Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Jere Moilanen

Cocoa-ohjelmistokehitys

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

14.04.2020

Tekijä Otsikko Sivumäärä Aika	Jere Moilanen Cocoa-ohjelmistokehitys 23 sivua 14.04.2020
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Toimitusjohtaja Ville Ranta Yliopettaja Auvo Häkkinen

Tässä insinöörityössä suunniteltiin ja valmistettiin Trafore Oy:lle iOS-mobiilisovellus, jonka tehtävänä oli sallia käyttäjien syöttää työaikatietoja Exim-järjestelmään. Projekti toteutettiin kevään ja kesän 2019 aikana. Tässä raportissa kerrotaan työn taustoista ja perustellaan syitä teknologia- sekä alustavalinnoille.

Insinöörityö on toteutettu Swift-ohjelmointikielellä käyttäen Cocoa- sekä Cocoa Touch-rajapintoja. Sovelluksen on kehitetty kyseisten rajapintojen kanssa suositellun Model-View-Controller-ohjelmistoarkkitehtuurin mukaan. Työskentely-ympäristönä toimi Xcode-kehitysympäristö.

Lopputuloksena syntyi toimiva ja käytännöllinen mobiilisovellus, joka täyttää sen ensisijaiset vaatimukset. Sovellus on saatavilla App Storessa, ja se on käytössä osalla Trafore Oy:n asiakkaista.

Avainsanat	iOS, Cocoa, Cocoa Touch, Xcode

Author Title Number of Pages Date	Jere Moilanen Cocoa Software Development 23 pages 14 April 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Ville Ranta, CEO Auvo Häkkinen, Principal lecturer

For this thesis, an iOS application was designed and built for Trafore Oy, with a goal of providing a way for the customers of the company to input working hours into its preexisting Exim software system. The project was executed during the spring and summer of 2019.

The study discusses the business requirements and foundation of the project and introduces the Cocoa and Cocoa Touch frameworks and their features. Furthermore, the architectural components of the project are explained, as well as the challenges that came up during the development.

The software was developed with Xcode using Cocoa Touch and Swift, embedded in a Model-View-Controller software architecture. With the help of colleagues from the company the software was successfully executed and is being distributed via App Store to customers of Trafore Oy.

Keywords	iOS, Cocoa, Cocoa Touch, Xcode

Sisällys

Lyhenteet

IDE	Integrated Development Environment. Ohjelmointiympäristö. Työpöytäsovellus, jolla kehitetään ja ajetaan koodia.
JSON	JavaScript Object Notation. Avoimen standardin tiedostomuoto, jota käytetään tiedonvälitykseen.
MVC	Model-View-Controller. Ohjelmistoarkkitehtuuri, jossa ohjelman eri komponentteja pyritään eristämään toisistaan, jotta saavutettaisiin parempi koodin uudelleenkäyttöaste.
OOP	Object-oriented programming. Olio-ohjelmointi. Ohjelmointiparadigma, jossa ohjelmistot suunnitellaan lähtökohtaisesti olioiden kanssakäymisen pohjalta.
XML	Extensible Markup Language. Merkintäkieli, jolla kuvataan tekstin rakennetta.

1. Johdanto

Insinöörityö tehtiin osana Trafore Oy:n Exim-tuotteen kehitystyötä. Sen ensisijaisena tavoitteena oli tarjota Eximin käyttäjille tapa kirjata työtunteja järjestelmään mobiilisovelluksella.

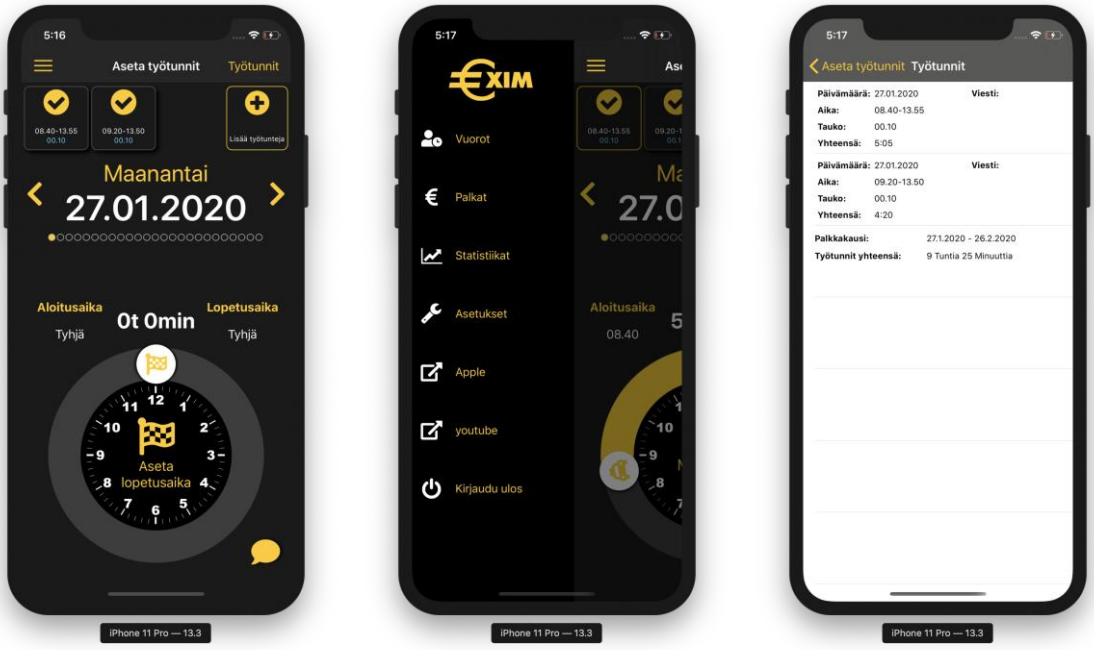
Valitsimme opinnäytetyön kehitykseen natiivin ohjelmointiympäristön aiemman kehityskokemukseni perusteella. Koen itse, että natiivit sovellukset ovat riippumattomampia kolmansista osapuolista, paremmin suoriutuvia, vakaampia, turvallisempia sekä pitkäikäisempiä kuin niiden hybridi vastineet. Natiiveja ohjelmointikieliä ja kehitysympäristöjä kehittävät laitevalmistajat itse. Tämä takaa niiden olemassaololle pitkän iän. Applen App Storea voidaan pitää ensimmäisenä valtavirtaan nousseena mobiilisovelluskauppana. App Store julkaistiin päivää ennen iPhone 3G:n julkaisua heinäkuussa 2008. Julkaisupäivänä App Storessa oli 500 sovellusta. 10 vuotta myöhemmin Apple ilmoitti WWDC konferenssissaan, että App Storen sovellusten määrä ylitti 3,5 miljoonaa ja latausten määrä on yli 130 miljardia. Mobiilisovellusten ulottuvuus on äärimmäisen korkea, sillä lähes puolet maapallon väestöstä omistaa älypuhelimen. Näistä noin 27 % on iOS-laitteita. [1.] Vaikka iOS-ympäristö on käyttäjäkunnaltaan pienempi, iOS-sovellukset tuottavat noin. 80 % enemmän rahaa kuin Android-sovellukset. [2.] Tämä tekee iOS-ympäristöstä houkuttelevan sovelluskehittäjille.

