Noriin Dorato – Joonas Jouttijärvi – Tuuli Kivisaari – Irina Romjalis

Raportti - Sprint 5: Sovelluksen lokalisaatio

OTP2

Metropolia Ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotuotantoprojekti 2
TX00CF84-3018
13.11.2023

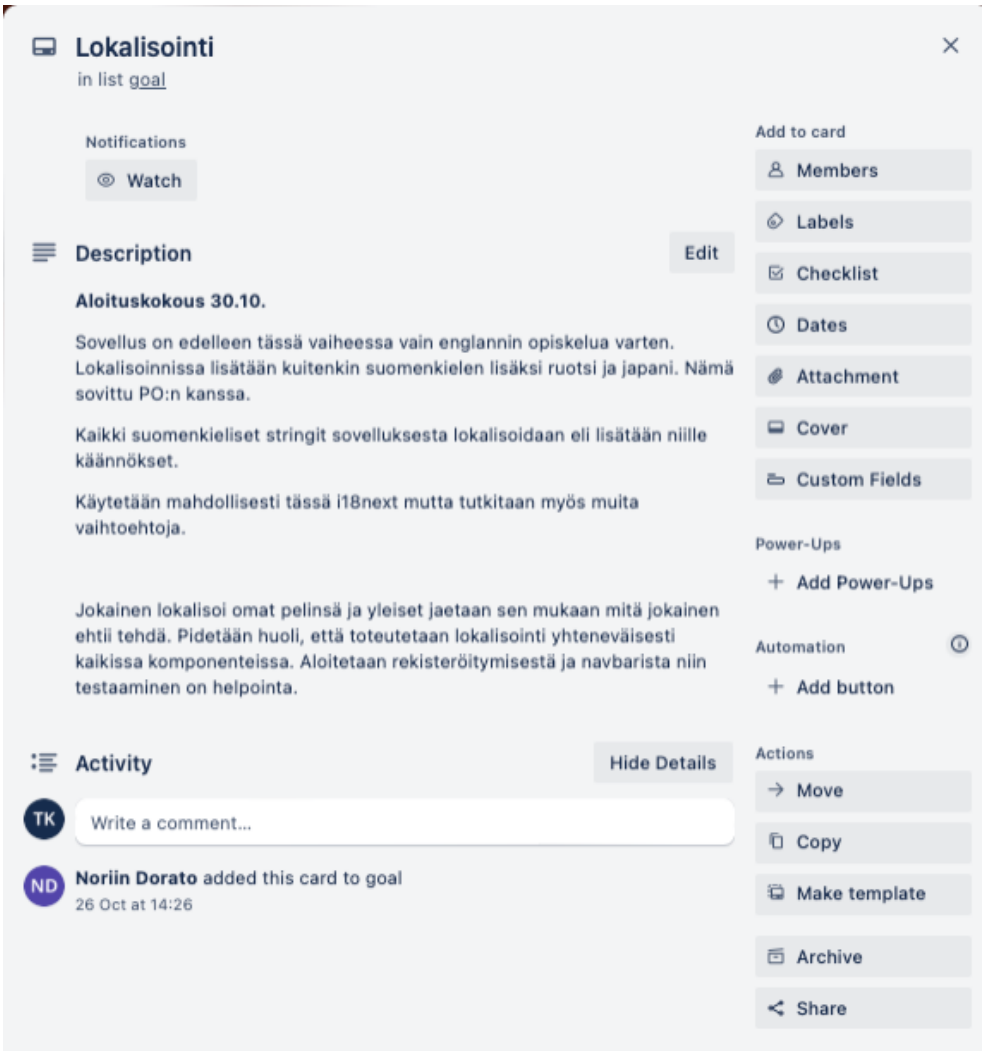
Johdanto

Sprintin 5 tavoitteena Ohjelmistotuotanto 2 -kurssilla oli lokalisoida kurssilla rakennettu sovellus. Lokalisoinnilla tarkoitetaan prosessia, jossa sovellus mukautetaan eri kielille, alueille ja kulttuureille, jotta se olisi halutun kielisille käyttäjille ymmärrettävä ja käyttökelpoinen. Lokalisointi voi sisältää tekstien kääntämistä, valuutan vaihtamista, päivämäärämuotojen sovittamista ja muiden paikallisten erojen huomioon ottamista, mutta LanguageApp-sovelluksen kohdalla lokalisoitiin vain tekstit, koska sovelluksessa ei käytetä esimerkiksi päivämääriä tai valuuttoja. Tässä raportissa kerrotaan siitä mitä ja miten LanguageApp-sovellusta lokalisoitiin. Raportin loppupuolella esitellään myös huomioita tietokannan lokalisoinnista ja siitä, miksi sitä ei ollut tarpeen tehdä LanguageApp-sovelluksen kohdalla. Lopussa on myös sprintin yhteenveto, sekä liitteenä muutamia kuvia lokalisoidusta sovelluksesta.

Sprintin 5 tavoite ja menetelmät

LanguageApp-sovellus on tarkoitettu englanninkielen opiskeluun hauskalla ja tehokkaalla tavalla. Suunnittelimme sovelluksen alunperin nimenomaan suomenkielisiä lapsia varten. Tässä sprintissä kuitenkin tavoitteenamme oli laajentaa sovellusta niin, että se olisi soveltuva myös ruotsin- ja japaninkielisille käyttäjille.

Asioista sovittiin sprintin aloituskokouksessa 30.10.2023:



Kuva 1: Sprintin aloituskokous Trellossa

Lokalisaation vaiheet

Tässä osiossa kerromme lokalisaation vaiheista ja siihen valituista menetelmistä.

