TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Alari Lagle 179660IADB

ALUSTE PAIGUTAMISE RAKENDUSE LOOMINE HAVEN KAKUMÄE JAHISADAMA NÄITEL

Bakalaureusetöö

Juhendaja: Jaanus Pöial
PhD
Kaasjuhendaja: Andres Käver
BSc

Tallinn 2020

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Alari Lagle

18.05.2020

Annotatsioon

Käesolev lõputöö käsitleb optimaalse pakkimise temaatikat. Uuritakse olukorda, kus kindlale alale on vaja paigutada teatud hulk elemente. Kulu vältimiseks on tarvilik seda teha võimalikult optimaalselt. Käesolevas töös on fookus seatud aluste hoiustamise planeerimisele Haven Kakumäe jahisadama näitel. Lõputöö eesmärgiks on kaardistada Haven Kakumäe jahisadama vajadused aluste hoiustamisel ning luua rakendus, mis tagaks ellingutes aluste paigutamisel võimalikult optimaalse ruumikasutuse.

Ülesandepüstitusele tuginedes on pakkimisalgoritmide strateegiate hulgast valitud BL algoritmi põhimõte ja täiendavalt kasutatud geomeetrilist tehnikat. Rakendus on kirjutatud hajusa süsteemina kasutades klient-server-mudelit. Kliendi poolele on ehitatud kasutajaliides, mille kaudu on võimalik aluseid hallata ehk neid lisada, vaadata ja kustutada. Ühtlasi võimaldab rakendus pakkimisskeeme moodustada ja genereeritud pakkimisplaane näha. Toimiv rakendus tagab sisestatud aluste optimaalse pakkimise. Lisaks arvestab rakendus vajadusel olukorraga, kus mõned alused on vaja ellingust eelisjärjekorras välja liigutada.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 34. leheküljel, kolme peatükki, 12. joonist.

Abstract

The present thesis addresses the problem of optimal packing. The question examined concentrates on how to place certain elements to a defined area. To minimize the loss of resources, it is important to do it as optimally as possible. The current thesis focuses on packing vessels. The problem is based on the situation of Haven Kakumäe marina. Planning the positioning of the boats in the hangars has not been automated and therefore does not result in the most efficient outcome.

The research problem focuses on placing non-standard objects to a certain area. The purpose of this thesis is to survey the needs of Haven Kakumäe marina and to create an application that would ensure the most optimal use of the hangars floor surface. The application improves the planning process of packing the vessels. It also takes into account the restrictions in the marina.

To ensure the most optimal solution of packing the vessels, Bottom Left algorithm is chosen. Geometrical technique is used, in order to exclude the overlap of the elements. The application is a distributed system using client-server-model. The server side of the application is created with .NET and C#. The server side is developed in a layered structure. The client side is created with Vue.js. Libraries are added to facilitate programming. In addition, TypeScript is used. The user interface created in the client side enables to use the application. It is possible to manage the vessels, which means to add, look and delete them. In addition, the application enables to form packing schemes and to display them.

Main functionality of the application is to pack all the vessels on the list and to do it as optimally as possible. In addition, the application packs the prioritized vessels in the front of the hangar, in order to take them out prior to the other vessels. As there are limitations to packing the larger vessels, the application takes them into account and packs the vessels accordingly.

The thesis is in Estonian and contains 34 pages of text, 3 chapters, 12 figures.

Lühendite ja mõistete sõnastik

.NET	Microsoft'i programmeerimisraamistik
Alus	paat, jaht, purjepaat
Angular	JavaScript'i raamistik
BFDH	*Best-Fit Decreasing Height*, pakkimise strateegia
BL	*Bottom Left*, pakkimise algoritm
BLL	*Business Logic Layer*, serveri struktuuri äriloogika kiht
BLL-DTO	BLL kihis olev DTO
BootstrapVue	CSS'i teek
C#	*C sharp*, Microsoft'i programmeerimiskeel
CSS	*Cascading Style Sheet*, kaskaadlaadistik märgistuskeele kujudamiseks
DAL	*Data Access Layer*, serveri struktuuri kiht andmete ligipääsu kiht
DAL-DTO	DAL kihis olev DTO
DTO	*Data Transfer Object*, andmeedastusobjekt.
Elling	ruum, kuhu pakitakse alused
F#	*F sharp*, Microsoft'i programmeerimiskeel
Factory	Disainimuster
FFDH	*First-Fit Decreasing Height*, pakkimise strateegia
HTML	*Hyper Text Markup Language*, hüperteksti märgistuskeel
Isend	klassiobjekt
Java EE	*Java Platform, Enterprise Edition*, Oracle'i programmeerimisplatvorm
JavaScript	Skriptimiskeel
JWT	*JSON Web Token*
Konva.js	Teek
LINQ	*Language-Integrated Query*, programmeerimiskeelde integreeritud spetsiaalne keel andmebaasi päringute tegemiseks
NFDH	*Next-Fit Decreasing Height*, pakkimise strateegia
NFP	*No-Fit Polygon*, mittelõikuvuse piirkond
React	JavaScript'i raamistik
Repository	Repositoorium
TypeScript	Tüübitud Javascript
UOW	*Unit Of Work* disainimuster
Vue.js	JavaScript'i raamistik

Sisukord

Jooniste loetelu

Sissejuhatus

Käesolev lõputöö käsitleb optimaalse pakkimise temaatikat. Probleem on mitmete sektorite ülene, esinedes valdavalt töötlevas tööstuses. Tegemist on olukorraga, kus kindlale alale on vaja paigutada teatud hulk elemente. Kulu vältimiseks on tarvilik seda teha võimalikult optimaalselt. Analoogset probleemipüstitust on võimalik laiendada teistesse valdkondadesse. Käesolevas töös keskendutakse aluste hoiustamise planeerimisele. Seda on tehtud Haven Kakumäe jahisadama näite toel. Aluste hoiustamise planeerimine on sadama personalile igahooajaline ülesanne, mille täitmine pole seni automatiseeritud. Sellest lähtuvalt on tegevuse tulem valdavalt intuitiivne ning nõuab palju käsitööd.

