallisuus suunnitellaan aina rinnakkain, eikä kumpikaan toimi ilman toista.

Harjoituskirjaamo-projektissa yhtenä tavoitteena oli toteuttaa selkeä ja toimiva käyttöliittymä, joka olisi helppo oppia ja nopea käyttää. Varsinaiseen graafiseen ulkoasuun saatiin Tilaajalta ohjeistusta värimaailman suhteen. Lisäksi Tilaaja toimitti valmiin taustakuvan ja ikonit. Web-sovellus toteutettiin kahdessa sprintissä käyttämällä Scrum-ohjelmistokehitysmenetelmää. Ensimmäisessä sprintissä toteutettiin suurin osa sovelluksen toiminnallisuudesta. Toisessa sprintissä toteutettiin käyttöliittymä, ulkoasu sekä loput toiminnallisuudesta.

5.4.1 Näkymät

Harjoituskirjaamo-sovelluksen alkuperäinen idea oli toteuttaa käyttöliittymä, jossa käyttäjätasot erottuisivat värien mukaan. Jokaisella käyttäjätasolla olisi oma teemaväri, joka toistuisi WWW-sivujen eri elementeissä. Etusivun väriksi valittiin neutraali harmaa, koska se liittyy ampuma-aseisiin. Tilaajan toivomassa värimaailmassa oranssi edusti ampumaseuroja, sininen kouluttajia ja vihreä käyttäjiä (kuva 21). Varsinaiseen järjestelmävalvojatiliin ei määritelty teeman mukaista ulkoasua, koska se ei tule näkymään asiakasrajapinnassa.

KUVA 21. Käyttäjätasojen ikonit

Varsinaisen sovelluskehityksen puolella käyttäjätaso-näkymät jaettiin eri kansioihin ja tiedostoihin. Jokaisen käyttäjätason jokaista näkymää ohjattiin kontrollereiden avulla.

Grails-ohjelmistokehys käyttää sovelluksen näkymiin Groovy-ohjelmointikieleen perustuvia GSP-tiedostoja (= Groovy Server Pages). Uusi näkymä voidaan luoda komentokehotteesta samalla tavalla kuin domain-luokka ja kontrollerikin. Oikeastaan näkymä on aivan tavallinen HTML-tiedosto, joka käyttää tietojenkä-

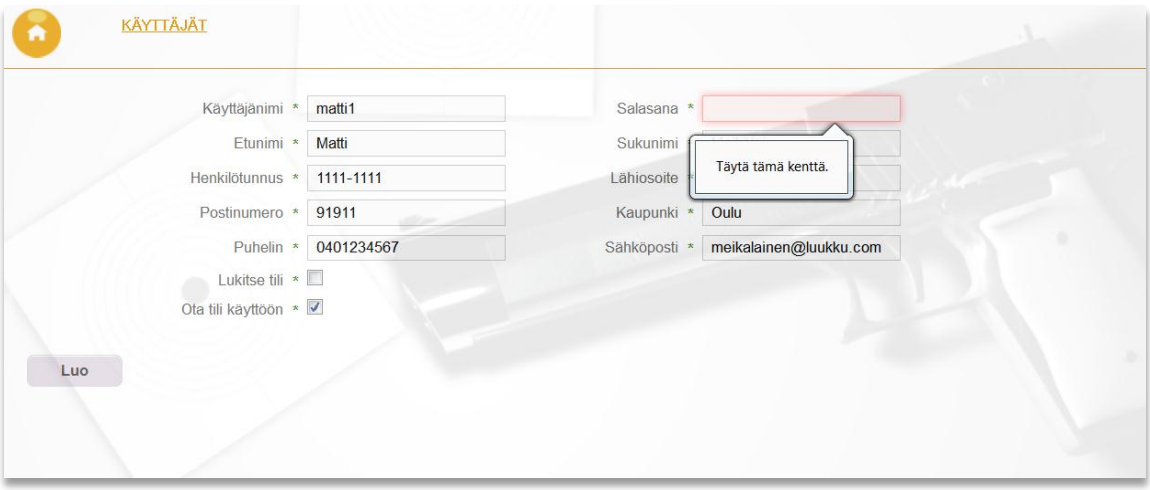
sittelyyn omia erityisiä Groovy-tageja. Seuraavassa esimerkissä nähdään peruskäyttäjän yhden GSP-tiedoston rakenne:

```
<%@ page import="org.pgp.BaseUser" %>
<!doctype html>
<html>
<head>
 <meta name="layout" content="baseUsermain">
 <g:set var="entityName" value="${message(code: 'baseUser.label', default: 'BaseUser')}" />
 <title><g:message code="Kyttäjä-omat tiedot"/></title>
</head>
  <body>

    …

 </body>

<html>
```

5.4.2 Tietojenkäsittely

Tässä luvussa syvennytään hieman tarkemmin siihen miten tieto liikkuu GSP-tiedoston ja domain-luokan välillä. Harjoituskirjaamo-sovellus perustuu lähes pelkästään käyttäjän antamiin tekstisyötteisiin ja tiedonlukuun tietokannasta.

Selvitetään prosessia esimerkin avulla. Esimerkissä ampumaseura A rekisteröi peruskäyttäjän Matti Meikäläinen Harjoituskirjaamo-sovellukseen ja muokkaa seuraavassa vaiheessa tämän yhteystietoja. Aluksi siirrytään peruskäyttäjän tilinhallinta-sivustolle ja luodaan käyttäjätili (kuva 22).