Työkalujen valinta

Aloitimme lokalisaation työstämisen tutkimalla erilaisia vaihtoehtoja lokalisaation toteuttamiseksi. Tarjolla on lukuisia eri teknologioita, joiden avulla kansainvälistämisen toteuttaminen on kätevää. Tutustuimme lähinnä Format.js:ään, joka on yksi käytetyimmistä työkaluista tähän tarkoitukseen, sekä i18nextiin, joka tarjoaa myös erittäin monipuoliset ja yksinkertaiset tavat uusien kielien implementoimiseen. Näistä kahdesta valitsimme i18nextin, koska sen käyttäminen vaikutti selkeämmältä, joskin lukemamme perusteella Format.js olisi ollut hieman kevyempi vaihtoehto.

118next-kirjasto tekee käännösten hallinnan erittäin helpoksi. I18n on työkalu, joka helpottaa monikielisen tuen integroimista verkkosovelluksiin. Se tarjoaa yksinkertaisen tavan kääntää ja hallita tekstejä eri kielille, toimien joustavasti eri kehyskirjastojen kanssa. I18n on modulaarinen ja voi ladata käännökset erilaisista lähteistä, kuten JSON-tiedostoista tai tietokannoista. Se tukee myös nimialueisiin (englanniksi namespace) perustuvaa järjestämistä ja mahdollistaa asynkronisen käännösten lataamisen, mikä on hyödyllistä isommissa sovelluksissa. Namespacet auttavat organisoimaan käännöksiä selkeämmin ja vähentävät mahdollisia konflikteja.Tässä sovelluksessa käytössä oli nimenomaan i18next, joka on i18n:n yksi konkereettinen kirjasto ja tarjoaa nimenomaan i18n-toiminnallisuutta.

Kun olimme valinneet työkalut, päätimme jatkaa prosessia niin, että jaamme komponentteja ja ryhmäläisten tehtävänä on tunnistaa niistä kaikki suomenkieliset tekstit. Englanninkielisiä tekstejä ei ollut tarpeen muuttaa, koska sovelluksen tarkoituksena olisi edelleen englannin opiskelu.

Konfiguraatio

Heti alussa luotiin myös i18n.js konfiguraatiotiedosto, jonka tarkoituksena oli määritellä konfiguraatio kansainvälisyyskäännösten (i18n) hallintaan sovelluksessamme. Tämä tiedosto määrittää asetukset, kuten tuettavat kielet, varakäännökset ("fallbackLng"), tietyn käännösresurssin nimiavaruuden (namespace), ja paljon muuta. Tuettavat kielet LanguageAppissa ovat siis

suomi, ruotsi ja japani, eli "fi_FI", "ja_JP" ja "sv_SE" ja englanninkielisten osioiden kohdalla “en_GB”.

```
import ...

i18n.use(HttpApi) // use http backend
    .init( options: {
        // don't define resources here
        fallbackLng: "fi_FI",
        debug: true,
        supportedLngs: ["en_GB", "fi_FI", "ja_JP", "sv_SE"],

        // have a common namespace used around the full app
        ns: ["common"],
        defaultNS: "common",

        keySeparator: false, // we use content as keys

        interpolation: {
            escapeValue: false, // not needed for react as it escapes by default
        },

        react: {
            useSuspense: true,
        },

        backend: {
            loadPath: function(lngs, namespaces) {
                return `/locales/${lngs[0]}/${namespaces[0]}.json`;
            },
        },
    });

2 usages  tuulikoo
export default i18n;
```

Kuva 2: i18n.js-tiedosto

Kielivalikko

Ennen varsinaisen käännösten tekemisen aloittamista, oli myös tarpeen luoda navigointipalkkiin valikko, mistä kieltä voi vaihtaa, jotta kukin ryhmäläinen pystyisi heti testaamaan käännösten toimintaa. Kielivalikko tehtiin niin, että näkyvissä on aina sen maan lippu, minkä maan kieli on käytössä. Lippua painamalla avautuu dropdown, jossa muut kielivaihtoehdot ovat nähtävillä, ja niistä painamalla koko sivuston kielen on tarkoitus vaihtua.

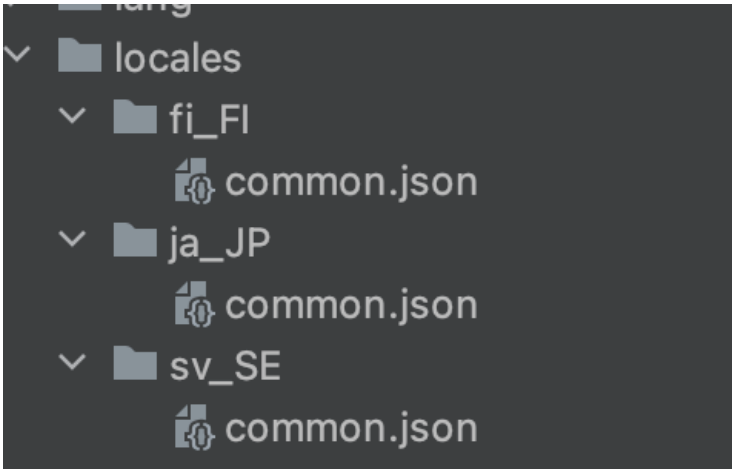
Kuva 3: Kielivalikko

Halusimme, että kieli on vaihdettavissa sekä kirjautuneelle käyttäjälle, että kirjautumattomalle käyttäjälle. Ratkaisimme asian niin, että jos käyttäjä on

kirjautunut sovellukseen, kielivalinta tallentuu aina tietokantaan. Jos taas käyttäjä ei ole kirjautunut, kielivalinta tallennetaan evästeeseen (cookie).