Lõputöö käsitleb uurimisprobleemina objektide optimaalset paigutamist teatud alale. Antud töös on konkreetselt lähtutud aluste paigutamisest Haven Kakumäe jahisadama kahte ellingusse. Lõputöö eesmärgiks on kaardistada Haven Kakumäe jahisadama vajadused aluste hoiustamisel ning luua rakendus, mis tagaks ellingutes aluste paigutamisel võimalikult optimaalse ruumikasutuse. Rakenduse loomine hõlbustab aluste paigutamise protsessi ning aitab vältida potentsiaalseid inimtegevusest tulenevaid eksimusi.

Lõputööle seatud ülesanded on järgnevad:

- sobiva algoritmi leidmine ning selle kohandamine eesmärgile vastavaks,
- kohaste vahendite leidmine ja raamistike valik,
- Haven Kakumäe jahisadama tingimuste kaardistamine,
- funktsionaalsete ja mittefunktsionaalsete nõuete määramine,
- rakenduse loomine.

Optimaalse pakkimise teemat on käsitletud nii korrapärastele kui ka korrapäratutele elementidele tuginedes. Töö fookusest lähtuvalt on tähelepanu suunatud irregulaarsete elementide pakkimist puudutavatele allikatele. Optimaalset pakkimist käsitlevaid algoritme on mitmeid, kuid nende üksühene ülekandmine tööle seatud eesmärgi saavutamiseks ei päde. Seetõttu on vajalik algoritme uurides ja varasematele allikatele tuginedes leida lahendus, mis lähtuks seatud tingimustest ja konkreetsest probleemipüstitusest.

Lõputöö jaotub kolme peatükki. Neist esimene hõlmab optimaalse pakkimise problemaatikat. Peatükis on avatud probleemi olemust ning kirjeldatud selle lahendamiseks kasutatavaid algoritme. Ülesandepüstitusele tuginedes on kirjeldatud uurimistöö probleemi lahendamiseks kasutatavat algoritmi ja selle kohandamist tulemuse saavutamiseks. Teises peatükis on välja toodud vahendid rakenduse loomiseks. Viimane eeldab nii serveri kui ka kliendi poole loomist, mille tarbeks kasutavaid raamistikke on peatükis kirjeldatud.

Kolmas peatükk keskendub konkreetse rakenduse loomisele. Selle eeldusena on antud ülevaade lähteolukorrast, sealhulgas ellinguid ja pakitavaid aluseid puudutavast. Lisaks on toodud välja rakendusele esitatavad funktsionaalsed ja mittefunktsionaalsed nõuded. Peatükk päädib töötava rakenduse kirjeldamisega, kuhu on lisatud ka testjuhud.

Allikatena on kasutusel valdavalt algoritme puudutav kirjandus ning optimaalse pakkimise probleemi käsitlevad artiklid. Vaatlusaluse olukorra kaardistamiseks on läbi viidud intervjuud Haven Kakumäe jahisadama kapteniga. Intervjuudest selguvad täpsemad võimalused ja kitsaskohad, millele rakendust luues tähelepanu pöörata.

1. Optimaalse pakkimise problemaatika

1.1 Probleemi olemuse kirjeldus

Käesolev töö keskendub optimaalse pakkimise teemale. Küsimus puudutab valdkonnaüleselt erinevaid sektoreid, mistõttu on teema varasemalt olulisel määral käsitlust leidnud. Lahendatakse olukorda, kus teatud alale on vaja paigutada kindel hulk elemente. Seejuures ei tohi elemendid üksteisega kattuda. Ühtlasi on oluline, et paigutamine leiaks aset võimalikult väikese ruumikaoga. Seega tuleb tagada, et pakitud elemendid oleksid võimalikult lähestikku ning kataksid kogu ala optimaalselt.

Taolisi optimaalse pakkimise ja lõikamise probleeme esineb eelkõige puu-, klaasi-, terase- ja nahatööstuses. Lisaks leidub neid ka ajalehetööstuses ning konteinerite ja autode pakkimisel. [1, pp. 36-1] Valdavalt on optimaalse pakkimise probleem omane tootmisele spetsialiseerunud ettevõtetele. Seda põhjusel, et paljudes tootmistes on vajalik konkreetsete detailide väljalõikamine. Metallitööstuses on näiteks tarvilik metalldetaile metall-lehelt välja lõigata. Seejuures on tähtis, et detailid saaksid paigutatud võimalikult kompaktselt. Sel moel tekib võimalikult vähe raisatud materjali, mis osutuks tootmisel otseseks kuluks. Selline detailide optimaalne tasapinnale paigutamine on üks konkreetne näide pakkimise problemaatikast. [2, p. 374]

