Varausjärjestelmä

Kaius Karvo

791597

Bioinformaatioteknologia

1. **Yleiskuvaus**

   Projektin tavoitteena oli luoda varausjärjestelmä palloiluhallille keskivaikean työn vaatimuksien puitteissa. Varausjärjestelmälle on luotu intuitiivinen graafinen käyttöliittymä, jonka taustalle on ohjelmoitu itse varausjärjestelmä. Varausjärjestelmän avulla käyttäjä pystyy luomaan eri pituisia varauksia itselleen palloiluhallin eri kentiltä. Lisäksi käyttäjä pystyy valitsemaan, haluaako hän mukaan mailavuokran vai pärjääkö ilman. Varausjärjestelmä tallentaa käyttäjän tiedot ja myöhemmin käyttäjä pystyy hakemaan varauksen tekijöiden nimillä heidän varaushistoriansa. Lisäksi varausjärjestelmä laskelmoi pallokentän varaukselle hinnan, joka näkyy käyttöliittymässä dynaamisesti. Loppujen lopuksi projekti vastasi vaativan vaikeustason projektia.

2. **Käyttöohje**

   Ohjelman käynnistys tapahtuu ajamalla main-tiedosto. Tiedoston ajettua näytölle aukeaa ajanvarausjärjestelmä-ikkuna. Samalla ohjelma lukee data.txt tiedoston ja jos sen lukemisessa esiintyy häiriöitä, ne tulostuvat IDE:n konsoliin.

   Ohjelman käyttäminen

   Varauksen luominen

   Kun käyttäjä haluaa luoda varauksen, hänen täytyy täyttää aluksi kaikki vapaana olevat kentät. Kenttien täyttöjärjestyksellä ei ole väliä. Ylhäältä alas käyden ensimmäinen kenttä on Laji-valikko. Laji-valikossa on palloiluhallin kaikki kentät ja niiden tuntihinnat nähtävillä. Laji-valikosta käyttäjä valitsee itselleen mieluisan lajin painamalla. Tämän jälkeen on Nimi-kenttä, johon käyttäjä kirjoittaa oman etu- ja sukunimen erikseen. Mallia ”Teemu Teekkari”. Tämä nimi tallentuu julkiseen data.txt tiedostoon, joten halutessaan käyttäjä voi käyttää kaksiosaista nimimerkkiä. Tämän jälkeen on Sähköposti- kenttä, johon käyttäjä kirjoittaa oman sähköpostiosoitteensa. Seuraavaksi on Puhelinnumero-kenttä, johon käyttäjä jälleen saa kirjoitettua oman puhelinnumeronsa. Seuraavaksi on Varauksen päivämäärä- kenttä. Käyttäjä saa valittua mieluisan päivämäärän Ajanvarausjärjestelmä-ikkunan oikeassa reunassa olevan kalenterin avulla. Käyttäjä painaa mitä tahansa **tulevaa** päivämäärää, jolloin päivämäärä automaattisesti päivittyy kenttään. Menneille päiville ei saa varattua aikaa. Kalenterin päivää painaessa myös ikkunan keskipalkissa oleva Varaukset-alue päivittyy automaattisesti näyttämään päivän varaustilanteen. Tämän avulla käyttäjä pystyy etsimään vapaan ajan haluamalleen kentälle. Tämän jälkeen käyttäjä kirjoittaa varaukselleen haluamansa aloitusajankohdan Aika-kenttään. Aika-kentän vierestä löytyy Varauksen pituus-valikko, josta käyttäjä pystyy valitsemaan varauksensa keston. Varauksen kesto on vähintään tunti ja korkeintaan kolme tuntia, puolen tunnin välein. Pituuden käyttäjä valitsee painamalla mieluisaa kestoa. Hinta-kenttään tulostuu varaukselle muodostunut hinta, tuntihinnan ja varauksen keston mukaan. Varaa-nappia painamalla käyttäjä saa varattua itselleen kentän. Napin painalluksen jälkeen kentät

tyhjenevät, jotta voidaan tehdä seuraava varaus. Jos varaus on toisen varauksen kanssa päällekkäinen, varoittaa ohjelma siitä ja pyytää etsimään ajan, jossa ei ole päällekkäisyyksiä. Napin painalluksen jälkeen varauksen tiedot myös tallentuvat data.txt tiedostoon.