Käännösten tekeminen

Kun suomenkielisiä tekstejä muutettiin muiden kielten käännöksiksi, alkuperäiset tekstit korvattiin i18nextin t-funktioilla. Tämä tarkoitti, että alkuperäisten tekstien tilalle asetettiin aina tietty nimiavaruus (namespace), ja itse käännökset sijoitettiin erillisiin tiedostoihin kansioon public/locales/”kielikoodi” kuten alla näkyvässä kuvassa:



Kuva 4: Käännöstiedostot kansioissa

Tiedostojen sijainnit kielikoodin mukaan oli konfiguroitu i18n.js tiedostoon. i18nextin t-funktio on käännettävän tekstin (translation) lyhenne, joka ottaa huomioon käännösten hakemisen ja käännöksissä käytetyn nimiavaruuden.

Käännöstiedostojen sisällöksi siis laitettiin namespace, joilla teksti korvattiin kussakin komponentissa, sekä siihen haluttu käännös. Esimerkkejä alla näkyvissä kuvissa:

```
{
    "startStudying": "Aloita opiskelu",
    "trainingMessage": "Olet opiskellut {{totalTime}} minuuttia",
    "greatJobMessage": "Hienoa! Olet opiskellut {{totalTime}} minuuttia",
    "keepGoingMessage": "Jatka samaan malliin! Olet opiskellut {{totalTime}} minuuttia",
    "UPwelcome": "Tervetuloa omalle sivullesi {{username}}!",
    "UPuserPoints": "Pisteesi: {{userPoints}} ",
```

Kuva 5: Suomenkielisiä käännöksiä tiedostossa

```
{
    "startStudying": "勉強を始める",
    "trainingMessage": "あなたは {{totalTime}} 分勉強しました",
    "greatJobMessage": "素晴らしい! あなたは {{totalTime}} 分勉強しました",
    "keepGoingMessage": "この調子で続けてください! あなたは {{totalTime}} 分勉強しました",
    "UPwelcome": "\"{{username}}さん、あなたのページへようこそ!",
    "UPuserPoints": "ポイント: {{userPoints}}",
```

Kuva 6: Japaninkielisiä käännöksiä tiedostossa

```
{
    "startStudying": "Börja studera",
    "trainingMessage": "Du har studerat {{totalTime}} minuter",
    "greatJobMessage": "Bra jobbat! Du har studerat {{totalTime}} minuter",
    "keepGoingMessage": "Fortsätt så! Du har studerat {{totalTime}} minuter",
    "UPwelcome": "Välkommen till din hemsida {{username}}!",
    "UPuserPoints": "Dina poäng: {{userPoints}} ",
```

Kuva 7: Ruotsinkielisiä käännöksiä tiedostossa

Käännösten lisääminen tiedostoihin oli hyvin yksinkertaista. Apuna käännösten tekemisessä käytimme DeepL Translate-sivustoa: https://www.deepl.com/en/translator.

Tietokannan lokalisointi LanguageApp

Tässä osiossa kerrotaan LanguageAppin tietokantarakenteesta lokalisoinnin suhteen. Lopuksi on mietitty myös sitä, mikä olisi voinut olla vaihtoehtoinen toteutustapa tietokannalle.

Nykyinen rakenne

LanguageApp-sovelluksen lokalisointi onnistuu ilman muutoksia tietokantaan. Koska sovellus käyttää MongoDB-tietokantaa ja Prisma-Orm työkalua, ratkaisu on hyvin dynaaminen.
Alla olevassa kuvassa on sovelluksen tietokanta. Kielestä on tehty oma muuttujansa tietokantaan, jolloin sen käyttö on hyvin yksinkertaista.

```
generator client {
 provider = "prisma-client-js"
}
```

```
datasource db {
 provider = "mongodb"
 url      = env("DATABASE_URL")
}
enum Language {
 fi_FI
 sv_SE
 ja_JP
}

model User {
 id         String @id @default(auto()) @map("_id")
@db.ObjectId
 firstName  String
 username   String @unique
 email      String @unique
 password   String
 avatarId   Int?
 userPoints Int    @default(0)
 lastLevel  Int    @default(1)
 userRole   String @default("user")
 timeSpent  Int?   @default(0)
 language   Language @default(fi_FI)
}


model Game4 {
 id        String @id @default(auto()) @map("_id")
@db.ObjectId
 category  String
 questions Json[]
}

model ListeningData {
 id       String   @id @default(auto()) @map("_id")
@db.ObjectId
 category String   @unique // Unique index on the 'category'
field
 words    String[]
}

model SchoolItem {
 id       Int    @id @map("_id")
 imageUrl String
 eng      String
 fin      String
}
```

```
model FruitItem {
 id       Int    @id @map("_id")
 imageUrl String
 eng      String
 fin      String
}

model AnimalItem {
 id       Int    @id @map("_id")
 imageUrl String
 eng      String
 fin      String
}

model FlashcardItem {
 id         Int    @id @map("_id")
 word       String
 definition String
}

model Flashcardsanat {
 id         String @id @default(auto()) @map("_id")
@db.ObjectId
 flashcards Json
}
```