Probleemi edasi arendades jõuame järgmise pakkimist puudutava küsimuseni. Kui detailid on optimaalselt ära paigutatud ja välja lõigatud, on need detailid vaja pakkida transpordimasinatesse, hinnates seejuures ka masinate vajadust. Seega esineb pakkimist puudutavaid probleeme laialdaselt. Nende spetsiifilised iseärasused võivad varieeruda, kuid küsimuse olemus jääb samaks. Taolised pakkimise ja optimeerimise ülesanded on NP-keerukad. Selle keerukusastmega on isegi kõige elementaarsemad pakkimisprobleemid. [2, p. 374]

Keerukusastmete alusel jaotatakse probleemid kolme kategooriasse. Esimeses kategoorias asuvad probleemid, millele on leitud algoritm, mis lahendab need polünomiaalse ajaga. Siia alla kuuluvad näiteks otsimise algoritmid keerukusega $\Theta(\log n)$ ja sorteerimise algoritmid keerukusega $\Theta(n \log n)$. Need algoritmid on polünomiaalse keerukusega elementide arvu n suhtes. Teise kategooriasse kuuluvad probleemid, mille kohta on tõestatud, et neid ei ole võimalik lahendada ehk

mittelahenduvad probleemid. Nendele ei saa anda eitavat või jaatavat vastust. Siia alla kuulub näiteks peatuvuse probleem, mille mittelahenduvuse tõestas Allan Turing. Kolmanda kategooria alla kuuluvad NP-keerukad probleemid. Siin esinevad kõik probleemid, millele ei ole leitud polünomiaalse keerukusega algoritme. Seejuures ei ole ka suudetud ära tõestada, et selliseid algoritme ei eksisteeri. Kategooria alla kuuluvad peaaegu kõik pakkimisega seotud probleemid. [3, pp. 402 - 403]

Tootmises esinevaid pakkimisega seotud probleeme on võimalik üle kanda teistesse valdkondadesse. Lahendamispõhimõtted jäävad seejuures võrdlemisi sarnaseks, hoolimata pakitavate elementide eripärast. Käesolevas töös on pakkimisprobleemi lahendamine seotud aluste hoiustamise planeerimisega. Nimelt on toodud optimaalse pakkimise probleem jahisadama konteksti. Ülesandeks on paigutada sadamas hoiustatavad alused ellingutesse. Töö teostatakse Haven Kakumäe jahisadama näitel, mistõttu on lõputöö koostamisel aluseks võetud jahisadama parameetrid ja eripärad.

Sadama personal on igal sügisel probleemi ees, kuidas täita ellingud alustega niimoodi, et need oleks optimaalselt pakitud. Kuna sadamal on varakult teada, millised alused talveks ellingutesse pakitakse, on probleemi lahendamine mõnevõrra lihtsam. Palju keerulisem oleks, kui nimekiri jooksvalt muutuks. Sellest olenemata on erinevaid faktoreid üsna palju, mis probleemi keerukaks muudavad. Konkreetses ülesandepüstituses on probleemi käsitletud kahemõõtmelisena. See tähendab, et aluste paigutamisel arvestatakse sellega, kuidas täita ellingu ristkülikukujuline põrand. Võimalusel oleks hea arvestada olukorraga, kus mõni alus oleks vaja varem välja liigutada. Mõnevõrra lihtsamaks teeb ülesande veel see, et aluseid pakkivate kraanade sõidu iseloomu meeles pidades saab aluseid paigutada ainult ühtepidi, nimelt ahter ellingu seina poole.

1.2 Kasutatava algoritmi kirjeldus

1.2.1 Sobiva algoritmi leidmine

Ülesandepüstitusele vastava algoritmi leidmiseks on autor analüüsinud pakkimist käsitlevaid algoritme. Välja on toodud erinevate algoritmide strateegiad, mille põhjal hinnatud nende kohasust püstitatud ülesannete teostamisel. Kuigi pakkimisalgoritmid on enamasti kirjeldatud ristkülikutele tuginedes, on neid võimalik rakendada ka

ebaregulaarsete elementide pakkimisel. Elementide kattuvuse tuvastamine osutub keeruliseks, kuid selle lahendamiseks on võimalik kasutada erinevad tehnikad.

Enamik kirjanduses käsitlust leidvatest algoritmidest on kirjeldatud kui ahned algoritmid. Nende alla kuuluvad ka pakkimisalgoritmid. Ahned algoritmid on jaotatud kahte klassi. Esimesse kuuluvad ühe faasi algoritmid. Nende abil pakitakse elemendid lõplike mõõtmetega alale. Teise klassi kuuluvad kahe faasi algoritmid. Nende alusel pakitakse elemendid ühele rajale, mis omab küll laiust, kuid on lõputu kõrgusega. Teises faasis kasutatakse esimese faasi tulemust, et pakkida elemendid lõplike mõõtmetega alale. [4, p. 6]

Lisaks sellele on paljud pakkimisalgoritmide lähenemised riiulalgoritmid (*shelf algorithms*). See tähendab, et elemendid asetatakse vasakult paremale ritta. Neid ridu nimetatakse riiuliteks. Esimene riiul on pakitava ala allosas ja järgnevad riiulid tekivad eelmise rea kõige suurema kõrgusega elemendi pealt tõmmatud horisontaalile. Riiulitesse pakkimisel on klassikaliselt kolm erinevat strateegiat. Kõik strateegiad sorteerivad eelnevalt pakitud elemendid kõrguse järgi kahanevalt. Olgu x hetkel pakitav element ja s viimane riiul. [4, p. 6]

