Antti Pilvilampi

MUISTIO-OHJELMA OHJELMISTOPROJEKTINA

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutus
Huhtikuu 2021

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Huhtikuu 2021	**Tekijä/tekijät** Antti Pilvilampi
Koulutus Tieto- ja viestintätekniikka		☒ AMK ☐ YAMK
Työn nimi MUISTIO-OHJELMA OHJELMISTOPROJEKTINA.		
Työn ohjaaja Kauko Kolehmainen		**Sivumäärä** 34 + 3
Työelämäohjaaja Kauko Kolehmainen		

Opinnäytetyön ohjaaja antoi kehittäjälle luvan tuottaa itse suunniteltu ohjelma. Ohjelmaksi valittiin erääntyviä merkintöjä hallitseva muistio-ohjelma, johon käyttäjä voi tallentaa esimerkiksi työ-, harrastus- tai koulutehtävien palautusmääräpäiviä.

Kehitys seurasi ohjelmistoprojektin eri vaiheita. Aluksi määriteltiin ohjelman vaatimukset, sitten suunniteltiin rakenne ja käyttöliittymän ulkoasu. Luodun suunnitelman perusteella ohjelma ohjelmointiin valmiiksi toteutusvaiheessa.

Ohjelman logiikka toteutettiin oliopohjaisesti C#-ohjelmointikielellä, ja Windows Forms -luokkakirjastoa käytettiin rakentamaan muistion käyttöliittymä. Käyttäjän luomat merkinnät kehitettiin tallentumaan käyttäjän kovalevylle.

Muistio-ohjelma kehitettiin onnistuneesti, ja sen toiminta saatiin vastaamaan määriteltyjä vaatimuksia.

Asiasanat
C#, lomake, ohjelmistoprojekti, ohjelmointi

ABSTRACT

Centria University of Applied Sciences	**Date** April 2021	**Author** Antti Pilvilampi
Degree programme Information Technology		
Name of thesis MEMO-SOFTWARE AS A SOFTWARE PROJECT.		
Centria supervisor Kauko Kolehmainen		**Pages** 34 + 3
Instructor representing commissioning institution or company Kauko Kolehmainen		

The thesis supervisor allowed the developer to produce a self-designed application. The application chosen for development was a Memo-application, that contains expiring notes, which can for example mark the return dates of work, hobby, or school related deliverables.

Development of the application followed software development phases. First, requirements were defined for the application's functionality, then the structure and user interface of the application was designed. Finally, the application was programmed in the development phase.

The object-oriented application's logic was programmed with C#. Object classes from the Windows Forms -library were used to create the user interface. Notes created by the user were designed to be saved in a database file on the hardware of the end user.

The memo-application was successfully developed, as its functionality filled all the defined requirements.

Key words
C#, form, programming, software project

KUVIOT

KUVAT

TAULUKOT

KOODIT

1 JOHDANTO

Ohjelmistoprojekti on vaiheilla ohjattua toimintaa ohjelmistojen kehittämistyöhön yrityksessä tai muussa ryhmässä. Yksi ohjelmistoprojekti ei kestä ikuisesti, vaan niin kauan, kunnes sen tavoite saavutetaan, jolloin sen varaamat resurssit, kuten kehittäjät ja työkalut, vapautuvat toista projektia varten. Ohjelmistoprojektin tavoite voi olla kokonaan uuden ohjelmiston kehitys, tai olemassa olevan palvelun täydentäminen.

Eri vaihemalleja ohjelmistoprojektille löytyi monta, mutta lähteistä löydetyt olivat pääpiirteittäin samoja. Projektin alussa kehittäjät vastaanottavat tehtävänannon tai asiakkaan tarjouspyynnön, ja alkavat vaatimusmäärittelyn, jossa analysoidaan tärkeimmät vaaditut toiminnot. Kun vaatimukset hyväksytään, suunnitellaan niiden avulla ohjelmiston rakenne, joka malleineen ohjeistaa kehittäjiä toteuttamaan ohjelmiston vaaditut toiminnot. Ohjelmisto testataan ja toimitetaan korjattuna käyttäjälle, jonka jälkeen projektin lopussa käyttäjälle tarjotaan ohjelmiston ylläpitoa. Ylläpidon aikana annetaan teknistä tukea ohjelmiston käytössä. (Elysium Academy Support 2017.)

Tässä työssä tehtävänantona oli kehittää muistio-ohjelma seuraten yllä kuvattuja vaiheita. Ohjelmaidea tehtiin itse, ja siitä aloitettiin määrittelyvaihe. Vaatimukset kirjattiin, ja esitettiin kuvioin. Vaatimuksia määriteltiin käyttötapauksille, sekä käyttöliittymälle että ohjelman rakenteelle.

Suunnitteluvaiheessa mallinnetaan rakennetta määritellystä vaatimuksesta kokonaisemmaksi, yrittäen korostaa muutoksia. Tehdään kuvio graafisen käyttöliittymän sisältävästä arkkitehtuurista, josta ohjelman rakenne selviää paremmin. Käyttöliittymän ulkoasu, ja sen käyttäytyminen viimeistellään, ja esitetään kuvien avulla.

Toteutuksessa esitetään lyhyesti valitut työkalut, kuten kehitysympäristö Visual Studio, sekä käytetty graafisen käyttöliittymän mahdollistava luokkakirjasto Windows Forms. Ohjelman onnistunut toteutus esitetään selittämällä koodia ohjelman päätoimista.

2 OHJELMISTOPROJEKTI

Ohjelmistoprojektin vaiheet on määritelty useimmissa ohjelmiston elinkaarimalleissa seuraavasti: Vaatimusten määrittely- ja analyysivaiheessa sovitaan ohjelmiston tärkeimmät ominaisuudet ja luodaan alustavasti aikataulu ja budjetti. Suunnitteluvaiheessa ohjelmiston toimintaa tarkennetaan suunnitelmaksi vaatimusten perusteella. Ohjelmointi tapahtuu toteutusvaiheessa, testauksessa varmistetaan ohjelmiston toimivuus, toimituksessa annetaan ohjelmisto asiakkaan käyttöön ja ylläpitovaiheessa autetaan asiakasta korjaamalla käytön aikana ilmestyneitä ongelmia. (Elysium Academy Support 2017.)

2.1 Määrittely ja analyysi

Alussa keskustellaan asiakkaan kanssa ohjelmiston vaatimuksista ja esitetään kysymyksiä tarkoituksesta, halutuista ominaisuuksista sekä käyttäjistä (Elysium Academy Support 2017). Projektin käsitetään alkavan jo tarjouspyynnöstä, koska vaatimusten perusteella arvioidaan budjettia. Budjettiin vaikuttavat projektin vaatima työmäärä, resurssit ja ulkopuolisen avun tarve. (Leppäniemi.) Työn määrä voidaan arvioida esimerkiksi jakamalla tehtävät osiksi, ja luomalla osien perusteella aikataulu (Estimating Software Engineering Effort: Project and Product Development Approach).

Asiakkaan esittämistä vaatimuksista tehdään analyysi, tutkitaan ovatko kaikki toiminnot mahdollisia ja kannattavia toteuttaa. Hankaliin vaatimuksiin voidaan etsitä parempia ratkaisuja. Hyväksytyt vaatimukset kirjataan dokumenttiin, jonka pohjalta aloitetaan suunnitteluvaihe. (Initiation Phase.)

2.2 Suunnittelu

Suunnitteluvaiheessa ohjelmiston toiminta halutaan rajata vaatimusten perusteella tarkemmin. Määritellään esimerkiksi verkkosivuja varten tietokannan rakenne lopulliseen muotoon, sekä viimeistellään käyttöliittymän ulkonäkö ja ohjelman rakenne. (Azarian 2013.) On kuitenkin hyväksyttävä, että on mahdotonta ennakoida kaikkea, koska toteutuksessa voi tapahtua jotain, joka pakottaa tekemään muutoksia, onpa se sitten kehittäjälle positiivista tai ei.

Toteutusta varten mallinnus aloitetaan hahmottamalla ohjelmiston eri osien, luokkien ja järjestelmien yhteyksiä toisiinsa. Voidaan aloittaa suurimmista osista, ja tarpeen tullen lisätä ja yhdistää pienempiä, jos niitä tarvitaan. (Design Phase.) Tähän voi käyttää esimerkiksi luokkakaaviota (KUVIO 3), josta nähdään ohjelmiston eri luokat ja niiden väliset suhteet.

Prototyyppien, eli varhaisten ohjelman mallien, luominen auttaa hahmottelemaan ohjelmistoa ja on kannattavaa, koska on helpompaa muokata toimintoja ennen tuotantovaihetta. Suunnittelijat mallintavat ohjelmistoa niin, että vaadittuja osia saadaan esitettyä asiakkaalle palautehakuisesti. (Goran 2019.)

Suunnittelu on myös projektiin tarvittavien resurssien tarkemman määrittämisen vaihe. Projektinjohtajat mm. arvioivat vaatimusten toteuttamiseksi aikataulun, jakavat tehtävät, ja asettavat standardit ohjelmiston toimintatason varmistamiseksi. (Planning Phase.)

2.3 Toteutus

Ohjelmiston toteutus on kestoltaan pisin vaihe (Elysium Academy Support 2017). Ohjelmisto rakennetaan noudattaen suunnitelmaa mahdollisimman tarkasti. Kehittäjät ohjelmoivat heille määrätyt osat, jotka suorittavat jonkun vaadituista toiminnoista. He testaavat, että osa toteuttaa toiminnon, ja liittävät sen ohjelmistoon, jolloin osan toimivuutta voidaan todentaa käyttämällä sitä muiden osien kanssa. (Development Phase.)

Toteutuksessa tuotettava dokumentaatio voi olla joko käyttäjää varten tehty ohjekirja, tai suoraan koodin viereen kirjoitettuja kommentteja (Goran 2019). Käytettyjen ratkaisujen selittäminen ja niiden käytön ohjeistus voi olla hyödyllistä paitsi muille toteutukseen osallistuville kehittäjille, myös itse kirjoittajalle (Sourour 2017).

2.4 Testaus ja toimitus

Testauksen tarkoitus on varmistaa ohjelmiston toimivuus sekä osittain että kokonaisuutena testiympäristössä. Tässä vaiheessa etsitään virheitä toiminnassa, sekä varmistetaan, että ohjelmisto vastaa asiakkaan vaatimuksia. Testaus tulisi toteuttaa perinpohjaisesti ennen ohjelmiston toimittamista asiakkaalle,

jotta vältytään vahingoittamasta heidän toimintaansa liian myöhään havaituilla ongelmilla. (Performance Lab.)

Toimitusvaihe sisältää ohjelmiston asentamisen kohdeympäristöön sekä ohjelmiston käytön kouluttamista käyttäjille. Toimitus suoritetaan asiaankuuluvalle henkilöstölle toimitetun aikataulun mukaisesti vaiheittain, jotta mahdollisesti tarvittavat tiedot saadaan siirrettyä, ja ohjelmiston toimivuus varmistetaan ennen kuin käyttäjät alkavat virallisesti käyttää ohjelmistoa. (Implementation Phase.)

2.5 Ylläpito

Ylläpito on ohjelmistoprojektin viimeinen vaihe. On todennäköistä, että testauksesta huolimatta käyttäjät voivat löytää uusia ongelmia ohjelmistossa (Elysium Academy Support 2017), joten on kehittäjän velvollisuus, sopimuksen mukaan, tarjota teknistä tukea.

Korjauksia ongelmiin, päivityksiä tai muutoksia tehdään määriteltyjen toimintaohjeiden mukaan, jotta teot eivät vahingoita ohjelmiston nykyistä toimintatasoa. Toimintaohjeet luodaan tasoittain, riippuen muutosten koosta ja kiireellisyydestä. Kriittisen vian korjaamisessa lyhennetään monta käytäntöä ja tarkistusta, joita on tärkeää suorittaa kokonaisesti suurempien, koko ohjelmistoon vaikuttavan muutoksen kanssa. (Maintenance Phase.)

3 MÄÄRITTELY JA ANALYYSI

Tässä luvussa esitellään kehitettävän ohjelman idea, jota analysoimalla määritellään tärkeimmät vaatimukset, jotka valmiin ohjelman tulisi täyttää toteutusvaiheen päätyttyä. Vaikka vaatimukset pyritään muodostamaan perinpohjaisesti tässä vaiheessa, on mahdollista, että vaatimukset muuttuvat tai lisääntyvät projektin myöhemmissä vaiheissa, riippuen projektin edetessä suoritettujen mallinnusten ja testausten tuloksista.

Ohjelman tulisi olla kalenterin kaltainen muistio-ohjelma, johon käyttäjä voi luoda merkintöjä. Merkinnät sisältäisivät otsikon ja sopivan määrän tilaa merkintää selittävälle tekstille. Käyttäjä asettaisi merkinnälle erääntymispäivämäärän, joka olisi viimeinen päivä täyttää merkinnän käyttäjän määrittämä tavoite, joka voisi olla vaikkapa viimeinen päivä palauttaa koulutehtävät.

Merkinnät tulisi jakaa kategorioittain riveihin, joihin ne järjestettäisiin erääntymispäivän mukaan. Ohjelma tarkastaisi merkinnät automaattisesti, ja jos merkinnän päivämäärä olisi ohitettu, sen ulkoasu muutettaisiin niin, että käyttäjä erottaisi vanhentuneen merkinnän helposti.