KUVA 22. Käyttäjätilin luominen ampumaseuran-sivustolla

On huomioitava, että kaikkiin tilinluontitapahtumiin on asetettu tyhjän tekstikentän poiskytkentä eli käyttäjän tulee syöttää jokaiseen kenttään jotain. Muita validiointeja ei tässä vaiheessa ole toteutettu. Käyttäjän tiedot siirretään sen jälkeen tietokantaan. Näkymän ohjelmakoodi näyttää seuraavanlaiselta:

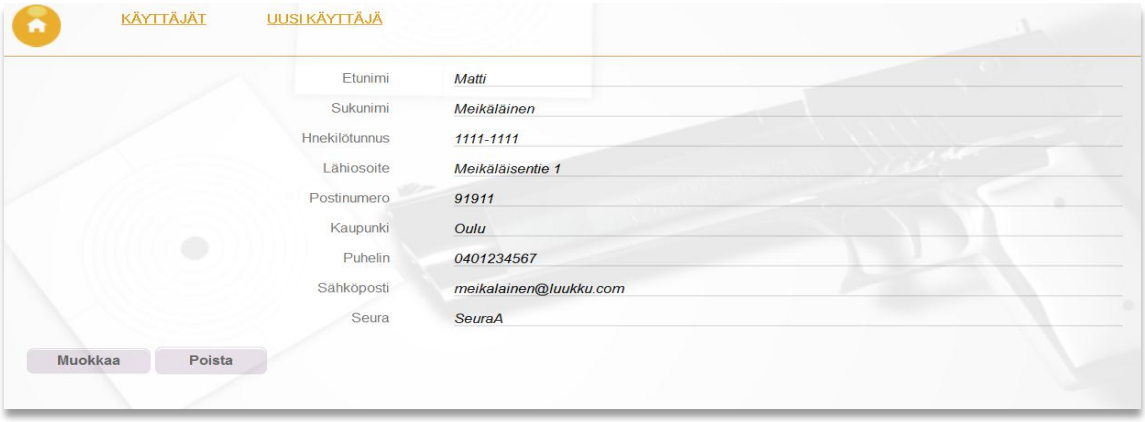
```
<div class="fieldcontain ${hasErrors(bean: baseUserInstance)} required">
   <label for="firstName">
        <g:message code="baseUser.firstName.label" default="Etunimi" />
                <span class="required-indicator">*</span>
   </label>
   <g:textField name="firstName" required="" value="${baseUserInstance?.firstName}"/>

   <label for="lastName">
       <g:message code="baseUser.lastName.label" default="Sukunimi" />
                <span class="required-indicator">*</span>
   </label>
  <g:textField name="lastName" required="" value="${baseUserInstance?.lastName}"/>
     …

</div>
```

Kun käyttäjä painaa Tallenna-painiketta, syötetyt arvot siirtyvät kontrollerin tallennus-funktion kautta domain-luokkaan:

```
def save() {
       def baseUserInstance = new BaseUser(params)
       if (!baseUserInstance.save(flush: true)) {
          render(view: "create", model: [baseUserInstance: baseUserInstance])
          return
 }
```

Ohjelmakoodia tarkastelemalla voidaan havaita sen perustuvan HTML- ja Groovy-ohjelmoinkielten sekoitukseen. HTML määrää sivuston rakenteen ja Groovy huolehtii tiedon kuljettamisesta eteenpäin. Tieto lähetetään määrittelemällä tekstikentän arvoksi domain-luokan dataolion muuttuja, jota kuljetetaan kontrollereiden avustamana tietokantaan ja takaisin.

Seuraavassa vaiheessa voidaan katsoa luodun käyttäjätilin tietoja omasta näkymästä (kuva 23).



KUVA 23. Käyttäjän tietojen katselu

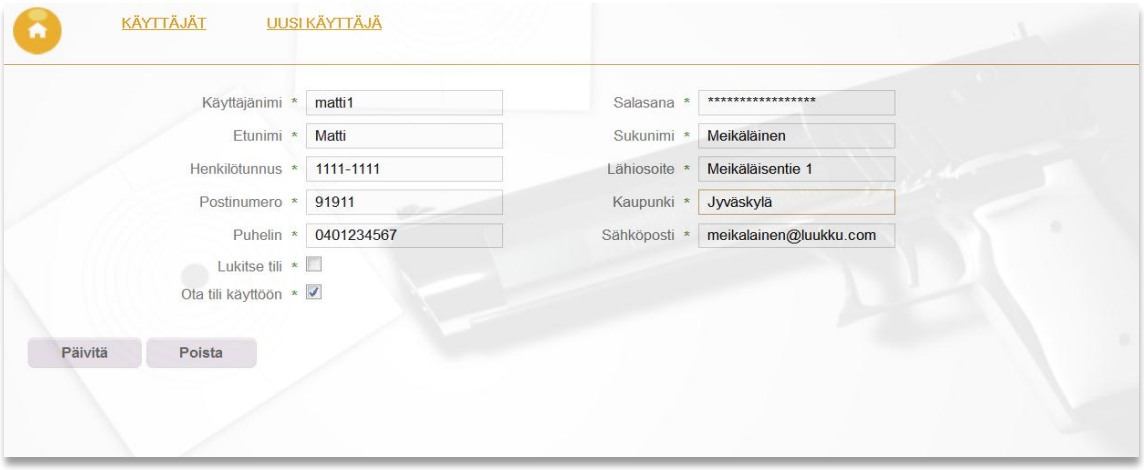
Ohjelmakoodissa osa WWW-sivusta näyttää seuraavanlaiselta:

```
<g:if test="${baseUserInstance?.firstName}">
<li class="fieldcontain">
<span id="firstName-label" class="property-label"><g:message code=
"baseUser.firstName.label" default="Etunimi" />
</span>

<span class="property-value" aria-labelledby="firstName-label"><g:fieldValue
bean="${baseUserInstance}" field="firstName"/>
</span>
</li>
</g:if>
```

Koodin perusteella havaitaan, että aikaisemmin viedyt dataolion arvot haetaan tietokannasta näytölle. Kontrollerit toimivat kummassakin tapauksessa tiedon välittäjinä. Kontrolleri välittää tiedon näkymälle seuraavasti:

```
def show() {

def baseUserInstance = BaseUser.get(params.id)
 if (!baseUserInstance) {
        flash.message = message(code: 'default.not.found.message', args: [message(code:
'baseUser.label', default: 'BaseUser'), params.id])
redirect(action: "list")
return
}
      [baseUserInstance: baseUserInstance]
}
```

Viimeisessä vaiheessa käyttäjätilin tietoja tulee pystyä muokkaamaan samaan tapaan kuin edellisissä. Linkistä siirrytään ensin oikealle WWW-sivulle ja syötetään muuttuneet tiedot (kuva 24). Tässä tapauksessa halutaan vaihtaa käyttäjän kotikaupungiksi Jyväskylä.