Vakiovuoron varaaminen

Aika-kentän alta löytyy Vakiovuoro-ruutu. Vakiovuoro-ruutua painamalla käyttäjä saa muutettua varauksensa vakiovuorovaraukseksi. Ruudun painamisen jälkeen ruudun alle ilmestyy Toistettavuus-kenttä, johon käyttäjä täyttää numeron. Vakiovuoro toistuu samana viikonpäivänä samaan kellonaikaan näin monen viikon välein. Eli jos käyttäjä kirjoittaa kenttään esimerkiksi 1, toistuu vakiovuoro joka viikko. Vakiovuorot varataan kerrallaan 12 viikkoa eli n. 3 kuukautta eteenpäin. Jos mikä tahansa tulevista varauksista on toisen varauksen kanssa päällekkäinen, ohjelma ilmoittaa siitä ja pyytää käyttäjää hakemaan uuden ajan. Käyttäjä on itse vastuussa vapaan vakiovuoron etsimisestä. Hinta-kentässä näkyy myös vakiovuoron hinta per kerta. Vakiovuoro tallentuu data.txt tiedostoon monena eri yksittäisenä varauksena.

Lisäpalveluiden tilaaminen

Varauksen pituus-kentän alta löytyy Mailavuokra-ruutu. Tätä ruutua painamalla käyttäjä pystyy lisäämään varaukseensa myös mailavuokran. Tätä ruutua painamalla varauksen hinta kasvaa mailavuokran verran. Uusi hinta tallentuu data.txt tiedostoon.

Historian katsominen

Kalenterin alla on Historia-nappi. Kun käyttäjä haluaa nähdä jonkun henkilön varaushistorian, kirjoittaa hän haluamansa henkilön nimen Nimi-kenttään, ja painaa Historia-nappia. Tällöin aukeaa uusi ikkuna, jossa on luettelona kaikki kyseisen henkilön vanhat varaukset. Jos nappia painetaan ilman, että Nimi-kenttään on kirjoitettu mitään, pyytää ohjelma kirjoittamaan siihen nimen. Jos puolestaan annetulla nimellä ei löydy varauksia, ilmoittaa ohjelma siitä.

Kalenterin ohessa on myös Tänään-nappi, jota painamalla kalenteri palaa nykyiseen päivään.

3. **Ulkoiset kirjastot**

   Projektissa on käytetty PyQt5 kirjastoa. Erityisesti moduuleita QtCore ja QtWidgets.

4. **Ohjelman rakenne**

   Ohjelma koostuu seitsemästä eri tiedostosta. Main tiedostossa on pelkästään main-funktio, jolla käynnistetään data.txt tiedoston lukeminen ja sovellus. Lisäksi siinä tulostetaan konsoliin data.txt tiedostosta luettujen asiakkaiden ja varausten lukumäärä muodossa XcYr missä c on asiakas ja r on varaus.

   Customer tiedostossa on määritelty customer luokka. Customer luokka sisältää asiakkaan nimen, sähköpostin, numeron ja listan asiakkaan tekemistä varauksista. Lisäksi luokka sisältää kullekin näille ominaisuuksille get_xx() metodin, jota käytetään näiden tietojen saamiseen luokan ulkopuolella.

Settings tiedostossa määritellään globaali muuttuja chunk_IO joka allokoidaan olemaan luokan ChunkIO olio. Tämä on tehty sitä varten, että voidaan globaalisti, monista eri tiedostoista lisätä varauksia ja asiakkaita saman olion alle. Yritin etsiä järkevämpiä tapoja tehdä tämän, mutta tämä oli lopulta toimiva ja yksinkertainen ratkaisu.

