jamk

Käännöstyökalun kehittäminen

Antti Ikonen

Opinnäytetyö, AMK
Huhtikuu 2025
Tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Ikonen, Antti

Käännöstyökalun kehittäminen

Jyväskylä: Jyväskylän ammattikorkeakoulu. **Huhtikuu, 2025**, 46 sivua.

Tieto- ja viestintätekniikka. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Kehittämistyön tavoitteena oli kehittää Movya Oy:lle verkkosovellus käännösten hallintaan asiakasprojekteja varten. Projektin tavoitteena oli mahdollistaa asiakkaille käännöksien ja sisältöjen muokkaamisen itsenäisesti tarkoitettu verkkosovellus. Tutkimus toteutettiin kehittämistutkimuksena, jossa laadittiin käännöstyökalu verkkosovelluksena.

Käännöstyökalun avulla asiakkaat voivat hallita kieliversioita, kieliavaimia, sekä sisältöä itsenäisesti ilman kehittäjän apua. Palvelu toteutettiin Next.js-kirjastolla ja AWS Amplify -palvelua hyödyntäen. AWS Amplify tarjosi kehitykseen hyviä osia, esimerkiksi backend rajapinta ja automaattinen pipeline ilman suurempaa konfiguraatiota. Projektin edetessä kuitenkin tuli huomattua AWS Amplify:n rajoittuneisuus, jos tarvitsee tehdä suurempia konfiguraatioita ja se sopii pikemminkin nopeaan prototyypitykseen tai pieniin projekteihin.

Kehitysprosessia vietiin eteenpäin käytännönläheisesti, yhdistäen teoriaa ja käytäntöä. Käyttöliittymästä tehtiin selkeä ja helppokäyttöinen, jotta asiakkaat voivat tehdä tarvittavat muokkaukset ilman syvällistä teknistä osaamista. Sovelluksen tietokantamalli suunniteltiin tukemaan monikielistä sisällönhallintaa skaalautuvalla rakenteella.

Kehittämistyön lopputuloksena syntyi toimiva ja selkeästi jäsennelty verkkosovellus, jonka avulla voidaan lisätä eri kieliversioita, käännöksiä ja sisältöjä. Sovellus tukee myös omien JSON-käännöstiedostojen tuontia ja vientiä. Verkkosovellus otetaan heti käyttöön asiakasprojekteissa ja jatkokehityksen kohteena on oman backend:n kehittäminen, sekä siirtäminen projekti monorepoon.

Avainsanat (asiasanat)

Ohjelmointi, web-kehitys, React.js, Next.js, TypeScript, AWS Amplify

Muut tiedot (salassa pidettävät liitteet)

-

jamk

Description

Ikonen, Antti

Developing a web-based translation tool

Jyväskylä: JAMK University of Applied Sciences, April 2025, 46 Pages

Degree Programme in Information and Communications Technology. Bachelor´s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

The objective of this development project was to create a web application for Movya Oy to manage translations in customer projects. The aim was to build a solution that allows clients to edit translations and content independently, without developer assistance. The project was carried out as a development project, resulting in a translations editing system integrated into the web application.

With the translation tool, clients can manage language versions, translation keys, and content autonomously. The service was implemented using the Next.js framework and AWS Amplify. Amplify offered useful tools for the development process, such as a backend API and an automatic deployment pipeline with minimal configuration. However, during the project it became evident that AWS Amplify has certain limitations when more advanced configurations are needed, making it more suitable for rapid prototyping or small-scale projects.

The development process followed a practical approach, combining theory with implementation. The user interface was designed to be easy to use, allowing clients to make necessary modifications without deep technical knowledge. The database model was designed to support multilingual content management with a scalable structure.

As a result, a functional and well-structured web application was created, enabling the addition of language versions, translations, and content. The application also supports importing and exporting custom translation files in JSON format. The system will be taken into immediate use in customer projects, and future development will focus on developing a custom backend and moving the project to a monorepo structure.

Keywords/tags (subjects)

programming, web development, React.js, Next.js, TypeScript, AWS Amplify

Miscellaneous (Confidential information)

-

Sisältö

Kuviot

Taulukot

1 Johdanto

1.1 Kehittämistyön tausta ja rakenne

Yksi yrityksen haasteista asiakasprojekteissa on ollut käännöstiedostojen hallinta. Tällä hetkellä käännöstiedot ja sisällöt sijaitsevat yleensä Excel-tiedostoissa, joita lähetellään sähköpostitse edestakaisin. Yksi merkittävimmistä haasteista on tarjota joustava ja tehokas tapa sisällön muokkaamiseen ilman, että siihen tarvitaan kehittäjää.

Tässä kehittämistyössä suunnitellaan ja toteutetaan sisällönmuokkausjärjestelmä verkkosovelluksille ja asiakasprojekteille hyödyntäen moderneja web-teknologioita. Järjestelmä toteutetaan erillisenä palveluna, jota voidaan helposti tarjota eri asiakkaille. Palvelun avulla asiakkaat voivat hallita sovellustensa sisältöä itsenäisesti, esimerkiksi muokkaamalla tekstejä ja hallitsemalla kieliversioita ilman kehittäjän apua. Näin parannetaan joustavuutta ja tarjotaan asiakkaalle kokonaisvaltaisempi palvelukokemus kustannustehokkaasti.

Kehitystyön aikana keskitytään sisällönmuokkausjärjestelmän toteutukseen. Työssä käsitellään myös kehitystä monorepossa, koska alun perin projektin piti olla osa monorepoa, mutta kehitys siirtyi toistaiseksi erillisenä palveluna. Painopiste on kevyessä ja helposti ylläpidettävässä arkkitehtuurissa, joka minimoi tarpeettomat riippuvuudet ja mukautuu eri asiakasprojektien vaatimuksiin.

Kehitystyön lopputuloksena syntyi käytännössä toimiva ja selkeästi jäsennelty käännöstyökalu, minkä avulla asiakas pystyy muokkaamaan, sekä lisäämään kieliversioita, käännöksiä ja sisältöjä. Jatkokehityskohteita ovat muun muassa oman backend-ratkaisun kehittäminen, sekä projektin siirtäminen monorepoon.

1.2 Movya oy

Movya Oy on vuonna 2009 perustettu digitaalisen median yritys, joka tarjoaa asiakkailleen elämyksiä, tarinoita ja sovelluksia, joilla on todellista merkitystä. Yrityksen palveluihin kuuluvat digitaalinen mainonta, videotuotanto, 3D-tuotanto sekä verkkosovellusten kehitys. Movya Oy:ssä työskentelee noin 20 IT-alan osaajaa, ja yritys on toteuttanut yli 2000 onnistunutta projektia.

Yksi Movya Oy:n merkittävimmistä innovaatioista on Storyverse™, edistyksellinen digitaalinen ratkaisu, joka muuntaa asiakas- ja työntekijäkokemukset virtuaalisten, immersiivisten ympäristöjen avulla. Storyverse™ mahdollistaa fotorealististen 3D-maailmojen rakentamisen, joihin pääsee käsiksi suoraan verkkoselaimen kautta. Tätä alustaa on hyödynnetty monipuolisesti eri asiakasprojekteissa, kuten tuotekonfiguraattoreina, koulutus- ja kokoustiloina sekä tuotejulkistuksissa. Lisätietoja Storyverse™-ratkaisusta löytyy osoitteesta www.storyverse.fi.

2 Kehittämistyö

2.1 Tarkoitus, tavoitteet ja tutkimuskysymykset

Tämän kehittämistyön tarkoituksena on kehittää yritykselle modulaarinen sisällönmuokkausjärjestelmä palveluna, joka mahdollistaa asiakasprojekteissa toimitettujen verkkosovellusten sisällön muokkaamisen ilman erillistä kehittäjää. Järjestelmä toteutetaan Next.js-pohjaisena palveluna, joka voidaan integroida joustavasti eri asiakasympäristöihin ja sovelluksiin.

Kehittämistyön tavoitteena on luoda tehokas ja käyttäjäystävällinen työkalu, joka nopeuttaa asiakasprojektien toimitusta ja parantaa asiakkaiden mahdollisuuksia tehdä pieniä muutoksia itse. Tämä lisää projektien joustavuutta, parantaa asiakastyytyväisyyttä ja tehostaa kehitystiimin työskentelyä.

Työn tutkimuskysymykset keskittyvät siihen, miten sisällönmuokkausjärjestelmä voidaan toteuttaa modulaarisesti ja kevyesti, minimoiden ulkoiset riippuvuudet. Lisäksi selvitetään, millaisia ominaisuuksia nykyiset sisällönmuokkausjärjestelmät tarjoavat ja miten niiden parhaita käytäntöjä voidaan hyödyntää kehitystyössä.

Tutkimuskysymykset:

1. Millaisia ominaisuuksia olemassa olevilla sisällönmuokkausjärjestelmillä on, ja miten niiden parhaat käytännöt voidaan hyödyntää tässä kehitysprojektissa?
2. Kuinka modulaarinen ja kevyt sisällönmuokkausjärjestelmä voidaan toteuttaa Next.js-pohjaisena palveluna minimoiden ulkoiset riippuvuudet?
3. Miten kehitetty järjestelmä voidaan integroida helposti asiakasprojekteihin ja tarjota loppukäyttäjille intuitiivinen sisällönmuokkauskokemus?

Kehittämistyön onnistumisen kannalta on keskeistä tarkastella myös järjestelmän käytettävyyttä ja sen tarjoamia mahdollisuuksia loppukäyttäjille. Koska järjestelmä otetaan käyttöön osana asiakasprojekteja, sen täytyy olla paitsi teknisesti toimiva, myös helppokäyttöinen asiakkaan näkökulmasta. Tämän vuoksi käyttöliittymän suunnittelussa painotetaan selkeyttä, jotta asiakkaat voivat tehdä tarvittavat sisällönmuokkaukset ilman syvällistä teknistä osaamista.

Tutkimuskysymysten lisäksi työssä on tarkoitus testata, kuinka hyvin järjestelmä skaalautuu erikokoisiin projekteihin ja erilaisiin asiakastarpeisiin. Testauksen avulla pyritään varmistamaan järjestelmän käytännön toimivuus ja keräämään tietoa sen jatkokehitystä sekä mahdollisia laajennuksia varten.

Lisäksi työssä arvioidaan järjestelmän dokumentaation selkeyttä ja kattavuutta. Hyvin laadittu dokumentaatio tukee järjestelmän helppoa käyttöönottoa ja ylläpitoa, mikä lisää sen arvoa sekä yrityksen sisäisessä kehityksessä että asiakasprojekteissa.

2.2 Tutkimusmenetelmä

Tutkimuksellinen kehittämistyö yhdistää käytännön kehittämisen ja tutkimukselliset menetelmät, mikä tekee siitä keskeisen lähestymistavan työelämän ongelmien ratkaisemisessa. Toikko ja Rantanen (2009) tuovat esiin, että sen tavoitteena on sekä uuden tiedon tuottaminen että käytännön toiminnan kehittäminen ja organisaatioiden tarpeisiin vastaaminen. Keskeistä on systemaattinen tiedonkeruu ja analyysi, jossa hyödynnetään sekä laadullisia että määrällisiä tutkimusmenetelmiä. Näin varmistetaan, että kehittämistyön tulokset perustuvat paitsi käytännön kokemuksiin myös tutkittuun tietoon, mikä lisää niiden luotettavuutta ja sovellettavuutta.

Opinnäytetyössä tutkimuksellinen kehittämistyö tarjoaa hyvän tavan yhdistää teoria ja käytäntö tarkoituksenmukaisella tavalla. Toikko ja Rantanen (2009) korostavat erityisesti osallistavaa otetta, jossa työelämän toimijat ovat aktiivisesti mukana kehittämisprosessissa. Tämä tarkoittaa, että kehittämistyöhön osallistuvat tahot eivät ole vain tutkimuksen kohteita, vaan toimijoita, joiden näkemykset ja kokemukset ohjaavat prosessia. Kehittämistyö etenee vaiheittain ja kehittyy saadun tiedon perusteella, jolloin lopputulos vastaa aidosti käytännön tarpeisiin ja on helposti hyödynnettävissä työelämässä.

Tässä tutkimustyössä hyödynnetään tutkimuksellista kehittämistyötä, jossa kehitetään konkreettinen sovellus tai palvelu ja sovelletaan tutkimusmenetelmiä sekä aineiston analysointia. Työn lähtökohtana on työelämästä tunnistettu tarve ja siihen liittyvät kysymykset, jotka ohjaavat kehitysprosessia ja varmistavat, että lopputulos vastaa käytännön vaatimuksiin.

2.3 Aineiston keruu ja analyysi

Aineiston keruu toteutettiin kartoittamalla aluksi palvelun tarpeita. Alkukehitysvaiheessa työkalun tavoitteena on toimia palveluna, johon asiakkaalle annetaan pääsy. Palvelun kehitys perustuu moderneihin web-teknologioihin, ja sen toteutuksen aikana hyödynnetään kehitysprosessissa syntynyttä dokumentaatiota sekä käytettävyystutkimuksia.

Analyysivaiheessa arvioitiin palvelun toimivuutta ja käytettävyyttä kerätyn aineiston perusteella. Erityistä huomiota kiinnitettiin siihen, miten järjestelmä tukee asiakkaiden tarpeita ja kuinka sen rakenne mahdollistaa joustavan sisällönhallinnan. Näiden havaintojen pohjalta tehtiin tarvittavia parannuksia, jotta palvelu vastaisi paremmin käytännön vaatimuksia ja skaalautuisi eri asiakasprojekteihin.