Joonisel 1 on kujutatud riiulitesse pakkimise strateegiad. Esimene strateegiatest on NFDH (*Next-Fit Decreasing Height*) strateegia. Kui element x mahub riiulisse s, siis paigutatakse see seal võimalikult vasakule. Vastasel juhul luuakse uus riiul (s := s + 1) ja element paigutatakse sinna reastatult vasakule. [4, p. 6] Teine strateegia on FFDH (*First-Fit Decreasing Height*) strateegia. Element paigutatakse esimesse riiulisse, kuhu see mahub. Kui sellist riiulit ei ole, siis luuakse uus riiul samamoodi nagu NFDH puhul. [4, p. 6] Kolmas strateegia on BFDH (*Best-Fit Decreasing Height*) strateegia. Element pakitakse sellisesse riiulisse, kuhu see mahub ja millel on horisontaalselt kõige väiksem kasutamata ruum. Kui sellist riiulit ei ole, siis luuakse uus riiul samamoodi nagu NFDH puhul. [4, p. 6]

Joonis 1. Elementide riiulitesse pakkimise strateegiad. [4, p. 7]

On ka algoritme, mis ei kasuta tulemuse saamiseks riiuleid. Selline kõige klassikalisem lähenemine on BL (*Bottom Left*) algoritm. Algoritmi sisuks on asetada element nii madalale kui võimalik järjestatult vasakule. Nagu ka eelnevate algoritmide korral, sorteeritakse sellegi puhul pakitavad elemendid vastavalt mingile suurusele, näiteks kõrgus või laius. Vastasel juhul ei pruugi algoritmi tulemus olla kõige optimaalsem. [4, p. 10]

Seni on käsitlust leidnud ristkülikute pakkimine. Kuid on olemas ka ebaregulaarsete elementide pakkimise algoritmid. Selliseid algoritme kasutatakse näiteks rõiva- ja laevatööstuses. Eesmärk on paigutada etteantud elemendid teatud alale, millel on kindel laius, kuid varieeruv kõrgus. Seejuures tuleb aga elementide paigutamine saavutada kõige minimaalsema kõrgusega. Suurim erinevus ebaregulaarsete ja regulaarsete elementide (näiteks ristkülikute) paigutamisel on see, et ebaregulaarsete objektide puhul on keeruline kontrollida, kas elemendid on paigutatud nii, et need ei lõikuks omavahel. Selleks, et seda tuvastada, on olemas mõned lähenemistehnikad. [1, pp. 36-10, 36-11]

Joonisel 2 on kujutatud geomeetrilist tehnikat, mida kasutatakse elementide lõikuvuse või mittelõikuvuse tuvastamiseks [1, pp. 36-11]. On antud kaks hulknurka A ja B. Hulknurga A ümbrust, mis on vajalik hulknurga B takistamatuks liigutamiseks A ümber, nimetatakse mittelõikuvuse piirkonnaks (*no-fit polygon*, NFP). Selle leidmiseks tuleb liigutada hulknurka B ümber A, samal ajal kui A on paigal. Hulknurka B tuleb liigutada nii, et see alati puudutab A-d, aga mitte kunagi ei oma sellega nullist suuremat ühisosa. Selle liikumisega moodustatakse ümbrus NFP_{AB}. Selleks, et seda moodustada, tuleb määrata hulknurga B punkt, mis joonistab kujundi samal ajal, kui hulknurk B liigub ümber A. [5, p. 3]

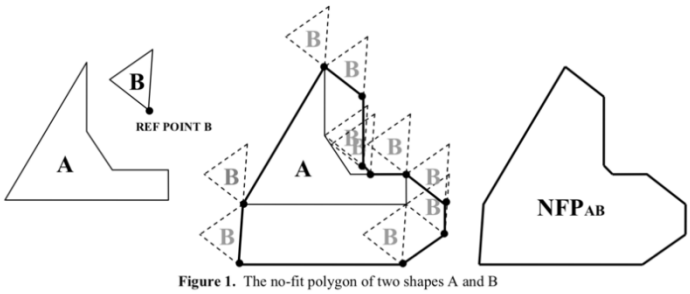
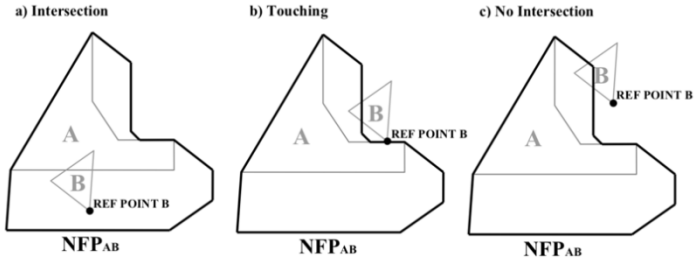


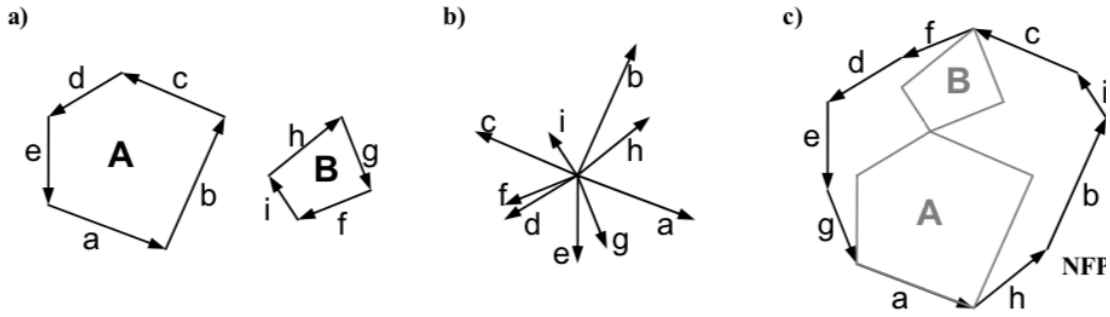
Figure 1. The no-fit polygon of two shapes A and B



