Ade Aiho, Heta Hartzell, Mika Laakkonen, Jonne Roponen

Pizzeria Simulaattori

Pizzasaurus Rex

Metropolia Ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotuotanto
Ohjelmointiprojekti
Päivämäärä 20.10.2024

Sisällys

1 Johdanto

Tämä dokumentti käsittelee olio-ohjelmoinnin perusteet kurssilla toteutettavaa simulaatioprojektia. Projektin tavoitteena on toteuttaa ryhmätyönä simulaattori millä voidaan tutkia oikean maailman ilmiötä. Simulaattorin avulla saadaan syvempi ymmärrys tutkittavasta järjestelmästä, minkä perusteella voidaan tehdä toiminnallisia parannuksia.

Projektin avulla ryhmä ensimmäisen vuoden ohjelmistotuotannon pääaineopiskelijoita saa kokemusta ja harjoitusta Java olio-ohjelmoinnista. Simulaattorin kehityksen ja luonnin parissa ryhmä pääsee hyödyntämään käytännössä olio-ohjelmointi kurssilla opittua teoriaa. Dokumentin avulla ryhmä pystyy asettamaan projektille tavoitteet ja pitämään kirjaa simulaattorin rakenteista ja käsitteistä. Se toimii myös ohjenuorana ja opasteena projektiryhmän jäsenille. Selkeästi ja hyvin suoritettu määrittelydokumentti tekee projektin toteuttamisesta helpompaa ja se auttaa hahmottamaan, mitä vaatimuksia ja rajoituksia simulaattorissa voi tulevaisuudessa ilmetä.

Dokumentti alkaa osiolla visio, missä kerrotaan minkälaisen simulaattorin ryhmä on toteuttamassa, sekä kiteytettynä projektin tavoitteen. Käsitteet ja määritelmät osiossa avataan kaikki dokumentissa käytetty sanasto ja ilmaisut. Käsitteellisessä mallissa käydään läpi yksityiskohtaisemmin simulaattorin rakenne, toiminta ja tavoitteet.

Mallin ohjelmointiteknisessä toteutuksessa puhutaan käytetyistä ohjelmointikielistä ja kirjastoista sekä ohjelmiston arkkitehtuurista, joka määrittelee rakenteen ja komponenttien yhteydet. Käyttöliittymän kuvaus kertoo käyttäjän vuorovaikutuksesta ohjelman kanssa ja sisäisen logiikan kuvaus taas selittää ohjelmiston toiminnan. Ulkoisten tietovarastojen kuvaukset osio käy läpi simulaattorissa hyödynnettyjä tietokantoja ja tiedostoja. Testaus puhuu miten yleistä ohjelman testausta suoritettiin, sekä minkälaisia JUnit-testejä kehitettiin eri ohjelman komponentteja varten.

Simulaattorin käyttöohje selittää miten ja minkälaisia syötteitä simulaatioon annetaan sekä kuinka saatuja tuloksia tulkitaan. Tehdyt simulointikokeet osio kertoo, mitä testejä simulaattorilla suoritettiin ja mitä niistä saatiin selville. Yhteenveto tiivistää projektin päätavoitteet ja havainnot.

2 Visio

Projektin tavoitteena on toteuttaa kolmivaiheinen jonotussimulaattori, jota voidaan hyödyntää mallintamaan oikean elämän ilmiötä. Projektin mallinnuksen kohteena on pizzeria, joka voidaan toteuttaa kolmivaiheisen simulaation avulla. Simulaattorin tarkoituksena on antaa käyttäjälle informaatiota, jonka avulla käyttäjä pystyy tekemään niin taloudellisia kuin tehokkuuteen vaikuttavia päätöksiä, oli kyse sitten tuotteiden hinnoista ja kustannuksista tai työvoiman jakamisesta. Simulaation kohdekäyttäjiä voivat olla niin yksityisyrittäjät, franchise-yrittäjät, perheyritykset tai muut suuremmat yrityskokonaisuudet.

Pizzeria-simulaation alussa asiakas saapuu pizzeriaan ja jonottaa ovi-palveluun. Asiakas jonottaa ovi-palvelussa, kunnes hänelle löytyy vapaa pöytä. Tämän jälkeen asiakas odottaa, että tarjoilija saapuu pöytään vastaanottamaan tilauksen. Tilaus viedään keittiöön, jossa se käy ensiksi kokin palvelupisteellä ja toiseksi pizzauunissa. Tilauksen valmistuttua seuraava vapaa tarjoilija toimittaa sen asiakkaalle. Lopuksi asiakas syö pizzansa, maksaa laskun, ja poistuu pizzeriasta, jolloin myös pöytä vapautuu seuraavalle asiakkaalle.

Haarautuva polku, joka liittyy aikaisempaan malliin, on noutotilaukset. Ensiksi, noutotilaus luetaan pizzerian noutotilausjärjestelmästä, minkä jälkeen siitä lähtee viesti keittiöön. Tätä kautta tulleet tilaukset päätyvät samaan keittiön jonoon kuin paikan päällä tilatut pizza. Noutotilauksen valmistuttua vapaa tarjoilija paketoi sen, ja lopuksi ilmoittaa tilauksen olevan valmis. Tilauksen valmistuttua se poistuu simulaation näkökulmasta järjestelmästä, koska sen noutaminen ei liity enää pizzerian prosesseihin.

3 Käsitteet, määritelmät

Tässä osioissa käydään läpi simulaatiossa keskeiset käsitteet ja niiden määritelmät.

- **Asiakas** on ravintolassa asioiva henkilö. Asiakas jonottaa pöytään pääsyä, tilaa aina pizzan ja juoman, odottaa ruokaansa ja syö.
- **Noutotilaus** on tilaus, jonka yhteydessä ravintolaan ei fyysisesti ilmaannu asiakasta. Tilaus jonottaa käsitellyksi tulemista ja sen jälkeen odottaa valmistumista ja pakkaamista.
- **Tarjoilija** työskentelee ravintolassa asiakkaiden parissa. Tarjoilijan tehtäviin kuuluu tilausten vastaanottaminen, pöytiintarjoilu ja noutotilausten pakkaaminen.
- **Kokki** työskentelee ravintolan keittiössä. Kokin tehtävä on valmistaa pizza, jotta se voidaan laittaa uuniin.
- **Palveluaika** on asiakkaan viettämä aika ravintolassa, saapumisajasta poistumiseen.
- **Palvelupiste** on pizzerian tuotantoketjussa oleva vaihe, jossa tapahtumia käsitellään. Esimerkiksi pizzauuni on palvelupiste, jossa pizzoja valmistetaan.
- **Tapahtuma** on pizzeriassa tapahtuva asia, joka määrittää mitä tapahtuu, kauan se kestää, ja mikä tapahtuma siitä seuraa.
    - Tapahtuma voi olla **ajastettu tapahtuma**, jolloin se suoritetaan annetulla ajanhetkellä ja siitä seuraa uusi tapahtuma. Esimerkiksi pizzan tekeminen aloitettiin jollain ajanhetkellä, kesti tietyn ajan, ja loi tapahtuman "pizza valmis" vastaavan ajan päähän.
    - Tapahtuma voi olla **ehdollinen tapahtuma**, eli sen toteutuminen vaatii tiettyjen ennalta määrättyjen ehtojen täyttymistä. Esimerkiksi pizzan tekemisen aloittaminen riippuu siitä, onko kokki vapaa.
- **Syöte** on simulaattoriin annettava parametri. Ne vaikuttavat simulaatiosta saataviin tuloksiin. Syötteisiin kuuluu muun muassa tarjoilijoiden, kokkien, ja pizzauunien määrä, sekä onko annokset alennuksessa.
- **Kulu** on pizzerian operatiivinen meno, joita tehdään ylläpitämään ja kasvattamaan pizzerian liiketoimintaa ja liikevaihtoa. Kulut jakautuvat kiinteisiin kuluihin, jotka eivät riipu simulaattorin syötteistä (esim. vesi, vuokra), ja muuttuviin kuluihin, jotka riippuvat simulaattorin syötteistä (raaka-aineet).

- **Tuloste** on simulaatiosta lopuksi saatu laskelmoitu arvo. Tulosteita on useita, kuten keskimääräinen asiakkaan jonotusaika, netto ja bruttotulot, työntekijöiden joutilas aika, jne..

4 Käsitteellinen malli

4.1 Tavoite

Simulaatio kuvaa pizzerian päivittäistä toimintaa, ja tavoitteena on optimoida prosessit siten, että kulut pysyvät minimissä ja tuotto maksimoituu. Selvityksessä on esimerkiksi jonojen odotusajat, asiakasvirran järkevin käsittely sekä työntekijöiden tehokkuus. Malli auttaa ravintoloitsijaa ymmärtämään, miten erilaiset tekijät, kuten alennukset ja työntekijöiden määrä, vaikuttavat ravintolan toimintaan. Se myös antaa tietoa siitä, miten asiakasvirrat ja työntekijöiden suorituskyky vaikuttavat ravintolan tehokkuuteen.

