TIE-20200 Ohjelmistojen Suunnittelu

Ryhmä 35	
Ilkka Kallioniemi	ilkka.kallioniemi@tuni.fi
Ville Saarinen	Ville.saarinen@tuni.fi
Henri Kasurinen	henri.kasurinen@tuni.fi
Lassi Vuotari	lassi.vuotari@tuni.fi

Sisällysluettelo

1. Johdanto

1.1. Tämän asiakirjan tarkoitus ja sisältö

Tämän asiakirjan tarkoitus on toimia toimeksiantajan, Tampereren Yliopiston, Finlandia-hiihtokilpailun tulosten tarkastelujärjestelmän toteutuksen kuvauksesta vastaavana dokumenttina. Tämän dokumentin pohjalta toivottavasti Ohjelmistojen Suunnittelu -opintojakson henkilöklunta pystyy arvioimaan ryhmämme onnistumista opintotavoitteiden valossa.

Tampereen Yliopisto on Tampereella toimiva Yliopisto, joka muodostui vuonna 2019 Tampereen yliopiston ja Tampereen teknillisen yliopiston muodostamasta korkeakoulusäätiöstä. Toimeksiantajalla on noin 5000 työntekijää ja 20600 opiskelijaa.

Toimeksiantaja on esiselvityksen pohjalta todennut, että tällainen järjestelmä on tarpeen, ja sen toteutus on päätetty antaa opiskelijaryhmille harjoitustyönä.

Tämän järjestelmän on laatinut Internet Explorers toimeksiantajan käyttöön.

Nykymuodossaan tämän asiakirjan tarkoitus on antaa yleiskuva toteutetusta tuotteesta ja tuotteen valmistamisen prosessista sekä itse asiakirjasta ja sen sisällöstä. Asiakirja kattaa käyttöympäristön, vaatimukset, terminologiamääritelmiä, lopulliset asiakasvaatimukset, järjestelmän kuvauksen ja refaktorointianalyysin. Asiakirjaa tulee yleisesti tarkastella prosessin kuvauksena.

1.2. Määriteltävä tuote, laajuus ja sen ympäristö

Toteutettava tuote on työpöytäpohjainen tulosanalysointityökalu. Sitä käytetään työpöytäasemilla. Ohjelman tarkoitus on helpottaa ja nopeuttaa toimeksiantajan Finlandia-hiihdon tulosten tarkastelua, sitä tukevaa työtä sekä johtamista.

Ohjelmalla käyttäjät löytävät nopeasti hiihtokisojen tiedot. Ohjelman selkeät näkymät (käyttöliittymät) ja graafit auttavat hahmottamaan olennaisen nopeasti ja toimimaan havaintojen pohjalta. Ohjelmalla on tarkoitus myös mahdollistaa kisailijoiden profilointia sekä tutkia kisakäyttäytymistä. Kaiken kaikkiaan ohjelman tarkoitus on auttaa käyttäjiään parantamaan kisatarkkailua ja tehdä työstä nykyistä miellyttävämpää.

Käytössä keskeistä on nopeus, helppous, luotettavuus ja ajantasaisuus.

1.3. Käyttäjät ja käyttötarkoitus

Ohjelma on ensisijaisesti asiakasyliopiston sisäinen työkalu, jota käyttävät eniten hiihdosta pitävä henkilöstö, paikallisjohto ja paikalliset IT-tukihenkilöt sekä jossakin määrin yrityksen muut työntekijät sekä emokonsernin puolella johto, tietohallinto ja avainhenkilöt. Myös yliopiston asiakkaat ja omistajat sekä viranomaistahot ovat käyttäjiä ainakin ohjelman tuottamien raporttien vastaanottajina.

Eniten järjestelmää käyttää informaatioteknologian ja viestinnän tiedekunta päivittäisessä työssään. Tietotekniikan laitokselle keskeistä kaikessa on nopeus ja helppous. Heille tärkeitä ohjelmiston näkymiä ovat haku ja kohteiden visualisointi, tiedot menneistä kiosoista, kisailijoiden kuuluminen ryhmiin tai kansalaisuuksiin, kisailijan kehittyminen eri aikaväleillä, ja profilointi sekä raportit, kuten raportit vuosituloksista seura- ja osallistujatasolla sekä raportit tuloksista.

Paikallisjohto käyttää ohjelmaa viikoittain. Ohjelma laatii raportteja tietosisällöistään ja antaa näkyvyyden toimintaan sekä kisailijatietoihin erityisesti parhaimipien sijoitusten osalta. Paikallisjohdolle tai muille sen määrittämille avaintahoille ohjelma voisi myös raportoida menestyksestä, säästä, kisaijuudesta tai kannassa tapahtuvista muutoksista sen mukaan, millaisia muutoksia ohjelman halutaan seuraavan.

Konsernin puolella ohjelmistoa käyttävät oletettavasti konsernin johto, tietohallinto ja avainhenkilöt. Tiedot tällaisesta konsernikäytöstä (mitä ja miten usein) saadaan vaatimusten keruussa paikallisjohdolta tai heidän nimeämiltään avainhenkilöiltä, kuten myös tiedot muista vaatimuksista.

Muu toimeksiantajan henkilöstö voi käyttää ohjelmistoa tulosten ihasteluun ja vedonlyöntinpäätösten tukemiseen. Käyttö on satunnaisempaa ja kausipainotteista.

Ohjelmiston ylläpitäjät yliopisto nimeää itse, ja he ovat oletettavasti yliopiston omia IT-tukihenkilöitä.

Käyttäjiä koulutetaan tässäkin dokumentissa ohjelmiston käyttöön, joten tästäkin näkökulmasta sen tulee olla helppokäyttöinen. Koulutuksessa käytetään metodeina ensin konkreettista kertomista – miten tieto virtaa ja tapahtumat etenevät fyysisinä objekteina – ja sitten todenmukaisten työtehtävien ohjattua hoitamista ohjelmalla.