Joonis 2. Hulknurkade kattuvuse tuvastamine mittelõikuvuse piirkonna abil. *Using the no-fit polygon to test for intersection between polygons A and B.* [5, p. 3]

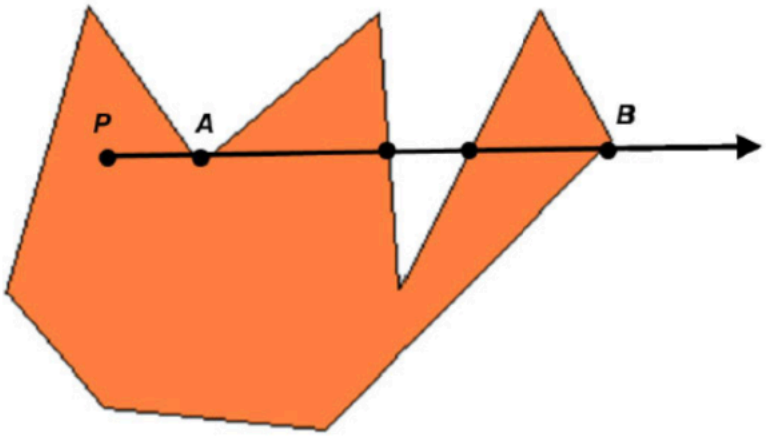
Selleks, et tuvastada, kas kaks hulknurka lõikuvad ehk omavad nullist suurema mõõduga ühisosa, kasutatakse ümbrust NFP_{AB} ja hulknurga B määratud punkti. Kui B määratud punkt peaks olema NFP_{AB} sees, siis A ja B lõikuvad. Kui B määratud punkt on NFP_{AB} piiril, siis A ja B puutuvad kokku. Juhul kui B määratud punkt on väljaspool NFP_{AB} hulknurka, siis A ja B ei lõiku ega ka puutu kokku. [5, p. 3]

Mittelõikuvuse piirkonna loomise tehnikad jaotatakse hulknurkade kuju järgi, mille vahel piirkonda otsitakse. Kujult jaotuvad hulknurgad kumerateks ja mittekumerateks. Mittekumerate hulknurkade vahel mittelõikuvuse piirkonna leidmise tehnikaid ei pea autor tarvilikuks kirjeldada, kuna töö eesmärgile tuginedes käsitletakse vaid kumeraid hulknurki. Joonisel 3 on kujutatud kahe kumera hulknurga vahelise mittelõikuvuse piirkonna leidmist. Tegemist on kõige tavalisema mittelõikuvuse piirkonna leidmise juhuga. On antud kaks kumerat hulknurka A ja B. Mõlema hulknurga külgedele tuleb määrata suunad. A küljed saavad suuna vastupäeva ja B küljed päripäeva. Nüüd tuleb kõik küljed tuua ühte punkti kokku suunaga sellest punktist eemale. Sellega luuakse suunatud külgede järjekord. Kui sellest järjekorrast hakata vastu päeva neid suunatud külgi üksteise järel kokku asetama, siis saadakse otsitav mittelõikuvuse piirkond. [5, p. 6]



Joonis 3. Mittelõikuvuse piirkonna moodustamine kumerate hulknurkadega. *No-fit polygon generation with convex shapes.* [5, p. 6]

Kui mittelõikuvuse piirkond on leitud, saab vaadata, kas kujundid kattuvad. Joonisel 4 on näidatud punkti P asukoha tuvastamine. Punktist P moodustatakse vektor niimoodi, et vektori y-koordinaat jääb samaks aga x-koordinaat määratakse lõpmata suureks või siis suuremaks kui mittelõikuvuse piirkonna kõige parempoolsem punkt. Seejärel vaadatakse kui mitu korda vektor P polügooni külgi läbib. Kui külgede läbimine on paarisarv, siis on punkt polügoonist väljas ja kui paaritu arv, siis on punkt polügooni sees. [6, p. 10]



Joonis 4. Punkti P asukoha tuvastamine mittelõikuvuse piirkonnas. *Ray from point P to the right actually touches 4 times the shape boundaries. The ray crosses the shape at vertex B. In contrast, the ray touches vertex A only tangentially and does not cross the shape at this point. Therefore, the count for crosses is 3. Since 3 is an odd number, we conclude that P is inside the shape.* [6, p. 10]

Võttes arvesse töös püstitatud ülesande eripära on selle lahendamiseks mõistlik kasutada BL algoritmi. Elementide lõikuvuse või mittelõikuvuse hindamine teostatakse geomeetrilise tehnika toel. Seejuures on aluseid vaadeldud lihtsustatult kumerate hulknurkadena.

1.2.2 Algoritmi kasutamine aluste pakkimisel

Tööle püstitatud eesmärgi täitmiseks on vaja kombineerida eelnevalt kirjeldatud teoreetilisi lähenemisi. Viimased on vaja omavahel siduda, et saavutada etteantud tingimustele vastav lahendus. Samuti peab pakitavate elementide kuju valik lähtuma elulistest proportsioonidest, mis tagab kohase tulemuse ja vastavuse optimaalsuspõhimõttele.