KUVA 24. Tietojen muokkaus-näkymä

Muokkausnäkymä käyttää hyväksi samaa lomakerakennetta kuin käyttäjän luontinäkymä, joten tietojenkäsittelyn osalta ohjelmakoodi on sama molemmissa näkymissä. Tällä tavoin tieto liikkuu näkymistä tietokantaan päin ja päinvastoin. Näkymät ovat vain perinteisiä HTML-sivuja, joissa käytetään tiedonkäsittelyyn Groovyn olio-ohjelmointimahdollisuuksia. Pääsivun ohjelmakoodi on esitetty liitteessä 5.

5.4.3 Graafinen ulkoasu

Harjoituskirjaamo-sovelluksen viimeisenä vaiheena oli toteuttaa sivustolle esittelykelpoinen ulkoasu. Ulkoasuun kohdistuivat seuraavat vaatimukset:

- Käyttöliittymän pitää pysyä kokonaan tavallisen näyttöresoluution (1368x768) sisällä ilman näytön vierityksiä.
- Jokaisen käyttäjätason tulee erottua omalla teemavärillään.
- Päätoiminnot sivustolla toteutetaan ikonien avulla.
- Valmiiksi suunniteltujen logon ja taustakuvan tulevat näkyä joka sivulla.
- Käyttöliittymän tulee olla selkeä ja yksinkertainen.

Sovelluksen ulkoasun toteutus aloitettiin määrittelemällä HTML-tiedostoihin tarvittavat elementit, jotka sitten muotoiltiin CSS-tyylitiedoston avulla. Logo ja taustakuva ladattaisiin vain yhden kerran. Etusivu toteutettiin ensimmäisenä ja sen rakenne toimi pohjana muille sivuille (kuva 25).

KUVA 25. Harjoituskirjaamo-sovelluksen etusivu

Etusivulle tehtiin kolme päälinkkiä ikonien avulla. Jokainen ikoni kantaa omaa teemaväriään ja kunkin käyttäjätason sivusto on ikonin värin mukainen. Taustaväri muuttuu punaiseksi, kun hiiren kursoria liikutetaan ikonin päälle. Tämä tyylimääritys on aivan tavallinen CSS-tiedoston ominaisuus, mutta Harjoituskirjaamo-sovelluksessa siihen päätettiin tehdä CSS3-tekniikan avulla ajastettu värin häivytys lisäefektinä. Lisäksi ikoneille tehtiin reunanpyöristykset sekä varjostukset niin ikään CSS3-tekniikalla. Tällä tavalla käyttöliittymästä saatiin pehmeämpi ja silmälle miellyttävämpi. Etusivu on kokonaisuudessaan nähtävissä liitteessä 4. Alla olevassa esimerkissä on CSS3-tekniikan avulla toteutettu värin häivytys:

```
.icons img {

    -moz-transition: background-color 0.5s linear;
    -o-transition: background-color 0.5s linear;
    transition: background 0.5s linear;
   -webkit-transition: background-color 0.5s linear;

}

.icons a img:hover {
   background-color: #f8b5b5;
}
```

Kirjautumisikkuna toteutettiin samalla tavalla kuin etusivu. Käyttäjä siirtyy samalle kirjautumissivulle jokaisesta ikonista. Esimerkissä kirjaudutaan sisään peruskäyttäjänä (kuva 26).

KUVA 26. Sovelluksen sisäänkirjautumissivu

Peruskäyttäjän kotisivu on teemavärin mukaisesti vihreä ja koti-ikonia klikkaamalla käyttäjä palautetaan aina omalle kotisivulleen (kuva 27). Vasta uloskirjautumisen jälkeen päästään takaisin sovelluksen varsinaiseen etusivunäkymään.

KUVA 27. Peruskäyttäjän kotisivu

Sovelluksen elementtien koko määriteltiin CSS-tiedostossa ja elementtejä pystyttiin liikuttelemaan kätevästi Grid 960-kehyksen avulla. Molemmat tiedostot ladataan etusivulle tultaessa, jolloin käyttäjä näkee sovelluksen esitetyllä tavalla.

6 JATKOKEHITYS

Harjoituskirjaamo-sovellus toteutettiin prototyyppinä ja se versioitiin numerona 0.1. Prototyyppi ei sinänsä ole valmis sovellus vaan eräänlainen testiversio varsinaisesta julkaisuversiosta. Prototyyppiin kohdistuvat kuitenkin melkein samat vaatimukset ja laatumääritykset kuin julkaisuversioonkin. Sen tulee olla toimiva ja käyttöliittymältään jo kehittynyt malli, jota voidaan testata ja markkinoida.. Tavoitteena on rakentaa työkalut, joiden avulla seuraavien versioiden kehitys nopeutuu ja yksinkertaistuu.

6.1 Kohti julkaisuversiota

Harjoituskirjaamo-projekti on uusi tuoteidea ja sovelluksen prototyyppi toimii markkinointivälineenä asiakasrajapinnassa. Varsinaisen julkaisuversion kehittäminen aloitetaan vasta jos asiakaspalaute on myönteistä ja myös viranomaiset näyttävät tuotteelle vihreää valoa.

On selvää, että mahdollisen julkaisuversion toteutukseen tarvitaan enemmän resursseja kuin prototyyppiin. Resursseja kuluu tuotekehitykseen, ohjelmakoodin parantamiseen, ylläpitoon ja testaukseen. Julkaisuversion rakentaminen aloitetaan alusta, mutta siinä käytetään hyväksi prototyypissä rakennettua ohjelmakoodia. Valmiissa tuotteessa vikasietoisuuden täytyy olla parempi kuin protomallissa ja sen tulee täyttää laadullisesti tarkemmat kriteerit. Harjoituskirjaamon julkaisuversio istutetaan luonnollisesti myös WWW-ympäristöön. Siihen tarvitaan paljon työtä myös palvelinpuolella. Lisäksi valmiin tuotteen täytyy läpäistä erilaiset testit ennen kuin se voidaan ottaa käyttöön. Testausta suoritetaan moduulitasolla ja lopuksi testataan koko ohjelmisto. Harjoituskirjaamo-sovellus tarvitsee myös jatkuvaa ylläpitoa ja vastuukysymykset tulisi selvittää mahdollisen projektin alkaessa.

