Jonne Niskanen

GWT - SOVELLUSKEHYS KÄYTTÖLIITTYMÄN TOTEUTTAMISEEN

Käyttöliittymä UiBinder-sovelluskehyksellä

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma

Marraskuu 2016

MAMK
University of Applied Sciences

KUVAILULEHTI

MAMK University of Applied Sciences	**Opinnäytetyön päivämäärä** 30.11.2016
Tekijä(t) Jonne Niskanen	**Koulutusohjelma ja suuntautuminen** Tietojenkäsittelyn koulutusohjelma

Nimeke

GWT - Sovelluskehys käyttöliittymän toteuttamiseen. Käyttöliittymä UiBinder-sovelluskehyksellä

Tiivistelmä

Opinnäytetyössä vastataan työn toimeksiantajana toimivan Observis Oy:n tarpeeseen saada selvitys nykyisen UiBinder-sovelluskehyksen soveltuvuudesta yrityksen käyttöön. Yritys on käyttänyt tähän asti GWT-sovelluskehitysprojekteissaan kehittämäänsä xUI-sovelluskehystä, joka on aikoinaan kehitetty vanhan UiBinder-sovelluskehyksen päälle. Tarkoituksena on myös tukea yrityksen arviota xUI-sovelluskehyksen jatkokehittämisen kannattavuudesta.

Teoriaosuudessa kerrotaan lyhyesti yrityksestä ja tapahtumasovelluksista, johon myös käytännön työhön liittyvä Jurassic Rock -festivaalisovellus pohjautuu, sekä yrityksen käyttämistä työkaluista, joita myös tässä opinnäytetyössä käytetään. Keskeisimpänä näistä työkaluista on GWT, joka aloittaa varsinaisen teoriaosuuden käsittäen GWT:n versiot, GWT:n hyödyt sekä kääntäjän pääpiirteiset ominaisuudet. Varsinaisen teoriaosuuden ytimessä selvitetään UiBinder- ja xUI -sovelluskehyksien pääpiirteisiä ominaisuuksia ja niiden toimintaa. xUI-sovelluskehyksen toimintaa selvitetään sen lähdekoodin pohjalta piirretyn UML-toimintokaavion kautta, jonka avulla on muodostettu teoriapohja xUI-sovelluskehyksen toiminnalle.

Käytännön työnä toteutetaan Observis Oy:n kehittämän ja julkaiseman Jurassic Rock -festivaalisovelluksen käyttöliittymä UiBinder-sovelluskehyksellä. Työ luodaan uuden projektin pohjalle, ja siinä käydään läpi alkuperäisen xUI-sovelluskehyksellä toteutetun Jurassic Rock -festivaalisovelluksen käyttöliittymän mukainen toteutus. Lopputulemana muodostetaan johtopäätökset molemmista sovelluskehyksistä xUI-sovelluskehyksen jäädessään nykytilassaan selvästi nykyisen UiBinder-sovelluskehyksen varjoon. Yritys siirtyy käyttämään uusissa GWT-sovelluskehitysprojekteissaan UiBinder-sovelluskehystä tämän opinnäytetyön tulosten perusteella.

Asiasanat (avainsanat)

Sovelluskehys, Käyttöliittymä, Ohjelmointi, Java, JavaScript

Sivumäärä 73	**Kieli** Suomi

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi Jukka Selin	**Opinnäytetyön toimeksiantaja** Observis Oy

DESCRIPTION

MAMK University of Applied Sciences	**Date of the bachelor's thesis** 30 November 2016
Author(s) Jonne Niskanen	**Degree programme and option** Business Information Technology

Name of the bachelor's thesis

GWT - A framework for implementing a user interface. User interface with the UiBinder framework.

Abstract

The thesis was assigned by Observis Oy to answer to the employer's need to find out, if the modern UiBinder framework would be suitable for the company to use. This far, the company have used in their GWT software development projects self-developed xUI framework, which was originally developed on an old UiBinder framework. The purpose of the thesis among other things was to support the company's evaluation about continuing the development of the xUI framework.

The theory section shortly introduced the company and event applications which the user interface of the Jurassic Rock festival application implemented in the practical section based on, and the tools used in the thesis and by the company. The most essential of these tools was GWT, which started the main theory section including GWT's versions, GWT's benefits and the main features and properties of the compiler. In the core of the main theory section was UiBinder and the xUI framework's functions and the main features and the properties of the both frameworks. The theory of the xUI framework based on the UML activity diagram drawn from its source.

The practical section included the implementation of the Jurassic Rock festival application's user interface with the UiBinder framework developed and published by Observis Oy. The implementation based on a fresh new project and it followed the original user interface of the Jurassic Rock festival application made with the xUI framework. As outcome the conclusion was that the modern UiBinder framework left the xUI framework clearly behind. The company started using modern UiBinder framework in their GWT software development projects based on the results of this thesis.

Subject headings, (keywords)

Framework, User Interface, Programming, Java, JavaScript

Pages 73	**Language** Finnish

Remarks, notes on appendices

Tutor Jukka Selin	**Bachelor's thesis assigned by** Observis Oy

SISÄLTÖ

LIITTEET

LYHENTEET

AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
CSS	Cascading Style Sheets
DI	Dependency Injection
DOM	Document Object Model
GIN	GWT Injection
GPE	Google Plugin for Eclipse
GSS	Google Style Sheets
GWT	Google Web Toolkit
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
JAR	Java Archive
Java EE	Java Enterprise Edition
JSNI	JavaScript Native Interface
JSP	JavaServer Pages
PHP	PHP: Hypertext Preprocessor
POM	Project Object Model
RAD	Rapid Application Development
RC	Release Candidate
RPC	Remote Procedure Call
SDK	Software Development Kit
UML	Unified Modeling Language
URI	Uniform Resource Identifier
WAR	Web Application Archive
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

TERMIT

Annotaatio	Javassa käytettävä eräänlainen kommentti, joka kirjoitetaan Java-koodiin. Annotaatio tarjoaa dataa muusta ohjelmasta, joka ei ole osa ohjelmaa itseään. Annotaatio voi esimerkiksi toimia ohjeena kääntäjälle tunnistaa virheitä ja hiljentää varoituksia. Jotkin annotaatiot voivat olla käsiteltävissä myös Java-ohjelman ajoaikana. (Lesson: Annotations The Java™… 2016.)
Debuggaus	Ohjelmakoodin testaus, vianetsintä ja -korjaaminen. Englanninkielinen termi *debugging.*
Deklaratiivinen	Esimerkiksi tietty ohjelmakoodi voidaan kirjoittaa deklaratiivisesti, millä pyritään kuvailemaan mitä ohjelman tulee tehdä sen sijaan, että ohjelmakoodissa kuvattaisiin yksittäisiä suoritettavia käskyjä. Deklaratiivinen ohjelmointi on ohjelmointiparadigma, eli ohjelmointikielen taustalla oleva tapa ajatella ja jäsentää ohjelman toimintaa. (Vihavainen & Luukkainen 2016.) Englanninkielinen termi *declarative.*
HTML-entiteetti	Eräänlainen HTML:ssä käytetty merkkijono, joka alkaa et-merkillä ja loppuu puolipisteeseen. HTML-entiteettiä käytetään esimerkiksi silloin kun halutaan näyttää HTML-sivulla symbolia tai merkkiä, joka on rajoitettu HTML:ssä. Englanninkielinen termi *HTML entity.* (Tacy ym. 2013, 172.)
Instantiointi	Termi, jolla ohjelmoinnissa tarkoitetaan toimenpidettä, jossa luodaan uusi objekti. Kun uusi objekti luodaan, niin silloin luodaan luokan instanssi, eli tällöin luokka instantioidaan. Englanninkielinen termi *instantiate*. (Creating Objects The Java™… 2016.)
Minifiointi	Termi, jolla tarkoitetaan ylimääräisten merkkien, kuten välilyöntien, rivinvaihtojen ja kommenttien siivoamista koodista, jotta tiedoston koko saataisiin mahdollisimman pieneksi (Basic minification 2016). Tällaiset tiedostot ovat yleensä tarkoitettu luettavaksi vain ohjelmallisesti. Englanninkielinen termi *minification.*
Obfuskointi	GWT:n oletuksena tekemä JavaScript-koodin tarkoituksenmukainen monimutkaistaminen. Tämä tehdään osittain ohjelman immateriaalioikeuksien suojaamiseksi ja JavaScript-tiedostojen koon pienentämiseksi. Englanninkielinen termi *obfuscation.* (Why is my… 2016.)
Rajapintaluokka	Javassa käytettävä luokka, jonka muut Java-luokat voivat toteuttaa, eli implementoida. Rajapintaluokan implementoivien luokkien on tarjottava käytännön toteutukset kaikista rajapintaluokan metodeista, ellei luokka ole abstrakti.

	Rajapintaluokka on myös viittaustyyppi. Englanninkielinen termi *interface*. (Interfaces The Java™… 2016.)
Templaatti	Sivupohja, malli tai runko jostain, esimerkiksi www-sivusta tai projektin rakenteesta. Englanninkielinen termi *template.*
Widgetti	Käyttöliittymäkomponentti, jonka kanssa sovelluksen käyttäjä voi olla vuorovaikutuksessa. Englanninkielinen termi *widget.* (Tacy ym. 2013, 73.)

1 JOHDANTO

Tämän opinnäytetyön toimeksiantajana toimii Observis Oy, jolla on tarve selvittää GWT-sovelluksien käyttöliittymän toteuttamiseen tarkoitetun nykyaikaisen UiBinder-sovelluskehyksen soveltuvuutta yrityksen käyttöön. Yritys on käyttänyt tähän asti kehittämäänsä xUI-sovelluskehystä, joka on aikoinaan kehitetty vanhan UiBinder-sovelluskehyksen päälle helpottamaan käyttöliittymän toteuttamista GWT-sovelluksissa. Tarkoitus on myös tukea yrityksen arviota xUI-sovelluskehyksen jatkokehittämisen kannattavuudesta tutkimalla nykyaikaista UiBinder-sovelluskehystä ja sen soveltuvuutta xUI-sovelluskehyksen korvaajaksi tulevissa ohjelmistokehitysprojekteissa. Pyrin vastaamaan yrityksen esittämään tarpeeseen tällä opinnäytetyöllä, jonka pääpainotteisessa teoriaosuudessa selvitän molempien sovelluskehysten nykytilaista toimintaa, sekä niiden ominaisuuksia ja pääpiirteitä.

Luvussa kaksi esittelen lyhyesti yrityksen ja yrityksen tapahtumasovelluksiin perustuvan Jurassic Rock -festivaalisovelluksen, jonka alun perin xUI-sovelluskehyksellä toteutetun käyttöliittymän toteutan käytännön työnä uuden projektin pohjalle UiBinder-sovelluskehyksellä. Kerron yrityksen käytössä olevista työkaluista, joita myös tulen käyttämään tässä opinnäytetyössä. Näistä työkaluista hyvin keskeisessä asemassa tämän opinnäytetyön aihepiirin kannalta on GWT, jonka pääpiirteiset ominaisuudet käyn läpi luvussa kolme, jossa kerron GWT:n versioista ja GWT:n hyödyistä. Käyn myös läpi GWT:n kääntäjän, jonka pääpiirteistä toimintaa olen kuvannut liitteessä yksi.

Luvussa neljä käsittelen molemmat tämän opinnäytetyön pääosassa olevat sovelluskehykset: UiBinder-sovelluskehyksen ja xUI-sovelluskehyksen. Kerron sovelluskehysten pääpiirteisistä ominaisuuksista ja tarkoituksesta, mitä varten ne on kehitetty ja miksi niitä käytetään. Pyrin myös mahdollisimman tarkasti kertomaan yksinkertaisilla esimerkeillä havainnollistaen miten nämä sovelluskehykset toimivat luoden vahvan pohjan käytännön osuutta varten. Olen tutkinut xUI-sovelluskehysksen lähdekoodia ja dokumentoinut sen toiminnan piirtämällä liitteessä kaksi näkyvän UML-toimintakaavion; tämän pohjalta olen muodostanut myös luvussa neljä käsitellyn teoriapohjan xUI-sovelluskehykselle, jotta lopputulemana saamani johtopäätös sovelluskehyksistä perustuisi sen nykyisen toiminnan ymmärtämiselle.

Käytännön työnä luvussa viisi toteutan alun perin xUI-sovelluskehyksellä toteutetun Observis Oy:n kehittämän ja julkaiseman Jurassic Rock -festivaalisovelluksen mukaisen käyttöliittymän UiBinder-sovelluskehyksellä uuden projektin pohjalta. Tällä pyrin muodostamaan vertauspohjaa xUI-sovelluskehyksellä toteutettuun alkuperäiseen Jurassic Rock -festivaalisovelluksen käyttöliittymään, sekä muodostamaan kokonaisvaltaisen käsityksen UiBinder-sovelluskehyksen toiminnasta ja sen tuomista mahdollisuuksista myös käytännön työn kautta. Lopputulemana luvussa kuusi vastaan yrityksen esittämään tarpeeseen saada selville nykyaikaisen UiBinder-sovelluskehyksen soveltuvuus yrityksen käyttöön esittämällä johtopäätökset sovelluskehyksistä ja niiden nykytilasta.

2 YRITYS

Observis Oy on vuoden 2010 lopussa perustettu mikkeliläinen yritys, jonka toimialana on ATK-laitteisto- ja ohjelmistokonsultointi (Observis Oy on… 2013; Toimiala 2016). Yritys jatkoi kasvuaan viime vuonna ja he uutisoivat verkkosivuillaan liikevaihdon kasvaneen noin 34 prosenttiyksikköä ja samalla yhtiön toiminnan kääntyneen selkeästi kannattavaksi. Yhtiö teki silloin historiansa parhaimman tuloksen. Lisäksi kuluneen vuoden aikana yritys on myös rekrytoinut lisää henkilöstöä. (Kasvumme jatkuu 2016.)

Observis Oy kertoo verkkosivuillansa tekevänsä paikkatietoon perustuvia sovelluksia, jotka auttavat organisaatioita palvelemaan asiakkaitaan älykkäämmin. Yrityksen palvelut kokoavat eri lähteiden tarjoamaa tietoa, johon perustuen ne myös muodostavat visualisoituja tilannekuvia, joiden ansiosta organisaatio ymmärtää omia asiakkaitaan paremmin, viestii osuvammin, sekä toimii tehokkaammin. (Älykkäämpiä sovelluspalveluja 2016.) Yhtenä esimerkkinä tämän kaltaisesta palvelusta toimii Observis Oy:n kehittämä ja julkaisema Jurassic Rock -festivaalisovellus.

2.1 Tapahtumasovellukset

Jurassic Rock -festivaalisovellus on yksi monista Observis Oy:n kehittämistä ja julkaisemista mobiililaitteilla toimivista tapahtumasovelluksista. Observis Oy teki kolmivuotisen sopimuksen festivaalituottaja N.C.D Production Oy:n kanssa toteuttaa mobiilisovellukset festivaaleille (Keskitalo 2016). Festivaalisovelluksia on tämän opinnäy-

tetyön kirjoitushetkellä julkaistu Android, iOS ja Windows Phone -alustoille jo 11 kappaletta, joista mm. ensimmäisenä julkaistiin sovellus kesäkuussa 2016 järjestetylle South Park Tampere -festivaalille ja viimeisimpänä elokuussa 2016 järjestetylle We Love The 90's Festival -festivaalille. Jurassic Rock -festivaalisovellus julkaistiin elokuun 5.–8. päivä vuonna 2016 järjestetylle kymmenvuotisjuhliaan viettävälle Jurassic Rock -festivaalille. Observis Oy on tehnyt mobiilisovelluksen Jurassic Rock -festivaalille jo viitenä vuonna ja tällä kertaa yrityksen porukka oli myös paikan päällä festivaaleilla mukana (kuva 1; Ojala 2016a, 6).

KUVA 1. Takarivin keskeltä löytyy myös tämän opinnäytetyön kirjoittaja. Kuva: Onni Ojala (2016b)

Observis Oy:n festivaalisovellukset perustuvat tapahtumasovellukseen, josta on nähtävissä tapahtuman ohjelma esiintyjineen ja aikatauluineen, tapahtuman kartta palvelupisteineen, tapahtuman info, Twitter-viestit sosiaalisen median syötteenä sekä tapahtuman järjestäjien lähettämät viestit. Sovellus vastaa haasteeseen, jossa festivaaleilla on paljon tarjontaa palveluista ja muista tapahtumista, mutta niistä tiedottaminen on hajanaista; paperilla saatavien oppaiden tiedot eivät pysy ajan tasalla, eikä tapahtuman järjestäjä tavoita vieraita tai henkilökuntaa reaaliajassa (Haaste 2016). Kyseessä siis on mobiililaitteissa toimiva sovellus, joka mahdollistaa mm. navigoinnin tapahtuman

alueella. Tapahtuman järjestäjillä on mahdollisuus myös itse päivittää sovelluksessa näkyvää infoa, sekä lähettää viestejä kaikille sovelluksen asentaneille käyttäjille.

2.2 Työkalut

Observis Oy:n tapahtumasovellukset ovat monien muiden yrityksessä kehitettyjen sovelluksien tapaan GWT-sovelluksia, joiden käyttöliittymä on toteutettu Observis Oy:n kehittämällä xUI-sovelluskehyksellä. Yritys käyttää ohjelmistokehitysprojekteissaan pääsääntöisesti Eclipse IDE:tä, sillä se on osoittautunut hyväksi integroiduksi kehitysympäristöksi yrityksen tarpeisiin; se tukee hyvin GWT:tä ja sen projektirakenteita erikseen asennettavan lisäosan kautta ja sisältää myös integraation Apache Mavenille, jota yritys käyttää riippuvuuksien hallintaan myös muissa ohjelmistokehitysprojekteissaan (Using Eclipse 2016; M2Eclipse).

Eclipse on tullut tunnetuksi integroituna kehitysympäristönä lähinnä Javalle, mutta Eclipse IDE:stä on saatavilla pakettina erikseen myös Java EE:lle, C-kielille ja PHP:lle oma integroitu kehitysympäristönsä. Kaikkiin Eclipse IDE:n oletuspaketteihin on mahdollista lisätä tuki myös muille ohjelmointikielille Eclipse Marketplace -palvelun kautta asennettavilla lisäosilla. Tämä tekee Eclipse IDE:stä erittäin mukautettavan ja hyvin laajennettavissa olevan integroidun kehitysympäristön. (Eclipse 2016.)

Erilaisten Eclipse IDE.n pakettien lisäksi Eclipse IDE:stä on saatavilla useita eri julkaisuja. Yritys käyttää tällä hetkellä Eclipse IDE:n Luna-julkaisua, joka on julkaistu 25.6.2014. Eclipse-säätiö tuottaa Eclipse IDE:stä uuden julkaisun vuosittain koordinoidulla aikataululla. Esimerkiksi Luna-julkaisua edeltävä Kepler on julkaistu 26.6.2013 ja Luna-julkaisua seuraava Mars on julkaistu 24.6.2015. (Simultaneous Releases 2016.)

Eclipse IDE:n julkaisuun sisältyy myös muita projekteja, kuten Luna-julkaisuun sisältyvä M2Eclipse, eli integraatio Apache Mavenille. Näiden sisältyvien projektien versionumero myös kasvaa Eclipse IDE:n julkaisujen myötä. Esimerkiksi Luna-julkaisuun sisältyy M2Eclipse 1.5.0 -versio, kun taas tällä hetkellä uusimpaan Mars-julkaisua seuraavaan 22.6.2016 julkaistuun Neon-julkaisuun sisältyy M2Eclipse 1.7.0 -versio. (M2Eclipse 1.5. 2016; M2Eclipse 1.7. 2016; Neon 2016.)

Yrityksen käyttämistä työkaluista keskeisessä asemassa ohjelmistokehitysprojektien osalta kuitenkin on GWT, joka on eri työkaluja sisältävä työkalupakki jo itsessään. GWT käsittää myös tässä opinnäytetyössä läpikäydyt sovelluskehykset: GWT:n Ui-Binder- ja Observis Oy:n xUI -sovelluskehykset. GWT on ratkaisu monien suurien *web*-sovellusprojektien kehittämiseen ja ylläpitämiseen.

3 GWT

GWT eli Google Web Toolkit 1.3 RC1-julkaisuehdokasversio julkaistiin 12.12.2006 ensimmäistä kertaa täysin avoimena lähdekoodina (The GWT Release Notes 2016; Versions 2016). Janakiraman (2013) kirjoitti blogitekstissään, että GWT on mm. siirtynyt vuoden 2012 aikana Googlen omistamasta tuotteesta täysin avoimen lähdekoodin projektiksi. Myös GWT:n verkkosivuilla ilmoitetaan GWT:n olevan 24.10.2012 julkaistusta 2.5-versiosta eteenpäin virallisesti avoimen lähdekoodin projekti (GWT pronounced gwit 2016; Versions 2016). Tämän opinnäytetyön kirjoitushetkellä uusin vakaa julkaisuversio GWT:stä oli 2.7.0, joka julkaistiin 20.11.2014. Uusin vakaa GWT 2.8.0 -versio julkaistiin 20.10.2016, jolloin tämän opinnäytetyön kirjoitusprosessi oli lopuillaan. (Versions 2016.) Pääsin työn aikana tutustumaan ja käyttämään 29.9.2016 julkaistua RC3-julkaisuehdokasversiota, joka pitää kuitenkin sisällään samat ominaisuudet kuin uusin vakaa 2.8.0-versio. RC3-julkaisuhedokasversion edeltäjä RC2 julkaistiin 11.8.2016 sisältäen lähinnä korjauksia 29.7.2016 julkaistuun RC1-julkaisuehdokasversioon. (The GWT Release Notes 2016.) Janakiraman (2016) ilmoitti blogitekstissään uusimman GWT 2.8.0 -version olevan yksi suurimpia päivityksiä tähän asti sisältäen mm. sulavan yhteistoimivuuden JavaScript-koodin kanssa JsInterop 1.0:n kautta, täyden Java 8 tuen, CSS3 tuen GSS-ominaisuuden kanssa, sekä paljon muita korjauksia ja suorituskykyyn liittyviä parannuksia.

GWT on nimensä mukaisesti työkalusarja, joka tarjoaa keinon mm. AJAX-sovelluksien kirjoittamiseen Javalla. GWT:tä ei ole tarkoitettu hyödyntämään Javan ajoympäristöä, vaan sen tarkoituksena on hyödyntää Javaa ohjelmointikielenä. (Tacy ym. 2013, 3.) Kaikessa yksinkertaisuudessaan GWT kääntää kirjoitetun Java-koodin selainkohtaisesti eri selaimissa toimivaksi JavaScript-koodiksi ja GWT:n tuottama JavaScript-koodi on Tacyn ym. (2013, 3) mukaan vieläpä erittäin hyvää (liite 1).

3.1 Syitä käyttöönottoon

Syitä GWT:n ottamisesta käyttöön ohjelmistokehityksen kannalta voi olla monia. GWT mahdollistaa kuitenkin Javan käyttämisen pääasiallisena ohjelmointikielenä JavaScript-ohjelmointikielen sijaan selaimessa toimivien *web*-sovellusten toteuttamiseen. Mielestäni Java on ohjelmointikielenä erittäin vahva, ja etenkin Javalle luontaista olio-ohjelmointityyliä käyttämällä laajatkin ohjelmistokehitysprojektit pysyvät selkeämpinä ja helpommin hallittavina kuin puhtaasti JavaScript-ohjelmointikieleen pohjautuvat ohjelmistokehitysprojektit. Observis Oy:n käyttämä Eclipse IDE, eli integroitu kehitysympäristö tukee kokemukseni perusteella Javaa ja sen projektirakenteita erinomaisen hyvin ja tarjoaa myös erittäin hyvät työkalut Java-koodin debuggaukseen.

Observis Oy:lle ohjelmistoja kehittävä kokenut ohjelmistokehittäjä Ville Venäläinen (2016) kertoo, että ohjelmistokehityksen kannalta ohjelmistojen laatuun ja hallittavuuteen liittyen Java on ohjelmointikielenä parempi vaihtoehto JavaScript-ohjelmointikielelle. Java on vahvasti tyypitetty ohjelmointikieli, mikä vähentää ongelmia jo merkittävästi. Tämän lisäksi sovelluksen lähdekoodi on laadukkaampaa ja vähemmän virhealtista. Eri selaimet ja selainversiot muodostavat hyvin monimuotoisen JavaScript-ohjelmointikielen ja murteiden joukon, joiden hallitseminen ohjelmistokehityksessä on haastavaa. GWT kuitenkin huolehtii Java-koodin kääntämisestä selainkohtaisesti optimoiduksi JavaScript-koodiksi, jolloin ohjelmistokehittäjän ei tarvitse itse huolehtia siitä. Tämä vaikuttaa merkittävästi ohjelmiston laatuun ja sen kehityksen tuottavuuteen. (Venäläinen 2016.)

Yhtenä mahdollisena syynä käyttää GWT:tä *web*-sovelluksien kehittämiseen Tacyn ym. (2013, 3–4) mukaan voisi hyvinkin olla se, että maailmassa on paljon enemmän taitavia Java-kehittäjiä verrattuna kokeneisiin JavaScript-kehittäjiin. Ohjelmakoodin kirjoittaminen Javalla ei välttämättä kuitenkaan ole ainoa syy ottaa GWT:tä käyttöön, sillä sen mukana tulee paljon ominaisuuksia, joista Tacy ym. (2013, 4) mainitsee seuraavia: täysi kokoelma widgettejä, joustava RPC-tuki, sisäänrakennettu sovelluksen kansainvälistäminen, käännetyn koodin ”obfuskointi” ja ”minifiointi”, kuvien paketointi ja paljon muuta. Tacyn ym. sanoin: ”GWT luotiin helpottamaan suurien selainpohjaisten asiakasohjelmien kehitystä painon ollessa hyvän käyttökokemuksen takaamisessa, ja Javan maailma sen kehittäjien mielestä sopi tämän tavoitteen saavuttamiseen hyvin.” (Mts. 4.)

3.2 Kääntäjä

GWT:n ydin on sen kääntäjä, englanninkieliseltä termiltään *compiler*, joka kääntää Java-lähdekoodin JavaScript-koodiksi muodostaen vastaavan Java-sovelluksen eri selainympäristöissä toimivaksi JavaScript-sovellukseksi (Compiling and Debugging 2016). Tacy ym. (2013, 7) selventää, että kääntöprosessin aikana GWT:n kääntäjä analysoi koodin optimoidakseen sen ja poistaakseen minkä tahansa koodin, joka ei ole sovelluksen tavoitettavissa. Prosessin jälkeen kääntäjä luo JavaScript-koodin tiedostoon, joka on oletuksena ”obfuskoitu” ja ”minifioitu” (mts. 7).