Kategorioita tulisi olla mahdollista poistaa, uudelleennimetä, lisätä ja niiden järjestystä muokata ohjelman sisällä. Ohjelman pitäisi toimia Windows 10 -käyttöjärjestelmässä.

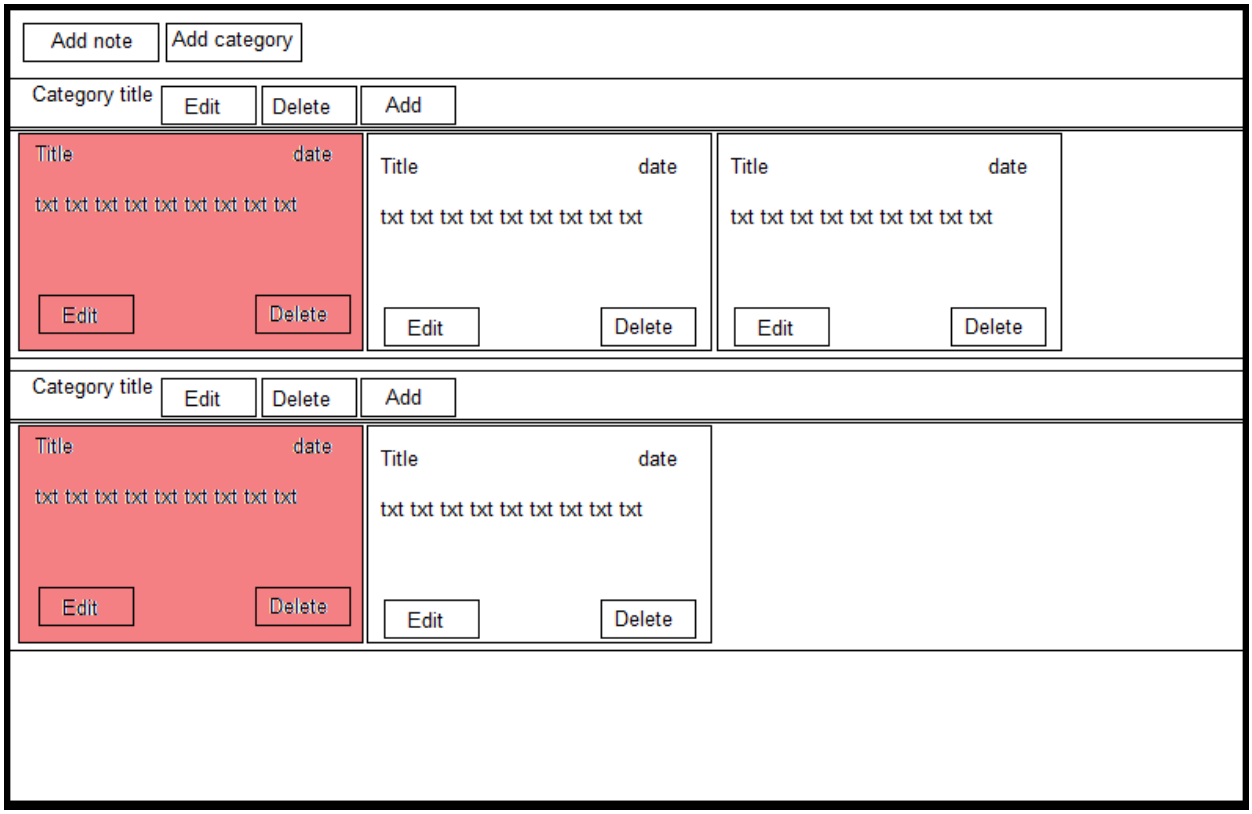
3.1 Toiminnalliset vaatimukset

Ohjelmiston toiminnalliset vaatimukset määrittelevät, mitä toimintoja ohjelmiston tulee sisältää, jotta ohjelmisto toteuttaa käyttäjän tavoitteet. Nämä vaatimukset ohjaavat siis ohjelmiston kehitystä suunnittelussa ja toteutuksessa kuvaamalla tiettyjen toimintojen haluttuja tuloksia. Toiminnallisia vaatimuksia kirjataan, ja niitä on hyvä esittää kuvioillakin. (AltexSoft 2018.)

Ohjelmaksi ei haluta kalenteria, vaan muistio, jossa merkinnät ja kategoriat rakenteeltaan muistuttavat kalenterin päiviä ja viikkoja. Kategoria ja merkintä ovat omat luokkansa, joista ohjelma luo kontrollit käyttäjälle. Koska ohjelma on pöytätietokoneella toimiva, käyttäjä hallitsee muistiota todennäköisesti hiirellä ja näppäimistöllä. Ohjelmalla on yksi käyttäjä, joten ei tarvitse olla mahdollista tallettaa enempää kuin yksi muistio.

3.1.1 Käyttöliittymän vaatimukset

Kuva 1 on mallinnus käyttöliittymästä. Merkintöjen tulee olla selkeästi erotettavissa toisistaan, kukin ”ruutu” sisältää merkinnän omaa tekstiä ja informaatiota. Nähtävä tieto tulee sisältää merkinnän otsikon, viestin ja erääntymispäivän. Kategorioittain merkinnät järjestetään niin, että ensiksi erääntyvät ovat vasemmalla. Erääntynyt merkintä tulee erottaa muista visuaalisesti, kuvassa 1 erääntyneet merkinnät erotetaan punaisella taustavärillä. Muistion kontrollien täytyy esittää toimintansa mahdollisimman itsestään selvästi. Käyttäjältä pyydettäessä tietoa tulee myös ilmoittaa virheistä syötössä.



KUVA 1. Käyttöliittymän malli (Pilvilampi 2021)

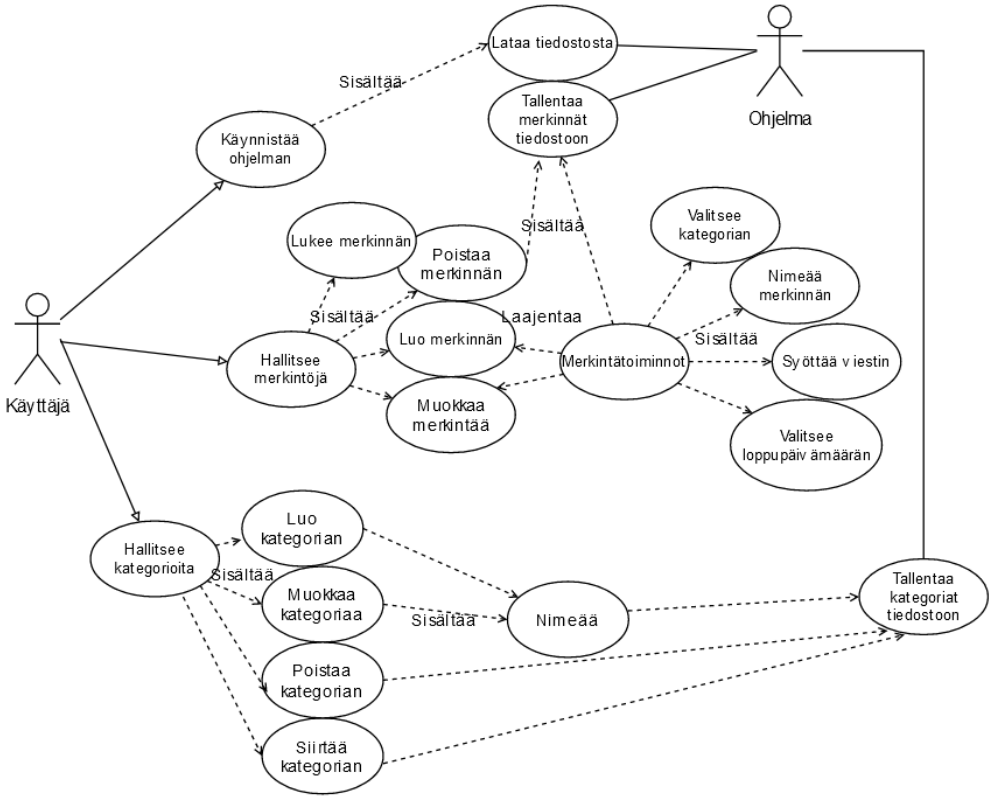
3.1.2 Käyttötapaukset

Kuvio 1 kuvailee ohjelman määriteltyjä käyttötapauksia. On kaksi tekijää: käyttäjä sekä itse ohjelma. Käyttötapaukset ovat vaatimuksia tekijän mahdollisuuksista käyttää ohjelmaa.

Muistion käyttäjän käyttötapaukset on kuviossa 1 jaettu merkintöjen ja kategorioitten hallintaan ja ohjelman käynnistykseen. Käyttäjän tulee pystyä lukemaan merkintöjen viesti kokonaan, poistamaan

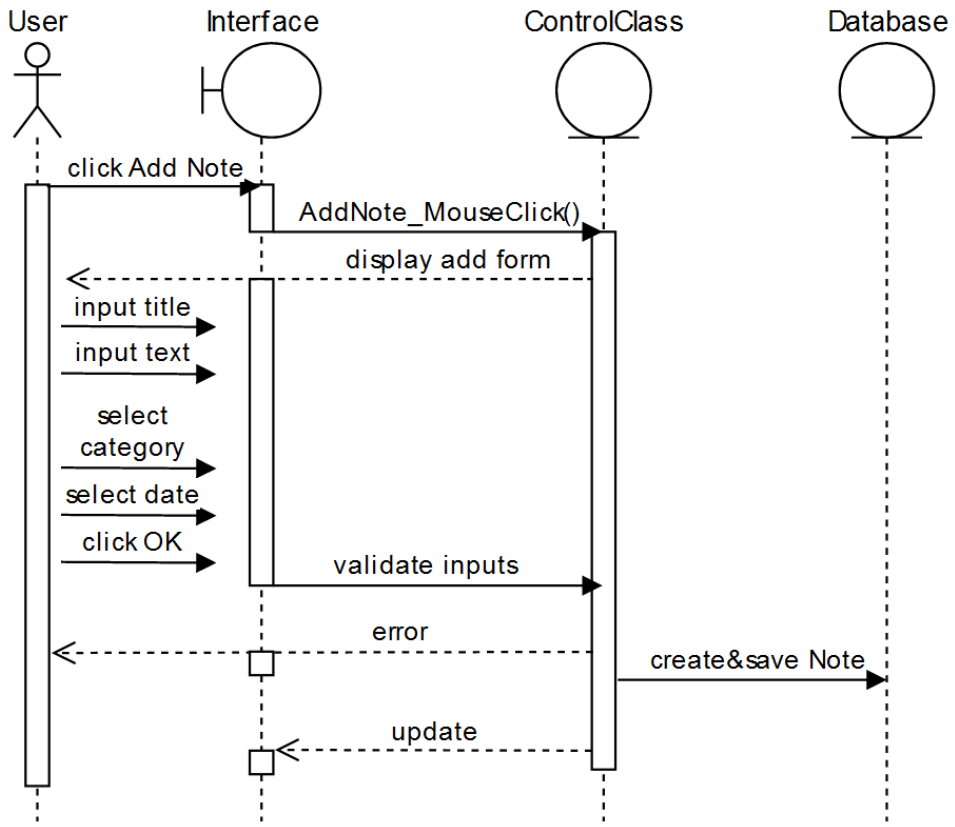
merkintöjä ja kategorioita, luomaan ja muokkaamaan niitä. Käyttäjän tarvitsee hallita vain yhtä merkintää tai kategoriaa kerrallaan.

Ohjelman tulee pystyä reagoimaan käyttäjän toimintaan tallentamalla muuttuneet tiedot tiedostoon, ja lataamalla sekä muodostamalla muistion tiedostosta käyttäjälle käynnistyessään.



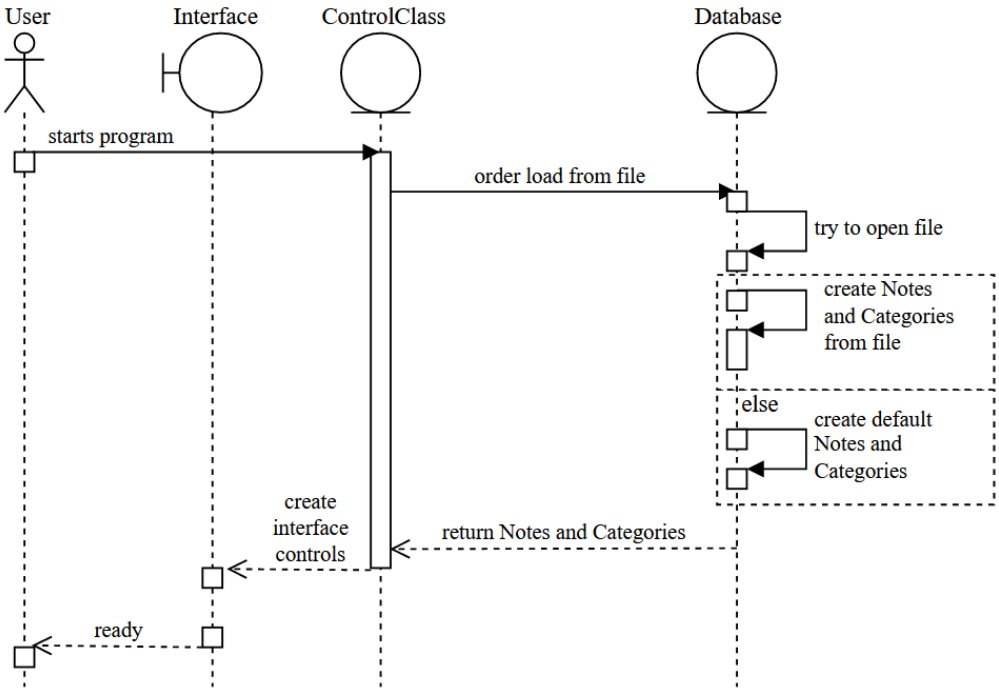
KUVIO 1. Ohjelman käyttötapaukset, määritysvaihe (Pilvilampi 2021)

Merkinnän luominen-käyttötapauksessa (KUVIO 2) käyttäjä aloittaa prosessin napsauttamalla ”Add Note”-painiketta, joka kutsuu ”ControlClass”-olion merkinnänlisäysmetodin. Ohjelma pyytää käyttäjää syöttämään tiedot, ja odottaa ”OK”-painikkeen napsautusta. Syötetty tieto tarkistetaan, ja jos havaitaan virhe, näytetään virheviesti. Muuten merkintä luodaan ja tallennetaan tietokantaan, jonka jälkeen merkinnän onnistunut lisäys näytetään päivittämällä käyttöliittymä.