Tässä raportissa keskityn olennaisesti Cocoa- sekä Cocoa Touch -ohjelmointirajapintojen osa-alueisiin, joita käytin insinöörityössäni. Toisessa luvussa kerron, mitä työni tarjoaa asiakkaille ja Trafore Oy:lle. Seuraavissa luvuissa kerron Cocoa-ohjelmointirajapinnan tarjoamista teknologioista ja sen jälkeen Cocoa Touch-ohjelmointirajapinnasta, joka suurilta osin pohjautuu macOS-ympäristössä käytettyyn Cocoaan. Lopuksi selostan vielä sovellukseni eri osa-alueista.

2. Työni Trafore Oy:llä

Trafore valmistaa Exim-nimistä tuoteperhettä, jonka päämääräinen tarkoitus on automatisoida kuljetusalan yrittäjien palkanmaksua sekä kirjanpitoa. Exim on integroitu osaan taksamittareista, jolloin Traforen palvelimille tulee suoraan tiedot ajovuorojen loppu- ja alkuajankohdista sekä kuljettajien ajamista ajovuoroista. Osa kuljetusalan yrityksistä tarvitsee kuitenkin myös manuaalisen tavan kirjata kuljettajien työtunteja palveluun, sillä esimerkiksi joillain bussiyhtiöillä kuljettajat työskentelevät kiinteillä työtunteilla ja kiinteillä tuntipalkoilla. Exim-verkkosovelluksessa tämä palvelu oli jo tarjottu, mutta työtuntien kirjaamiseen mobiililaitteilla tarvittiin ratkaisu. Vaikka Exim-verkkosovellus oli tehty skaalautuvaksi mobiililaitteille, päätimme hyödyntää aiempaa natiivi iOS- ja Android-kehityskokemustani luomalla molemmille alustalle omat mobiilisovellukset, jotka ovat yhteydessä Exim-järjestelmään. Mobiilisovellusten avulla kuljettajat pystyvät kirjaamaan työtuntinsa helpommalla ja visuaalisemmalla tavalla.

Kuvassa 1 on kuvankaappaus tekemästäni iOS-sovelluksesta, jossa havainnollistetaan sen erilaisia näkymiä:



Kuva 1. Exim-sovelluksen näkymiä

Oikealla on päänäkymä, jossa työtunnit kirjataan pyörittämällä valkoista nappulaa kellotaulun ympärillä. Keskimmäisessä näkymässä on menuvalikko, jonka painikkeet ohjaavat Eximin web-palveluihin. Kirjautumisnappulan yläpuolella on kaksi käyttäjän vapaavalintaista linkkiä, joita voi hyödyntää vaikkapa asettamalla niihin käyttäjän työnantajayrityksen sisäisten palveluiden linkkejä. Vasemmalla lista merkityistä työtunneista.

3. Cocoa-ohjelmointirajapinta

Cocoa on Applen kehittämä olio-ohjelmointiin perustuva ohjelmointirajapinta, jota käytetään lähinnä macOS-ympäristössä. Cocoa rakentuu kolmesta Objective-C-kirjastosta, Foundation Kit:istä, Application Kit:istä sekä Core Datasta. Cocoa on kehitetty käytettäväksi MVC-arkkitehtuurissa. MVC- eli Model-View-Controller on ohjelmistoarkkitehtuuri, joka pyrkii lisäämään koodin uudelleenkäytettävyyttä. MVC-mallissa sovelluksen olioille määritetään yksi kolmesta roolista: Model (Malli), View (Näkymä) tai Controller (Ohjain). Cocoa-ohjelmointirajapinta tarjoaa NSController-luokkia, joiden tehtävänä on suorittaa Ohjain-kerrokselle tyypillisiä toimintoja, kun taas Application Kit tarjoaa Näkymä-kerrokselle tyypillisiä toimintoja ja Core Dataa käytetään Malli-kerroksessa.

Cocoa-ohjelmistoja kehitetään tyypillisesti Xcodella. Objective-C oli Cocoa- sekä Cocoa Touch -ohjelmointirajapintojen ensisijainen ohjelmointikieli vuodesta 1984 vuoteen 2014, jonka jälkeen se korjattiin Swiftillä. Objective-C on C-ohjelmointikielen yläjoukko, jonka tarkoitus on tarjota oliokeskeisiä ominaisuuksia. Objective-C peri C:n syntaksin, alkeelliset tyypit ja lisäsi syntaksin luokkien ja metodien määrittelemiseen. [3.]

3.1. Foundation Kit -perustyökalut

Foundation Kit on ohjelmistokehys, joka tarjoaa monia perustason työkaluja sekä käyttöjärjestelmän palveluita.

Esimerkkeinä mainittakoon:

- primitiiviset arvot sekä datatyypit
- merkkijonot (String) sekä niiden käsittelyyn tarvittavat toiminnallisuudet
- taulukot (Array), sanakirjat (Dictionary) sekä muut olioiden säilytykseen ja iterointiin tarkoitetut tietorakenteet

- datan suodattamiseen ja järjestelyyn tarkoitettuja mekanismeja
- userPreferences, sovellusten paikalliseen konfigurointiin käytetty tiedon tallennusjärjestelmä
- iCloud-toiminnallisuudet, tiedostojen hallintaan, tallennukseen, sekä saman iCloud-tilin eri laitteiden välisten asetusten hallintaan tarkoitettu työkalu
- verkkotoiminnallisuudet, URL Loading System, työkalu palvelinten kanssa kommunikointiin käyttäen standardisoituja internetprotokollia
- säikeistystyökalut, joilla voi hallita ja luoda säikeitä sekä tehtäväjonoja.

Foundation Kit on integroitu osaksi Swift-ohjelmointikieltä. [4.]

3.2. Application Kit -käyttöliittymätyökalut

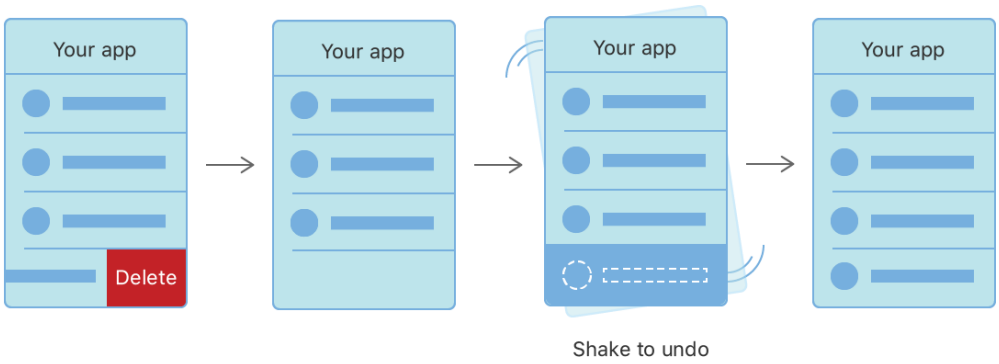
Application Kit, lyhyemmin AppKit, on kokoelma macOS-sovellusten graafisten käyttöliittymien luontiin tarkoitettuja työkaluja. Muutamia esimerkkejä ovat:

- ikkunoiden koon ja sijainnin hallinta
- UI:n väriteemojen hallinta
- animaatiot
- äänet, haptiset syöttölaitteet
- hiiren, näppäimistön ja Trackpad-laitteiden tapahtumien hallinta
- kuvien, bittikarttojen ja muiden formaattien luonti.