Reservation tiedostossa on Reservation luokka. Luokan tietoihin kuuluvat, urheilulaji, päivä, kuukausi, vuosi, tunti, minuutti, hinta ja varauksen pituus. Kullekin näille on luotu get_xx() metodi joka palauttaa niiden arvon, jotta arvoja on mahdollista käsitellä luokan ulkopuolella. Luokkaan kuuluu myös set_variables() metodi, jonka avulla luokan ulkopuolella voidaan asettaa luokan muuttujille arvoja. Tämä metodi myös käsittelee joitain odotettuja käyttäjävirheitä ja käsittelee tietyt tiedot toivottuihin muotoihin. Lisäksi luokalla on muutamia return_xx() metodeita, jotka palauttavat tietoja tietyssä muodossa. Nämä on luotu poistamaan tukkeita GUI ja chunkIO tiedostoista.

ChunkIO tiedosto sisältää Chunk_IO luokan. Chunk_IO luokka lukee lohkopohjaisen data.txt tiedoston ja sen perusteella luo asiakkaita ja varauksia. Luokassa on tätä lukemista varten muutamia apumetodeita. Lisäksi luokkaan kuuluu get_customers() ja get_reservations() metodit, jotka palauttavat luokassa luodut asiakas- ja varauslistat, koska niitä jälleen tarvitaan luokan ulkopuolella. Luokkaan vielä kuuluu tärkeä metodi check_reservation(), joka tarkastaa annetun varauksen ja vertaa sitä muihin, aiemmin tehtyihin varauksiin. Metodi havaitsee päällekkäisyydet varauksissa, jolloin loppu ohjelma tekee jatkotoimenpiteet.

Gui tiedosto sisältää GUI luokan. GUI-luokka sisältää suurimman osan ohjelman koodista. Tämän koodin olisi voinut pilkkoa useampaan tiedostoon, mutta havahduin siihen vasta kun valtaosa koodista oli kirjoitettuna. GUI-luokassa aluksi määritellään graafisen käyttöliittymän ulkoasu ja eri painikkeiden toiminnallisuus. Seuraava metodi vv_change() muuttaa käyttöliittymää sen perusteella, onko vakiovuoro-valinta valittuna vai ei. To_today() metodi asettaa kalenterin valitun päivän nykyiseen päivään. Calendar_date() määrittää Päivämäärä-kentässä näkyvän string muotoisen päivämäärän. Lisäksi se piilottaa ja näyttää keskipalkissa näkyvät varaukset kalenterin päivämäärän mukaan. Sprt_change() muuttaa varauksen hintaa tuntihinnan, varauksen pituuden ja mailavuokran perusteella. Init_customer() alustaa asiakkaan, jos asiakkaan varaus ei ole minkään toisen varauksen kanssa päällekkäinen. Init_reservation() kutsuu aluksi Chunk_IO luokan metodia jonka avulla tarkastetaan päällekkäisyys. Päällekkäisyyden puuttuessa metodi alustaa varauksen. Jos on päällekkäisyys, avaa metodi varoitusikkunan, jossa kehotetaan etsimään uusi aika. Init_line() funktio alustaa lohkopohjaisen merkkijonon joka kirjoitetaan data.txt tiedostoon. Press_res() metodi palauttaa kentät niiden oletusarvoille ja kutsuu metodia joka kirjoittaa lohkomerkkijonon tiedostoon. Show_history() avaa uuden ikkunan, jossa näkyy annetun henkilön varaushistoria. Check_fields() luodaan vakiovuorovaraus ja kutsutaan muita metodeita jotka tarkastavat onko kenttiä jäänyt tyhjäksi tai onko niissä käyttäjälähtöisiä virheitä. Jos on, metodi avaa varoitusikkunan ja kehottaa käyttäjää täyttämään kaikki kentät. Lisäksi metodi kutsuu check_empty() ja check_vakiovuoro() metodeita. Check_vakiovuoro() metodi tarkastaa onko vakiovuoro laatikko valittu, jos on palauttaa se luvun kuinka monta vuoroa kolmen kuukauden vakiovuoroon mahtuu. Check_empty() tarkastaa onko mikään kenttä jäänyt tyhjäksi, jos ei se kutsuu check_date() metodia joka tarkastaa onko päivämäärä tulevaisuudessa. Jos ei, ohjelma jälleen varoittaa siitä. Jos on, kutsuu metodi vielä check_time() metodia, joka puolestaan tarkastaa onko annettu kellonaika validi. Jos ei ole, jälleen varoitus. Show_reservations() metodi on apuna

näyttämässä varauksia. Hide_reservations() piilottaa keskipalkissa näkyvät varaukset kun päivämäärää vaihdetaan. Write_line() kirjoittaa aiemmin alustetun lohkopohjaisen merkkijonon data.txt tiedostoon.