Ohjelman käyttöliittymä on suomenkielinen, sillä valtaosalla henkilöstöä äidinkieli on suomi. Myös kieliversiointia englanniksi (joko kokonaan, vaiheistetusti tai osittain) on syytä harkita, erityisesti emokonsernin tarpeiden osalta. Määrätynlaista lisäarvoa toisi myös kieliversio, joka on höystetty runsaalla Porin murteella.

Ohjelmiston tulee tukea toimintoja, joilla hallitaan, ylläpidetään ja toteutetaan osittain automatisoiden kampanjoita, kilpailija- ja seuraryhmien profilointia, erilaisia kerroinohjelmia (laittomaan vedonlyöntiin kahvihuoneessa) sekä tietopyyntöjä.

Ohjelmiston yhteydet kuhunkin yksittäiseen järjestelmään tarkentuvat vaatimusten keruun myötä.

Ohjelmistossa käsitellään tietoja, kuten henkilötietoja. Tietoturva ja yksityisyyden suoja ovat ensiarvoisen tärkeitä seikkoja. Ne tulee huomioida yleisesti ohjelmiston suunnittelussa ja toteutuksessa sekä käytössä työtehtäviin, profilointi mukaan lukien.

Ohjelmiston tulee sisältää EU:n tietosuoja-asetuksen GDPR:n (General Data Protection Regulation) vaatimukset toteuttava hallintajärjestelmä.

Ohjelmiston valmiiksi toteutettavat oletuskäyttöoikeudet käyttäjätyypeittäin tarkentuvat yleisesti vaatimusten keruun myötä. Yleinen lähtökohta on se, että oihjelmistolla voidaan teknisesti määritellä kisailijat kuuluviksi useisiin ryhmiin, joille kullekin on teknisesti mahdollista tehdä kyselyitä lähes rajoituksetta. Tämän lähtökohdan vastaiset poikkeukset ovat olennaisia: ohjelmiston suunnittelussa on pyrittävä ehkäisemään vakavia inhimillisiä virheitä esimerkiksi käyttöoikeuksien myönnössä sekä estämään esimerkiksi lainsäädännön tai toimeksiantajan/konsernin käytäntöjen mukaan säilytettävien tietojen poistaminen. Yliopiston ja konsernin käytännöt tulee olla toteutettavissa ohjelmistoon, eli sen tulee teknisesti tukea niitä.

1.4. Määritelmät, termit ja lyhenteet

Termi	Määritelmä
johto	Tampereen yliopiston johtoryhmä
konserni	Tampereen yliopiston säätiö (”TAU+TAMK”)
konsernijohto	Tampereen yliopiston johtoryhmä
paikallisjohto	Tampereen yliopiston johtoryhmä
profilointi	asiakkaan tai markkinoinnin kohteen tunnistaminen tiettyyn ryhmään kuuluvaksi
toimeksiantaja	Tampereen yliopisto kokonaisuutena, mukaan lukien kaikki konsernin sisäiset Asihan sidosryhmät
ylläpitäjä	tietojärjestelmän käyttäjä, jolla on tavallisia käyttäjiä laajemmat käyttöoikeudet järjestelmään
näkymä	yksi nimetty käyttöliittymänäkymä, jonka avulla käyttäjä käyttää ohjelmistoa
taulu	yksi nimetty tietokantataulu
raportti	yksi nimetty raportti
toiminto	yksi nimetty toiminto

2. Vaatimusten keruusuunnitelma

2.1. Taustatilanne

Tulosseuranta toimeksiantajalla on aikaisemmin toiminut hyvin henkilökeskeisesti, epävirallisesti ja perinteisin, ei-dokumentoiduin menetelmin. Suurin osa kisailutiedoista oli pitkään toimeksiantajan erään työntekijän saatavilla ja hallussa siten, että hän on pystynyt huijaamaan kahvihuonevedonlyönnissä. Vanha järjestelmä ilmeisesti toimi aivan viime aikoihin asti hyvin, kunnes kyseinen työntekijä jäi kiinni ja joutui ostamaan kahvipaketin. Tässä tulee ilmi vanhan järjestelmän suurin puute – varmuutta ei ollut. Muut vanhan järjestelmän hyvät ja huonot puolet selvitetään vaatimusten keruussa.

2.2. Nykyisen dokumentaation analyysi

Kurssihenkilökunnan kehittämä kehyskertomus on toistaiseksi ollut ainoa ohjelmistoa varten tarjolla oleva dokumentaatio. Kehyskertomuksen perusteella olemme päätelleet mahdollisia sidosryhmiä ja alustavia vaatimuksia. Nämä ovat tällä hetkellä vain oletuksia, ja sidosryhmistä ja vaatimuksista saadaan varmuus assistentin tapaamisessa.

Kilpailevia valmiita ja laajempaa räätälöintiä vaativia Finlandia-hiihtoanalyysi-ratkaisuja on tarjolla markkinoilla niukasti. Näiden kartoitus ei ole vaatimusmäärittelyn alkuvaiheessa olennainen seikka.

2.3.Suositus: käyttötilanteiden seuranta jälkikäteen

Tuotteen valmistuttua suosittelemme toimeksiantajaa teettämään myös käyttötilanteiden seurannan. Siinä seurataan keskeisten käyttäjien, ainakin tieto- ja sähkötekniikanhenkilöstön työskentelyä vierestä esimerkiksi 4-7 h ja pyritään näin saamaan rehellistä käyttäjäpalautetta ja hiljaista tietoa kuuluviin. Tämän pohjalta ohjelmiston tilanne- ja tehtäväkohtaista käytettävyyttä parannetaan tai optimoidaan, mikä taas voi osaltaan lisätä tuottavuutta ja työtyytyväisyyttä ja viime kädessä parantaa asemaa kilpailijayliopistoihin.