2.4 Luotettavuus ja eettisyys

Kehittämistyön luotettavuus varmistettiin hyödyntämällä ajankohtaista aineistoa web-teknologioista sekä testaamalla palvelun toimivuutta eri kehitysvaiheissa. Tulosten paikkansapitävyys pyrittiin takaamaan dokumentoimalla kehitysprosessi huolellisesti ja analysoimalla palvelua eri käyttötarkoituksissa. Lisäksi käytetyt web-teknologiat valittiin vakauden ja laajan käytön perusteella, mikä osaltaan lisää ratkaisun luotettavuutta ja pitkäikäisyyttä.

Eettiset näkökohdat huomioitiin erityisesti käyttäjätietojen käsittelyssä ja palvelun saavutettavuudessa. Kehitystyössä noudatettiin tietosuojakäytäntöjä ja varmistettiin, että ratkaisu ei sisällä tarpeettomia käyttäjätietoja tallentavia ominaisuuksia. Lisäksi käyttöliittymä suunniteltiin siten, että se tarjoaa yhdenvertaisen käyttökokemuksen kaikille käyttäjäryhmille, mikä parantaa palvelun saavutettavuutta ja käytettävyyttä.

3 Web-sovelluskehityksen teknologiat

3.1 React.js

React on JavaScript-kirjasto, jota hyödynnetään erityisesti käyttöliittymien rakentamiseen. Se on suunniteltu palvelemaan yhden sivun sovelluksia (SPA), joissa korostuvat suorituskyky, nopea reagointi ja sujuva käyttäjäkokemus. Kirjaston kehittäjänä toimi Meta (entinen Facebook) ja se julkaistiin avoimena lähdekoodina vuonna 2013, kaksi vuotta alkuperäisen kehitystyön aloittamisen jälkeen. Vuosien mittaan React on noussut yhdeksi suosituimmista työkaluista verkkopohjaisten käyttöliittymien toteuttamisessa. (Hamori, 2024)

React perustuu komponenttipohjaiseen arkkitehtuuriin, jonka ansiosta käyttöliittymä voidaan jakaa pienempiin ja uudelleenkäytettäviin osiin. Tällä tavoin saadaan rakennettua monimutkaisista käyttöliittymistä helpommin hallittavia ja niiden laajentaminen on helpompaa. Komponenttipohjainen arkkitehtuuri on ollut yksi merkittävimmistä tekijöistä Reactin suosion kasvussa. Se tarjoaa selkeän rakenteen, joka tukee skaalautuvuutta ja helpottaa lähdekoodin uudellekäyttöä (*Hamori, 2024)*

React hallitsee käyttöliittymän muutoksia virtuaalisen DOMin avulla. Tämä tarkoittaa sitä, että kun sovelluksen tila muuttuu, React ei suoraan päivitä koko näkymää, vaan luo ensin muistissa kevyen version DOMista. Tätä virtuaalista rakennetta verrataan edelliseen tilaan, ja vain aidosti muuttuneet elementit päivitetään varsinaiseen DOMiin. Tämä voi vähentää renderöintikertoja ja nopeuttaa käyttöliittymän toimintaa. (Hamori, 2024)

3.2 Next.js

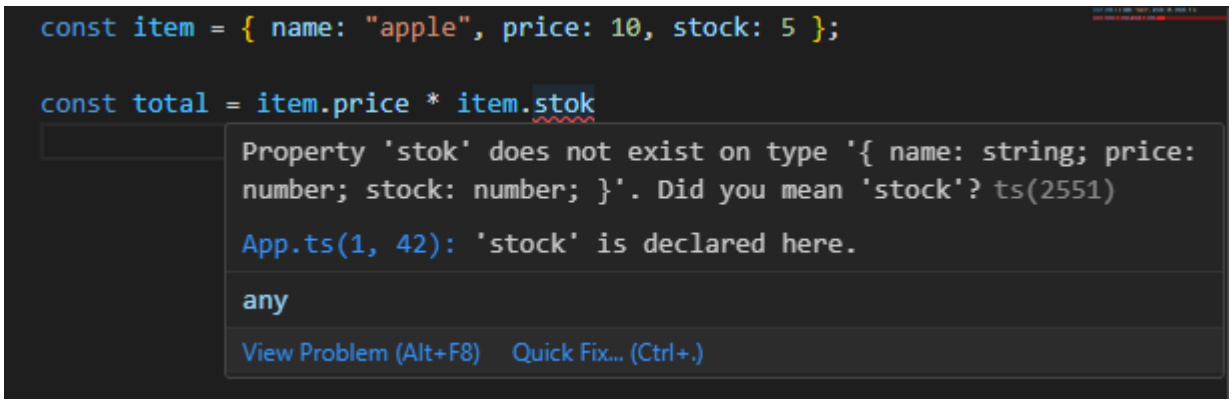
Next.js on suosittu avoimen lähdekoodin React-sovelluskehys, jonka kehittäjänä toimii Vercel. Se julkaistiin vuonna 2016, ja sen tavoitteena oli tehdä React-sovelluskehityksestä suoraviivaisempaa sekä tarjota valmiiksi sisäänrakennettuja ominaisuuksia, mitä React-kirjasto ei sisällä. Next.js sisältää esimerkiksi server-side renderingin (SSR), static site generationin (SSG) sekä hakukoneoptimointiin (SEO) liittyviä toimintoja, jotka parantavat verkkosovellusten suorituskykyä ja hakukonenäkyvyyttä (Server-Side Rendering, N.d.)

Next.js:n tiedostopohjainen reittien määrittäminen helpottaa sovellusten navigoinnin rakentamista. Uusia sivuja voidaan lisätä projektiin yksinkertaisesti luomalla tiedostoja oikeaan kohtaan kansiorakenteessa, ilman erillisiä asetuksia. Lisäksi Next.js tukee suoraan API-reittejä, mikä tarkoittaa, että palvelinpuolen toiminnallisuuksia voidaan rakentaa samaan projektiin ilman erillistä backendiä. (Server-Side Rendering, N.d.)

3.3 TypeScript web-kehityksessä

TypeScript on suosittu JavaScript kirjasto, sillä se tarjoaa staattisen tyypityksen, joka auttaa välttämään koodivirheitä jo kehitysvaiheessa. Microsoftin kehittämä TypeScript (2024) mahdollistaa virheiden tunnistamisen ennen ohjelman ajoa, mikä parantaa ohjelmistojen laatua ja vähentää virheiden aiheuttamia kehityskustannuksia. TypeScriptin dokumentaatio (n.d.) mainitsee myös, että kieli ei tarjoa ajonaikaisia kirjastoja, mutta se toimii saumattomasti JavaScriptin kanssa, mikä takaa sen yhteensopivuuden olemassa olevien ekosysteemien kanssa.

Monet ohjelmointikielet eivät salli ohjelmien käynnistymistä, jos lähdekoodi sisältää virheitä. TypeScriptin staattinen tarkastus tunnistaa virheet ennen ohjelman suoritusta, mikä estää potentiaaliset ongelmat jo koodausvaiheessa. Tyyppien tarkastus puolestaan varmistaa, että parametrit ja muuttujat vastaavat odotettuja tietotyyppejä. (TypeScript for the New Programmer, N.d.)



Kuvio 1. TypeScript Virhe.

TypeScript tunnistaa virheet ennen ohjelman suorittamista ja ilmoittaa välittömästi, jos esimerkiksi objektin ominaisuus on kirjoitettu väärin (ks. Kuvio 1).

TypeScriptin kääntäjä tarkistaa kaikki tyypit ja lopulta muuntaa lähdekoodin JavaScript-muotoon. Tämä tarkoittaa myös sitä, että TypeScriptin tyyppien tarkastus ei toimi ajon aikana, vaan kaikki tarkistukset suoritetaan ennen koodin ajamista. Lisäksi TypeScript ei tarjoa erillisiä ajonaikaisia kirjastoja, vaan se käyttää samoja kirjastoja kuin JavaScript. (TypeScript for the New Programmer, N.d.)

3.4 Monorepo

3.4.1 Monorepon soveltaminen laajojen ohjelmistokokonaisuuksien hallinnassa

Monorepo (monolithic repository) on versionhallintastrategia, jossa useita projekteja, kirjastoja ja paketteja hallitaan yhdestä keskitetystä repositoriosta. Tämä mahdollistaa yhteisen koodipohjan käytön ja helpottaa riippuvuuksien hallintaa, sillä kaikki osat kehittyvät samassa ympäristössä. Monorepo koostuu useista työtiloista (workspaces), joilla on omat testausympäristöt, koodin laadunvarmistusprosessit (linting) ja käännösprosessit (build). Suurissa projekteissa monorepoon voi kuulua tuhansia tehtäviä, mikä korostaa tehokkaan tehtävienhallinnan ja välimuistijärjestelmien, kuten Turborepon, merkitystä.

Monorepo-malli on erityisen suosittu suurissa ohjelmistokehitysprojekteissa, koska se edistää koodin uudelleenkäyttöä, parantaa yhteistyötä ja mahdollistaa versionhallinnan keskitetysti. Monet teknologiayritykset, kuten Google, Facebook ja Microsoft, hyödyntävät monorepo-ratkaisuja hallitakseen laajoja ja monimutkaisia koodikantojaan tehokkaasti.

Potvin ja Levenberg (2016) korostavat, että Googlen monorepo toimii kymmenien tuhansien kehittäjien kesken, mikä edistää yhtenäistä versiointia, laajaa koodin jakamista ja uudelleenkäyttöä sekä yksinkertaistaa riippuvuuksien hallintaa. Heidän mukaansa lähestymistapa mahdollistaa myös atomiset muutokset, laajamittaiset refaktoroinnit, sekä yhteistyön eri tiimien välillä.

3.4.2 Monorepon hyödyt ja haasteet

Monorepon käyttö ohjelmistokehityksessä tuo mukanaan monia etuja, erityisesti silloin kun halutaan keskittää koodi ja hyödyntää samoja toimintoja useissa projekteissa. Wieruch (2022) kuvaa, miten monorepojen avulla eri sovellukset voivat hyödyntää samoja kirjastoja ja työkaluja ilman,

että ne ovat erillisten versionhallintajärjestelmien takana. Kehitysympäristön johdonmukaisuus paranee ja vähentää päällekkäistä työtä.

Toisaalta CircleCI:n (n.d.) mukaan monorepojen hallinta voi aiheuttaa suorituskykyongelmia, erityisesti suurissa kehitysympäristöissä, joissa tiedostojen määrä kasvaa eksponentiaalisesti. Tämä on tärkeä huomio kehitettäessä npm-kirjastoa, jonka tavoitteena on tarjota modulaarinen ja helposti hallittava sisällönmuokkausratkaisu asiakasprojekteihin.

Monorepon hallinta voi muodostua vaikeaksi, jos projekti kasvaa erittäin suureksi, mikä voi aiheuttaa suorituskykyongelmia. Suuret monorepot voivat sisältää useita laajoja projekteja, mikä johtaa kehitysympäristön hidastumiseen. Esimerkiksi:

- Kääntäminen ja testien suorittaminen voi hidastua, koska muutokset jaettuihin komponentteihin voivat laukaista useiden sovellusten uudelleenkäännön ja testauksen.
- Suuri tiedostomäärä voi hidastaa versionhallintatyökalun, kuten Gitin, toimintaa.
- Jatkuvan integroinnin (CI) putket voivat ruuhkautua, kun muutokset aiheuttavat tarpeettomia rakennusprosesseja eri sovelluksille.
- Hallinnointi ilman oikeita työkaluja voi muuttua monimutkaiseksi ja hidastaa kehitystä merkittävästi. (Benefits and challenges of monorepo development practices, N.d.)

Näihin haasteisiin on kehitetty monorepon hallintatyökaluja, kuten Turborepo, Nx ja Lerna, jotka optimoivat riippuvuuksien hallintaa, tehtävien suorittamista ja välimuistimekanismeja. Ilman oikeanlaisia työkaluja monorepoista voi kuitenkin tulla nopeasti vaikeasti hallittavia, jolloin kehitystyö voi hidastua merkittävästi. (Benefits and challenges of monorepo development practices, N.d.)

3.4.3 Turborepo ratkaisuna monorepon hallintaan

Turborepo on työkalu, mikä helpottaa monorepojen hallintaa JavaScript-projekteissa. Turborepon kehittäjät (2024) esittävät, että se optimoi tehtävien suoritusta ja vähentää samojen komentojen ajamista useita kertoja eri paketeissa (esim. build). Turborepon avulla voidaan ratkaista monorepojen hallintaan liittyviä haasteita, joita CircleCI (n.d.) on aiemmin tuonut esiin.