KUVIO 2. Sekvenssikaavio uuden merkinnän luomisesta (Pilvilampi 2021)

Ohjelman käynnistys -käyttötapauksessa (KUVIO 3) käynnistys ohittaa käyttöliittymän, koska sitä ei vielä ole. Ohjelman logiikka alkaa hallintaoliosta, joka luo tietokantaolion. Tietokantaolio käsketään sitten yrittämään hakea muistion tietokantatiedostoja. Tietokantaolio muodostaa merkintä- ja kategoriaoliot joko olemassa olevista tiedostoista, tai niiden puuttuessa oletusarvoilla. Näitä olioita käyttäen hallintaolio rakentaa käyttöliittymän valmiiksi käyttöä varten.



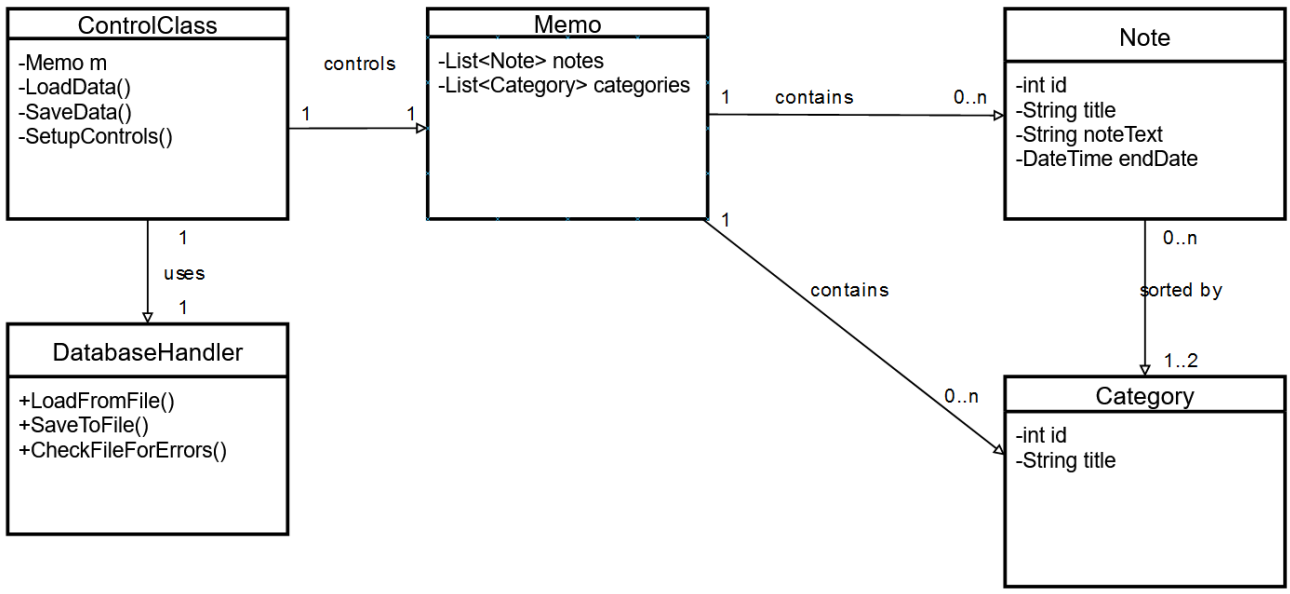
KUVIO 3. Sekvenssikaavio ohjelman käynnistys-käyttötapauksesta (Pilvilampi 2021)

3.1.3 Luokkakaavio

Kuvio 4 on ohjelman analyysin perusteella tehty alustava luokkakaavio, joka kuvaa paitsi eri ohjelmassa käytettävien luokkien ominaisuuksia, myös niiden välisiä suhteita. Analyysi aloitettiin pienimmistä tarvittavista osista.

Ohjelma tarvitsee merkintäluokan olioita sisältämään merkinnän tiedot: tunnisteen, otsikon, viestin sekä päivämäärän. Kategoriaoliolla on nimi, ja sillä järjestetään merkinnät, tähän ohjelma tarvitsee jokaiselle kategorialle oman tunnisteen. Merkinnällä tulisi olla yksi kategoria, johon se kuuluu. Toisaalta yhdellä kategorialla ei ole pakko olla yhtäkään merkintää. Muistio-olion listat sisältävät merkinnät ja kategoriat.

Kontrolliluokka hallitsee muistiota. Se hakee muistio-oliolta listat muodostaessaan käyttöliittymän kontrolleja, ja kun oliot pitää tallentaa tietokantaan, käyttää se kuvion 3 mukaan tietokannan hallintaan kehitettyä oliota.



KUVIO 4. Ohjelman luokkakaavio, analyysivaihe (Pilvilampi 2021)

3.1.4 Luokat

Tässä osiossa esitetään analyysin perusteella taulukoihin kerätyt luokkien ominaisuudet: niiden sisältämät muuttujat ja oliot, sekä molempien tyypit. Sarake ”KEY” merkitsee, onko ominaisuus luokan olion avainarvo.

TAULUKKO 1. ”DisplayControl”-luokka, attribuutit.

NAME	KEY	Attribute Type	DATA TYPE	SIZE	Null Allowed
m	No	Object	Memo	1	No
dh	No	Object	DatabaseHandler	1	No
updater	No	Object	Timer	1	No
addNoteButton	No	Object	Button	1	No
addCategoryButton	No	Object	Button	1	No

Taulukko 1 kuvaa muistiota hallitsevan luokan olion sisältöä. Merkinnät ja kategoriat haetaan muistiota edustavan ”Memo”-luokan oliolta. Ajastinolio ”updater” suorittaa toistuvasti päivityksen, jossa

tarkistetaan, onko muistiossa vanhentuneita merkintöjä. Tietokantaolio ”dh” tulee tallentamaan ja lataamaan tiedot tietokantatiedostosta. ”Button”-oliot ovat painikkeita, joilla käyttäjä voi lisätä merkintöjä ja kategorioita. Hallintaluokan oliolla ei ole avainarvoa, ja se tarvitsee vain yhden kutakin oliota.

TAULUKKO 2. ”Memo”-luokka, attribuutit

NAME	KEY	Attribute Type	DATA TYPE	SIZE	Null Allowed
allNotes	No	Array	Note	0…*	No
allCategories	No	Array	Category	0…*	No

”Memo”-luokan olio (TAULUKKO T2) sisältää muistion merkinnät ja kategoriat listoissaan. Ohjelma hakee listat olioineen toimintoihin. Taulukosta 1 nähdään, että ohjelma tarvitsee vain yhden muistio-olion, joten oliolla ei tarvetta avainarvolle, jolla ohjelma erottaisi sen toisista.

TAULUKKO 3. ”DatabaseHandler”-luokka, attribuutit

NAME	KEY	Attribute Type	DATA TYPE	SIZE	Null Allowed
notesFileName	No	Constant	String	20…*	No
categoriesFileN	No	Constant	String	20…*	No

”DatabaseHandler”-luokan olio (TAULUKKO 3), joka hallitsee tietokantaa, tulee sisältämään kaksi arvoa, jotka osoittavat muistion tietokantatiedostojen sijainnin. Molempien tekstimuuttujien arvioidaan olevan ainakin 20 merkkiä pitkiä, eikä niiden arvoja haluta muuttaa ohjelman käytön aikana.

TAULUKKO 4. ”Category”-luokka, attribuutit

NAME	KEY	Attribute Type	DATA TYPE	SIZE	Null Allowed
id	Yes	Variable	Integer	1…*	No
sorting_Order	No	Variable	Integer	1…*	No
title	No	Variable	String	1…10	No

Kategoriaa edustava ”Category”-luokan olio sisältää taulukon 4 mukaisesti avainarvon id, numeroarvon ”sorting_Order” joka määrää kategorian järjestyksen ohjelmassa, sekä kategorian otsikon. Avainarvoa tarvitaan merkintöjen järjestämiseen ohjelmassa.

TAULUKKO 5. ”Note”-luokka, attribuutit

NAME	KEY	Attribute Type	DATA TYPE	SIZE	Null Allowed
id	Yes	Variable	Integer	0…*	No
categoryID	No	Variable	Integer	1…*	No
noteTitle	No	Variable	String	1…*	No
noteText	No	Variable	String	1…500	No
endDate	No	Variable	Date	~50	No
expired	No	Variable	boolean	1	No

”Note”-luokan oliot edustavat merkintöjä. Jokainen merkintä tarvitsee taulukon 5 mukaan avainarvon, jolla sen erottaa toisista. Lisäksi merkintä tarvitsee kategoriaan yhdistävän arvon, merkinnän otsikon, tekstin, ja erääntymispäivämäärän sekä totuusarvon, joka kertoo, onko merkintä erääntynyt. Merkinnän viestin merkkimäärää tulisi rajata, koska ei ole tarkoitus luoda päiväkirjaohjelmaa.

3.2 Ei-toiminnalliset vaatimukset

Ei-toiminnalliset vaatimukset varmistavat toiminnan standardit, jotka koskevat koko ohjelmaa. Nämä vaatimukset siis eivät kosketa vain yhtä osaa, luokkaa, tai käyttötapausta. Tähän lukuun on kerätty vaatimukset esim. ohjelman turvallisuudesta.

Ohjelman käyttöliittymän kontrollien ja osien tarkoitus tulisi olla mahdollisimman itsestään selvä. Mitään käyttäjän aloittaman toiminnan alkamista tai päättymistä ei saa viivästyttää. Muistion kontrollien tekstien tulisi olla englanninkielisiä, mutta merkinnän sisällön kieltä ei saa rajata.

Turvallisuusvaatimus on, että käyttäjän merkintöihin kirjoitettuja tietoja ei saa jakaa muiden kanssa. Ohjelman halutaan olevan täysin irti verkosta, joten ainoa sallittu tiedon tallennuspaikka on käyttäjän koneella sijaitseva tietokanta.

Luotettavuuden varmistamiseksi tietokannassa olevat tiedot muistosta, merkinnöistä ja kategorioista tulevat olla aina samat kuin ohjelmassa. Tiedot täytyy tallentaa aina, kun niihin havaitaan muutoksia. Ohjelma tulee lisäksi suorittaa tietokannan virheentarkastusta ja korjausta käynnistyksen aikana.

Järjestelmävaatimukset ohjelmistolle on tietokone, joka kykenee pyörittämään Windows 10-käyttöjärjestelmää. Ohjelman suorituskyvyn halutaan olevan mahdollisimman hyvä, ottaen huomioon, että käyttäjä uskoo ohjelman käsittelevän vain tekstiä.

Ylläpidon aikana muutosten teon tai viankorjauksen vuoksi, metodit, muuttujat ja oliot tulee nimetä niin, että niiden tarkoitus on selvä. Olioista saa käyttää lyhenteitä, jos lyhenne on tehty luokan nimestä. Jokaiseen tärkeään metodiin tulisi selittää kommentein sen toimintaa tai logiikkaa. Suunnittelussa tai kehityksessä ei saa asettaa tahallisia rajoituksia, jotka estäisivät tulevaisuudessa kehitettäviä ominaisuuksia.

4 SUUNNITTELU

Tässä luvussa esitellään määrittelyvaiheen pohjalta tehty suunnitelma, jossa on tarkemmin mallinnettu käyttöliittymää ja ohjelman rakennetta. Suunnittelussa on tarkoitus lyödä ohjelma lukkoon mallitasolla, jotta toteutukseen saadaan mahdollisimman tarkat ohjeet.