GWT:n kääntäjän ominaisuuksiin kuuluu mm. *deferred binding*, jonka toimii luomalla useita versioita käyttäjän tekemästä koodista kääntämisen aikaan. Esimerkiksi *web*-sovelluksen avautuessa *bootstrap*-lataaja huolehtii, että asiakasohjelman tarvitsee ladata vain yksi näistä luoduista versioista. Jokainen kääntäjän tuottama versio käyttäjän kirjoittamasta koodista on luotu eri selaimiin ja GWT-moduulissa kääntäjälle kerrottuihin ohjeisiin pohjautuen. (Deferred 2016; liite 1.)

Näitä kääntäjän tuottamia versioita kutsutaan permutaatioiksi (Tacy ym. 2013, 539). Permutaatioita on Tacyn ym. (2013, 539–541) mukaan vähintään kuusi, koska GWT käyttää *deferred binding* -ominaisuutta DOM-hallintaan ja se tietää kuusi erityyppistä selainmoottoria ts. selainta. Tapauksessa jossa käytetään sovelluksen kansainvälistämistä kääntämällä se eri kielille ja kääntäjälle on kerrottu tehtäväksi permutaatiot esimerkiksi kolmella eri kielellä, niin permutaatioiden määrä kasvaa silloin kahdeksaantoista. Tämä siksi, koska jokaista selainmoottoria kohden tehdään jokaiselle kielelle oma permutaatio. (Mts. 541.) GWT:n verkkosivuilta (Deferred 2016) käy ilmi, että *deferred binding* -ominaisuutta voi hyödyntää esimerkiksi itse kirjoitetun GWT:n Generator-luokkaa laajentavan koodigeneraattorin kautta. Koodigeneraattorit ovat Tacyn ym. (2013, 7) mukaan myös yksi kääntäjän kätkemistä salaisuuksista.

Koodigeneraattori, Tacyn ym. (2013, 7) käyttämältä englanninkieliseltä nimeltään *code generator*, on GWT:n Generator-luokkaa laajentava Java-luokka, joka herätetään GWT:n kääntäjän toimesta kääntämisen aikaan (Tacy ym. 2013, 7; Deferred 2016; liite 1). Kääntäjä voi herättää yhden tai useamman koodigeneraattorin perustuen siihen, mitä se löytää käyttäjän kirjoittamasta koodista. Tacy ym. antaa kirjassaan esi-

merkkejä erilaisista koodigeneraattoreista, joista olennaisimpana mainittakoon koodigeneraattori, joka muuntaisi templaatin toimivaksi Java-koodiksi. (Tacy ym. 2013, 7.) Tähän esimerkkiin perustuvia koodigeneraattoreita on käytetty myös GWT:n UiBinder-sovelluskehyksen ja Observis Oy:n kehittämän xUI-sovelluskehyksen toteutuksissa.

GWT:n kääntäjässä on myös ominaisuus yhdistellä käyttäjän kirjoittamaa Java-koodia olemassa olevaan JavaScript-koodiin JSNI:n avulla (liite 1). JSNI antaa mahdollisuuden upottaa JavaScript-koodia Java-koodiin. (Tacy ym. 2013, 8.) Näin käyttäjä voi kirjoittaa ja kutsua JavaScript-funktioita Java-metodeina ja myös välittää parametreja Java-koodista JavaScript-koodiin. JSNI:stä on erityisen paljon hyötyä silloin, kun halutaan esimerkiksi välittää Java-koodin tarjoamaa dataa sellaisille JavaScript-kirjastojen tarjoamille funktioille, jotka eivät ole GWT:ssä suoraan tuettuja (kuva 2).

```
private native void setAllAlertsNative(JavaScriptObject alertArray)
/*-{
  if ($wnd.cordova != null) {
    $wnd.cordova.plugins.notification.local.schedule(alertArray);
  }
}-*/;
```

KUVA 2. Esimerkki JSNI-metodista

JSNI toimii antamalla Javassa olevan *native*-avainsanan ja kirjoittamalla Java-metodin sisältö kommenttimerkkien sisään. Javan kääntäjä ei huomioi kommenttimerkkien sisällä olevaa koodia, mutta GWT:n kääntäjä nappaa koodin ja yhdistää ne kääntöprosessissa (liite 1). Kommenttimerkkien sisällä oleva koodi on siis JavaScript-koodia lukuun ottamatta JavaScriptistä mm. löytyviä *window*- ja *document* -objekteja, joita kutsutaan GWT:n edellyttämällä tavalla muuttujien ”$wnd” ja ”$doc” kautta. (Tacy ym. 2013 13–14; Writing Native JavaScript Methods 2016.)

Kääntäjä luo lopuksi eri selaimille optimoitujen JavaScript-koodien lisäksi *bootstrap*-lataajan (liite 1). Se linkitetään sovellukseen HTML-sivulle samaan tapaan kuin mikä tahansa muu JavaScript-tiedosto. Se myös päättelee asiakasohjelman, esimerkiksi selaimen selainmoottorin perusteella mikä Java-koodista käännetty JavaScript-koodi ladataan ja otetaan sovelluksen käyttöön. (Tacy ym. 2013.)

4 SOVELLUSKEHYS KÄYTTÖLIITTYMÄN TOTEUTTAMISEEN

GWT-sovelluksissa käyttöliittymän koodi mukaan lukien on oltava Java-koodia, jotta kääntäjä pystyisi lopuksi kääntämään siitä JavaScript-koodia (liite 1). Pohjimmiltaan GWT-sovellus on kuitenkin verkkosivu, ja luonnollisin tapa kirjoittaa verkkosivuja on tehdä se HTML- ja CSS -tekniikoita käyttäen. GWT:n mukana tuleva UiBinder-sovelluskehys ja Observis Oy:n kehittämä xUI-sovelluskehys antavat tehdä juuri tämän mahdollistaen käyttöliittymän suunnittelemisen ja toteuttamisen erilliselle templaatille. Lisäksi nämä sovelluskehykset pystyvät tekemään sovelluksesta tehokkaamman hyödyntämällä selainten kykyä rakentaa DOM-rakenteita paremmin asettamalla isoja HTML-merkkijonoja *innerHTML*-attribuutteihin kuin joukolla API-kutsuja. Sovelluskehykset myös rohkaisevat säästämään selaimen resursseja tekemällä kevyempien HTML-elementtien käytön mukavammaksi raskaiden widgettien sijaan. (Overview 2016.) Sovelluskehykset siis tarjoavat mahdollisuuden toteuttaa GWT-sovelluksien käyttöliittymän myös deklaratiivisesti.

Usein on hyvä erottaa sovelluksen käyttöliittymä ja muu toiminnollisuus toisistaan, sillä Tacyn ym. (2013, 168) mukaan ohjelmoijat eivät yleensä ole kaikista parhaimpia käyttöliittymän suunnittelijoita ja päinvastoin, käyttöliittymän suunnittelijat parhaimpia ohjelmoijia. Tästä syystä monessa perinteisessä *web*-sovelluskehyksessä on aina jonkinlainen templaattityökalu, joka luo sillan käyttöliittymän suunnittelijan ja ohjelmistokehittäjän välille. UiBinder-sovelluskehys on GWT:n ratkaisu suunnittelijan ja kehittäjän väliselle sillalle antaessaan mahdollisuuden suunnitella ja toteuttaa käyttöliittymä HTML:n kaltaiselle XML-templaatille. (Tacy ym. 2013, 168.)

4.1 UiBinder-sovelluskehys

UiBinder-sovelluskehys on sisältynyt GWT SDK:hon 8.12.2009 julkaisusta GWT 2.0.0 -versiosta lähtien. Se julkistettiin silloin merkittävänä uutena ominaisuutena, joka mahdollisti käyttöliittymän toteuttamisen enimmäkseen deklaratiivisesti. Ennen UiBinder-sovelluskehystä käyttöliittymä GWT-sovelluksissa oli toteutettava luomalla ja kokoamalla widgetit Java-koodissa, mikä vaati paljon koodia. UiBinder-sovelluskehys mahdollisti käyttöliittymän kuvaamisen XML:n avulla, tehden koodista

luettavampaa, helpommin hallittavaa ja ylläpidettävää, sekä nopeammin kehitettävää. (The GWT Release Notes 2016; Versions 2016.)

4.1.1 Hyödyt ja ominaisuudet

UiBinder-sovelluskehys helpottaa ohjelmistokehittäjien ja käyttöliittymäsuunnittelijoiden välistä yhteistyötä tarjoamalla ratkaisun Java-koodin ja käyttöliittymäkoodin erottelemiseen (Guermeur & Unruh 2010, 66); Java-koodi kirjoitetaan UiBinder-luokkaan ja käyttöliittymäkoodi XML-templaattiin. Tästä syystä sovelluskehystä käyttäen kirjoitettu koodi on myös helpommin luettavaa kuin ilman sitä kirjoitettu koodi (mts. 66). Ilman UiBinder-sovelluskehystä käyttöliittymäkoodi on kirjoitettava muun toimintalogiikan kanssa Java-koodiin, mikä on toisaalta suoraviivaista mutta tuloksena se voi olla hyvin vaikealukuista ja ulkoasun kannalta vaikeasti hahmotettavissa ennen koodin ajamista. Sovelluskehys huolehtii myös käyttöliittymäkoodin syntaksin tarkistamisesta, sekä CSS-tyylien validoinnista koodin kääntämisen aikana, mikä osaltaan estää virheitä ja varmistaa, että julistetut CSS-tyylit ovat oikeasti olemassa. (Guermeur & Unruh 2010, 66–67.)

Tacy ym. (2013, 168) mainitsee muista templaattityökaluista JSP:n, joka mahdollistaa käyttöliittymäsuunnittelijan suunnitteleman HTML:n sekoittamisen ohjelmistokehittäjän tarjoaman datan kanssa. Guermeur ja Unruh (2010, 67) huomauttaa, että UiBinder-sovelluskehys poikkeaa muista templaattikielistä, kuten esimerkiksi PHP:stä ja JSP:stä, sillä UiBinder-sovelluskehyksessä ei ole toistorakenteita, ei muuttujia, eikä muutakaan logiikkaa. Se on vain käyttöliittymän kuvaamiseen tarkoitettu kieli (mts. 67).

Tacy ym. (2013, 185) kertoo, että UiBinder-sovelluskehyksellä on kuitenkin oma ilmaisukieliensä, jota käyttämällä on mahdollista suorittaa XML-templaatissa julistettujen kenttien tarjoamia metodeja. Kentän tarjoaman metodin palauttaman tuloksen voi asettaa muiden XML-templaatissa julistettujen elementtien tai widgettien attribuuttien arvoiksi (mts. 185). Tacy ym. (2013, 169) lisää vielä, että yhteistä muiden templaattityökalujen kanssa on se, että muiden templaattityökalujen tapaan UiBinder-sovelluskehys on enemmän kuin vain silta ohjelmistokehittäjän ja käyttöliittymäsuunnittelijan välille, sillä se tarjoaa myös toiminnollisuudet ohjelmistokehittäjille nostaa tuottavuutta ja vähentää kirjoitettavan Java-koodin määrää.

UiBinder-sovelluskehyksen avulla käyttöliittymä on GWT:n Generator-luokkaa laajentavien koodigeneraattoreiden ansiosta mahdollista toteuttaa erilliselle XML-templaatille. Tacy ym. (2013, 567) määrittelee koodigeneraattorit seuraavasti: ”generaattori käynnistyy kääntämisen aikaan, se ottaa tiettyjä resursseja tekemästäsi lähdekoodista ja se luo uudet korvaavat resurssit, joita on käytetty käännöksessä alkuperäisten sijaan.” UiBinder-sovelluskehyksen ytimessä on koodigeneraattori, joka kääntämisen alkuvaiheessa ottaa käyttöliittymäkoodin sisältävän XML-templaatin kääntäen sen Java-koodiksi ja sitoen sen Java-luokkaan. Yhtenä avainasioista UiBinder-sovelluskehyksen toiminnan ymmärtämiseen on tiedostaa kuinka koodigeneraattori tietää miten toimia. (liite 1; Tacy ym. 2013, 181.)

4.1.2 Koodigeneraattori

GWT:n UiBinder-sovelluskehyksellä XML-templaatin liittäminen UiBinder-luokkaan ja sen juurielementin tai -widgetin saaminen sekä XML-templaatissa määriteltyjen widgettien tai elementtien alustaminen vaatii Java-koodissa tehtäväksi kolme asiaa. Ensimmäisenä on luotava Java-rajapintaluokka, joka laajentaa GWT:n UiBinder-rajapintaluokkaa. Tämän jälkeen on kutsuttava GWT-luokasta *create*-metodia luodakseen UiBinder-rajapintaluokasta konkreettisen luokan, joka implementoi ensimmäisenä tehdyn Java-rajapintaluokan. Viimeisenä on kutsuttava *createAndBindUi*-metodia välittäen sille parametrina viittauksen objektiin *this* saadakseen Java-luokan juurielementin tai -widgetin (Tacy ym. 2013, 181; kuva 3).

```
public class HelloPage{
  interface HelloPageUiBinder extends
  UiBinder<DivElement, HelloPage>{}
  private static HelloPageUiBinder uiBinder = GWT
                      .create(HelloPageUiBinder.class);
  private DivElement root;
  @UiField SpanElement nameSpan;
  public HelloPage() {
    this.root = uiBinder.createAndBindUi(this);
  }
  public Element getElement(){
    return this.root;
  }
  public void setName(String name){
    nameSpan.setInnerHTML(name);
  }
  public String getId(){
    return "helloPage";
  }
}
```

KUVA 3. UiBinder-luokka

Tacy ym. (2013, 181–182) selventää, että viimeisenä asiana tehty viittaus objektiin *this* välitetään parametrina *createAndBindUi*-metodille, koska *createAndBindUi*-metodi sitoo XML-templaatissa määritetyt elementit tai widgetit Java-luokassa *UiField*-annotaatiolla varustettuihin kenttiin. Toisena asiana kutsuttava GWT-luokan *create*-metodi luo instanssin UiBinder-rajapintaluokasta. Tämä käynnistää koodigeneraattorin (liite 1), joka kirjoittaa Java-koodin luodakseen XML-templaatissa määritellyn käyttöliittymän ja sitoen sen UiBinder-luokkaan tehden siitä sitojaluokan. Koodigeneraattori tietää ensimmäisenä asiana luodun ja kuvan 3 UiBinder-luokassa näkyvän GWT:n UiBinder-rajapintaluokkaa laajentavan HelloPageUiBinder-rajapintaluokan ansiosta mihin Java-luokkaan XML-templaatissa määritelty käyttöliittymä tullaan viimekädessä sitomaan ja missä itse XML-templaatti sijaitsee. (Mt.)

Kuvan 3 UiBinder-luokan UiBinder-rajapintaluokalle välitetään kaksi tyyppiparametria, joista ensimmäinen on DivElement-tyyppi, jonka *createAndBindUi*-metodi palauttaa juurielementtinä. Toisena välitettävä HelloPage-tyyppi on Java-luokan nimi, josta koodigeneraattori tietää mihin Java-luokkaan XML-templaatti tullaan sitomaan. (kuva 3; Hello World 2016; Tacy ym. 2013, 182.). Kääntämisen alkuvaiheessa tapahtuvan koodigeneraattoreiden ajon aikana (liite 1) koodigeneraattori käy läpi mm. Java-luokasta löytyvät *UiField*-annotaatiolla varustetut kentät, kuten esimerkiksi kuvan 3 UiBinder-luokan SpanElement-tyyppisen *nameSpan*-kentän, jotta se voisi luoda asianmukaista koodia *createAndBindUi*-metodia varten (Tacy ym. 2013, 182). Jokaista

UiField-annotoitua kenttää kohden XML-templaatista on löydyttävä *ui:field*-attribuutti, jonka arvona on UiBinder-luokassa määritellyn *UiField*-annotoidun kentän nimi. Esimerkkinä kuvan 4 XML-templaatissa olevan *span*-elementin *ui:field*-attribuutin arvona on kuvan 3 UiBinder-luokassa *UiField*-annotoidun SpanElement-tyyppisen *nameSpan*-kentän nimi.

```
<!DOCTYPE ui:UiBinder SYSTEM
"http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder">
  <div>
    Hello, <span class="name-span" ui:field="nameSpan" />
  </div>
</ui:UiBinder>
```

KUVA 4. UiBinder-luokan XML-templaatti

Koodigeneraattori osaa löytää tiedoston, jossa XML-templaatti sijaitsee etsimällä sitä UiBinder-luokan UiBinder-rajapintaluokalle toisena tyyppiparametrina annetun tiedon, eli Java-luokan nimen perusteella. Oletuksena koodigeneraattori siis luottaa käytäntöön, että XML-tiedosto on nimetty samalla nimellä kuin toisena annettu tyyppiparametri. (Tacy ym. 2013, 182.) Tämä tarkoittaa esimerkiksi sitä, että kuvan 3 tapauksessa kuvan 4 XML-templaatin tiedoston nimi täytyy olla ”HelloPage.ui.xml”. Tämä on kuitenkin mahdollista yliajaa käyttämällä *UiTemplate*-annotaatiota, joka mahdollistaa XML-tiedoston nimeämisen halutun mukaiseksi (kuva 5). XML-templaatin on kuitenkin sijaittava samassa paketissa tai kansiossa kuin sen omistava Java-luokka, ellei parametrina anna vain eteenpäin suuntautuvaa relatiivista polkua, esimerkiksi ”templates/FrontPage.ui.xml”. Tällä tavoin on myös mahdollista jakaa sama XML-templaatti usean eri UiBinder-luokan kesken (Tacy ym. 2013, 182.)

```
@UiTemplate("FrontPage.ui.xml")
interface HelloPageUiBinder extends UiBinder<DivElement, HelloPage> {}
```

KUVA 5. *UiTemplate*-annotaation käyttäminen UiBinder-luokassa

Näiden toimien jälkeen kuvan 3 UiBinder-luokka on mahdollista instantioida esimerkiksi EntryPoint-luokassa käyttämällä *new*-operaattoria, joka herättää luokan rakentajan (kuva 6). Kuvan 3 UiBinder-luokan rakentajassa kutsuttu *createAndBindUi*-metodi palauttaa sille viittauksena annetun *this*-objektin, eli objektin itsensä juurielementin ja se asetetaan DivElement-tyyppiseen muuttujaan, joka on saatavilla myöhempää käyttöä varten kutsumalla kuvan 3 UiBinder-luokassa määritettyä *getElement*-

metodia. Instantioinnin jälkeen EntryPoint-luokassa voidaan käyttää HelloPage-tyyppistä *helloPage*-objektia ja sen tarjoamia metodeja esimerkiksi kuvassa 6 näkyvällä tavalla.

```
public class HelloWorld implements EntryPoint {
  @Override
  public void onModuleLoad() {
    HelloPage helloPage = new HelloPage();
    helloPage.setName("World");
    addPage(helloPage.getId(), helloPage.getElement());
  }
  public static void addPage(String pageId, Element page){
    RootPanel.get(pageId).getElement().appendChild(page);
  }
}
```

KUVA 6. GWT-sovelluksen EntryPoint-luokka

EntryPoint-luokan *onModuleLoad*-metodi kutsutaan ensimmäisenä *bootstrap*-lataajan toimesta kun permutaatio on ladattu (Tacy ym. 2013, 26; kuva 6). Kuvan 6 esimerkissä *helloPage*-objektin juurielementti asetetaan HTML-dokumentista *getId*-metodin palauttamaan *id*-attribuutilla löytyvään elementtiin. Sovelluksen käynnistyessä HTML-dokumentissa määritellyn *id*-attribuutin arvona olevalla ”helloPage”-merkkijonolla varustetun elementin sisälle siis lisätään kuvan 4 XML-templaatin mukainen HTML-elementti.

Mikäli kuvassa 3 näkyvälle UiBinder-rajapintaluokalle antaisi ensimmäisenä tyyppiparametrina DivElement-tyypin sijaan Widget-tyypin ja kuvan 4 XML-templaatin *div*-elementin vaihtaisi GWT:n HTMLPanel-widgetiksi, niin silloin kuvan 3 UiBinder-luokan rakentajassa *uiBinder*-objektista kutsuttu *createAndBindUi*-metodi palauttaisi *this*-objektin juurena olevan Widget-tyyppisen objektin. Widgetit on kuitenkin ensin tuotava XML-templaattiin XML-nimiavaruuden mukana (Tacy ym. 2013, 172).

4.1.3 XML-nimiavaruudet

Kuvien 2 ja 3 UiBinder-luokka ja sen XML-templaatti perustuvat HTML-elementteihin. XML-templaatissa on kuitenkin mahdollista käyttää GWT:n widgettejä tuomalla ne erikseen XML-nimiavaruuden mukana. UiBinder-sovelluskehystä käyttäessä kaikki HTML sisältö tulee olla XML-templaatin *ui:UiBinder*-juurielementin sisällä. Tämän juurielementin attribuutteina on yleensä ainakin yksi XML-nimiavaruus,

jota käytetään jokaisessa UiBinder-sovelluskehyksen XML-templaateissa. (Tacy ym. 2013, 172.) Ensimmäisenä kuvassa 7 julistettu nimiavaruus kertoo, että kaikki *ui:*-alkuiset XML-elementit tulevat tämän nimiavaruuden mukana, jopa itse XML-templaatin *ui:UiBinder*-juurielementti. Toisena nimiavaruutena kuvassa 7 tuotu *xmlns:g*-attribuutin arvona oleva nimiavaruus osoittaa Java-pakettiin, jossa GWT:n widgetit sijaitsevat (kuva 7; Tacy ym. 2013, 172).

```
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
  xmlns:g="urn:import:com.google.gwt.user.client.ui">
</ui:UiBinder>
```

KUVA 7. XML-templaatin juurielementti ja XML-nimiavaruudet

Tacyn ym. (2013, 173) mukaan XML-nimiavaruus on tapa liittää joukko XML-elementtejä nimeen, joka on URI. UiBinder-sovelluskehyksen kanssa on mahdollista käyttää XML-templaatissa joukkoa merkintöjä, kuten ”<UiBinder>” ja ”<style>”, joiden nimiavaruus on kuvassa 7 näkyvän *xmlns:ui*-attribuutin arvona oleva nimiavaruus. Kun XML-templaattiin lisätään toinen joukko merkintöjä, kuten ”<HTMLPanel>” ja ”<Button>”, niin ne eivät ole osa *xmlns:ui*-attribuutin arvona olevaa nimiavaruutta, vaan ne ovat osa kuvassa 7 näkyvää *xmlns:g*-attribuutin arvona olevaa nimiavaruutta.

Käyttääkseen näitä kahta nimiavaruutta samassa XML-templaatissa XML-jäsentäjän täytyy pystyä kertomaan ero näiden kahden eri merkintäjoukon välille. Tämä tapahtuu julistamalla jokainen nimiavaruus ja antamalla niille erilaiset etuliitteet, joita käytetään esimerkiksi *xmlns:g*-attribuutin arvona annetusta nimiavaruudesta löytyvän GWT:n HTMLPanel-widgetin määrittelemiseen XML-templaatissa *g:HTMLPanel*-elementtinä (Mts. 173; kuva 7; kuva 8). Tämä siis tuo kaikki GWT:n widgetit XML-templaattiin, joita on mahdollista käyttää kirjoittamalla XML-elementti, joka on *g:*-alkuinen kuvan 8 esimerkin mukaisesti. GWT-widgetin Java-luokan nimi seuraa *g:*-etuliitettä. (Mts. 173.)

```
<g:HTMLPanel>
  <g:Button text="Button-widgetti"></g:Button>
  <g:Label text="Label-widgetti"></g:Label>
</g:HTMLPanel>
```

KUVA 8. HTMLPanel-widgetti

Kuvan 8 HTMLPanel-widgetti voi sisältää muiden widgettien, kuten Button-widgetin lisäksi myös HTML-elementtejä. Widgeteillä on kuitenkin HTML-elementeistä poikkeavia ominaisuuksia, kuten esimerkiksi attribuutit. HTML-elementille on mahdollista määrittää tyylit antamalla CSS-luokan nimi *class*-attribuutin arvona, kun taas widgeteille tyylit voidaan määrittää käyttämällä *styleName*-attribuuttia *class*-attribuutin sijaan. Sääntönä on, että widgetille voidaan asettaa sellaisia attribuutteja ja niiden arvoja, joille sillä itsellään on olemassa *set*-metodi. Tämä tarkoittaa, että kuvan 8 widgeteille voisi asettaa *styleName*-attribuutin ja sille CSS-luokan nimen arvona, koska niiltä kaikilta löytyy *setStyleName*-metodi. (Tacy ym. 2013, 174.)

HTML-elementtien ja widgettien attribuuttien arvoja on myös mahdollista asettaa XML-templaatissa käyttämällä UiBinder-sovelluskehyksen ilmaisukieltä. Tämä tapahtuu kutsumalla *get*-metodeja *ui:field*-attribuutin arvona olevasta kentästä tai kentästä, joka on luotu viittauksena ulkoiseen objektiin. Kentän tyyppi ja sen tarjoamat metodit riippuvat HTML-elementin, widgetin tai XML-templaattiin viittauksena luodun kentän ulkoisen objektin tyypistä. (Tacy ym. 2013, 185.)

4.1.4 UiBinder-ilmaisukieli

Mikä tahansa templaattityökalu tai -teknologia ei olisi kokonainen ilman ilmaisukieltä, josta Tacy ym. (2013, 184) käyttää englanninkielistä termiä *expression language*. UiBinder-ilmaisukieli mahdollistaa yksinkertaisten lausekkeiden kirjoittamisen XML-templaatissa. Tacy ym. (mt.) myös vahvistaa Guermeurin ja Unruhin (2010, 67) esittämän havainnon, että UiBinder-ilmaisukieli ei tarjoa *if*-, *else*-rakenteita tai toistorakenteita. Tacyn ym. (2013, 185) listaa mukaillen UiBinder-ilmaisukielellä on kuitenkin mm. seuraavia ominaisuuksia ja piirteitä:

- Metodien suorittaminen ainoastaan XML-templaatin sisällä julistetuista kentistä.
- Ilmaisukielen käyttö ainoastaan elementtien attribuuttien arvojen asettamiseen XML-templaatissa.
- Metodin on palautettava oikean tyyppinen arvo attribuutille, jota yritetään asettaa.
- Metodien kutsuminen metodin palauttamasta arvosta on sallittua.

- Ilmaisukielen lauseke on rajattu aaltosulkein

Kuvan 9 esimerkissä *g:Label*-elementin *text*-attribuutin arvoksi on asetettu merkkijonona ”Etunimi”. Tämä on mahdollista, koska *g:Label*-elementti viittaa GWT:n Label-widgettiin ja sille on määritelty *setText*-metodi. Elementille on myös asetettu *ui:field*-attribuutti ja sille arvo ”labelWidget”. Tällä julistetaan siis kyseiseen Label-widgettiin viittaava Label-tyyppinen *labelWidget*-kenttä. Tätä *labelWidget*-kenttää on nyt mahdollista käyttää XML-templaatissa kutsuen sen sisältämiä metodeja. (Kuva 9.)