Turborepo tukee myös älykästä välimuistia, jonka avulla se tallentaa aiemmin suoritetut tehtävät ja välttää tarpeettomia uudelleensuorituksia. Tämä on erityisen hyödyllistä suurissa projekteissa,

joissa käytetään CI/CD ympäristöä. Välimuistin ansiosta kehittäjät voivat keskittyä tekemiseen, ilman että heidän tarvitsee odottaa hitaita build-prosesseja tai säätää riippuvuuksien kanssa. Se on yhteensopiva npm-, pnpm- ja Yarn-pakettienhallintatyökalujen kanssa. Turborepon käyttöönotto on yksinkertaista ja sen konfigurointi tapahtuu turbo.json tiedostolla. (Turborepo Documentation, 2024)

3.5 AWS

Amazon Web Services (AWS) on yksi suosituimmista pilvipalvelualustoista. Se tarjoaa laajan valikoiman työkaluja ja palveluita, joiden avulla voidaan rakentaa skaalautuvia verkkosovelluksia ilman omaa palvelinympäristöä. AWS:n etuina ovat sen joustavuus, luotettavuus ja globaali kattavuus. *(Mikä on AWS-pilvipalvelu? N.D.)*

AWS:n tarjoaa erilaisia pilvipalveluja perustason ratkaisuista, kuten tietokannoista ja tiedostojen tallennuksesta, aina kehittyneisiin sovelluksiin, kuten koneoppimiseen, analytiikkaan ja serverless-arkkitehtuuriin. Tarjontaan kuuluvat muun muassa Amazon EC2, DynamoDB, RDS, Amazon S3 sekä AWS Lambda. Yksi AWS:n vahvuuksista on sen laaja ja globaali infrastruktuuri. Se mahdollistaa korkean saatavuuden ja hyvän suorituskyvyn eri puolilla maailmaa. AWS:n hinnoittelumalli perustuu käytön mukaan maksamiseen. *(Mikä on AWS-pilvipalvelu? N.D.)*

3.6 AWS Amplify

AWS Amplify on työkalu, joka nopeuttaa pilvipohjaisten web- ja mobiilisovellusten kehittämistä. Se tarjoaa valmiita ratkaisuja esimerkiksi käyttäjien tunnistautumiseen, tietokantoihin ja API-yhteyksiin. Amplifyn avulla voi julkaista verkkosovelluksia suoraan pilveen ilman monimutkaista konfigurointia. (Amplify Concepts, N.D.)

Amplify toimii hyvin yhdessä nykyaikaisten JavaScript-kehysten, kuten Reactin, Next.js:n ja Vue:n kanssa. Se tukee myös mobiilialustoja, kuten iOS:ää ja Androidia. Kehittäjälle tarjolla on helppokäyttöinen käyttöliittymä ja CLI-työkalut, joilla pilvipalveluiden hallinta onnistuu sujuvasti. Amplify sopii erityisen hyvin projekteihin, joissa halutaan päästä nopeasti liikkeelle ilman raskasta infrastruktuuria. (Amplify Concepts, N.D.)

Vaikka Amplify nopeuttaa sovelluskehitystä, sen käytössä on tiettyjä rajoitteita. Esimerkiksi Amplifyn tarjoama backend-ratkaisu ei toimi monorepossa siten, että se voitaisiin jakaa useampaan eri projektiin. Sen sijaan jokaiselle projektille on määritettävä oma backend, mikä voi aiheuttaa haasteita skaalautuvuuden ja hallinnan kannalta. Projekteissa, joissa tarvitaan laajempaa skaalautuvuutta, voi olla suositeltavaa käyttää vaihtoehtoista ratkaisua tai toteuttaa oma backend-ratkaisu. Amplify toimii loistavasti pikaisissa prototyypeissä tai kevyissä projekteissa.

4 Sovelluksen kehittämistyö

4.1 Käyttöliittymän mockup ja sen merkitys suunnittelussa

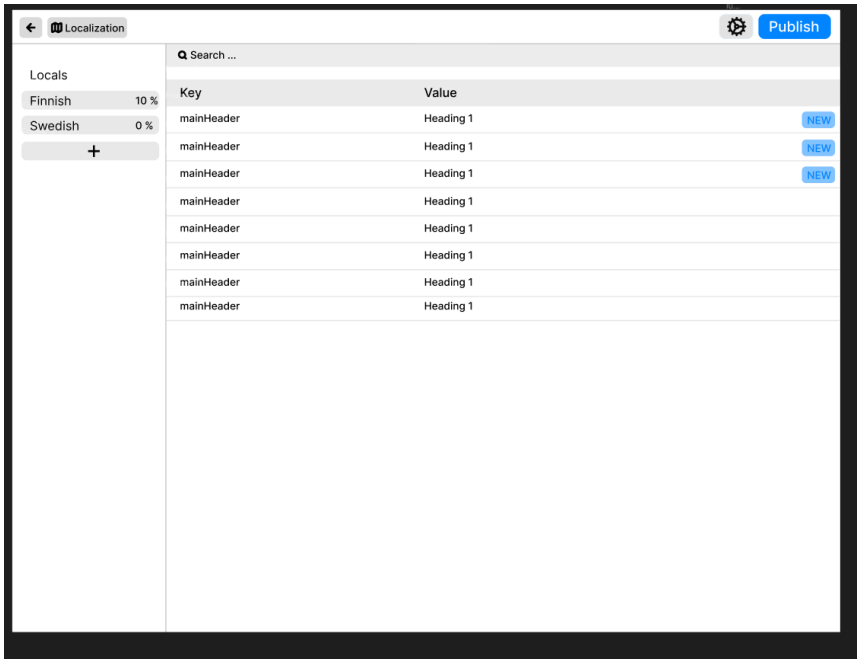
Mockup on käyttöliittymän suunnitelma miltä se voi näyttää. Sen avulla voidaan tarkastella käyttöliittymän rakennetta, visuaalisia elementtejä sekä käyttäjäkokemusta jo suunnittelun alkuvaiheessa. Mockup eroaa wireframesta siinä, että se sisältää enemmän graafisia yksityiskohtia, kuten värejä, fontteja ja ikonografiaa, mutta ei vielä interaktiivisia toimintoja. (Garrett, 2011)

Figma on yksi suosituimmista työkaluista mockupien luomiseen, ja Design Tools Survey 2021 -tutkimuksen mukaan 63 % vastaajista käyttää sitä päätyökalunaan käyttöliittymäsuunnittelussa.

Figma mahdollistaa:

- Yhteistyön reaaliajassa, jolloin tiimit voivat muokata suunnitelmia samanaikaisesti
- Responsiivisten asettelujen rakentamisen, jolloin suunnitelmat mukautuvat eri näyttökokoihin
- Käyttöliittymän arvioinnin ennen toteutusta, jotta elementtien asettelu ja käyttäjäkokemus voidaan testata ajoissa

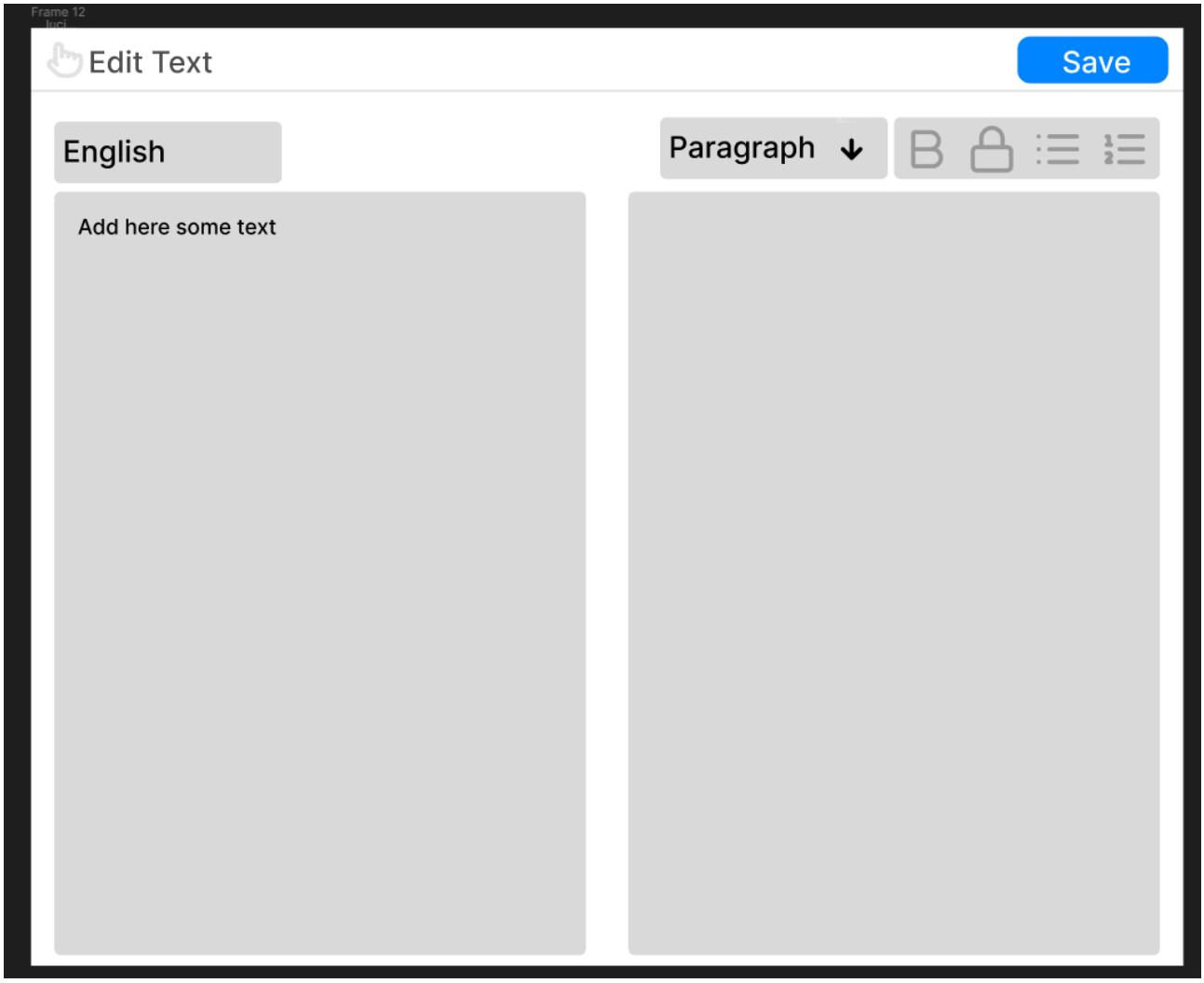
Mockupin hyödyntäminen suunnittelussa mahdollistaa käyttöliittymän arvioinnin jo ennen varsinaista toteutusta. Tämä auttaa tunnistamaan käytettävyyteen liittyviä ongelmakohtia aikaisessa vaiheessa. (Holst, N.D.)



Kuvio 2. Käyttöliittymän mockup.

Kuviossa 2 on yleisnäkymä käännöstyökalusta. Ylhäällä sijaitsee navigointipalkki, jossa on painike edelliselle sivulle, asetuspainike sekä julkaisupainike. Vasemmalla puolella näkyvät tuetut kielet listattuna, ja siellä on myös mahdollisuus lisätä uusia kieliä. Oikealla puolella lopputila on varattu käännösavainten listaukselle. Käyttäjä voi vaihtaa listattuja käännösavaimia napsauttamalla vasemmalla olevaa kielivalintaa.

Listausnäkymässä on lisäksi hakupalkki, jonka avulla voi suodattaa käännösavaimia hakuehtojen mukaisesti. Klikkaamalla yksittäistä käännösavainta käyttäjä avaa varsinaisen käännöstyökalun, joka on esitetty kuviossa 2.



Kuvio 3. Yksittäisen käännöksen muokkausnäkymä

Kuviossa 3 on hahmoteltu yksittäisen käännöksen muokkausnäkymä. Vasemmalla puolella näkyy alkuperäinen teksti, ja oikealle puolelle kirjoitetaan käännös. Yläosassa on tekstin muokkaamiseen liittyviä työkaluja sekä elementtivalitsin, jonka avulla voi valita esimerkiksi listauksen tai muun rakenteellisen elementin. Tämä näkymä mahdollistaa tehokkaan käännöstyön hallinnan ja tarjoaa selkeän erottelun alkuperäisen ja käännetyn sisällön välillä.

4.2 Shadcn/ui ja Radix UI

Sovelluksen käyttöliittymä on toteutettu hyödyntäen Shadcn/ui-komponenttikirjastoa, joka tarjoaa saavutettavia ja helposti muokattavia React-komponentteja. Toisin kuin perinteiset npm-paketit, Shadcn/ui toimii niin sanotulla "copy-paste"-periaatteella, jossa komponentit lisätään suoraan sovelluksen lähdekoodiin. Tämä vähentää kirjastoriippuvuuksia ja tekee komponenttien muokkaamisesta suoraviivaisempaa omien tarpeiden mukaan. Kyseinen lähestymistapa on nopeasti yleistynyt, ja kirjastosta on tullut yksi suosituimmista React-ekosysteemissä. Florian (2024) toteaa, että Shadcn on kerännyt yli 30 000 tähteä GitHubissa ja noussut monien kehittäjien oletusvalinnaksi käyttöliittymäkomponenttien toteutukseen. (Florian, 2024)

Shadcn/ui rakentuu Radix UI -kirjaston varaan, joka tarjoaa niin sanottuja “headless”-komponentteja. Ne on tarkoituksella jätetty tyylittelemättä, jotta kehittäjä voi itse määritellä ulkoasun haluamallaan tavalla. Samalla säilyy komponenttien saavutettavuus ja toiminnallisuus. Tämä nopeuttaa kehitystyötä, koska valmiit komponentit, kuten modaalit, valikot ja lomakekentät, toimivat heti sellaisenaan, mutta ne on helppo muokata sopimaan omaan käyttöliittymään. (Shadcn, N.D)

Shadcn/ui yhdistää Radix UI:n toiminnallisuuden Tailwind CSS:n tyylittelymahdollisuuksiin, tarjoten valmiiksi tyyliteltyjä, mutta helposti muokattavia komponentteja. (Shadcn, N.D)

Shadcn asennettaan projektissa käyttöön seuraavasti. Kuvion 4:n komennon mukaan initialisoidaan projekti ja komento luo tarvittavat määritykset ja tiedostot projektiin, jotta komponentit saa käyttöön.

```
1. pnpm dlx shadcn@latest init
```

Kuvio 4. Shadcn käyttöönotto.