Varovasti arvioiden julkaisuversion kehittämiseen menisi aikaa kahden hengen tiimiltä puolesta vuodesta vuoteen. Lisäksi ylläpitoon tarvittaisiin yksi henkilö kokoaikaisesti.

6.2 Visio

Vuonna 2015 Harjoituskirjaamo-sovellus täyttää kolme vuotta ja se on käytössä koko Suomen ampumaharrastuspiirissä. Sen avulla ampumaharrastaja voi pitää omaa sähköistä harjoituspäiväkirjaa ja tulostaa erilaisia yhteenvetoja omista harjoituksistaan. Harrastaja pystyy syöttämään harjoittelutiedot matkapuhelimen avulla jo harjoitusalueella. Harrastajan käynnit alueella rekisteröidään tietokantaan ja ne toimivat virallisena dokumenttina ampuma-aselupaa anottaessa. Luvan anominen tai jatkoaika toteutetaan suoraan ohjelmiston avulla.

Ampuma-asekouluttajat käyttävät Harjoituskirjaamoa ampumaharjoitusten ja kilpailutoiminnan seuraamiseen sekä ylläpitoon. Heillä on käytössä omilla sivuilla automaattinen laskuri, joka laskee kunkin kouluttajan piiriin kuuluvan harrastajan suorituskerrat vuosittain. Kouluttajille tulee sähköinen pyyntö harrastajalta ampumaharrastuksen aktiivisuuden varmistamisesta. Kouluttajat voivat kirjoittaa todistuksen sovelluksen avulla ja lähettää sen sähköisesti viranomaisille luettavaksi.

Ampumaseurat pitävät Harjoituskirjaamon avulla yllä jäsenrekisteriä ja voivat helpommin saada sovelluksen kautta uusia jäseniä seuran piiriin. Ampumaseuran edustajat voivat pitää sähköistä ilmoitustaulua mahdollisista kilpailuista sekä esimerkiksi rataolosuhteista, jotka ovat heti harrastajien luettavissa.

Harjoituskirjaamo-sovellus on käytössä kaikissa Suomen virallisissa ampumaseuroissa. Se kattaa kaikki maan ampumaradat ja sen käyttäjäpiirin kuuluu lähes 600 000 ampumaharrastajaa ja -asekouluttajaa. Sovelluksesta on kehitteillä versio 2.0, joka on suunnattu mobiilikäyttöön ampumaharrastuspaikoille.

7 YHTEENVETO

Tämän opinnäytetyön tarkoituksena oli tehdä prototyyppi ampumaharrastustoimintaan tarkoitetusta Web-sovelluksesta. Työn tavoitteena oli toteuttaa valmis toimintamalli, jota voitaisiin käyttää markkinointi- ja esittelytarkoitukseen. T:mi Proguns P. Aro on tuoteidean kehittäjä ja yrityksellä on vahva aikomus saada tuote julkaisukelpoiseksi. Tuoteidea oli mielenkiintoinen ja oli todella antoisaa olla mukana suunnittelemassa ja kehittämässä täysin uutta tuotetta. Toivottavasti se nähdään vielä markkinoilla.

Omasta mielestäni opinnäytetyön laajuus ja haasteellisuus oli juuri sopiva. Sain toteuttaa ohjelmiston itse alusta loppuun, jolloin pääsin näkemään koko ohjelmistotuotannon elinkaaren aina määrittelyvaiheesta lopputestaukseen. Tällainen projekti on oppimisen kannalta erittäin hyödyllinen ja se antaa paljon ammattitaitoa tukevia valmiuksia työelämään. Lopputulosta voidaan pitää onnistuneena ja projekti pysyi hyvin suunnitellussa aikataulussa.

Ainoaksi ongelmaksi työssä muodostuivat liian epätarkat asiakasvaatimukset. Tilaaja ei vielä tiennyt projektin alussa mitä kaikkia ominaisuuksia prototyypin tulisi sisältää. Vaatimuslistaa kasvatettiin projektin kuluessa, mutta se olisi voinut olla alussa tarkempi. Lisäksi tietoa ampumaharrastuksen nykyisestä käytännöstä olisi tarvittu enemmän.

Grails-ohjelmistokehyksellä toteutettu web-sovellus oli kokonaisuutena erittäin mielenkiintoinen projekti. MVC-arkkitehtuurin ohjelmistokehyksistä tätä ennen oli tuttu ainoastaan Codeigniter, joka perustuu PHP-ohjelmistokieleen. Grails ja Groovy-ohjelmointikieli olivat molemmat uusia tuttavuuksia, mutta pienen harjoittelujakson jälkeen uudenlainen ohjelmistotekniikka osoittautui todella monikäyttöiseksi. Aion myös jatkossa käyttää Grails-ohjelmistokehystä web-ohjelmointiin. Projektin aikana opin käyttämään myös Scrum-menetelmää sekä päivitin omia dokumentaatiotaitojani.

LÄHTEET

1. Aro, Petteri 2012. Projektin lähtötietomuistio. Word-dokumentti.

2. Projekti. 2012. Saatavissa: http://fi.wikipedia.org/wiki/Projekti. Hakupäivä 20.4.2012.

3. Projektinhallinta. 2012. Saatavissa: http://fi.wikipedia.org/wiki/Projektinhallinta. Hakupäivä 10.10.2011.

4. Ketterät eli Agile-menetelmät. 2012. Saatavissa: http://reaktor.fi/osaaminen/agile/ Hakupäivä 20.4.2012.

5. Ketteryys haltuun: Scrum pähkinäkuoressa. 2012. Saatavissa: http://www.meteoriitti.com/fi-FI/tiedotteet/ajankohtaista/ketteryys-haltuun-scrum-pahkinankuoressa. Hakupäivä 20.4.2012.