Test-tiedosto sisältää kaikki käytetyt yksikkötestit.

Chunk_IO luokka on läsnä kaikkialla ohjelmassa, sillä siihen on tallennettuna kaikki asiakas- ja varaustiedot. Chunk_IO:ssa käytetään Customer ja Reservation luokkia asiakas ja varaustietojen luomisessa. GUI:ssa puolestaan käytetään Chunk_IO:n kautta Customer ja Reservation luokkia molempiin suuntiin. GUI sekä lukee, että kirjoittaa Chunk_IO:n sisältäviä asiakas ja varaustietoja.

5. **Algoritmit**

Tässä projektissa ei kovin monimutkaisia algoritmeja ole. Varauksen hinta lasketaan kaavan $Hinta(€) = kenttävaraus\left(\frac{€}{h}\right) * varauksen\ kesto(h) + mailavuokra(€)$ mukaisesti. Varauksen mahdollinen päällekkäisyys tarkastetaan siten, että lasketaan varauksen kellonaika auki minuuteiksi esim. 8:30 = 8*60+30=510. Tässä on siis varauksen alkuaika, johon vielä lisätään varauksen kesto, esim. 60 min jolloin varauksen loppuaika on 510+60=570. Tämän jälkeen käydään varauslistasta löytyvät varaukset läpi yksi kerrallaan ja tehdään niille sama ajan auki laskeminen. Olemassa oleville varauksille lasketaan alku- ja loppuaika samalla tavalla, jonka jälkeen verrataan, että onko tekeillä olevan varauksen alku ja loppuajan välissä yhdenkään aiemman varauksen aloitus. Sen jälkeen vielä verrataan, sijoittuuko tehtävän varauksen alkuajankohta yhdenkään olemassa olevan varauksen alku ja loppuajan väliin. Jos vastaus on ei kumpaankin, varaus ei mene päällekkäin toisen kanssa ja varauksen voi tehdä.

Erilaisten käyttäjävirheiden estäminen ja niistä varoitusten luominen tapahtuu kutsumalla yhdestä metodista toista ja toisesta kolmatta jne. Jos metodit havaitsevat jonkin käyttäjä virheen, esim. kellonajan, joka ei muodostu numeroista, ne palauttavat tietyn numeron, joka vastaa tiettyä virhettä ja ohjelma avaa varoitusikkunan, joka kertoo virheen tapahtuneen. Jos missään metodissa ei havaita virhettä, se palauttaa arvon 0 joka vastaa toimivaa varausta. Varauksen ajankohta tarkistetaan jälleen samalla tavalla kertomalla se auki. Halli on auki vain 7-20, jolloin varausta ei saa tehdä ennen klo 7 ja varauksen loppuajankohta ei saa ylittää klo 20, joten se tarkistetaan kertomalla aika auki.

6. **Tietorakenteet**

Projektiin sopii erittäin hyvin listat, jotka sisältävät olioita. Lisäksi asiakas olio sisältää itsessään listan, joka sisältää kaikki asiakkaan tekemät varaukset. Tällä tavalla saadaan historian tarkistusta varten helposti kaikki yhden asiakkaan tekemät varaukset. Käytin vain Pythonin valmiita tietorakenteita.

7. **Tiedostot**

Ohjelma käsittelee lohkopohjaista tekstitiedostoa data.txt. Data.txt tiedostossa aina yksi rivi vastaa yhtä varausta. Projektini mukana on data.txt tiedosto joka sisältää validia dataa, jonka luettuaan ohjelma luo varauksia. Yksi rivi on muotoa:

```
NAM13TeemuTeekkariNUM100101231234EML23teemu.teekkari@aalto.fiDAT0805062025TIM040915SPO0203LEN0260PRC0214END00
```

Lohkot muodostuvat 3 kirjaimen pituisista otsakkeista ja 2 merkin pituisesta numerosta, joka kertoo, kuinka monta seuraavaa merkkiä otsakkeeseen kuuluu. 2 numeroa riittää 99 merkkiin asti, jonka pitäisi olla riittävästi. Eri otsakkeiden merkitys: NAM Nimi, NUM puhelinnumero, EML sähköposti, DAT päivämäärä, TIM aloitusajankohta, SPO urheilulaji, LEN varauksen pituus minuuteissa, PRC hinta, END lopetus. Tyhjään tekstitiedostoon pystyy luomaan varauksia indikoivia rivejä ohjelmaa ajamalla.