3.3. Core Data -tietokantaliittymä

Core data on iOS -ja macOS-ympäristöissä käytettävä Malli-kerroksen olioiden kanssakäymisten kuvaukseen ja tallentamiseen käytettävä ohjelmistokehys. Sen tehtävä on siis muuttaa olioita tietokantaan talletettaviksi tauluiksi ja riveiksi. Core data on suora rajapinta SQLite-tietokantajärjestelmälle, jonka päälle iOS- ja macOS-järjestelmien tiedon tallennus rakentuu. Se abstrahoi kehittäjältä tietokannan kanssakäymiseen vaadittavat toiminnat tarjoamalla helppokäyttöisiä Swfit- tai Objective-C-komentoja. [5.]

Core data mahdollistaa hyödyllisiä toimintoja kuten kumoa/toista-toiminnallisuuden, JSON-objektien jäsennystä taustasäikeillä, tiedon tallentamista välimuistiin sekä datamallin versioimista. Kuvassa 2 on visualisaatio iOS-käyttöjärjestelmän kumoa toiminnosta, joka on toteutettu Core Datalla:

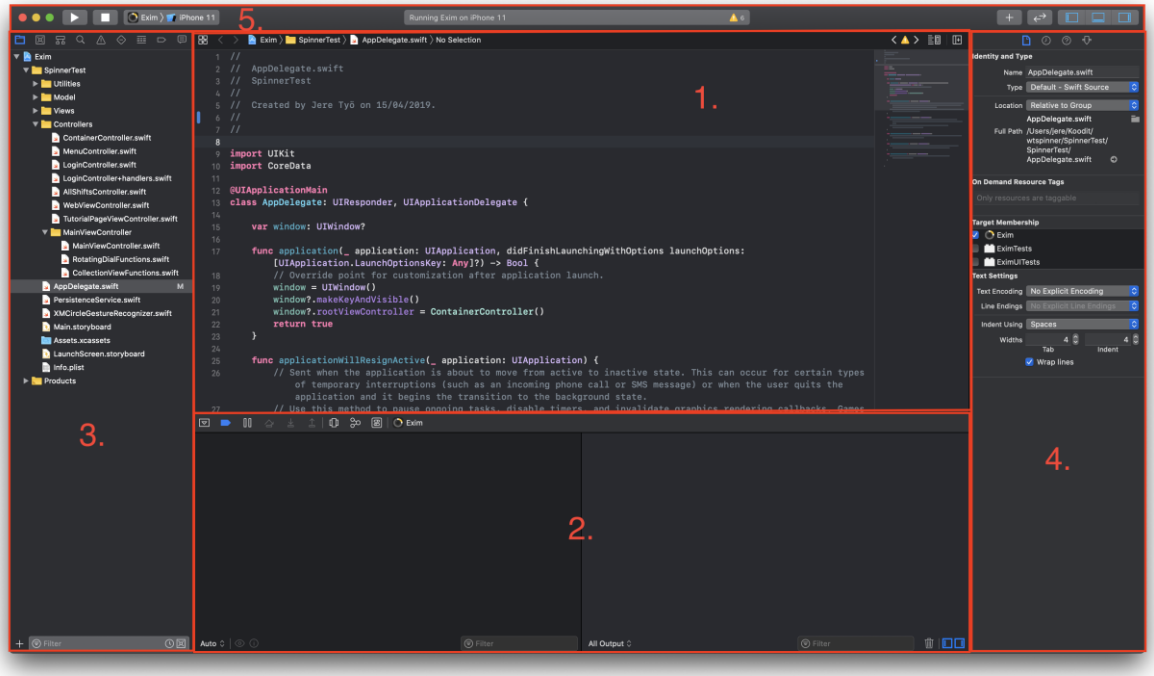


Kuva 2. Kumoa toiminto

Kuvassa 2, Core data pitää välimuistissa UITableViewn datalähteen oliot, ja ravistusele on kytketty palauttamaan olio datalähteeseen. [5.]

3.4. Xcode-ohjelmointiympäristö

Xcode on Applen kehittämä ohjelmointiympäristö (IDE), johon Cocoa sekä Cocoa Touch on integroitu. Cocoa-ohjelmistoja tyypillisesti kehitetään Xcodella. Xcode on saatavillailmaiseksi kaikille macOS-käyttäjille. Kuvassa 3 on havainnollistettu Xcoden käyttöliittymän eri osa-alueita: [6]



Kuva 3. Xcode-IDE:n käyttöliittymän osa-alueita: 1. Editori 2. Debug-alue 3. Navigator-alue 4. Utilities alue

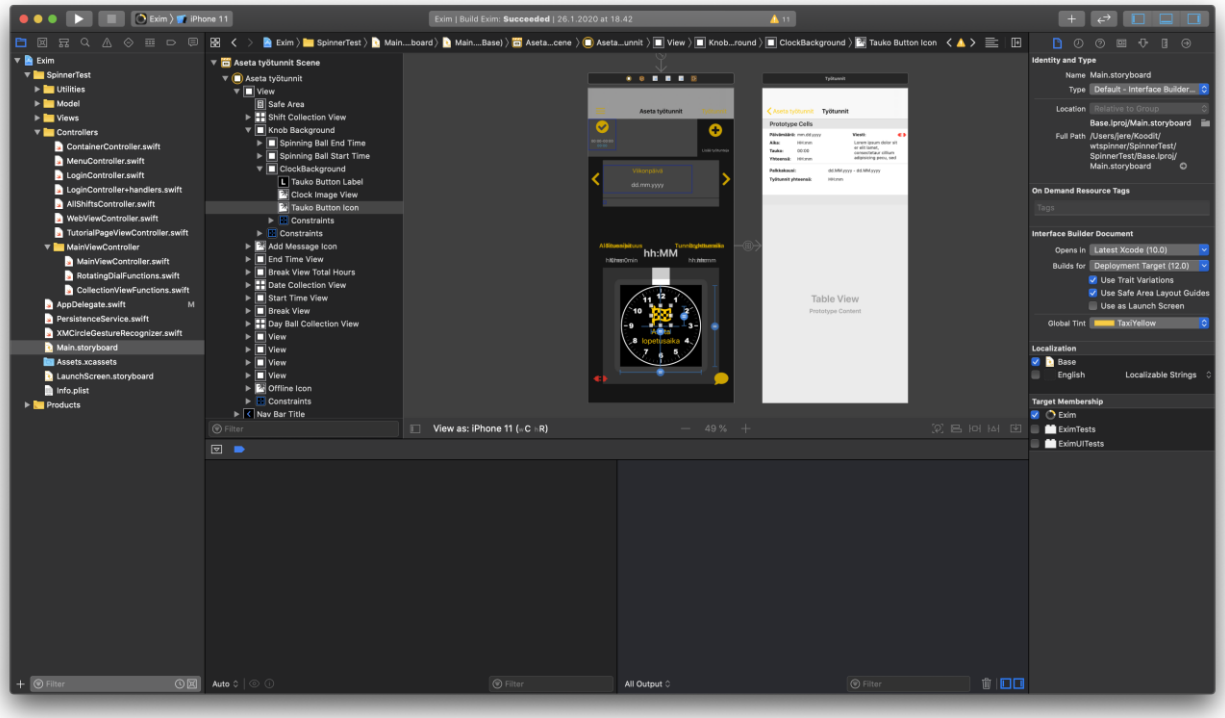
Editor-näkymässä (1) tapahtuu lähdekoodin muokkaus. Myös .storyboard-tiedostojen editointi Interface Builderilla sekä eräät graafiset virheiden etsintätyökalut näkyvät editor-näkymässä. Debug-alueella (2) näkyy oikeassa lohkossa konsoli sekä vasemmassa lohkossa muuttujanäkymä. Utilities-alue (4) on mukautuva alue, joka näyttää erilaisia työkaluja riippuen siitä, mikä objekti on käyttäjällä valittuna.