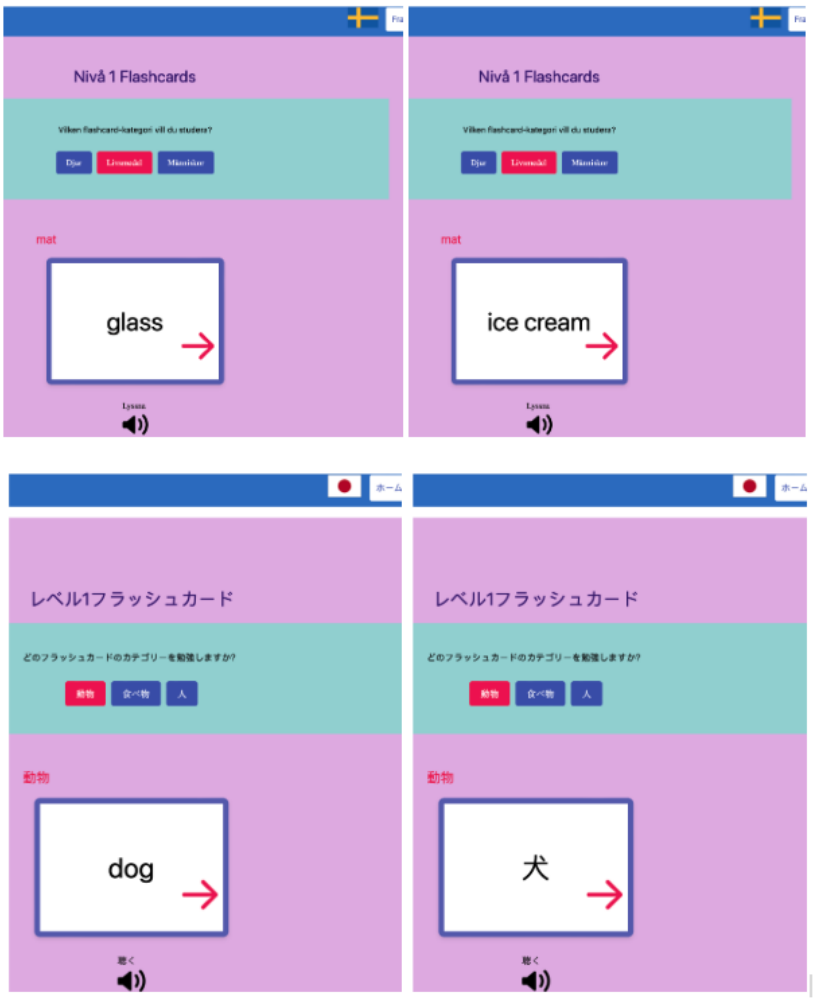
Kuva 8: Tietokannan rakenne

Oleellisin tietokantataulu on model User. Tauluun tallennetaan käyttäjädata, joka ei ole kielisidonnaista. Jos käyttäjä haluaa kirjoittaa itselleen japaninkielisen nimen, hän voi tehdä niin, ja nimi on silti sama kaikilla valittavissa olevilla kielillä. User-modeliin tallennetaan myös käyttäjän kielivalinta. Tallennettavissa tiedoissa ei ole päivämääriä, eikä numeroita, joiden formaatteja pitäisi voida muokata.

Jos tutkaillaan loppuja moduleja, mitä tietokannasta löytyy, niin ne eivät myöskään varsinaisesti vaadi muutoksia tietokantarakenteeseen. Lokalisointiin liittyvien tekstitietojen päivitys voi olla helpompaa, kun käytetään dynaamista rakennetta, kuten JSON. Esimerkiksi uuden kieliversion lisääminen ei välttämättä vaadi tietokantarakenteen muuttamista lainkaan. Tehtävien materiaalit on tallennettu tietokantaan json-muodossa, ja kielivaihtoehdot huomioidaan JSONin sisällä. Alla olevissa esimerkeissä näkyy kuvat flashcards-harjoituksesta, kun valittuna kielenä on japani tai ruotsi.

Kuva 9: Flashcard-harjoitus ruotsiksi

Kuva 10: Flashcard-harjoitus japaniksi



Kuva 11: Flashcard-harjoitusta ruotsiksi ja japaniksi

Tehtäväaineisto tulee json-tiedostosta:

```
{
  "flashcards": {
    "food": [
      {
        "word": "cake",
        "definitions": {
          "fi_FI": "kakku",
          "sv_SE": "tårta",
          "ja_JP": "ケーキ"
        }
      },
      {
        "word": "food",
        "definitions": {
          "fi_FI": "ruoka",
          "sv_SE": "mat",
          "ja_JP": "食べ物"
        }
      },
```

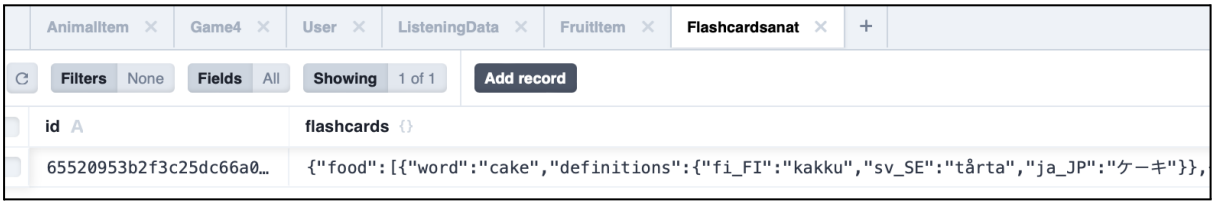
Kuva 12: Flashcard-aineisto json-tiedostossa

Json-tiedosto on kokonaisuudessaan lisätty tietokantaan näin:

```
model Flashcardsanat {
 id          String @id @default(auto()) @map("_id") @db.ObjectId
 flashcards Json
}
```

Kuva 13: Flashcard-aineisto tietokanta moduulissa

Ja näyttää tältä kun sitä tarkastellaan sisältöineen Prisma studion kautta:



Kuva 14: Flashcard-aineisto tietokantaan tallennettuna

Dynaaminen rakenne mahdollistaa siis sanastojen joustavan käytön myös eri kieliversioina.