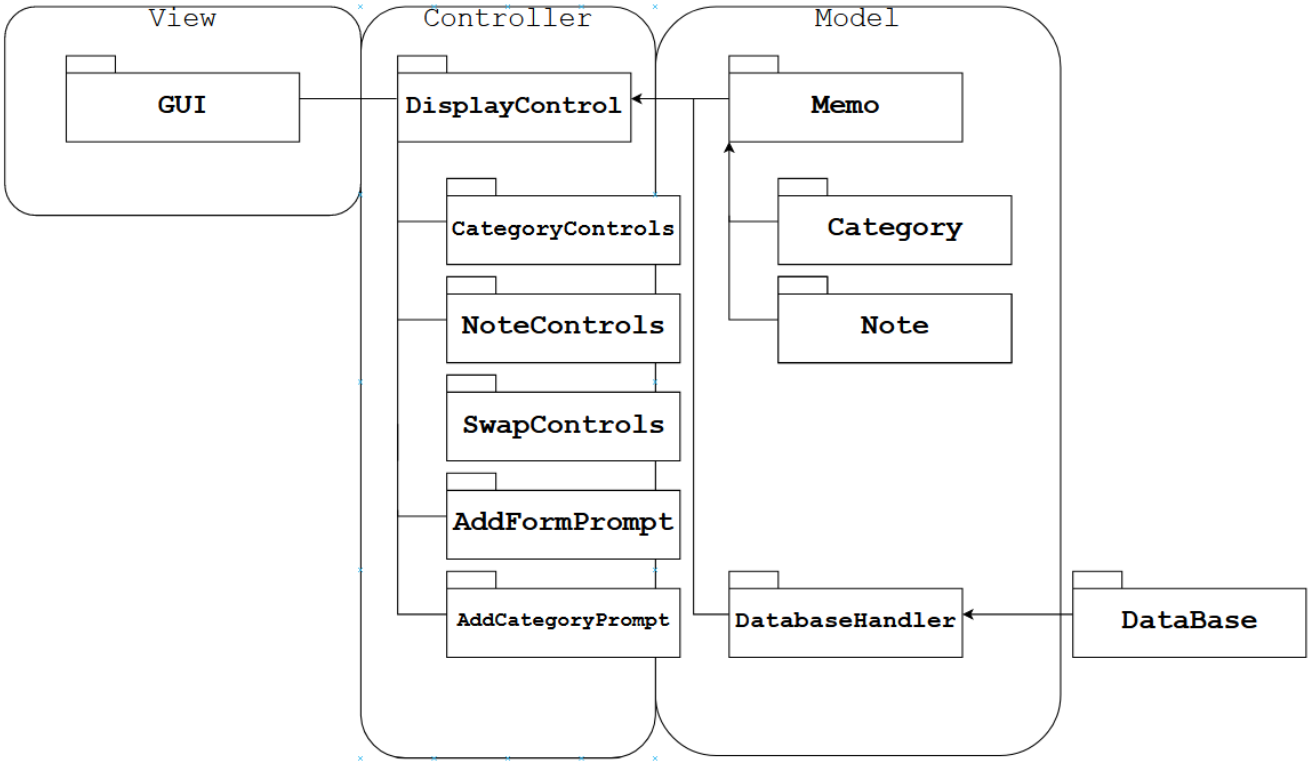
Ohjelman rakennetta jouduttiin muuttamaan suunnittelun aikana prototyyppitestien tulosten perusteella. Yksi suuri muutos tehtiin merkintäluokkaa koskien. Haluttiin, että käyttäjän on mahdollista nähdä jokainen merkintä yhdessä kategorianäkymässä. Koska kategorioiden määrälle ei ole rajaa, voi jokin kategoria unohtua ruudun ulkopuolelle merkintöjensä kanssa. Merkinnöille annettu paneeli kontrolleineen tulisi olemaan siis maksimissaan kahdessa paikassa, mikä on mahdotonta. Siksi suunniteltiin merkintöjä edustava merkintäkontrolliluokka ”NoteControls”, jonka olio sisältää viittauksen edustettuun merkintään. Viittaus merkintäolioon sallii erillisten, samaa merkintää hallitsevien painikkeiden ja paneelien luomisen myös suunniteltuun ”All”-kategoriaan.

4.1 Arkkitehtuuri

MVC eli Model-View-Controller on yksi tapa suunnitella ohjelman arkkitehtuuri. Siinä osat erotetaan kolmeksi suuruudeksi, ”Model” tai malli on ohjelman sisältämä tieto, jota hallitsee käyttäjän toimintoihin reagoiva ”Controller”, eli ohjain. ”View”, eli näkymä esittää mallin tiedot käyttäjälle. Tämä osien erottelu auttaa kehittäjiä toteutusvaiheessa, sillä eri osissa olevia komponentteja voidaan kehittää samalla aikaa. (Visual Paradigm.)

Tämän ohjelman arkkitehtuuri (KUVIO 5), on sovellettu versio MVC-mallista. Näkymän ainut osa on graafinen käyttöliittymä, jota ohjain päivittää mallin muuttuessa. Käyttöliittymä ei kuitenkaan ole konkreettinen, oma osa ohjelmassa, vaan se muodostetaan ohjainosion sisältämien luokkien toimista. Samalla käyttöliittymästä käyttäjän tahto toteutetaan ohjaimen reaktioilla.

Kuviossa 5 ohjaimen sisällä olevat kontrolliluokat, jotka kuuluvat hallintaluokalle, kuvataan olevan osittain malliosion päällä. Nämä kontrolliluokat voi luokitella osaksi mallia, koska niillä kyllä hallitaan mallia, mutta niitä myös hallitaan ja esitetään näkymässä.

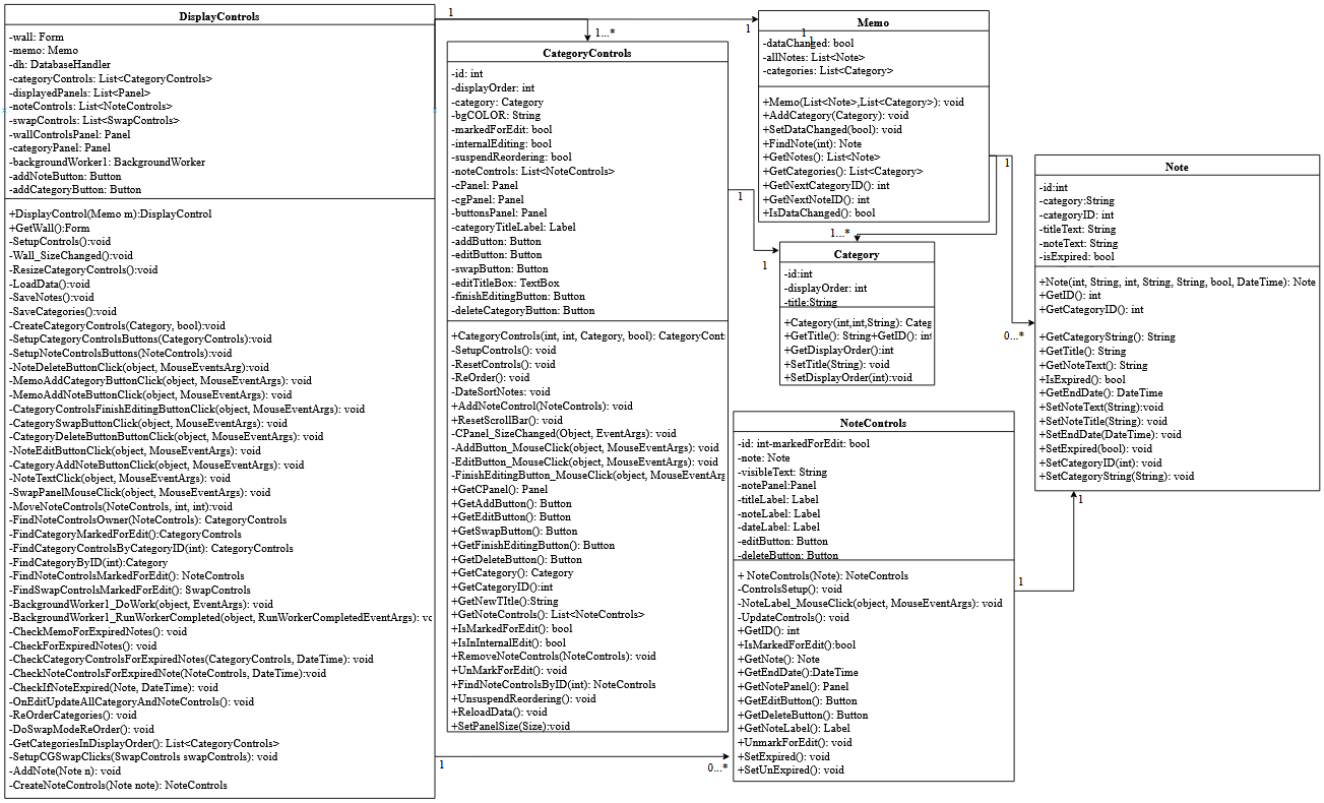


KUVIO 5. Ohjelman arkkitehtuuri (Pilvilampi 2021)

Tietokannan käsitetään olevan kuviossa 5 mallin ulkopuolella, koska sitä ei käytetä esittämään tietoa ohjelman toiminnan aikana, vaan ainoastaan käynnistäessä muodostamaan muistion merkinnät ja kategoriat. Tietokantaa hallitseva luokka on kuvattu osaksi mallia, koska ohjain näkee sen hakemat tiedot.

4.2 Luokkakaavio

UML-standardien kaaviot kuvaavat oliopohjaisen ohjelmiston rakennetta. Luokkakaaviosta (KUVIO 6) nähdään muistio-ohjelman luokkien ominaisuudet ja metodit, sekä miten luokkien oliot ovat yhteydessä toisiin, erityisesti mitkä hallitsevat toisiaan. Kuviosta 6 puuttuvat tilan vuoksi luokat ”AddFormPrompt”, ”AddCategoryPrompt” ”DatabaseHandler” ja ”SwapControls”.



KUVIO 6. Muistion luokkakaavio (Pilvilampi 2021)

4.3 Tietokantatiedoston mallinnus

Tietokantatiedoston tarkoitus on säilyttää käyttäjän muistio käyttökertojen välillä. Tietokanta tallentuu käyttäjän kovalevyllä sijaitseva tiedostoon, josta ohjelma muodostaa merkinnät ja kategoriat käynnistyksessä. Latauksen jälkeen ohjelma pitää muistion oliot listoissaan, jotta käyttäjä voi muokata esimerkiksi merkintää ilman, että se pitäisi joka kerta erikseen rakentaa tietokantatiedostosta.

Tietokannan tiedostotyyppi on tekstitiedosto (txt). Ohjelma rakentaa tietokantatiedoston taulukonomaisesti, jakaen tiedot riveihin ja sarakkeisiin. Yksi rivi edustaa yhtä merkintää tai kategoriaa, ja jokainen sarake tiettyä ominaisuutta. Sarakkeet erotetaan toisistaan puolipisteellä. Lukiessaan ohjelma havaitsee, milloin kukin solu päättyy ja yrittää asettaa luettua tietoa olioon. Alla on esimerkki yhdestä tietokantaan tallennetusta merkinnästä.

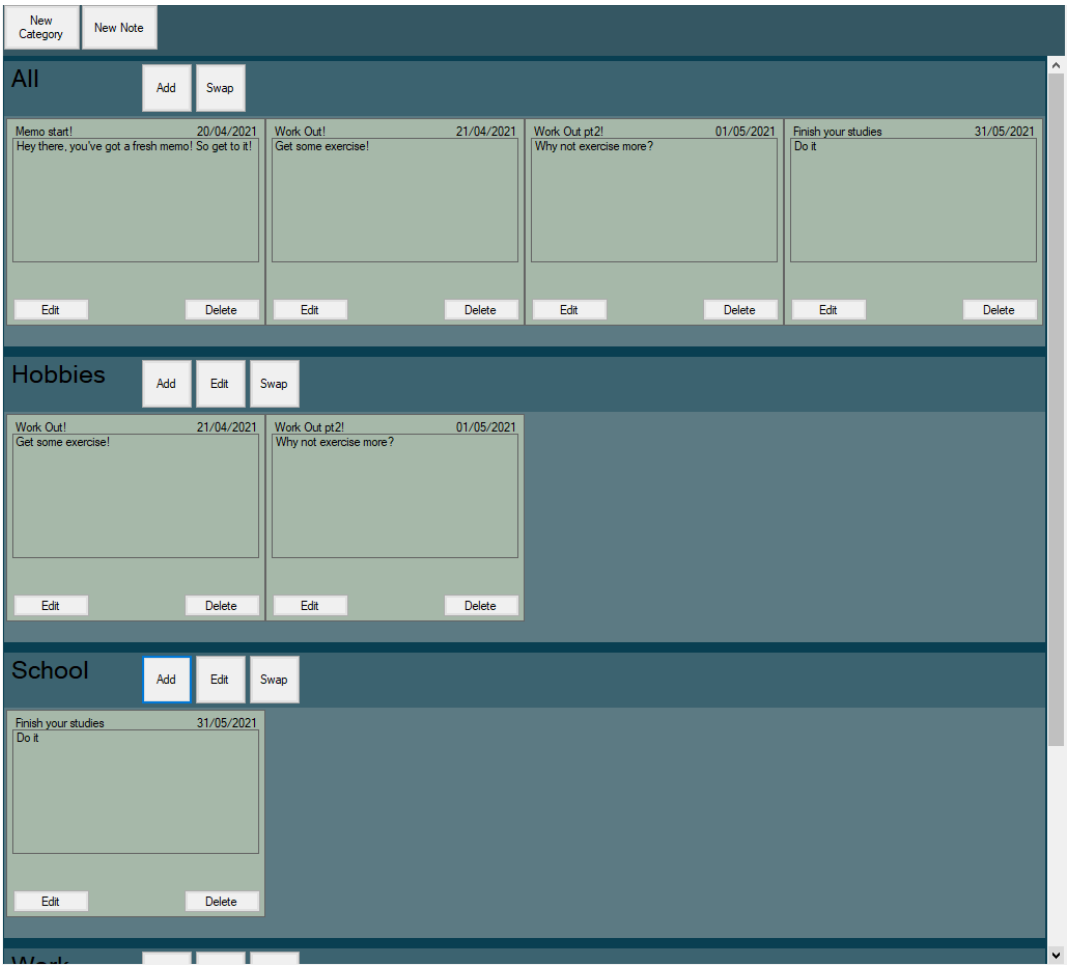
1;Hobbies;5;Memo start!;Hey there, you've got a fresh memo! So get to it!;False;12/04/2021 23.59.59

Solut sisältävät merkinnän tiedon järjestyksessä: merkinnän tunniste, kategorian nimi, kategorian tunniste, merkinnän otsikko, merkinnän viesti, merkinnän erääntymien totuusarvo ja merkinnän erääntymisaika.