6. Scrum. 2012. Saatavissa: http://reaktor.fi/osaaminen/scrum/. Hakupäivä 20.4.2012.

7. Scrum. 2012. Saatavissa: http://en.wikipedia.org/wiki/Scrum_%28development%29 Hakupäivä 20.4.2012.

8. Larvanko Lasse 2007. Web 2.0 mullisti markkinoinnin. Saatavissa: http://www.inventive.fi/web-20-mullisti-markkinoinnin/ . Hakupäivä 21.4.2012.

9. Web 2.0. 2012. Saatavissa: http://fi.wikipedia.org/wiki/Web_2.0. Hakupäivä 21.4.2012.

10. Grails. 2012. Saatavissa: http://en.wikipedia.org/wiki/Grails_%28framework%29. Hakupäivä 22.4.2012.

11. Grails - the search is over. 2012. Saatavissa: http://grails.org/. Hakupäivä 22.4.2012.

12. Judd, C. – Nusairat, J. – Shingler, J. 2008. Beginning Groovy and Grails: From Novice to Professional. New York: Apress.

13. Enberg, P. – Raunio, M. 2007. Sulkeumat. Saatavissa: http://www.cs.helsinki.fi/u/wikla/OKP/ArtikkelitK07/sulkeumat.pdf. Hakupäivä 24.4.2012.

14. Object Relational Mapping (GORM) - Reference Documentation. 2012. Saatavissa: http://grails.org/doc/latest/guide/GORM.html. Hakupäivä 23.4.2012

15. NetBeans. 2012. Saatavissa: http://en.wikipedia.org/wiki/NetBeans. Hakupäivä 24.4.2012.

16. Apache friends – XAMPP. 2012. Saatavissa: http://www.apachefriends.org/en/xampp.html Hakupäivä 25.4.2012

17. XAMPP. 2012. Saatavissa: http://www.avkymppi.net/joomla/wamp/xampp.html. Hakupäivä 25.4.2012.

18. Apache Tomcat. 2012. Saatavissa: http://tomcat.apache.org/. Hakupäivä 25.4.2012.

19. Oulun seudun ammattikorkeakoulu – Web-sovellusten ohjelmointi: Web-sovellusten toteutustekniikoita. 2012. Saatavilla: http://www.oamk.fi/sbc/www/websovellus.php. Hakupäivä 26.4.2012.

20. HTML5. 2012. Saatavilla: http://en.wikipedia.org/wiki/HTML5. Hakupäivä 26.4.2012

21. HTML5 Ominaisuudet. 2012. Saatavilla: http://www.kulmaus.com/2012/01/10/html5-ominaisuudet/. Hakupäivä 26.4.2012.

22. CSS. 2012. Saatavilla: http://fi.wikipedia.org/wiki/Cascading_Style_Sheets. Hakupäivä 26.4.2012.

23. Jyväskylän yliopiston informaatioteknologian tiedekunta – Luennot: CSS-asemointi, CSS3, mediatyypit ja mediakyselyt. Saatavissa: http://appro.mit.jyu.fi/opetusteknologia/luennot/luento4/#TOC10. Hakupäivä 26.4.2012.

24. Henri Heiskanen. 2011. 960 Grid System Web-kehityksen apuvälineenä. Saatavissa: http://www.gofore.com/blogi/201103/960-grid-system-web-kehityksen-apuv%C3%A4lineen%C3%A4. Hakupäivä 26.4.2012

25. 960 Grid System. 2012. Saatavissa: http://960.gs/demo_24_col.html. Hakupäivä 26.4.2012.

26. Ohjelmiston vaatimusmäärittely. 2012. Saatavissa: http://fi.wikipedia.org/wiki/Ohjelmiston_vaatimusm%C3%A4%C3%A4rittely. Hakupäivä 26.4.2012.

27. Tietokanta. 2012. Saatavissa: http://fi.wikipedia.org/wiki/Tietokanta. Hakupäivä 27.4.2012.

28. Groovy and Grails 2012. Saatavissa: http://www-igm.univ-mlv.fr/~dr/XPOSE2009/Groovy_and_Grails/grails_architecture.php. Hakupäivä 27.4.2012.