Simulaatio tarjoaa näin ollen kokonaisvaltaisen kuvan ravintolan toiminnasta, joka voi auttaa yrittäjää kehittämään liiketoimintaansa tehokkaammaksi ja asiakasystävällisemmäksi.

4.2 Syötteet

Simulaatiossa käyttäjä voi asettaa kahden erilaisia arvoja ohjelman parametreiksi. Näitä arvoja ovat työntekijöiden ja pöytien määrä sekä onko annokset tarjouksessa. Lisäksi käyttäjä voi määrittää haluamansa arvot pizzojen ja juomien hinnoiksi, kuluiksi ja todennäköisyyksisksi sekä myös vaihtaa asiakasmäärän keskiarvoa ja varianssia. Tämä mahdollistaa erilaisten skenaarioiden tutkimisen ja antaa ravintoloitsijalle työkaluja päätöksentekoon.

4.3 Tulosteet

Tulokset tarjoavat arvokasta tietoa ravintolan suorituskyvystä, kuten asiakasmäärästä jonotusajoista, netto- ja bruttotuloista sekä työntekijöiden tehokkuudesta. Näiden avulla ravintoloitsija voi arvioida palvelun laatua ja resurssien käyttöä, ja tehdä tarvittavia muutoksia toiminnan parantamiseksi.

4.4 Sisältö

Simulaatio kuvaa rajallisesti todellista pizzerian toimintaa keskittyen olennaisiin tekijöihin, kuten asiakkaiden jonotusprosesseihin ja työntekijöiden tehtäviin. Asiakasryhmä käsitellään yhtenä kokonaisuutena, eikä yksittäisten henkilöiden eroavaisuuksia syömisajoissa huomioida. Lisäksi malli ei sisällä mahdollisuutta, että raaka-aineet loppuisivat kesken tai että asiakkaat tekisivät lisätilauksia aterian aikana, mikä voi olla todellisessa ravintolaympäristössä tavallista.

Mallissa tarjoilijan ja kokin työtehtävät on jaoteltu selkeisiin osiin, mutta todellisuudessa niiden määrä ja moninaisuus ovat laajempia. Esimerkiksi tarjoilijan työtehtäviin kuuluu tilausten vastaanotto ja noutotilausten pakkaus, mutta monia muita tehtäviä ei mallissa huomioida. Samoin kokin työnkuva on yksinkertaistettu keskeisiin tehtäviin, kuten pizzan valmistamiseen ja uunitukseen, ilman lisähuomioita, kuten raaka-aineiden täyttö päivän aikana.

4.5 Oletukset ja yksinkertaistukset

Simulaatiossa oletetaan, että pizzeriaan saapuu tietyn ajan sisällä asiakkaita määrä, joka vaihtelee satunnaisesti pienen rajatun välin sisällä. Lounas- ja illallisaikaan pizzeria on kiireisimmillään, ja asiakkaita saapuu silloin enemmän. Jokainen asiakas tilaa oletuksena yhden pizzan ja juoman ja kaikki ainekset ovat aina saatavilla. Pizzerialla on määritellyt aukioloajat ja muina aikoina se on kiinni. Oletuksena on myös, että työntekijöiden tehokkuus ja taitotaso on vakio tai vaihtelee pienen rajatun välin sisällä. Simulaatiossa ei oteta huomioon suuria vahinkoa aiheuttavia tapahtumia kuten tulipaloja, vesivahinkoja tai sähkökatkoksia.

Mallia yksinkertaistetaan rajaamalla pizzerian ruokalista kolmeen pizzatyyppiin (margarita, vegaani, meat lover), joista jokaisesta on kolme kokoa (small, medium, large). Tarjolla on myös kolme juomaa (vesi, limu, olut). Pizzan valmistusprosessi on yksinkertaistettu kokin valmistusaikaan ja uunin paistoaikaan, eikä tarkempia valmistusvaiheita oteta huomioon. Noutotilausten haku on yksinkertaistettu siten, että tilaukset noudetaan aina pizzeriasta, kun ne ovat pakattu, eikä kotiinkuljetusta simuloida. Pizzerian mahdollisia kilpailijoita, markkinatilannetta tai sääoloja ei huomioida simulaatiossa.

4.6 Mallin kuvaus

4.6.1 Komponenttilista

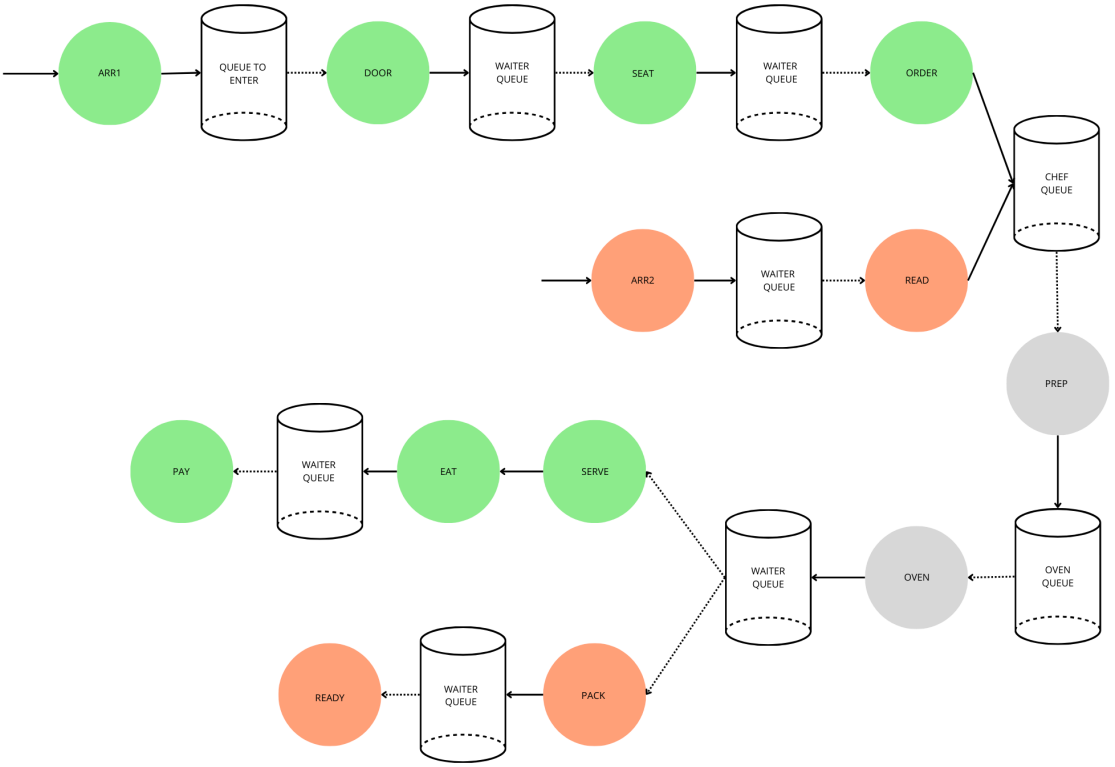
Seuraavassa taulukossa 1. käydään läpi ravintolan toimintaan liittyvät keskeiset komponentit, sekä niiden ominaisuudet.

Komponentti	**Ominaisuuksia**
asiakas	saapumisväliiaikojen jakauma huomioiden lounas- ja illallispiikit ruokailuajan jakauma
ruokailutila	ovijono, pöytien määrä on rajallinen eli jonolla on kapasiteetti vaikuttaa myös tarjoilijoiden vapaus
tarjoilijan jono	kapasiteetti ääretön
tarjoilija	tilausten vastaanottoon, tarjoiluun ja noutotilausten pakkaamiseen kuluvan ajan jakaumat
kokin jono	kapasiteetti ääretön
kokki	ruoan valmistusajan jakauma
uunin jono	uunin kapasiteetti määrittää jonon kapasiteetin
uuni	paistoajan jakauma
noutotilaus	tilauksien saapumisiväliaikojen jakauma huomioiden lounas- ja illallispiikit

Taulukko 1. Ravintolan komponenttilista.

4.6.2 Prosessikaavio

Kuva 1 havainnollistaa simulaation kulkua. Vihreät pallot kuvaavat asiakkaan saapumista ravintolaan, ja tilauksen kulkua simulaatiossa. Oranssit pallot kuvaavat noutotilauksen saapumista ja kulkua simulaatiossa.



Kuva 1. Prosessikaavio pizzeria-simulaattorista.