On syytä huomata, että työskentelyä vierestä seuraavat henkilöt on koulutettava/valittava vastaanottamaan myös suodattamatonta, kielteistä ja vihaista palautetta oikealla ja vastaanottavaisella tavalla niin, ettei palautteen saamista koeta henkilökohtaisesti, vaan asiapohjalta. Palaute tulee nähdä ja kokea arvokkaana ja välttämättömänä – se on toimittajalle arvokasta tuotteen ja vastaavien tuotteiden jatkokehityksen kannalta, kahvihuonetyytyväisyydestä puhumattakaan. Lopulta vain kahvihuoneessa kävijät voivat kertoa johdolle, saiko tältä työhönsä sopivat työrukkaset.

3. Käyttötapaus

3.1. Esimerkkikäyttötapaus: Kisailijan etsiminen

(Tämä käyttötapauskuvaus on korkean abstraktiotason yleisluonteinen käyttäjätarina, joka kattaa useita käyttäjäryhmiä ja jonka muoto on lausetarinaa yksityiskohtaisempi käyttäjäskenaario.)

Toimijat: Henkilöstö, joka sattuu tuntemaan Finlandiassa hiihtäviä.

Alkutila: Käyttäjä on web-selaimella yrittänyt etsiä kaverinsa tietoja. Käyttäjä tarkastelee omaa päänäkymäänsä, joka on käyttäjäroolin mukainen.

Lopputila: Ohjelmisto tuottaa halutun raportin, lisää sen tuotettujen raporttien välisivuun, jonka sisältöä käyttäjä voi tarkastella ohjelmistosta.

Tapahtumat: Käyttäjä valitsee haluamansa valmiiksi määritellyistä kyselyn muokkauksista, jotka hänelle näkyvät. Kyselystä riippuen käyttäjä saattaa tehdä lisävalintoja, kuten määritellä raportin kattaman aikavälin ja muut parametrit.

Poikkeustilanteet:

> Käyttäjä hakee lisävalinnoillaan tietoja, joita ohjelmasta ei löydy. Tällöin ohjelma näyttää tyhjää ja antaa käyttäjän muokata hakuehtoja.

4. Käyttöliittymä

Asihassa on päänäkymät haulle, tuloksille sekä tulosten visualisoinnille. Kaikki päänäkymät näyttävät olennaiset tiedot yhdellä silmäyksellä, niin että päänäkymää seuraamalla käyttäjä tietää, mitä hänen on ohjelmassa tehtävä pysyäkseen tilanteen tasalla.

Päänäkymän tietosisällöt on jaettu **välilehtiin**, ja se, mikä välilehti näkyy näkymässä, on mukautettavissa. Välilehden otsikkoa napsauttamalla käyttäjälle avautuu kyseisen tietoruudun tietosisällöistä laajempi alinäkymä, joka täyttää koko selainsivun, eli alinäkymä avautuu päänäkymän selainsivun tilalle.

Käyttöliittymästä on jo luonnosvaiheessaan pyritty tekemään helppokäyttöinen, niin että sen toiminta avautuisi käyttäjälle mahdollisimman intuitiivisesti.

5. Vaatimukset

Vaatimusten keruu toteutettiin assistentille esitellyn alustavan suunnitelman mukaisesti. Keruussa käytettiin alustavasti suositeltuja keruumenetelmiä ja aikataulua.

Asiakkaalta saatiin seuraavat toiminnallisuusvaatimukset:

Haetaan n vuoden tuloksista tietyn ajan sisällä oleva. Esimerkki: hae vuoden 2011 tuloksista hiihtäjät, jotka ovat olleet maalissa 2-5 h sisällä. Parametrit: aika/aikaväli, vuosi/vuosiväli paluuarvo resultsWithinTimes(int year, Time lowTime, Time highTime)

Saman matkan tulosten vertailu kahdelta eri vuodelta. Parametrit: matkat/kilpailut, vuosi/vuosiväli paluuarvo resultsByDistance(int year, string distance Issues: Jos toisessa tuloksia ei ole / on yli 10000? / Kurssipuolelta vaatimusmäärittely rajannut ulos.

Kahden eri matkan tulosten vertailu samalta vuodelta. Esimerkki: hae vuoden 2011 tuloksista hiihtäjät, jotka ovat olleet maalissa 2-5 h sisällä. Parametrit: matkat/kilpailut, vuosi/vuosiväli paluuarvo resultsByDistance(int year, string distance)

Näytetään kullekin vuodelle osallistujien määrä sekä voittajan ja viimeiseksi maaliin tulleen ajat ja keskinopeudet jokaiselta matkalta.

Näytetään visuaalisesti aikakehitys tietylle urheilijalle annetulta ajanjaksolta.

Näytetään annetulta ajanjaksolta keskinopeus sijoituksen mukaan. Esimerkki1: Mikä on ollut keskinopeus vuosien 2011-2016 voittajalla? Esimerkki2: Mkä on ollut keskinopeus TOP 5 –urheilijoilla 2012-2017.

Näytetään parhaan miehen/naisen sijoitus tietyllä aikavälillä.
Esimerkki: Hae parhaan naisen sijoitus välillä 2010-2018.

Järjestetään tietyn vuoden tulokset seuran nimen mukaan aakkosjärjestykseen.

Huomioidaan mahdolliset virhetilanteet.
Esimerkki 1: Nimi alkaa pienellä alkukirjaimella.
Esimerkki 2: Tiettynä vuonna ei ole ollut käyttäjän hakemaa matkaa.

Näytetään urheilijoiden jakauma maittain

Näytetään kymmenen parasta joukkuetta. (Joukkue saadaan laskemalla 4 samassa seurassa olevan ajat yhteen)

6. Suunnitteluprosessi

6.1. Alustava suunnitelma

Alustava karkea suunnitelma laadittiin kokouksessa 22.1.2020. Tällöin todettiin, että toteutus jaetaan kahteen osastoon: ”backendiin” ja ”fronttiin”. Backend hoitaisi tiedonhakua ja säilömistä ja frontti hoitaisi käytön ja suuremman dokumentaation.