Käesolevas töös otsitakse lahendust aluste optimaalse pakkimise probleemile. See tähendab, et võimalikult ruumisäästliku tulemuse saamiseks on vaja arvestada aluste vöörinurgaga. Lihtsustatult võiks probleemi vaadelda ristkülikute pakkimisena, kuid sel juhul jääks liialt palju kasutamata ala. Seetõttu on algoritmide kasutamisel lähtutud ebakorrapäraste elementide paigutamisest.

Aluste kuju on võimalik võrrelda kumerate hulknurkadega. Seetõttu leitakse pakitavate elementide kattuvus või mittekattuvus kumeratele hulknurkadele tuginedes. Ristküliku asemel kumera hulknurga kasutamine aitab minimeerida ellingu kasutamata jäävat põrandapinda. Sadama jaoks on see oluline tulude maksimeerimise kontekstis, kuna ellingusse on oluline paigutada võimalikult palju aluseid (Lisa 1). Aluse vöörinurgaga arvestamine võimaldab neid veel tihedamalt paigutada. Käesoleval juhul on kõigile alustele kohaldatud 60 kraadine nurk (Lisa 2).

Lähtudes pakkimisel riiulalgoritmidest, tuleks elemendid ehk konkreetsel juhul alused paigutada nii-öelda riiulitesse. Aluste paigutamisel ellingusse alustatakse sel juhul seina äärest ning järgmise riiuli põhi tuleneks pikima aluse vöörist. Esimese riiuli pakkimise puhul tundub see mõistliku lahendusena. Järgmise riiuli põhi tõmmatakse aga horisontaalselt kõige pikemast elemendist lähtuvalt. See võib luua olukorra, kus esimesele riiulile jääb üsna palju kasutamata ala. Kui aluste pikkuste vahe on väga suur, ei võimalda algoritm tagada kõige optimaalsemat lahendust. Pakitud pinna maht sõltub sellest, kui erineva suurusega alused riiulitele pakitakse.

Eelnevast lähtudes ilmneb, et riiulalgoritme ei ole mõistlik kasutada. Seda enam, et olemas on ka teistele põhimõtetele tuginevad algoritmid, mis ei kasuta riiuleid. Efektiivse pakkimise võimaldab tagada BL algoritm. Paremate tulemuste saamiseks nõuab algoritm, et pakkimist ootav nimekiri oleks mingi suuruse järgi sorteeritud.

Alused pakitakse ellingusse ahter vastu tagaseina. Sorteerimine võiks toimuda esmalt kaalu ja seejärel pikkuse järgi.

Piiranguid aluste pakkimisel seab eeskätt raamkraana kasutamine. Seda kasutatakse aluste puhul, mis kaaluvad rohkem kui 25 tonni. Raamkraana nõuab mõlemalt poolt alust meetri ulatuses ruumi. Seega ei ole võimalik suuri aluseid kõrvuti paigutada. Piiranguga tuleb arvestada aluste nimekirja koostamisel. Sorteerimisel tuleb arvesse võtta, et üle 25 tonni kaaluvale alusele peaks järgnema väiksem alus, mida ei ole vajalik raamkraanaga liigutada. (Lisa 1)

Kuna tegemist on ebaregulaarsete elementide pakkimisega, tuleb kasutada mittelõikuvuse piirkonna leidmise võtet. See tagab, et alused pakitakse üksteisele võimalikult lähedale, ilma et nad kattuksid üksteisega. Tehnika abil on võimalik leida kõik positsioonid, kus kaks elementi omavahel ei kattu. Millegi põhjal on aga vaja otsustada, milline positsioon on kõige sobivam. Selle tarbeks on võimalik kasutada BL algoritmi. Valik tehakse selle järgi, milline positsioon on kõige madalamas kihis ja samas ka kõige vasakpoolsem.

Lahenduse leidmiseks on esmalt ülesandeks sorteerida pakitavate aluste nimekiri vastavalt eelnevalt kirjeldatud kriteeriumitele. Seejärel tuleb paigutada esimene alus ellingusse. Kui esimene on paigutatud, siis lähtudes mittelõikuvuse piirkonnast, pakitakse järgmine alus. Parima positsiooni valimiseks tuleb kasutada BL algoritmi põhimõtet. Kui valik on tehtud, siis saab liikuda uue pakitava aluse juurde. Protsessi jätkatakse kuni ellingute täitumiseni.

2. Vahendite valik

Käesoleva töö praktiline väljund on hajusa veebirakenduse loomine. Viimane peab lahendama tööle püstitatud probleemi, milleks on aluste optimaalne paigutamine ellingutesse. Rakenduse loomise eelduseks on sobiv ja kohane vahendite valik. Alljärgnevalt on välja toodud potentsiaalsed võimalused rakenduse loomiseks ning põhjendatud nende hulgast sooritatud valikuid.