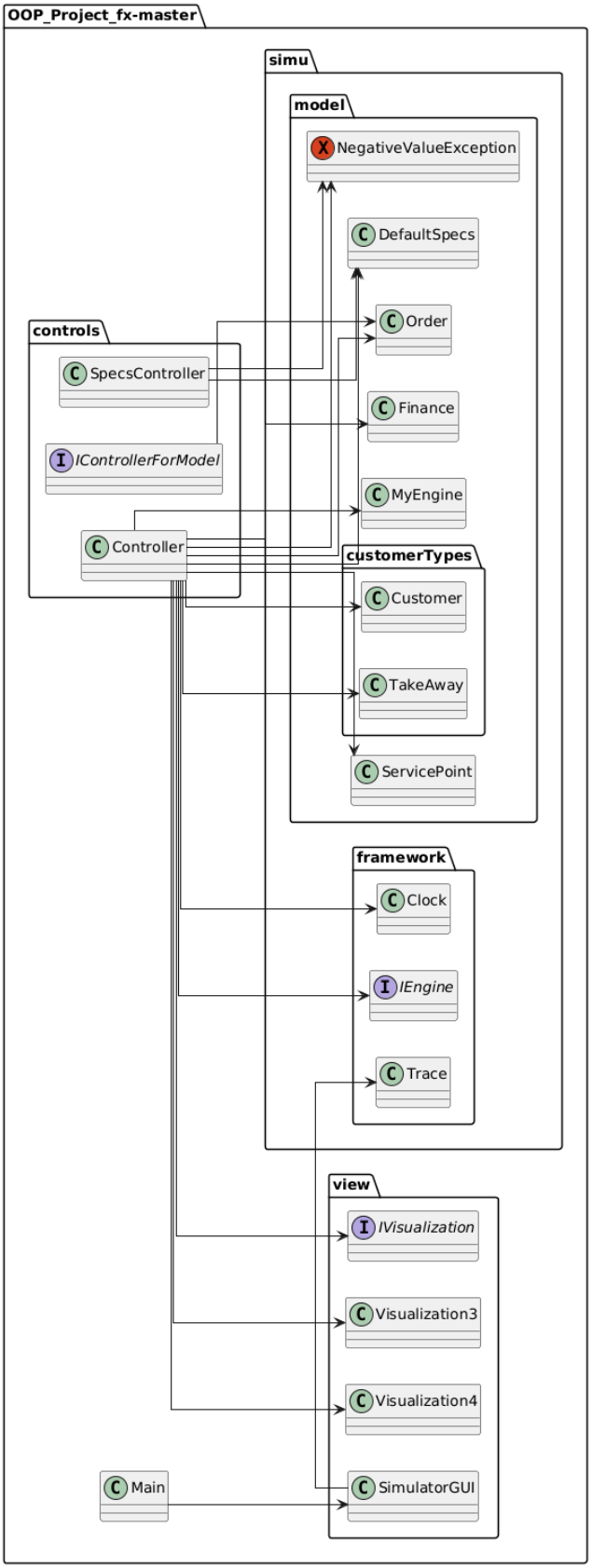
5 Mallin ohjelmointitekninen toteutus

5.1 Käytetyt ohjelmointikielet ja kirjastot (ulkoiset API:t).

Koko ohjelma on pääasiassa toteutettu Java-ohjelmointikielellä. JavaFX-kirjastoa käytetään laajasti käyttöliittymän rakentamiseen ja kehittämiseen, mukaan lukien komponentit kuten Canvas, Scene, Stage, FXML ja muita. JavaFX-kirjaston lisäksi käyttöliittymässä useasti esiintyvien elementtien tyyliä määritetään myös CSS-tyylitiedostolla. Projekti käyttää Mavenia riippuvuuksien hallintaan, mukaan lukien JavaFX, JPA, Hibernate, MariaDB ja Logback. MariaDB JPA (Jakarta Persistence API) hallinnoi tietokantayhteyksiä ja operaatioita, mukaan lukien entiteettien käsittely ja tietokantakyselyt. Hibernate toimii ORM-työkaluna (Object-Relational Mapping) tietokantaoperaatioiden automatisointiin, ja MariaDB Java Client mahdollistaa yhteyden MariaDB-tietokantaan. Logbackia käytetään kirjaamaan virheilmoituksia ja muita sovelluksen tapahtumia lokitiedostoihin. Lisäksi ohjelmaan manuaalisesti tuotua eduni.distributions-kirjastoa käytetään satunnaisgenerointiin ja tilastollisiin jakaumiin. Kirjasto tarjoaa tilastollisia funktioita, kuten Bernoulli-, Beta- ja normaalijakaumia, joista normaalijakaumaa ja Uniform-jakaumaa käytetään sovelluksen logiikassa.

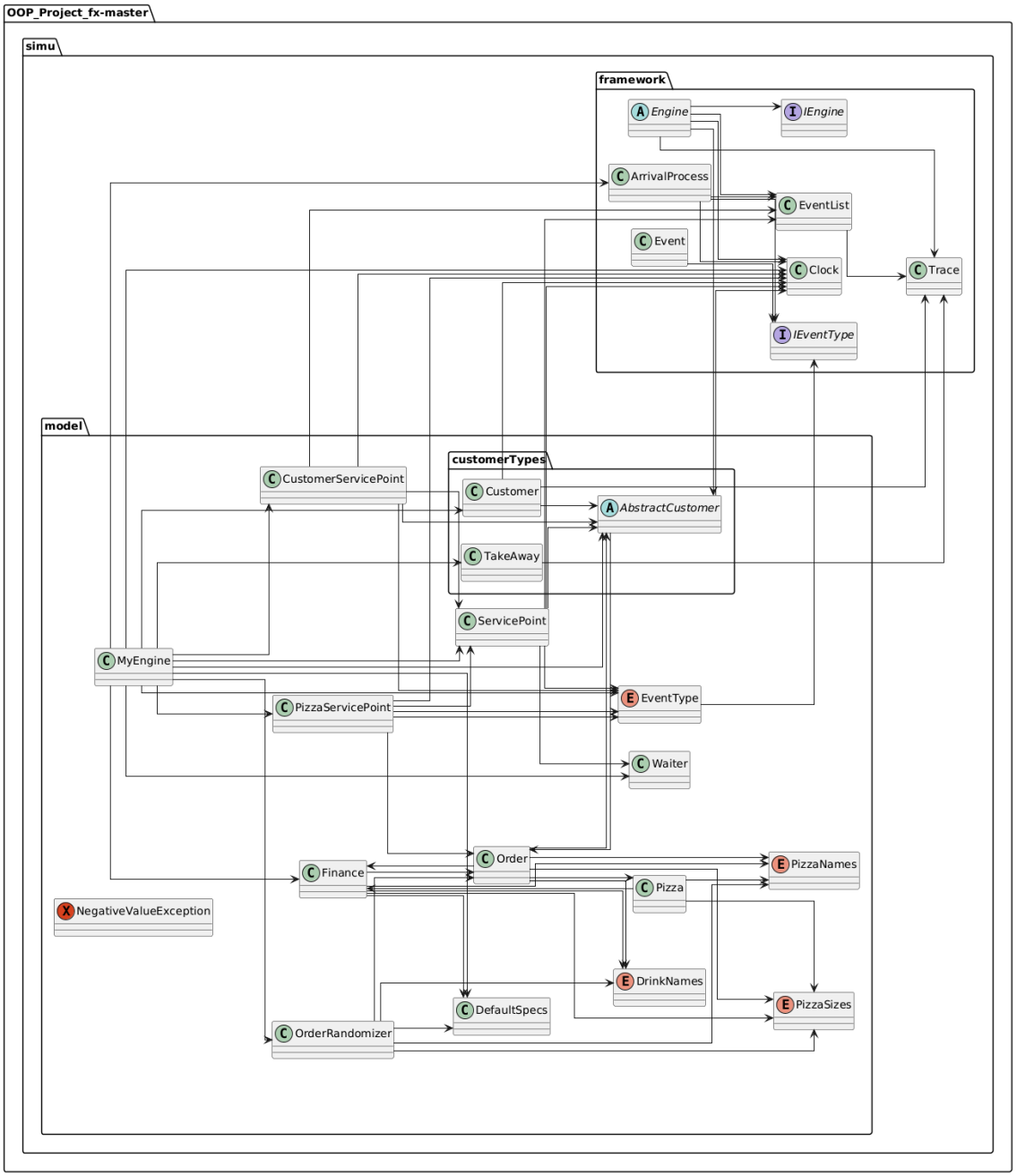
5.2 Arkkitehtuuri

Kuva 2. havainnollistaa simulaation arkkitehtuuria. Kaaviossa on kuvattu yhteydet MVC-mallin (Model-View-Controller) välillä. Luokat, joilla ei ole ulkopuolisia yhteyksiä eivät näy kaaviossa.