LIITTEET

Liite 1 Tuotteen työlista

Liite 2. Grails-arkkitehtuuri

Liite 3. Sovellusarkkitehtuuri

Liite 4. Etusivunäkymä

Liite 5. Pääasivun ohjelmakoodi

Liite 6. Kouluttajan todistusnäkymä

Liite 7. Crud-toiminnallisuus

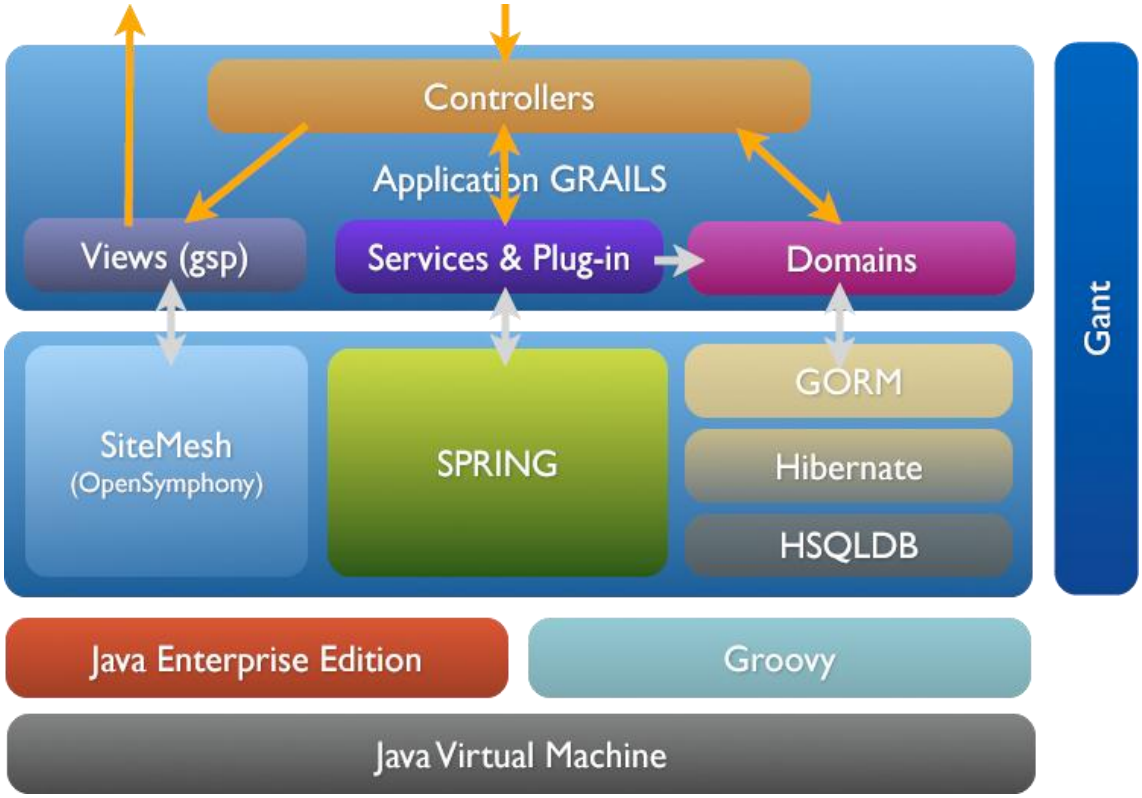
Erittäin tärkeä	ID	Ominaisuuden kuvaus/toiminnallisuus/laatuvaatimus	Sprintin numero	Tila
	1	Luodaan Grails-projekti ja varmuuskopiointijärjestelmä	1	
	2	Määritellään projektin asetukset (tietokanta, configuraatio)	1	
	3	Suunnitellaan ja toteutetaan tietokanta	1	
	4	Toteutaan järjestelmähallinnoijan- käyttäjätili bootstrap-osioon (aina ohjelman uudelleenkäynnistyessä tili luodaan	1	
	5	Haetaan ja ladataan tarvittavat pluginit projektiin	1	
	6	Määritetään Grails Spring Security asetukset sekä otetaan pluginin authentikointi ja kirjautuminen käyttöön	1	
	7	Toteutetaan roolitus sivustolle (admin, seura, kouluttaja, käyttäjä) sekä roolituksen poisto kannasta käyttäjää poistettaessa	1	
	8	Toteutetaan adminsivusto (authentikointi, linkit, seurat(listaus, lisäys, muokkaus)	1	
	9	Toteutetaan seurasivusto (authentikointi, linkit, omat tiedot(muokkaus), käyttäjät(listaus, lisäys, poisto, muokkaus), kouluttajat(listaus, lisäys, poisto, muokkaus), radat(listaus, lisäys, poisto,muokkaus))	1	
	10	Toteutetaan käyttäjäsivusto (authentikointi, linkit, omat tiedot(muokkaus),harjoitukset(listaus, lisäys)	1	
	11	Toteutetaan kouluttajasivusto (authentikointi, linkit, harjoitukset(poisto)	1	
	12	Toteutetaan pääsivu (linkit)	1	
	13	Määritetään projektin sähköpostiasetukset	1	
	14	Toteutetaan rekisteröintisivu (tietojen syöttö, captcha sekä sähköpostivarmenne)	1	
	15	Toteutetaan toisten käyttäjien harjoitusten näkemisen esto kirjautuneelle käyttäjälle	1	

Tärkeä	ID	Ominaisuuden kuvaus/toiminnallisuus/laatuvaatimus	Sprintin numero	Tila
	16	Toteutaan kouluttaja- sekä ratahaku harjoituksen luontiin	1	
	17	Määritetään listausasetukset harjoituksille, käyttäjille, kouluttajille radoille ja seuroille	1	
	20	Suunnitellaan toimiva käyttöliittymämuoto	2	
	21	Suunnitellaan sivuston ulkoasurakenne	2	
	22	Toteutetaan käyttöliittymäkoodaus (html, css)	2	
	23	Toteutetaan ulkoasukoodaus (html, css)	2	

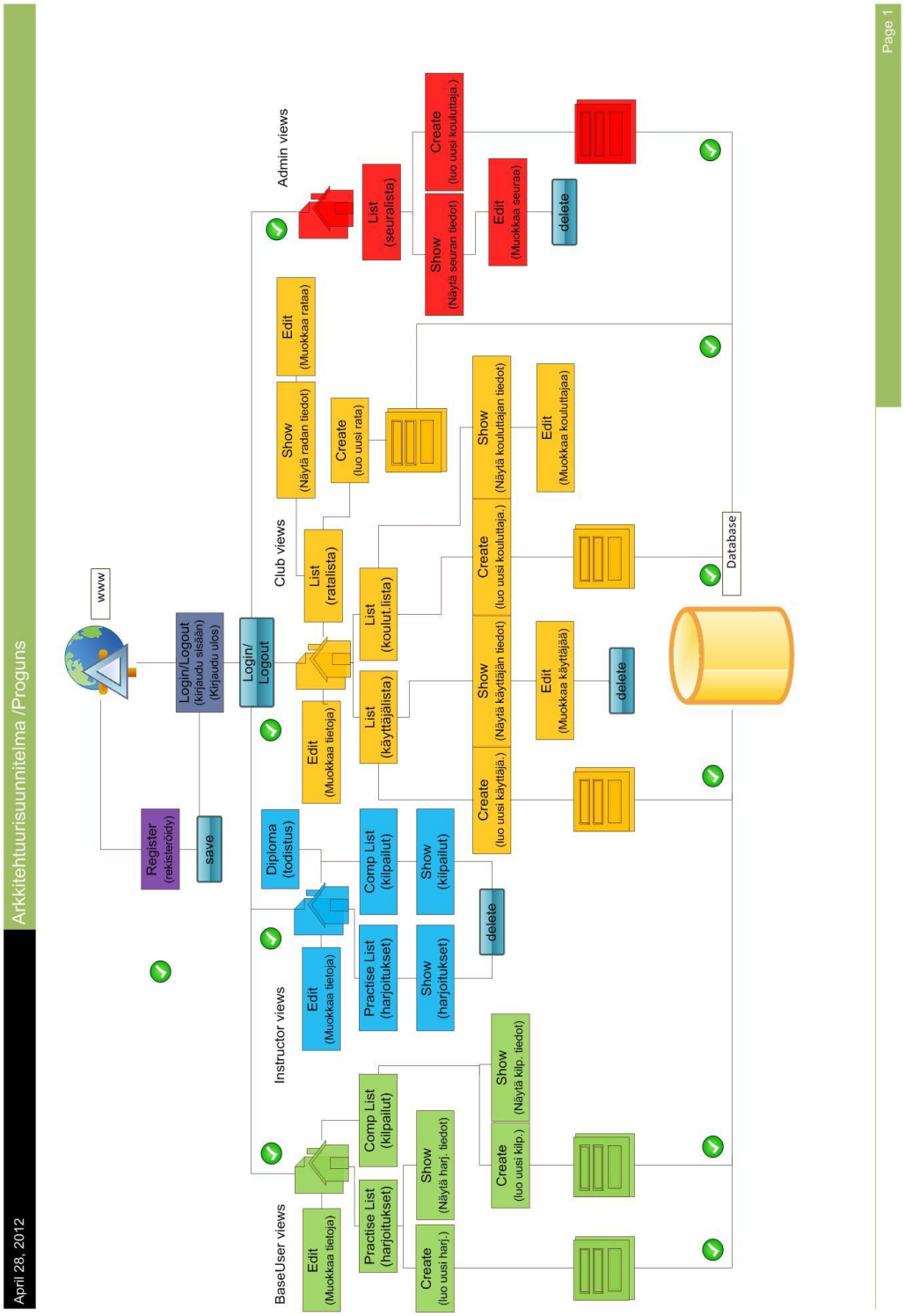
Keskitaso	ID	Ominaisuuden kuvaus/toiminnallisuus/laatuvaatimus	Sprintin numero	Tila
	18	Määritellään sivujen osoitteet (URL mapping)	1	
	19	Määritellään sivujen virheilmoitukset	1	