Hajusa veebirakenduse realiseerimisel on kasutatud klient-server-mudelit. See tähendab, et on olemas server, mis haldab mingisugust ressurssi. Samas eksisteerib ka klient, kellel on soov konkreetset ressurssi kasutada. Klient saadab serverile nõude, mis sisaldab soovi kasutada viimase hallatavat ressursi. Kui nõue osutub serverile vastuvõetavaks, teostab server nõutud tegevused. Vajadusel saadab server ka kliendile vastuse. [7, p. 18]

Mudeli valikust tingituna hõlmab rakenduse loomine endas nii serveri kui ka kliendi poole arendamist. Neist esimese puhul on valdavalt kasutusel kas Oracle'i Java EE või Microsoft'i .NET raamistik. Valiku puhul on autor tuginenud senistele kogemustele ning koolis omandatule. Sellest lähtuvalt on jätkatud .NET raamistiku ülevaadet. .NET toetab erinevaid programmeerimiskeeli, mille hulgast Microsoft ise toetab ja arendab kolme keelt: C#, F# ja Visual Basic. C# on lihtne ja võimas keel, mis on tugevalt tüübitud ja objektorienteeritud programmeerimiskeel. F# on funktsionaalne programmeerimiskeel, mis toetab ka traditsioonilist objektorienteeritud ja imperatiivset programmeerimist. Visual Basic on lihtsalt õpitav ja ligilähedane tavalisele inimkeelele. Tihtipeale on see lihtne just inimestele, kes on äsja tarkvaraarendusega kokku puutunud. [8]

Serveri poole loomisel kasutatakse C#'i. Programmeerimiskeele rakendamise kasuks räägivad mitmed võimsad funktsionaalsused, näiteks mitteeksisteerivad andmetüübid, delegaadid, lambda avaldised ja LINQ (Language-Integrated Query). Märkimisväärseks võib loetelust pidada eeskätt viimast, mille abil on võimalik koodis kasutatavate andmebaasipäringute puhul kasutada programmeerimiskeelde integreeritud spetsiaalset keelt. Seda funktsionaalsust näiteks Java's ei ole. [9] Lisaks on ka autoril varasem kokkupuude C#'iga, mis toetab keele kasutamist käesoleva probleemi lahendamisel.

Kliendi poole kirjutamiseks kasutatakse HTML'i (*Hyper Text Markup Language*), CSS'i (*Cascading Style Sheet*) ja JavaScript'i. HTML märgendeid kasutades luuakse veebilehtedele vajalik struktuur. CSS'i toel muudetakse olemasolev struktuur presenteeritavaks. Selle abil on võimalik lisada veebilehele disain. Muuhulgas saab kirjeldada erinevate elementide värvid, paigutused, suurused jne. Kuid selleks, et panna veebilehel midagi liikuma või värvi muutma, kasutatakse JavaScript'i. Viimane on skriptimiskeel, millega lisatakse veebilehele funktsionaalsus. [10]

JavaScript'i abil kirjeldatakse ära kõik kasutaja tegevused. Sinna alla kuulub näiteks juht, kuidas veebileht reageerib nupu vajutusel. JavaScript'i abil ei ole vaja veebilehte pärast muudatuse tegemist uuendada. Ühtlasi valideeritakse kasutaja sisend JavaScript'i abil enne, kui pöördutakse serveri poole. [11] JavaScript'i käitatakse otse veebibrauseris, mis muudab selle väga kiireks. Ühtlasi langeb selle kasutamisel serveri koormus väiksemaks, kuna seda käitatakse kliendi pool. Paljud inimesed siiski keelavad põhjendamatult oma brauseris JavaScript'i kasutuse. Nimelt kardetakse pahatahtlikku ärakasutamist. Kuid keelates skriptimiskeele kasutuse, veebileht ei tööta. Siiski räägib JavaScript'i kasuks asjaolu, et seda toetab enamik brausereid. Kuid erinevad brauserid võivad interpreteerida seda erinevalt. [12]

JavaScript'il puuduvad andmetüübid, mistõttu võivad programmeerimisel sisse tulla vead. See võib juhtuda unustades deklareerida mõnda muutujat või kutsudes välja funktsiooni, mida tegelikult ei eksisteeri. Samuti võib töötava koodi katki teha funktsiooni vale parameetriga välja kutsudes. JavaScript'is puudub ka programmeerimiskeskkonna toetus, kus koodi kirjutatakse. See tähendab, et koodi kirjutamisel ei vihjata õigetele isendimuutujatele. Olukorra parandamiseks on olemas TypeScript. Viimane on oma olemuselt sama, mis JavaScript, kuid sisaldab andmetüüpe. TypeScript'i kasutamine muudab koodi kirjutamise lihtsamaks ja aitab vältida vigade tekkimist. [15] Juhul kui kutsuda TypeScript'i kasutamisel välja funktsioon vale parameetriga, siis ilmneb veateade ja koodi kirjutamisel oskab see vihjata isendiparameetritele [16].

Rakenduste kasvav keerukus on kliendi poole loomisel tavalise JavaScripti'i kasutamise vähem mõistlikuks muutnud. Keeruliste ja mahukate koodide paremaks haldamiseks on loodud erinevad raamistikud, mis hõlbustavad protsessi. Raamistikud toetavad ja muudavad lihtsamaks JavaScript'i kirjutamise ning koodi haldamise. [13] Järgnevalt on

kirjeldatud nelja JavaScript'i raamistikku. Viimaste eeliste ning puuduste kaardistamine võimaldab hinnata, milline raamistik osutub töö kontekstist lähtuvalt sobivaimaks.