LanguageAppista on lokalisoitu kaikki muut sivut paitsi admin-osio, koska se ei kuitenkaan ole loppukäyttäjälle ulospäin näkyvä osio. Admin-osiota ei lokalisoida.

Vaihtoehtoinen tietokantasuunnitelma lokalisointia varten

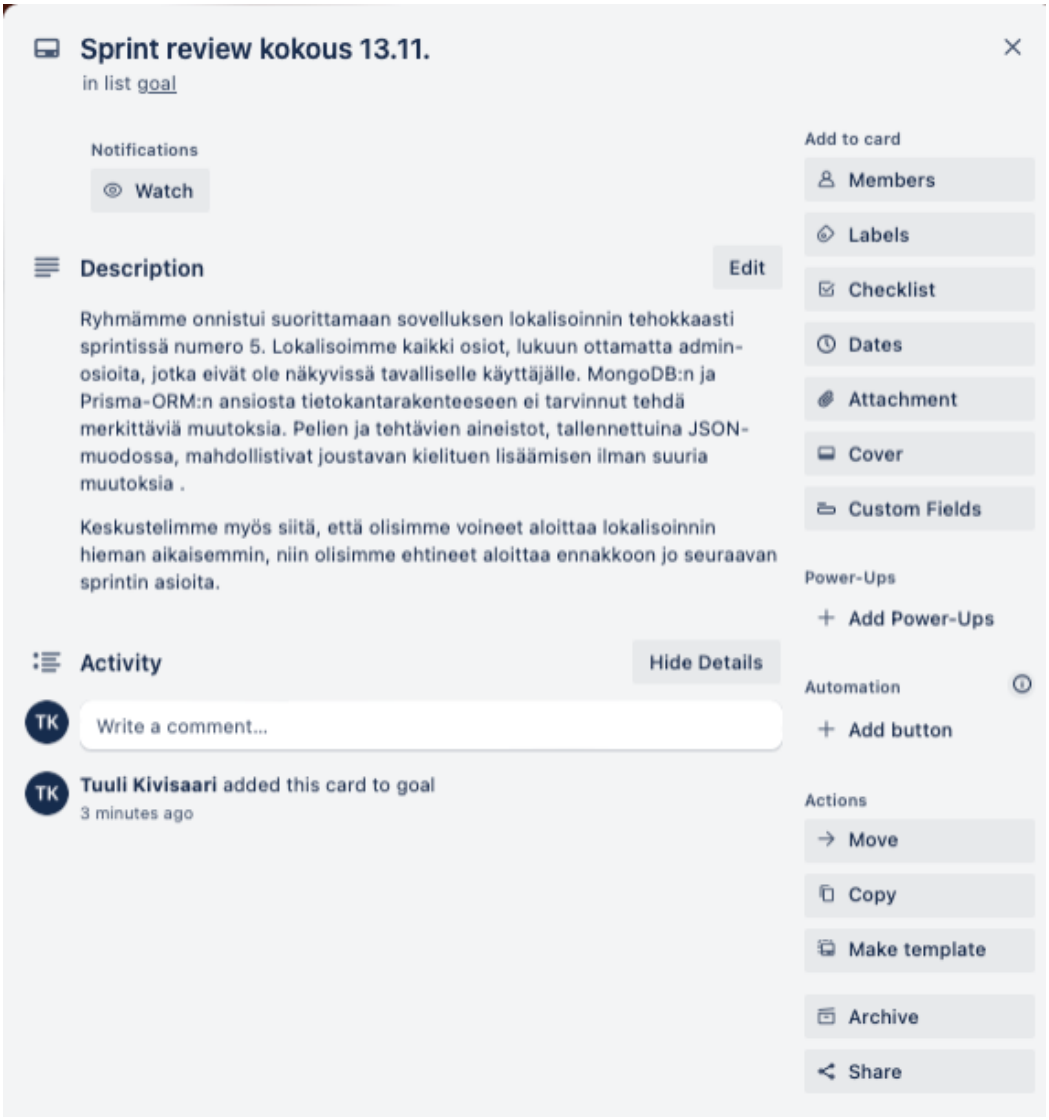
Jos sovellus olisi alun pitäen kehitetty niin, että lokalisointi olisi ollut tiedossa, olisi voinut olla kannattavaa, tai ainakin mahdollista, tehdä se tavallisena SQL-tietokantana. SQL-tietokannan rakenne on strukturoidumpi kuin no-sql tietokannan. Strukturoidumpi rakenne voi olla tehokkaampi tietyissä tapauksissa ja mahdollistaa selkeämmän hallinnan monikielisen sisällön suhteen. Kieliversiot olisi voitu alusta alkaen rakentaa esimerkiksi rivittäisellä lähestymistavalla, missä eri kielet ovat eri riveillä niin tehtäväaineistojen kuin muunkin tekstin osalta, eikä json-muotoista dataa olisi tehtävienkään osalta tarvittu. Esimerkiksi question_id, language_code ja text olisivat rivittäisessä lähestymistavassa voineet näyttää seuraavanlaisilta:

1, fi_FI, “Valitse oikea vaihtoehto”
1, ja_JP, “正しい選択肢を選ぶ”
1, sv_SE, “Välj rätt option”
2, fi_FI jne.