Alustavan suunnitelman mukaan data haettaisiin aina palvelimelta. Pohdittiin myös, että tehdäänkö suuri haku muistiin vai API, jolla haetaan käyttäjän määrittelemät tiedot. Suunnitelmaksi muodostui oma API. API:n palauttamaa paluuarvoakin mietittiin. Voittavaksi ajatukseksi muodostui funktion mukaan mukautuvaa paluuarvoa.

Visualisointitekniikoita arvioitiin ja todettiin, että QtCharts on käyttötarkoitukseemme soveltuvin. Tällöin myös tehtiin ensimmäinen Postman -kysely.

Rajapinnan synkronisuutta pohdittiin ja asynkronisuus todettiin soppivaksi, sillä halusimme ehdottomasti latauspalkin.

30.1.2020 saimme toisessa kokuksessa hyväksyttyä liitteen 1 mukaisen rajapinnan backendin ja frontin välillä.

6.2. GUI-suunnittelu

Koska organisaatiorakenteemme oli varsin joustava eikä liite 1 mukainen suunnitelma varsinaisesti sitonut meitä siinä esiintyvään MVC-malliin lähdimme tekemään ensimmäistä UI-protoa.

Totesimme QML-pohjaisten ratkaisujen olevan varsin hyviä mobiilimpien ratkaisujen käyttöön ja testasimme omassa Throwaway-protossa käyttäjäkokemusta välilehtiratkaisussa. Välilehdellisyys ei tuntunut vaikuttavan negatiivisesti käyttöön ja kokeilu osoitti käyttöliittymän intuitiivisuuden. Välilehdellisyys myös antoi tärkeiksi koettujen asioiden esittämiseen enemmän pinta-alaa.

Asiakasvaatimusten mukaisen UI-proton teimme QtWidget pohjaiseksi siitä syystä, että halusimme varmistua QMLbased-throwaway-proton positiivisen käyttäjäkokemuksen välittyvän myös tekniikkojen yli, ts. varmistua kokemuksen kumpuavan nimenomaan suunnitellullisista seikoista eikä toteutuksellisista seikoista.

GUI:ta on viety eteenpäin rapid prototyping -menetelmällä, missä erilaisia ominaisuuksia testataan ja katsotaan mistä saadaan paras käyttäjäkokemus vaadituilla toiminnallisuuksilla.

6.3. Backend-suunnittelu

Backendin suunnitteluprosessissa backend-haaran toimijat kokoontuivat analysoimaan vaadittuja toiminnallisuuksia ja keskustelun päätteeksi tuotettiin liitteen 1 kaavion karkea versio. Tämän pohjalta tehtiin feature-list, jota backend-haaran tekijät suorittivat ominaisuus kerrallaan.

Määräaviä suunnittelutekijöitä iteroiduissa versioissa on ollut nimenomaan hakujen luotettavuus, internethakuun kuluva aika sekä lokaalin datan robustisuus.

Backend-haaraa on viety eteenpäin toiminnallinen ominaisuus kerrallaan.

6.4. Refraktorointi

Ensimmäinen refaktoroitu asia tiedon varastointi. Käyttäjäkokemus etähakujen kanssa ei ollut kovin hyvä, joten käyttäjän kokemusta koetettiin lähteä parantamaan ajoittaisella isolla haulla ja lokaalin datan käytöllä.

Toinen refraktoroitu asia oli käyttöliittymän QML-pohjaisuus. Vaikka kehitystiimi näkee suuren lisäarvon, jota MVC tarjoaa MV:seen verrattuna, on kehitystiimi protoilun jälkeen tullut siihen päätökseen, että tässä käyttöympäristössä Qt-Widget -tyyppinen ratkaisu tarjoaa paremman käyttäjäkokemuksen.

Kielen käännettävyys loi meille myös pienen teknisen haasteen – filtterin luominen. Tämän ympäri päästiin validoinnin suorittavalla luokalla.

Lisäominaisuuksien luonti myös toi eteen tilanteen, jossa liitten 1 mukainen backEnd sai hieman lisämaustetta. Ajantasaiset lisäykset löytyvät Documentation -kansion Finlandia.svg tiedostosta. Käytännössä ollaan lisätty Logger ja Crypter -luokat.

6.5. DataHandler -luokan refraktorointiharjoitus

Totesimme suunnittelukokouksessa DataHandler-luokan olevan tarpeettoman suuri, sillä sen muutokset vaikuttivat niin moneen muuhun osa-alueeseen ohjelmassa. Päätimme toteuttaa uusimisen sijaan suuren refraktorointi- / uudelleensuunnitteluharjoituksen. Ajatuksena on siis kurssin oppimistavoitteita mukailevasti suunnitella DataHandlerin muodostama lohko uudelleen. Emme lähteneet koodissa refraktoroimaan luokkaa, sillä emme kokoeneet toteutuksen muuttamisen tuovan niin paljon lisäarvoa kurssin kontekstissa, sillä kyseessä on kuitenkin suunnittelupainotteinen kurssi.

Laajennettavuus nousi välittömästi kysymykseksi, sillä sitä kautta koimme esimerkiksi Jukolan tulosten analysoinnin olevan yksi CreateFactory luokan palauttama ConcreteFactory omine karaktireeneineen. Ohjelmistomme ei saa riippua olioiden luontitavassta, sen pitää olla konfiguroitava erilaisille olioille (Jukola / Finlandia) sekä DataHandlerista halutaan antaa ulospäin vain palveluiden rajapinnat.