Navigator-näkymässä (3) on useita välilehtiä [7]:

-  **Project navigator.** Projektin kansiorakenteen sekä tiedostojen luontiin ja muokkaukseen.
-  **Symbol navigator.** Projektin symbolit. Lista tai hierarkianäkymässä.
-  **Find navigator.** Etsimistyökalu.
-  **Issue navigator.** Projektin diagnostiikkanäkymä. Tässä näkyvät varoitukset ja virheet, jotka ohjelmiston suorittamisessa ilmenevät.
-  **Test navigator.** Yksikkötestien luontiin, hallintaan, ajamiseen ja tarkkailuun.
-  **Debug navigator.** Ohjelman säikeisiin sekä pinoihin liittyvä informaatio.
-  **Breakpoint navigator.** Pysäytyspisteiden hienosäätämiseen. Xcodessa on mahdollista esimerkiksi määritellä pysäytyspisteille ehtoja, jotka sen täytyy täyttää lauetessaan.
-  **Report navigator.** Projektin koonnin, ajamisen virheiden raportoinnin sekä historian tarkasteluun.

3.5. Käyttöliittymän suunnittelutyökalut

Xcode:lla on mahdollista luoda käyttöliittymiä graafisesti käyttäen Interface Builderia tai koodilla luomalla olioista instanssit manuaalisesti. Käyttöliittymätyökalun UI-tiedostot loppuvat päätteeseen *.storyboard* tai *.xib.* Xib-tiedostolla määritellään yleensä yhden UI-Kit-elementin tai näkymäohjaimen (View Controller). Storyboard määrittelee usean näkymäohjaimen ja niiden siirtymät. Xib- ja .storyboard-tiedostot ovat Xcoden sisällä säilötty XML-formaatissa, mutta koonnissa Xcode luo niistä *.nib*-binääritiedostoja. [8] Kuvassa 4 näkyy työn käyttöliittymätyökalulla luotuja .storyboard-tiedostoja:



Kuva 4. Xcode Interface Builder. Editor-näkymässä näkyy insinöörityön storyboard-tiedosto.

Käyttöliittymätyökalussa olioita lisätään näkymään drag & drop -periaatteella. Oliot yhdistetään koodiin joko Toiminnoilla (Action) tai Ulostulopisteellä (Outlet). Toimintoyhteys luodaan, jos halutaan lähettää käyttöliittymäobjektista viestin koodiin. Ulostulopisteellä käyttöliittymäobjektiin voi viitata koodista.

Jotta käyttöliittymätyökalulla luodut näkymät skaalautuisivat kaikenkokoisiin näyttöihin, voidaan käyttää Auto Layouteja. Tämä tehdään luomalla olioiden välille rajoitteita, jotka

määrittelevät, mitkä ovat olioiden keskeiset mittasuhteet. Auto Layout myös varmistaa, että ohjelmisto skaalautuu oikein, kun laite käännetään horisontaaliseen asentoon.

4. Cocoa Touch -ohjelmointirajapinta

Cocoa Touch on iOS-kehitykseen tarkoitettu ohjelmointirajapinta. Se jakaa monia seikkoja Cocoa-rajapinnan kanssa, kuten Objective-C-ajoympäristön sekä Foundation-ohjelmistokehyksen. Cocoa Touchilla on myös sille ominainen käyttöliittymien kehitykseen tarkoitettu UI-kehys. UIKit on hyvin samanlainen sen Cocoasta tutun vastapuolen, AppKitin kanssa, mutta UIKit on luotu vastaamaan kosketukseen perustuvan mobiiliympäristön tarpeisiin.

UIKit on Cocoa Touchin tarjoama käyttöliittymien luontiin tarkoitettu ohjelmistokehys. UIKit tarjoaa myös kosketusnäytöillä toimivien kosketustapahtumien käsittelijöitä, jotka tunnistavat painalluksia, pyyhkäisyjä, pyörityksiä sekä pitkiä painalluksia.

Insinöörityössä tärkeimmässä roolissa olivat UIKit-luokat UIViewController, UINavigationController, UICollectionView ja UITableView.

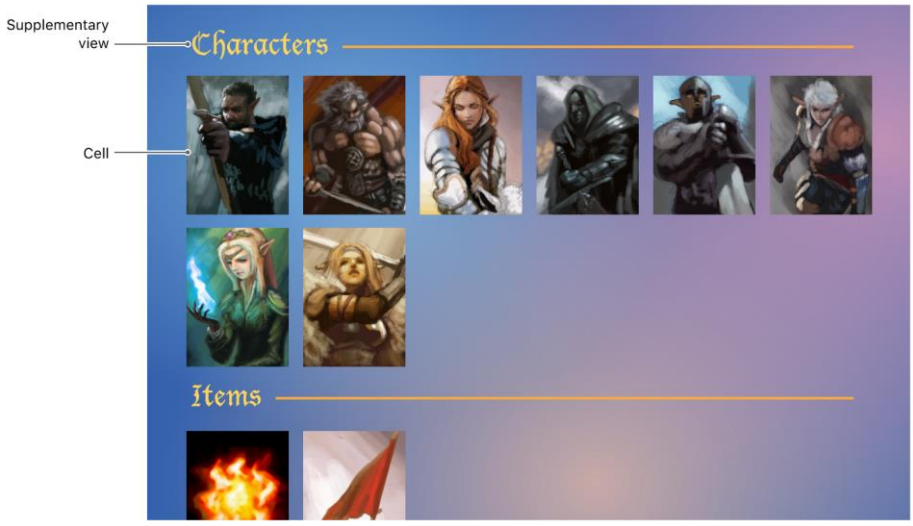
UIViewController on olio, joka hallitsee sovelluksen näkymähierarkiaa. UIViewControllerin tehtäviä ovat muun muassa

- päivittää sen sisältämiä näkymiä
- reagoida käyttäjän syötteisiin
- hallita sen sisältämien näkymien mittasuhteita.

UINavigationController on säiliö, joka sisältää UIVIewController-olioita. NavigationController hallitsee sen sisältämiä ViewControllereita navigaatiopinossa. Navigaatiopinoon työnnetään ja poistetaan elementtejä käyttäjän navigoidessa sovelluksessa. Navigaatiopinon voi nähdä korttipakkana, jonka ylin kortti on sovelluksessa näkyvä näkymä. Kun käyttäjä haluaa siirtyä seuraavaan näkymään, hän painaa painiketta, joka on linkitetty johonkin toiseen “korttiin” eli ViewControlleriin. UINavigationController pitää muistissa viittauksia edellisiin näytettyihin ViewControllereihin, joka halutessa mahdollistaa automaattisesti toimivan “taaksepäin”-painikkeen luonnin. Vain yksi ViewController voi olla

kerrallaan näkyvissä navigaatiopinosta, ellei kyseinen ViewController itse ole ViewController-säiliö.