UiBinder-ilmaisukieltä voidaan myös käyttää HTML-elementtien kanssa (Tacy ym. 2013, 185). Kuvan 9 esimerkissä asetetaankin *input*-elementin *placeholder*-attribuutin arvoksi *labelWidget*-kentän sisältämän *getText*-metodin palauttama merkkijono ”Etunimi”, joka siis on *g:Label*-elementissä asetettu *text*-attribuutin arvo. Kuvassa 9 myös näkyy muita *labelWidget*-kentän sisältämiä metodeja, joita on mahdollista käyttää UiBinder-ilmaisukielessä *getText*-metodin tapaan.

```
1  <!DOCTYPE ui:UiBinder SYSTEM "http://dl.google.com/gwt/DTD/xhtml.ent">
2  <ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
3      xmlns:g="urn:import:com.google.gwt.user.client.ui">
4      <ui:with type="fi.observis.example.client.resources.Resources"
5          field="res">
6      </ui:with>
7      <g:HTMLPanel styleName="{res.style.container} {res.style.element}">
8          <g:Label ui:field="labelWidget"
9              text="Etunimi">
10         </g:Label>
11         <input type="text" placeholder="{labelWidget.}"></input>
12         <g:Button ui:field="buttonWidget"
13             styleName="{res.style.element_button}"
14             text="Button-widgetti">
15         </g:Button>
16     </g:HTMLPanel>
17 </ui:UiBinder>
```

getText() : String - Label
asEditor() : LeafValueEditor<String> - Label
getDirection() : Direction - Label
getStyleName() : String - UIObject
getStylePrimaryName() : String - UIObject

KUVA 9. UiBinder-ilmaisukielen käyttö XML-templaatissa

XML-templaatissa julistettujen widgettien tarjoamien metodeihin pääsyn lisäksi on mahdollista luoda viittauksia ulkoisiin objekteihin kuvissa 9 ja 10 näkyvien *ui:with*-elementtien avulla (Tacy ym. 2013 185). Kuvan 9 esimerkissä *ui:with*-elementin *type*-attribuutin arvona on paketti ja Java-luokan nimi, joka tarjoaa mm. GWT-sovelluksen tyylit ja muut resurssit. Attribuutin *field* arvona taas on ”res”, joka on XML-templaatissa käytettävän kentän nimi. Kuvan 9 esimerkissä asetetaan mm. *g:HTMLPanel*-elementin *styleName*-attribuutin arvoksi *res*-kentän *style*-metodin tarjoamien *container*- ja *element* -metodien palauttamat CSS-luokkien nimet merkkijonoina.

Kuvan 10 esimerkissä taas on käytetty ulkoista Settings-luokkaa, jossa on määritelty String-tyyppinen *placeholderText*-kenttä. Kentän arvoksi on alustettu merkkijono ”Sukunimi” (kuva 10). Settings-luokkaan on myös määritetty kentän arvon asettava *set*-metodi, sekä sen arvon palauttava *get*-metodi, jota on mahdollista käyttää XML-templaatissa UiBinder-ilmaisukielen avulla kuvan 10 esimerkin mukaan.

```
1  <!DOCTYPE ui:UiBinder SYSTEM "http://dl.google.com/gwt/DTD/xhtml.ent">
2  <ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
3      xmlns:g="urn:import:com.google.gwt.user.client.ui">
4      <ui:with type="fi.observis.example.client.settings.Settings"
5          field="settings">
6      </ui:with>
7      <g:HTMLPanel>
8          <g:Label ui:field="labelWidget"
9              text="{settings.getPlaceholderText}">
10         </g:Label>
11         <input type="text"
12             placeholder="{settings.}">
13         </input>
14     </g:HTMLPanel>
15 </ui:UiBinder>
```

```
getPlaceholderText() : String - Settings
toString() : String - Object
getClass() : Class<?> - Object
```

```
1  package fi.observis.example.client.settings;
2
3  public class Settings {
4
5      private String placeholderText = "Sukunimi";
6
7      public String getPlaceholderText(){
8          return placeholderText;
9      }
10
11     public void setPlaceholderText(String placeholderText){
12         this.placeholderText = placeholderText;
13     }
14 }
```

KUVA 10. Yläpuolella XML-templaatissa on viitattu ulkoiseen objektiin ja myös käytetty sitä. Alapuolella on viitatun ulkoisen objektin Java-luokka.

Tacyn ym. (2013, 186) mukaan *ui:with*-elementin käyttäminen XML-templaatissa on widgettien julistamisen kanssa samantapaista; UiBinder-luokkaan ei tarvitse lisätä mitään niiden käyttämiseksi (mts. 186). Sama koskee myös julistettuja HTML-elementtejä. Oletuksena UiBinder-sovelluskehys luo uuden instanssin viitatusta objektista ja käyttää sitä, kuten se tekee myös widgettien tai HTML-elementtien kohdalla (Tacy ym. 2013, 186). Tämä tarkoittaa sitä, että XML-templaatissa viitatun ulkoisen objektin ei tarvitse sisältää staattisia metodeja, vaan UiBinder-sovelluskehys instantioi *type*-attribuutin arvona asetetusta Java-luokasta uuden objektin, jota on mahdollista

käyttää myöhemmin *field*-attribuutin arvona annetun kentän nimen kautta, kuten kuvan 10 esimerkki osoittaa.

Observis Oy:n kehittämä xUI-sovelluskehys tarjoaa mahdollisuuden UiBinder-sovelluskehyksen tapaan toteuttaa käyttöliittymä erilliselle templaatille. Yhteistä näille sovelluskehyksille on siis kaksi tiedostoa, joiden kanssa sen käyttäjä on tekemisissä: käyttöliittymäkoodin sisältävä templaattitiedosto ja Java-luokka. Templaattitiedoston sisältämä käyttöliittymäkoodi tullaan kääntämisen yhteydessä myös sitomaan sen omistavaan Java-luokkaan (Tacy ym. 2013, 181).

4.2 xUI-sovelluskehys

Observis Oy:n teknologisen johtajan Ville Kanervan (2016a) mukaan ennen yrityksen vuosina 2013 ja 2014 kehittämää xUI-sovelluskehystä tietotaito GWT:stä, UiBinder-sovelluskehyksestä ja sen mahdollisuuksista yrityksessä oli vähäistä. Myös *web*-käyttöliittymäpuolen osaajia oli vain vähän verrattuna Java-osaajiin, joten tästä syystä oli pienempi kynnys lähteä toteuttamaan GWT-sovelluksien käyttöliittymää valmiiden käyttöliittymäkomponenttien, eli widgettien avulla (Kanerva 2016a). GWT-sovelluksien käyttöliittymä siis toteutettiin ilman erillistä sovelluskehystä kirjoittamalla widgetit Java-koodiin.

Observis Oy:n käyttöliittymäsuunnittelijan Tuomas Kämpin (2016) mukaan käyttöliittymäsuunnitelman toteuttaminen GWT:n widgeteillä oli työlästä ja vei todella paljon aikaa; käännetyn koodin HTML-rakenteet olivat monessa kohtaa väärin oikean asettelun kannalta ja tästä syystä käyttöliittymä oli toteutettava jälkeenpäin vielä toiseen kertaan. Hyvin usein tilanne myös oli se, että käyttöliittymästä ei ollut edes olemassa suunnitelmaa; oli olemassa vain pelkkä toiminnollisuus, jonka pohjalta käyttöliittymä muovattiin. (Kämppi 2016.)

4.2.1 Tarve sovelluskehyksen kehittämiseen ja sen tuomat hyödyt

Yrityksessä kokeiltiin GWT:n widgettien lisäksi mm. käyttöliittymäkomponenttien piirtämistä RAD-työkalulla; näistä molemmista paljastui lopputuloksena tehoton HTML-dokumentti ja huono ulkoasun muokattavuus. Myös GWT:n mobiiliwidgettejä kokeiltiin ja todettuaan ne huonoiksi, kehitettiin UiBinder-sovelluskehyksen pohjalta

xUI-sovelluskehys, jonka avulla käyttöliittymäkehitykseen saatiin helpotusta. Lopputuloksena oli tehokkaampi ja ulkoasuun liittyviltä ominaisuuksiltaan parempi HTML-dokumentti. (Kanerva 2016a.)

xUI-sovelluskehyksen kehitys ajoittui samaan aikaan mm. erään toisen ohjelmistokehitysprojektin sovelluskehyksen kehityksen kanssa, jossa pyrittiin toteuttamaan dynaamisen sovelluksen luonti alusta alkaen. Yrityksessä kokeiltiin GWT:n UiBinder-sovelluskehystä ja sitä päätettiin laajentaa, sekä helpottaa sen aikaisen version käyttöä kehittämällä xUI-sovelluskehys sen päälle. Tähän liittyi myös GIN-sovelluskehyksen käyttöönotto. (Kanerva 2016b.) xUI-sovelluskehys oli siis yritykselle sen ajan tarpeisiin nähden ajanmukainen ja tarpeellinen.

xUI-sovelluskehys mahdollisti toimintatavan, jossa ohjelmistokehittäjä pystyi aloittamaan GWT-sovelluksen toiminnollisuuden toteuttamisen ilman suunnitelmaa käyttöliittymästä ja käyttöliittymäsuunnittelijalle riitti sovelluksen HTML-rakenteen muokkaaminen. Käyttöliittymäsuunnittelijan ei siis esimerkiksi enää tarvinnut mennä tutkimaan Java-koodia etsien sieltä oikeaa widgettiä, lisäten sille oikeita CSS-luokkia ja vaihtaen widgettien järjestystä. GWT-sovelluksen kehittäminen oli myös mahdollista aloittaa käyttöliittymäsuunnittelijan laatiman HTML-demonstraation pohjalta, jonka pystyi toteuttamaan suoraan GWT-sovellukseen ohjelmistokehittäjän tehdessä sille toiminnollisuuden. (Kämppi 2016.)

4.2.2 Ominaisuudet

Kanervan (2016b) mukaan xUI-sovelluskehyksen templaatin kohdalla voidaan puhua merkkauskielenä XHTML:stä. HTML:n kaltaisen merkintätavan lisäksi xUI-sovelluskehyksen XHTML-templaatissa on mahdollista käyttää myös standardin mukaisia HTML-entiteettejä, joita puhtaassa XML-templaatissa ei voisi käyttää. Esimerkiksi GWT:n UiBinder-sovelluskehyksessä HTML-entiteettejä käyttääkseen ne on tuotava XML-templaattiin erikseen *DOCTYPE*-lausekkeen mukana. (Tacy ym. 2013, 171–172.)

Vaikka xUI-sovelluskehys jakaa samoja tavoitteita UiBinder-sovelluskehyksen kanssa ja sen perusidea on erotella UiBinder-sovelluskehyksen tapaan käyttöliittymä ja muu toimintalogiikka toisistaan, niin xUI-sovelluskehys vaatii kuitenkin asioiden tekemistä

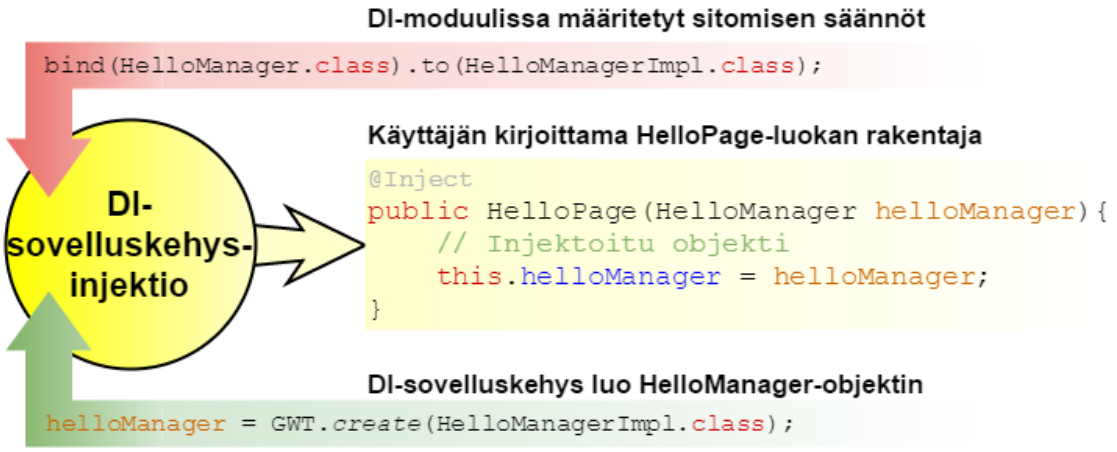
käytännössä hieman eri tavalla. Se käyttää GWT:n generaattoreiden lisäksi mm. riippuvuuksien injektointia asiakasohjelmissa toimivalla ja Guice-sovelluskehykseen pohjautuvalla GIN-sovelluskehyksellä. GIN-sovelluskehys puolestaan hyödyntää paljon kahta edistynyttä GWT:n tekniikkaa: *deferred binding* -ominaisuutta ja generaattoreita (xUI 2016; Tacy ym. 2013, 517; Tacy ym. 2013, 537).

xUI-sovelluskehys käyttää myös Apachen Java-pohjaista Velocity-templaattimoottoria koodigeneraattoreiden ajon yhteydessä. Avainasiana xUI-sovelluskehyksen ymmärtämiseen koodigeneraattoreiden lisäksi on tiedostaa kuinka riippuvuuksien injektointi toimii GWT:ssä ja kuinka se on toteutettu xUI-sovelluskehyksessä. Riippuvuuksien injektointi on hyvin merkittävä osa xUI-sovelluskehyksen toimintaa.

4.2.3 Riippuvuuksien injektointi

Riippuvuuksien injektointi, eli DI on tekniikka, joka mahdollistaa riippuvuuksien kuvaamisen asioiden välillä. Määritetyt objektit injektoidaan DI-sovelluskehyksen avulla koodiin, jotta niiden luomista ei tarvitsisi erikseen kirjoittaa. (Tacy ym. 2013, 516; kuva 11.) DI antaa mahdollisuuden työskennellä rajapintaluokkien parissa ja se antaa myös koodigeneraattoreiden, sekä mukautettujen luokkien lataajien tarjota konkreettiset luokkatyypit (mts. 518).

Avainasiana on ymmärtää, että xUI-sovelluskehys hyödyntää Apachen Velocity-templaattimoottoria, GIN-generaattoria ja muita GWT:n tarjoamia generaattoreita konkreettisten Java-luokkien luomiseen (liite 2), ja ne luodaan vasta kääntämisen aikana koodigeneraattoreiden ajon yhteydessä (liite 1), joten niitä ei ole vielä olemassa käyttäjän kirjoittaessa Java-koodia. xUI-sovelluskehyksen tarjoamien rajapintaluokkien ansiosta käyttäjän kirjoittama Java-koodi on kuitenkin sen kirjoittamishetkellä syntaksiltaan oikein; esimerkiksi kuvassa 11 DI-sovelluskehys löytää DI-moduulissa määritetyt sitomisen säännöt HelloManager-rajapintaluokan ja HelloManagerImpl-luokan väliltä ja HelloManager-viittaustyypin nähdessään se injektoi Inject-annotoidun rakentajan parametrit (Tacy ym. 2013, 525–531).



KUVA 11. Tacya ym. (2013, 517) mukaillen kaaviossa on kuvattu DI-sovelluskehyksen toimintaa GWT:ssä

DI-sovelluskehyksistä GWT:ssä voidaan käyttää palvelinpuolella toimivaa Guice-sovelluskehystä ja asiakasohjelmissa toimivaa GIN-sovelluskehystä. Guice-sovelluskehys on suosittu DI-sovelluskehys Javassa, jota voidaan helposti käyttää missä tahansa palvelinpuolen Java-koodissa (Tacy ym. 2013, 522). Guice-sovelluskehys tukeutuu kuitenkin vahvasti ajonaikaisen Java-ohjelman ominaisuuteen tutkailla itseään ja manipuloida ohjelman sisäisiä ominaisuuksia (Tacy ym. 2013, 257; McCluskey 1998). Tästä ominaisuudesta käytetään Javassa englanninkielistä termiä *reflection*, joka on loistava ominaisuus juuri palvelinpuolen GWT-koodissa, koska siellä *reflection* on hyvin saatavilla. Ajonaikainen *reflection*-ominaisuus ei ole kuitenkaan käytettävissä GWT:n asiakasohjelmien toimintaympäristössä, joten Guice-sovelluskehystä ei voi sellaisenaan käyttää GWT:n asiakasohjelmissa tapahtuvaan injektointiin. GWT *Injection*, eli GIN-sovelluskehys kuitenkin suorittaa tarvittavat injektoinnit GWT-koodin kääntämisen yhteydessä, jonka aikana Javan *reflection*-ominaisuus on käytettävissä. (Tacy ym. 2013, 527.)

Tacy ym. (2013, 527) tarkentaa, että Guice-sovelluskehys mm. injektoi tarvittavat luokat toteuttamalla mukautetun luokan lataajan, kun GIN-sovelluskehys taas injektoi tarvittavat keskitason muodot, jotka sitten käännetään tuloksena syntyvään ohjelmaan. GIN perustuu Guice-sovelluskehykseen ja sovelluskehykset ovat lähes identtisiä. GIN-sovelluskehys kuitenkin edellyttää AbstractGinModule-luokan laajentamista GIN-moduulissa Guice-sovelluskehyksen moduulissa laajennettavan AbstractModule-luokan sijaan. Myös Ginjector-rajapintaluokkaa laajentava injektorina toimiva rajapin-

taluokka on kirjoitettava itse ja siitä täytyy luoda GWT-luokan *create*-metodin palauttama instanssi ajaakseen GWT:n generaattoreita. (Mts. 527.) xUI-sovelluskehys tekee juuri tämän tarjotakseen instanssin kääntämisen aikana luodusta injektorista, jossa luetellaan *get*-metodeja objekteille, jotka halutaan injektoida (kuva 12; Tacy ym. 2013, 530).

Tacy ym. (2013, 530) selvittää myös, että injektorina toimiva rajapintaluokka on luotava itse, koska GIN-sovelluskehys ei tarjoa mahdollisuutta Guice-sovelluskehyksellä onnistuvaan automaattiseen injektorin luomiseen. Osa GIN-sovelluskehyksen tekemistä ”taioista” on, että se käyttää sisäisesti GWT-luokasta löytyvää *create*-metodia instantioidakseen objektit, jotka injektoidaan kääntämisen aikana vaikka sitomiselle ei olisi määritetty sääntöjä (mts. 530). Tacyn ym. (2013, 530) mukaan tämä tarkoittaa sitä, että GIN-generaattori luo vaaditun Java-luokan ja ratkoo riippuvuudet rekursiivisesti.

```
@GinModules(GeneratedGinModule.class)
public interface GeneratedGinInjector extends Ginjector {
  public static GeneratedGinInjector INSTANCE = GWT
           .create(GeneratedGinInjector.class);
  FactoryManager getFactoryManager();
}
```

KUVA 12. xUI-sovelluskehyksen injektori (xUI 2016)

Injektorina toimiva GeneratedGinInjector-rajapintaluokka sisältää itsensä tyyppisen ”INSTANCE”-nimisen muuttujan, johon asetetaan GWT-luokasta löytyvän *create*-metodin palauttama instanssi itsestään (kuva 12). Kuvan 12 GeneratedGinInjector-rajapintaluokka sisältää myös *getFactoryManager*-metodin, joka palauttaa injektoidun FactoryManagerImpl-objektin kuvan 13 GeneratedGinModule-moduulissa määritettyihin sitomisen sääntöihin perustuen. xEntryPoint-luokan *getFactoryManager*-metodi taas puolestaan palauttaa lopuksi injektorin instanssin palauttaman FactoryManager-rajapintaluokan toteuttavan FactoryManagerImpl-objektin (xUI 2016).

xUI-sovelluskehyksessä injektorina toimivalle GeneratedGinInjector-rajapintaluokalle määritetään *GinModules*-annotaatiota käyttäen sitomisen säännöt antamalla sille parametrina sovelluskehyksen itsensä luoman GeneratedGinModule-moduulin lisäksi kaikki muut GIN-moduulit, jotka käyttäjä on omassa Java-koodissaan määritellyt käytettäviksi (liite 2, 1). xUI-sovelluskehyksen luomasta GeneratedGinModule-luokasta

löytyvän *configure*-metodin toimesta määritellään mm. sitomisen säännöt, jossa FactoryManager-rajapintaluokka sidotaan sen toteuttavaan FactoryManagerImpl-luokkaan (kuva 13; Tacy ym. 2013, 523). Tämä tarkoittaa, että nähdessään Java-luokan rakentajan, kentän tai metodin yhteydessä *Inject*-annotaation ja FactoryManager-rajapintaluokan viittaustyyppinä, niin GIN-sovelluskehyksen on injektoitava FactoryManagerImpl-objekti (Tacy ym. 2013, 525). GinFactoryModuleBuilder-objektit luovat xUI-luokkien instantioimisen mahdollistavat tehtaat (kuva 13).

```
public class GeneratedGinModule extends AbstractGinModule{
  @Override
  protected void configure() {
    // Määritetyt sitomisen säännöt
    bind(FactoryManager.class)
    .to(FactoryManagerImpl.class)
    .in(Singleton.class);
    // Jokaista xUI-luokkaa kohden luodaan tehdas
    install(new GinFactoryModuleBuilder()
    .implement(HelloPage.class, HelloPageBinder.class)
    .build(HelloPage.HelloPageFactory.class));
  }
}
```

KUVA 13. xUI-sovelluskehyksen GIN-moduuli (xUI 2016)

xUI-sovelluskehys toimii luomalla konkreettiset Java-luokat koodigeneraattoreiden ajon aikana (liite 2, 2; liite 1) ja käyttämällä hyväksi DI:tä GIN-sovelluskehyksen avulla (xUI 2016). Kuvan 14 esimerkissä GWT-luokasta kutsuttu *create*-metodi ja sille parametrina annettu xEntryPoint-viittaustyyppi herättää kääntämisen aikana xUI-sovelluskehyksen xEntryPointGenerator-koodigeneraattorin, joka lopuksi palauttaa instanssin xEntryPoint-luokasta, joka todellisuudessa on instanssi xEntryPointGenerator-koodigeneraattorin luomasta xEntryPoint-rajapintaluokan toteuttavasta xEntryPointImpl-luokasta (liite 2, 1). Tämä mahdollistaa muiden käyttäjän määrittelemien ja xUI-sovelluskehyksen vaatimusten mukaisten konkreettisten tai abstraktien Java-luokkien käsittämisen ja niiden rakentamisen tehtaiden kautta tarjoamalla kuvan 12 (kuva 12) mukaisen injektorin, eli GWT-luokan *create*-metodin kääntämisen aikana luoman instanssin xUI-sovelluskehyksen koodigeneraattorin luomasta GWT:n Ginjector-rajapintaluokkaa laajentavasta GeneratedGinInjector-rajapintaluokasta.

```
private static xEntryPoint entryPoint = GWT
        .create(xEntryPoint.class);
public static FactoryManager factoryManager = entryPoint
        .getFactoryManager();
```

KUVA 14. FactoryManager-tyyppisen objektin alustaminen *on-demand*-injektiolla (xUI 2016; Tacy ym. 2013, 531)

Jokaista kuvan 15 esimerkin mukaista xUI-sovelluskehyksen xBinder-rajapintaluokan implementoivaa luokkaa, eli xUI-luokkaa kohden asennetaan myös kuvassa 13 näkyvä uusi GinFactoryModuleBuilder-objekti. Asennettu GinFactoryModuleBuilder-objekti luo tehtaan käyttäjän xUI-luokkaan kirjoittamasta rajapintaluokasta, jonka GIN-sovelluskehys injektoi alustaakseen sen käyttöä varten (Configuring simple factories 2016; Defining a factory 2016). Kun tehdasta käytetään, niin instanssi xUI-luokasta rakennetaan yhdistämällä argumentit injektorista tuleviin arvoihin (Using the factory 2016). Tehdasta ei voi kuitenkaan käyttää ennen kuvan 14 FactoryManager-tyyppisen objektin alustamista (Configuring simple factories 2016).

```
public abstract class HelloPage extends DivImpl implements
xBinder<Div, HelloPage.HelloPageFactory> {
  // xUI-luokan tehdas
  public interface HelloPageFactory {
    // Tehtaaseen määritetty metodi
    HelloPage create();
  }
  @Inject
  public HelloPage(){}
}
```

KUVA 15. Esimerkki xUI-luokasta

Kuvassa 14 *on-demand*-injektoitu FactoryManager-objekti on oikeasti Factory-Manager-rajapintaluokan toteuttava FactoryManagerImpl-luokka, joka sisältää *Inject*-annotoidun rakentajan, jolle välitetään parametrina kaikki xUI-luokissa määritetyt ja GinFactoryModuleBuilder-objektien luomat tehtaat. Tässä rakentajassa asetetaan avain-arvoparit, jossa avaimena on esimerkiksi kuvassa 15 näkyvä Class-tyypinen xUI-luokka ja arvona Object-tyyppinen saman xUI-luokan sisältä löytyvä tehdas. (xUI 2016.) Tehdas voidaan ottaa käyttöön antamalla FactoryManager-rajapintaluokasta löytyvälle *getFactory*-metodille parametrina xUI-luokka (kuva 16). Esimerkiksi kuvan 15 xUI-luokka voidaan tämän jälkeen instantioida kutsumalla luokan tehtaaseen määritettyä *create*-metodia, joka palauttaa rakennetun tyypin tai sen supertyypin. xUI-

luokan tehtaaseen määritetty metodi on mahdollista myös nimetä käyttäjän valinnan mukaan. (Defining a factory 2016.)

```
HelloPage.HelloPageFactory helloPageFactory = factoryManager
    .getFactory(HelloPage.class);
HelloPage helloPage = helloPageFactory.create();
```

KUVA 16. xUI-luokan instantiointi

Yksi xUI-sovelluskehyksen ideoista onkin tarjota mahdollisuus luokkien instantioimiseen tehtaiden kautta niin, että luokkaa käyttääkseen ohjelman ei itsensä tarvitsisi tietää kuinka tarvittava luokka luodaan (Tacy ym. 2013, 521). Tämä antaa mahdollisuuksia esim. eri luokkien testaamiseen vaihtamalla sitomisen sääntöjä GIN-moduulista. xUI-sovelluskehyksen käyttäjällä on mahdollisuus antaa omia GIN-moduulejaan xUI-sovelluskehyksen koodigeneraattoreille lisäämällä kuvassa 17 näkyvän ominaisuuden GWT-moduulin, joka on GWT-projektiin kuuluva XML-tiedosto. xUI-sovelluskehys lisää myös käyttäjän antamat GIN-moduulit injektorin *GinModules*-annotaation parametreiksi kääntämisen aikana (liite 2, 1).

```
<!-- xUI-sovelluskehyksen GWT-moduulissa määritetty ginModules-
ominaisuus -->
<define-configuration-property name="ginModules"
    is-multi-valued="true"/>
…
<!-- Käyttäjän GWT-moduuliin lisäämä ginModules-ominaisuutta
laajentava määritys -->
<extend-configuration-property name="ginModules"
   value="fi.observis.ont.helloworld.client.ginModules.MasterModule"/>
```

KUVA 17. xUI-sovelluskehyksen käyttäjä voi lisätä omia GIN-moduulejaan laajentamalla sovelluskehyksessä määritettyä *ginModules*-ominaisuutta

xUI-sovelluskehyksen perimmäinen idea on kuitenkin tarjota mahdollisuus käyttöliittymän suunnitteluun ja sen toteuttamiseen erilliselle XHTML-templaatille. xUI-sovelluskehys on Observis Oy:n oma tuote, joten se ei sisälly GWT SDK:hon ja tästä syystä riippuvuus täytyy hakea yrityksen omalta palvelimelta. Riippuvuuden hakemisen lisäksi xUI-sovelluskehys on tuotava käyttäjän omaan projektiin periyttämällä sovelluskehyksen GWT-moduuli käyttäjän omassa GWT-moduulissa.