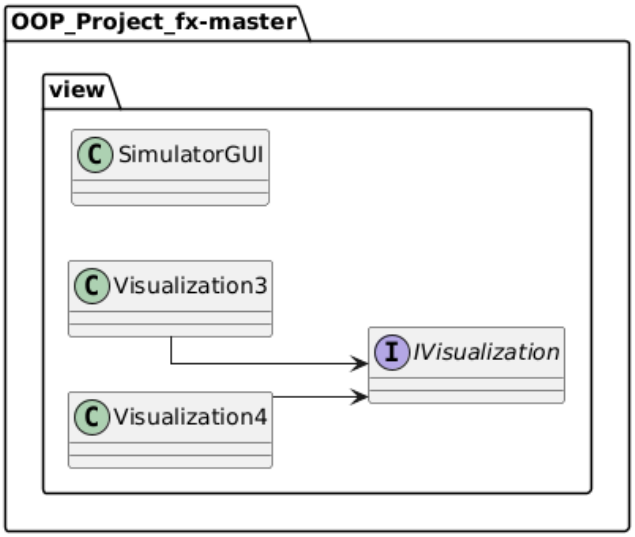
Kuva 2. UML-kaavio simulaation arkkitehtuurista.

Simulaatiossa malli (Model) vastaa ohjelman toimintalogiikasta ja tietojen käsittelystä. Kuva 3 havainnollistaa mallin välisiä yhteyksiä.



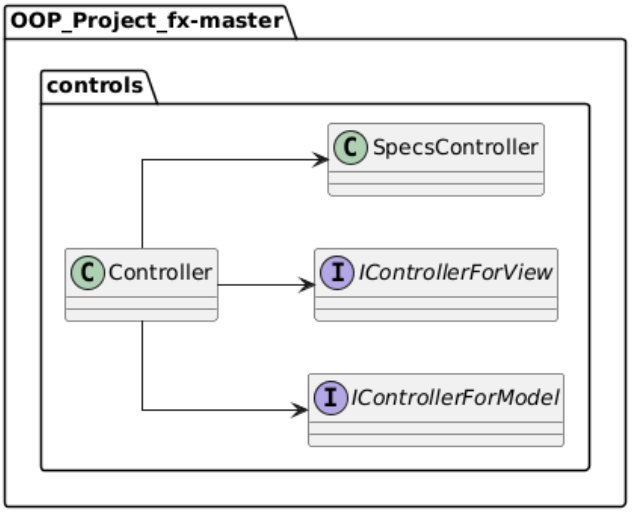
Kuva 3. UML-kaavio mallin arkkitehtuurista.

Näkymä (View) määrittää käyttöliittymän ulkoasun sekä sen, mitä tietoja käyttäjälle näytetään. Kuva 4 havainnollistaa näkymän välisiä yhteyksiä.



Kuva 4. UML-kaavio mallin arkkitehtuurista.

Käsittelijä (Controller) ottaa vastaan käyttäjän syötteet, käsittelee ne ja kommunikoi sekä mallin että näkymän välillä. Kuva 5 havainnollistaa käsittelijän välisiä yhteyksiä.

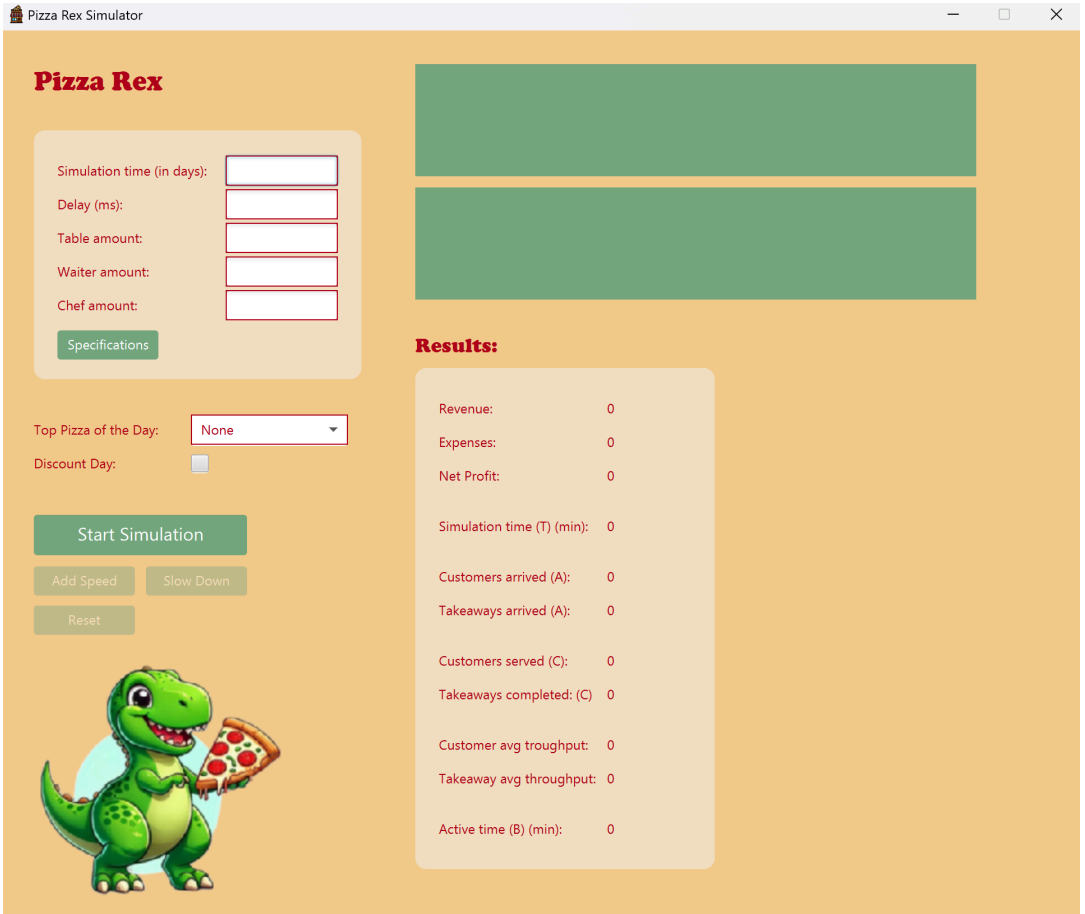


Kuva 5. UML-kaavio käsittelijän arkkitehtuurista.

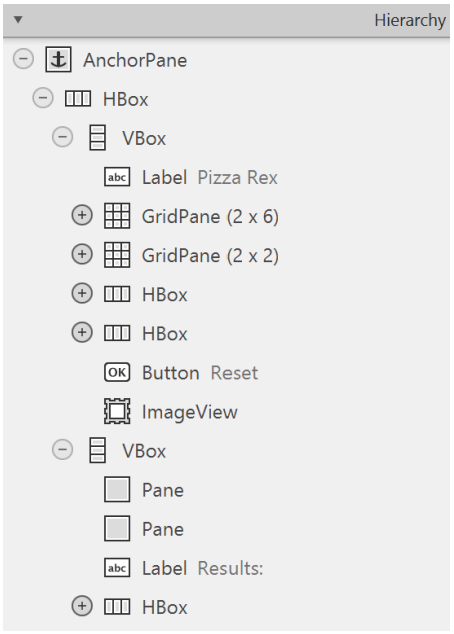
5.3 Käyttöliittymän rakennekuvaus

Kuvat 6-12 esittävät ohjelman käyttöliittymän visuaalisia elementtejä. Kuvassa 6 näkyy ohjelman aloitusikkuna. Ikkunaelementille on asetettu otsikoksi "Pizza Rex Simulator" ja siihen on lisätty kustomoitu kuva. Kuvassa 7 on esitetty ohjelman elementtien välinen hierarkia. Pohjana toimii AnchorPane-säiliöelementti, jonka päälle on asetettu

HBox-kontti. HBox jakaa ohjelman kahteen VBox-säiliöelementtiin. Vasemmanpuoleisessa VBox-elementissä on ohjelman nimike, käyttäjän syötteet, napit ohjelman aloitusta ja hallintaa varten sekä logo. Oikeanpuoleisessa VBox-elementissä on kaksi paneelielementtiä ohjelman kannalta tärkeiden jonojen visualisointiin, tulosten nimike ja GridPane-taulukko ajonaikaisille tuloksille. Ajonaikaisten tulosten oikealla puolella visualisoidaan simulaation etenemistä latauskuvalla, jonka päälle ilmestyvät ajon jälkeiset lopulliset tulokset. Kuvan ja tulosten vaihtumisen toteutuksessa on käytetty StackPane-elementtiä.