Matala	ID	Ominaisuuden kuvaus/toiminnallisuus/laatuvaatimus	Sprintin numero	Tila
	24	Kommentoidaan/poistetaan koodia	2	

Muu vaatimus	ID	Asiakas- /ohjelmisto- /järjestelmävaatimukset	Sprintin numero	Tila
	25	Järjestelmähallinnoija voi luoda seuroja ja hallinnoida niitä		
	26	Ampumaseura voi luoda käyttäjätilejä ja hallinnoida niitä		
	27	Käyttäjä voi rekisteröityä ja luoda itselleen käyttäjätilin		
	28	Käyttäjän kirjautuessa käytetään omaa käyttäjätunnusta ja salasanaa		
	29	Harrastajan tiedot (salasana) tulee olla salattuja		
	30	Harrastaja syöttää järjestelmään harrastustietonsa /-aktiviteettinsa		
	31	Harrastustiedoissa tulisi olla tiedot harjoituksesta ja aika		
	32	Sijainti haluttaisiin varmistaa mobiililaitteesta		
	33	Pyytää tarvittaessa harrastuksestaan ampuma-asekouluttajalta todistuksen lupaviranomaisia varten		
	34	Kirjautunut käyttäjä pääsee näkemään kaikki syöttämänsä harrastustiedot		