Shadcn sivuilla on näkyvissä saatavilla olevat komponentit. Komponentti lisätään projektiin Kuvio 5:n mukaisella komennolla.

```
1. pnpm dlx shadcn@latest add input
```

Kuvio 5. Shadcn komponentin lisäys.

Lisäsin package.json:iin Kuvio 6 komento skriptiksi, helpommalla komennolla, jotta se olisi helpompi muistaa.

```
1.   "scripts": {
6.     "add:ui": "pnpm dlx shadcn@latest add"
7.   },
8.
```

Kuvio 6. Package.json script.

```
import { cn } from "@/lib/utils"

const Input = React.forwardRef<HTMLInputElement, React.ComponentProps<"input">>(
  ({ className, type, ...props }, ref) => {
    return (
      <input         type={type}
        className={cn(
          "flex h-9 w-full rounded-md border border-input bg-transparent px-3 py-1 text-base shadow-sm
transition-colors file:border-0 file:bg-transparent file:text-sm file:font-medium file:text-foreground
placeholder:text-muted-foreground focus-visible:outline-none focus-visible:ring-1 focus-visible:ring-
ring disabled:cursor-not-allowed disabled:opacity-50 md:text-sm",
          className         )}
        ref={ref}
        {...props}
      />     )
  }
)
Input.displayName = "Input"
export { Input }
```

Kuvio 7. Shadcn input-komponentin lähdekoodi (Shadcn, N.D)

Esimerkki Shadcn asentamasta komponentista Kuvio 7. Komponentti on helposti muokattavissa koska se on projektissa asetettu haluamaan kansioon. (Shadcn, N.D)

4.3 Tailwind CSS

Tailwind CSS on moderni utility-first-lähestymistapaa hyödyntävä CSS-kehys, joka mahdollistaa tyylien määrittämisen suoraan HTML-rakenteessa ilman perinteisiä tyylitiedostoja. Tailwindcss-tyyliparametrit määritetään HTML-komponenttien luokkamäärittelyissä. Tyyliparametrit ovat lyhennettyjä normaalista CSS-määrittelyistä, esimerkiksi "display: flex" määritellään tailwindcss:ssä "flex". Tyylien määrittely suoraan HTML-komponentteihin jakaa mielipiteitä, ja siinä on omat heik-

koutensa, esimerkiksi komponenttien lukeminen voi olla raskasta, jos Tailwind-määrittelyjä on paljon, verrattuna siihen, että CSS-tyylit olisi eriytetty omiin tiedostoihin. (Get started with Tailwind CSS, N.D.)

Yksi Tailwindin suurimmista eduista on kuitenkin sen kyky optimoida lopullinen CSS-tiedosto automaattisesti. Tuotantoversiossa kaikki käyttämättömät tyylit poistetaan, mikä tekee lopullisesta CSS-bundlesta erittäin kevyen. Tämä parantaa suorituskykyä ja nopeuttaa sivujen latautumista. Lisäksi Tailwind tukee nykyaikaisia CSS-ominaisuuksia, kuten CSS Grid -asettelua, animaatioita ja muuttujia. (Get started with Tailwind CSS, N.D.)

4.4 PNPM

pnpm on moderni paketinhallintatyökalu, joka nopeuttaa riippuvuuksien asentamista ja säästää levytilaa verrattuna perinteisiin työkaluihin kuten npm tai Yarn. pnpm asentaa tietyn paketin vain kerran ja se jaetaan eri projektien kesken hard linkkien avulla. Tämä vähentää tiedostojen kopiointia ja säästää tilaa etenkin projekteissa, joissa on useita sovelluksia tai kirjastoja. Lisäksi pnpm päivittää vain muuttuneet tiedostot sen sijaan, että lataisi koko paketin uudelleen, mikä nopeuttaa työskentelyä entisestään. (Motivation, N.D.)

pnpm:n asennusprosessi etenee kolmessa vaiheessa: ensin se lataa tarvittavat riippuvuudet yhteiseen varastoon, sitten laskee node_modules-hakemiston rakenteen ja lopuksi linkittää paketit projektiin. Tämä tekee pnpm:stä selvästi nopeamman vaihtoehdon verrattuna perinteisiin menetelmiin, joissa jokainen paketti kirjoitetaan erikseen levylle. Lisäksi pnpm hyödyntää symlinkkejä varmistaakseen, että vain projektin itse määrittelemät riippuvuudet ovat saatavilla juuritasolla. (Motivation, N.D.)

4.5 AWS Amplify

Projekteissa käytetään AWS amplify työkalua. Amplify:ssa on valmiina hyödyllisiä osia, mihin ei itse tarvitse käyttää resursseja tällä hetkellä. Harmillisesti AWS amplify:n tarjoama backend ratkaisu ei sovellu vahvasti monorepon käyttöön, esimerkiksi dynaamisen tyypitysten vuoksi, mitä ei saa tehtyä erillisenä pakettina vaan jokainen amplify backend täytyy projektikohtaisesti määrittää repositorioon, tässä olisi parempi kehittää oma backend-ratkaisu, mutta rajalliset resurssit. Amplify

myös integroituu erinomaisesti aws palveluihin mitä käytetään yrityksessä. Osittain käännöstyökalu on myös omanlaisensa palvelu ja se ei oikein sovi täysin npm-paketti ajatteluun. Tämän takia käännöstyökalua kehitetään omassa repositoriossa omana palvelunaan.

```
version: 1
backend:
  phases:
    preBuild:
      commands:
        - npm install -g pnpm
    build:
      commands:
        - pnpm install
        - pnpm exec ampx pipeline-deploy --branch $AWS_BRANCH --app-id $AWS_APP_ID --debug
frontend:
  phases:
    build:
      commands:
        - pnpm run build
  artifacts:
    baseDirectory: .next
    files:
      - "**/*"
  cache:
    paths:
      - .next/cache/**/*
      - "$(pnpm store path)"

```

Kuvio 8. Amplify.yml.

Kuviossa 8 on amplify.yml pipeline-määritystiedosto, jota on muokattu mahdollistamaan pnpm-paketinhallinnan käyttö. Alkuperäinen pipeline-määritys oli suunniteltu ainoastaan npm:lle, jolloin oli kokeilta ratkaisu pnpm käyttöä varten.

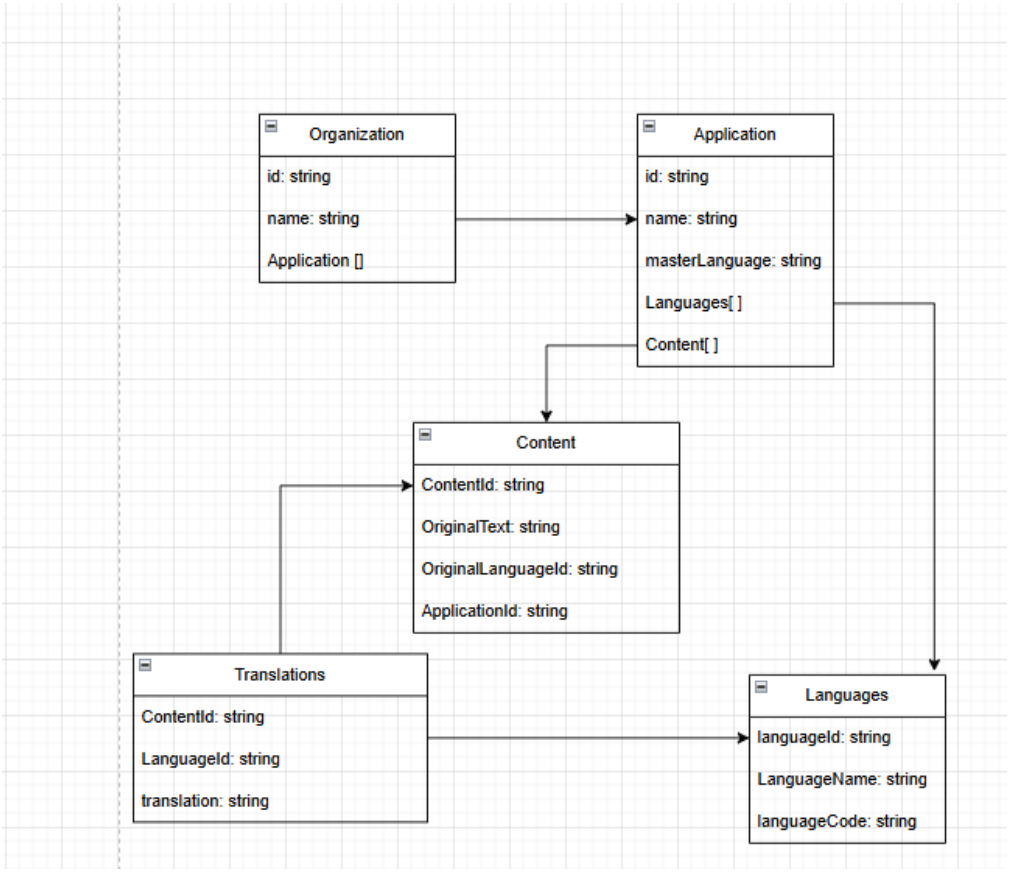
preBuild-vaiheessa asennetaan ensin pnpm globaalisti, mikä mahdollistaa pnpm-komentojen käytön. Lisäksi välimuistin hallintaa optimoidaan lisäämällä Linux-komento $(pnpm store path), joka varmistaa, että pnpm:n käyttämä content-addressable store otetaan mukaan välimuistiin. Tämä nopeuttaa seuraavia build-prosesseja, koska riippuvuuksia ei tarvitse ladata uudelleen jokaisessa pipeline-suorituksessa.

4.6 Tietokannan suunnittelu ja relaatiot Amplify:ssä

Monikielinen tietokantasuunnittelu on tärkeä osa sovelluskehitystä silloin, kun tavoitteena on tarjota tuki useille kielille. Suunnittelussa keskeistä on valita sellainen tietomalli, joka mahdollistaa uusien kielten lisäämisen ilman tarvetta jatkuville muutoksille sovelluksen koodiin. Du Mortier (2022) korostaa, että yksinkertaisimmat ratkaisut voivat vaikuttaa nopeilta toteuttaa, mutta ne

saattavat johtaa ylläpidon ja skaalautuvuuden kannalta ongelmallisiin rakenteisiin. Siksi suunnittelussa kannattaa panostaa joustavuuteen ja pitkäikäisiin ratkaisuihin.

Tässä työssä valittu tietomalli perustuu siihen, että jokainen käännös tallennetaan omalle rivilleen saman taulun sisällä. Jokaisella rivillä on viittaus alkuperäiseen tietueeseen sekä kielikoodi, joka määrittää, mille kielelle sisältö kuuluu. Du Mortier (2022) toteaa, että tällainen rakenne mahdollistaa uusien kielten lisäämisen ilman, että taulun rakennetta tarvitsee muuttaa. Vaikka ratkaisu saattaa kasvattaa tietokantakyselyiden määrää, se säilyttää tietomallin loogisuuden ja normalisoinnin, mikä helpottaa ylläpitoa pitkällä aikavälillä.



Kuvio 9. Sovelluksen tietokantamalli.

Sovelluksen tietokantamalli (ks. Kuvio 9) on suunniteltu tukemaan monikielistä sisällönhallintaa selkeällä ja skaalautuvalla rakenteella. Mallissa hyödynnetään yleisesti käytettyä kielimallia, jossa jokainen käännös tallennetaan erilliselle riville Translations-tauluun. Rakenne mahdollistaa uusien kielten lisäämisen ilman taulujen rakennemuutoksia.

Rakenne koostuu seuraavista päätauluista:

- **Organization**: Sisältää organisaation perustiedot ja sen sovellukset.

- **Application**: Liittyy tiettyyn organisaatioon ja määrittelee käytettävän pääkielen (master-Language) sekä kielivalikoiman (Languages[]) ja sisällöt (Content[]).

- **Content**: Sisältää alkuperäiset tekstielementit ja niihin liittyvän kielen ja sovelluksen tunnisteet.

- **Languages**: Yleinen kielitaulu, johon on tallennettu kielten nimet ja kielikoodit. Tämä tukee esimerkiksi en_US, fi_FI -muotoisia tunnisteita.

- **Translations**: Yhdistää sisällön ja kielen ja tallentaa käännöstekstit. Yksi ContentId voi liittyä useisiin käännöksiin eri LanguageId-arvoilla.

Taulukko 1. Amplify:n relaatiot.

Relaatio	Funktio	Kuvaus	Esimerkki
Yksi-moneen	a.hasMany(...) & a.belongsTo(...)	Luo yksi-moneen-suhteen kahden mallin välille.	Joukkueella on monta Pelaajaa. Pelaaja kuuluu yhteen joukkueeseen.
yksi-yhteen	a.hasOne(...) & a.belongsTo(...)	Luo yksi-yhteen-suhteen kahden mallin välille.	Pelaajalla on yksi maila. Maila kuuluu yhdelle pelaajalle.

moni-moneen	Kaksi `a.hasMany(...)` & `a.belongsTo(...)` liittymätauluissa	Luo kaksi yksi-moneen-suhdetta liittymätaulun kautta.	Opiskelijoilla on useita kursseja. Kurssilla on useita opiskelijoita.