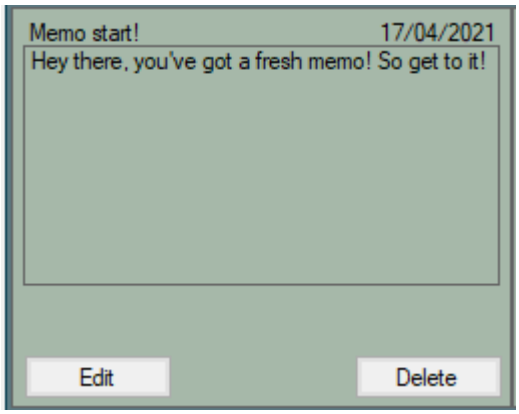
4.4 Käyttöliittymän suunnittelu

Tässä osiossa esitetään suunnittelun aikana eri prototyyppien avulla kehitelty lopullinen versio muistio-ohjelman käyttöliittymästä. Pohjaväriksi valittiin tummanvihreä, koska sen sävyillä saatiin hyvin erotettua eri käyttöliittymän osat toisistaan. Musta tekstikin erottuu hyvin.

Kuvasta 2 näkee, että jokaisella ohjelman komponentilla on oma sävynsä, ja esimerkiksi kuinka kategoriat erottuvat toisistaan raon tummemmalla vihreällä. Kirkkauden haluttiin tarkoittavan interaktiivisuutta, siksi painikkeiden värinä pidettiin perusasetuksena oleva harmaa.

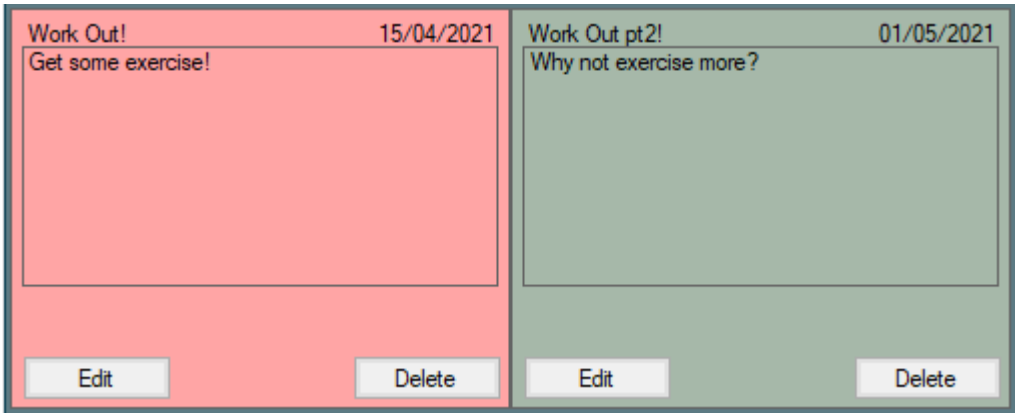


KUVA 2. Muistion päänäkymä (Pilvilampi 2021)



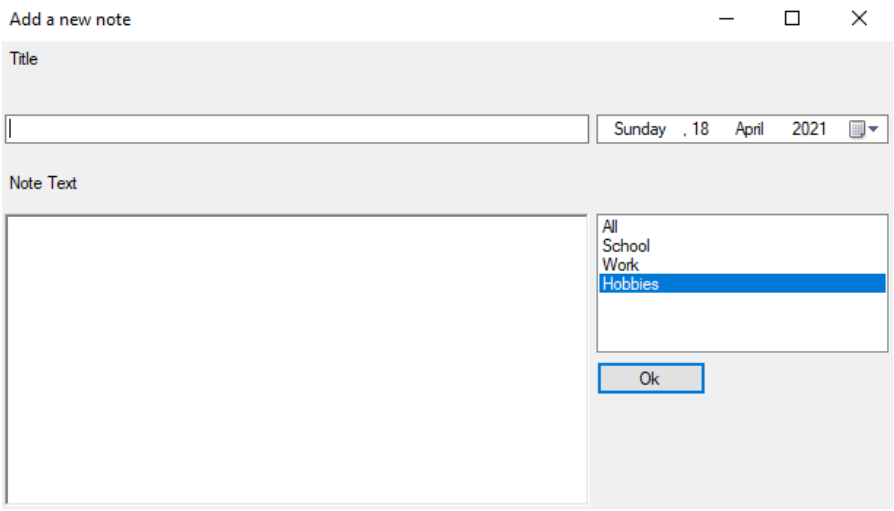
KUVA 3. Merkinnän esitys (Pilvilampi 2021)

Kuvassa 3 on esimerkki merkinnästä käyttöliittymässä. Ylävasemmalla oleva teksti on otsikko, jota vastapäätä on erääntymispäivä. Keskellä oleva rajattu tekstialue sisältää merkinnän viestin. Jos viestissä on liikaa merkkejä, katkaistaan rajan ylittävä teksti ja korvataan kolmella pisteellä. Käyttäjä voi lukea koko viestin napsauttamalla tekstialuetta. Merkinnän alaosassa olevat painikkeet sallivat muokkauksen ja poiston. Kuva 4 näyttää miten erääntynyt merkintä erottuu tavallisesta merkinnästä vaaleanpunaisella taustavärillään.



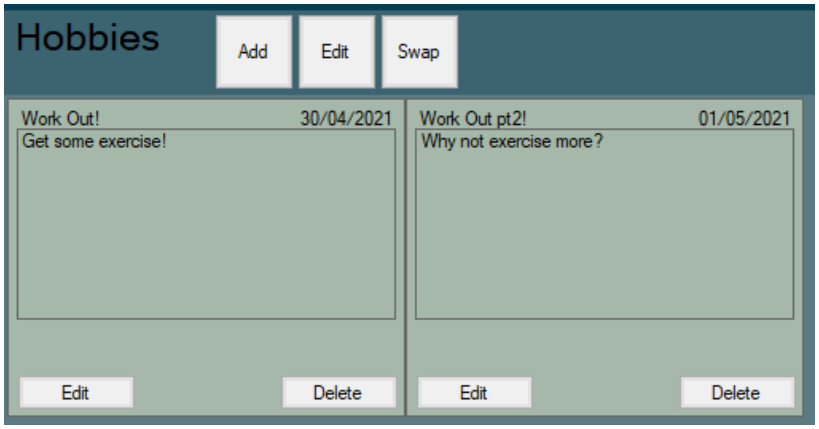
KUVA 4. Erääntynyt merkintä ja merkintä vierekkäin (Pilvilampi 2021)

Uuden merkinnän tekemiseen ja merkinnän muokkaamiseen on suunniteltu yksi lomake (KUVA 5). Muokatessa merkintää, tulee kentät täyttää olemassa olevilla arvoilla. Uutta luodessa otsikko- ja viestikentät ovat tyhjiä, mutta erääntymispäivä asetetaan seuraavaksi päiväksi, ja kategorian oletetaan oleva se, minkä merkinnänlisäyspainiketta käyttäjä napsautti. Kuvasta 5 nähdään, että käyttäjä on luomassa uutta merkintää ”Hobbies”-kategoriaan.

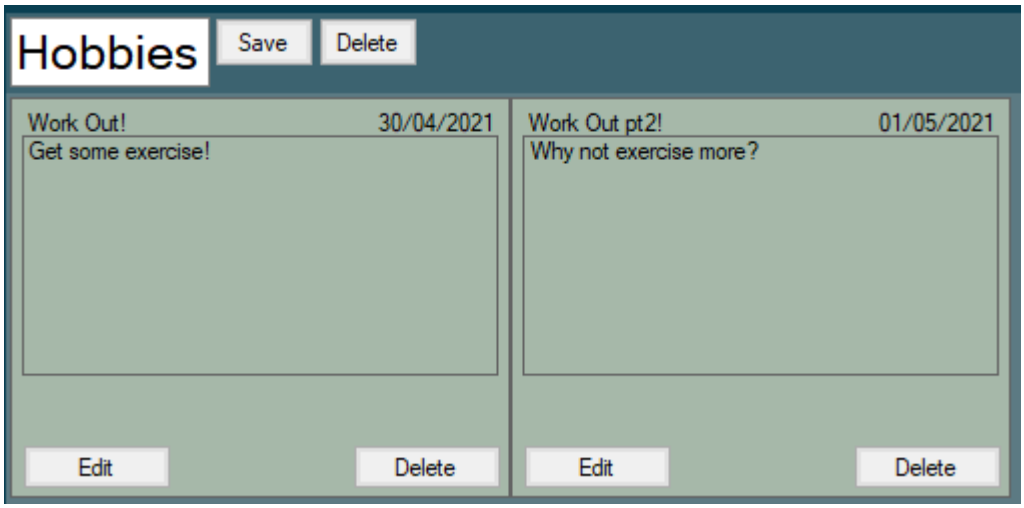


KUVA 5. Merkinnän muokkaus- ja lisäyslomake (Pilvilampi 2021)

Kategoriasta näkyy kuvan 6 kaltaisesti otsikko ja kategorian hallintaan käytettävät painikkeet. Iso teksti näyttää selkeästi mikä kategoria on kyseessä, ja vaaleampi tausta osoittaa tilaa merkinnöille. ”Edit”-painike muuttaa kategorian otsikkopaneelia niin, että mahdollistetaan kategorian muokkaus suoraan paneelissa (KUVA 7). Kun otsikkopaneeli muuttuu, voi käyttäjä poistaa tai uudelleennimetä kategorian. ”Delete”-painiketta napsauttaessa ohjelma kysyy, että käyttäjä haluaa poistaa kategorian.



KUVA 6. Kategorian esitys (Pilvilampi 2021)



KUVA 7. Kategorian muokkaaminen (Pilvilampi 2021)

Ohjelmassa käytettävien lomakkeiden vähentämiseksi suunniteltiin kategorian siirtämiseen tilanvaihto-toiminto, ns. ”SwapMode”. Kategorian ”Swap”-painikkeen napsautuksella käyttöliittymä muuttuu niin, että kategoriat korvataan paikan vaihdon mahdollistavilla kontrolleilla. Kuva 8 näyttää, miten valkoinen kontrolli osoittaa kategorian alkuperäisen sijainnin. Siniset kontrollit ovat tyhjiä paikkoja, jotka sallivat siirron esimerkiksi kahden kategorian väliin. Vihreät ovat muita kategorioita, ja niitä napsauttamalla käyttäjä voisi kuvassa 8 esitetyssä tapauksessa vaihtaa ”All”-kategorian paikkaa esimerkiksi ”Hobbies”-kategorian kanssa.

KUVA 8. SwapMode (Pilvilampi 2021)

5 TOTEUTUS

Muistio-ohjelman toteutukseen valittiin C#-ohjelmointikieli, sekä Visual Studio -kehitysympäristö. Valintaa perustellaan sillä, että ne ovat kehittäjälle entisestään tuttuja, ja kehitysympäristöön helposti liitettävä luokkakirjasto Windows Forms sopii hyvin yksinkertaisiin tietokoneohjelmiin.

Visual Studio on Microsoftin kehittämä integroitu kehitysympäristö (IDE) Windows-käyttöjärjestelmille. Se on ohjelmointityökalu, joka sisältää koodieditorin lisäksi graafisen editorin käyttöliittymien suunnitteluun. Visual Studio käyttää versionhallinnassa ensisijaisesti GitHub-palvelua. (Computer Hope 2019.)

Kehitysympäristön virheenkorjaaja työskentelee ohjelman testauksen ollessa käynnissä, kehittäjä voi nähdä koodin toimintaa yksityiskohtaisesti, kuten milloin ja miksi muuttujan arvo muuttuu. Kehittäjä voi hallita testausta asettamalla koodiin pysäkkejä, joihin testaus pysähtyy odottamaan. Virheenkorjaaja sallii myös koodin muokkaamisen testauksen aikana, kun testaaja ensiksi keskeyttää testauksen väliaikaisesti. (Microsoft 2019.)

Windows Forms on .NET -kehyksen osa, kokoelma koodikirjastoja, joka on erikoistunut käyttöliittymän kehittelyyn. Se tarjoaa mahdollisuuden käyttää Windowsin luonnollisia käyttöliittymäelementtejä ja grafiikkaa. Jokainen painike ja kontrolli on oma olionsa, näillä hallitaan ohjelman toimintaa niille koodattavien tapahtumien, kuten hiirennapsautusten avulla. (Techopedia 2021.)