Nyt listatuissa käyttötapauksissa käytännössä suurin muutos olisi filter-constantsien muutos, joten Builder luokkaa ei liene joustavin tapa toteuttaa ohjelmaamme vrt. sama rakentaja, mutta toiseen taloon laitetaan tiiliseinä ja toiseen hirsiseinä. Vaikka tämä ei ole Builder-luokalle opetettu käyttötapaus, koemme että Builderin välttäminen mahdollistaisi tulevaisuudessa ohjelman käytettävyyden erilaisiin muihinkin kohteisiin ja heijastaisi näin ollen enemmän OCP-henkeä.

Prototyyppi metodista emme kokeneet hyötyvämme, sillä keskustelun perusteella saisimme DataHandlerin perattua SOLID-mallin mukaiseksi ilman liian pitkää AbstractFactory-ketjua. Emme myöskään kokeneet Prototyypin tarjoaman dynaamisuuden tarjoavan tässä toteutuksessa kauheasti lisäarvoa, etenkään sen monimutkaisen alustuksen hinnalla. Prototyyppi vosi tarjota vrsin hyvän phjan, mikäli kaikkea tehtäisiin nyt alusta. Koimme kuitenkin refraktorointi-operaatioomme soveltuvammaksi AbstractFactory metodin.

Singletonin emme kokeneet tuovan mahdottomasti lisäarvoa toteutukselle, vaikkakin AbstraktiTehdas metodi ei sinällään estä singletonin käyttöä. Singletonin tarjoamat hyödyt, kuten oikeaan (mahdollisesti ainoaan) ilmentymään pääsy tulivat tiukasta MV-mallin käytöstä tapahtumien hallinnassa ja tämän tarjoamasta Qt:n parent-child hengestä. Käytimme omaa nimiavaruutta, joka rajasi globaalien muuttujien käyttöä. Halusimme myös varmistaa OCP:n toteutumisen ja hyvän yksikkötestauksen mahdollistamisen. Pelkäsimme, että singletonin käyttö olisi saattanut haitata näiden core-arvojen näkymisen tuotteessa.

Emme kokeneet että Sovitin-suunnittelumetodia tarvittaisiin tämän hetkisellä vaatimusmäärittelyllä ohjelmaamme, vaan että nykyisin tiedossa oleviin käyttökohteisiin saataisiin AbstraktinTehtaan-metodilla muokattua sopiva KonkreettinenTehdas. InternetExplorers pistää kuitenkin sovittimen välimuistin sijasta kirjanmerkkeihin, sillä asiakkaan innostuessa tähän saatettaisiin joutua palaamaan.

Refraktoroitu DataHandler voisi käytännössä mieltää osittain silta-metodilla höystetyksi luokaksi. UI:n MV-mallissa abstraktin tason herätteeseen aletaan reagoimaan ja DataHandler on herätteeseen vastaava toteuttaja. Refraktoroitu

DataHandler olisi myös varsin kerroksittainen toteutus. Keskustelimme myös AbstraktinTehtaan tuottamasta tehtaasta ja siitä onko abstraktille toteuttaja-luokalle tarvetta. Konsensus oli, että abstraktisuuden lisääminen tässä tapauksessa olisi liian kova hinta hyötyyn nähden. Sitouduimme kuitenkin toteuttamaan asioita sillan tarjoamia hyötyjä punniten, etenkin Control-luokan puuttumisen valossa käyttäjäsyötteiden validointiin ja debuggaukseen.

Emme kokeneet, että saisimme Decorator-metodilla parempia tuloksia tämän osakokonaisuuden refraktoroinnissa, etenkään verrattuna hyvin mietittyihin ja toteutettuihin AbstrakteihinTehtaisiin nähden. Koimme tehtaidenkin kautta pääsevän samalla rajapinnalla haluttuihin lopputuloksiin. Myöskin yksikkötestattavuus ja tulosten toistettavuus aiheutti huolta. Decorator herätti keskustelua eri UI:n toiminnallisuuksien toteuttamisen suhteen.

Emme myöskään kokeneet Facade-metodin tarjoavan meille merkittäviä hyötyjä, sillä halusimme palvelullemme sopivan ja yhtenevän rajapinnan käyttökohteesta riippumatta. Tämä sisältää mielestämme myös sen, että palveluiden käyttäjille ei tarvitse tarjota palveluita alijärjestelmästä. Myöskin uskoimme Fasaadiuuden näkyvän yhtä jumal-tasoisena toteutuksena kuin nykyinen DataHandler.

Välittäjä-metodia emme kokeneet sopivaksi, sillä DataHandler-kokonaisuus keskustelee usean muun olion kanssa. Myöskään DataHandlerin sisäsiten komponenttien ei tarvitse olla tietoisia toisten komponenttien sisäisestä tilasta. Näin ollen emme kokeneet suurta hyötyä DataHandlerin refraktoroinnissa Observer-metodista.

Yhteenvetona voitaneen sanoa, että näkemyksemme lähtivät liikkeelle Factory-metodista ja GOF suunnittelumallien analysoinnin jälkeen vaihtui AbstractFactory-metodiin. Ryhmä tunnisti muista malleista runsaasti synergisia etuja jatkotyöhönsä.

6.6. Toiminnan dokumentointi

Ryhmän sisällä viestintäväylänä on käytetty Telegram -sovelluksessa omaa ryhmäkeskustelukanavaa, olemme suorittaneet aikataulullista seurantaa sekä sisäistä dokumentointia Trellossa korttikohtaisesti ja olemme pyrkineet mahdollisimman selvään gitlab -toimintaympäristön käyttämiseen mm. selvillä commit viesteillä sekä issuetabin ahkeralla käytöllä.

7. Ohjelmiston kuvaus

7.1. Ulkoinen rajapinta

Paikallinen datahandler kutsuu tarvittaessa FinlandiaAPI luokkaa, joka puolestaan laittaa FinlandiaCallerin töihin hakemaan dataa tulospalvelusta.