AWS Amplify tukee relaatiotietokantojen yleisimpiä suhteita kuten taulukosta 1 näkyy. Yksi-moneen, yksi-yhteen ja moni-moneen -suhteita. Suhteet määritellään hasMany, hasOne ja belongsTo -funktioilla, joiden avulla voidaan hallita eri mallien välisiä yhteyksiä.

4.7 Next.js Route Groups

Next.js Route Groups on ominaisuus, joka mahdollistaa reittien järjestämisen ilman, että ryhmän nimi vaikuttaa lopulliseen URL-rakenteeseen. Tämä tekee siitä erityisen hyödyllisen sovelluksissa, joissa halutaan organisoida tiedostorakennetta selkeämmin ilman, että käyttäjälle näkyvä osoitepolku muuttuu. *(Route Groups, N.D)*

Route Groups -toiminnolla voidaan hallita myös edistyneitä layout-ratkaisuja. Sen avulla on mahdollista luoda useita sisäkkäisiä layoutteja samalle reittitasolle, jolloin eri näkymät voivat jakaa rakenteellisia elementtejä. Tämä mahdollistaa esimerkiksi sen, että vain tietyillä reittialueilla käytetään erillistä ulkoasua tai rakennetta. Lisäksi ominaisuutta voidaan hyödyntää latausanimaatioiden ja skeleton-komponenttien lisäämiseen, mikä parantaa käyttökokemusta varsinkin lataustilanteissa. *(Route Groups, N.D)*

Yksi yleinen käyttötapa on reittien jakaminen esimerkiksi ominaisuuksien, käyttöoikeuksien tai kieliversioiden mukaan – ilman että jaottelu näkyy suoraan URL-polussa. Käyttöönotto on yksinkertaista: hakemisto nimetään hakasuluilla, kuten `app/(admin)/dashboard/page.tsx`. Tällöin käyttäjä siirtyy osoitteeseen `/dashboard`, mutta tiedostorakenteessa admin-reitit pysyvät omassa ryhmässään. *(Route Groups, N.D)*

Projektin kansiorakenne.

```
src/
├── app/
│   ├── (authenticated)/          # Autentikoituja reittejä sisältävä ryhmä
│   │   ├── (admin)/              # Ryhmä admin-oikeuksien omaaville
│   │   ├── application/          # Sovellusten hallintaan liittyvät sivut
│   │   │   ├── [appId]/          # Dynaaminen reitti yksittäiselle sovellukselle
│   │   │   │   ├── page.tsx      # Sovelluksen käännösten hallintasivu
│   │   │   │   └── layout.tsx    # Sovelluksen sivun asettelu
│   │   │   └── page.tsx          # Sovellusten listaussivu
│   │   └── layout.tsx            # Autentikoitujen sivujen yhteinen asettelu
│   ├── components/               # Sovelluksen komponentit
├── components/                   # Shadcn/ui-komponentit
├── lib/                          # Apufunktiot
└── hooks/                        # React-hookit
```

Kuvio 10. Projektin reititys.

Kuvassa 10 on avattu projektin kansiorakenne ja reititykset. app/-hakemiston sisällä on (authenticated)/-kansio, joka sisältää reitit, jotka vaativat käyttäjän autentikoinnin. Sen sisällä on edelleen (admin)/-kansio, joka erottelee reitit, joihin vain järjestelmänvalvojilla on pääsy. Sovellusta hallintaan osoitteesta /application mikä näyttää organisaatio näkymän, sekä listan sovelluksista, mitkä pitää kääntää. Dynaaminen [appId]/-reitti näyttää sovelluksen tiedot ja listan käännöstiedostoista.

Sovelluksen UI-komponentit sijaitsevat components/-kansiossa, ja erilliset Shadcn/ui-kirjaston komponentit löytyvät ylemmältä tasolta components/-hakemistosta. lib/-hakemisto sisältää sovelluksen apufunktiot, kun taas hooks/-kansiossa sijaitsevat uudelleenkäytettävät React-hookit.

4.8 Amplify client Nextjs sovelluksessa.

Sovelluksessa Amplify Client konfiguroidaan omaksi komponentikseen, joka alustaa Amplify.configure-funktion avulla amplify_outputs.json-tiedoston sisällön. Tämä tiedosto sisältää AWS-palveluihin liittyvät määritykset, jotka Amplify on automaattisesti luonut projektin käyttöönoton yhteydessä.

```
"use client";
```

```
 2.
 3. import { Amplify } from "aws-amplify";
 4. import { Schema } from "@/amplify/data/resource";
 5. import { generateClient } from "aws-amplify/api";
 6. import config from "../../../../amplify_outputs.json";
 7.
 8. Amplify.configure(config);
 9.
10. export default function ConfigureAmplifyClientSide() {
11.   return null;
12. }
13.
14. export const amplifyClient = generateClient<Schema>();
```

Kuvio 11. Amplify client komponentti.

Komponentti Kuviossa 11 ConfigureAmplifyClientSide ei renderöi mitään, vaan sen ainoa tehtävä on käynnistää Amplify Clientin konfigurointi Next.js-sovelluksen asiakaspuolella. generateClient-funktiolla luodaan amplifyClient, jota voidaan hyödyntää sovelluksen eri osissa esimerkiksi datan hakemiseen TanStack Queryn avulla.

```
 1. export default function AuthenticatedLayout({
 2.   children,
 3. }: {
 4.   children: React.ReactNode;
 5. }) {
 6.   return (
 7.     <Providers>
 8.       <ConfigureAmplifyClientSide />
 9.       {children}
10.     </Providers>
11.   );
12. }
13.
```

Kuvio 12. Amplify client komponetin lisääminen Nextjs komponenttiin.

Kuvan 11 komponentti ConfigureClientSide() lisätään Kuvion 12 komponenttiin , jolloin se suoritetaan aina, kun käyttäjä on kirjautuneena. Näin varmistetaan, että Amplify Client on käytettävissä kaikissa suojatuissa näkymissä ilman, että konfigurointia tarvitsee suorittaa erikseen jokaisessa komponentissa.

4.9 Tanstack Query

TanStack Query on tehokas kirjasto, joka yksinkertaistaa palvelinperäisen datan hakua, välimuistitusta ja synkronointia verkkosovelluksissa. Se ratkaisee perinteisiä haasteita, kuten tietopyyntöjen liiallisen lataamisen ja datan automaattisen päivittämisen taustalla, jolloin verkkosovellus tehokkuus paranee. (TanStack Query Overview, N.D)

Toisin kuin perinteiset tilanhallintaratkaisut, TanStack Query on suunniteltu nimenomaan palvelinperäisen datan hallintaan. Se optimoi suorituskyvyn välimuistin avulla ja vähentää turhia tietopyyntöjä, mikä parantaa sovelluksen nopeutta ja käyttäjäkokemusta. Tämä tekee siitä erinomaisen työkalun moderneille web-sovelluksille, joissa palvelindatan hallinta on keskeinen osa kehitystä. (TanStack Query Overview, N.D)

```
1. import { useQueryClient, useQuery } from "@tanstack/react-query";
 2. import { useParams } from "next/navigation";
 3.   const params = useParams();
 4.   const appId = params.appId as string;
 5.
 6.   const { data: existingLanguages } = useQuery({
 7.     queryKey: ["languages", appId],
 8.     queryFn: async () => {
 9.       try {
10.         const result = await amplifyClient.models.Language.list({
11.           filter: { applicationId: { eq: appId } },
12.         });
13.         return result.data || [];
14.       } catch (error) {
15.         console.error("Error fetching languages:", error);
16.         return [];
17.       }
18.     },
19.     enabled: !!appId,
20.     staleTime: 0,
21.   });
22.
23.
```

Kuvio 13. Esimerkki React-Query:n käyttämisestä Amplify client:n kanssa.

useQuery hook yhdistetään Amplify Clientiin kuvion 13 mukaisesti. useQuery-hook hyödyntää queryKey-parametria, mikä tarkoittaa, että kysely suoritetaan uudelleen aina, kun appId muuttuu. queryFn-parametri sisältää Amplify Clientin, jonka avulla sovellus hakee tarvittavat tiedot palvelimelta. Tässä haetaan lista kielistä, jotka liittyvät tiettyyn sovellukseen. queryFn suorittaa Amplify Clientin kyselyn, joka suodattaa tulokset applicationId-kentän perusteella.

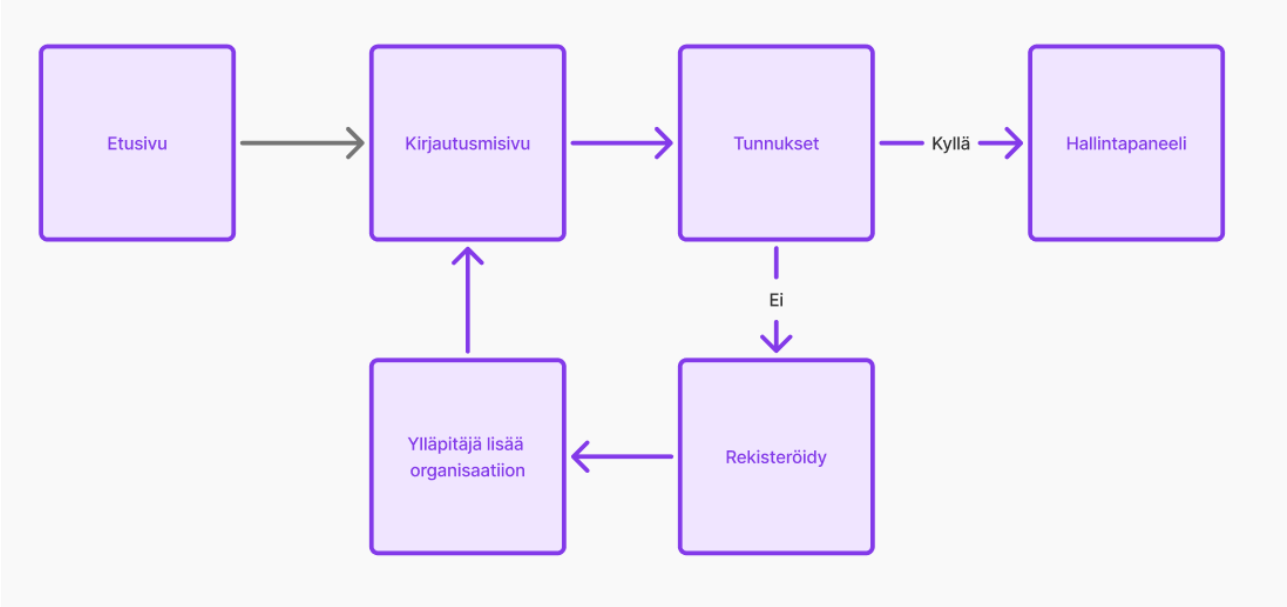
5 Sovellus

5.1 Alkunäkymä ja kirjautumisprosessi

Kuvio 14. Etusivu.

Sovelluksen etusivulle on toteutettu yksinkertainen niin sanottu landing page (ks. Kuvio 14), jossa käyttäjä kohtaa heti selkeän ja kutsuvan näkymän. Sivun keskellä on painike ”Get Started Now”, josta klikkaamalla pääsee siirtymään varsinaiseen sovellukseen. Sovellus kulkee tällä hetkellä nimellä MovyaFlow, mutta nimi saattaa vielä muuttua kehityksen edetessä.

Landing page toimii ikään kuin lyhyenä johdantona palveluun, siinä kerrotaan lyhyesti, mitä sovellus tekee ja mitä hyötyä käyttäjälle on. Tämä auttaa erityisesti uusia käyttäjiä hahmottamaan nopeasti palvelun tarkoituksen ennen kuin siirrytään varsinaisiin toiminnallisuuksiin.



Kuvio 15. Käyttäjän kirjautumisen

Kuviossa esitetään sovelluksen kirjautumisprosessi vaiheittain (ks. Kuvio 15). Käyttäjä aloittaa etenemisen etusivulta, josta siirrytään kirjautumissivulle. Mikäli käyttäjää ei tunnisteta (eli hänellä ei ole oikeuksia organisaatioon), ohjataan hänet joko rekisteröitymään tai järjestelmänvalvojan toimesta lisättäväksi olemassa olevaan organisaatioon.

Uusi käyttäjä voi rekisteröityä järjestelmään itsenäisesti. Rekisteröitymisen jälkeen järjestelmänvalvoja lisää käyttäjän oikeaan organisaatioon hallintapaneelissa. Kun rekisteröinti on onnistunut ja organisaatioyhteys on määritelty, käyttäjälle avautuu pääsy hallintapaneeliin, jossa hän voi muokata käännöksiä ja hallita sisältöä.

5.2 Sovelluksen hallinta pääkäyttäjänä

Lambda funktio Amplify:llä

Amplify-projektiin on luotu erillinen Lambda-funktio, jonka tehtävänä on hakea lista kaikista rekisteröityneistä käyttäjistä Cognito User Poolista. Tämä toiminnallisuus on oleellinen osa hallintanäkymää, jossa järjestelmänvalvoja voi tarkastella käyttäjiä ja liittää heidät oikeaan organisaatioon. Järjestelmänvalvojalla on oma näkymä, jossa on kaikki rekisteröityneet käyttäjät ja mahdollisuus määrittää mihin organisaation kukin käyttäjä kuuluu.