4.2.4 Moduulin periyttäminen ja koodigeneraattorit

Ajaakseen xUI-sovelluskehyksen generaattoreita sovelluskehyksen GWT-moduuli täytyy periyttää GWT-sovelluksen moduuliin. Moduulin periyttäminen tapahtuu määrittelemällä GWT-moduuliin *inherits*-elementtillä ja *name*-attribuutilla paketti, josta xUI-sovelluskehyksen GWT-moduuli löytyy (How do I know… 2016). Tämän jälkeen xUI-sovelluskehyksen moduuliin määriteltyä *viewPackages*-ominaisuutta laajentamalla sovelluskehykselle kerrotaan paketit, joista käyttöliittymäkoodi Java-luokkineen ja XHTML-templaatteineen löytyy. (Kuva 18.)

```
<!-- xUI-sovelluskehyksen GWT-moduulissa määritetty viewPackages-
ominaisuus -->
<define-configuration-property name="viewPackages"
    is-multi-valued="true"/>
…
<!-- Periytä xUI-sovelluskehyksen GWT-moduuli -->
<inherits name='fi.observis.xUI.xui.XUI' />
<!-- Laajenna xUI-sovelluskehyksen GWT-moduulissa määritettyä
viewPackages-ominaisuutta -->
<extend-configuration-property name="viewPackages"
    value="fi.observis.ont.helloworld.client.ui" />
```

KUVA 18. Ylempänä osa xUI-sovelluskehyksen GWT-moduulista. Alempana xUI-sovelluskehyksen vaatimat määritykset käyttäjän GWT-moduulissa

Kun xUI-sovelluskehyksen käyttäjä kutsuu GWT-luokasta löytyvää *create*-metodia ja antaa sille parametrina kuvassa 19 näkyvän *when-type-assignable*-elementin *class*-attribuutin arvona määritellyn xEntryPoint-viittaustyypin, niin xUI-sovelluskehyksen moduulissa oleva ja myös samassa kuvassa näkyvä *generate-with*-elementti ohjaa GWT:n kääntäjää herättämään *generate*-metodin GWT:n Generator-luokan alaluokassa, eli xEntryPointGenerator-koodigeneraattorissa (Tacy ym. 2013, 568; Generator Configuration in Module XML). *generate-with*-elementin *class*-attribuutin arvona on GWT:n Generator-luokkaa laajentava xEntryPointGenerator-koodigeneraattori, jolla pyydetty instanssi xEntryPoint-luokasta luodaan. Elementin sisällä on *when-type-assignable*-elementti ja sen *class*-attribuutin arvona on xEntryPoint-viittaustyyppi, jonka antamalla parametrina GWT-luokasta löytyvälle *create*-metodille käynnistää siis *generate-with*-elementissä määritetyn xEntryPointGenerator-koodigeneraattorin. (liite 2, 1; Kuva 19.)

```
<generate-with
  class="fi.observis.xUI.xui.generators.xEntryPointGenerator">
    <when-type-assignable
      class="fi.observis.xUI.xui.client.xEntryPoint"/>
</generate-with>
```

KUVA 19. xUI-sovelluskehyksen GWT-moduulissa määritetty toimintaohje GWT:n kääntäjälle

xEntryPointGenerator-koodigeneraattorin käynnistämä XuiGeneratorManager-luokka käsittelee kuvan 18 GWT-moduulin *viewPackages*-ominaisuudessa määritetyn paketin Java-luokat ja asettaa kaikki xBinder-rajapintaluokan toteuttavat Java-luokat xUI-luokkien joukkoon (liite 2, 2). Näitä xBinder-rajapintaluokan toteuttavia Java-luokkia voidaan siis yleisesti nimittää xUI-luokiksi. xBinder-rajapintaluokalle on annettava kaksi tyyppiparametria, joista ensimmäinen on xUI-sovelluskehyksen tarjoaman rajapintaluokan viittaustyyppi, joka laajentaa Element-rajapintaluokkaa (xUI 2016). Kuvan 20 esimerkissä ensimmäisenä tyyppiparametrina xBinder-rajapintaluokalle on annettu xUI-sovelluskehyksen tarjoama Div-viittaustyyppi, joka on Element-rajapintaluokkaa laajentava rajapintaluokka. Element-rajapintaluokassa on määritetty kaikki tapahtumankäsittelijät, metodit ja elementin manipulaation, sekä CSS-tyylien muokkaamisen mahdollistavat metodit, jotka Element-rajapintaluokan implementoiva ElementImpl-luokka toteuttaa (xUI 2016).

```
public abstract class HelloPage extends DivImpl implements
xBinder<Div, HelloPage.HelloPageFactory>{
  public interface HelloPageFactory{
    HelloPage create();
  }
  @ViewField
  protected Span nameSpan;
  @Inject
  public HelloPage(){}
  public void setName(String name) {
    nameSpan.text(name);
  }
  public String getId(){
    return "helloPage";
  }
}
```

KUVA 20. xUI-luokka

Toisena tyyppiparametrina xBinder-rajapintaluokalle on annettava xUI-luokan sisältä löytyvä tehdas. Tehdas on rajapintaluokka, johon on määriteltävä xUI-luokan tyypin tai sen supertyypin palauttava metodi. Metodin nimen voi itse päättää, mutta yleensä

hyvänä käytäntönä on ollut nimetä se *create*-metodiksi. (Kuva 20; Defining a factory 2016.) Itse tehdas täytyy kuitenkin olla nimettynä xUI-luokan mukaan ja sijaita xUI-luokan sisällä, jotta xUI-sovelluskehyksen koodigeneraattori osaa löytää sen (liite 2, 2; xUI 2016). Tehtaan nimi muodostuu kuvan 20 esimerkin mukaisesti xUI-luokan nimen perään liitetystä *Factory*-nimestä.

xUI-sovelluskehyksen koodigeneraattori löytää xUI-luokalle kuuluvan XHTML-templaatin xUI-luokan nimen perusteella, joten XHTML-templaatti on nimettävä sen mukaan. XHTML-templaatin tiedostotunnisteena toimii ”xt”-tunniste. Kuvassa 21 näkyvän XHTML-templaatin täytyy siis olla esimerkin mukaisessa tilanteessa nimettynä ”HelloPage.xt”-nimiseksi ja sijaita samassa paketissa kuin sen omistava xUI-luokka. (xUI 2016.)

```
<content>
  <div>
    Hello, <span bid="nameSpan" class="name-span"></span>
  </div>
</content>
```

KUVA 21. xUI-luokan XHTML-templaatti

XHTML-templaatin juurielementtinä on oltava *content*-elementti. XHTML-templaatissa määritellyn *content*-juurielementin sisälle on mahdollista asetella käyttöliittymän toteuttamiseen tarvittavat HTML-elementit, joiden attribuutteina voidaan käyttää normaalisti vastaavan HTML-elementin attribuutteja. (Kuva 21.) Kuvan 21 XHTML-templaatissa *span*-elementtiin on määritetty *bid*-attribuutti ja *class*-attribuutti, joista *bid*-attribuutti on xUI-sovelluskehyksen *ViewField*-annotoitua kenttää varten tarkoitettu elementin tunniste ja *class*-attribuutti on normaali HTML-elementille annettava attribuutti. *bid*-attribuutin antamalla xUI-sovelluskehykselle siis kerrotaan, että kyseinen elementti halutaan sitoa xUI-luokan *ViewField*-annotoituun kenttään ja sitä halutaan käsitellä myös Java-koodin puolella; tällä tavoin voidaan mahdollisesti manipuloida elementtiä, lisätä tapahtuman käsittelijöitä tai käsitellä elementin tarjoamaa dataa (xUI 2016). Muussa tapauksessa *bid*-attribuuttia ei ole välttämätöntä käyttää ja elementin voi julistaa ilman attribuutteja, kuten kuvan 21 XHTML-templaatissa julistettu *div*-elementti.

Jokaista XHTML-templaatissa määriteltyä *bid*-attribuuttia kohden xUI-luokassa täytyy siis olla *ViewField*-annotoitu kenttä ja vastoin jokaista *ViewField*-annotoitua kent-

tää kohden on oltava *bid*-attribuutti. Kentän täytyy olla xUI-sovelluskehyksen tarjoamaa tyyppiä, sekä olla myös nimettynä *bid*-attribuutin arvon mukaan. Kuvan 21 XHTML-templaatissa *span*-elementille on annettu *bid*-attribuutti, jonka arvona on *nameSpan*. Tämä tarkoittaa sitä, että kuvassa 20 näkyvässä xUI-luokassa on oltava *ViewField*-annotoitu Span-viittaustyyppinen kenttä, joka on nimetty kuvassa 21 näkyvän *span*-elementin *bid*-attribuutin arvon mukaan *nameSpan*-nimellä.

xUI-luokan rakentaja pitää olla varustettuna *Inject*-annotaatiolla, sillä sitojaluokat rakentava xUiGenerator-luokka käsittelee vain *Inject*-annotoidut rakentajat ja niiden parametrit (liite 2, 2; xUI 2016). Ilman *Inject*-annotaatiota xUiGenerator-luokka ei luo rakentajaa sitojaluokkaan, joten sitä ei voi kääntämisen yhteydessä instantioida GinFactoryModulerBuilder-objektin luoman tehtaan kautta (xUI 2016). Kuvan 20 xUI-luokan rakentajalle ei välitetä parametreja. Parametrien välittäminen rakentajalle onnistuu kuitenkin käyttämällä esimerkiksi kuvan 22 tapaan *Assisted*-annotaatiota. (xUI 2016; Creating a type… 2016.)

```
public interface HelloPageFactory {
  HelloPage create(String str);
}
@Inject
public HelloPage(@Assisted String str) {
  Window.alert(str);
}
```

KUVA 22. xUI-luokan rakentajalle voidaan välittää parametreja *Assisted*-annotaatiota käyttäen

XHTML-templaatin *content*-elementistä tulee JavaScript-koodissa lopuksi xUI-luokan laajentaman ”Impl”-päätteisen luokan mukainen HTML-elementti, jolla on sitojaluokalle tunnisteena annettu *vid*-attribuutti (xUI 2016). Kuvassa 20 näkyvä xUI-luokka laajentaa xBinder-rajapintaluokalle ensimmäisenä tyyppiparametrina annettua Div-rajapintaluokan toteuttavaa DivImpl-luokkaa, joka puolestaan laajentaa ElementImpl-luokkaa (xUI 2016). DivImpl-luokan rakentajaa kutsuttaessa käytetään yliluokan, eli ElementImpl-luokan rakentajaa, jolle välitetään merkkijonona luotavan HTML-elementin nimi. Elementtien yliluokkana toimivan ElementImpl-luokan rakentaja kutsuu GWT:n DOM-luokan *createElement*-metodia, joka luo lopuksi varsinaisen HTML-elementin. Tähän perustuen xUI-sovelluskehyksessä on mahdollista luoda elementtejä myös Java-koodissa. (xUI 2016.)

```
Div div = new DivImpl();
Select select = new SelectImpl();
Option option = new OptionImpl();
option.value("1");
select.add(option);
div.add(select);
```

KUVA 23. Yksittäisten elementtien luominen ja käyttäminen xUI-sovelluskehyksessä

Elementtejä instantioidessa on syytä käyttää ensisijaisesti xUI-sovelluskehyksen tarjoamia ElementImpl-luokan aliluokkia, eli ElementImpl-luokkaa laajentavia ”Impl”-päätteisia luokkia, kuten esimerkiksi kuvassa 23 näkyviä DivImpl-, SelectImpl- ja OptionImpl -luokkia. Tämä siitä syystä, että jollakin luokalla elementin tyypistä riippuen saattaa olla määritelty lisäksi metodeja, jotka vain ElementImpl-yliluokan aliluokka toteuttaa. Esimerkiksi ElementImpl-yliluokka ei toteuta sen SelectImpl-aliluokan toteuttamaa *selectedIndex*-metodia, joka palauttaa *select*-elementistä valitun *option*-elementin indeksinumeron (xUI 2016). Lisäksi muuttujan viittaustyyppinä on hyvä käyttää ”Impl”-päätteisen luokan toteuttamaa rajapintaluokkaa, sillä Oraclen Java-dokumentaation antamaa esimerkkiä mukaillen rajapintaluokat ovat yleisesti ottaen sopimuksia sen toteuttavan luokan kanssa tarjota käytännön toteutus rajapintaluokan tarjoamille toiminnoille. Tässä sopimuksessa luetellaan, kuinka luokan tai ohjelmiston kanssa voidaan olla vuorovaikutuksessa. (Interfaces The Java™ 2016.)

Kuvan 23 esimerkissä ensimmäisenä *new*-operaattorilla instantioitu DivImpl-luokka asetetaan Div-viittaustyyppiseen muuttujaan. Vaikka objektin tyyppinä olisikin mahdollista käyttää DivImpl-tyyppistä objektia, on suositeltavampaa käyttää Div-viittaustyyppistä objektia Div-rajapintaluokassa lueteltujen toimintojen toteutuksien takaamiseksi, sillä rajapintaluokan on nimensä mukaisesti tarkoitus toimia rajapintana sen toteuttavan luokan käyttämiseksi. xUI-luokan taas täytyy laajentaa ”Impl”-päätteisiä Java-luokkia, sillä Java-luokka ei voi suoraan laajentaa Java-rajapintaluokkia.

Kuvan 20 xUI-luokka instantioidaan kuvassa 24 injektorin tarjoaman Factory-Manager-objektin ja siitä löytyvän *getFactory*-metodin avulla hakemalla GinFactoryModuleBuilder-objektin luoma tehdas, joka osaa valmistaa halutun tyyppisen xUI-objektin (xUI 2016; Configuring simple factories 2016). Kun xUI-luokka on instanti-

oitu, sen tarjoamia metodeja ja kenttiä voidaan käyttää Java-koodissa normaaliin tapaan. Kuvan 24 esimerkissä *span*-elementin sisään kirjoitetaan merkkijono ”World”.

```
public class HelloWorld implements EntryPoint {
  // Luodaan instanssi xEntryPoint-luokasta
  private static xEntryPoint entryPoint = GWT
          .create(xEntryPoint.class);
  // Suoritetaan on-demand-injektio
  public static FactoryManager factoryManager = entryPoint
          .getFactoryManager();
  @Override
  public void onModuleLoad() {
    // Haetaan HelloPage.HelloPageFactory-tyyppinen tehdas, jonka
    // GinFactoryModuleBuilder-objekti luo kääntämisen yhteydessä
    HelloPage.HelloPageFactory helloPageFactory = factoryManager
           .getFactory(HelloPage.class);
    // Instantioidaan HelloPage-luokka kutsumalla tehtaaseen
    // määritettyä metodia, joka palauttaa HelloPage-tyyppisen
    // xUI-objektin
    HelloPage helloPage = helloPageFactory.create();
    helloPage.setName("World");
    addPage(helloPage.getId(), helloPage);
  }
  public static void addPage(String pageId, Element page){
    RootPanel.get(pageId).getElement()
           .appendChild(page.getRootGwtElement());
  }
}
```

KUVA 24. EntryPoint-luokka

HelloPage-tyyppinen xUI-objekti on mahdollista asettaa näkyviin esimerkiksi kuvassa 24 näkyvällä *addPage*-metodilla. Metodille annetaan ensimmäisenä parametrina merkkijono, joka toimii HTML-sivun elementille annettuna *id*-attribuuttina ja toisena parametrina sivulle lisättävä objekti, joka on Element-viittaustyyppinen. HelloPage-tyyppinen xUI-objekti kelpaa myös xUI-sovelluskehyksen tarjoamalle Element-viittaustyypille, koska HelloPage-luokka laajentaa xUI-sovelluskehyksen tarjoamaa DivImpl-luokkaa, joka puolestaan on Element-rajapintaluokan toteuttavan ElementImpl-luokan aliluokka (kuva 24; xUI 2016).

Observis Oy on käyttänyt tähän asti xUI-sovelluskehystä GWT-sovelluskehitysprojekteissaan, joten käyttöliittymän toteuttaminen käytännössä UiBinder-sovelluskehyksellä on yritykselle toistaiseksi vieras asia. Seuraavassa luvussa dokumentoidulla käytännön työllä on tarkoitus esitellä miten GWT-sovelluksen käyttöliittymä voidaan toteuttaa käytännössä käyttäen UiBinder-sovelluskehystä sen tuo-

mia ominaisuuksia hyödyntäen. Käytännön työ aloitetaan alusta alkaen luomalla uusi GWT-Maven-projekti.

5 KÄYTTÖLIITTYMÄ UIBINDER-SOVELLUSKEHYKSELLÄ

Aloitan tässä luvussa tekemäni käytännön työn luomalla uuden GWT-Maven-projektin, sillä uuden ja toimivan GWT-Maven-projektin luominen alusta alkaen ei ole kovin selkeää tai nopeaa yrityksen käyttämillä työkaluilla. Uuden GWT-sovelluksen kehitys kuitenkin kannattaa aloittaa puhtaan projektin pohjalta, sillä vanhojen projektien pohjalta aloitettuihin uusiin projekteihin jää monesti uuden projektin kannalta täysin ylimääräisiä riippuvuuksia, koodia, asetuksia ja muuta. Tässä on selvä kehitystarve ja tulevaisuudessa tavoitteena onkin tehdä uuden GWT-Maven-projektin luomiseen oma Maven-arkkityyppinsä, jolloin uuden projektin luominen onnistuu helposti ja nopeasti. Uuden projektin luomisen on myös tarkoitus palvella tämän tavoitteen saavuttamista.

Seuraavana toteutan alkuperäisen Observis Oy:n kehittämän ja julkaiseman Jurassic Rock -festivaalisovelluksen käyttöliittymän käyttäen GWT:n UiBinder-sovelluskehystä. Tällä on tarkoitus osoittaa, että UiBinder-sovelluskehyksellä pystyy tekemään vähintäänkin samat käyttöliittymän toteutukseen liittyvät asiat kuin alkuperäisen Jurassic Rock -festivaalisovelluksen käyttöliittymän toteuttamiseen käytetyllä xUI-sovelluskehyksellä. Tarkoituksena työssä on myös demonstroida UiBinder-sovelluskehyksen tuomia ominaisuuksia ja mahdollisuuksia xUI-sovelluskehykseen nähden, kuten widgettien tehokas käyttäminen, HTML-elementtien käyttäminen, sovelluksen tyylien toteuttaminen käyttäen ja hyödyntäen GWT:n GSS-ominaisuutta, sekä tyylien määrittäminen XML-templaatissa UiBinder-ilmaisukielen avulla. Samalla muodostan käsityksen UiBinder-sovelluskehyksen toiminnasta ja siitä, että vastaako se Observis Oy:n tarpeita GWT-sovelluksien käyttöliittymien toteuttamisen osalta.

5.1 Uuden projektin luominen

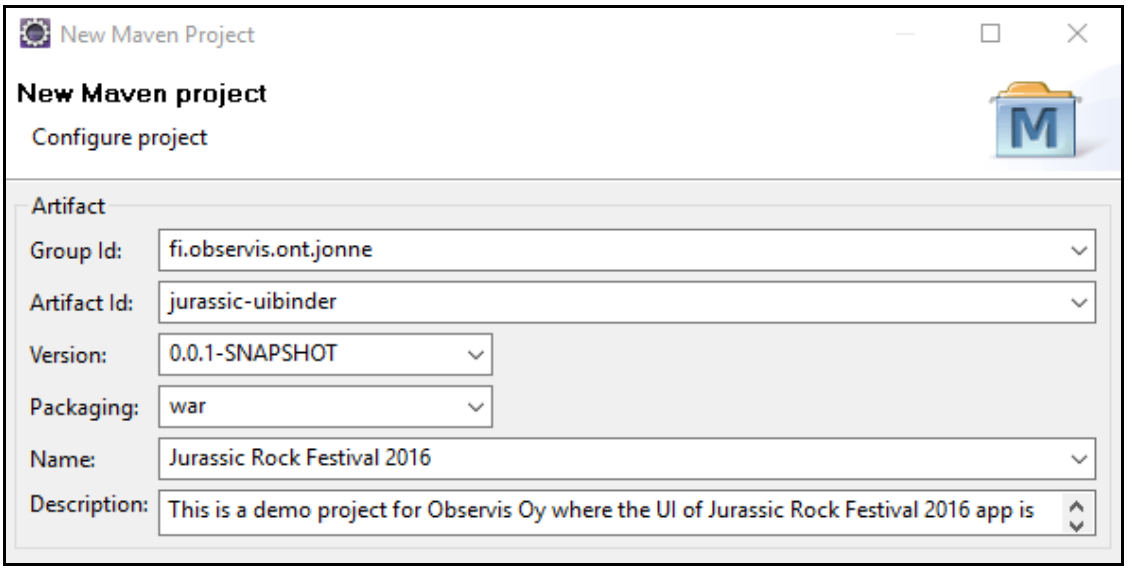
Käytän työssäni Eclipse IDE:n *Eclipse IDE for Java EE Developers* Luna-julkaisupakettia, joka sisältää oletuksena työkalut mm. Java EE ja *web*-sovellusten rakentamiseen. GWT on mahdollista ottaa käyttöön Eclipse IDE:hen asennettavan

Google Plugin for Eclipse lisäosan, eli GPE:n kautta. Tämä lisäosa sisältää vain viimeisimmän vakaan julkaisuversion GWT SDK:sta, joten työssä käyttämäni uusin GWT 2.8.0 SDK RC3-julkaisuehdokasversio on ladattava ja lisättävä Eclipse IDE:n asetuksien kautta erikseen. Uusimman GWT SDK:n voi myös ladata GWT:n verkkosivuilta ja sieltä löytyviä ohjeita noudattamalla suorittaa sen asennuksen. (Using Eclipse 2016.) Eclipse IDE -pakettin sisältyvä M2Eclipse tarjoaa integraation Apache Mavenille Eclipse IDE:ssä, jota Observis Oy käyttää myös GWT-sovelluksien riippuvuuksien hallintaan. Integrointi Apache Mavenille mahdollistaa mm. Maven-rakentamisen, riippuvuuksien hallinnan ja niiden lataamisen automaattisesti palvelimelta, opasohjelmat uusien Maven-projektien ja POM:n luomiseen, sekä Maven-tuen puhtaille Java-projekteille Eclipse IDE:n kautta. (M2Eclipse 2016.)

Uuden GWT-projektin luominen GPE:n tarjoamalla opasohjelmalla on helppoa, mutta projektin rakenteen muokkaaminen jälkeenpäin halutun laiseksi Maven-projektiksi on melko työlästä. Uusi GWT-projekti kuitenkin kannattaa luoda GPE:n avulla ilman Googlen *App Engine* -moottoria ja generoitua näytekoodia malliksi, koska siitä on myöhemmin helppo poimia vaaditut tiedostot varsinaiseen GWT-Maven-projektiin. Uuden Maven-projektin rakenne kuitenkin vastaa hyvin myös sellaisia projekteja, joissa ei GWT ole käytössä, joten rakenne voidaan pitää samankaltaisena mahdollisimman monessa projektissa. Tämä nopeuttaa joiltakin osin myös vanhoihin projekteihin perehtymistä uudelta henkilöltä, sillä projektin perusrakenne on ainakin tuttu.

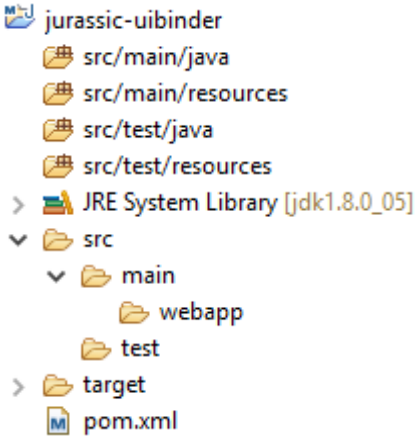
5.1.1 Maven-projektin luominen ja GWT SDK:n lisääminen

Aloitan työn luomalla Eclipse IDE:ssä uuden Maven-projektin ilman Maven-arkkityyppiä UiBinder-sovelluskehyksellä toteutettavaa Jurassic Rock -festivaalisovelluksen käyttöliittymän toteuttamista varten. Eclipse IDE:ssä Maven-projektille on määritettävä ainakin *group id,* eli ryhmätunnus ja *artifact id,* eli artefaktitunnus. Käytän projektissa ryhmätunnusta, joka on yleensä nimetty organisaation tai julkaisijan mukaan, joten käytän työssäni ryhmätunnuksena ”fi.observis.ont.jonne”-nimeä. Artefaktitunnuksena käytän sovelluskehyksen mukaan nimeä ”jurassic-uibinder”. Maven-projektille on myös hyvä jo tässä vaiheessa määrittää paketointi valitsemalla JAR, WAR tai POM. (Kuva 25.) Valitsen paketoinniksi Java-palvelimille asentuvan WAR-paketoinnin, sillä sovellus on kuitenkin luonteeltaan verkkoselaimessa toimiva *web*-sovellus.



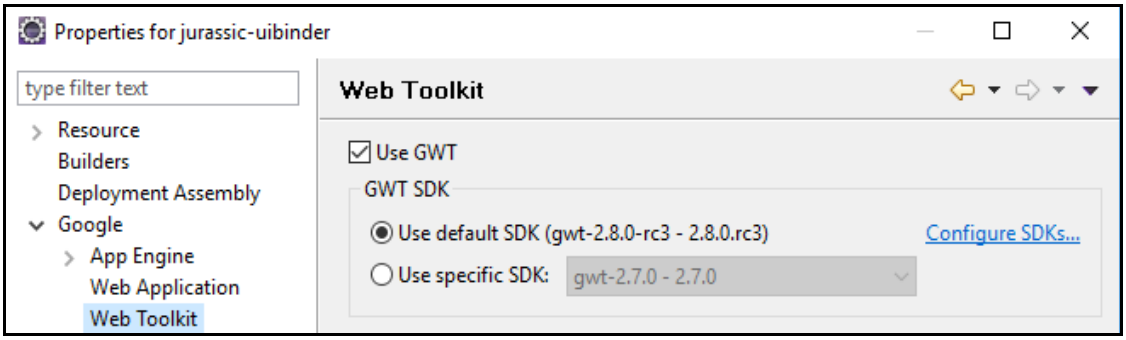
KUVA 25. Eclipse IDE opastaa uuden Maven-projektin luonnissa käynnistämällä opasohjelman

Päätän opasohjelman, jonka jälkeen Eclipse luo Maven-projektin perusasetuksineen ja -rakenteineen. Huomioitavaa on, että Eclipse-projektin nimeksi tulee Maven-projektille annettu artefaktitunnus. Projektista löytyy Java-lähdekoodia ja resurssitiedostoja varten luodut ”src/main/java”-, ”src/main/resources”-, ”src/test/java”- ja ”src/test/resources” -lähdekansiot. *src*-kansion *main*- ja *test* -alikansiot on tarkoitettu varsinaiselle ohjelmalle ja sen testikoodille. *main*-kansion *webapp*-alikansio on varattu *web*-sovelluksen tiedostoille, kuten HTML-dokumenteille, JavaScript-koodeille, CSS-tyylitiedostoille ja muille *web*-sovelluksen käyttämille resursseille. *src*-kansion rinnalla olevaan *target*-kansioon tallennetaan Maven-rakennuksen rakentamat paketit. (Introduction to the… 2016.) ”pom.xml” on tiedosto, joka sisältää Maven-projektin kokonaisuudessaan XML-muodossa; Maven-projektin ei välttämättä edes tarvitse sisältää muuta koodia. (What is the POM? 2016.)