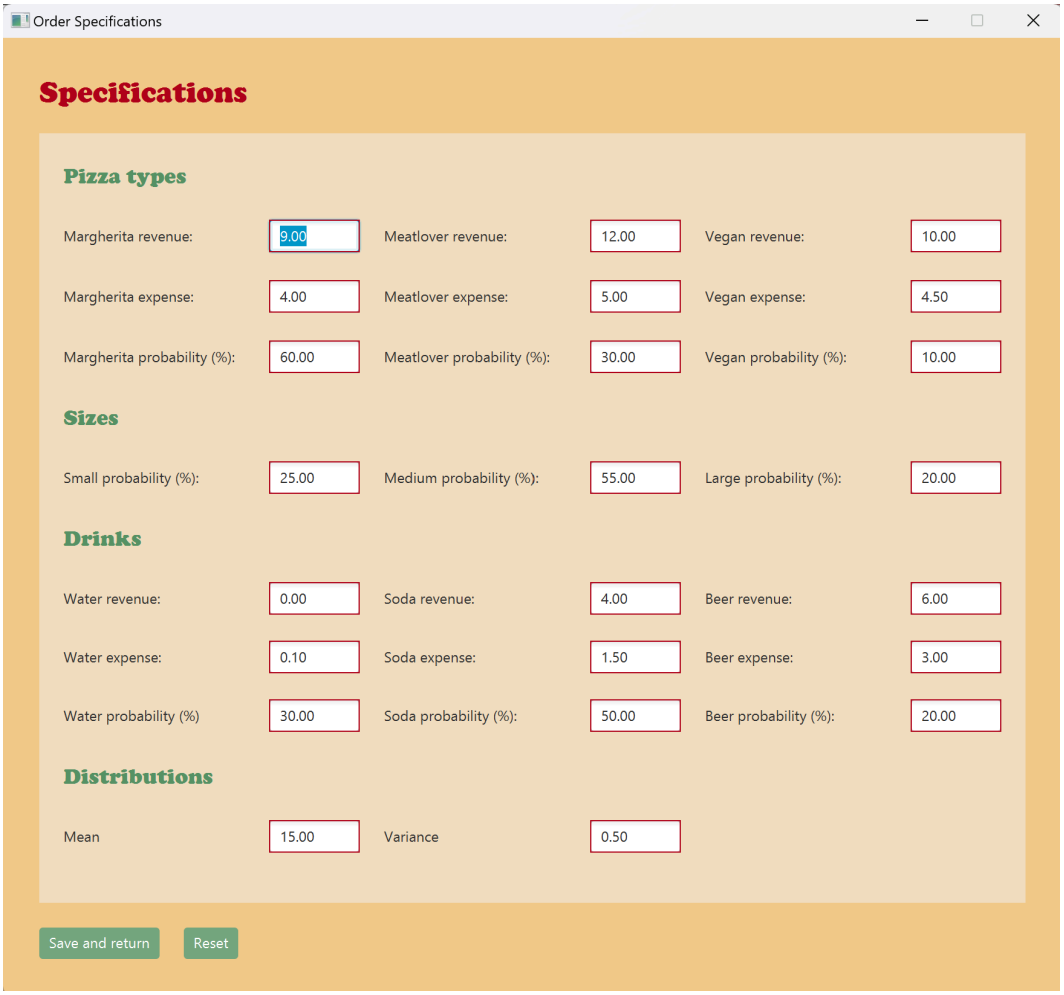


Kuva 6. Ohjelman käyttöliittymä.

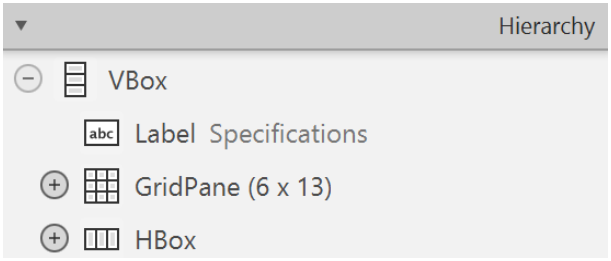


Kuva 7. Käyttöliittymän rakenteen hierarkia.

Kuvassa 8 näkyy käyttäjän lisämääritysten ikkunaelementti, jossa voidaan vaikuttaa simulaation kulkuun. Kuvassa 9 on esitetty elementtien välinen hierarkia. Sivulla on GridPane-kontissa otsikoita ja tekstikenttiä. Ikkunan alareunassa on napit tietojen tallennukseen sekä resetointipainike, jolla valinnat palautetaan ennalta määritettyihin oletusarvoihin.



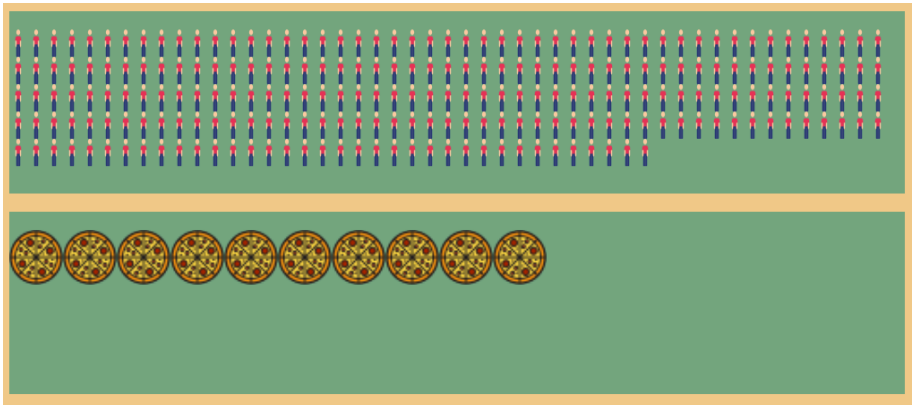
Kuva 8. Käyttäjän syötteiden muokkausnäkymä.



Kuva 9. Syötteiden muokkausnäkymän hierarkia.

Kuvassa 10 on paneelielementteihin visualisoituja jonoja. Ylempi jono kuvaa asiakkaiden sekä noutotilausten saapumista ravintolaan. Yksi asiakas näkyy yhtenä figuurina, ja viidennen rivin täyttyessä ylin rivi tyhjentyy, figuurit siirtyvät yhden rivin ylemmäs, ja täyttö alkaa taas vasemmasta laidasta. Alempaan paneelielementtiin

visualisoituu keittiön jonotilanne. Kun tarjoilija ottaa vastaan tilauksen, kuvaan ilmestyy figuuri. Kun tilaus valmistuu uunista, figuuri poistuu.



Kuva 10. Asiakas- ja pizzajonon ajonaikainen visualisointi.

Kuvassa 11 näytetään käyttäjälle ohjelman ajon jälkeen suorituskykymuuttujat. Visualisointi tapahtuu GridPane-kontissa.

Kuva 11. Suorituskykymuuttujien tulokset.

Kuvassa 12 näkyy latautuva pizza, joka kertoo ohjelman edistymisestä. Ohjelman suorituksen edetessä pizza täyttyy neljäsosa kerrallaan. Jokaisen suorituksen neljänneksen valmistuttua pizzaan ilmestyy yksi uusi neljäsosa.

Kuva 12. Ohjelman keston visualisointi latautuvalla pizzakuvalla.

5.4 Sisäisen logiikan kuvaus

Simulaattorin logiikka perustuu kolmivaihe-simulaatio malliin. Siihen sisältyy kolme seuraavaa vaihetta: A-vaihe, eli kellon siirtäminen seuraavan tapahtuman loppumisajankohtaan, B-vaihe, jossa kyseinen tapahtuma suoritetaan loppuun ja asiakas (tai muu tapahtuman kohteena ollut olio) siirretään jonoon, ja C-vaihe, jossa tarkastetaan, että pystyykö edeltävän tapahtuman päättymisen vuoksi aloittamaan palvelua missään palvelupisteessä. Tapahtuma ja palvelu eroaa siten, että palvelussa suoritetaan jotain simulaatiossa mallinnettavaa prosessisia, kun taas tapahtuma kuvaa palvelun päättymistä. Palveluiden välissä oliot (asiakas tai tilaus) joutuvat jonoihin, joissa ne odottavat seuraavaan palveluun pääsemistä.

Tulevat tapahtumat tallennetaan tapahtumalistaan, eli ”eventlist” -nimiseen olioon, jossa on prioriteettijonolista. Tämä lista lajittelee tapahtumat kronologiseen järjestykseen automaattisesti, ja siitä pystyy poistamaan aina viimeisimmän tapahtuman. A-vaiheessa ”Singleton” kello-olion kelloa siirretään aina viimeisimmän tapahtuman aikaan. Singleton tarkoittaa sitä, että kello olioita voi olla vain yksi, ja sen instanssia kutsutaan sen sijaan, että se luotaisiin joka luokassa uudelleen. Tämä on tärkeää, koska kaikkien luokkien pitää pysyä samassa ajassa ja päivittää samaa kelloa.