Amplify:ssä ei ole suoraan kaikkien käyttäjien listausta ja on luotava oma lambda funktio tätä varten.

Luodaan tiedosto polkuun: data/resource/all-users/handler.ts

```
import { env } from "$amplify/env/all-users-handler";
import type { Schema } from "../resource";
import {
  CognitoIdentityProviderClient,
  ListUsersCommand,
} from "@aws-sdk/client-cognito-identity-provider";

const client = new CognitoIdentityProviderClient();

export const handler: Schema["allUsers"]["functionHandler"] = async () => {
  const command = new ListUsersCommand({
    UserPoolId: env.AMPLIFY_AUTH_USERPOOL_ID,
  });

  const response = await client.send(command);

  const Users: Schema["AllUsersResponse"]["type"] = {
    users:
      response.Users?.map((user) => {
        return {
          Username: user.Username || "",
          UserStatus: user.UserStatus || "",
          UserCreateDate: user.UserCreateDate?.toISOString() || "",
          UserLastModifiedDate: user.UserLastModifiedDate?.toISOString() || "",
          Enabled: user.Enabled || false,
          Attributes:
            user.Attributes?.map((attribute) => {
              return {
                Name: attribute.Name || "",
                Value: attribute.Value || "",
              };
            }) || [],
        };
      }) || [],
  };

  return Users;
};


```

Kuvio 16. Lambda funktio käyttäjien hakuun.

Kuviossa 16 esitetään AWS Amplify -projektiin liitetty Lambda-funktio, joka hakee Cognito User Poolista listan kaikista rekisteröityneistä käyttäjistä. Funktio hyödyntää AWS SDK:n CognitoIdentityProviderClient-asiakasta ja ListUsersCommand-komentoa, jolla saadaan haettua käyttäjät tietystä user poolista. Palautettava Users-objekti muotoillaan vastaamaan määriteltyä skeemaa. Se sisältää käyttäjien perustiedot, kuten käyttäjänimen, tilan ja aikaleimat. Lisäksi haetaan kaikki käyttäjään liittyvät attribuutit, jotka voidaan hyödyntää esimerkiksi käyttäjäprofiilien luonnissa tai käyttöoikeuksien määrittelyssä.

```
allUsers: a
    .query()
    .returns(a.ref("AllUsersResponse"))
    .authorization((allow) => [allow.group("Admins")])
    .handler(a.handler.function(allUsersHandler)),

  UserAttribute: a.customType({
    Name: a.string(),
    Value: a.string(),
  }),

  User: a.customType({
    Attributes: a.ref("UserAttribute").array(),
    Enabled: a.boolean(),
    UserCreateDate: a.datetime(),
    UserLastModifiedDate: a.datetime(),
    UserStatus: a.string(),
    Username: a.string(),
  }),

  AllUsersResponse: a.customType({
    users: a.ref("User").array(),
  }),

```

Kuvio 17. Käyttäjien hakuun liittyvät tietokantamäärittelyt.

Kuviossa 17 Rakennetaan allUsers-kysely, jota voidaan hyödyntää käyttöliittymäpuolella käyttäjien hallintaan. Kysely palauttaa AllUsersResponse-rakenteen ja on rajattu käyttöoikeuksiltaan ainoastaan käyttäjille, jotka kuuluvat "Admins"-ryhmään. UsersAttribute on avain-arvo -pari, jonka avulla käyttäjien yksittäiset ominaisuudet voidaan tallentaa ja hakea. User-määritys sisältää kaikki oleelliset käyttäjätiedot, kuten käyttäjätunnuksen, käyttäjästatuksen, rekisteröinti- ja muokkauspäivämäärät, sekä listan attribuuteista. Näiden perusteella voidaan hallita käyttäjäkohtaisia asetuksia järjestelmässä. AllUsersResponse muodostaa kokonaisuuden, joka koostuu taulukosta User-määrittelyjä.

```
export const auth = defineAuth({
  loginWith: {
    email: true,
  },
  groups: ["Admins"],
  access: (allow) => [allow.resource(allUsersHandler).to(["allUsers"])],

```

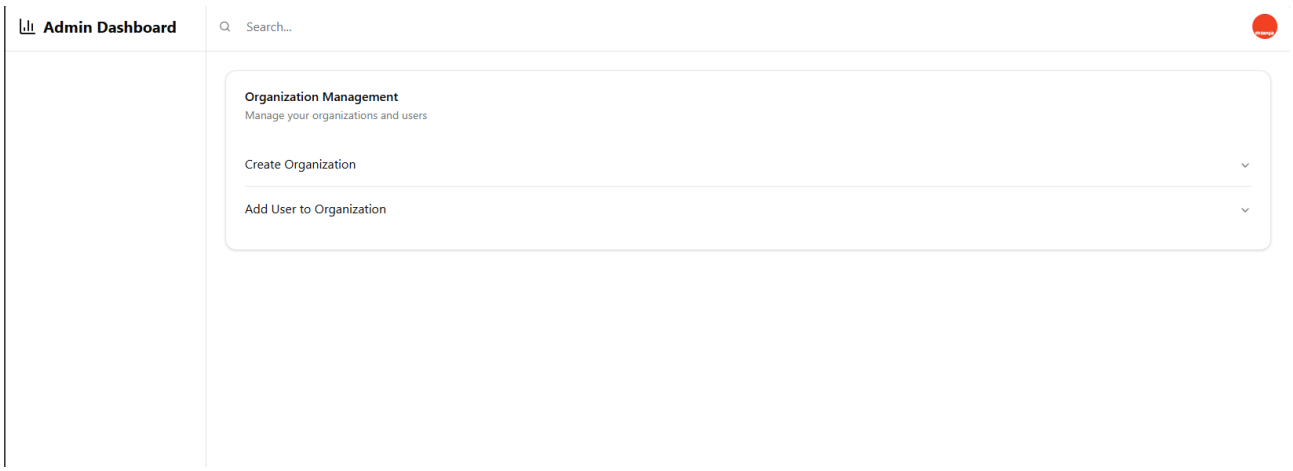
Kuvio 18. Amplify oikeuksien määrittely.

Annetaan järjestelmälle oikeudet käyttäjätietojen hakemiseen määrittelemällä auth-määritykseen resurssille pääsy (ks. kuvio 18). Määrittelyssä otetaan käyttöön sähköpostipohjainen kirjautuminen sekä määritetään käyttäjäryhmäksi "Admins". Tämän lisäksi rajataan pääsy allUsersHandler-resurssiin vain niille käyttäjille, joilla on oikeus suorittaa allUsers-toiminto.

```
1. import { amplifyClient } from "@/app/components/amplify-client/amplify-client";
2.
3.
4. const response = await amplifyClient.queries.allUsers({
5. authMode: "userPool",
6. });
7.
8.
```

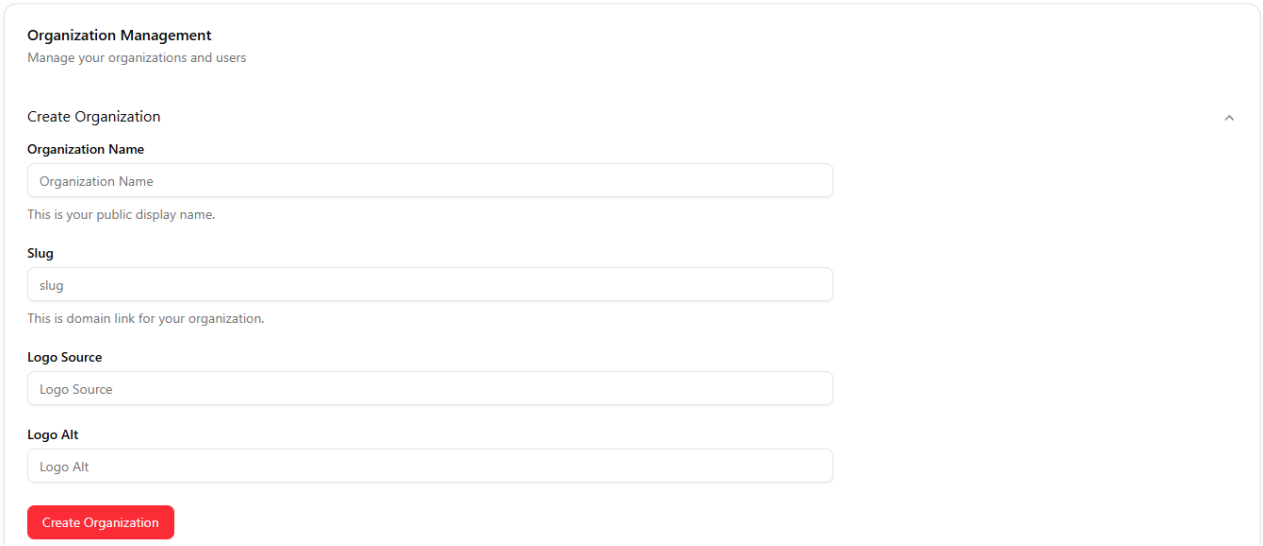
Kuvio 19. Esimerkki käyttäjien hausta käyttöliittymän lähdekoodissa.

Kuviossa 19 on esitetty esimerkki allUsers-funktion käytöstä käyttöliittymän lähdekoodissa. Funktion avulla voidaan hakea kaikki Cognito User Pooliin rekisteröidyt käyttäjät. Kysely suoritetaan Amplify Clientin allUsers-funktion kautta, ja autentikointitavaksi määritetään userPool.



Kuvio 20. Ylläpitäjän etusivu.

Perusnäkymä admin-polusta (ks. Kuvio 20). Tällä hetkellä ylläpitäjä hallitsee organisaation lisäämistä ja siihen liitettäviä käyttäjiä. Asiakas rekisteröityy sähköpostilla sovellukseen ja ylläpitäjä käy erikseen lisäämässä käyttäjän tiettyyn organisaatioon.



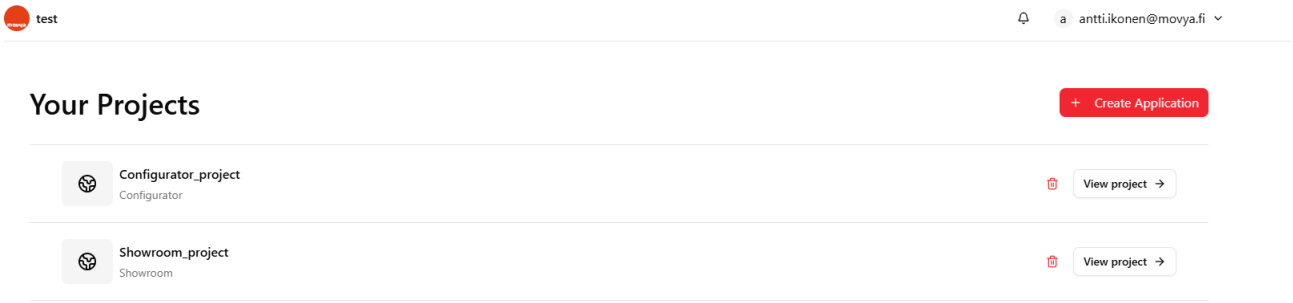
Kuvio 21. Organisaation lisääminen

Uuden organisaation lisääminen (ks. Kuvio 21) ja siinä tarvittavat tiedot. Organisaatiolle annetaan nimi, polkutieto (slug), yrityksen kuvan polku ja kuvan kuvailuteksti.

Kuvio 22. Käyttäjän lisääminen organisaatioon.

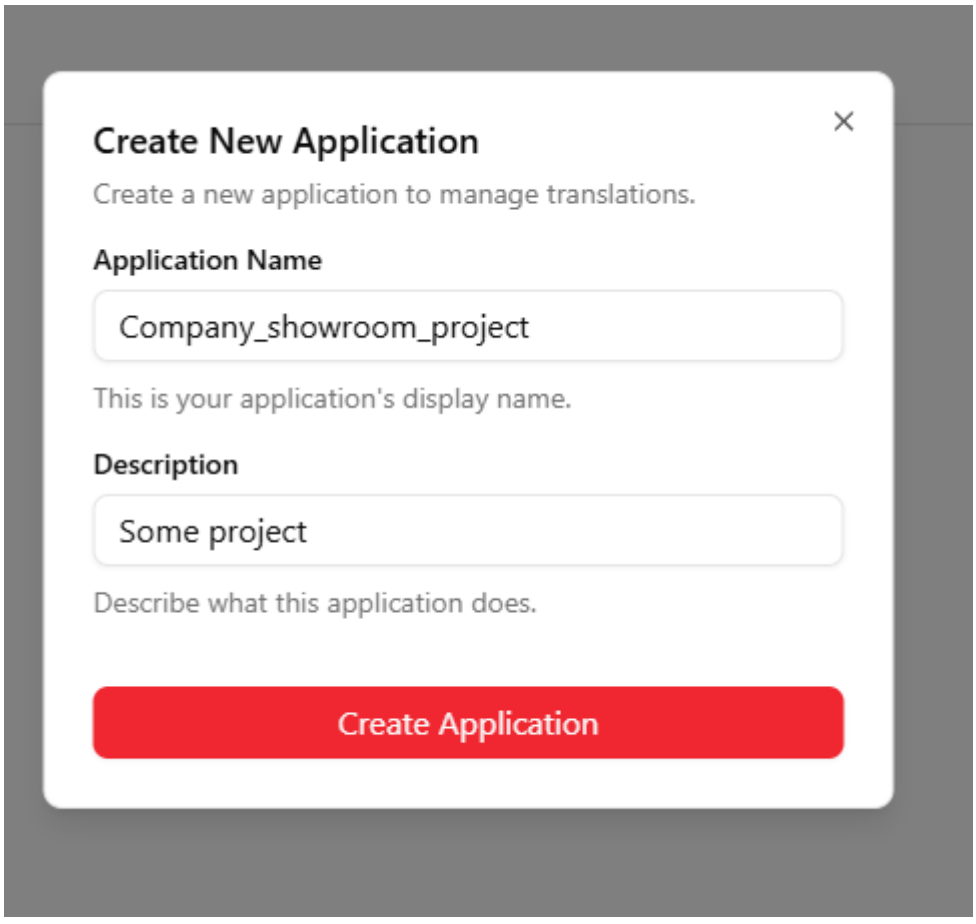
Käyttäjä lisätään organisaatioon (ks. Kuvio 22). Aiemmin oli käyty läpi kaikkien käyttäjien haku, User kohdassa on pudotusvalikko, mikä listaa kaikki rekisteröityneet käyttäjät. Tämän jälkeen valitaan organisaation pudotusvalikosta, minkä jälkeen käyttäjä pääsee nyt kirjautumaan verkkosovellukseen.