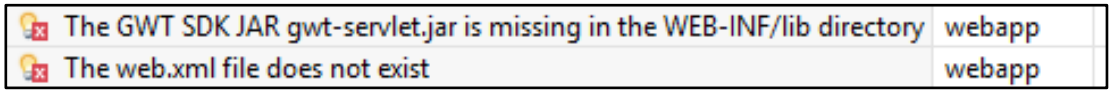
KUVA 26. Maven-projektin rakenne Eclipse IDE:ssä

Seuraavaksi lisään projektiin GWT SDK:n avaamalla projektin *properties*-ikkunan. Ikkunassa vasemmalla on nähtävissä kaikki asetukset pääkategorioittain. Google-kategorian alta *Web Toolkit* -valikosta on mahdollista tehdä projektista GWT-projekti ottamalla GWT käyttöön ja lisäämällä valinnan mukaisen GWT SDK:n. Tässä kohtaa siis valitsen GWT:n uusimman julkaisun, eli GWT 2.8.0 RC3-julkaisuehdokasversion. (Kuva 27.)



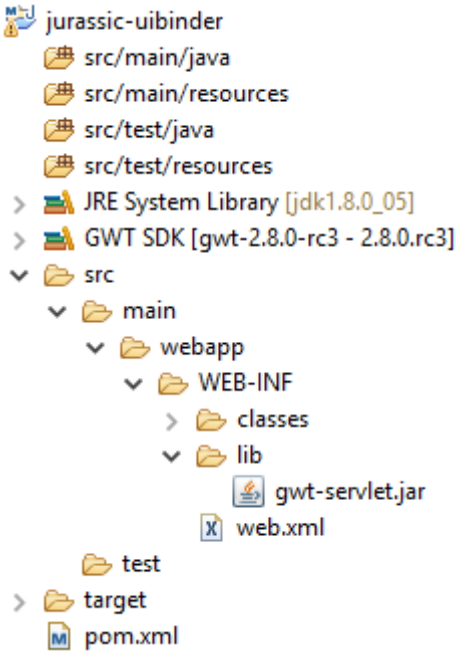
KUVA 27. GWT SDK:n käyttöönotto Eclipse-projektissa

GWT SDK:n lisäämisen jälkeen Eclipse IDE ilmoittaa projektien puuttuvista tiedostoista. GWT siis vaatii, että projektille määritetyssä WAR-polussa on oltava *WEB-INF*-kansio, jonka *lib*-alikansiossa sijaitsee GWT:n ”gwt-servlet.jar”-tiedosto. *WEB-INF*-kansiossa *lib*-kansion rinnalla on myös oltava ”web.xml”-tiedosto. (Kuva 28.)



KUVA 28. Eclipse IDE:n näyttämät virheilmoitukset GWT SDK:n lisäämisen jälkeen

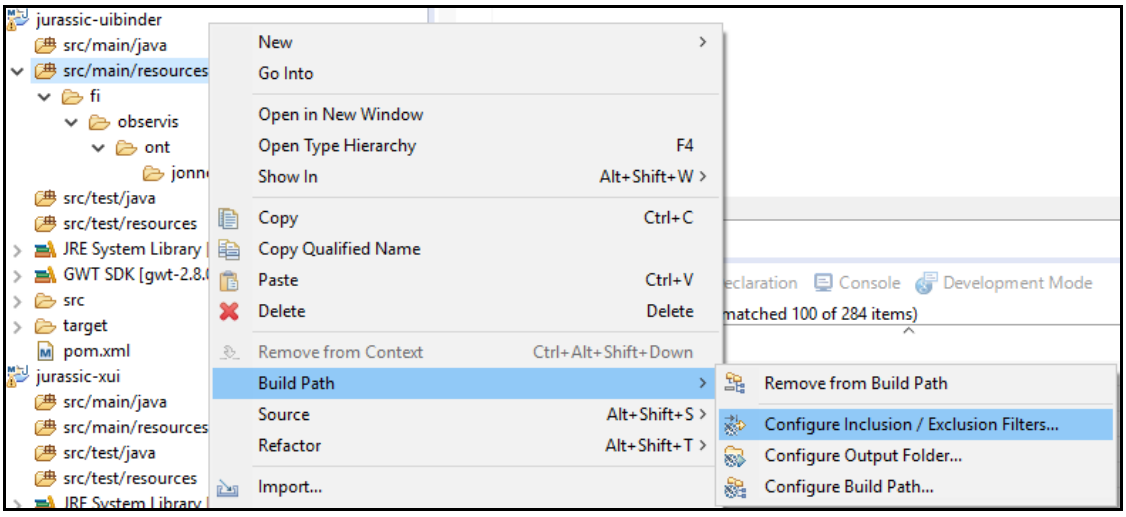
Projektin WAR-polun voi määrittää kuvassa 27 näkyvästä *Web Toolkit* -valikon yläpuolella sijaitsevasta *Web Application* -valikosta. Käytän projekteissani ”src/main/webapp”-polkua, jonne mm. GWT:n kääntäjän Java-lähdekoodista kääntämät JavaScript-tiedostot ja muut *web*-sovelluksen toimintaan vaaditut tiedostot tullaan asettamaan. Luon tässä vaiheessa malliksi tyhjän GWT-projektin ilman *App Engine* -moottoria ja generoitua näytekoodia GPE:n tarjoamalla opasohjelmalla ja lisään GWT:n vaatimat tiedostot kopioimalla ne malliprojektista varsinaiseen GWT-Maven-projektiin, jolloin GWT-Maven-projektista tulee kuvan 29 rakenteen mukainen.



KUVA 29. GWT-Maven-projektin rakenne

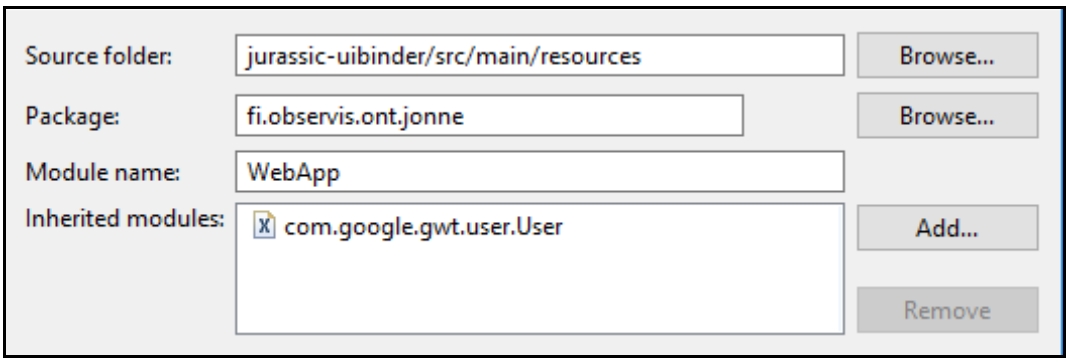
Vaikka Eclipse IDE ei ilmoita enää virheistä, niin toimiakseen GWT vaatii vielä projekteihin lisättäväksi seuraavat tiedostot: GWT-moduulin, joka sisältää ohjeet GWT:n kääntäjää varten, ja EntryPoint-luokan, joka toimii sovelluksen pääluokkana, sekä HTML-dokumentin, joka pyytää tiettyä permutaatiota käännettyä sovellusta varten (Tacy ym. 2013, 34). Luon uuden paketin ”src/main/resources”-lähdekansioon GWT-moduulia varten ja nimeän sen ”fi.observis.ont.jonne”-nimiseksi. Mikäli tämän jälkeen lisätty paketti näkyy kansiorakenteena, niin pakettirakenteen saa näkyviin poistamalla Eclipsen IDE:n oletuksena lisäämän ”**”-kaavan jättää pois ”src/main/resources”-lähdekansiosta kaikki tiedostot ja kansiot. Lähdekansioiden alikansioita ja tiedostoja voi sisällyttää tai jättää pois avaamalla esimerkiksi kuvan 30 osoittaman *Configure Inclusion / Exclusion Filters* -ikkunan *Build Path* -valikon alta.

Poistan siis ”**”-kaavan ja suljen ikkunan, jolloin luomani paketti tulee näkyviin ”src/main/resources”-lähdekansion alle normaalisti.



KUVA 30. *Build Path* -valikon avaaminen Eclipse IDE:ssä

Seuraavaksi luon varsinaisen GWT-moduulin käyttämällä hyväksi GPE:n tarjoamaa opasohjelmaa. Painan paketin päältä oikeaa hiiren nappia ja luon uuden GWT-moduulin valitsemalla *New*-valinnasta avautuvasta listasta valinnan *Module*. Mikäli valintaa *Module* ei näy listassa, niin sen löytää valitsemalla valinnan *Other*, joka avaa ikkunan, jossa näkyy kansioittain kaikki Eclipse IDE:ssä luotavat tiedostot ja projektit. Opasohjelman käynnistävä valinta *Module* löytyy GWT-kansion alta.



KUVA 31. GPE:n opasohjelma GWT-moduulin luomiseen

Opasohjelma kysyy projektin lähdekansiota ja pakettia minne GWT-moduuli luodaan (kuva 31). Mikäli nämä on jo luotuna ja opasohjelman on käynnistänyt GWT-moduulia varten luodun ”fi.observis.ont.jonne”-paketin ollessa valittuna, niin opasohjelma osaa täyttää nämä kentät automaattisesti. GWT-moduulin nimi täytyy määrittää itse, joten nimeän molempien projektien moduulit ”WebApp”-nimisiksi. Tässä vai-

heessa on mahdollista myös määrittää muut periytettävät GWT SDK:n mukana tulevat moduulit. Toistaiseksi en periytä tässä vaiheessa muita GWT-moduuleja kuin User-moduulin. Oletuksena kaikkien GWT-sovellusten moduulien on perittävä ainakin GWT:n User-moduuli, sillä se sisältää GWT-kääntäjän perussäännöt mille tahansa GWT-sovellukselle (Tacy ym. 2013, 35).

Luon GWT-moduulin viimeistelemällä opasohjelman, joka luo ”src/main/resources”-lähdekansion alle ”fi.observis.ont.jonne”-pakettiin ”WebApp.gwt.xml”-nimisen GWT-moduulina toimivan XML-tiedoston. Nimeän moduulin kuvassa 32 näkyvän esimerkin mukaan lisäämällä *rename-to*-attribuutin ja antamalla sen arvoksi ”WebApp”, jotta GWT käyttäisi yksinkertaisempaa moduulin nimeä sen vakiona käyttämän pitkän paketin nimen sijaan. Samassa kuvassa näkyvä *entry-point*-elementti luodaan myöhemmin *Entry Point Class* -opasohjelman toimesta, joten sitä ei tarvitse tässä vaiheessa lisätä käsin (kuva 32).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC
"-//Google Inc.//DTD Google Web Toolkit 2.8.0.rc3
//EN" "http://gwtproject.org/doctype/2.8.0.rc3/gwt-module.dtd">
<module rename-to="WebApp">
  <inherits name="com.google.gwt.user.User" />
  <source path="client"/>
  <entry-point
      class="fi.observis.ont.jonne.client.WebApp">
  </entry-point>
</module>
```

KUVA 32. GPE:n *Module*- ja *Entry Point Class* -opasohjelmien luoma GWT-moduuli

GPE:n opasohjelma luo myös automaattisesti ”fi.observis.ont.jonne.client”-paketin samalle tasolle GWT-moduulin sisältämän paketin kanssa. Siirrän tämän paketin raahaamalla sen hiirellä ”src/main/java”-lähdekansioon, sillä *client*-paketeissa olevat tiedostot tulevat sisältämään Java-koodia. Juuri luodun GWT-moduulin mukaan *source*-elementin *path*-attribuutin arvona annettu *client* kertoo GWT:n kääntäjälle kääntää kaikki *client*-paketeissa ja sen alipaketeissa oleva Java-koodi JavaScript-koodiksi (kuva 32; Tacy ym. 2013, 36).

Tämän jälkeen luon uuden EntryPoint-luokan ”fi.observis.ont.jonne.client”-pakettiin samaan tapaan kun loin aiemmin uuden GWT-moduulin, eli käynnistän GPE:n tar-

joaman opasohjelman painamalla hiiren oikeaa nappia ”src/main/java”-lähdekansioon siirtämäni paketin päältä ja valitsen listasta valinnan *Entry Point Class*. Tämä käynnistää opasohjelman, jonka avulla luon GWT:n EntryPoint-rajapintaluokan toteuttavan Java-luokan (kuva 33). Opasohjelman kentät ovat *Name*-kenttää lukuun ottamatta esitäytettyjä, joten lisään EntryPoint-luokan nimeksi GWT-moduulin nimenä käyttämäni ”WebApp”-nimen ja viimeistelen opasohjelman. Opasohjelma luo ”src/main/java”-lähdekansion alle ”fi.observis.ont.jonne.client”-pakettiin ”WebApp.java”-tiedoston, eli EntryPoint-luokkana toimivan WebApp-luokan ja lisää samalla GWT-moduuliin kuvassa 32 näkyvän *entry-point*-elementin *class*-attribuutin arvoksi paketin, jossa luotu EntryPoint-luokka sijaitsee.

```
public class WebApp implements EntryPoint {
  public void onModuleLoad() {
    // TODO Auto-generated method stub
  }
}
```

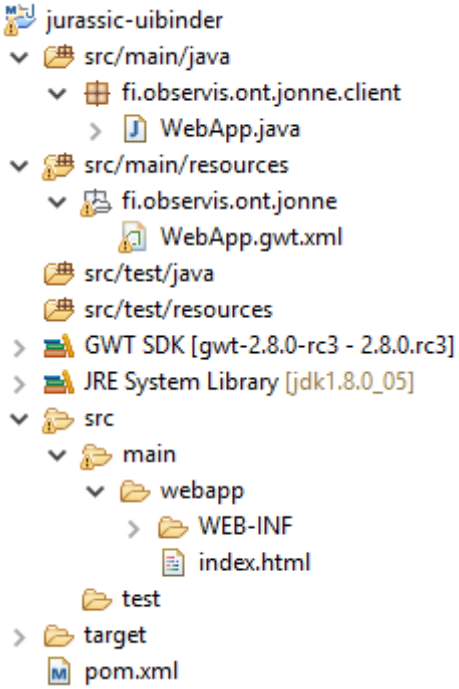
KUVA 33. GPE:n *Entry Point Class* -opasohjelman luoma EntryPoint-luokka

Viimeisenä luon GWT-sovelluksen pohjana toimivan HTML-dokumentin WAR-poluksi määritetyn *webapp*-kansion juureen käyttäen GPE:n *HTML Page* -opasohjelmaa edellä käytettyjen *Module*- ja *Entry Point Class* -opasohjelmien tapaan. Opasohjelma täyttää kentät *File name* -kenttää lukuun ottamatta automaattisesti. Nimeän HTML-tiedoston ”index”-nimiseksi ja annan myös Tacyn ym. (2013, 39) suosittelemana opasohjelmalle luvan luoda HTML-tiedostoon *iframe*-elementin, joka lisää tuen selainhistorialle (kuva 34). Viimeistelen opasohjelman, joka tämän jälkeen luo ”index.html”-tiedoston määrittämääni polkuun.

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="content-type"
      content="text/html; charset=UTF-8">
    <title>index</title>
    <script type="text/javascript"
      src="WebApp/WebApp.nocache.js">
    </script>
  </head>
  <body>
    <iframe src="javascript:''" id="__gwt_historyFrame" tabIndex='-1'
      style="position:absolute;width:0;height:0;border:0">
    </iframe>
  </body>
</html>
```

KUVA 34. GPE:n *HTML Page* -opasohjelman luoma HTML-dokumentti

Luotu HTML-dokumentti sisältää mm. Tacyn ym. (2013, 40) mainitsemat neljä pääkohtaa, joista ensimmäisenä on Java-koodin kautta RootPanel-luokasta löytyvä *get*-metodilla käsiteltävissä oleva *body*-elementti. Toisena pääkohtana on *iframe*-elementti, jota käytetään tukena selainhistorian hallintaan. Kolmantena pääkohtana on *script*-elementti, johon linkitetty *bootstrap*-lataaja lataa ja käynnistää GWT-kääntäjän Java-koodista kääntämän GWT-sovelluksen JavaScript-version. Viimeisenä pääkohtana on ensimmäisellä rivillä määritelty dokumentin tyyppi, joka kertoo selaimelle, että dokumenttia pitää kohdella oletusarvoisesti HTML-dokumenttina. (Mts. 39–40; Kuva 34.)



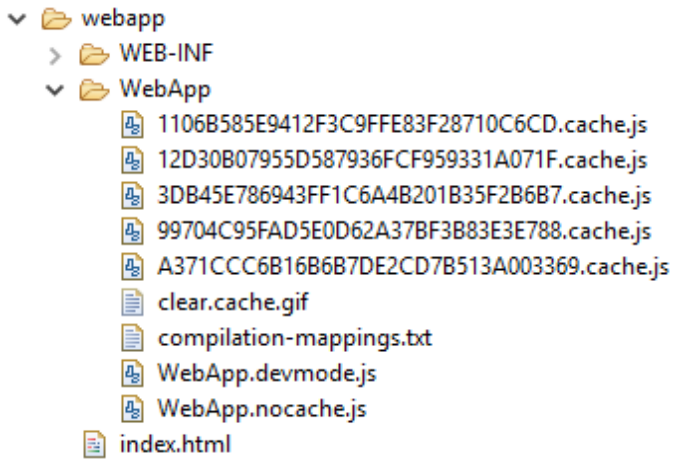
KUVA 35. Valmiin GWT-Maven-projektin rakenne

Nyt luodussa projektissa on tehtynä kaikki määritykset tiedostoineen mitä GWT-Maven-projekti vaatii mm. kääntääkseen GWT:llä Java-koodin JavaScript-koodiksi, sekä rakentaakseen projektin ja hallitakseen riippuvuuksia käyttäen M2Eclipse integraatiota Apache Mavenille. Varmistan projektin toiminnan ajamalla GWT:n kääntäjän painamalla hiiren oikeaa nappia projektin päältä ja valitsemalla Google-valikosta valinnan GWT *compile*, jolloin GPE:n GWT *compile* -opasohjelma käynnistyy. Tässä vaiheessa on mahdollista määrittää käännettävän projektin lisäksi muita kääntämisen lopputulokseen vaikuttavia valintoja.

5.1.2 Kääntäminen ja rakentaminen

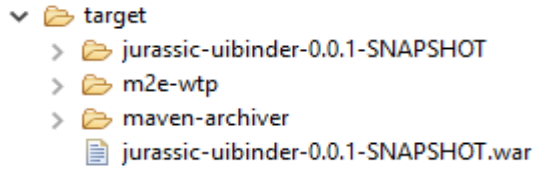
Kääntäjälle voi antaa lisäksi erilaisia argumentteja *Advanced*-valinnan kautta. Valitsen kuitenkin oletusarvot ja viimeistelen opasohjelman, jolloin Eclipse IDE avaa ikkunan kysyen WAR-kansion sijaintia. Valitsen projekteihin määritetyn WAR-polun mukaisen sijainnin, joka siis on projektin juuressa oleva ”src/main/webapp”-polun *webapp*-kansion juuri. Tämän jälkeen GWT:n kääntäjä käynnistyy ja suorittaa kääntämisen tehden 5 permutaatiota, jotka ovat kuvassa 36 ”WebApp”-kansion sisällä näkyviä pitkällä nimellä varustettuja ”cache.js”-tiedostoja. Samaisessa kansiossa on myös aikaisemmin luotuun HTML-tiedostoon linkitetty ”WebApp.nocache.js”-tiedosto, eli

bootstrap-lataaja, joka lataa palvelimelta oikealle selainympäristölle käännetyn permutaation (kuva 36; liite 1).



KUVA 36. GWT-kääntäjän WAR-kansioon luoma GWT-moodulin nimen mukainen *WebApp*-kansio sisältöineen

Kokeilen vielä, että Maven-rakennus toimii valitsemalla *Run As* -valikosta valinnan Maven *install*. Maven-rakennus käynnistyy rakentaen projektin alkuvaiheessa määritetyn paketoinnin tyypin mukaisen WAR-paketin *target*-kansioon (kuva 37). Maven-rakennus ilmoittaa Eclipse IDE:n konsolissa onnistuneesta rakentamisprosessista.



KUVA 37. Maven-rakennuksen tulos

Aloitan tämän projektin pohjalta varsinaisen työn toteuttaen Jurassic Rock -festivaalisovelluksen käyttöliittymän UiBinder-sovelluskehyksellä. Sovelluskehys on tämän projektin pohjalta valmis otettavaksi käyttöön, sillä se sisältyy GWT SDK:hon. UiBinder-sovelluskehystä ei siis tarvitse periyttää GWT-moduuliin, eikä tuoda projektiin riippuvuutena erikseen tai Apache Mavenin kautta.

5.2 Käyttöliittymän toteuttaminen

Teen työn edellisessä luvussa tekemäni ”jurassic-uibinder”-nimisen GWT-Maven-projektin pohjalle. Käytän tässä työssä myös suoraan valmiita Observis Oy:n kehittä-

män ja julkaiseman Jurassic Rock -festivaalisovelluksen tyylejä, kuvia ja muuta grafiikkaa, sillä työn tarkoituksena on vain toimia esimerkkinä UiBinder-sovelluskehyksellä toteutettavalle *web*-sovelluksen käyttöliittymälle, sekä demonstroida UiBinder-sovelluskehyksen toimintaa ja sen ominaisuuksia kyseisen sovelluksen kautta. Tämä tarkoittaa myös, että jätän työstä pois kaiken käyttöliittymän toiminnan esittelyn kannalta epäolennaisen toiminnallisuuden tekemisen ja dokumentoinnin, kuten palvelinrajapinnan rakentamisen, datan lukemisen ulkoisista lähteistä ja yleisesti varsinaisen Jurassic Rock -sovelluksen oikean toimintalogiikan toteuttamisen.

Toteutan vain toiminnot, jotka ovat sovelluksen käyttäjän havaittavissa olevia käyttöliittymällisiä toimintoja. Työssä käyttämäni Java-luokat, Java-metodit ja Jurassic Rock -festivaalisovelluksen Java-lähdekoodiin viittaavat resurssit nimeän omalla tavallani. Lisään myös tarvittavat natiivit JavaScript-kirjastot ja niihin kuuluvat tyylitiedostot, sekä fontit tarpeen mukaan ilman erillistä mainintaa.

5.2.1 Alkuvalmistelut

Aloitan työn siirtämällä Jurassic Rock -sovelluksen kuvat ja grafiikat omaan projektiin. Luon projektin WAR-kansioon, eli *webapp*-kansioon *img*-kansion, jonka alle siirrän valmiin sovelluksen kuvat ja grafiikat sisältämän *jurassic*-kansion. Tämän jälkeen asetan sovellukseen latauskuvan lisäämällä WAR-kansiossa olevaan ”index.html”-tiedostoon *body*-elementin sisälle uuden *div*-elementin, joka toimii kehyksenä latauskuvalle (kuva 38). Ideana tässä on asettaa latauskuva näkyviin heti kun sovellus avautuu ja poistaa se näkyvistä heti kun GWT-sovellus kokonaisuudessaan on latautunut taustalla näkyviin. Tästä johtuen lisään poikkeuksellisesti tyylit tätä varten luoden *webapp*-kansioon uuden *css*-kansion ja sen sisälle uuden tarvittavat tyylittelyt sisältävän ”loader.css”-tiedoston. Liitän myös nämä tyylit osaksi HTML-dokumenttia kuvassa 38 näkyvän *link*-elementin avulla. Varsinaisen sovelluksen tyylit on myöhemmin tarkoitus toteuttaa kuitenkin käyttämällä GWT:n GSS-ominaisuutta.

```
…
  <link rel="stylesheet" href="css/loader.css"/>
  <script type="text/javascript"
    src="WebApp/WebApp.nocache.js">
  </script>
</head>
  <body>
    <div id="loader">
      <img class="logo" src="img/jurassic/app/jurassic.png">
      <img src="img/jurassic/app/jurassic.svg">
    </div>
…
```

KUVA 38. Sovelluksen HTML-dokumentti

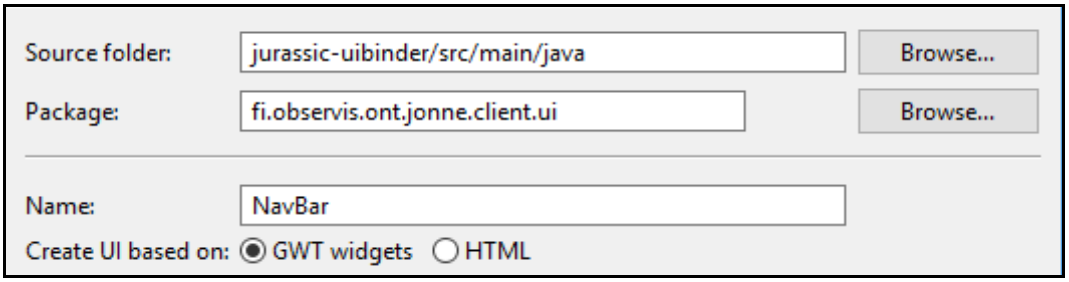
Ajan seuraavaksi sovelluksen ensimmäistä kertaa käynnistämällä GWT:n *Super Dev Mode* -tilan painamalla hiiren oikeaa nappia projektin päältä ja valitsemalla sen *Run As* -valinnan alta. GWT käynnistää palvelimen Eclipse IDE:ssä ja ilmoittaa antamalla linkin sovelluksen etusivuksi määritettyyn HTML-dokumenttiin *Development Mode* -konsolissa kun palvelin on valmiina. Linkki käynnistää selaimen ja vie paikalliseen osoitteeseen ”http://127.0.0.1:8888/index.html”. Kun pääsen sivulle, niin GWT:n kääntäjä aloittaa Java-koodin kääntämisen lennosta JavaScript-koodiksi tehden vain yhden permutaation kyseistä selainta kohden. Sivu näkyy kääntämisen ajan harmaan tason läpi ja valkoisessa palkissa lukee GWT:n ilmoittama viesti: *Compiling WebApp* (kuva 39).