8. **Testaus**

Ohjelmaa testattiin hyvin vajanaisesti. Testaamisesta ei jäänyt paljoa mieleen viikkoharjoituksista ja yrityksestä huolimatta en kerennyt opiskelemaan yksikkötestauksesta tarpeeksi, että olisin saanut mitään kovin merkityksellistä testattua. Yritin testata suurinta osaa funktioista, mutta ilmeisesti yksikkötestaus ei toimi metodeissa, joiden sisällä on toisia funktioita tai kutsuja toisiin metodeihin. En ainakaan saanut sitä toimimaan. Onnistuin testaamaan add_customer() metodin ja set_variables() metodin. Tämän lisäksi testasin read_fully(), get_chunk_name() ja get_chunk_size() metodit.

Suunnittelin, että olisin testannut myös varausten lisäämisen ja sen, että saako jo varatun ajan päälle varattua toista aikaa, mutta tähän en löytänyt tapaa.

Ohjelmaa rakentaessa testasin sitä debuggerin avulla. Kokeilin lukuisia kertoja oikeilla syötteillä ja lukemattomia kertoja väärillä syötteillä. Aina kun implementoin uuden ominaisuuden, yritin löytää tavan, miten voisin ”rikkoa” sen eli löytää jonkin syötteen millä ominaisuus ei toimi toivotulla tavalla.

9. **Ohjelman tunnetut puutteet ja viat**

Ohjelman yksikkötestaus jäi aivan turhan vajaaksi. Olisin tykännyt saada enemmän testejä aikaiseksi ohjelmaan. Viikkoharjoituksissa testauksen hyödyllisyys ja toteutus meni hieman ohi, joten oli hyvin vaikea implementoida kunnollista testausta tähän projektiin.

10. **3 parasta ja 3 heikointa kohtaa**

Ohjelmassani erityisen hyvä on käyttöliittymä kokonaisuutena. Käyttöliittymä on yksinkertainen ja intuitiivinen. Käyttöliittymän dynaamisuus on mielestäni erittäin hyvä ominaisuus, joka tekee käyttäjäkokemuksesta huomattavan hyvän. Erityisesti siinä pidän vakiovuoro napin avulla ilmestyvää valintamahdollisuutta sekä päivämäärää valitsemalla varausten dynaamisen muuttumisen.

Lisäksi ohjelmassa erityisen hyvää on mielestäni käyttäjävirheisiin puuttuminen. Se, että käyttäjän virheellisestä syötteestä ja väärän kellonajan tai päivämäärän valitsemisesta tulee huomautusikkuna, jossa selitetään mikä on väärin, on mielestäni hyvin toteutettu.

Vakiovuoro varaus toimii myös hyvin. Mielestäni se on hyvä lisä käyttäjille, jotka tietävät haluavansa käydä pelaamassa esimerkiksi kahden viikon välein. Tällöin käyttäjä voi vain tarkastaa onko vapaita aikoja ja tehdä vakiovuorovarauksen kolmeksi kuukaudeksi. Vakiovuorovarauksen tekemisen implementointi on mielestäni onnistunut hyvin.

Ohjelmassa kehitettävää on käyttöliittymän keskipaneeli, se voisi olla jäsennelty esimerkiksi kenttä kerrallaan tai kronologisesti. Olisin todennäköisesti muuten tehnyt sen, mutta aika loppui valitettavasti kesken.

Kehitettävää on myös luokkajako ja erityisesti GUI luokan pilkkominen muihin pienempiin luokkiin. Koodissa on myös jonkin verran toistoa, jonka olisi voinut välttää järkevämmällä luokkajaolla ja koodaustyylillä.