5.1 Merkintöjen automaattinen tarkastaminen

Vaatimuksissa määriteltiin, että ohjelman tulisi automaattisesti tarkistaa merkinnät, ja muuttaa merkintäkontrolleja niin, että käyttäjälle näkyisi eräpäivän ohittaneet merkinnät. Erääntymispäivänä merkintä ei siis olisi vielä erääntynyt, vaan vasta keskiyön jälkeen. Kellonaika erääntymiselle asetetaan muokkauksessa ja luonnissa.

Kun ohjelma käynnistyy ja oliot sekä muut kontrollit on luotu, luo ohjelma taustatyöntekijäolion ”backgroundworker1” (KOODI 1). Tämä olio odottaa 5 sekuntia, jonka kuluttua se kutsuu tapahtuman

”BackgroundWorker1_RunWorkerCompleted”. Tämä tapahtuma kutsuu muistion tarkistavan metodin, jonka jälkeen tapahtuma luo uuden taustatyöntekijäolion.

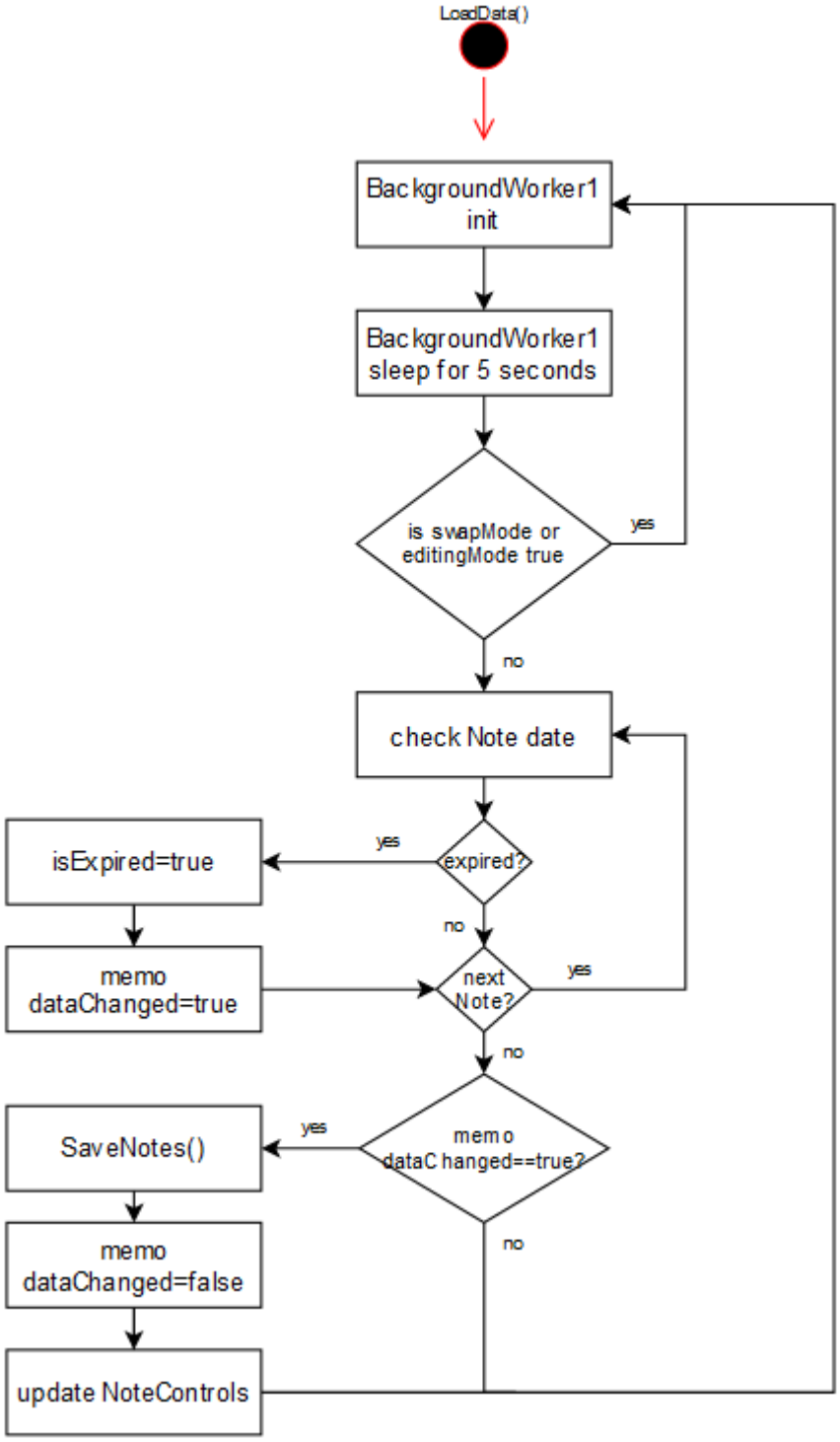
```
backgroundWorker1 = new BackgroundWorker();
backgroundWorker1.DoWork += new DoWorkEventHandler(Back-
groundWorker1_DoWork);
backgroundWorker1.RunWorkerCompleted += new RunWorkerCom-
pletedEventHandler(BackgroundWorker1_RunWorkerCompleted);
backgroundWorker1.RunWorkerAsync();
```

KOODI 1. Taustatyöntekijäolion luonti käynnistyksessä

Metodi ”CheckMemoForExpiredNotes” (KOODI 2) käy läpi merkinnät, verraten jokaisen eräpäiväarvoa metodin alussa laskettuun aikaan. Jos havaitaan erääntynyt merkintä, sen erääntymistotuusarvo asetetaan todeksi, ja metodi muuttaa muistiossa olevan totuusarvon ”dataChanged” todeksi. Metodin lopussa, jos tämä arvo on tosi, kutsutaan merkintöjen tallennus, sekä käyttöliittymän päivitys. Koko käyttöliittymän päivitys on helppo ratkaisu välttää yksittäisten kontrollien etsimistä päivitystä varten. Kuvio 7 esittää tarkastuksen toimintaa graafisesti.

```
private void CheckMemoForExpiredNotes(){
DateTime now = DateTime.Now;
foreach(Note note in memo.GetNotes())
{
CheckIfNoteExpired(note, now);
}
if (memo.IsDataChanged()){
SaveNotes();
OnEditUpdateAllCategoryAndNoteControls();
memo.SetDataChanged(false);
}
}
```

KOODI 2. ”CheckMemoForExpiredNotes”-metodi



KUVIO 7. Päivityksen logiikkakaavio (Pilvilampi 2021)

5.2 Ohjelman käynnistys

Ohjelman logiikka alkaa ”Program”-tiedostosta (KOODI 3). Tämän tiedoston ohjeiden mukaan luodaan ensiksi tietokantaolio, joka käsketään muodostamaan merkintä- ja kategoriaoliot tietokantatiedoston sisällöstä ja antamaan ne muistio-oliolle. Muistio-olio annetaan seuraavaksi luotavalle hallintaolille, joka luo ohjelman kontrollit käyttäjälle. Lopuksi annetaan ohjelmalle käsky avata luodun hallintaolion lomake.

```
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);

DatabaseHandler dh = new DatabaseHandler();
Memo memo = new Memo(dh.LoadNotes(),dh.LoadCategories());
DisplayControl dc = new DisplayControl(memo);

Application.Run(dc.GetWall());
```

KOODI 3. ”Program”-tiedoston koodi

”LoadData”-metodin käydessä ohjelma luo kontrollit jokaiselle kategorialle ja merkinnälle. Metodi toimii luomalla muistio-oliosta haetuille olioille kontrollit foreach-silmukoilla (KOODI 4) kutsumalla metodia, joka myös asettaa luotujen kontrollien painikkeille tapahtuman sekä hallintaoliossa (KOODI 5), että esimerkiksi kategoriakontrollioliossa.

```
foreach(Category c in categories)
{
CreateCategoryControls(c);
}
```

KOODI 4. Esimerkki ”LoadData”-metodin foreach-silmukoista

```
private void SetupCategoryControlButtons(CategoryControls
cc){
cc.GetAddButton().MouseClick += CategoryAddNoteButtonClick;
cc.GetSwapButton().MouseClick += CategorySwapButtonClick;
cc.GetFinishEditingButton().MouseClick += CategoryCon-
trolsFinishEditingButtonClick;
cc.GetDeleteButton().MouseClick += CategoryDeleteButtonBut-
tonClick;}
```

KOODI 5. Painikkeiden napsautustapahtumien asettamismetodi

5.3 Merkinnän luominen ja muokkaaminen

Merkintä luodaan napsauttamalla joko ohjelman ylävalikon ”New Note”-painiketta, tai kategorian ”Add”- painiketta. Näille painikkeille on asetettu eri merkinnänluonnin aloittavat hiirennapsautusta-pahtumat. Ainut ero, joka välittyy ”AddFormPrompt”-luokan oliolle, on kategorian paikka ohjelman listassa. Kategorian painiketta napsauttaessa välittyy kategorian paikan numero, kun taas ylävalikon painikkeella valitaan automaattisesti ensimmäinen, eli ”All”-kategoria.

Koodi 6 on metodi, joka kutsutaan ylävalikon painikkeelle annetusta tapahtumasta. Ensimmäiseltä riviltä nähdään, että merkintäluontimetodi keskeytetään, jos kategoriansiirtotila on käynnissä. Metodi luo lomakeolion, jonka sisältämillä kontrolleilla käyttäjä luo merkinnän. Metodi luo merkinnän vain, jos käyttäjä on napsauttanut lomakeolion ”OK”-painiketta, ja lomakeolio on hyväksynyt käyttäjän syöttämän tiedon.

```
if (swapMode) return;
AddFormPrompt addForm = new AddFormPrompt(categoryControls,
0);
editingMode = true;
if (addForm.GetForm().ShowDialog() == DialogResult.OK &&
addForm.IsDataReady()){
Note newNote = new Note(
memo.GetNextNoteID(),
memo.GetCategories()[addForm.GetCategoryListBoxIn-
dex()].GetTitle(),
memo.GetCategories()[addForm.GetCategoryListBoxIn-
dex()].GetID(),
addForm.GetNoteTitle(),addForm.GetNoteText(),false,
addForm.GetEndDate());
AddNote(newNote);
SaveNotes();}
editingMode = false;
```

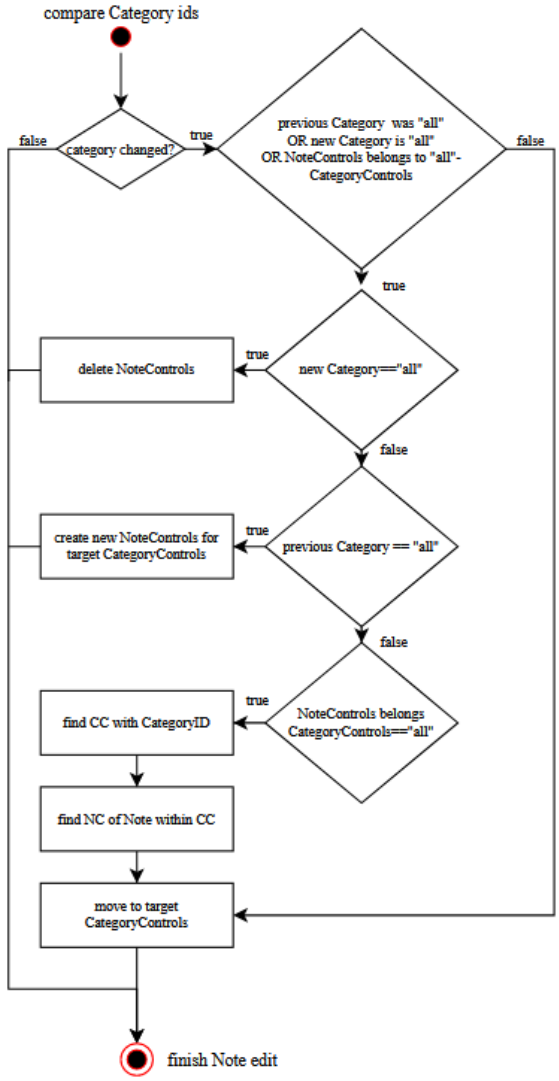
KOODI 6. Ylävalikon merkinnänluontipainikkeen tapahtuman kutsuman metodin koodi

Ehtosilmukan sisällä metodi (KOODI 6) luo uuden merkintäolion hakemalla käyttäjän syöttämät tiedot lomakeoliolta. Merkinnän avainarvo lasketaan automaattisesti kutsumalla muistio-olion ”GetNextNoteID”-metodi, joka hakee olemassa olevien merkintöjen suurimman avainarvon, ja palauttaa sitä yhdellä kasvatetun numeron. Ehtosilmukan lopussa kutsutaan ”AddNote”-metodia luomaan merkintäoliolle kontrollit käyttöliittymään, ja kutsutaan myös merkintöjen tallennus tietokantaan.

Muokatessa tai lisätessä merkintöjä, tulee asettaa totuusarvo ”editingMode” todeksi, jotta merkintöjä automaattisesti tarkastava taustatyöntekijä ei keskeytä tai muuten haittaa ohjelman toimintaa. Tämä arvo tulee asettaa takaisin epätodeksi muokkaus- ja lisäystoimintojen lopussa.