KUVA 39. GWT kääntää *web*-sovellusta *Super Dev Mode* -tilassa

Seuraavaksi lisään uuden ”fi.observis.ont.jonne.client.ui”-paketin ”src/main/java”-lähdekansioon. Tarkoituksena on rakentaa vain jo valmis Jurassic Rock -festivaalisovelluksen käyttöliittymä UiBinder-sovelluskehyksellä, joten suurin osa

Java-koodista tulee olemaan tässä käyttöliittymäkoodin sisältävässä paketissa. Tähän pakettiin on siis tarkoitus lisätä kaikki UiBinder-luokat ja niiden XML-templaatit.

Lisään luomaani pakettiin GPE:n *UiBinder*-opasohjelman avulla uuden UiBinder-luokan, josta tulee sovelluksen navigaatiopalkki. Täytän siis *Name*-kenttään UiBinder-luokan nimen ”NavBar” ja valitsen käyttöliittymän perustuen GWT:n widgetteihin ilman generoitua näytekoodia. Noudatan tässä kohtaa Tacyn. ym. (2013, 106) antamaa neuvoa widgettien käytön tehokkuuteen liittyen; mikäli käyttöliittymäkomponentin tarvitsee reagoida tapahtumiin, niin se on parempi luoda widgettinä kuin HTML-elementtinä. Tämä näkyy käytännössä myöhemmin mm. tapahtumankäsittelijän kirjoittamisessa, sekä UiBinder-luokan lisäämisessä HTML-dokumenttiin. Päätän opasohjelman, joka luo NavBar-nimisen UiBinder-luokan ja NavBar-nimisen XML-templaatin ”fi.observis.ont.jonne.client.ui”-pakettiin.



KUVA 40. GPE:n *UiBinder*-opasohjelma

Ensimmäisen UiBinder-luokan pitäisi nyt olla valmis instantioitavaksi ja lisättäväksi sivulle näkyviin. Lisään siis sovelluksen pohjana toimivaan HTML-dokumenttiin *div*-elementin, jolle annan *class*- ja *id* -attribuuttien arvoksi ”navbar”. Tämä elementti toimii navigaatiopalkin ankkurielementtinä. Instantioin UiBinder-luokan EntryPoint-luokassa ja lisään objektin HTML-dokumenttiin kutsumalla tekemääni *addToBody*-metodia, jonka avulla on helpompi lisätä UiBinder-widgettejä näkyviin. Metodi ottaa parametrina merkkijonon, joka on HTML-dokumentissa *body*-elementissä sisällä olevan ankkurielementin *id*-attribuutti ja Widget-tyyppisen objektin, jonka elementti lisätään ankkurielementtiin (kuva 41). Käytän tätä metodia jokaisen instantioidun widgetin lisäämiseksi HTML-dokumentin *body*-elementtiin ja tarvittaessa ylikuormitan samaa metodia ottamaan parametrina myös muun tyyppisiä objekteja widgettien lisäksi; esimerkiksi UIObject-tyyppisen objektin voi lisätä samaa metodia käyttäen.

```
public static void addToBody(String elementId, Widget w){
  RootPanel.get(elementId).add(w);
}
```

KUVA 41. *addToBody*-metodi

Käynnistän sovelluksen *Super Dev Mode* -tilassa ja katson selaimessa kehittäjän työkalun kautta, että UiBinder-luokassa opasohjelman oletuksena luoma HTMLPanel-widgetti tulee näkyviin *div*-elementtinä sille tarkoitetun ankkurielementin sisälle. Tämän jälkeen kutsun EntryPoint-luokassa kutsutun *addToBody*-metodin jälkeen *removeLoader*-metodia, joka poistaa sovelluksen latauskuvan näkyvistä JSNI:n avulla käyttäen jQuery-kirjaston *fadeOut*-funktiota. Latauskuva poistuu näkyvistä kun sovelluksen näkymä on rakennettu valmiiksi.

5.2.2 GSS-ominaisuuden käyttöönotto ja navigaatiopalkin tyylit

Seuraavaksi lisään varsinaiset tyylit sovellukseen käyttäen GSS-ominaisuutta. Ominaisuuden käyttöönotto vaatii kuvan 42 mukaisen rivin lisäämisen GWT-moduuliin, jolla kerrotaan kääntäjälle ottaa huomioon GSS-tyylit kääntämisen yhteydessä. Käyttöönotto vaatii myös uuden GWT:n *ClientBundle*-resurssin tekemisen, sekä GSS-tiedostojen määrittämisen. Ensimmäisenä kuitenkin lisään kuvan 42 mukaisen rivin GWT-moduuliin, joka on toistaiseksi pakollinen GSS:n käyttöönoton kannalta, sillä GSS-ominaisuus ei ole oletuksena käytössä.

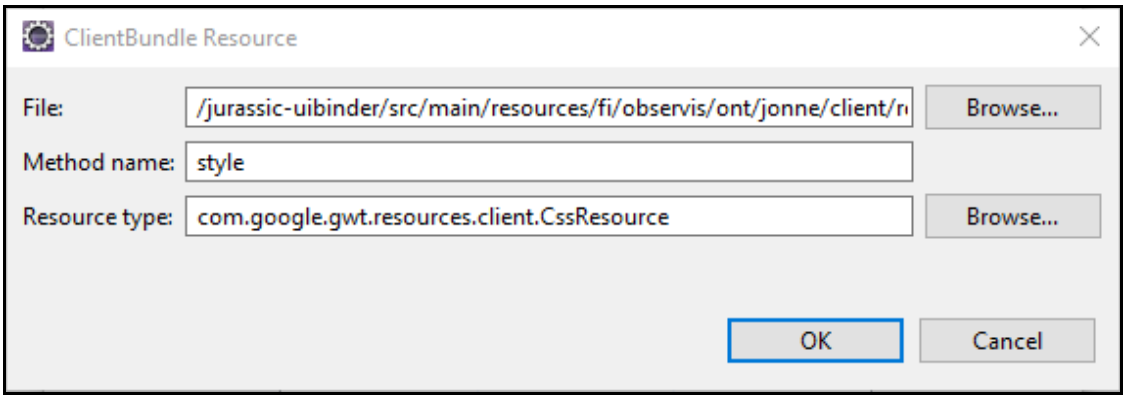
```
<set-configuration-property
    name="CssResource.enableGss" value="true" />
```

KUVA 42. GSS-ominaisuuden käyttöönotto GWT-moduulissa

Tämän jälkeen luon uuden ”fi.observis.ont.jonne.client.resources.css”-paketin ”src/main/resources”-lähdekansion alle ja lisään sinne uuden ”style.gss”-nimisen tiedoston. Varmistan että tiedosto aukeaa Eclipse IDE:n CSS *resource editor* -editorissa, jotta CSS:n syntaksi näkyisi oikealla tavalla. En kuitenkaan vielä lisää tyylejä tähän tiedostoon, vaan annan tiedoston olla toistaiseksi tyhjänä. Tämä siksi, koska GPE tukee vain GWT:n uusinta vakaata julkaisuversiota, eli GPE:n opasohjelmat tekevät asiat vielä GWT 2.7.0 -version mukaisesti, jossa GSS oli vielä kokeellinen ominaisuus (CSS3 and GWT in perfect harmony 2015, 107). Esimerkiksi jos GSS-tyylitiedosto

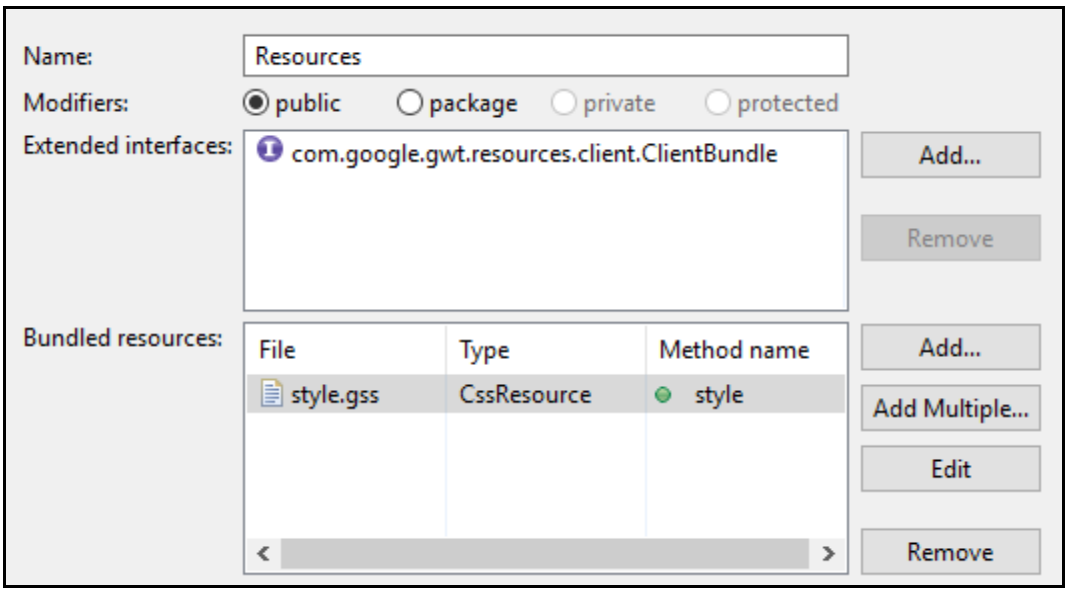
sisältää tyylejä uutta *ClientBundle*-resurssia tehdessä, niin opasohjelma tarkastaa GSS-tyylit ja ilmoittaa virheistä, mikäli ne eivät ole CSS2:n mukaisia. Yhtenä GSS:n ominaisuuksista on kuitenkin tukea CSS3:sta.

Seuraavaksi luon uuden ”fi.observis.ont.jonne.client.resources”-paketin ”src/main/java”-lähdekansion alle ja lisään pakettiin GPE:n *ClientBundle*-opasohjelman avulla uuden ClientBundle-resurssin. Ensimmäisenä annan opasohjelman *Name*-kenttään rajapintaluokan nimen, joka laajentaa ClientBundle-rajapintaluokkaa. Seuraavana lisään tyylit painamalla *Bundled resources* -ikkunan *Add*-painiketta. (Kuva 44.) Tämä avaa kuvan 43 mukaisen ikkunan.



KUVA 43. Uuden *ClientBundle*-resurssin lisääminen

File-kentän *Browse*-painikkeesta avautuu ikkuna, josta on helppoa löytää ensimmäisenä luotu ”style.gss”-tiedosto. Kun tiedoston polku on lisätty *File*-kenttään, niin opasohjelma täyttää loput kentät automaattisesti. (Kuva 43.) Hyväksyn nämä painamalla *OK*-painiketta, joka lisää *ClientBundle*-opasohjelman *Bundled resources* -ikkunaan uuden tyyliresurssin (kuva 44). Tämän jälkeen viimeistelen opasohjelman.



KUVA 44. GPE:n *ClientBundle*-opasohjelma

Opasohjelma luo kuvan 44 mukaisilla valinnoilla kaksi rajapintaluokkaa: Resources- ja Style -rajapintaluokat. Resources-rajapintaluokka pitää sisällään GWT-sovellukselle määritetyt resurssit ja Style-rajapintaluokka huolehtii GSS-tiedostossa määritettyjen tyylien tarjoamisesta GWT-sovellukselle. Avainasiana ClientBundle-resurssien käyttöönottamiselle on luoda GWT-luokan *create*-metodin palauttama instanssi tehdystä Resources-rajapintaluokasta. Opasohjelma ei kuitenkaan tee tätä automaattisesti, joten julistan Resources-viittaustyyppiä olevan ”INSTANCE”-muuttujan, johon asetan GWT-luokan *create*-metodin palauttaman instanssin rajapintaluokasta itsestään. Opasohjelma luo automaattisesti tyylit käsittävän Style-rajapintaluokan mukaan *Source*-annotoidun *style*-metodin, joka palauttaa Style-viittaustyyppisen objektin. *Source*-annotaatioon on parametrina asetettu polku, josta luotu ”style.gss”-tiedosto löytyy. (Kuva 45.)

```
public interface Resources extends ClientBundle {
  public static final Resources INSTANCE = GWT
          .create(Resources.class);
  @Source("fi/observis/ont/jonne/client/resources/css/style.gss")
  Style style();
}
```

KUVA 45. Resources-rajapintaluokka

Seuraavana avainasiana tyylien käyttöönottamiseksi tyylit on injektoitava DOM:iin, jotta ne näkyisivät *web*-sovelluksessa. Lisään siis sovelluksen EntryPoint-luokan *onModuleLoad*-metodin sisälle heti ensimmäiseksi kuvan 46 mukaisen rivin, jolla var-

mistan, että tyylit injektoidaan heti ensimmäisenä kun *web*-sovellus käynnistyy. Style-rajapintaluokka perii *ensureInjected*-metodin laajentamaltaan CssResource-rajapintaluokalta.

```
Resources.INSTANCE.style().ensureInjected();
```

KUVA 46. Tyylien asettaminen näkyväksi *web*-sovelluksessa

Nyt GSS-ominaisuus on otettu käyttöön, joten seuraavana jatkan luomani NavBar-luokan XML-templaatin muokkaamista toimivaksi ja näyttäväksi navigaatiopalkiksi lisäämällä alkuperäisen Jurassic Rock -sovelluksen navigaatiopalkin mukaisen HTML:n GWT:n widgetteinä ja DOM-elementteinä XML-templaatin juurielementin sisälle. Ennen tätä kuitenkin lisään alkuperäiset Jurassic Rock -sovelluksessa käytetyt navigaatiopalkin tyylit ”style.gss”-tiedostoon ja korvaan CSS-luokkien nimistä sanojen erottimena käytetyt väliviivat alaviivalla, sillä GSS:n mukaisten CSS-luokkien nimet on käännyttävä Java-metodien nimiksi ja Java-metodeissa taas ei voi käyttää väliviivaa.

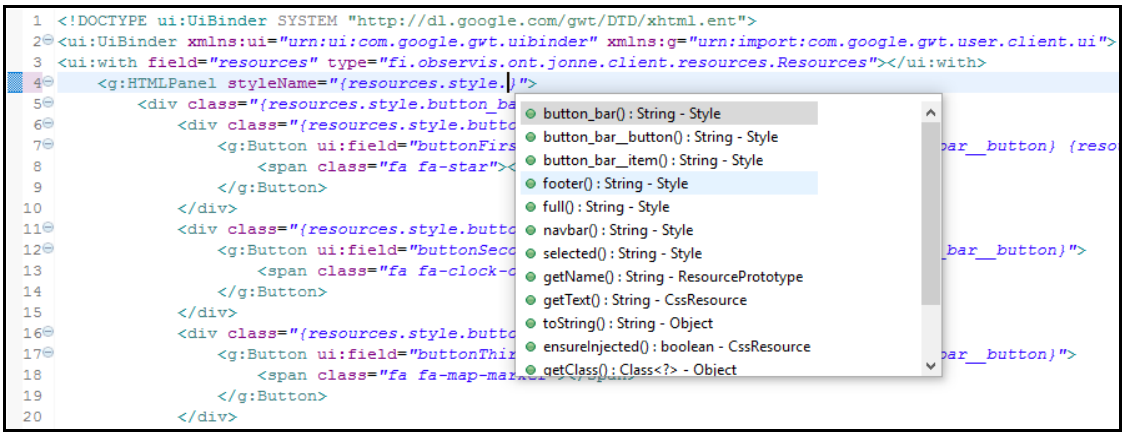
Määrittelen siis CSS-luokkien nimet Java-metodeiksi *ClientBundle*-opasohjelman avulla luotuun Style-rajapintaluokkaan. Java-metodien niminä toimii täsmälleen sama nimi kuin ”style.gss”-tiedostossa määritelty CSS-luokan nimi. Kaikki Style-rajapintaluokkaan määritetyt metodit palauttavat String-tyyppisen objektin. (Kuva 47.)

```
public interface Style extends CssResource {
  // NavBar styles
  String navbar();
  String footer();
  String button_bar();
  String full();
  String button_bar__item();
  String button_bar__button();
  String select();
}
```

KUVA 47. Style-rajapintaluokka

Tämän jälkeen tuon luomani resurssit UiBinder-luokan XML-templaattiin lisäämällä kuvassa 48 näkyvän *ui:with*-elementin, jonka *field*-attribuutille annetun arvon alta löytyy kaikki *type*-attribuutin arvon tarjoamat resurssit. Varmistan myös, että XML-templaatti avautuu Eclipse IDE:n *UiBinder Template Editor* -editorissa, jotta auto-

maattinen täydennys ja haku toimisivat kuvan 48 mukaisesti. *type*-attribuutin arvo siis viittaa aiemmin GPE:n *ClientBundle*-opasohjelman avulla luotuun Resources-rajapintaluokkaan, jossa määritelty *style*-metodi palauttaa Style-viittaustyyppisen objektin, jonka on toteutettava Style-rajapintaluokassa määritellyt metodit. Tämän ansiosta kuvan 48 esimerkin mukaan voin löytää tyylit UiBinder-luokan XML-templaatissa *resources*-kentän *style*-metodin alta ja käyttää niitä navigaatiopalkin tai minkä tahansa widgetin tai elementin tyylittelyyn.



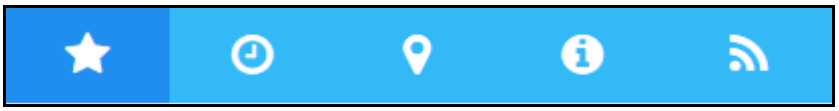
KUVA 48. Tyylien käyttäminen UiBinder-luokan XML-templaatissa

Etuliite ”g:” XML-elementin nimessä tarkoittaa, että kyseessä on GWT:n määrittelemä käyttöliittymäkomponentti, eli widgetti. Widgetille voi määrittää käytettävän CSS-luokan nimen *styleName*-attribuutilla elementtien *class*-attribuutin sijaan. (Kuva 48.) GWT hallitsee widgettien tapahtumankäsittelijät mm. *UiHandler*-annotaation kautta, mikä tekee tapahtumankäsittelijöiden kirjoittamisesta paljon nopeampaa ja tehokkaampaa. Widgettien käyttäminen vaatii kuitenkin, että UiBinder-luokka laajentaa GWT:n Composite-luokkaa ja kutsuu sieltä *initWidget*-metodia kuvan 49 mukaisesti. UiBinder-luokan XML-templaatissa taas on käytettävä juurielementtinä widgettiä, esimerkiksi kuvan 48 mukaista GWT:n HTMLPanel-widgettiä, joka kirjoitetaan XML-templaattiin *g:HTMLPanel*-elementtinä.

```
public class NavBar extends Composite {
  private static NavBarUiBinder uiBinder = GWT
          .create(NavBarUiBinder.class);
  interface NavBarUiBinder extends UiBinder<Widget, NavBar> {}
  public NavBar() {
    initWidget(uiBinder.createAndBindUi(this));
  }
}
```

KUVA 49. UiBinder-luokka

Seuraavaksi lisään alkuperäisen Jurassic Rock -sovelluksen navigaatiopalkin HTML-elementit UiBinder-luokan XML-templaattiin HTML-elementteinä ja painonapit GWT:n Button-widgetteinä. Näiden Button-widgettien on tarkoituksena toimia painonappina sovelluksen näkymän vaihtamiseksi, joten lisään samalla myös *ui:field*-attribuutin jokaiselle painonapille, jotta voin myöhemmin lisätä UiBinder-luokassa tapahtumankäsittelijän toiminnollisuuden toteuttamiseksi. Ennen tätä kuitenkin varmistan navigaatiopalkin tyylien näkymisen käynnistämällä sovelluksen *Super Dev Mode* -tilassa.



KUVA 50. Sovelluksen navigaatiopalkki

Kuvan 50 navigaatiopalkissa tummempi sininen on korostusväri, joka tarkoittaa valittua näkymää. Ideana tässä on asettaa korostuksen määrittelyt sisältävä CSS-luokka sellaiselle GWT Button-widgetille, jota sovelluksen käyttäjä painaa. Seuraavana luon tarvittavat *UiField*-annotoidut Button-tyyppiset kentät UiBinder-luokkaan ja lisään *UiHandler*-annotoidun metodin, joka luo tarvittavan tapahtumankäsittelijän sille parametrina annetuille widgeteille (kuva 51).

```
@UiField Button buttonFirstPage;
@UiField Button buttonSecondPage;
@UiField Button buttonThirdPage;
@UiField Button buttonFourthPage;
@UiField Button buttonFifthPage;

@UiHandler({"buttonFirstPage", "buttonSecondPage",
"buttonThirdPage", "buttonFourthPage", "buttonFifthPage"})
void onClick(ClickEvent event){
  Button sender = (Button) event.getSource();
  this.removeAllSelected();
  sender.addStyleName(Resources.
          INSTANCE.style().selected());
}
```

KUVA 51. NavBar-luokan kentät ja tapahtumankäsittelijä

UiHandler-annotaation avulla widgetissä itsessään, eli tässä tapauksessa GWT:n Composite-luokkaa laajentavassa UiBinder-luokassa oleville widgeteille on mahdollista asettaa sama tapahtumankäsittelijä. Tapahtuman lähdettäjän, eli ts. painonapin,

jota on painettu, voi määritellä *event*-objektista löytyvällä *getSource*-metodilla, joka puolestaan mahdollistaa sovelluksen toiminnan haarautumisen ja määrittelyn sen käyttäjän tekemien valintojen perusteella. Teen ensin kuitenkin toiminnollisuuden painonapin korostuksen tyylit sisältävän CSS-luokan vaihtamiseen.

Aina kun *UiHandler*-annotaatiolle parametrina annettua *UiField*-annotoitua widgettiä painaa hiiren vasemmalla painonapilla tai näytön ollessa kosketusnäyttö koskettaa, niin *onClick*-metodi tulee kutsutuksi. Button-tyyppisestä *sender*-objektista löytyy kaikki Button-luokalle määritetyt kentät ja metodit. Käytän siis *addStyleName*-metodia Style-rajapintaluokassa määritellyn *selected*-metodin palauttaman tyylin lisäämiseksi painonapille ja ennen tätä poistan kyseisen tyylin kaikista painonapeista tekemäni *removeAllSelected*-metodin avulla, joka kutsuu widgeteistä löytyvää *removeStyleName*-metodia *selected*-tyylin poistamiseksi kaikista *UiField*-annotoiduista Button-tyyppisistä kentistä. (Kuva 51.)

5.2.3 Suosikit-lista

Seuraavaksi teen Jurassic Rock -sovelluksen kaikkien näkymien yläpuolella sijaitsevan suosikit-listan. Lisään sovelluksen HTML-dokumenttiin, eli ”index.html”-tiedostoon uuden ankkurielementin tätä varten. Tämän jälkeen luon NavBar-luokan tapaan uuden Playlist-nimisen UiBinder-luokan GPE:n opasohjelman avulla. Tähän listaan lisättävät ja poistettavat elementit on tehtävä dynaamisesti käyttäjän tekemien valintojen mukaan, joten luon samalla myös PaylistItem-nimisen UiBinder-luokan valmiiksi. PlaylistItem-luokkaan ei ole tarvetta luoda tapahtumankäsittelijöitä, joten valitsen GPE:n *UiBinder*-opasohjelmassa käyttöliittymän perustuen HTML:ään.

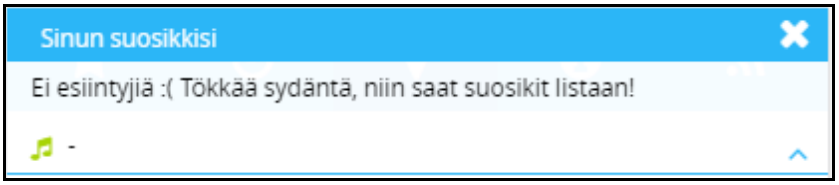
Lisään Playlist-luokan XML-templaattiin tarvittavan HTML:n ja *ui:field*-attribuutit tarvittaville elementeille. Katson myös tarvittavien CSS-luokkien nimet ja lisään ne GSS-tyylitiedostoon ja Style-rajapintaluokkaan. Määrittelen Playlist-luokassa *UiField*-annotoidun UListElement-tyyppisen kentän, joka on GWT:n DOM *ul*-elementti ja LIElement-tyyppisen kentän, joka on GWT:n DOM *li*-elementti, sekä SpanElement-tyyppisen kentän, joka on taas GWT:n DOM *span*-elementti. (Kuva 52.) Nämä eivät siis ole widgettejä, joten tapahtumankäsittelijää ei voi kirjoittaa kuvan 51 mukaan, vaan tapahtumankäsittelijä pitää kuvan 52 mukaisella tavalla ensin upottaa, jonka jälkeen sille voi asettaa kuuntelijan.

```
@UiField UListElement ulPlaylist;
@UiField LIElement liFavorites;
@UiField SpanElement spanGetDown;
public Playlist() {
  initWidget(uiBinder.createAndBindUi(this));
  EventListener listener = new EventListener() {
    @Override
    public void onBrowserEvent(Event event) {
      togglePlaylist();
      setPlayListHeight();
    }
  };
  Event.sinkEvents(ulPlaylist, Event.ONCLICK);
  Event.setEventListener(ulPlaylist, listener);

  Event.sinkEvents(spanGetDown, Event.ONCLICK);
  Event.setEventListener(spanGetDown, listener);
}
```

KUVA 52. Tapahtumankäsittelijä HTML-elementeille UiBinder-luokan rakentajassa

Kuvassa 52 näkyvä *onBrowserEvent*-metodi kutsutaan aina kun *click*-tapahtuma käynnistyy. Ensimmäisenä metodissa kutsutaan tekemään *togglePlaylist*-metodia, joka lisää tai poistaa CSS-luokkia sen mukaan onko soittolista avattu vai suljettu. Toisena kutsuttava *setPlaylistHeight*-metodi asettaa soittolistan korkeuden kohdalleen sen sisälle dynaamisesti asetettujen elementtien mukaan. (Kuva 52.)