11. **Poikkeamat suunnitelmasta**

    Aluksi suunnitteilla oli tehdä kentistä eri hintaisia riippuen kellonajasta, mutta en lopulta kokenut sitä tarpeelliseksi. Lisäksi suunnittelin aluksi, että vakiovuorossa olisi ollut mahdollista valita tapahtumaväliksi viikon lisäksi kuukausi, mutta kolmen kuukauden varauksen kanssa ei olisi kovin järkevää varata kuukauden tai jopa useamman kuukauden välein vuoroa.

    Ajankäyttöarvio osui lähestulkoon oikeaan. Käyttöliittymän tekemiseen meni vähemmän aikaa kuin olin budjetoinut. Käyttöliittymän tekeminen olikin huomattavasti helpompaa ja intuitiivisempaa kuin olin odottanut. Varausjärjestelmän implementoimiseen meni n. 45h eli suunnitellun verran. Toteutusjärjestys pysyi samana, aluksi GUI, jonka jälkeen ohjelmiston implementointi ja lopuksi testien tekeminen.

12. **Toteutunut työjärjestys ja aikataulu**

    Aluksi toteutin käyttöliittymän. Käyttöliittymän lähes lopullinen versio oli valmis 18.3. eli reilusti ennen ensimmäistä checkpointtia. Seuraavaksi loin lohkopohjaisen tiedostotyypin pohjan ja aloitin sen avulla luomaan itse järjestelmän ohjelmointia. Osan tästä sain valmiiksi 17.4. mennessä. Alkuperäisestä aikataulusta poikettiin tässä vaiheessa. Olin suunnitellut, että varausjärjestelmän implementointi olisi valmis 15.4. eli toiseen checkpointiin mennessä, mutta muut kurssit ja muu tekeminen vei aikaa siten, etten kerennyt tekemään. Toisen checkpointin jälkeen seuraavat kaksi viikkoa olivatkin niin täynnä, etten kerennyt tekemään koulutöitä lähes ollenkaan. 2.5. palasin jälleen projektin pariin ja tein pitkiä päiviä ehtiäkseni deadlineen. Tämän projektin aikana ollut periodi oli poikkeuksellisen kiireinen sekä koulun, että muun elämän kanssa.

13. **Arvio lopputuloksesta**

    Loppujen lopuksi projekti meni oikein hyvin. Olen hyvin tyytyväinen omaan varausjärjestelmääni ja olen ylpeä siitä mitä olen saanut aikaan.

    Ohjelman koodaustyyliä olisi voinut parantaa huomattavasti. Uskon, että samaan lopputulokseen olisi voinut päästä paljon pienemmällä rivimäärällä ja koodia olisi voinut selkeyttää ja jäsennellä paljon paremmin. Tästä huolimatta koodini on luettavaa ja helposti ymmärrettävää. Luokkajaossa on kehittämisen varaa, sillä GUI luokka on tarpeettoman suuri. GUI:ssa olevat käyttöliittymään liittymättömät osat olisi voinut ulkoistaa toiseen luokkaan, josta niitä olisi voinut kutsua tarvittaessa. Mielestäni käyttöliittymäni on hyvin intuitiivinen, mutta siinä olisi voinut kehittää keskipalkkia esimerkiksi näyttämään kentät jäsennellysti tai luokiteltuna.

    Käyttäjähistoria osion olisi myös voinut asettaa kronologiseen järjestykseen, jolloin käyttäjä näkisi helposti, milloin seuraava varaus on. Lisäksi menneet varaukset voisivat olla eri värisiä, jolloin ne erottaisivat helpommin.

    Ohjelma sopii hyvin mahdollisiin laajennuksiin, kenties useampi kenttä samalla lajilla tai jopa vahvistussähköpostien lähettäminen. Myös erilaiset tilastot eri käyttäjille kuten kokonaisaika tai kokonaishinta voisivat olla mielenkiintoisia.

14. **Viitteet**

- A+ kurrsimateriaali
- https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/index.html#module-PySide2.QtWidgets
- https://doc.qt.io/qtforpython-5/PySide2/QtCore/index.html#module-PySide2.QtCore