UICollectionView on olio, joka näyttää dataa järjestellyissä soluissa (Cell). UICollectionView-oliota voi vierittää horisontaalisesti tai vertikaalisesti ja sen sijoittelua sekä sen soluja voidaan kustomoida. Kuvassa 5 näkyy eräs UICollectionView'n implementaatio:



Kuva 5. UICollectionView-esimerkkinäkymä [9]

Kuvassa 5 näkyvät solut eivät ole tietoisia datasta, joita ne näyttävät, vaan niiden luokkatiedostoissa on määritelty pelkästään niiden rakenne. UICollectionView hallitsee, mitkä solut näkyvät milläkin hetkellä ja mitä ne sisältävät. UICollectionView kierrättää sen sisältämiä soluja. Kun käyttäjä vierittää näkymää eteenpäin, joitakin soluja poistuu näytöltä ja uusia ilmaantuu. Sen sijaan että UICollectionView loisi jatkuvasti uusia soluinstansseja, se kierrättää jo luotuja instansseja ja korvaa niiden sisällön uudella.

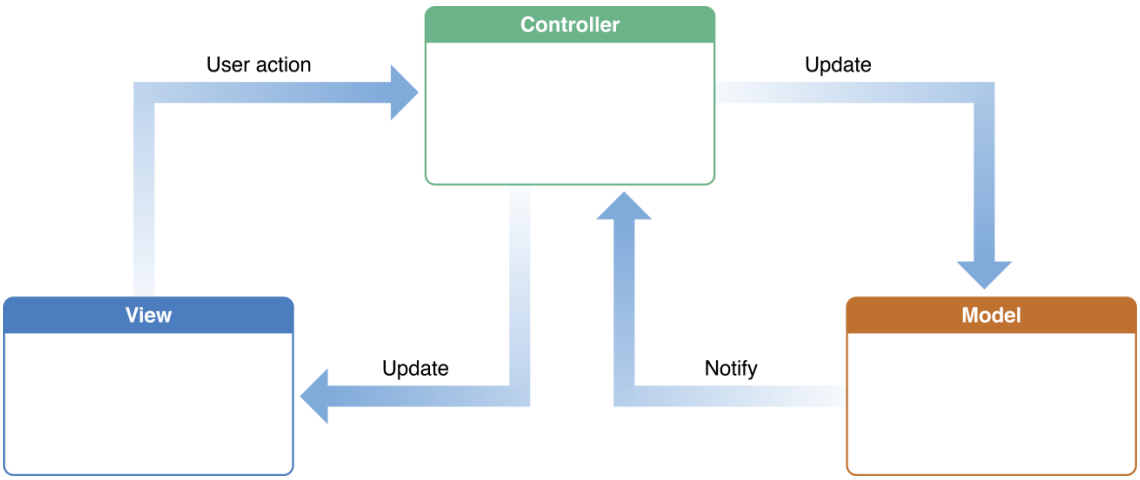
UITableView on olio, joka näyttää dataa vertikaalisen rullattavan listan muodossa. UITableView'n soluihin pätevät samat säännöt kuin aikaisemmassa kappaleessa mainitussa UICollectionView'ssä. Näkymät ovat keskenään hyvin samankaltaisia, mutta UITableView on vähemmän kustomoitaviassa sen solujen järjestelyn kannalta. Poiketen UICollectionViewstä, UITableView'n on myös integroitu mekanismeja solujen poistoon ja editoimiseen esimerkiksi pyyhkäisemällä solua vasemmalle. Kuvassa 6 on insinöörityön näkymä, joka näyttää kaikki käyttäjän syöttämät työvuorot valitun palkkakauden ajalta.

Kuva 6. Sovelluksen ShiftTableView-näkymä. Se listaa käyttäjän luomia työvuoroja käyttäen UITableView-oliota.

5. Projektin toteutus

Insinöörityö on kehitetty Swift ohjelmointikielellä. Swift on Applen kehittämä avoimen lähdekoodin ohjelmointikieli, joka korvasi Objective-C:n Applen ensisijaisena kehityskielenä vuonna 2014.

Sovellus on luotu MVC-arkkitehtuurilla. MVC on ohjelmistoarkkitehtuuri, jossa sovelluksen olioille määritetään yksi kolmesta roolista: Model (Malli), View (Näkymä) tai Controller (Ohjain). MVC myös määrittää, miten oliot keskustelevat toistensa kanssa. Tämä mahdollistaa koodin kapseloinnin siten, että mikään yksittäinen komponentti ei ole riippuvainen toisesta ja esimerkiksi ohjelmiston käyttöliittymä voidaan vaihtaa muuttamatta ohjainta tai mallia. Yksi MVC-mallin (kuva 7) avaintekijöistä on se, että View ja Model eivät koskaan kommunikoi keskenään. Kaikki kommunikaatio tapahtuu ohjaimen välityksellä, joka mahdollistaa eri View- ja Model-kerrosten löysän liitännäisyyden toisiinsa, jolloin ne ovat hyvin modulaarisia ja helposti uudelleenkäytettävissä ja korvattavissa. [10.]



Kuva 7. MVC-arkkitehtuurin toimintaperiaate [10]

Malli (Model) sisältää ohjelmiston tietomallit ja datan käsittelyyn liittyvän logiikan. Malli ottaa käskyjä vastaan ohjaimelta (Controller) ja palauttaa dataa. Käyttäjä lähettää näkymän (View) kautta komentoja tai syötteitä, esimerkiksi näppäinkomentoja tai tekstiä. Näkymä välittää ohjaimelle tiedon saadusta syötteestä. Ohjain ottaa komennot vastaan ja

välittää ne mallille. Kun malli vastaa, ohjain kertoo näkymälle, miten data näytetään käyttäjälle. Seuraavissa luvuissa on tiivistelmä siitä, miten työssä toteutettu sovellus on jaoteltu käyttämään MCV-mallia:

5.1. Model

Sovelluksen Model-lohko on jaettu kolmeen ensisijaiseen luokkaan. Ne ovat:

- PaymentPeriod
- WorkDay
- Shift.

PaymentPeriod eli palkkakausi kuvastaa työnantajan määräämää ajanjaksoa, jonka ajalta palkka lasketaan. Tämä voi olla vaikkapa kuukausi tai kaksi viikkoa. Palkkakausi määrittää, kuinka usein asiakkaan kirjanpidolle lähetetään kooste palkoista, ja myös sovelluksen perusnäkymässä olevan työjakson pituuden.

Palkkakausi sisältää WorkDay, eli työpäivä-olioita. Työpäivä on kääre Shift- eli vuoro-olioille, jotka sisältävät datan työntekijän ajovuorosta. Yksi työpäiväolio voi sisältää yhden tai useita vuoro-olioita.