FinlandiaCaller tekee hakuja APIsta saaduilla parametreillä ja parametrein saaduista sivusta parsii HTTP:n joukosta oleellisen tiedon

FinlandiaCaller myös tarkkailee onko tuloksia liikaa.

FinlandiaCaller on toteutettu tarkkailemalla Postmanilla tehtyjä hakuja ja parsimalla dataa HTTP -koodista.

7.2. Korkean tason kuvaus

7.2.1. Yleinen kuvaus

Liite 1 kuvaa hyvin ohjelmiston toimintaa UI:n alla.

Main -osuus kutsuu MainWindowin ja Finlandia -ikkunan.

Finlandia kutsuu DataHandlerin. Datahandler tarkistaa paikallisen datan ehyyden LocalAPIn kautta ja tarvittaessa hakee uudestaan datan FinlandiaAPIn palveluilla, jotka on toteutettu FinlandiaCallerissa. DataHandler pääsee käsiksi lokaaliin dataan LocalAPIn kautta.

7.2.2. Käytetyjen komponenttien ja teknologioiden kuvaus

Finlandia -luokka on QMainWindow. Se utilisoi STL:n vectoria ja QtCharts -kirjastoa. Sen tehtävä on toimia MV-mallin mukaisena luokkana, jonka kanssa käyttäjä suorittaa interaktiot ja jonka käyttäjä näkee.

DataHandler on periytetty QObjectista. Se hyödyntää QStringiä, Qhash -kirjastoa, vectoria, stringiä, unordered_mappia, functional -kirjastoa, QUrlQueryä, QHttpMultiParttia, QNetworkProxyä, QFileä, sstreamia, QXmlStreamReaderia, ctimeä, ratiota, chronoa, threadausta, settiä, algorithm -kirjastoa ja cctypeä. Syy DataHandlerin riippuvuusraskaudelle on sen tehtävä. DataHandlerin tehtävä on käsitellä dataa. Tarkastelimme kriittisesti luomaamme luokkaa suunnittelutapaamisessa. Keskustelimme onko luokalla tehtäviä, jotka voisi eriyttää johonkin muualle hoidettavaksi. Tulimme palaverissa siihen tulokseen, että on tarkoituksenmukaisinta ja synergiaedullisinta pitää datan käsittely yhdessä luokassa.

LocalAPI on periytetty QObject, joka hyödyntää mappia, QStringiä, vectoria, QCoreApplicatrion-, QFileInfo-, QDir-, QTextStream, QDirIterator-, QcryptographicHash- sekä QTextCodec -kirjastoja. Sen tehtävä on hoitaa ohjelmiston ja paikallisen datan välinen vuorovaikutus.

FinlandiaAPI on periytetty QObject, joka hyödyntää mappia, unordered_mappia, vectoria, stringiä, sstreamia, QXmlReaderiä, QUrlQueryä, QHttpMultiParttia,

QNetworkProxyä, QEventLooppia, QTimeria sekä memory- ja mutex -kirjastoja. APIn tehtävä on ohjata FinlandiaCalleria ja palauttaa DataHandlerille haluttu data.

FinlandiaCaller on periytetty QObject, joka hyödyntää mappia, unordered_mappia, vectoria, QTimeria ja threadia sekä memory-, atomic- ja mutex -kirjastoja. Sen tehtävä on suorittaa hakuja tulospalveluun.

Logger käyttää mutexia ja QStringia. Sen tehtävä on kirjoittaa logeja.

Crypter käyttää vectoria, stringia QCryptographicHashia ja QStringia. Sen tehtävä on kryptata lokaali data.

InterfaceFilter käyttää mappia, iostreamia, QStringia, vectoria, ja QExceptionia. Sen tehtävä on käsitellä interfaces luomia filttereitä.

7.3. Rajat ja sisäpinnat

Ohjelmassa dataa välittyy jatkuvasti komponenttien yli. Mahdollisesti käynnistystilanteessa haetaan data internetistä, jolloin dataa liikkuu vectoreittain FinlandiaCallerilta FinlandiaAPI:lle, joka vaatii palvelua Callerilta ja joka vastaa DataHandlerin vaatimuksiin, kaikki API:lta saatu data liikkuu mapissa DataHandlerille. DataHandler puolestaan syöttää datan LocalAPI:lle, joka tallettaa dataa kansiorakenteisesti paikallisesti saatavaksi. Finlandia -luokka pääsee käsiksi dataan DataHandlerin kautta. DataHandler palauttaa vectoreittain dataa.

7.4. Komponenttien ja vastuiden kuvaus

Finlandia -luokka toimii MV-mallin mukaisena luokkana, jonka kanssa käyttäjä suorittaa interaktiot ja jonka käyttäjä näkee. Finlandia vastaa käyttäjän tekemiin hakuihin ja valintoihin. Finlandia tarjoaa käyttöliittymän Finlandia.ui:lla, jossa on kaikki käyttäjälle mahdolliset hakuvaihtoehdot. Finlandia.cpp reagoi hakuparametreihin, kun käyttäjä painaa pushButtonia. Finlandia myös tallentaa tehdyt haut ja hakujen perusteella saadun datan.

Finlandia validoi luomansa filtterit InterfaceFilterillä.

DataHandler käsittelee kaikki muistissa tapahtuvat käsittelyt, jotka liittyvät valmiiseen tulosdataan. DataHandler vastaa Finlandia osoittamiin pyyntöihin getDataWithFilter funktiolla ja applyFilterToData funktiolla. Se lataa välimuistiin datan useassa threadissa. Sillä on monia sisäisiä palveluita, joilla se käsittelee dataa filteröimällä.

LocalAPI vastaa datan paikallisesta talletuksesta, paikallisesta lukemisesta ja datan koherenttiuuden tarkistuksesta. SaveData ylikrjoittaa vanhan datan. LoadData tarkistaa lokaalin datan ja sitten lataa sen välimuistiin.