B-vaiheessa suoritetaan viimeisin tapahtuma loppuun. Tähän kuuluu olion jonosta pois vetäminen (olio on jonossa palvelun ajan, vaikkei jonota) ja sen lisääminen seuraavan palvelupisteen jonoon. Palvelupisteillä ja tapahtumilla on siis selkeä looginen järjestys, jonka läpi oliot kulkevat aina samalla tavalla. Esimerkiksi ensimmäinen tapahtuma on aina asiakkaan luominen, ja viimeinen sen poistuminen. Tässä välissä voidaan myös päivittää tuloksia, tietokantaa, tai suorittaa muita tapahtumakohtaisia asioita. Koska kyseisessä simulaatiossa mallinnetaan pizzeriaa, on sen rakenne kompleksi. Käyttäjä pystyy syötteillään vaikuttamaan pöytien, tarjoilijoiden, ja kokkien määrään. Useampi

pöytä johtaa siihen, että pöytäkohtaisia palvelupisteitä luodaan useampi, ja että asiakkaan pöytänumeroa pitää seurata jokaisessa B-vaiheen tapahtumassa, jotta asiakas pysyy saman pöydän palvelupisteiden jonossa. Asiakkaiden Lisäksi, pizzojen valmistuspisteitä pitää luoda kokkien määrän mukaan, ja pizza (tilaus) pitää aina asettaa sillä hetkellä lyhyimpään jonoon. Asiakkaiden, tilausten, tapahtumien, ja pöytien välillä on suora assosiaatio. Tämä helpottaa oikean olion tai pöydän löytämistä tilanteissa, joissa useampi olio liikkuu järjestelmässä samanaikaisesti. Tiedon pystyy välittämään myös tapahtuman kautta, jolloin tieto välittyy prosessien välillä helposti.

C-vaiheessa palvelu aloitetaan, jos asiakas on palvelupisteen jonossa ja palvelupiste on vapaa. Palvelupisteen vapaus ei aina ole niin yksiselitteistä, kuin että palvellaanko siellä sillä hetkellä asiakasta. Siihen voi myös vaikuttaa se, että onko tarjoilijoita vapaana, onko pöytiä vapaana, tai onko pizzauunissa tilaa. Jokainen palvelupiste käydään aina kerran läpi c-vaiheessa, riippumatta siitä, että pystyykö sen palvelua aloittamaan vai ei.

5.5 Ulkoisten tietovarastojen (tiedostot, tietokannat) kuvaukset.

5.5.1 SQL-Tietokantaratkaisu

Projektissa hyödynnetään paikallista SQL-tietokantaratkaisua simulaation asiakkaiden ja tilausten tallentamiseen. Olioita tallenetaan tietokantaan simulaation ajon aikana, ja niitä hyödynnetään lopuksi tulosten laskemiseen. Olio-taulukoita pystyy tarkastella, kunnes uusi simulaatio-ajo aloitetaan tai simulaattori käynnistetään uudelleen, jolloin taulut pyyhitään puhtaiksi. Tämä tarkoittaa, että vain viimeisimmät tulokset näkyvät tietokannassa. Taulukoiden päätarkoitus on antaa käyttäjälle lisäinformaatioita tulosten tarkasteluun ja analysointiin. Esimerkkinä kuvassa 13 näkyvä tietokantataulukko, johon asiakkaiden tiedot ovat varastoitu.

DTYPE	id	arrivalTime	exitTime	hasEaten	queueStartTime	table	totalTimeInQueue	type	order_aC_id
TakeAway	1	29.809439541645 95	44.198551109669616	false	44.134232055267 99	0	0	Takeaway	1
TakeAway	2	58.668126975838 28	69.638362439989 88	false	67.307303543595 27	0	0	Takeaway	2
Customer	3	87.438219594936 47	131.89209425667124	true	129.127566596252 88	0	0	Customer	3
Customer	4	87.438219594936 47	131.89209425667124	true	129.127566596252 88	0	0	Customer	4
Customer	5	97.213689318985 37	144.53065178728 69	true	142.707520538752	1	0	Customer	5
Customer	6	97.213689318985 37	144.53065178728 69	true	142.707520538752	1	0	Customer	6
Customer	7	97.213689318985 37	144.53065178728 69	true	142.707520538752	1	0	Customer	7
Customer	8	107.348018452371 15	160.35336245063615	true	158.306127909613 14	2	0	Customer	8
Customer	9	107.348018452371 15	160.35336245063615	true	158.306127909613 14	2	0	Customer	9
Customer	10	107.348018452371 15	160.35336245063615	true	158.306127909613 14	2	0	Customer	10
TakeAway	11	117.577818775520 24	149.6332938226855	false	148.231526663439 07	0	0	Takeaway	11
Customer	12	126.951581257112 08	176.96317717960684	true	174.788400393031 34	3	0	Customer	12
Customer	13	126.951581257112 08	176.96317717960684	true	174.788400393031 34	3	0	Customer	13
Customer	14	136.487370454192 22	178.00521353341557	true	175.424538009783 36	0	0	Customer	14
Customer	15	146.790145174584	188.34687265410648	true	185.888987065230 42	1	0	Customer	15
Customer	16	146.790145174584	188.34687265410648	true	185.888987065230 42	1	0	Customer	16

Kuva 13. Asiakas-taulukko tietokannassa

Olioiden tallentamisessa tietokantaan hyödynnetään Jakarta Hibernate API:ta. Hibernate mahdollistaa olioiden haun ja tallentamisen suoraan Java-luokan pohjalta, jolloin perinteisiä SQL-kyselyitä ei enää tarvitse suorittaa muutamaa poikkeusta lukuun ottamatta. Esimerkiksi olion tallentaminen tietokantaan onnistuu aloittamalla Hibernaten EntityManagerin kanssa transaktio, kutsumalla sen ”persist”-metodia, johon annetaan parametrina tallennettava olio, ja lopuksi suorittamalla transaktio. Kuvassa 14 näkyy kyseinen persist-metodi. Muut metodit, joissa tietokannan sisältö muuttuu, toimivat samalla periaatteella. Transaktio-osuus varmistaa, että tietokannan sisältö ei vaurioidu tai päivity epähalutulla tavalla, jos sen sisällä suoritettava toiminto epäonnistuu. Haut, joissa sisältö ei muutu, onnistuvat ilman transaktiota.

```
public void persist(Order order) {  4 usages  Goliathuzzzz
    EntityManager em = MariaDbJpaConnection.getInstance();
    em.getTransaction().begin();
    em.persist(order);
    em.getTransaction().commit();
}
```

Kuva 14. Tallennus-metodin toteutus tilaus-olioille.

5.5.2 Tietokannan käyttöönotto

Ennen simulaation ensimmäistä ajoa käyttäjän täytyy ajaa database-skripti. Kyseinen tiedosto löytyy projektin resurssi-kansion SQL-kansiosta. Skripti luo tietokannan sekä käyttäjän ”pizzauser”. Salasana löytyy skriptistä. Pizzauserilla on oikeus tarkastella simulaattorin tuloksia. Koska simulaattori hoitaa tietojen lisäämisen tietokantaan, ei käyttäjä tarvitse muita oikeuksia, kuin taulukoiden tarkastelu ja hakujen suorittaminen.

5.6 Testaus

Testauksen tavoitteena oli varmistaa, että simulaattori toimii odotetulla tavalla ja se täyttää sille asetetut vaatimukset. Simulaattorin testaaminen toteutettiin pääasiassa yksikkötesteillä, käyttäen JUnit-testauskehystä. Se soveltuu yksittäisten koodikomponenttien, kuten metodien ja luokkien testaamiseen. Näiden testien avulla pidetään huolta, että koodikomponentit toimivat oikein myös erillään muista osista.

Yleistä ohjelmiston testausta toteutettiin siten, että ohjelma pyrittiin saada kaatumaan ja tekemään virheitä. Yksi lähestymistapa tähän oli virheelliset syötteet, joiden avulla ohjelmisto saatiin yksinkertaisesti jumiin tai kaatumaan kokonaan. Toinen lähestymistapa oli käyttäjälle saatavilla olevat toiminnallisuutta hallitsevat painikkeet. Osa näistä elementeistä oli käyttäjälle toiminallisia testiajojen väärissä vaiheissa tuottaen virheellisiä tuloksia, tai yksinkertaisesti niiden interaktiota ei käsitelty enää ajonaikana.

5.6.1 Yleinen Testaus

Yleisessä testaamisessa etusivun syötekenttiin simulaatioaika, viive, pöytien, tarjoilijoiden ja kokkien lukumäärä annettiin virheellisiä ja puutteellisia arvoja. Näitä arvoja olivat esimerkiksi negatiiviset luvut, numeroista poikkeavat merkit tai kenttiä jätettiin kokonaan tyhjäksi. Simulaattori ajoi virheellisillä syötteillä joko jonkin aikaa ja kaatui tai se ei käynnistynyt ollenkaan vaan kaatui saman tien. Välillä ohjelma ajoi, mutta sen tuottamat tulokset olivat virheellisiä. Ohjelmiston täytyisi siis reagoida poikkeuksiin oikein ja ilmoittaa käyttäjälle, mikä syötteessä/syötteissä oli puutteellista tai virheellistä. Ohjelmistoon asetettiin poikkeuksenkäsittelyä ja käyttöliittymä antaa jatkossa ilmoituksen käyttäjälle, kun simulaatiota yritetään aloittaa, jos syötetty arvo on negatiivinen, merkki on numerosta poikkeava tai jokin tai jotkin syötekentät ovat tyhjiä.