Työntekijän työsopimuksesta riippuen yksi vuoro-olio voi käsittää koko työpäivän, tai vaikkapa yhden yksittäisen ajovuoron. Vuoro-oliot on mallinnettu .xcdatamodeld-tiedostoon, joka mahdollistaa niiden tallennuksen levylle käyttäen View- ja Model- kerrosten. .xcdatamodeld-tiedostossa määritellään dataolioiden tyypit ja välisuhteet. Levylle tallennuksella sovelluksessa pyritään mahdollistamaan työvuorojen tallennus yhteydettömässä tilassa. Jos työvuoron lähetys epäonnistuu, se tallennetaan levylle. Kun sovellus seuraavan kerran käynnistetään, se tarkastaa, löytyykö levyltä lähettämättömiä työvuoroja ja lähettää ne palvelimelle.

5.2. View

View-lohko sisältää erilaisten UI-luokkien implementaatioita. Niitä ovat:

- ShiftCollectionView
- DateCollectionView
- DayBallCollectionView
- ShiftTableView
- ShiftTableViewCell
- ShiftCollectionViewCell
- MenuOptionCell
- PresentedDayCell
- AddNewShiftCell
- SummaryCell
- DayBallCell.

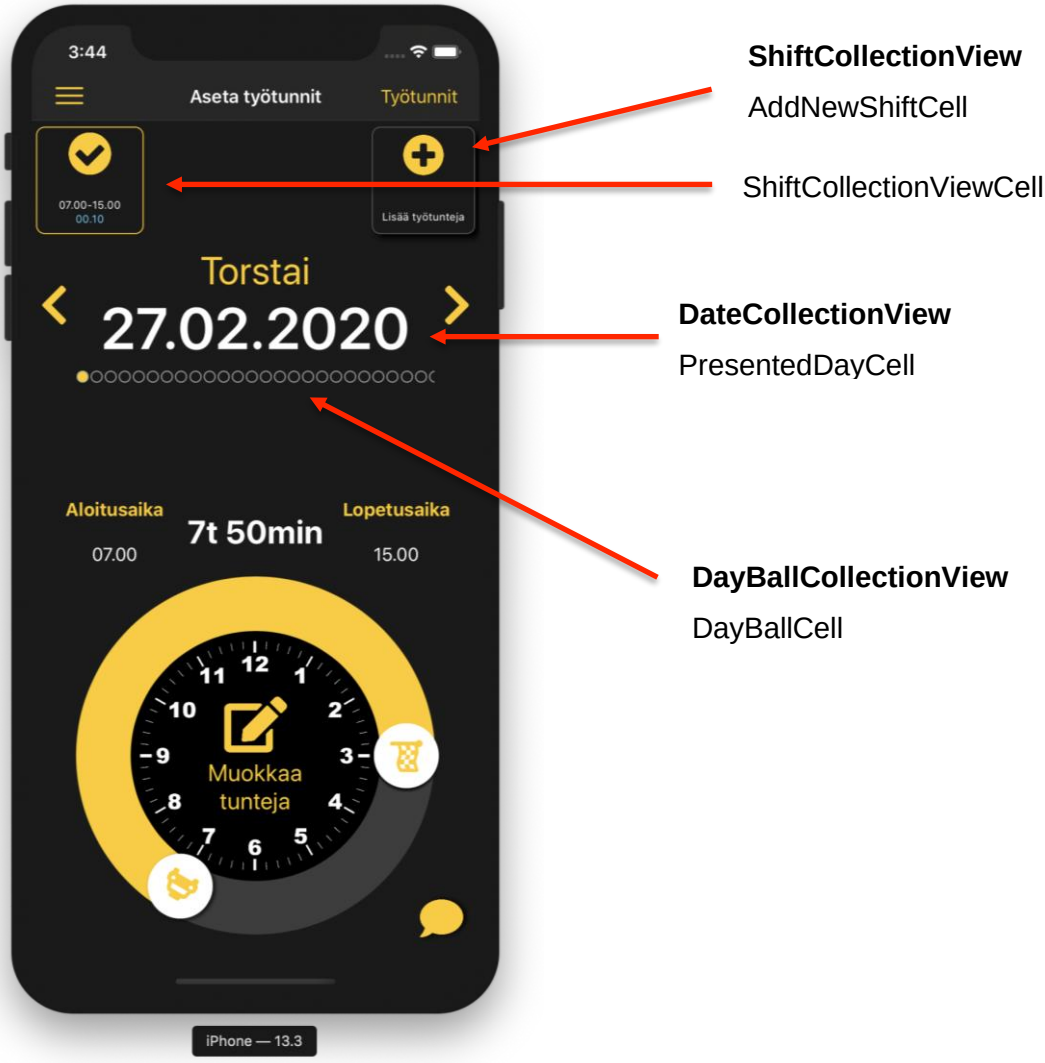
Sovelluksen datan näyttämiseen käytetään suurimmaksi osaksi UIKitin tarjoamia UITableView- sekä UICollectionView-luokkia, joiden implementaatioita ovat ShiftCollectionView, DateCollectionView, DayBallCollectionView sekä ShiftTableView.

ShiftCollectionView näyttää soluissaan kaikki valitulle päivälle tallennetut työvuorot.

DateCollectionView näyttää käyttäjän työnantajan valitseman palkkakauden pituisen ajanjakson päivinä, joita voi selata vierittämällä horisontaalisesti.

DayBallCollectionView on eräänlainen indikaattori, jolla näkee. missä kohtaa palkkakautta vieritys on tällä hetkellä ja mille päiville työtunteja on kirjattu.

ShiftTableView sijaitsee työtunnit painikkeen takana ja näyttää kaikki kirjatut työtunnit riveittäin, ajankohdan mukaan lajiteltuna. Kuvassa 8 näkyy yllä mainittujen luokkien implementaatioita.



Kuva 8. Sovelluksen MainViewController-näkymä, jossa näkyy edellisessä luvussa mainittuja UIKit-luokkia

Kun edellämainituista luokista luodaan CollectionView-olioita, Cell-päätteisten luokkien instanssit ovat puolestaan olioita, joita CollectionView-oliot hallitsevat. Cell-objektit ovat siis säiliöitä kunkin CollectionViewn näyttämälle datalle. Kuvan 8 kuvasta huomaa miten erilaisiin käyttötarkoituksiin ja olomuotoihin CollectionView-soluja voi käyttää.

Suurin osa toteutuksista, kuten esimerkiksi ShiftTableView-olion ShiftTableViewCell-luokka on luotu käyttöliittymätyökalulla (Interface Builder), joka mahdollistaa hyvin vähäisen tyylimäärittelyyn ja alustukseen tarkoitetun koodin kirjoituksen. Ohessa on luokka kokonaisuudessaan:

```
import UIKit

class ShiftCollectionViewCell: UICollectionViewCell {

    @IBOutlet weak var checkBox: UIView!

    @IBOutlet weak var dayLabel: UILabel!

    @IBOutlet weak var clockIcon: UIImageView!

    @IBOutlet weak var breakLabel: UILabel!

}
```

Luokan erilaiset tyylimäärittelyt, rajoitteet ja näkymähierarkian määrittely on tehty Xcoden käyttöliittymätyökalulla, joka generoi ne XML-muodossa projektin .storyboard-tiedostoon. Käyttöliittymätyökalun käyttöä usein perustellaan sen alustuskoodia vähentävällä vaikutuksella.