LocalAPIn lokaalioperaatiot pyörähtävät Crypterin kautta, joka vastaa data en- ja dekryptaamisesta.

FinlandiaAPI vastaa siitä, että FinlandiaCallerit toimivat oikein ja oikeilla parametreilla. Se threadaa hakuprosessin ja johtaa Callereita. FinlandiaAPI kerää dataa Callereiltaan ja ylläpitää omaa tietokantaansa muistissa.

FinlandiaCaller vastaa yksittäisen haun tekemisestä etätietovarastoon. Se tekee yksittäisiä hakuja etävarastoon ja parsii sitä. Sillä on myös tarkasteluja ja protokollia virheellisten hakutulosten kanssa toimimiseksi.

Kaikki loggaamisen arvoiseksi koettu loggaantuu Logger luokan kautta.

8. Ympäristö

8.1. Ympäristön asettamat vaatimukset

Ohjelmisto on työpöytäpohjainen ja sen toiminta perustuu Internetin avulla tapahtuvaan hakuun ensimmäisellä kerralla, sekä tietojen vanhentuessa. Ohjelmistossa on oltava paikallinen tietovarasto, joka takaa sujuvan työnkulun hitailla Internet-yhteyksillä ja tilanteissa, joissa Internet-yhteyttä ei hetkellisesti ole. Ohjelmiston tulee hallinnoida tiedonsiirtoa, tarkemmin replikointia, paikallisen tietovaraston ja keskustietovaraston välillä äärimmäisen varmatoimisesti, tehokkaasti ja edellyttämättä käyttäjän toimia. Esimerkiksi käyttäjän ei tule joutua huolehtimaan siitä, että tietoja haetaan.

Ohjelmisto vaatii ympäristöltä Internet-yhteyden. Tarkka työpöytäympäristö jätetään käyttäjän määritettäväksi. Toimeksiantaja haluaa käyttää ohjelmaa työpöytä-tietokoneilla (Linux (Redhat)), kannettavilla Macbook-tietokoneilla (OS X, uusimmat kaksi versiota) sekä omilla tietokoneillaan (??). Aluksi ohjelmasta toteutetaan mahdollisimman monilla oikeilla työpöydillä toimiva versio, jonka suhteen tärkeimmät päätelaitteet ovat linux-työpöytätietokone (desktop workstation, ei-etä) verraten suurella näytöllä, kannettava Linux-tietokone järkeväresoluutioisella näytöllä sekä kannettava MacBook-tietokone. Tämän jälkeen ohjelmiston kehitys myös puhelimella ja tabletilla toimivaksi jää avoimeksi. Tällöin on syytä harkita, kehitetäänkö ohjelmistosta yliopiston oma, puhelimelle ja tabletille natiivi app-sovellus, jota ei asenneta sovelluskaupan kautta vaan sisäisenä sovelluksena asianmukaisella sertifikaatilla.

9. Jatkokehitysajatukset

Ohjelmistoa voi jatkojalostaa eli jatkokehittää sen jälkeen, kun ensiversio on otettu tuotantokäyttöön, sillä toimeksiantaja omistaa lähdekoodin. Jatkokehityksen perustapoja on kaksi: voi teettää lisäosia (moduuleja tai komponentteja), jotka lisäävät uusia toiminnallisuuksia tai ominaisuuksia, tai hankkimalla erillisiä ohjelmistoja, jotka sitten integroidaan suoraan ohjelmistoon tai sen ympäristöön.

Markkinoilta löytyy muun muassa ratkaisuja (ohjelmistoja ja mahdollisesti palveluita), jotka analysoivat vedonlyöntikertoimia ennustemallintamisen tueksi. Tällaiset ratkaisut hyödyntävät erilaisia tekniikoita. Toimialalla painopiste on siirtynyt viime aikoina massiivisista datamääristä oppivista järjestelmistä ekstrapoloitua analysointia tekeviin järjestelmiin. Tämänlainen ratkaisu saattaisi tuottaa etenkin kahvihuonekäyttäytymisen kannalta todella hyödyllistä tietoa.

Ohjelmistoon voi teettää lisäosia, mikäli markkinasuhdanteiden muutosten myötä syntyy tarpeita, joihin toimeksiantajan johdon mielestä on mielekästä reagoida kustannusten valossa. Jatkokehitysprojektia varten kannattaa teetättää vaatimusmäärittely, jonka toteutuksen kilpailuttaa, jotta esimerkiksi ohjelmistorajapintojen yhteensopivuudet ja erilaisten markkinoilla olevien valmiiden ratkaisujen hyödyntämismahdollisuudet selvitetään perusteellisesti ja asianmukaisesti.

Toimeksiantaja on kasvuyliopisto, jonka tarpeet kasvavat tulipalotoiminnan kasvun myötä. Käyttöönottohetkellä ohjelmisto on monipuolisuutensa puolesta hieman ylimitoitettu nykyisen toiminnan kokoon suhteutettuna. Tulipalotoiminnan kasvu, muuttuvat tarpeet, muuttuva ympäristö, teknologinen kehitys ja Ohjelmiston skaalaaminen muihin konsernin jäseniin (mahdollisesti yrityskohtaisesti mukautettuna) saattavat edellyttää määriteltyä monipuolisuutta ja laajuutta nopeastikin. Ohjelmiston kehitystyössä eräs johtoajatus on ollut se, etteivät perustavaa laatua olevat järjestelmäratkaisut tule esteiksi kasvulle, muokattavuudelle ja jatkokehitykselle. Esimerkiksi tietokannan on vaatimusmäärittelyn mukaisesti skaalauduttava senhetkiseen palvelinkapasiteettiin, eli tietokannan laajentumisen tulee lähtökohtaisesti edellyttää vain palvelinkapasiteetin lisäämistä.