5.3 Sovelluksen toiminta käyttäjän näkökulmasta



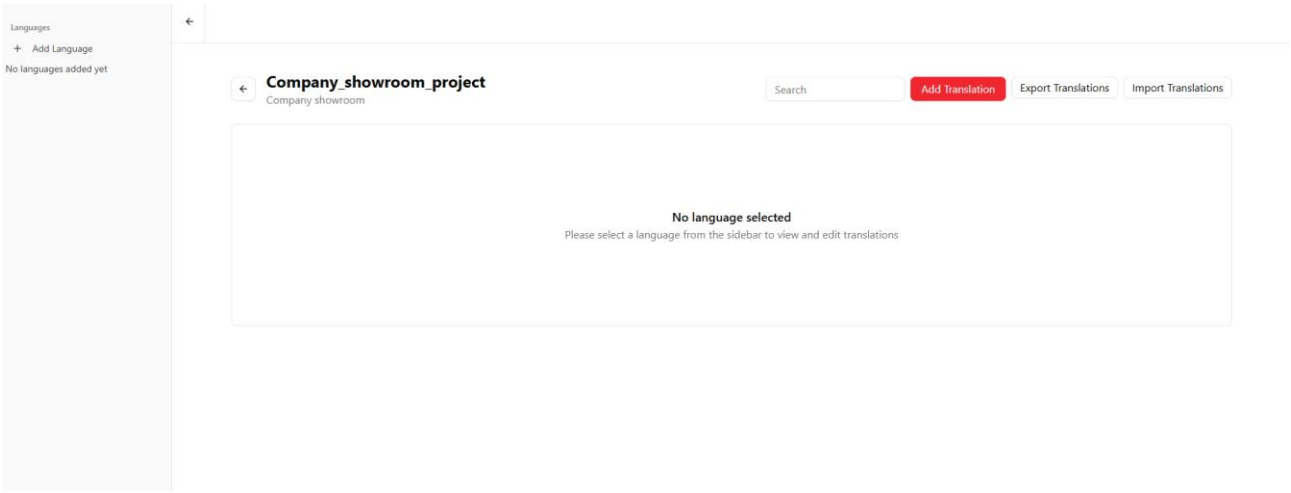
Kuvio 23. Käyttäjän hallintapaneelin näkymä.

Kun käyttäjä on kirjautunut, tulee näkyviin organisaation hallintapaneeli (ks. Kuvio 23), missä on lueteltu projektit ja mahdollisuus luoda uusi projekti.



Kuvio 24. Projektin lisääminen.

Luodaan uusi projekti, projektille on mahdollista antaa nimi ja lyhyt kuvaus (ks. Kuvio 24).



Kuvio 25. Projektin hallintasivu

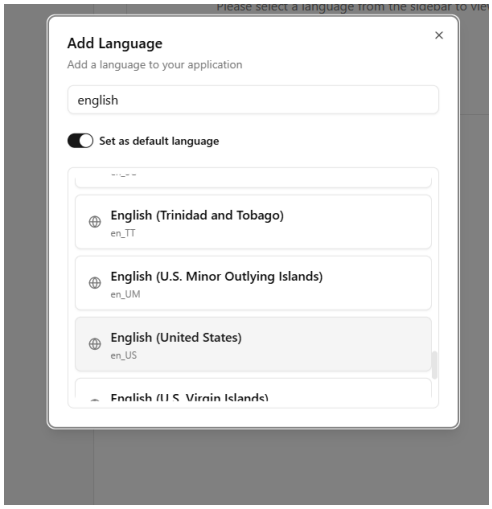
Projektin hallintasivu (ks Kuvio 25) sisältää sivupalkin, missä on lueteltu kielet ja päänäkymän, jossa on lueteltu kieliavaimet, sen hetkisen valitun kielen mukaan. Toimintoina oikealla on kieliavaimen lisääminen, haku/suodatustoiminto, kielitiedostojen tuonti ja vienti.

```
{
    "aa_DJ":[
        "Afar (Ddjibouti)",
        "ISO-8859-1"
    ],
    "aa_ER":[
        "Afar (Eritrea)",
        "UTF-8"
    ],

```

Kuvio 26. Esimerkki kielien koodeista JSON-muodossa.

Sovellus tukee kaikkia mahdollisia kieliversiota, mukaan lukien eri lokaalit kuten en_GB, en_US ja fi_FI. Kieliversiot on haettu JSON-locales sivustolta. Tämä laaja kielivalikoima toteutettiin hyödyntämällä JSON-muotoista tietorakennetta, johon on listattu kielikoodit ja merkistöasetukset. Esimerkki kieliversio JSON-objektista (ks. Kuvio 26).

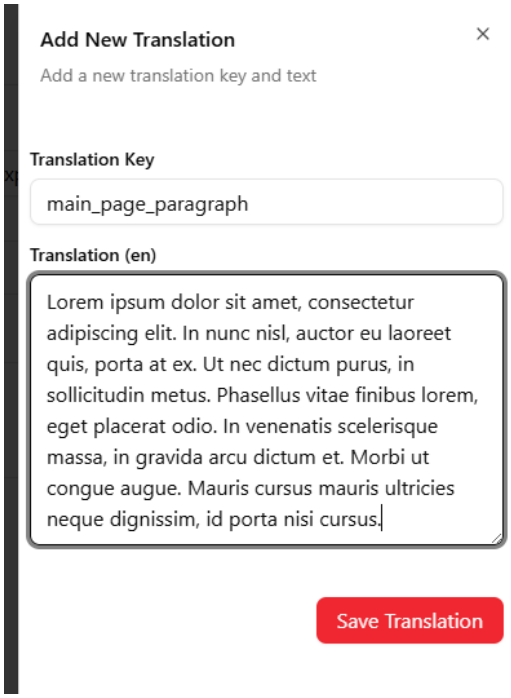


Kuvio 27. Kielen lisääminen.

Kieliversioita pääsee lisäämään sivupalkin ”Add languages” painikkeesta, joka aukaisee valintaikkunan, missä on lista kielistä ja haku- tai suodatustoiminto, sekä oletuskielen valinta (ks. Kuvio 27).

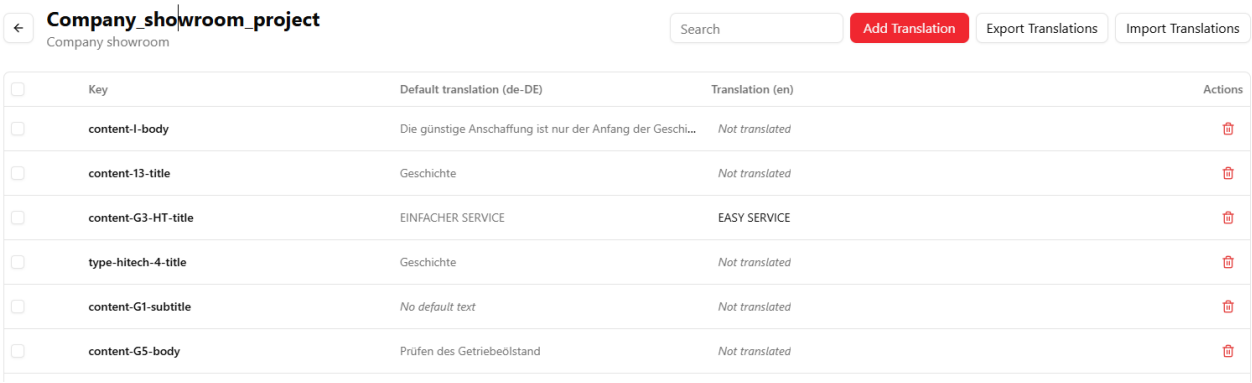
Kuvio 28. Oletuskielen valinnan vaihto.

Sivupalkissa on erillinen toiminto oletuskielen vaihtamiseen (ks. Kuvio 28).



Kuvio 29. Uuden käännöksen lisääminen.

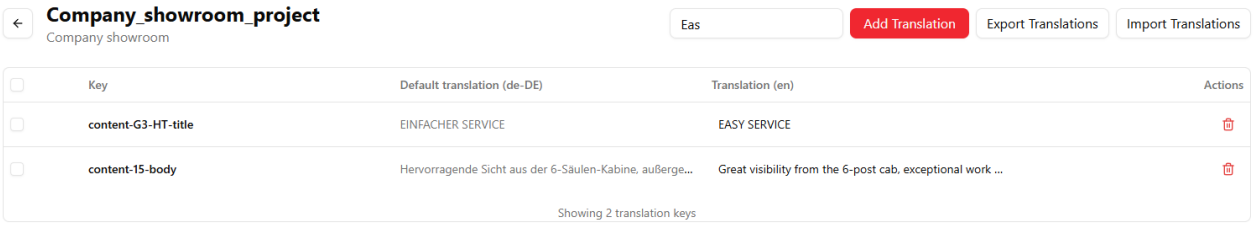
Käännöksen lisäämisessä on vain avain arvo pari (ks. Kuvio 29). Pari tallentuu ”Save Translation” painikkeesta.



Kuvio 30. Lista käännösavaimista.

Käyttöliittymässä näkyy luettelo käännösavaimista, joihin on liitetty oletuskäännös (tässä saksaksi, de-DE) sekä käännös valitulle kohdekielelle (tässä englanniksi, en) (ks. Kuvio 30). Jokaiselle

avaimelle näytetään myös mahdolliset toiminnot, kuten poistaminen. Hakutoiminnolla voidaan suodattaa käännösavaimia, ja yläreunan painikkeilla voidaan lisätä uusia käännöksiä, sekä viedä tai tuoda käännöstiedostoja.

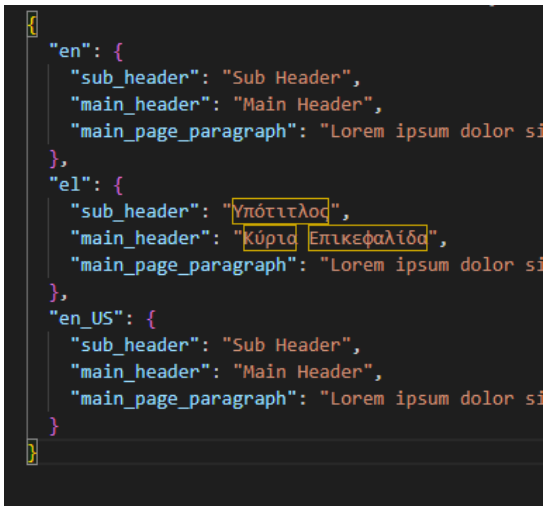


Kuvio 31. Hakutoiminto.

Sovelluksessa on hakutoiminto, mikä etsii osumia avaimesta, oletuskielestä ja käännöksestä. Käyttöliittymässä näytetään vain rivit, jotka vastaavat hakusanaa (ks. Kuvio 31).

Käännöstiedostot

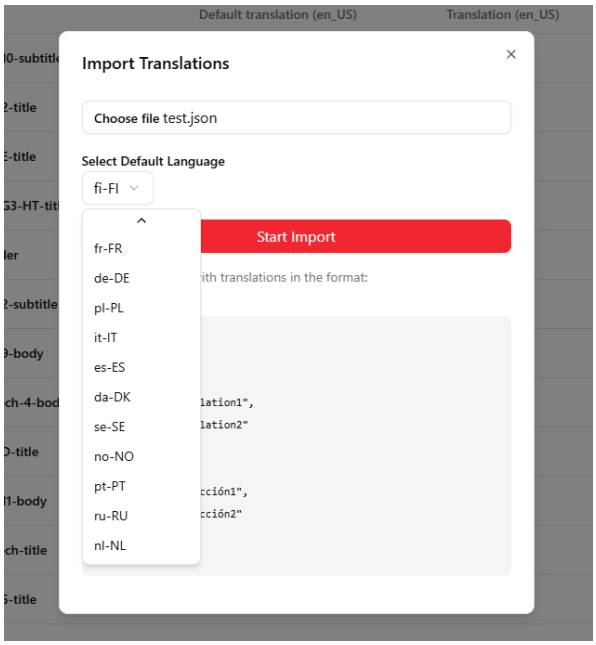
Sovelluksessa käännöstiedostot on toteutettu JavaScript-objekteina, koska se on tehokas ratkaisu käännösten hakemiseen ja käsittelyyn. JavaScript-objektien hakuoperaatiot ovat aikakompleksisuudeltaan O(1), mikä tarkoittaa, että käännösten hakeminen on erittäin nopeaa riippumatta käännösten määrästä.