Ohessa on ote luokasta MenuOptionCell, joka on ohjelmoitu täysin manuaalisesti. Huomaa määrittelyyn vaadittavan koodin määrä:

```
import UIKit

class MenuOptionCell: UITableViewCell {

    // MARK: - Properties

    let iconImageView: UIImageView = {
        let iv = UIImageView()
        iv.contentMode = .scaleAspectFit
        iv.clipsToBounds = true
        iv.tintColor = UIColor.white
        return iv
    }()
```

```
    let descriptionLabel: UILabel = {
        let label = UILabel()
        label.textColor = .white
        label.font = UIFont.systemFont(ofSize: 16)
        return label
    }()

    // MARK: - Init

    override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        self.backgroundColor = UIColor.black

        addSubview(iconImageView)
        iconImageView.translatesAutoresizingMaskIntoConstraints = false
        iconImageView.centerYAnchor.constraint(equalTo: centerYAnchor).isActive = true
        iconImageView.leftAnchor.constraint(equalTo: leftAnchor, constant: 12).isActive = true
        iconImageView.heightAnchor.constraint(equalToConstant: 30).isActive = true
        iconImageView.widthAnchor.constraint(equalToConstant: 30).isActive = true

        addSubview(descriptionLabel)
        descriptionLabel.translatesAutoresizingMaskIntoConstraints = false
        descriptionLabel.centerYAnchor.constraint(equalTo: centerYAnchor).isActive = true
        descriptionLabel.leftAnchor.constraint(equalTo: iconImageView.rightAnchor, constant: 18).isActive
= true
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
}
```

Yllä olevassa luokassa iconImageView- sekä descriptionLabel-olioiden alustuskoodi on luotu manuaalisesti. Myös edellämainittujen olioiden rajoitteet ja asettaminen näkymähierarkiaan on tehty manuaalisesti init()-metodissa.

5.3. Controller

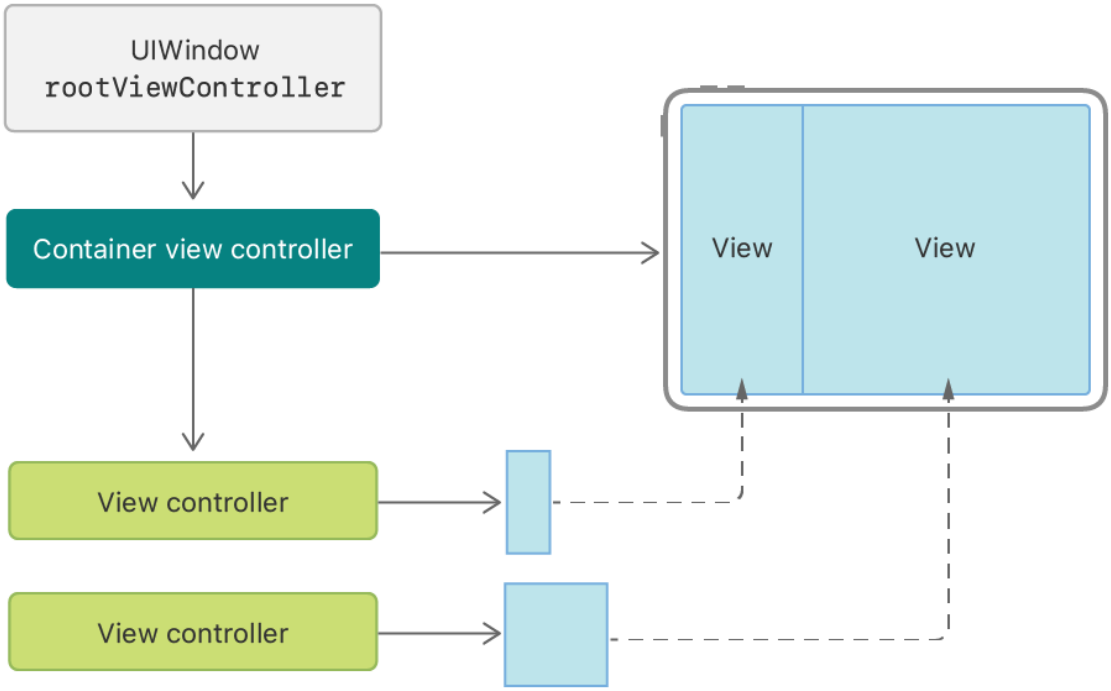
Sovelluksen ohjainkerrokseen kuuluu oheisia luokkia:

- ContainerController
- MenuController
- LoginController
- AllShiftsController
- WebViewController

- MainViewController.

Suurin osa sovelluksen logiikasta sijaitsee Controller-luokissa. Ne ovat kaikki implementaatioita UIViewController-luokasta. Käyttäjälle ne edustavat kukin jotain näkymää sovelluksessa. ContainerController, MenuController sekä MainViewController muodostavat yhdessä sovelluksen päänäkymän. Siinä käyttäjä asettaa työaikansa sekä navigoi päivissä eteen- ja taaksepäin.

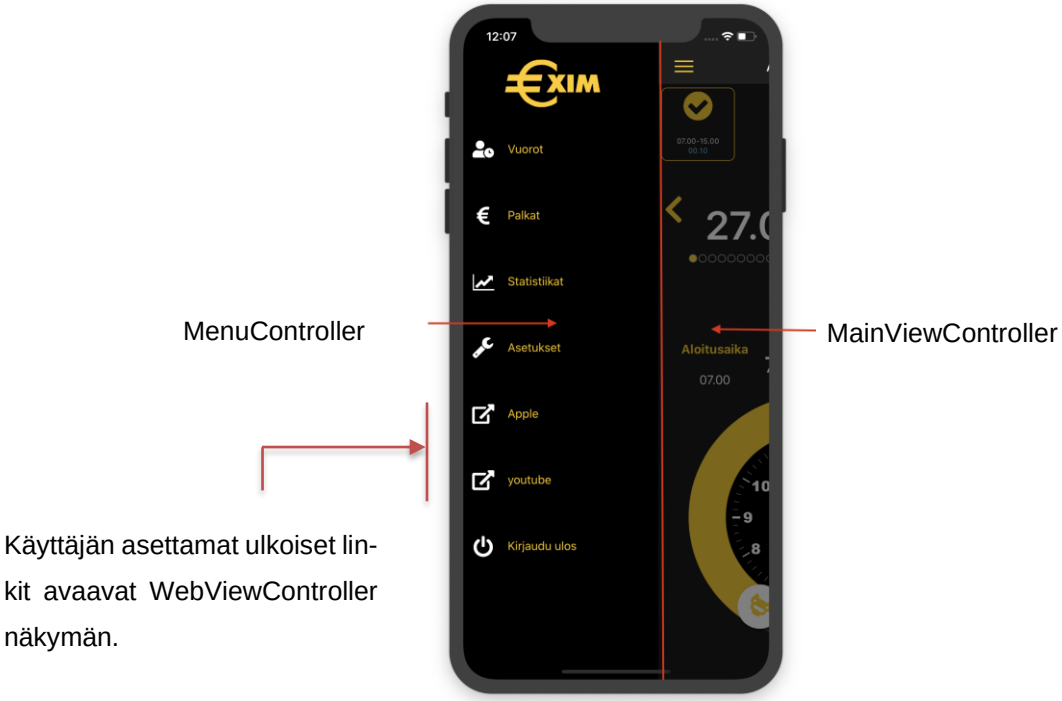
Päänäkymässä näkymäohjain-hierarkia on toteutettu Container view controller -tekniikalla. Tässä kaavassa toimii näkymän perustana yksi ylempi näkymäohjain, jolla on kaksi tai useampi lasta. Tällä tekniikalla on mahdollista näyttää kahta ViewController -instanssia ruudulla samaan aikaan. Kuvassa 9 on havainnollistus eri luokista ja niiden suhteista, joita tarvitaan ContainerViewController näkymään.