Koska toimeksiantaja omistaa lähdekoodin, sitä voidaan hyödyntää jatkossa siis myös muissa konsernin tytäryhtiöissä. Koska ohjelmistoon liittyy monta erillistä tietojärjestelmää, jotka on integroitu, saattaisi olla hyödyllistä samalla yhtenäistää konsernin sisäisiä ohjelmistorajapintoja, jos ohjelmisto osoittautuu siinä määrin hyödylliseksi, että sen skaalaaminen koetaan konsernin strategiatasolla mielekkääksi.

10. Itsearviointi

10.1. Arvio alkuperäisen suunnitelman seuraamisesta

Kappaleessa 6.4 Refraktorointi analysoidaan teknisiä muutoksia suunnitelmaan. Tiedon varastointi paikallisesti on parantanut huomattavasti käyttäjäkokemusta ja uusi käyttöliittymä on ollut käytettävyydeltään hyvä.

Alkuperäiseen suunnitelmaan kuulunut työnjako on pysynyt varsin hyvin. Alkuperäisen työnjaon mukaisesti Ilkka ja Ville hoitavat backendin, Lassi ja Henri FrontEndin ja dokumentoinnin.

Toiminnallisuuksia on saatu toteutettua todella hyvin suunnitelmaan nähden.

Suunnitelma mahdollistaa etenkin backEnd -puolella hyvin iteratiivisen parantamisen. FrontEnd -puolella joudutaan välipalautukseen mennessä elämään varsin kädestä suuhun.

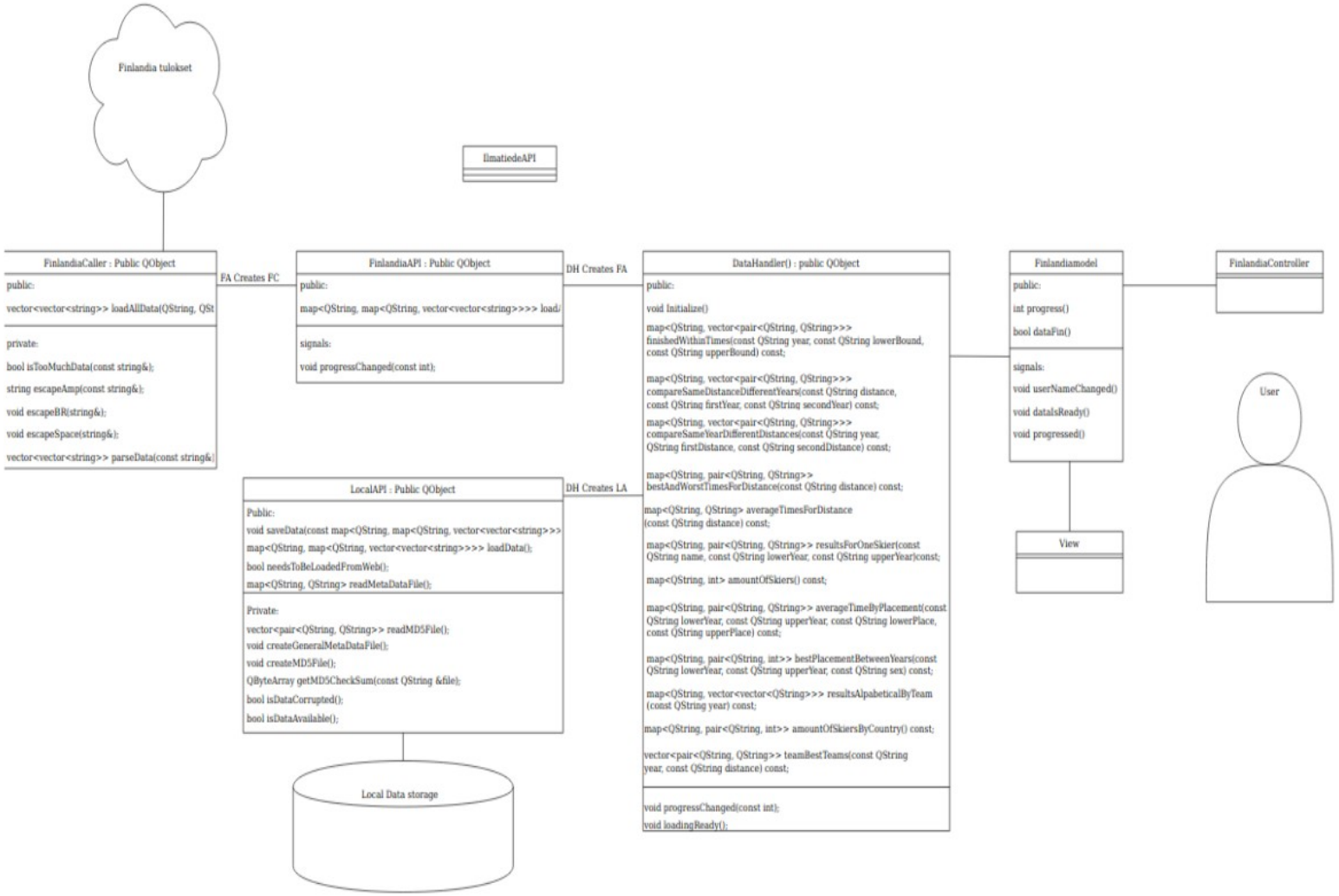
Jos lähtisimme samoista lähtökuopista nykyisen tiedon valossa, jakaisimme työtä Gantt-kaaviossa tasaisemmin riveittäin, sillä työmäärällisesti backEnd -alitiimi on joutunut tekemään huomattavasti paljon enemmän töitä ajallisesti välipalautukseen mennessä.

Lähteet

Lähdeviitteitä ei ole.

Liitteet

Liite 1: Alustava luokkajakokaavio



Liite 1: Alustava kaavio