Toiminnallisten painikkeiden kohdalla Specifications, Discount day, Start simulation, Add speed, Slow down ja Reset kokeiltiin, että painike välittää toimintonsa ja tekee mitä halutaan. Simulaation aloittaminen Start simulation painikkeella on mahdollista vasta, kun kaikki tarvittavat syötteet on annettu. Add speed painike laskee aikaisemmin syötteenä annettua viivettä, kunnes viive on nolla eikä mene sen alapuolelle. Slow down puolestaan nostaisi aikaisemmin syötteenä annettua viivettä. Viiveen muutosta pystyi seuraamaan käyttöliittymästä simulaation nopeuden muutoksena sekä konsolista, mihin viive tulostuksessa näkyi heti kun sen arvo muuttui. Add speed ja Slow down painikkeiden ei tulisi myöskään olla painettavissa vasta, kun simulaatio on käynnissä. Painikkeiden näkyvyys toiminnallisuus asetettiin epätodeksi, kun simulaatio ei ole ajossa.

Reset painikkeen puolestaan tulisi nollata kaikki tulokset ja arvot. Aluksi huomattiin, että tilastot ja syötteet nollautuivat oikein, mutta ohjelmaa jälleen ajettaessa

huomattiin joidenkin aktiivisten tuloksien jatkuvan siitä mihin ne olivat aikaisemmalla ajolla jääneet. Myös simulaatiota visualisoivat kuvakkeet hävisivät oikein, mutta jatkoivat siitä pisteestä mihin olivat jääneet. Ohjelmiston puolella tehtiin tarvittavat muutokset, minkä jälkeen tulokset nollautuivat oikeaoppisesti ja visualisointi alkoi sille tarkoitetusta kohdasta.

Specifications-sivulla tehtiin samanlaista testaamista kuin etusivulla, että syötteet ovat positiivisia numeroita. Todennäköisyyksien kohdalla oli pidettävä huolta, että niiden summa on 100 %. Tasajakauman tekijöiden summa tulee olla 1, joten syötteiden ei tule ylittää tätä. Menojen ja tulojen kohdalla testattiin, että niiden muutoksien vaikutukset näkyivät tulostaulussa. Todennäköisyyksien kohdalla pizzatyyppien yleisyys näkyi konsolin puolella selvästi, kun arvoja muutettiin. Jakauman kohdalla kokeiltiin erilaisia syötteitä keskiarvolle, pienellä keskiarvolla huomattiin, että saapumistapahtumien tiheys kasvoi ja kasvatettaessa keskiarvoa saapumistapahtumat tulivat harvemmaksi. Varianssin muutoksen piti vaikuttaa siihen miten paljon arvo poikkeaisi keskiarvosta. Tällaisen testaamisen perusteella rinnastettuna JUnit-testien kanssa varmistettiin, että simulaatio toimii odotetulla tavalla.

5.6.2 Junit-testit

Testeissä keskityttiin valittuihin ohjelman olennaisimpiin luokkiin ja metodeihin sekä niiden eristettyyn testaamiseen. Testiohjelmia pyrittiin käyttämään mahdollisimman monen eri paketin luokkiin.

simu.framework-paketti**:**

- **ArrivalProcessTest.java** testaa, että saapumisprosessit toimivat oikein ja että ne käynnistyvät ja tekevät mitä niiden pitää.
- **EventListTest.java** testaa tapahtumalistan toiminnallisuuden esim. tapahtumien lisäämisen ja poistamisen listasta.
- **EventTest.java** testaa yksittäisten tapahtumien oikeanlaisen toiminnan simulaatiossa**.**

simu.model-paketti:

- **FinanceTest.java** testaa että talousmalli toimii niin kuin pitää ja varmistaa, että kaikki sen hallitsemat laskelmat tehdään oikein.
- **OrderRandomizerTest.java** testaa, että tilausten luonti on satunnaista asetettujen parametrien mukaisesti.
- **OrderTest.java** testaa tilausprosessi on tehty oikeaoppisesti, kuten esim. tilausten luominen ja niiden käsittely.
- **PizzaServicePointTest.java** testaa palvelupisteen toimivuutta ja pitää huolen siitä, että tilaukset käsitellään oikein
- **WaiterTest.java** testaa, että tarjoilijat tekevät tehtävänsä ja tilausten käsittely tarjoilijoiden avulla toimii kuin odotettu.

view-paketti:

- **SimulatorGUITest.java** testaa TestFX-kirjaston avulla, että graafinen käyttöliittymä latautuu ja näkyy oikein.

6 Simulaattorin käyttöohje

Käyttöohje on laajuutensa takia liitteenä.

7 Tehdyt simulointikokeet

Simulointikokeet suoritettiin käyttämällä erilaisia lähtöarvoja ja niiden avulla saataisiin selville, miten muutokset asiakasvirrassa, työvoiman määrässä ja kustannuksissa mahdollisesti vaikuttavat pizzerian toimintaan. Simulointikokeilla haluttiin löytää esim. ne lähtöparametrien yhdistelmät, millä saataisiin ilmi selvästi häviötä tekevä tapaus sekä tapaus missä ravintola tekee voittoa. On myös tärkeää ottaa selvää, miten parametrien muutokset vaikuttavat asiakkaiden saapumiseen, jotta voidaan simuloida niin ruuhkaisia kuin myös hiljaisia päiviä pizzeriassa.

Simulointikokeilla käy ilmi miten paljon työvoimaa tulisi varata erilaisille kävijämäärille esim. montako tarjoilijaa tarvitaan, ettei tarjoilijoiden määrästä synny järjestelmään pullonkaulaa. On myös mahdollista, että tarjoilijoita on liikaa kävijämäärään nähden ja he ovat joutilaana ja heille maksetaan palkkaa, vaikka heitä ei varsinaisesti tarvitakaan.

Erilaisilla pizzojen sekä juomien hinnoilla ja kustannuksilla saadaan selville tulosnäytöltä tuottaako ravintola tappiota vai voittoa.

Simulointikokeilla voidaan testata ravintolan kuormitusta erilaisilla asiakasvirroilla muuttamalla todennäköisyysjakauman keskiarvoa ja varianssia. Näitä arvoja muuttamalla saadaan tehtyä asiakkaiden saapumistapahtumista ravintolaan tiheämpiä tai harvempia, sekä vaikutettua miten paljon saapumistapahtumat mahdollisesti poikkeavat keskiarvosta.

7.1 Tappiota tuottava tapaus

- Lähtöarvot:
    - Simulaatioaika = 1 päivä
    - Viive = 1 ms
    - Pöytiä = 1, tarjoilijoita = 1, kokkeja = 1
    - Pizzatyyppien hinnat ja kustannukset perusarvojen mukaiset (default)
    - Juomatyyppien hinnat ja kustannukset perusarvojen mukaiset (default)
    - Pizza- ja juomatyyppien valinta todennäköisyydet perusarvojen mukaiset (default)
    - Pizzatyyppien eri kokojen valinta todennäköisyydet perusarvojen mukaiset (default)
    - Jakauman keskiarvo = 30 (default)
    - Jakauman varianssi = 0.50 (default)
    - Ei ole alennuspäivä (default)

Tulos:

Ajamalla useamman ajon perusasetuksilla nähdään, että ravintola tekee tappiota näillä pöytä-, tarjoilija- ja kokkimäärillä. Tappiota syntyy väliltä 80-120 euroa yhdeltä päivältä. Asiakkaita saapuu väliltä 45-65 henkilöä päivässä ja noutotilauksia saapuu noin 10-20 päivässä. Kun saapuneiden asiakkaiden ja noutotilausten määrää verrataan palveltuihin asiakkaisiin ja valmistettuihin noutotilauksiin huomataan, että järjestelmään saapuu enemmän asiakkaita kuin heitä pystytään käsittelemään. Palveltujen asiakkaiden määrä on 25-35 välillä. On selvää että tarjoilijoiden tai pöytien määrää tulee nostaa, jotta pystytään tekemään parempaa tulosta. Noutotilaukset ravintola puolestaan pystyy

käsittelemään kaikki, koska niille ei kerkeä syntyä kovin suurta jonoa missään vaiheessa. Ravintolan hyötykäyttö on väliltä 12-17 %, mistä tiedetään ettei palvelua hyödynnetä oikein. Jonon pituus on väliltä 7-10 henkilöä pitkä.