LanguageAppin nykyinen tietokantarakenne mahdollistaa siis tehokkaan lokalisoinnin ilman muutoksia tietokantaan. Käyttöön otettu dynaaminen JSON-rakenne mahdollistaa joustavan kielituen, kun taas i18next-kirjasto helpottaa käännösten hallintaa. Vaihtoehtoisesti SQL-tietokanta olisi tarjonnut strukturoidumman lähestymistavan, mutta nykyinen ratkaisu tukee tehokkaasti monikielistä sisältöä.

Yhteenveto

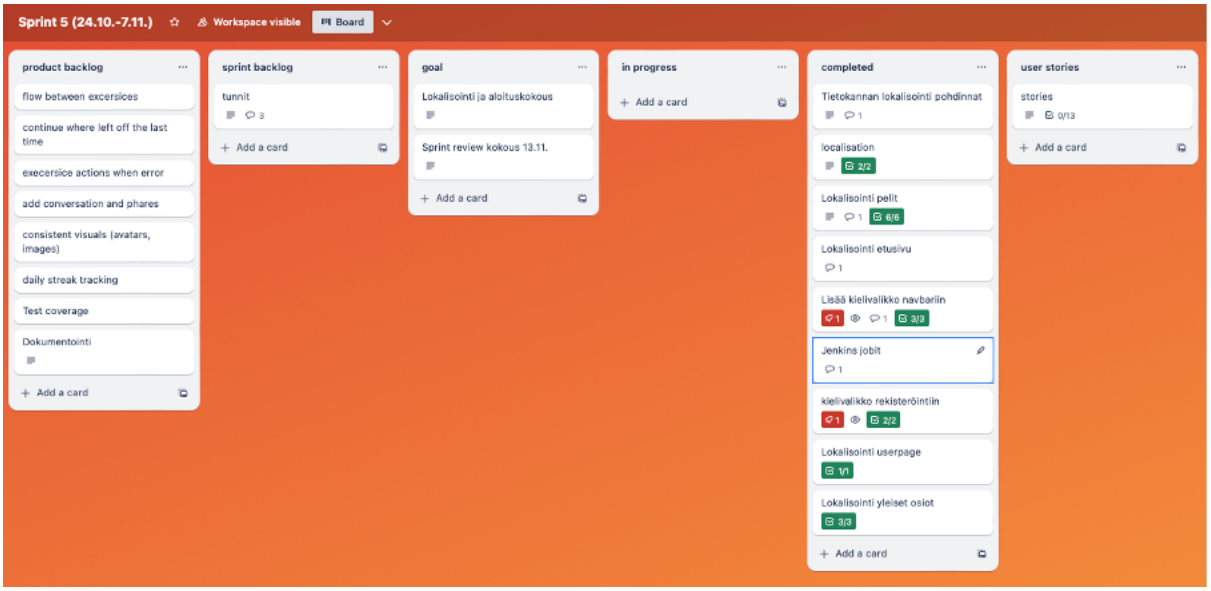
Lokalisointi sujui todella hyvin, ja oli ehkä jopa helpompaa kuin mitä olimme etukäteen ajatelleet. Joitakin ongelmiakin kuitenkin tuli, sillä huomasimme, että joissakin tapauksissa kieli päivittyy viiveellä, kun ohjelma ei pystynyt löytämään käännöksiä aina tarpeeksi nopeasti. Myös cookien kanssa vaikutti olevan jotakin viivettä. Emme ehtineet korjata noita ongelmia sprintin aikana, joten jätämme ne seuraavaan tai sitä seuraavaan sprinttiin. Alla olevissa kuvissa on nähtävissä Trelloon kirjatun sprint review -kokouksen sisältö, kuva tunneista sekä sprint 5:n backlogia:



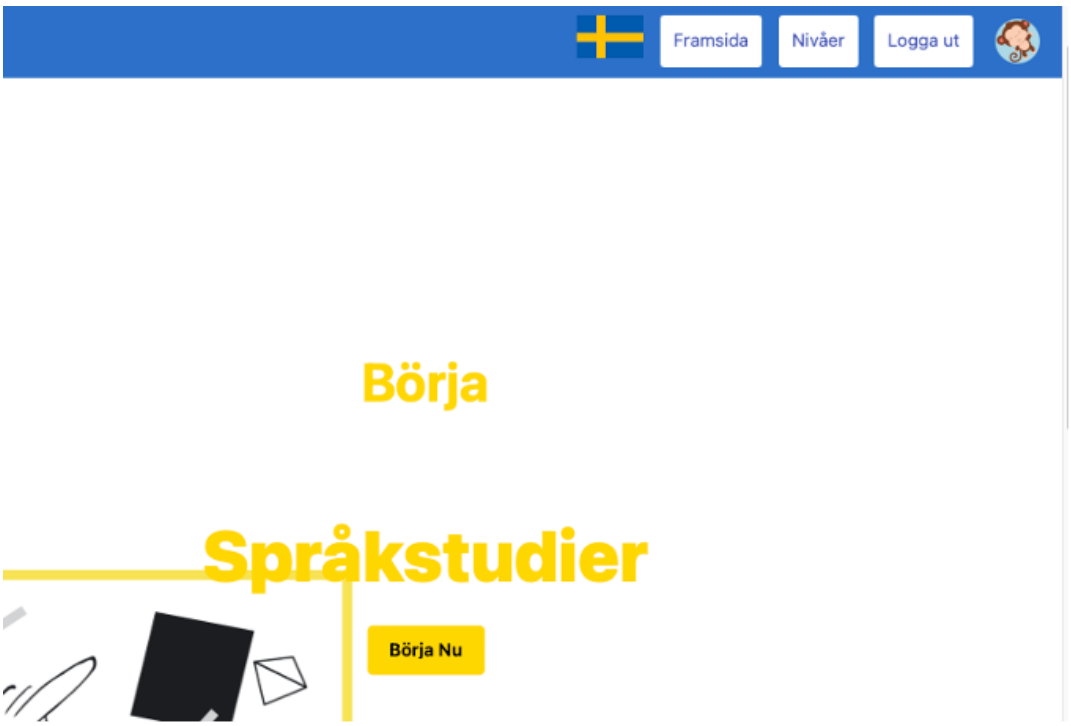
Kuva 15: Lopetuskokous Trellossa

tunnit	Noriin	Tuuli	Irina	Joonas
sprint 5				
viikko1				2h
viikko2	5h	16h	10h	11h
viikko3	6h	12h	13h	12h