Kun käyttäjä napsauttaa merkinnän muokkauspainiketta, merkintää edustavan ”NoteControls”-olion sisällä ensiksi painikkeelle asetettu tapahtuma muuttaa ”markedForEdit”-totuusarvon todeksi. Tämä arvo merkitsee olion muokkauksen kohteeksi. Muokkausmetodi, joka käynnistyy muistionhallinnassa painikkeelle toiseksi napsaustapahtumaksi asetetusta tapahtumasta, etsii merkintää edustavan, muokattavaksi valitun merkintäkontrolliolion listasta. Tämän edustaman merkinnän tiedot välitetään lomakeoliolle. Lomakeolio täyttää tekstilaatikot merkinnän tiedoilla ja valitsee eräpäivän valintakontrollin arvoksi merkinnän nykyisen eräpäivän. Merkinnän eräpäivä ei muokkauksen jälkeen voi kuitenkaan olla muokkausajankohtaa varhaisempi, joten erääntynyt merkintä saa automaattisesti uuden eräpäivän, eikä ole siksi enää erääntynyt.

Jos käyttäjä vaihtaa merkinnän kategoriaa muutoksessa, kutsutaan sitä varten kehitetty merkintäkontrollien siirtometodi ” MoveNoteControls” (LIITE 1). Koska yhdellä merkinnällä voi olla edustavat merkintäkontrollit kahden kategorian kohdalla, eli valitun kategorian lisäksi aina ”All”-kategoriassa, täytyi kehittää logiikka, joka paitsi säilyttää kaikissa tapauksissa ”All”-kategoriassa olevan merkintäkontrollin, mutta sallii myös tästä merkintäkontrollista merkinnän toisen merkintäkontrollin siirron, poiston ja luonnin. Kuvio 8 esittää siirtologiikan graafisesti.



KUVIO 8. Merkintäkontrollien siirron logiikkakaavio (Pilvilampi 2021)

5.4 Merkinnän poistaminen

Kuten muokkaustapauksessa, merkintäkontrollin ”Delete”-painikkeen ensimmäinen tapahtuma merkitsee merkintäkontrollin muokkauksen kohteeksi. Poistopainikkeen tapahtuma (KOODI 7) etsii tämän merkintäkontrollin, ja poistaa sen sisältämän merkintäolion merkintäkontrollit listoista. Myös itse merkintäolio poistetaan muistio-olion listasta, jonka jälkeen merkintä katoaa tietokannasta, kun sitä ei välitetä tietokantaoliolle tallennuksessa.

```
NoteControls nc = FindNoteControlsMarkedForEdit();
List<NoteControls> ncsToDelete = new List<NoteControls>();
Note noteToDelete=nc.GetNote();
foreach(NoteControls ncs in noteControls)
{
if (nc.GetNote() == ncs.GetNote()) ncsToDelete.Add(ncs);
}
foreach(NoteControls ncs in ncsToDelete)
{
FindNoteControlsOwner(ncs).RemoveNoteControls(ncs);
noteControls.Remove(ncs);
ncs.UnmarkForEdit();
}
memo.GetNotes().Remove(noteToDelete);
SaveNotes();
```

KOODI 7. Poistopainikkeen tapahtuma

5.5 Kategorian luominen

Kuten merkintäkin, kategoria luodaan lomakkeella, joka esitetään, kun käyttäjä napsauttaa ylävalikossa olevaa ”New Category”-painiketta Tämä lomake pyytää käyttäjältä vain otsikkoa kategorialle. Syötetty otsikko tarkastetaan, sen tulee olla vähintään yhden, mutta enintään kahdeksan merkin pituinen. Otsikko ei saa olla ”All”, koska toteutuksessa ennustettiin mahdollisia ongelmia tietokannan latauksen aikana suoritettavassa virheentarkistuksessa.

Painiketapahtumassa (KOODI 8) lomakeolio luodaan, ja se käynnistetään ”ShowDialog”-komennolla. Kun kategorian otsikko on syötetty hyväksytysti, luodaan kategoriaolio ehtosilmukassa. Kategoriaolille annetaan järjestysnumero, joka on viisi kertaa kategorioitten lukumäärä. Tällä varmistetaan, että luotu kategoria esitetään käyttöliittymässä alimpana.

```
if (swapMode) return;
AddCategoryPrompt acp = new AddCategoryPrompt();
Console.WriteLine(acp.ShowDialog());
Category newCategory;
if (acp.IsDataReady()){
int newID = memo.GetNextCategoryID();
newCategory = new Category(newID, categoryControls.Count *
5,acp.GetCategoryTitle());
memo.AddCategory(newCategory);
CreateCategoryControls(newCategory);
categoryControls[categoryControls.Count - 1].UnsuspendReor-
dering();
SaveCategories();
ResizeCategoryControls();
ReOrderCategories();}
```

KOODI 8. Kategorianluonnin painiketapahtuman koodi

Uusi kategoriakontrolli on tilassa, jossa sen ei ole sallittu järjestellä merkintäkontrolleja. Tämä tila on kehitetty siksi, että käynnistäessäkin kategoriakontrollit järjestävät sisältönsä turhasti uudelleen joka kerta, kun niihin lisätään merkintäkontrolleja. Kategoriakontrollioliolle pitää siis käyttäjän aloittaman luonnin jälkeen antaa lupa järjestellä kutsumalla sen ”UnsuspendReordering”-metodi.

5.6 Kategorian muokkaaminen ja poistaminen

Kategorian muokkaus ja poisto tapahtuu napsauttamalla ensiksi kategoriakontrollin ”Edit”-painiketta. Painikkeen tapahtuma muuttaa kategoriakontrollissa totuusarvon ”internalEditing” todeksi, ja kutsuu kategoriakontrollin kontrollien uudelleenjärjestyksen (KOODI 9). Tämä muuttaa kategoriakontrollin yläosan sisältöä, paljastaen uudet kontrollit kategorian muokkaukselle. Käyttäjä voi syöttää tekstikenttään uuden nimen, ja tallentaa viereisellä tallennuspainikkeella tietokantaan.

```
buttonsPanel.Controls.Clear();
if (!internalEditing){
buttonsPanel.Controls.Add(categoryTitleLabel);
buttonsPanel.Controls.Add(addButton);
if (GetCategory().GetID() != 0)
buttonsPanel.Controls.Add(editButton);
else swapButton.Location = new Point(addButton.Loca
tion.X+addButton.Width+2,2);
buttonsPanel.Controls.Add(swapButton);
}
else{
buttonsPanel.Controls.Add(editTitleBox);
buttonsPanel.Controls.Add(finishEditingButton);
buttonsPanel.Controls.Add(deleteCategoryButton);
}
```

KOODI 9. Kategoriakontrollin uudelleenjärjestysmetodi ”ResetControls”

Poistopainikkeen tapahtuma (KOODI 10) pyytää dialogilla käyttäjää varmistamaan, haluaako hän poistaa kategorian. Myöntävän vastauksen vastaanotettuansa tapahtuma käy läpi jokainen kategoriakontrollissa sijaitsevan merkintäkontrollin merkintä, ja asettaa niille kategoriaksi ensimmäisen, eli ”All”-kategorian. Lopuksi kutsutaan merkintöjen tallennus ja kategoriakontrollien uudelleenjärjestys.

```
editingMode = true;
CategoryControls cc = FindCategoryMarkedForEdit();
DialogResult dialogResult =
MessageBox.Show("Are you sure you want to delete category "
+cc.GetCategory().GetTitle()+"?","Confirm",MessageBoxBut-
tons.YesNo);
if (dialogResult.Equals(DialogResult.Yes)) {
foreach(NoteControls nc in cc.GetNoteControls()){
nc.GetNote().SetCategoryID(0);
nc.GetNote().SetCategoryString(categoryControls[0].GetCate-
gory().GetTitle());
noteControls.Remove(nc); }
categoryControls.Remove(cc);
memo.GetCategories().Remove(cc.GetCategory());
SaveCategories();
SaveNotes();
ReOrderCategories();}
editingMode = false;
```

KOODI 10. Kategorianpoistopainikkeen tapahtuma ”CategoryDeleteButtonButtonClick”

5.7 Kategorian siirtäminen

Sivulla 20 esitelty kategoriansiirtotila eli ”SwapMode”, aloitetaan napsauttamalla kategoriakontrollin ”Swap”-painiketta. Painikkeen tapahtuma asettaa ohjelman ”swapMode”-totuusarvon todeksi, ja kutsuu kategorioitten uudelleenjärjestelymetodin. Tässä metodissa tyhjennetään ensiksi lista, josta ohjelma lisää kategoriakontrollien paneelit käyttöliittymään. Sen jälkeen, koska kategoriansiirtotila on käynnissä, kutsutaan metodi ”DoSwapModeReOrder”, joka luo kategorian siirron mahdollistavat kontrollit. Tämä metodi lisää luotujen ”SwapControls”-olioiden paneelit listaan, jotka uudelleenjärjestelymetodissa (KOODI 11) lisätään ja asetellaan käyttöliittymään.

```
foreach (Panel p in displayedPanels)
{
int padding = 10;
if (!hasRun) {  padding = 5; hasRun = true; }
p.Location = new Point(0, lastPanelLocation.Y + lastPanel-
Size.Height + padding);
categoryPanel.Controls.Add(p);
lastPanelLocation = p.Location;
lastPanelSize = p.Size; }
```

KOODI 11. Paneelien lisäys ja asettelu metodissa ”ReOrderCategories”

Metodissa (LIITE 2), jossa luodaan kategorianvaihtotilan kontrollit, listataan ensiksi kategoriakontrollit esitysjärjestyksessä kutsumalla ”GetCategoriesInDisplayOrder”-metodi. Foreach-silmukassa toistetaan jokaiselle listan kategoriakontrollille samat toiminnot.

Silmukassa luodaan ensiksi tyhjää paikkaa edustava kontrolli, jolla voidaan siirtää kategoria toisen eteen, jälkeen tai kahden muun kategorian väliin. Tämän jälkeen ehtosilmukassa tutkitaan, edustaako foreach-silmukan tällä kierroksella käsiteltävä kategoriakontrolli siirrettävänä olevaa kategoriaa, tai jotain toista. Ensimmäisessä tapauksessa luodaan kontrolli, jota napsauttaessa kategorian paikka pysyy samana. Napsauttaessa toisessa tapauksessa luotua kontrollia, kategoriat vaihtavat paikkaa keskenään. Siirtotila päättyy, sama mitä kontrollia napsautetaan, ja ohjelman palautuu perusnäkymään.

6 POHDINTA

Työn tavoite oli kehittää suunniteltu muistio-ohjelma määriteltyjen vaatimusten perusteella seuraten mahdollisimman montaa ohjelmistoprojektin kuvattua vaihetta. Lopulta määrittely, suunnittelu ja toteutus olivat ainoat projektin läpikäydyt vaiheet.

Analysoimalla esitetty ohjelman idea saatiin vaatimukset määriteltyä ja dokumentoitua kuvioina ja tekstinä. Suunnittelussa ohjelman lopullinen rakenne ja ulkonäkö mallinnettiin käyttämällä prototyyppitestausta. Ohjelma valmistui onnistuneesti toteutuksessa suunnittelu- ja määrittelyvaiheista saatujen mallien avulla.

Testausta on tapahtunut paljon kehityksen aikana, mutta siitä ei luotu dokumentteja eikä muodostettu lukua tähän raporttiin. On todettu, että ohjelma toimii ja täyttää vaatimukset, mutta todisteen puute haittaa tämän asian uskottavuutta.

Toimitusvaihetta ei tehty, koska ohjelma toimii jo kohdeympäristössä, ja sitä käyttää jo ainut siitä kiinnostunut käyttäjä. Muistio-ohjelman toimittaminen toisille ja ohjeistaminen ohjelman käyttöön on toki mahdollista, mutta ei toteutunut vielä työn aikana. Ohjelman asentamista varten on tehty ohjeet (LIITE 3). Suunnitteluluku ja etenkin luku 4.4 voivat toimia käyttöohjeena.

Ohjelmaa voi kehittää, ja siinä on varmasti paranneltavaa. Korjausjonoon voi odottaa tuntemattomia ongelmia, johtuen siitä, että perusteellista testausta ei tehty. Raportin avulla voi kehittäjä saada paremman kuvan ohjelman rakenteesta, mutta koodirivien määrä on suuri, ja kokonaisuuden ymmärtäminen voi viedä paljon aikaa. Yksi kehitysidea on toiminto, joka laajentaisi yhden kategorian näkymän niin, että käyttöliittymässä näkyy vain valittu kategoria merkintöineen.

Olen yrittänyt kirjoittaa raportin mahdollisimman tarkasti ohjeiden mukaan. Pyrin erityisesti siihen, että käytin aina esimerkiksi käyttöliittymän painikkeesta puhuessani samoja löytämiäni termejä, kuten ”napsautus” ja ”tapahtuma”. Koodinselostus on minulle uutta. Toivon, että onnistuin siinä hyväksyttävästi.

LÄHTEET

AltexSoft. 2018. Functional and Nonfunctional Requirements: Specification and Types. Saatavissa: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/. Viitattu 19.4.2021.

Azarian, I. 2013. Key Phases of Software Development Projects. Saatavissa: https://www.segue-tech.com/key-phases-software-development/. Viitattu 19.4.2021.

Computer Hope. 2019. Visual Studio. Saatavissa: https://www.computerhope.com/jargon/v/visual-studio.htm. Viitattu 9.4.2021.