7.2 Voittoa tuottava tapaus

- Lähtöarvot
    - Simulaatioaika = 1 päivä
    - Viive = 1 ms
    - Pöytiä = 3, tarjoilijoita = 1, kokkeja = 1
    - Pizzatyyppien hinnat: margherita 12.00, meatlover 14.00, vegan 12.00 Pizzatyyppien kustannukset: margherita 2.00, meatlover 4.00, vegan 3.00
    - Juomatyyppien hinnat: perusarvojen mukaiset (default) Juomatyyppien kustannukset: vesi 0.10, virvoitusjuoma 0.80, olut 1.00
    - Pizza- ja juomatyyppien valinta todennäköisyydet perusarvojen mukaiset (default)
    - Pizzatyyppien eri kokojen valinta todennäköisyydet perusarvojen mukaiset (default)
    - Jakauman keskiarvo = 30 (default)
    - Jakauman varianssi = 0.50 (default)
    - Ei ole alennuspäivä (default)

Tulos:

Tekemällä usean koeajon nähdään, että ravintola tekee voittoa hyvän määrän yhden päivän aikana. Voittoa kertyy väliltä 520-750 euroa. Asiakkaita saapuu väliltä 50-70 henkilöä ja noutotilauksia väliltä 10-20. Huomataan että pöytien määrän nostamisella ravintolassa pystytään palvelemaan useampaa asiakasta kerrallaan. Tarjoilijasta ja kokista saadaan enemmän hyötyä irti kun asiakkaita on ravintolassa pöydissä enemmän. Palveltujen asiakkaiden määrä on kutakuinkin joka ajolla yhtä paljon kuin ravintolaan saapuu asiakkaita. Noutotilauksetkin pystytään myös lähes aina valmistamaan kaikki. Ravintola hyötyy myös pizzojen muutaman euron korotuksella, sekä vähentämällä valmistuskuluja. Juomien tapauksessa vain hankintakuluja vähennettiin ja hinnat pysyivät samoina. Ravintolan hyötykäyttö oli silti vain 30-40 %,

mikä tarkoittaa, että ravintolan toiminnassa on vielä parannettavaa. Keskimääräinen jonon pituus oli vain 1-4 ihmistä pitkä, kun pöytiä on enemmän.

7.3 Ruuhkatapaus

- Lähtöarvot:
    - Simulaatioaika = 1 päivä
    - Viive = 1 ms
    - Pöytiä = 3, tarjoilijoita = 1, kokkeja = 1
    - Pizzatyyppien hinnat: margherita 12.00, meatlover 14.00, vegan 12.00
      Pizzatyyppien kustannukset: margherita 2.00, meatlover 4.00, vegan 3.00
    - Juomatyyppien hinnat: perusarvojen mukaiset (default)
      Juomatyyppien kustannukset: vesi 0.10, virvoitusjuoma 0.80, olut 1.00
    - Pizza- ja juomatyyppien valinta todennäköisyydet perusarvojen mukaiset (default)
    - Pizzatyyppien eri kokojen valinta todennäköisyydet perusarvojen mukaiset (default)
    - Jakauman keskiarvo = 15
    - Jakauman varianssi = 0.50 (default)
    - On alennuspäivä

Tulos:

Usean ajon jälkeen huomataan, että ravintola tekee runsaasti voittoa yhden päivän aikana. Voittoa kertyy väliltä 900-1200 euroa. Nyt todennäköisyysjaukaman keskiarvoa on laskettu puoleen aikasemmasta 30 -> 15 ja on alennuspäivä, mikä puolittaa keskiarvon ja laskee pizzojen hintoja 15 % alkuperäisestä. Nähdään, että asiakkaiden saapumistapahtumien tiheys on todella korkea verrattuna aikasempaan. Asiakkaita saapuu päivän aikana väliltä 210-240 ja noutotilauksia puolestaan 55-75. Palveltujen asiakkaiden määrä on kuitenkin vain väliltä 70-85 ja valmistuneita noutotilauksia on muutama vähemmän kuin itse saapuneita tilauksia. Ravintolan hyötykäyttö on jatkuvasti 100 %, mikä ilmentää, että kaikki resurssit hyödynnetään. Keskimääräinen jonon pituus on väliltä 25-35 henkilöä pitkä. Ravintolan tulisi siis lisätä työvoimaa, että näin suureen asiakasvirtaan pystytään vastaamaan tarpeeksi tehokkaasti.

7.4 Hiljainen tapaus

- Lähtöarvot:
    - Simulaatioaika = 1 päivä
    - Viive = 1 ms
    - Pöytiä = 3, tarjoilijoita = 1, kokkeja = 1
    - Pizzatyyppien hinnat: margherita 12.00, meatlover 14.00, vegan 12.00
      Pizzatyyppien kustannukset: margherita 2.00, meatlover 4.00, vegan 3.00
    - Juomatyyppien hinnat: perusarvojen mukaiset (default)
      Juomatyyppien kustannukset: vesi 0.10, virvoitusjuoma 0.80, olut 1.00
    - Pizza- ja juomatyyppien valinta todennäköisyydet perusarvojen mukaiset (default)
    - Pizzatyyppien eri kokojen valinta todennäköisyydet perusarvojen mukaiset (default)
    - Jakauman keskiarvo = 75
    - Jakauman varianssi = 0.50 (default)
    - Ei ole alennuspäivä (default)

Tulos:

Monen koeajon jälkeen huomataan, että ravintola jää joko hieman tappiolle tai hieman voitolle. Nettotulot ovat siis hyvin lähellä nollaa. Asiakkaita saapuu väliltä 15-25 henkilöä ja noutotilauksia väliltä 3-6. Palveltujen asiakkaiden määrä on miltein sama kuin sama kuin saapuneiden asiakkaiden määrä, satunnaisesti asiakas tai pari jää ilman palvelua. Noutotilaukset pystytään jälleen käsittelemään ja valmistamaan kaikki, koska jonoa ei kerkeä koskaan syntyä. Ravintolan hyötykäyttö on väliltä 4-6 %, mistä voidaan päätellä että pöytiä on liikaa ja tarjoilija sekä kokki ovat päivästä osan toimettomana. Keskimääräinen jonon pituus on tietenkin nolla, sillä kaikki saapuvat asiakkaat ja tilaukset päätyvät heti käsittelyyn. Todella harvalla asiakasvirralla siis ravintola tekee helposti tappiota, vaikka hinnat ja kustannukset olisivat asetettu kohtalaisen hyvin. Palvelupisteiden hyödyntäminen jää todella alhaiseksi, mikäli asiakkaita saapuu näin harvoin.

8 Yhteenveto

Projektin tavoitteena oli kehittää kolmivaiheinen jonotussimulaattori, joka mallintaa pizzerian toimintaa ja tukee taloudellisten ja tehokkuuteen liittyvien päätösten tekemistä. Simulaattorin avulla voidaan arvioida esimerkiksi ravintolan työntekijämäärän, kapasiteetin sekä annosten hintojen ja kustannusten vaikutuksia ravintolan kannattavuuteen ja toiminnan tehokkuuteen.

Simulaatio pohjautuu kolmivaiheiseen malliin, jossa pizzerian päivittäistä toimintaa analysoidaan työvoiman määrän, kapasiteetin, asiakasvirran ja kustannusten näkökulmasta. Simulointikokeiden avulla testattiin erilaisia lähtöarvoja ja niiden vaikutuksia taloudellisiin tuloksiin sekä resurssien käyttöön. Mallin avulla pystyttiin arvioimaan sekä tappiollisia että voitollisia skenaarioita, ja tunnistamaan tilanteita, joissa pizzaria toimi optimaalisesti tai resursseja käytettiin tehottomasti.

Simulointikokeet osoittivat, että työntekijämäärän, kapasiteetin ja hinnoittelun hallinta ovat keskeisiä tekijöitä pizzerian toiminnan sujuvuuden kannalta. Liian suuri tai liian pieni henkilöstömäärä ja kapasiteetti voi johtaa joko pullonkaulojen syntyyn tai työntekijöiden joutilaisuuteen, mikä heikentää ravintolan taloudellista tulosta ja asiakaspalvelun laatua. Erilainen annosten hinnoittelu simulaatiossa osoitti, kuinka hintarakenteen optimointi yhdistettynä tehokkaaseen resurssien käyttöön voi parantaa pizzerian taloudellista tulosta merkittävästi.

Simulaattori tarjoaa käyttäjälle arvokasta tietoa ravintolan toiminnan optimointiin. Simulaattorin tarjoamat tulokset auttavat tunnistamaan niin ongelmakohtia kuin kehitysmahdollisuuksia ravintolan toiminnassa, mikä voi parantaa ravintolan taloudellista kannattavuutta ja asiakaskokemusta.

Liitteet

1. Javadoc löytyy GitHubista. Javadoc pitää ladata zip-tiedostona ja purkaa. Tämän jälkeen voi avata index.html-tiedoston ja javadoc tulee näkyviin. Linkki javadoc.zip-tiedostoon. Vaihtoehtoisesti, jos projektitiedostot ovat ladattuina, voi javadocin avata suoraan projektin resurssikansiosta.
2. Käyttöohje löytyy GitHubista. Linkki käyttöohjeeseen.