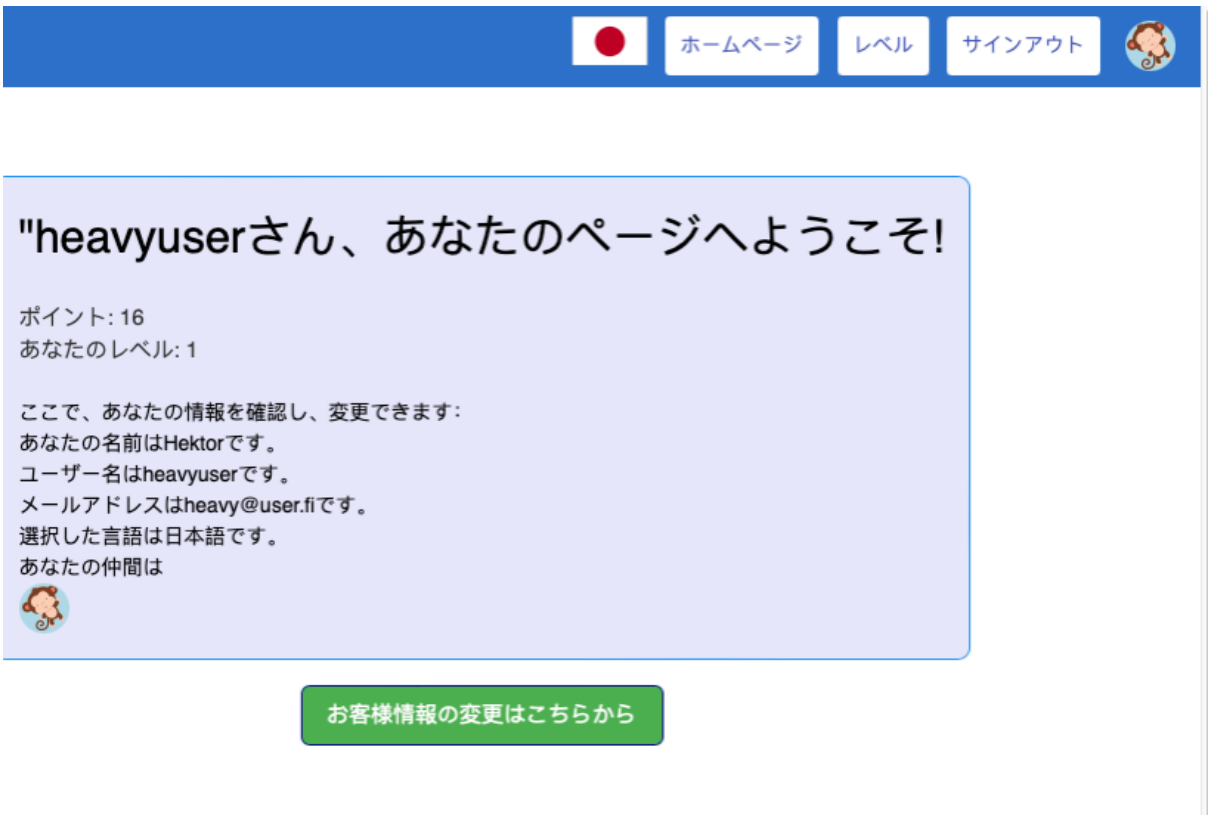
Kuva 16: Ryhmäläisten tunnit sprintissä



Kuva 17: Backlog



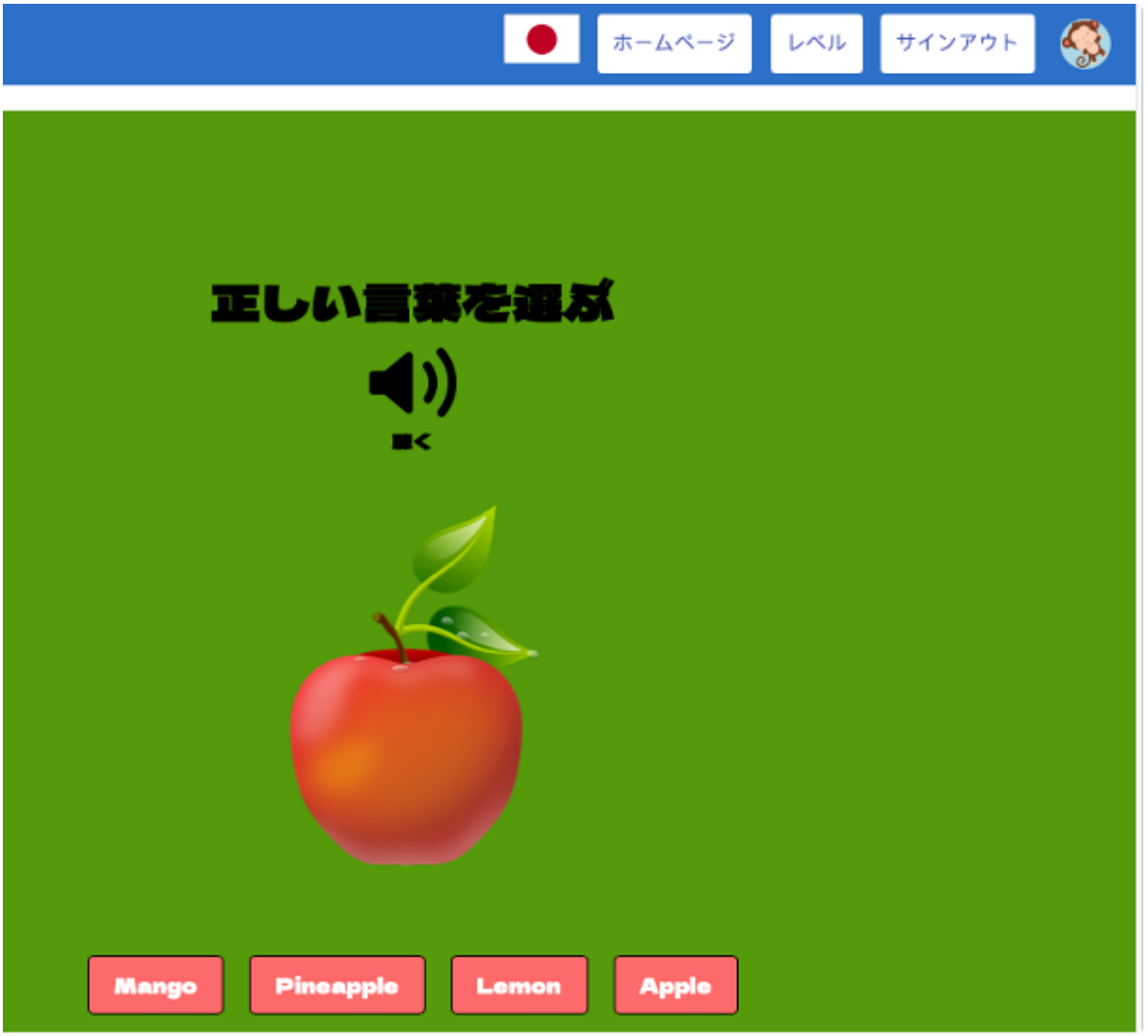
Kuva 1: Etusivu ruotsiksi



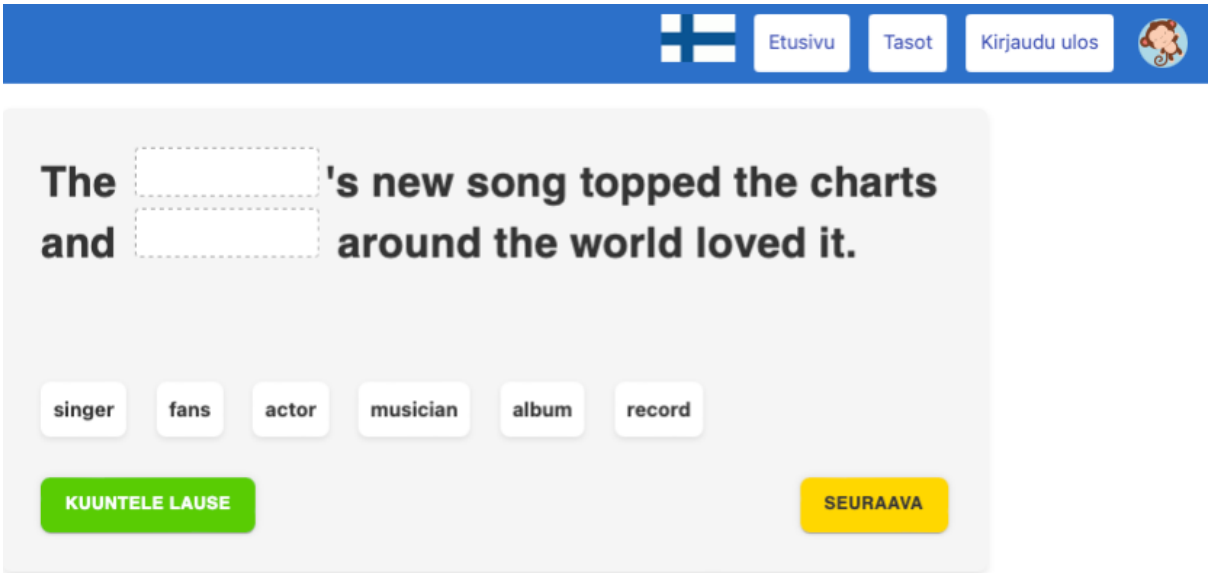
Kuva 2: UserPage japaniksi

Liite 1: Kuvia lokalisoidusta sovelluksesta

Kuva 3: Valintaikkuna japaniksi



Kuva 4: Game 4 japaniksi



Kuva 5: Game 5 suomeksi