Kuvio 32. Viety käännöstiedosto.

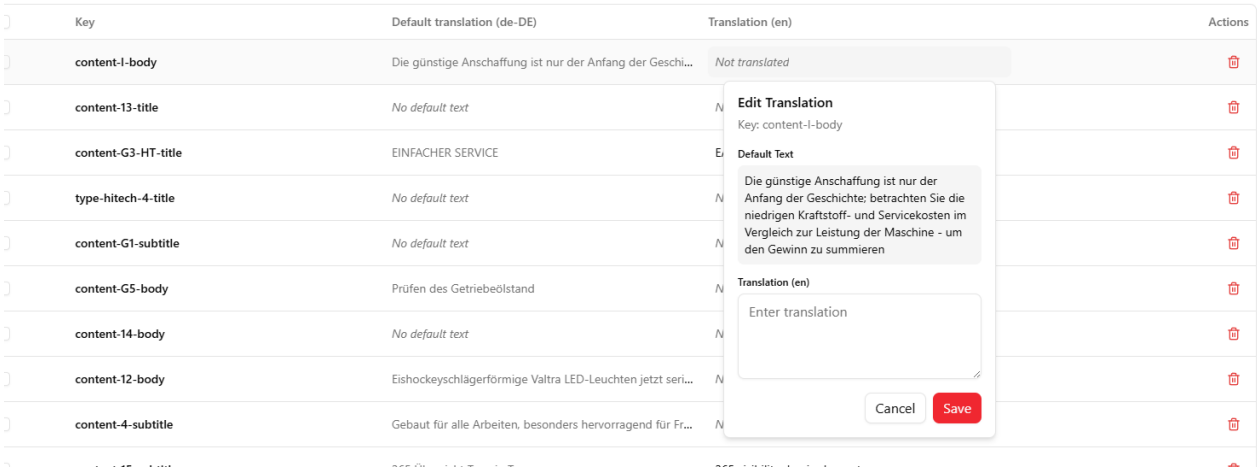
Exporter-komponentissa luodaan käännöstiedosto, joka on rakenteeltaan kaksiulotteinen objekti (ks Kuvio 32).

Käännöksien tuominen



Kuvio 33. Käännöstiedostosta haku.

Käännöksien tuomiselle sovellukseen on erillinen toiminto, mihin voi antaa JSON vastaavanlaisella rakenteelle kuin, mitä exporter-komponentti luo (ks. Kuvio 32). Jos objektin avainta (kieliversiota) ei löydy hyväksytyistä kielistä, sitä kieliversiota ei lisätä. Halutun mukainen JSON-tiedosto ladataan valmiiksi ja käyttöliittymä listaa tiedostossa olevat kielikoodit ja niistä muodostetaan lista, joista sitten valitaan oletuskieli. Tietojen viemisen tietokantaan voidaan aloittaa ”Start Import” painikkeesta (ks. Kuvio 33).



Kuvio 34. Käännöksen muokkaaminen.

Käännöksen lisäämistä tai muokkaamista eri kielivaihtoehdoissa pääsee tekemään painamalla halutun avaimen kohdalta "Translation" (en) -saraketta (ks. Kuvio 35). Muokkausikkunassa näytetään oletuskielen käännös (esimerkiksi saksa), ja käyttäjä voi syöttää uuden käännöksen valitulle kielelle (tässä englanniksi). Käännös tallennetaan painamalla Save-painiketta.

5.4 Tietueen poisto

CASCADE DELETE (kaskadipoistooperaatio) on tietokannan ominaisuus, joka automaattisesti poistaa kaikki riippuvaiset tietueet, kun päätietue poistetaan. Tämä varmistaa tietokannan eheyden ja estää "orpojen" tietueiden syntymisen. Tämä toiminto on SQL tyyppisissä tietokannoissa. (Viite-eheyden varmistaminen tietokannoissa, N.D.) Huomasin toisessa projektissa, kun tietokannasta löytyi huomattava määrä vanhoja avaimia, että Amplifyssä tätä toimintoa ei ole ja tein myös hiukan tutkimusta ja Amplifyn Github:sta löytyi tämä toiminto oli “feature request” listalla jo useamman vuoden. Kun poistaa tietuetta, millä on riippuvuuksia, joutuu nämä riippuvuudet käymään läpi ja poistamaan erikseen.

```
const handleDelete = async () => {
    try {
      const translations = await amplifyClient.models.Translation.list({
        filter: { contentId: { eq: contentId } },
      });

      if (translations.data && translations.data.length > 0) {
        await Promise.all(
          translations.data.map((translation) =>
            amplifyClient.models.Translation.delete({
              id: translation.id,
```

```
          })
        )
      );
    }

    await amplifyClient.models.Content.delete({
      id: contentId,
    });
```

Kuvio 35. Kaskadipoisto Amplifyssä.

Kuviossa 36 on esimerkki miten Amplify:ssä, saadaan toteutettua kaikkien riippuvuuksien poisto oikein. Kun halutaan poistaa tietty käännösavain, on haettava ensin kaikki käännökset (Translation) jotka liittyvät sisältöön (Content) ja haku tapahtuu suodattamalla contentId:n perusteella. Kaikki käännökset poistetaan rinnakkain Promise.all -metodin avulla. Lopuksi poistetaan itse sisältö (Content) -tietue.

6 Pohdinta

6.1 Yhteenveto

Tämän kehittämistyön tavoitteena oli rakentaa käännöstyökalu, joka mahdollistaa käännöstiedostojen muokkaamisen ilman kehittäjän apua. Työn tuloksena syntynyt järjestelmä vastaa konkreettisesti asiakasprojekteissa esiin nousseisiin tarpeisiin, erityisesti silloin kun kielisisältöjä täytyy päivittää tai laajentaa jälkikäteen helposti ja nopeasti.

Kehitystyön aikana onnistuttiin luomaan käytännössä toimiva ratkaisu, joka on helposti käytettävä. Käytettävyyteen panostaminen jo alkuvaiheessa osoittautui tärkeäksi päätökseksi. Käyttöliittymä on selkeä ja helppokäyttöinen, mikä parantaa asiakaskokemusta huomattavasti.

Työssä kohdattiin kuitenkin myös haasteita. AWS Amplify, vaikka tarjosi nopean tavan julkaista palvelu, osoittautui monessa kohtaa rajoittuneeksi – erityisesti silloin, kun oli tarvetta muokata ratkaisuja monorepo-rakenteessa tai toteuttaa teknisesti poikkeavia ratkaisuja. Amplifyn tarjoama Cognito-autentikointi on edullinen, mutta rajoittunut käyttökokemukseltaan ja ominaisuuksiltaan, mikä aiheutti tarpeen kiertää sen rajoitteita omalla logiikalla. Cognitossa ei voi myös jälkikäteen muokata UserPoolin attribuutteja, kun ne on määritetty niin ne on määritetty, koko UserPool on poistettava ja tehtävä uusi jos haluaa muokata omia attribuutteja. On erikoista, että tämän kokoluokan yrityksellä on tällaisia ratkaisuja skaalautuvissa pilvipalveluissa.

Ratkaisu osoittautui käyttökelpoiseksi asiakasprojektiin. Kehityksen aikana selvisi, että dokumentaation tekeminen ja selkeä rakenne nopeuttaa ja helpottaa jatkokehitystä.

Kehitystyö tarjosi laajasti käytännön kokemusta moderneista teknologioista, kuten Next.js, TypeScript ja AWS Amplify. Projekti myös herätti ajatuksia ja kritiikkiä valmiita ratkaisuja kohtaa, kuten AWS Amplify. Vaikka teknologioiden käyttö toi esiin tiettyjä rajoituksia, niihin löydettiin ratkaisuja, jotka tukevat työn alkuperäisiä tavoitteita.

Järjestelmässä on potentiaalia laajentaa käyttöä tulevissa asiakasprojekteissa. Jatkokehityksen kannalta olisi hyödyllistä tutkia, miten saavutettavuutta, skaalautuvuutta ja kieliversioiden hallintaa voitaisiin parantaa entisestään. Lisäksi järjestelmän avaaminen myös muille kuin käännöksille ja olisi mahdollista laajentaa sen käyttöaluetta. Pohdinnassa oli myös oman backend-ratkaisun kehittäminen, koska Amplify sopii nopeaan prototyypitykseen, mutta jos on erilaisia tarpeita tai monimutkaisia vaatimuksia, niin Amplify vain hidastaa ja vaikeuttaa tekemistä.

6.2 Jatkokehitys

Kehittämistyötä lähdetään viemään seuraavin askelin. Tarkoituksena on irrottaa Amplify kokonaan pois, koska se on tosiaan liian rajoittunut. Kunnollista REST apia ei Amplify:n kautta saanut tehtyä, että saisi luotua erillisen järjestelmän erillisille api-avaimille. Täytyy tehdä oma backend-ratkaisu esim. serverless lambda funktioilla. Kun projektiin on kehitetty oma backend-ratkaisu, se siirretään monorepoon.

Tällä hetkellä verkkosovellus ei tallenna muuta kuin normaalia tekstiä ja ei sisällä WYSIWYG-editoria. Jatkokehityksessä lisätään WYSIWYG-editori ja tuetaan myös html elementtejä mitä WYSIWYG-editorista on sisällytetty, mikä parantaisi asiakkaan muokkausmahdollisuutta. Tarkoituksena on myös kehittää työkalusta enemmän CMS-tyyppinen npm-paketti, mikä saadaan integroitua helposti projektiin.

Lähteet

Amplify Concepts. N.D. Viitattu 16.3.2025. https://docs.amplify.aws/react/how-amplify-works/concepts/

Benefits and challenges of monorepo development practises. N.d. Circleci. Viitattu 5.10.2024. https://circleci.com/blog/monorepo-dev-practices/

Communications of the ACM, 59(7), 78–87. Viitattu 2.2.2025 https://dl.acm.org/doi/10.1145/2854146

Design Tools survey, 2021. Viitattu 26.2.2025. https://uxtools.co/survey/2021/

Du Mortier, G. 2022. Best Practices for Multi-Language Database Design. Vertablo. Viitattu 26.2.2025. https://vertabelo.com/blog/multi-language-database-design/

Garrett, J. J. (2011). *The Elements of User Experience: User-Centered Design for the Web and Beyond*. New Riders.

Get started with Tailwind CSS, N.D. Viitattu 30.3.2025. https://tailwindcss.com/docs/

Hamori, F. The History of React.js on a Timeline. 2024. Risingstack. Viitattu 5.10.2024. https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/

Holst, C. N.D. The Best Practices and Key Principles of UX Design. Baymard. Viitattu 24.2.2025 https://baymard.com/learn/ux-design-principles

Florian, M. 2024. How Shadcn Cut Through the Noise and Became React's Default Component Library. Viitattu 1.4.2025. https://blog.api-fiddle.com/posts/shadcn-for-react

JSON-Locales. N.D. Viitattu 30.3.2025. https://richardevcom.github.io/JSON-Locales/

Mikä on AWS-pilvipalvelu? N.D. Viitattu 16.3.2025. https://petrosoft.fi/blogs/uutiset/mika-on-aws-pilvipalvelu

PNPM Motivation. N.D. Viitattu 16.3.2025. https://pnpm.io/motivation

Potvin, R., & Levenberg, J. (2016). *Why Google stores billions of lines of code in a single repository*.

Route Groups. N.D. Viitattu 16.3.2025. https://nextjs.org/docs/app/building-your-application/routing/route-groups

Server-Side Rendering. N.d. Viitattu 5.10.2024. https://www.sanity.io/glossary/server-side-rendering

Shadcn. N.D. Viitattu 16.3.2023. https://ui.shadcn.com/docs/components/input

TanStack Query Overview. N.D. Viitattu 16.3.2025. https://tanstack.com/query/latest/docs/framework/react/overview

Toikko, T. & Rantanen, T. (2009). *Tutkimuksellinen kehittämistoiminta: Näkökulmia kehittämistyön tutkimuksellisuuteen ja tutkivan työotteen soveltamiseen.* Tampere University Press. Viitattu 27.2.2025. https://trepo.tuni.fi/bitstream/handle/10024/100802/Toikko_Rantanen_Tutkimuksellinen_kehittamistoiminta.pdf?sequence=1&isAllowed=y

Turborepo Introduction. N.d. Viitattu 5.10.2024. https://turbo.build/repo/docs

TypeScript for the New Programmer. N.d. Typescript dokumentaatio. Viitattu 5.10.2024. https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html

Viite-eheyden varmistaminen tietokannoissa. N.D. Viitattu 1.4.2025 https://www.vpnunlimited.com/fi/help/cybersecurity/referential-integrity

Wieruch, R. 2022. The Road to React. Leanpub. Viitattu 5.10.2024.

Liitteet