KUVA 53. Sovelluksen suosikit-lista

Jurassic Rock -sovelluksen toiminnan mukaan etusivulla näkyvien artistien sydän-ikonia painamalla kuvassa 53 näkyvään suosikit-listaan voi lisätä omia suosikkejaan. Lisätyt suosikit siis tulevat olemaan luomani PlaylistItem-luokan XML-templaatin mukaisia elementtejä. Elementit tulevat Playlist-luokan XML-templaatin *ul*-elementin sisälle dynaamisesti ja ne ovat GWT:n DOM *div*-elementtejä, joten lisään PlaylistItem-luokan XML-templaatin *div*-juurielementin sisälle sydän-ikonin, sekä *span*-elementit esiintyjän nimeä ja esiintymispaikkaa varten. Lisään *span*-elementteihin myös *ui:field*-attribuutit ja niiden arvojen mukaiset *UiField*-annotoidut kentät Ui-

Binder-luokkaan, jotta pääsen myöhemmin manipuloimaan niitä Java-koodin puolelta. (Kuva 54.)

```
<!DOCTYPE ui:UiBinder
SYSTEM "http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder">
<ui:with type="fi.observis.ont.jonne.client.resources.Resources"
field="res"></ui:with>
<div>
  <i class="fa fa-heart"></i>
  <p>
    <span ui:field="spanBandName"></span>
    <span ui:field="spanBandStage"
      class="{res.style.playlist_stage_name}"></span>
  </p>
</div>
</ui:UiBinder>
```

KUVA 54. PlaylistItem-luokan XML-templaatti

Muokkaan PlaylistItem-luokan rakentajaa ottamaan String-tyyppiset objektit parametrina ja asettamaan ne merkkijonona SpanElement-tyyppisten objektien sisältämän HTML-elementin sisälle *setInnerHTML*-metodin avulla. Tällä tavoin uuden PlaylistItem-tyyppisen objektin instantioiminen pakottaa antamaan tarvittavan suosikin nimen ja esiintymispaikan ennen elementin asettamista soittolistaan. (Kuva 55.) Luon Playlist-luokkaan *addToFavorites*- ja *removeFromFavorites* -metodit suosikkien lisäämiseen ja poistamiseen. Luon myös Javan HashMap-tyyppisen objektin, jonka avulla voin helposti pitää ”kirjaa” suosikit-listaan lisätyistä objekteista sovelluksen sisällä.

```
@UiField SpanElement spanBandName;
@UiField SpanElement spanBandStage;

public PlaylistItem(String bandName, String bandStage) {
  setElement(uiBinder.createAndBindUi(this));
  spanBandName.setInnerHTML(bandName);
  spanBandStage.setInnerHTML(bandStage);
  if(bandStage.equalsIgnoreCase("Päälava")){
    spanBandStage.addClassName(Resources.INSTANCE
            .style().mainstage());
  }
}
```

KUVA 55. PlaylistItem-luokan kentät ja rakentaja

HashMap-tyyppisen objektin avaimena toimii PageItem-viittaustyyppisten objektien tarjoama String-tyyppinen merkkijono, joka toimii esiintyjän yksilöivänä tunnisteena. HashMap-tyyppisen objektin arvona taas on PlaylistItem-tyyppinen objekti. ”PageItem” on tekemäni rajapintaluokka, jonka kaikkien näkymien ”PageItem”-päätteiset UiBinder-luokat tulevat toteuttamaan. PageItem-rajapintaluokan toteuttavien UiBinder-luokkien on toteutettava mm. PlaylistItem-luokan rakentajan parametrina pyytämät String-tyyppisten objektien mukaisten merkkijonojen palauttavat metodit, eli esiintyjän nimen merkkijonona palauttava *getBandName*-metodi ja esiintymispaikan nimen merkkijonona palauttava *getBandStage*-metodi. Rajapintaluokka tarjoaa myös CSS-luokkien nimet sydän-ikonin tyylien vaihtamiseen.

Seuraavaksi valmistelen sovelluksen etusivuna toimivan näkymän, missä näkyy festivaaliesiintyjien kuvat, tiedot, esiintymisajat, sekä sydän-ikonit omissa komponenteissaan. Sydän-ikoni on painonapin sisällä, jota painamalla on mahdollista lisätä kyseinen esiintyjä sovelluksen suosikit-listaan. Käytän tässä myös suosikit-listan kohdalla käyttämääni ideaa lisätä festivaaliesiintyjät dynaamisesti festivaaliesiintyjille tarkoitettuun näkymään sovelluksen käynnistyessä.

5.2.4 Sovelluksen näkymät

Luon uuden UiBinder-luokan ja widgetteihin pohjautuvan XML-templaatin, joka toimii sovelluksen etusivuna ja kehyksenä dynaamisesti asetettaville komponenteille. Nimeän tämän UiBinder-luokan FirstPage-nimellä. FirstPage-luokan XML-templaatti sisältää vain *g:HTMLPanel*-juurielementin, jonka sisällä on sivun otsikko ja ankkurina toimiva HTMLPanel-widgetti festivaaliesiintyjien komponenteille.

Festivaaliesiintyjien näkymään asetettavat komponentit ovat myös itsenäisiä widgettejä, jotka sisältävät painonapin, jota painamalla voi lisätä kyseisen esiintyjän suosikit-listaan. Tähän painonappiin on siis mahdollista kirjoitta tapahtumankäsittelijä hyödyntäen *UiHandler*-annotaatiota. Luon uuden widgetteihin pohjautuvan UiBinder-luokan ja XML-templaatin, johon lisään Jurassic Rock -sovelluksen esiintyjien elementin HTML:n pohjaksi.

Aloitan näkymän tekemisen muokkaamalla ensin yhden festivaaliesiintyjän widgetin ja tekemällä tälle toiminnallisuuden lisäten tarvittavien elementtien ja widgettien

ui:field-attribuutit ja *UiField*-annotoidut kentät. Tällaisia kenttiä ovat mm. esiintyjän nimi, kuvan polku, esiintyjän kuvausteksti, esiintymisaika, sekä esiintymispaikka. Asetan UiBinder-luokan rakentajan ottamaan kenttien arvot parametrina, joka pakottaa antamaan tarvittavan datan luokan instantioinnin yhteydessä (kuva 56). Kuvan 56 rakentajassa myös lisätään samalla kyseinen Widget-tyyppinen luokkaan itseensä viittaava *this*-objekti etusivun näkymään kutsumalla GWT:n widgeteistä löytyvää *add*-metodia.

```
public FirstPageItem(String id, String bandName, String imgSrc,
String bandDesc, String bandGigTime, String bandStage) {
  initWidget(uiBinder.createAndBindUi(this));
  this.id = id;
  hBandName.setInnerHTML(bandName);
  imgBand.setAttribute("src", imgSrc);
  pBandDesc.setInnerHTML(bandDesc);
  pBandGigTime.setInnerHTML(bandGigTime);
  pStage.setInnerHTML(bandStage);
  WebApp.firstPage.panelItems.add(this);
}
```

KUVA 56. FirstPageItem-luokan rakentaja

Lisään vielä FirstPageItem-luokkaan tapahtumankäsittelijän kuvan 57 mukaisella tavalla käyttämällä hyväksi widgeteissä toimivaa *UiHandler*-annotaatiota. Tapahtumankäsittelijä lisää tai poistaa esiintyjän kutsumalla Playlist-luokan *addToFavorites*- ja *removeFromFavorites* -metodeita välittäen parametrina kyseisen PageItem-viittaustyyppisen objektin.

```
@UiHandler("buttonLike")
void onClick(ClickEvent e){
  if(buttonLike.getElement()
          .hasClassName(Resources.INSTANCE.style().liked())){
    WebApp.playlist.removeFromFavorites(this);
  }else{
    WebApp.playlist.addToFavorites(this);
  }
}
```

KUVA 57. FirstPageItem-luokan painonapin tapahtumankäsittelijä

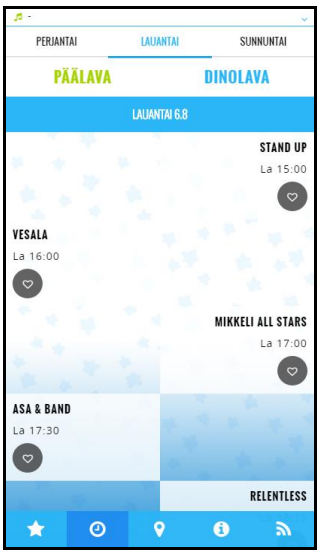
Näiden toimien jälkeen sovellus näyttää kuvan 58 mukaiselta. Etusivun näkymä on siis valmis ja navigaatiopalkin sekä soittolistan perustoiminnallisuus on olemassa. Seuraavana lisään toiminnon navigaatiopalkkiin, joka piilottaa etusivun ja näyttää toisen sivun. Kutsun myös etusivun avaavan painonapin *click*-metodia aktivoidakseni

sille määritellyn *click*-tapahtumankäsittelijän ohjelmallisesti näkymän alustamisen yhteydessä sovelluksen EntryPoint-luokassa.

KUVA 58. Sovelluksen etusivun ulkoasu

Toisen sivun näkymässä pitäisi näkyä festivaalien aikataulu, painonapit festivaalipäiville ja myös mahdollisuus lisätä esiintyjä suosikit-listaan. Käytän tämän näkymän rakentamisessa samaa ideaa kuin etusivun näkymässä, jossa lisäsin kehyksenä toimivalle sivulle dynaamisesti tarvittavat komponentit. Nämä komponentit tulevat olemaan FirstPageItem-luokan kaltaisia painonapilla varustettuja widgettejä. Kehyksenä toimiva UiBinder-luokka perustuu myös widgetteihin. Luon siis etusivun näkymän tapaan uuden SecondPage-nimisen UiBinder-luokan, joka toimii sivulle dynaamisesti asetettavien widgettien kehyksenä ja SecondPageItem-nimisen UiBinder-luokan, joka taas toimii sivulle dynaamisesti asetettavana widgettinä.

Siirrän alkuperäisen Jurassic Rock -sovelluksen HTML:n erotellen sen luotuihin UiBinder-luokkiin ja katson tarvittavat CSS-luokat, jotka lisään ”style.gss”-tiedostoon. Määrittelen CSS-tyylien luokkien nimet GSS:n mukaisesti ja lisään myös *ui:field*-attribuutit UiBinder-luokkien XML-templaattien widgeteille ja elementeille. Määrittelen myös tarvittavat *UiField*-annotoidut kentät UiBinder-luokkiin.



KUVA 59. Sovelluksen toisen sivun ulkoasu

Kun tyylit ovat kohdillaan, alan tekemään toiminnollisuutta päivien painonapeille (kuva 59). Painonapeista painamalla sivua vieritetään määritetyn ankkurielementin kohdalle. Määritän SecondPage-luokan rakentajassa jokaiselle painonapille *data-scroll*-attribuutin, jonka arvona on ankkurielementin *id*-attribuutin arvo. Painonapin *click*-tapahtumankäsittelijässä kutsutaan JSNI-metodilla jQuery-kirjaston *animate*-funktiota, joka tekee vieritysanimaation pysähtyen kyseiseen ankkurielementtiin (kuva 60).

```
native void scrollTo(String id, String classname) /*-{
  $wnd.$('body').animate({
              scrollTop : $wnd.$("#" + id).offset().top -
              $wnd.$("." + classname).outerHeight()
          }, 400);
}-*/;
```

KUVA 60. JSNI-metodi

SecondPageItem-luokan rakentajassa tehdään FirstPageItem-luokan rakentajan mukaiset toimenpiteet. SecondPageItem-luokan rakentajassa komponentit on kuitenkin lajiteltava päivien ja esiintymispaikan mukaan. Käytän molemmissa tapauksissa Javan *switch*-lauseketta, joista esiintymispaikan tapauksessa määrittelen *this*-objektin elementille lisättäväksi CSS-luokan nimen esiintymispaikan mukaan ja esiintymispäivän tapauksessa määrittelen oikean SecondPage-objektin HTMLPanel-widgetin, johon kyseinen *this*-objekti lisätään käyttäen *add*-metodia (kuva 61).

```
switch(gigDay.substring(0, 2).toLowerCase()){
  case "pe": WebApp.secondPage.panelFri.add(this); break;
  case "la": WebApp.secondPage.panelSat.add(this); break;
  case "su": WebApp.secondPage.panelSun.add(this); break;
}
```

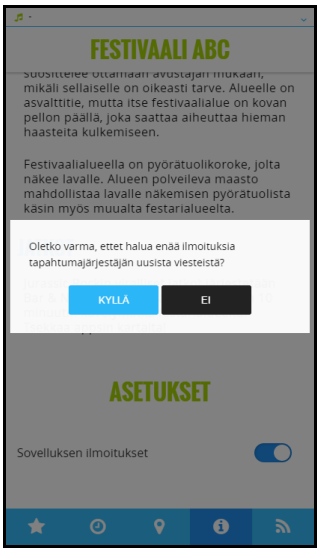
KUVA 61. SecondPageItem-luokan rakentajan *switch*-lauseke

Sovelluksen karttasivu, info-sivu, sekä sosiaalisen median syöte ja viestit -sivu ovat GWT:n puolelta yksinkertaisia ja niiden tekemiseen on mahdollista käyttää ja soveltaa kaikkia edellä läpikäytyjä asioita. Kuvassa 62 näkyvä karttasivu perustuu kokonaan ulkoiseen JavaScript-kirjastoon, joten sen toiminnallisuuden tekemiseen käytin paljon erilaisia JSNI-metodeita, joiden avulla oli mahdollista kutsua JavaScript-kirjaston funktioita. Karttasivua varten loin kuitenkin ThirdPage-nimisen UiBinder-luokan, jonka XML-templaatissa on juurielementtinä *div*-elementti ja sen *id*-attribuutin arvona *map,* jota kartan toiminnallisuuden toteuttava JavaScript-kirjasto käyttää karttaa varten. Käytin JSNI:tä välittääkseni JavaScript kirjastolle alustavia tietoja kartan näyttämistä varten, kuten tiedostopolun, jossa karttapohjaan asetettavat kuvat sijaitsevat, *latitude*- ja *longitude* -arvot, jotta kartta keskittyisi oikeaan maantieteelliseen sijaintiin.

KUVA 62. Sovelluksen karttasivu

Info-sivu on ilmeeltään melko yksinkertainen; se sisältää pääasiassa vain tekstiä, joka kertoo festivaalitapahtumasta, alueesta, aukioloajoista, lippujen hinnoista, leirinnästä jne. Näkymän asetukset-osion painonappi avaa *popup*-ikkunan, jolla varmistetaan käyttäjältä sovelluksen ilmoituksien kytkeminen pois päältä (kuva 63). Käytin *popup*-

ikkunan animointiin GWT:n Element-tyyppisistä objekteista löytyvää *getStyle*-metodia, joka tarjoaa työkalut CSS:n manipulointiin (kuva 64).



KUVA 63. Sovelluksen info-sivu

Kuvassa 64 näkyvä *panelPopup*-objekti on Widget-tyyppinen, joten objektista on ensin kutsuttava *getElement*-metodia saadakseen Element-tyyppisen objektin, kun taas samassa kuvassa näkyvä *divInner*-objekti on jo valmiiksi Element-tyyppiä laajentava DivElement-tyyppinen objekti, joten siitä voi kutsua suoraan *getStyle*-metodia. Kuvassa 64 näkyvän *getStyle*-metodin kautta on siis mahdollista manipuloida CSS-tyylejä käyttäen Javaa, joten JavaScript-kirjastojen, kuten esimerkiksi jQuery-kirjaston käyttö tähän tarkoitukseen JSNI:n kautta ei ole välttämätöntä.

```
public void hide() {
    panelPopup.getElement().getStyle().setOpacity(0);
    divInner.getStyle().setOpacity(0);
    divInner.getStyle().setProperty("transform", "translate3d(0,0%,0)");
    Timer t = new Timer() {
        public void run() {
            panelPopup.getElement().getStyle().setDisplay(Display.NONE);
            panelPopup.getElement().getStyle().setVisibility(Visibility.HIDDEN);
        }
    };
    t.schedule(200);
}
public void show() {
    panelPopup.getElement().getStyle().setDisplay(Display.BLOCK);
    panelPopup.getElement().getStyle().setVisibility(Visibility.VISIBLE);
    panelPopup.getElement().getStyle().setOpacity(1);
    Timer t = new Timer() {
        public void run() {
            divInner.getStyle().setOpacity(1);
            divInner.getStyle().setProperty("transform", "translate3d(0,-50%,0)");
        }
    };
    t.schedule(150);
}
```

KUVA 64. Popup-luokan *hide*- ja *show* -metodit

Sovelluksen viimeisen näkymän sisällä on kaksi näkymää: näkymä Twitter-viesteille ja järjestäjien lähettämille viesteille (kuva 65). Tein uuden UiBinder-luokan perustuen GWT-widgetteihin, sillä näkymässä on kaksi painonappia, jotka ovat GWT:n Button-widgettejä. Tämä UiBinder-luokka sisältää molemmat näkymät, joita on mahdollista vaihdella painonappien avulla. Painonapit ovat GWT:n Button-widgettejä ja ne myös näyttävät samalta kuin toisen sivun näkymän, eli aikataulunäkymän painonapit. Toiminnollisuudeltaan painonapit ovat kuitenkin enemmän navigaatiopalkin painonappien kaltaisia, eli kun yksi näkymä valitaan, niin muut näkymät piilotetaan.

KUVA 65. Sovelluksen sosiaalisen median syöte- ja viestit -sivu

Nyt sovelluksen kaikki näkymät ja käyttöliittymässä havaittava toiminnollisuus ovat valmiita. Käytin sovelluksen ulkoasun tekemiseen alkuperäisen Jurassic Rock -sovelluksen tyylejä, jotka on tehty liittämällä ulkoinen CSS-tiedosto *web*-sovelluksen HTML-dokumenttiin. Työssä käyttöönotettu GWT:n GSS-ominaisuus kuitenkin tuo paljon lisämahdollisuuksia tyylien tekemiseen ja niiden hallitsemiseen, joten muokkasin alkuperäisen sovelluksen tyylien rakennetta hyödyntämällä joitain GSS:n tarjoamia ratkaisuja.

5.2.5 Tyylien ja työn viimeistely

Viimeistelen työn muokkaamalla sovelluksen tyylejä esitelläkseni joitain GSS-ominaisuuden tarjoamia ratkaisuja. Ensimmäisenä julistan sovelluksessa käytetyt värit vakioiksi ”style.gss”-tiedostoon. Tästä on erityisen paljon hyötyä, sillä esimerkiksi koko sovelluksen värit on mahdollista vaihtaa muuttamalla vain yhden vakion arvoa, jolloin kaikki samaa vakiota käyttäneet CSS-luokat käyttävät siihen määriteltyä arvoa. Vakiot voivat pitää sisällään värien lisäksi mitä tahansa CSS-ominaisuuksille annettavia arvoja. Arvot voidaan myös määritellä Java-luokassa, jolloin ne on mahdollista tuoda GSS-tiedostoon antamalla *eval*-metodille parametrina täydellisen Java-luokan vakion nimen merkkijonona. (Kuva 66.)

```
@def JURASSIC_COLOR_MAIN #2bb6f7;
@def JURASSIC_COLOR_MAIN_RGBA rgba(43,182,247,0.95);
@def JURASSIC_COLOR_SECONDARY #a8d406;
@def JURASSIC_COLOR_HIGHLIGHT #208eee;

@def JURASSIC_BG_URL
url(../../img/jurassic/app/jurassic-stars.png);

@def COLOR_LIKED
eval("fi.observis.ont.jonne.client.settings.Settings.COLOR_LIKED");

button.like.liked {
  background-color: COLOR_LIKED;
}
```

KUVA 66. Vakioiden määrittelyt ja käyttö

GSS antaa mahdollisuuden myös muodostaa eräänlaisia CSS-ominaisuuksien ryhmiä, joille voidaan antaa arvot parametreina käyttäen itse nimettyä CSS-ominaisuutta. Tästä käytetään *mixin*-nimeä. *Mixin* on määriteltävä vakioiden tapaan, jotta sitä voidaan

käyttää myöhemmin. Otan käyttöön *size*-nimetyn *mixin*-määrityksen jokaisessa sovelluksen CSS-luokassa, jossa on käytetty *width*- ja *height*-ominaisuuksia. Tällä tavoin voin antaa arvot molemmille CSS-ominaisuuksille käyttäen vain yhtä *mixin*-ominaisuutta. (Kuva 67.)

```
@def mixin size(WIDTH, HEIGHT){
  width: WIDTH;
  height: HEIGHT;
}

button {
  color: #fff;
  padding: 10px;
  @mixin size(40px, 40px);
  background: none;
  border: none;
}
```

KUVA 67. *Mixin*-määrittely ja sen käyttäminen

GSS:n mahdollistamat vakioiden ja *mixin*-ominaisuuksien määrittelyt vievät tyylien tekemisen *web*-sovelluksessa aivan uudelle tasolle. GSS tuo UiBinder-sovelluskehyksen kanssa paljon mahdollisuuksia ja kuria tyylien määrittelemiseen ja niiden käyttämiseen *web*-sovelluksessa. Tätä kautta *web*-sovelluksen ylläpidettävyys säilyy hyvänä myös sen tyylien osalta.

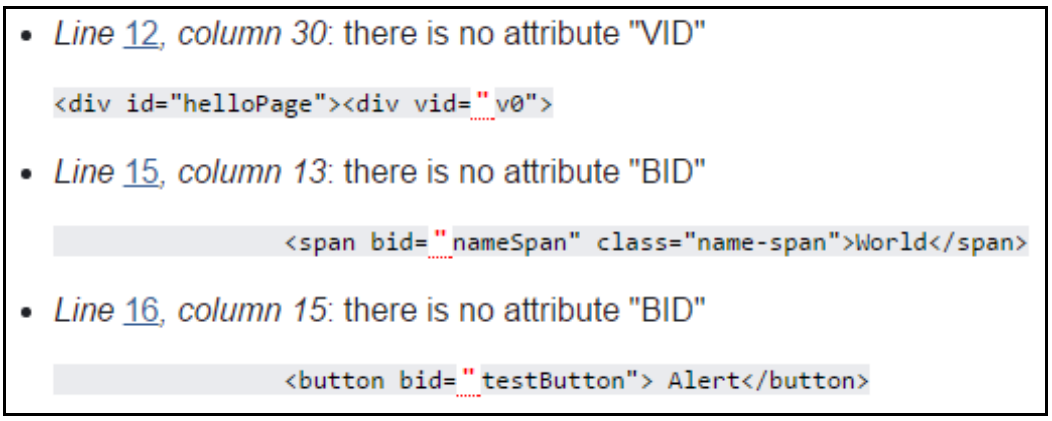
Viimeisenä dokumentoin työn lähdekoodia kirjoittamalla Javadoc-dokumentaatioita ja kommentteja englanniksi Observis Oy:n koko henkilöstöä varten, sillä työn yhtenä tarkoituksena on toimia esimerkkinä UiBinder-sovelluskehyksellä toteutetulle *web*-sovelluksen käyttöliittymälle. Tästä syystä olen pyrkinyt ottamaan työssäni huomioon erilaisia lähestymistapoja UiBinder-luokkien ja XML-templaattien määrittelyissä; esimerkiksi widgetteihin tai HTML-elementteihin perustuvat UiBinder-luokat ja XML-templaatit sekä niiden tapahtumankäsittelijät poikkeavat toisistaan ja niistä molemmista löytyvät myös omat käyttötarkoituksensa ja ominaisuutensa. Ei ole vain yhtä ”oikeaa” tapaa käyttää UiBinder-sovelluskehystä tai yleensä GWT:n tarjoamia työkaluja, vaan on osattava soveltaa ja löytää hyvät ratkaisut kokeilemalla erilaisia ratkaisuja niin onnistumisen kuin myös välillä epäonnistumisen kautta.

6 JOHTOPÄÄTÖKSET SOVELLUSKEHYKSISTÄ

Observis Oy:n kehittämän xUI-sovelluskehyksen ja GWT:n UiBinder-sovelluskehyksen periaatteet erotella käyttöliittymäkoodi muusta koodista ja generoida konkreettiset sitojaluokat koodigeneraattoreiden ajon yhteydessä ovat sovelluskehyksien välillä yhteisiä (liite 1; liite 2, 2). Molemmat sovelluskehykset siis mahdollistavat käyttöliittymän toteuttamisen deklaratiivisesti erilliselle templaatille, jonka koodigeneraattori kääntämisen yhteydessä sitoo sen omistavaan Java-luokkaan. Tämä poistaa pakollisuuden kirjoittaa käyttöliittymän toteuttamiseen tarvittavat käyttöliittymäkomponentit ja elementit Java-koodissa, mikä oli myös aikoinaan yhtenä perusteena Observis Oy:n tarpeelle kehittää xUI-sovelluskehys.

xUI-sovelluskehyksen mukainen yksinkertainen XHTML-templaatti voi olla kokemattoman käyttöliittymäsuunnittelijan silmään hieman tutumpi johtuen siinä käytetystä merkkauskielestä, joka on Kanervan (2016a) mukaan käytännössä laajennettua HTML:ää. Mutta xUI-sovelluskehyksen XHTML-templaatin ollessa yksinkertainen, se ei myöskään tarjoa UiBinder-sovelluskehyksen XML-templaatin tarjoamia etuja, kuten mahdollisuutta XML-nimiavaruuksien, GWT:n widgettien, UiBinder-ilmaisukielen ja GSS-ominaisuuden hyödyntämiseen. UiBinder-sovelluskehyksen toiminta ja UiBinder-luokkien instantiointi ilman xUI-sovelluskehyksen käyttämää riippuvuuksien injektointiin perustuvaa ja tehtaan kautta tapahtuvaa xUI-luokkien instantiointia saattaa myös vaikuttaa aluksi paljon yksinkertaisemmalta ja helpommalta ymmärtää.

xUI-sovelluskehys myös jättää sen XHTML-templaatissa käytetyt *bid*-attribuutit ja sen koodigeneraattorin luomat *vid*-attribuutit käännettyyn HTML-dokumenttiin, joten xUI-sovelluskehyksen tuottama HTML ei ole validia HTML:ää (kuva 68). UiBinder-sovelluskehys siivoaa pois mm. käyttämänsä *ui:field*-attribuutit ja XML-nimiavaruuksiin määritetyt etuliitteet, sekä tekee widgeteistä ja niille annetuista attribuuteista HTML-elementtien mukaisia luodessaan käännetyn HTML-dokumentin. UiBinder-sovelluskehys siis tuottaa validia HTML:ää, joka vaikuttaa parantavasti ohjelmiston laatuun.



KUVA 68. xUI-sovelluskehyksen tuottaman HTML-dokumentin validointi W3C *Markup Validation Service* -palvelulla dokumentin tyypin ollessa HTML 4.01 *Strict*

xUI-sovelluskehyksen antamat ilmoitukset virhetilanteissa olivat todella epäselviä, joten päivitin tämän opinnäytetyön aikana xUI-sovelluskehyksen lähteen tason GWT 2.5.1 -versiosta GWT 2.7.0-versioon ja lisäsin liitteessä 2, 2 näkyvien *check if the list of bids contains view fields*- ja *check if the list of view fields contains bids and there are no multiple bids* -toiminnoiden mukaiset tarkistukset. Mikäli nämä toiminnat ovat epätosia, niin xUI-sovelluskehyksen generaattori kaatuu ja antaa asiallisen virheilmoituksen. Lisäsin myös *find the template resource from the list of resources* -toiminnan mukaisen tarkistuksen, jossa tarkistetaan, että onko XHTML-templaatti olemassa ja onko se nimetty oikein. (mts. 2.) Virheilmoitukset kertovat täsmälleen mistä ongelma johtuu ja mitä ongelman korjaamiseksi voidaan tehdä. Näiden virheilmoitusten puuttuminen oli xUI-sovelluskehyksen huomattavimpia puutteita ja pelkästään näiden virheilmoitusten lisääminen vaikutti GWT-sovelluskehitysprojektien tuottavuuteen ja ylläpidettävyyteen merkittävästi.