Kuva 9. Container ViewController -mallin näkymähierarkia [11]

UIWindow on olio, joka on säiliö näkymähierarkian alimmalle ViewControllerille. UIWindow sisältää Container view controllerin, joka puolestaan näyttää kahta muuta View Controlleria, jotka puolestaan näyttävät itse käyttäjälle näkyvät näkymät.

Kuvassa 10 näkyy, miten tässä insinöörityössä tekniikkaa käytetään Menu-valikon luonnissa. Painettaessa vasemmalta ylhäältä valikkopainiketta avautuu vasemmalle liukuva valikko, joka on instanssi MenuControllerista.



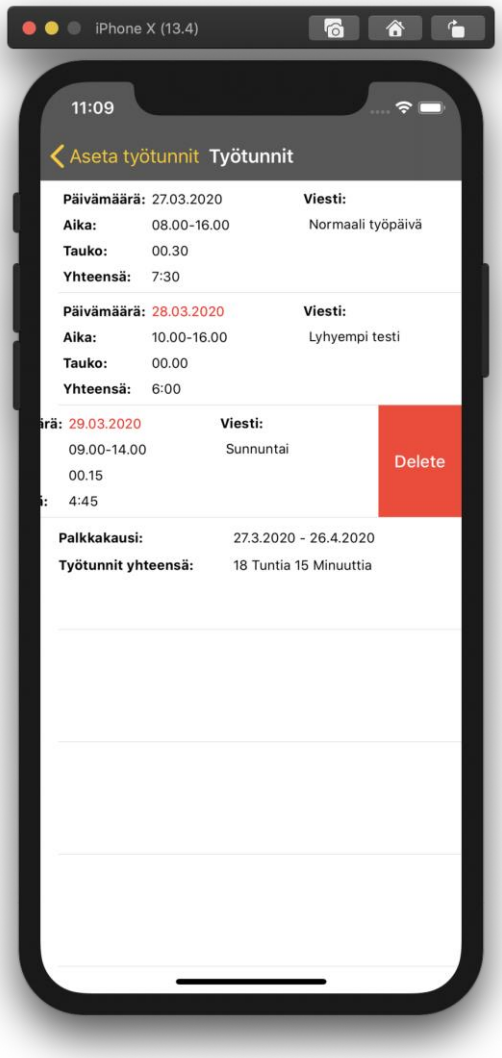
Kuva 10. Container ViewController hallitsee MenuControlleria sekä MainViewControlleria

Kuvassa 11 näkyy myös, miten Container view controller -tekniikalla on mahdollistettu päänäkymän alta paljastuva viestikenttä, jolla käyttäjä voi asettaa työvuoromerkinnälleen viestin.

Kuva 11. Viestikenttä aukeaa painamalla oikealta alhaalta puhekuplapainiketta

LoginViewController on nimensä mukaisesti kirjautumisruudusta vastaava ViewController, kun taas WebViewControlleria käytetään avaamaan käyttäjän itse tallettamia weblinkkejä sovelluksesta, joita näkee MenuViewControllerissa. Näiden käyttötarkoitus on käyttäjäkohtainen, mutta niillä voi avata vaikkapa kunkin taksiyrityksen sisäisten verkkopalveluiden sivuja (Kuva 10).

AllShiftsController on UITableViewController instanssi, jonka tehtävänä on listata kaikki käyttäjän syöttämät Shift-oliot kunkin käynnissä olevan palkkakauden ajalta.



Kuva 12. Käyttäjä voi pyyhkäisemällä poistaa haluamansa työaikamerkinnän, ja painalluksella hän pääsee editoimaan merkintää.

6. Yhteenveto

Insinöörityöllä pyrittiin tarjoamaan kätevä ja mobiiliystävällinen tapa kirjata työtunteja Exim-järjestelmään. Saimme aikaan sovelluksen, joka täyttää ensisijaiset vaatimukset ja on jokapäiväisessä käytössä asiakkaillamme. Samalla todettiin, että sovelluksen laajentaminen tarkemman työtuntidatan tallennukseen on hankalaa, jos halutaan ylläpitää hyvää ja yksinkertaista käyttöliittymää. Mobiililaitteiden näyttökoot ovat tässä suurin rajoittava tekijä.

Lopputuloksena valmistui toimiva sovellus, joka on käytössä osalla Trafore Oy:n asiakkaista. Sovellus on saatavilla App Storesta, joskin siihen kirjautuminen vaatii Exim-tunnukset.

Työ oli loistava oppimiskokemus minulle ja Trafore Oy:lle. Se toimi myös hyvänä ponnistusalustana uralleni mobiilisovellusten kehittämisen pariin. Insinöörityötä tehdessä pääsin kokemaan mobiilisovellusten kehittämisprosessin alusta loppuun. Se loi ymmärrystä erilaisten työskentelykäytäntöjen, arkkitehtuurien ja suunnittelumallien tärkeydelle, sekä vahvisti tuntemustani front end- ja backend-palvelujen kanssakäymisestä.

Lähteet

1. Statcounter Global Stats. Maaliskuu 2020. Mobile Operating System Market Share Worldwide. Verkkoaineisto <https://gs.statcounter.com/os-market-share/mobile/worldwide#> Luettu 1.4.2020.

2. SensorTower. 2019. Worldwide Mobile App Revenue and Downloads. Verkkoaineisto. <https://sensortower.com/blog/app-revenue-and-downloads-1h-2019> Luettu 1.4.2020.

3. Apple Developer Documentation Archive. 2013. What is Cocoa? Verkkoaineisto. <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html> Luettu 12.2.2020.

4. Apple Developer Documentation. Foundation. Verkkoaineisto. <https://developer.apple.com/documentation/foundation> Luettu 14.2.2020.

5. Apple Developer Documentation. Core Data. Verkkoaineisto. <https://developer.apple.com/documentation/coredata> Luettu 14.2.2020.

6. Apple Developer. Xcode 11. Verkkoaineisto. <https://developer.apple.com/xcode/ide/> Luettu 25.2.2020.

7. Apple Developer. 2016. Navigating Your Workspace. Verkkoaineisto. <https://developer.apple.com/library/archive/documentation/ToolsLanguages/Conceptual/Xcode_Overview/NavigatingYourWorkspace.html> Luettu 9.2.2020.

8. Apple Developer. 2016. Using Interface Builder. Verkkoaineisto. <https://developer.apple.com/library/archive/documentation/ToolsLanguages/Conceptual/Xcode_Overview/UsingInterfaceBuilder.html> Luettu 9.2.2020.

9. Apple Developer. 2016. UICollectionView. Verkkoaineisto. <https://developer.apple.com/documentation/uikit/uicollectionview> Luettu 1.2.2020.

10. Apple Developer. 2018. Model-View-Controller. Verkkoaineisto. <https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html> Luettu 13.3.2020.

11. Apple Developer. Creating a Custom Container View Controller. Verkkoaineisto. <https://developer.apple.com/documentation/uikit/view_controllers/creating_a_custom_container_view_controller> Luettu 29.3.2020.