Design Phase. FFIEC. Saatavissa: https://ithandbook.ffiec.gov/it-booklets/development-and-acquisition/development-procedures/systems-development-life-cycle/design-phase.aspx. Viitattu 9.4.2021.

Development Phase. FFIEC. Saatavissa: https://ithandbook.ffiec.gov/it-booklets/development-and-acquisition/development-procedures/systems-development-life-cycle/development-phase.aspx. Viitattu 9.4.2021.

Elysium Academy Support. 2017. What are the Software Development Life Cycle (SDLC) phases? Saatavissa: https://www.linkedin.com/pulse/what-software-development-life-cycle-sdlc-phases-private-limited. Viitattu 9.4.2021.

Estimating Software Engineering Effort: Project and Product Development Approach. AltexSoft. Saatavissa: https://www.altexsoft.com/whitepapers/estimating-software-engineering-effort-project-and-product-development-approach/. Viitattu 9.4.2021.

Goran, J. 2019. What is SDLC? Phases of Software Development, Models, & Best Practices. Saatavissa: https://phoenixnap.com/blog/software-development-life-cycle. Viitattu 9.4.2021.

Implementation Phase. FFIEC. Saatavissa: https://ithandbook.ffiec.gov/it-booklets/development-and-acquisition/development-procedures/systems-development-life-cycle/implementation-phase.aspx. Viitattu 9.4.2021.

Initiation Phase. FFIEC. Saatavissa: https://ithandbook.ffiec.gov/it-booklets/development-and-acquisition/development-procedures/systems-development-life-cycle/initiation-phase.aspx. Viitattu 9.4.2021.

Leppäniemi, P. Ohjelmistoprojektit. Saatavissa: ohjelmistotuotanto.panuleppaniemi.com/ohjelmisto-projektit/. Viitattu 9.4.2021.

Maintenance Phase. FFIEC. Saatavissa: https://ithandbook.ffiec.gov/it-booklets/development-and-acquisition/development-procedures/systems-development-life-cycle/maintenance-phase.aspx. Viitattu 9.4.2021.

Microsoft. 2019. First look at the Visual Studio Debugger. Saatavissa: https://docs.microsoft.com/en-us/visualstudio/debugger/debugger-feature-tour?view=vs-2019. Viitattu 10.4.2021.

Performance Lab. Importance of Software testing in SDLC. Saatavissa: https://performancelabus.com/software-testing-importance-sdlc/. Viitattu 10.4.2021.

Planning Phase. FFIEC. Saatavissa: https://ithandbook.ffiec.gov/it-booklets/development-and-acquisition/development-procedures/systems-development-life-cycle/planning-phase.aspx. Viitattu 9.4.2021.

Sourour, B. 2017. Putting comments in code: the good, the bad, and the ugly. https://www.freecodecamp.org/news/code-comments-the-good-the-bad-and-the-ugly-be9cc65fbf83/. Viitattu 17.4.2021.

Techopedia. Windows Forms. Saatavissa: https://www.techopedia.com/definition/24300/windows-forms-net. Viitattu 9.4.2021.

Visual Paradigm. What is Model-View and Control? Saatavissa: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-model-view-control-mvc/. Viitattu 19.4.2021.

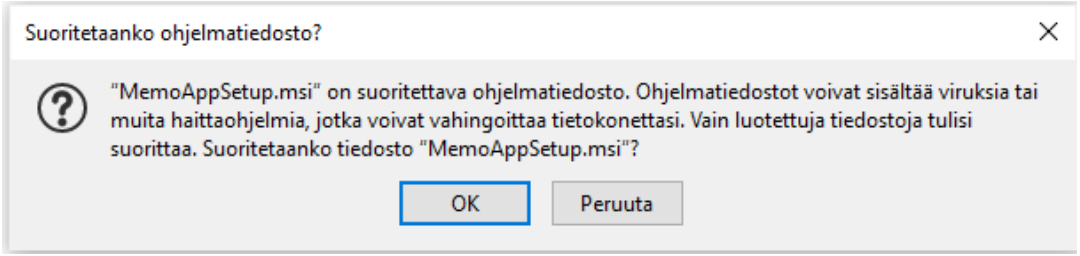
```
if (originCC != categoryControls[0] && targetCC != catego-
ryControls[0] && ownerCC != categoryControls[0])
{
System.Diagnostics.Debug.WriteLine("Not owned by 0");
originCC.RemoveNoteControls(nc);
targetCC.AddNoteControl(nc);
categoryControls[0].ReOrder();
}

else if (targetCC==categoryControls[0])
{
System.Diagnostics.Debug.WriteLine("Target was 0");
ownerCC.RemoveNoteControls(nc);
noteControls.Remove(nc); //remove from notecontrols list}
else if (originCC==categoryControls[0]){
System.Diagnostics.Debug.WriteLine("Origin was 0");
NoteControls newNC = CreateNoteControls(nc.GetNote());
targetCC.AddNoteControl(newNC);
}

else if(ownerCC==categoryControls[0])
{
System.Diagnostics.Debug.WriteLine("Owned by 0");
CategoryControls ownerCCC = FindCategoryControlsByCatego-
ryID(
nc.GetNote().GetCategoryID());
NoteControls ncToMove = ownerCCC.FindNoteControls-
ByID(nc.GetNote().GetID());
System.Diagnostics.Debug.WriteLine(ownerCCC.GetCate-
gory().GetTitle()+" "+targetCC.GetCategory().GetTitle());
ownerCCC.RemoveNoteControls(ncToMove);
targetCC.AddNoteControl(ncToMove);
}
```

```
foreach (CategoryControls cg in cgInOrder){
String s = String.Format("Click to place category in this
spot");

SwapControls swap1 = new SwapControls(

categoryControls[0].GetCPanel().Size, s, newOrder, null,
false);

SetupCGSwapClicks(swap1);
newOrder++;

SwapControls swap2 = null;

if (cg.IsMarkedForEdit()){
s = String.Format("This is where the category {0} is lo-
cated",
cg.GetCategory().GetTitle());
cg.GetCategory().SetDisplayOrder(newOrder);
swap2 = new SwapControls(
categoryControls[0].GetCPanel().Size, s, newOrder, cg,
true);
SetupCGSwapClicks(swap2);
}
else{
s = String.Format("Click here to swap with category {0}",
cg.GetCategory().GetTitle());
cg.GetCategory().SetDisplayOrder(newOrder);
swap2 = new SwapControls(categoryCon-
trols[0].GetCPanel().Size, s, newOrder, cg, false);
SetupCGSwapClicks(swap2);}
newOrder++;
swapControls.Add(swap1);
swapControls.Add(swap2);
}
```

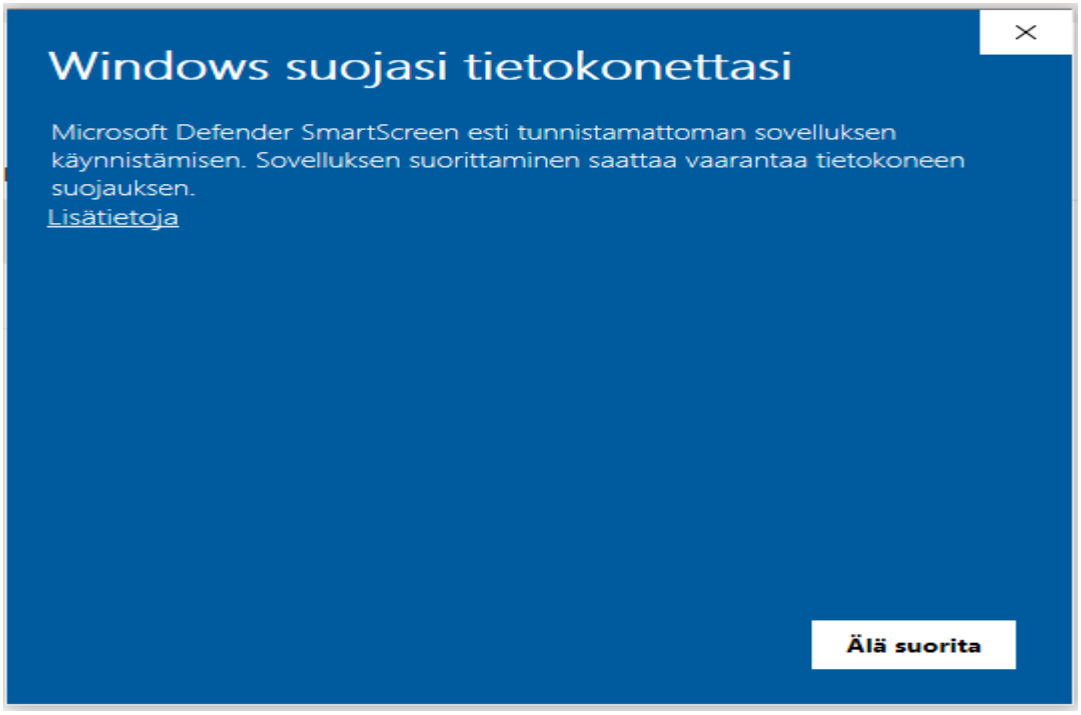
MUISTIO-OHJELMAN ASENNUSOHJEET

Tämä dokumentti ohjaa käyttäjää MemoApp-ohjelman asennuksessa. Ohjelma on kehitetty Windows 10-käyttöjärjestelmässä käytettäväksi, eikä sitä ole testattu vanhemmilla Windowsin versioilla. Ohjelma tarvitsee asennukseen vain noin 150 kilobittiä tilaa levyllä. Asentaaksesi ohjelman, lataa ensiksi sen asennuspaketti osoitteesta: https://drive.google.com/file/d/12mcU6O0cNkyPbAbF5-9QsrniaKETNgX0/view?usp=sharing

Käynnistäessäsi asennusohjelman MemoAppSetup.msi, voi Windows näyttää varoituksia:



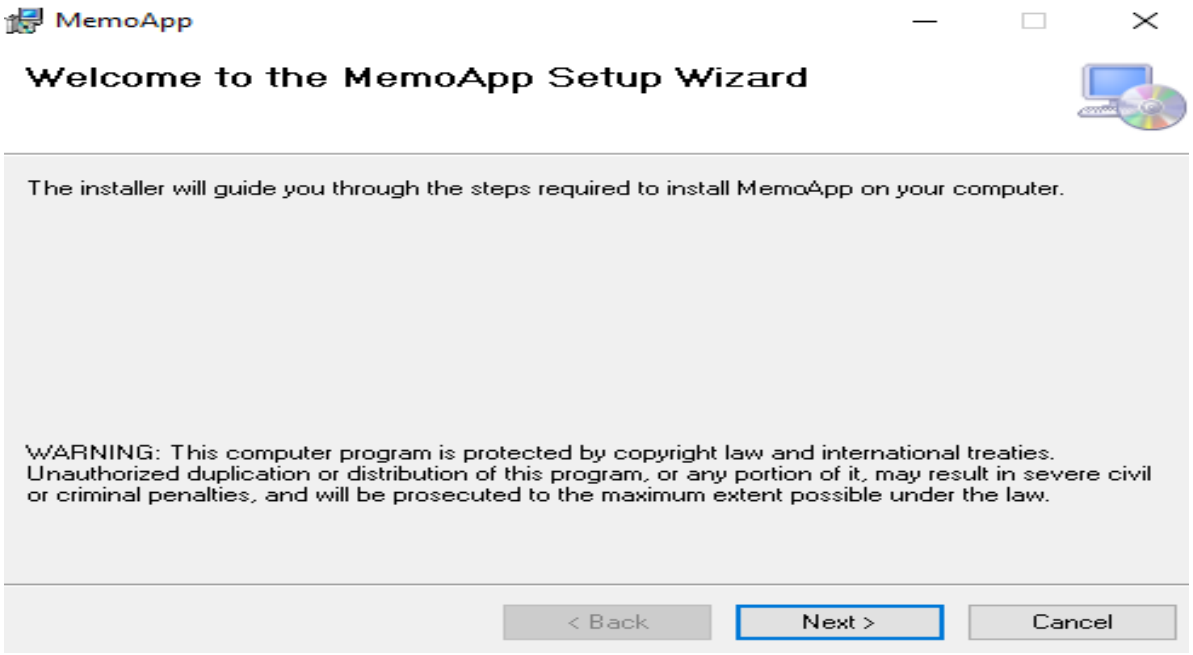
Paina OK-painiketta.



Paina ”Lisätietoja”, jolloin varoitus paljastaa painikkeen ”Suorita joka tapauksessa”, paina tätä.

Windows voi pyytää vielä varmistamaan, että annetaanko asennusohjelmalle lupa tehdä muutoksia koneeseen. Haluttu muutos on ohjelman toimintaan tarvittavan .NET ohjelmistokomponenttikirjaston asennus.

Annettuasi asennusohjelmalle luvan toimia, se avautuu:



Paina Next, eli Seuraava-painiketta. Seuraavalla sivulla voit muuttaa ohjelman asennuspaikkaa, sekä päättää asennetaanko ohjelma kaikille käyttäjille.

Asennuksen jälkeen ohjelman voi käynnistää käynnistysvalikosta löytyvällä pikakuvakkeella. Muistion merkinnät ja kategoriat tallentuvat Tiedostot-kansioon tekstitiedostoina, tiedostonimillä ”memoNotes” ja ”memoCategories”.