UiBinder-sovelluskehyksen antamat virheilmoitukset sen sijaan ovat erittäin selkeitä ja niiden ansiosta sain ratkottua työssäni vastaan tulevat ongelmat nopeasti. Ei siis tarvinnut debugata sovelluskehyksen generaattoreita tietääkseen mistä ongelma johtui ja mitä sen korjaamiseksi voisi tehdä. Näillä virheilmoituksilla oli merkittävä rooli työn onnistumisen kannalta.

Myös missä GWT:n UiBinder-sovelluskehys on uusimman käytössä olevan GWT SDK -version tasolla, niin xUI-sovelluskehys käyttää nykytilassaan lähteen tasona jo vanhentunutta GWT 2.7.0 -versiota (xUI 2016). Tämä saattaa tulevaisuudessa aiheuttaa ongelmia uusien GWT-versioiden kanssa ja vaikuttaa ohjelmiston ylläpidettävyy-

teen, sillä lähteen taso ei ole yhtenäinen. Tämän osalta xUI-sovelluskehys vaatisi aktiivista kehitystä. Etenkin jo GWT 2.8.0 -version ollessa yksi suurimpia päivityksiä GWT-julkaisuista tähän asti ja sisältäessään paljon ominaisuuksia ja parannuksia (Janakiraman 2016). xUI-sovelluskehyksen lähteen tason nostaminen uusimpaan GWT 2.8.0 -versioon ei onnistunut suoraan päivittämällä, vaan päivitys vaatisi suurempien muutosten tekemistä lähdekoodissa.

Observis Oy:n käyttämä Eclipse IDE tukee GPE:n ansiosta GWT:tä ja UiBinder-sovelluskehystä hyvin. Eclipse IDE ilmoittaa jo ennen koodin kääntämistä mahdollisista XML-templaatista tai UiBinder-luokasta löytyvistä virheistä. GPE tuo myös mukanaan mahdollisuuden käyttää opasohjelmia, joiden avulla mm. uusien GWT:n UiBinder-luokkien luominen onnistuu vaivatta. Nämä toiminnot säästävät paljon aikaa ja vaivaa samalla lisäten GWT-sovelluskehitysprojektin tuottavuutta ja ylläpidettävyyttä. Samojen toimintojen toteuttaminen xUI-sovelluskehyksellä tarkoittaisi yritykselle oman Eclipse IDE -lisäosan kehittämistä, joka suorittaisi koodin validointiin liittyviä tarkistuksia ja mahdollistaisi uusien xUI-luokkien tekemisen opasohjelman avulla.

xUI-sovelluskehyksen kohdalla uuden xUI-luokan tekemiseen on mahdollista käyttää Eclipse IDE:n normaalin Java-luokan luomiseen tarkoitettua opasohjelmaa tai kirjoittaa se alusta alkaen itse. Kummassakin tapauksessa on kuitenkin tiedettävä xUI:n toiminta vähintäänkin luvussa 4.2.4 läpikäytyjen asioiden osalta, jotta uusien xUI-luokkien luominen onnistuisi sujuvasti ja tarkoituksenmukaisesti. Kynnys siis uuden xUI-luokan luomiseen käyttäen pohjana jotain toista vanhaa xUI-luokkaa on matala. Tämä voi osaltaan vaikuttaa negatiivisesti GWT-sovelluskehitysprojektien tuottavuuteen ja etenkin sen ylläpidettävyyteen nostaessaan mahdollisuutta inhimillisten virheiden tekemiseen. Virhe voi tapahtua esimerkiksi toistamalla vanhassa xUI-luokassa mahdollisesti tehtyjä virheitä jättämällä uuteen xUI-luokkaan ylimääräistä koodia, mikä voi puolestaan aiheuttaa väärinkäsityksiä ja sitä kautta lisätä virhetilanteita GWT-sovelluskehitysprojekteissa.

Sovelluskehyksen ensisijainen tarkoitus on kuitenkin helpottaa ja nopeuttaa ohjelmakoodin kirjoittamista tai kokonaisprosessia, joka ohjelmiston kehittämiseen vaaditaan ja sitä kautta lisätä tuottavuutta. Hyvä sovelluskehys tukee ohjelmistokehityksen tuottavuutta ja ylläpidettävyyttä minimoimalla käyttäjän tekemiä inhimillisiä virheitä tarjoten työkalut ohjelmakoodin testaamiseen ja tarkastamiseen. Hyvä sovelluskehys

myös tuottaa validia ja standardien mukaista koodia, mikä vaikuttaa parantavasti ohjelmiston laatuun. Hyvän ja sopivan sovelluskehyksen löytämiseen vaikuttaa myös muut yrityksen käytössä olevat työkalut, henkilöstö ja resurssit. Joidenkin sovelluskehyksien hyödyt saattavat tulla paremmin esille vasta projektien kasvaessa tarpeeksi suureen mittakaavaan.

Nämä seikat mielessä pitäen ja Observis Oy:n ollessa kasvava yritys, jonka tarpeet ohjelmistokehityksen tuottavuutta ja ohjelmistojen ylläpidettävyyttä sekä laatua koskien ovat myös kasvaneet, niin GWT:n UiBinder-sovelluskehys jättää nykytilassaan olevan xUI-sovelluskehyksen selvästi varjoonsa. Mikäli yritys ei halua varata tarvittavia resursseja xUI-sovelluskehyksen aktiiviseen kehittämiseen, niin siirtyminen UiBinder-sovelluskehykseen on selvää. Olen tällä opinnäytetyöllä osoittanut, että UiBinder-sovelluskehys pystyy käyttöliittymän toteuttamisen osalta samaan mihin xUI-sovelluskehys tarjotessaan paljon muita ohjelmistokehityksen tuottavuuteen ja ohjelmiston ylläpidettävyyteen, sekä laatuun positiivisesti vaikuttavia ominaisuuksia ja piirteitä.

7 PÄÄTÄNTÖ

Tämän opinnäytetyön perimmäisenä tarkoituksena oli selvittää UiBinder-sovelluskehyksen soveltuvuutta Observis Oy:n käyttöön siten, että se tukee myös yrityksen arviota kehittämänsä xUI-sovelluskehyksen jatkokehittämisen kannattavuudesta. Työ on siinä mielessä onnistunut, että yritys on siirtynyt tämän opinnäytetyön tulosten pohjalta käyttämään UiBinder-sovelluskehystä uusissa GWT-sovelluskehitysprojekteissaan, mikä tekee opinnäytetyön merkityksestä yrityksen tulevaisuutta ajatellen suuren. Tämä opinnäytetyö ja sen käytännön työn tulos on hyödyllinen siirtymisprosessin ja uusissa GWT-sovelluskehitysprojekteissa alkuun pääsemisen kannalta, sillä dokumentaatiota on paljon ja se käsittää perusteet kattavasti.

Vaikka Observis Oy siirtyykin käyttämään UiBinder-sovelluskehystä, niin dokumentaation tekeminen xUI-sovelluskehyksestä ja sen päivittäminen opinnäytetyön aikana ei kuitenkaan mennyt missään tapauksessa hukkaan. Yritys tukee ja ylläpitää vielä GWT-sovelluskehitysprojektejaan, joissa xUI-sovelluskehys on käytössä. Tämä opin-

näytetyö siis palvelee myös sellaisten projektien ylläpidettävyyttä ja lisää näin merkitystään.

Toivon että tämä opinnäytetyö toimii myös muiden ohjelmistokehittäjien apuna GWT:n ja UiBinder-sovelluskehyksen ymmärtämisen, sen käyttöönoton ja sen käyttämisen osalta. Suomenkielisiä GWT:tä ja UiBinder-sovelluskehystä käsitteleviä oppaita ei juuri ole, ja viimeisin niitä käsittelevä kirja on tämän opinnäytetyön pääasiallisena lähteenä käyttämäni Tacyn ym. kirjoittama ja vuonna 2013 julkaistu *GWT in Action, Second Edition*. Tie GWT:n ja UiBinder-sovelluskehyksen maailmaan ja niiden hallitsemiseen kulkee melko jyrkän oppimiskäyrän kautta.

Omalla kohdallani suurimpia haasteita tässä opinnäytetyössä oli ymmärtää ja hahmottaa xUI-sovelluskehyksen koodigeneraattoreiden, riippuvuuksien injektoinnin ja Velocity-templaattimoottorin yhteistoiminta. Se olisi ollutkin mahdotonta käsittää ilman liitteessä kaksi piirtämääni UML-toimintakaaviota. Erittäin suuri haaste oli myös aiheen rajaaminen niin, että tärkeimmät asiat tulisivat kirjoitettua ja dokumentoitua niiden menettämättä merkitystään työn lopputuloksen ollessa sovelluskehyksen valinnan kannalta kumpi tahansa.

Haluan vilpittömästi kiittää Observis Oy:tä ja yrityksen koko henkilöstöä tarjotessaan mahdollisuuden tämän opinnäytetyön tekemiseen. Olen oppinut tämän opinnäytetyön aikana todella paljon siinä läpikäytyjen asioiden lisäksi myös Javasta ohjelmointikielenä, ohjelmistokehityksestä ja sen kannalta tärkeistä asioista, kuten sen tuottavuuteen ja ohjelmiston laatuun, sekä ylläpidettävyyteen vaikuttavista tekijöistä. Tämän opinnäytetyön merkitys on myös sen tekijälle erittäin suuri.

LÄHTEET

CSS3 and GWT in perfect harmony. 2015. Arcbees. Diaesitys. http://www.slideshare.net/Arcbees/css3-and-gwt-in-perfect-harmony. Päivitetty 4.2.2015. Luettu 2.11.2016.

Eclipse. 2016. Eclipse. WWW-dokumentti. https://www.eclipse.org/ide/. Ei päivitystietoa. Luettu 22.11.2016.

Neon. 2016. Eclipse. WWW-dokumentti. https://projects.eclipse.org/releases/neon. Ei päivitystietoa. Luettu 22.11.2016.

Simultaneous Releases. 2016. Eclipse. WWW-dokumentti. https://projects.eclipse.org/releases. Ei päivitystietoa. Luettu 22.11.2016.

Configuring simple factories. 2016. GinFactoryModuleBuilder. Javadoc-dokumentaatio. https://github.com/fmgasparino/google-gin/blob/master/src/com/google/gwt/inject/client/assistedinject/GinFactoryModuleBuilder.java. Ei päivitystietoa. Luettu 2.9.2016.

Creating a type that accepts factory parameters. 2016. GinFactoryModuleBuilder. Javadoc-dokumentaatio. https://github.com/fmgasparino/google-gin/blob/master/src/com/google/gwt/inject/client/assistedinject/GinFactoryModuleBuilder.java. Ei päivitystietoa. Luettu 23.9.2016.

Defining a factory. 2016. GinFactoryModuleBuilder. Javadoc-dokumentaatio. https://github.com/fmgasparino/google-gin/blob/master/src/com/google/gwt/inject/client/assistedinject/GinFactoryModuleBuilder.java. Ei päivitystietoa. Luettu 2.9.2016.

Using the factory. 2016. GinFactoryModuleBuilder. Javadoc-dokumentaatio. https://github.com/fmgasparino/google-gin/blob/master/src/com/google/gwt/inject/client/assistedinject/GinFactoryModuleBuilder.java. Ei päivitystietoa. Luettu 23.9.2016.

Guermeur, Daniel & Unruh, Amy 2010. Google App Engine Java and GWT Application Development : Build Powerful, Scalable, and Interactive Web Apps in the Cloud (1). Birmingham: Packt Publishing Ltd.

Basic minification. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/doc/latest/DevGuideClientBundle.html#Basic_minification. Ei päivitystietoa. Luettu 26.9.2016.

Browsers and Servers. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/doc/latest/FAQ_GettingStarted.html#What_browsers_does_GWT_support? Ei päivitystietoa. Luettu 8.11.2016.

Compiling and Debugging. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/doc/latest/DevGuideCompilingAndDebugging.html#DevGuideJavaToJavaScriptCompiler. Ei päivitystietoa. Luettu 25.7.2016.

Deferred. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/doc/latest/DevGuideCodingBasicsDeferred.html. Ei päivitystietoa. Luettu 25.7.2016.

Generator Configuration in Module XML. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/doc/latest/DevGuideCodingBasicsDeferred.html#generator. Ei päivitystietoa. Luettu 22.9.2016.

GWT pronounced gwit. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org. Ei päivitystietoa. Luettu 1.6.2016.

Hello World. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/doc/latest/DevGuideUiBinder.html#Hello_World. Ei päivitystietoa. Luettu 15.8.2016.

How do I know which GWT modules I need to inherit – How To. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/doc/latest/DevGuideOrganizingProjects.html#DevGuideInheritingModules. Ei päivitystietoa. Luettu 22.9.2016.

Overview. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/doc/latest/DevGuideUiBinder.html#Overview. Ei päivitystietoa. Luettu 15.8.2016.

The GWT Release Notes. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/release-notes.html. Ei päivitystietoa. Luettu 5.10. 2016.

Using Eclipse. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/usingeclipse.html. Ei päivitystietoa. Luettu 10.10.2016.

Versions. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/versions.html. Ei päivitystietoa. Luettu 1.6.2016.

Why is my GWT-generated JavaScript gibberish? 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/doc/latest/FAQ_DebuggingAndCompiling.html#Why_is_my_GWT-generated_JavaScript_gibberish?. Ei päivitystietoa. Luettu 26.9.2016.

Writing Native JavaScript Methods. 2016. GWT. WWW-dokumentti. http://www.gwtproject.org/doc/latest/DevGuideCodingBasicsJSNI.html#writing. Ei päivitystietoa. Luettu 25.7.2016.

Janakiraman, Bashkar 2016. Google Web Toolkit Blog. Verkkoblogi. http://googlewebtoolkit.blogspot.fi/2016/10/gwt-28-released.html. Päivitetty 25.10.2016. Luettu 4.11.2016.

Janakiraman, Bashkar 2013. Google Web Toolkit Blog. Verkkoblogi. http://googlewebtoolkit.blogspot.fi/2013/07/gwt-news.html. Päivitetty 15.7.2013. Luettu 1.6.2016.

Kanerva, Ville 2016a. Sähköpostiviesti 4.8.2016. Teknologiajohtaja. Observis Oy.

Kanerva, Ville 2016b. Henkilökohtainen tiedonanto 9.11.2016. Teknologiajohtaja. Observis Oy.

Toimiala. 2016. Kauppalehti. Verkkolehti. http://www.kauppalehti.fi/yritykset/yritys/observis+oy/23735085. Ei päivitystietoa. Luettu 9.11.2016.

Keskitalo, Elina 2016. Mikkeliläisfirman sovellus käyttöön ainakin 12 kesäfestareille. Verkkolehti. http://www.lansi-savo.fi/uutiset/lahella/mikkelilaisfirman-sovellus-kayttoon-ainakin-12-kesafestarille-340444. Päivitetty 20.4.2016. Luettu 4.10.2016.

Kämppi, Tuomas 2016. Sähköpostiviesti 1.8.2016. Käyttöliittymäsuunnittelija. Observis Oy.

Introduction to the Standard Directory Layout. 2016. Maven. WWW-dokumentti. https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html. Ei päivitystietoa. Luettu 6.10.2016.

What is the POM? 2016. Maven. WWW-dokumentti. https://maven.apache.org/pom.html. Ei päivitystietoa. Luettu 6.10.2016.

McCluskey, Glen 1998. Using Java reflection. WWW-artikkeli. http://www.oracle.com/technetwork/articles/java/javareflection-1536171.html. Päivitetty tammikuussa 1998. Luettu 30.9.2016.

M2Eclipse. 2016. M2Eclipse. WWW-dokumentti. http://www.eclipse.org/m2e/. Ei päivitystietoa. Luettu 7.10.2016.

M2Eclipse 1.5. 2016. M2Eclipse. WWW-dokumentti. http://www.eclipse.org/m2e/documentation/release-notes-15.html. Ei päivitystietoa. Luettu 22.11.2016.

M2Eclipse 1.7. 2016. M2Eclipse. WWW-dokumentti. http://www.eclipse.org/m2e/documentation/release-notes-17.html. Ei päivitystietoa. Luettu 22.11.2016.

Observis Oy on Mikkelin Seudun vuoden yrittäjä 2013. 2013. Mikkelin Yrittäjät. Verkkouutinen. https://www.yrittajat.fi/etela-savon-yrittajat/mikkelin-yrittajat/a/uutiset/445852-observis-oy-mikkelin-seudun-vuoden-yrittaja-2013. Päivitetty 1.11.2013. Luettu 9.11.2016.

Haaste. 2016. Observis Oy. WWW-dokumentti. http://observis.fi/caset/jurassic-app-profest/. Ei päivitystietoa. Luettu 8.11.2016.

Kasvumme jatkuu. 2016. Observis Oy. Verkkouutinen. http://observis.fi/2016/05/kasvumme-jatkuu/2016/. Päivitetty 27.5.2016. Luettu 9.11.2016.

xUI. 2016. Observis Oy. Suljettu lähdekoodi. Päivitetty 22.9.2016. Luettu 14.11.2016.

Älykkäämpiä sovelluspalveluja. 2016. Observis Oy. WWW-dokumentti. http://observis.fi/yritys/. Ei päivitystietoa. Luettu 9.11.2016.

Ojala, Onni 2016a. Jurassic käynnistyi sateisissa merkeissä. Länsi-Savo 6.8.2016, 6.

Ojala, Onni 2016b. Jurassic_vieraat. Kuva. http://www.lansi-svo.fi/sites/default/files/styles/flexslider_full/public/images/news_item/jurassic_vieraat.jpg. Päivitetty 5.8.2016. Luettu 4.10.2016.

Creating Objects The Java™ Tutorials Classes and Objects. 2016. Oracle. WWW-dokumentti. https://docs.oracle.com/javase/tutorial/java/annotations/. Ei päivitystietoa. Luettu 22.8.2016.

Interfaces The Java™ Tutorials Learning the Java Language Interfaces and Inheritance. 2016. Oracle. WWW-dokumentti. https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html. Ei päivitystietoa. Luettu 26.9.2016.

Lesson: Annotations The Java™ Tutorials Learning the Java Language. 2016. Oracle. WWW-dokumentti. https://docs.oracle.com/javase/tutorial/java/annotations/. Ei päivitystietoa. Luettu 15.8.2016.

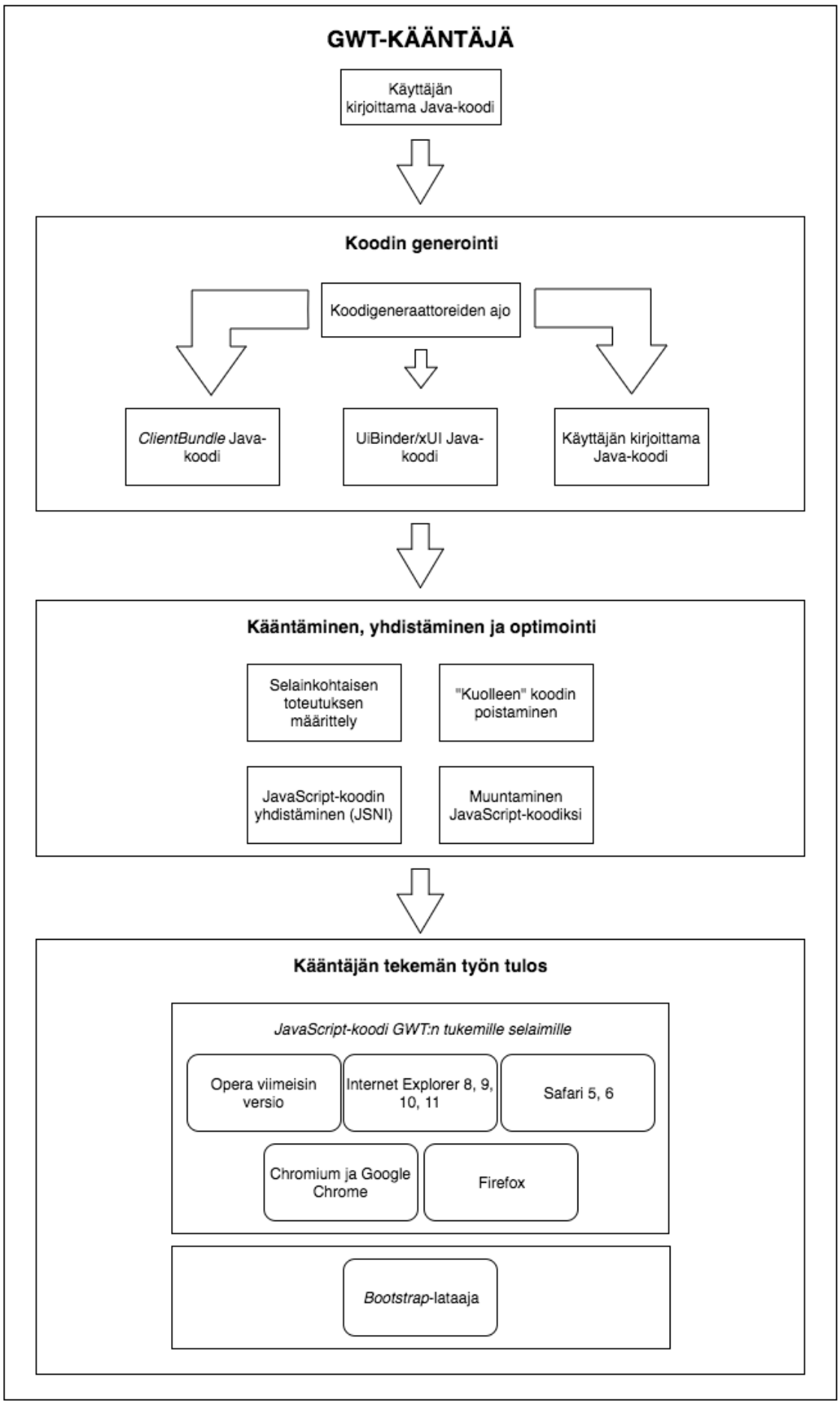
Tacy, Adam, Hanson, Robert, Essington, Jason & Tökke, Anna 2013. GWT in Action, Second Edition. New York: Manning Publications Co.

Venäläinen, Ville 2016. Sähköpostiviesti 9.10.2016. Ohjelmistokehittäjä. Observis Oy.

Vihavainen, Arto & Luukkainen, Matti 2016. Verkkokurssin materiaali. https://www.cs.helsinki.fi/group/java/s15-materiaali/viikko14/. Ei päivitystietoa. Luettu 9.11.2016.

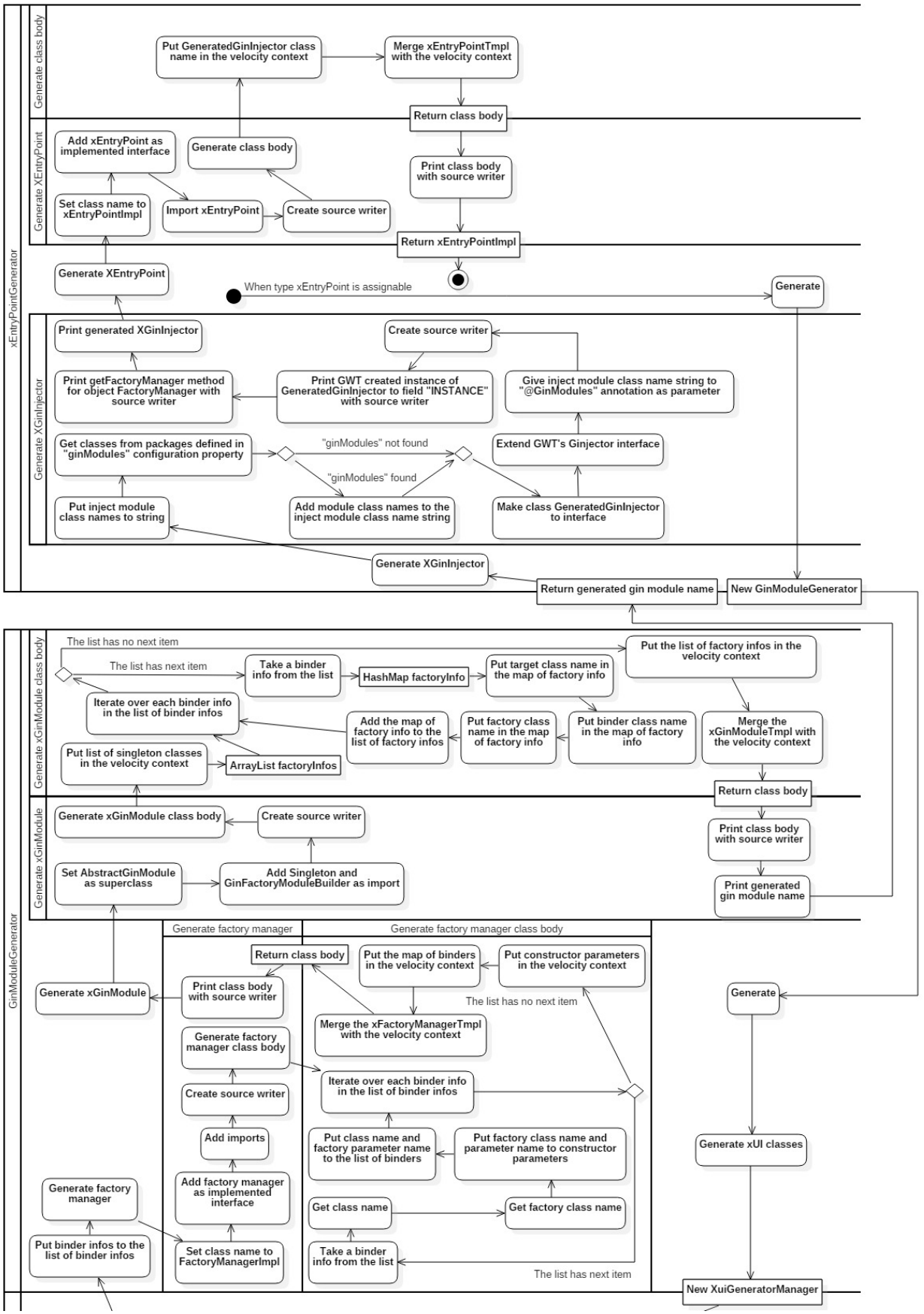
LIITE 1.

GWT-kääntäjän pääpiirteinen toiminta (Tacy ym. 2013, 8; Browsers… 2016)



LIITE 2(1).

UML-toimintakaavio xUI-sovelluskehyksen generaattoreista (xUI 2016)



LIITE 2(2).

UML-toimintakaavio xUI-sovelluskehyksen generaattoreista (xUI 2016)