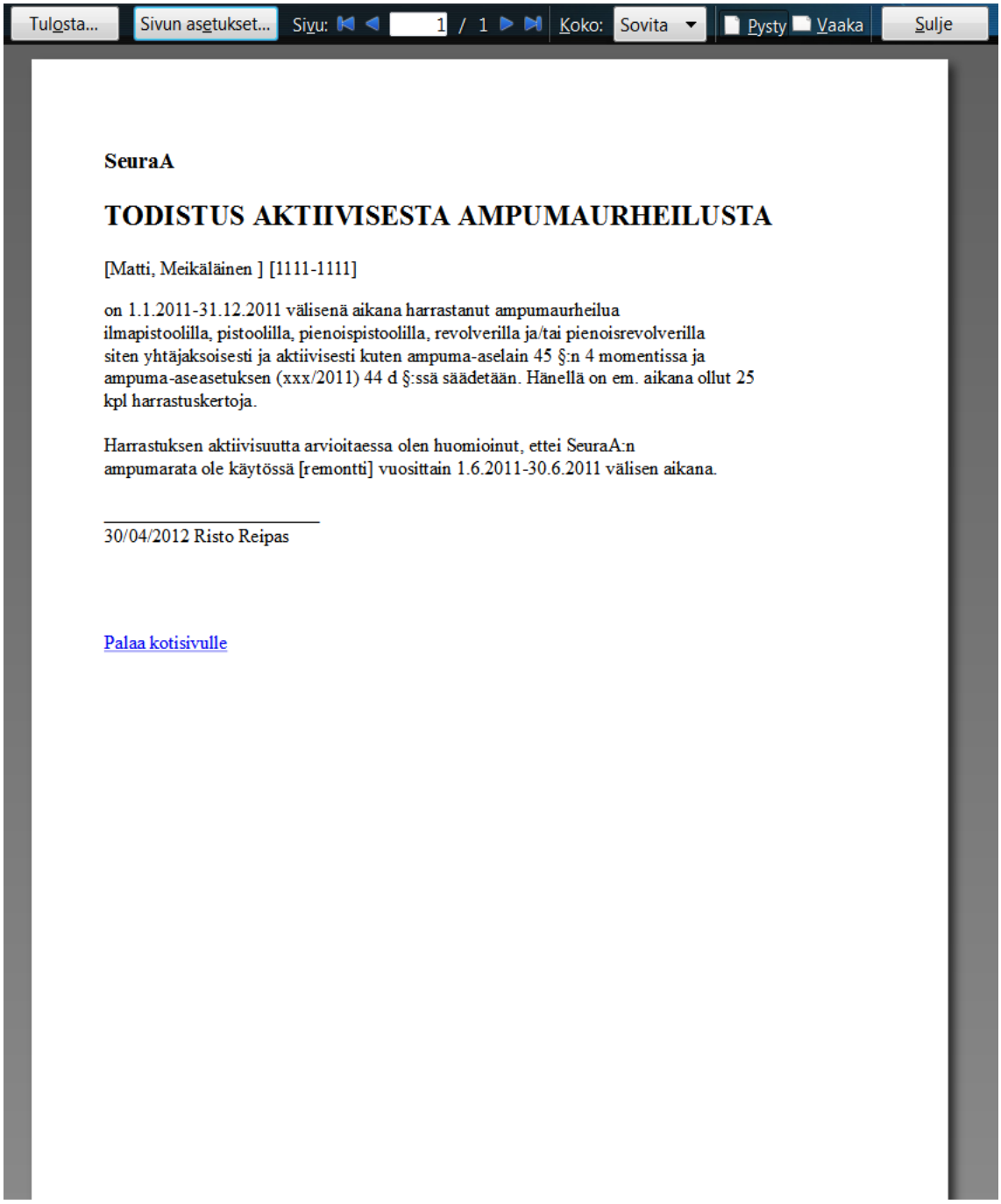

Controllers
Application GRAILS
Views (gsp)
Services & Plug-in
Domains
SiteMesh
(OpenSymphony)
SPRING
GORM
Hibernate
HSQLDB
Gant
Java Enterprise Edition
Groovy
Java Virtual Machine

(28.)


Arkkitehtuurisuunnitelma /Proguns
April 28, 2012
Page 1
www
Register (rekisteröidy)
save
Login/Logout (kirjaudu sisään) (Kirjaudu ulos)
Login/ Logout
BaseUser views
Edit (Muokkaa tietoja)
Practise List (harjoitukset)
Comp List (kilpailut)
Show (Näytä harj. tiedot)
Create (luo uusi harj.)
Create (luo uusi klp.)
Show (Näytä klp. tiedot)
Instructor views
Edit (Muokkaa tietoja)
Practise List (harjoitukset)
Comp List (kilpailut)
Show (harjoitukset)
Show (kilpailut)
delete
Diploma (todistus)
Club views
Edit (Muokkaa tietoja)
List (käyttäjälista)
List (koulut.lista)
List (ratalista)
Create (luo uusi käyttäjä.)
Show (Näytä käyttäjän tiedot)
Create (luo uusi kouluttaja.)
Show (Näytä kouluttajan tiedot)
Edit (Muokkaa käyttäjää)
Edit (Muokkaa kouluttajaa)
delete
Create (luo uusi rata)
Show (Näytä radan tiedot)
Edit (Muokkaa rataa)
Admin views
List (seuralista)
Show (Näytä seuran tiedot)
Edit (Muokkaa seuraa)
delete
Create (luo uusi kouluttaja.)
Database

HARJOITUSKIRJAANKO.FI
Eikö sinulla ole tiliä? Rekisteröidy käyttäjäksi
SEURA
KOULUTTAJA
KÄYTTÄJÄ

```
<!doctype html>
<!--[if lt IE 7 ]> <html lang="en" class="no-js ie6"> <![endif]-->
<!--[if IE 7 ]>    <html lang="en" class="no-js ie7"> <![endif]-->
<!--[if IE 8 ]>    <html lang="en" class="no-js ie8"> <![endif]-->
<!--[if IE 9 ]>    <html lang="en" class="no-js ie9"> <![endif]-->
<!--[if (gt IE 9)|!(IE)]><!--> <html lang="en" class="no-js"><!--<![endif]-->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
<title><g:layoutTitle default="Grails"/></title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="shortcut icon" href="${resource(dir: 'images', file: 'favicon.ico')}" type="image/x-icon">
<link rel="apple-touch-icon" href="${resource(dir: 'images', file: 'apple-touch-icon.png')}">
<link rel="apple-touch-icon" sizes="114x114" href="${resource(dir: 'images', file: 'apple-touch-
icon-retina.png')}">
 <link rel="stylesheet" href="${resource(dir: 'css', file: '960.css')}" type="text/css">
<link rel="stylesheet" href="${resource(dir: 'css', file: 'default.css')}" type="text/css">
<link rel="stylesheet" href="${resource(dir: 'css', file: 'mobile.css')}" type="text/css">
 <r:require modules="grailsui-autocomplete"/>
<g:layoutHead/>
<r:layoutResources />
</head>
    <body>
        <div id="header">
        <div class="container_16" id="logo"><img src="${resource(dir: 'images', file: 'logo.png')}"
alt="Home"/>
        <div id="header_menu">
            <ul>
                Eikö sinulla ole tiliä? <li><a href="${createLink(uri:
'/BaseUserRegister')}"><g:message code="Rekisteröidy käyttäjäksi"/></a></li>
            </ul>
        </div>
        </div>
        </div>

        <g:layoutBody/>
    <div id="spinner" class="spinner" style="display:none;"><g:message code="spinner.alt"
default="Loading&hellip;"/></div>
       <g:javascript library="application"/>
        <resource:autoComplete skin="default" />
       <r:layoutResources />
</body>
</html>
```



SeuraA

TODISTUS AKTIIVISESTA AMPUMAURHEILUSTA

[Matti, Meikäläinen] [1111-1111]

on 1.1.2011-31.12.2011 välisenä aikana harrastanut ampumaurheilua ilmapistoolilla, pistoolilla, pienoispistoolilla, revolverilla ja/tai pienoisrevolverilla siten yhtäjaksoisesti ja aktiivisesti kuten ampuma-aselain 45 §:n 4 momentissa ja ampuma-aseasetuksen (xxx/2011) 44 d §:ssä säädetään. Hänellä on em. aikana ollut 25 kpl harrastuskertoja.

Harrastuksen aktiivisuutta arvioitaessa olen huomioinut, ettei SeuraA:n ampumarata ole käytössä [remontti] vuosittain 1.6.2011-30.6.2011 välisen aikana.

_____

30/04/2012 Risto Reipas

Palaa kotisivulle


> grails create-app products
(then move to the folder 'products')
> grails create-domain-class Product
Add properties to generated class Product
> grails generate-all Product
> grails run-app
Application will be available at http://localhost:8080/products