React on JavaScript'i raamistik, mis on arendatud ja loodud Facebook'i poolt. Raamistikul on komponentide põhine struktuur, mis võimaldab rakenduses neid taaskasutada. Muutuste tegemisel uuenevad ainult muutusi sisaldavad komponendid, mistõttu jäävad teised komponendid puutumata. React on aga detailirohke raamistik, mis nõuab põhjalikku tutvumist, et omandada kõik selle nüansid. Tegemist on populaarse ja suurt huvi tekitanud raamistikuga, mistõttu on selle arendus kiire ja uuendusi üsna palju. See tingib olukorra, kus React'i dokumentatsioon on kohati puudulik. Sellest tulenevalt on ka raskendatud detailse ja värskeima informatsiooni leidmine. [14]

Vue.js on võrdlemisi uus ja arenev raamistik kasutajaliidese ehitamiseks, mille kasutajaskond kasvab kiiresti. Rakenduse kasuks räägib selle lihtne omandamine ja kasutamine. Brauseritele on loodud mugavamaks arendustegevuseks spetsiaalne Vue.js tööriist. Võrreldes teiste Javascript'i raamistikega on Vue.js'i maht erakordselt väike. Selle tulemusena on ka raamistiku jõudlus parem. Vue.js'il on täielik ja detailne dokumentatsioon, mis on kaasaegne ja arusaadavalt kirjutatud. Loogilise struktuuri abil on võimalik luua paindlikke komponente, mida saab taaskasutada nii samas projektis kui ka teistes projektides. [14]

Angular on uue loogikaga raamistik, mis on ümber kirjutatud vanematest Angular'i versioonidest. Tegemist on täieliku raamistikuga, mis ei vaja juurde lisa teeke. See välistab arenduse käigus erinevate puuduolevate teekide otsimise. Komponentide põhine arhitektuur võimaldab erinevaid komponente taaskasutada. Raamistik kasutab TypeScript'i, mis muudab koodi kirjutamise lihtsamaks ja vead esinevad väiksema tõenäosusega. Siiski on Angular'iga alustamine esialgu pigem keeruline, kuna selgeks tuleb teha palju detaile. [14]

Käesolevas töös ei ole tegemist väga keerulise kliendipoolse kasutajaliidese vaate ehitamisega. Keerukus peitub pigem serveripoolses äriloogika realiseerimises, mistõttu ei ole vajadust täieliku raamistiku järele. Selle tõttu võib Angular'i raamistiku välistada. Nii Vue.js kui ka React raamistik eeldavad põhjalikumat tutvumist ja läbitöötamist. Vue.js'i kasuks räägib eeskätt selle dokumentatsiooni põhjalikkus ja sellest tulenevalt

omandamise hõlpsus. Ühtlasi on autoril plaanis Vue.js'i oma tulevastes projektides kasutada. Eelnevale tuginedes on kliendi poole arendamiseks valitud Vue.js raamistik.

Programmeerimise hõlbustamiseks on Vue.js'i puhul võimalik kasutada TypeScript'i, mille toetus raamistikul on [17]. Rakenduse disainimiseks lisatakse Vue.js'ile BootstrapVue teek. Teek võimaldab juba eelnevalt kirjeldatud komponente kasutada. Näiteks on võimalik kasutada varem kirjeldatud nuppu, mis on korrektselt ära kujundatud. [18] Kuna rakenduses on vajalik kuvada genereeritud plaan, kus on esitatud ka aluste kujutised, kasutatakse Konva.js teeki. Teek võimaldab joonestada vajaliku skeemi ning soovi korral lisada sellele erinevaid sündmusi. [19]

Serveri ja kliendi poole arendamine tugineb erinevatele raamistikele. Serveri poolel kasutatakse .NET raamistikku ning C# programmeerimiskeelt. Kliendi poole arendamiseks kasutatakse Vue.js raamistikku. Potentsiaalsete vigade vältimiseks kasutatakse täiendavalt TypeScript'i. Valitud lahendused toetavad rakenduse sihipärast toimimist ja võimaldavad tagada eesmärgipärase tulemuse.

3. Aluste hoiustamise rakenduse loomine Haven Kakumäe jahisadama näitel

3.1 Lähteolukorra kirjeldus

3.1.1 Haven Kakumäe jahisadama ellingu kaardistamine

Loodav rakendus peab tagama Haven Kakumäe jahisadama ellingutes aluste paigutamise võimalikult väikese ruumikaoga. Toimiva rakenduse kirjutamine eeldab eelnevalt lähteolukorra kaardistamist koos esinevate nõuete ja piirangutega. Seetõttu antakse järgnevalt ülevaade Haven Kakumäe jahisadamast, mis loob eeldused ülesande lahendamiseks. Kirjeldatud on nii sadama võimalusi, ellingute parameetreid kui ka aluseid puudutavat.

Talvehooajaks on vaja jahisadamas olevad alused pakkida. Seda tehakse Haven Kakumäe jahisadamas nii sise- kui välistingimustes. Viimase korral paigutatakse alused staapelplatsile. Sisetingimustes hoiustamiseks on jahisadamas kaks ellingut, mille põrandapinnale alused paigutatakse. Lisaks asuvad ühes ellingus riiulid, mida kasutatakse väiksemate aluste pakkimiseks. Sinna alla kuuluvad kuni 10 meetri pikkused või kuni 7 tonni kaaluvad mootorpaadid. Probleemseimaks ja töömahukamaks on aluste pakkimine ellingute põrandapinnale, kuna siseruum seab rohkelt piiranguid. (Lisa 1) Seetõttu on autor rakenduse loomisel lähtunud ellingute põrandapinna pakkimise probleemist. Aluste paigutamine riiulitesse ja staapelplatsile on välja jäetud. Küll aga näeb autor siinkohal rakenduse edasiarendamise võimalust, mis tagaks jahisadamas aluste pakkimisel terviklahenduse.

Haven Kakumäe jahisadamas on kaks ellingut. Nende mõõtmed erinevad nii põrandapinna kui ka kõrguse poolest. Kõrgema ellingu põrandapind on 35m x 57m. Madalama ellingu põrandapind on 33,7m x 57m. Ellingutes on põrandapindala kokku 3915,9 ruutmeetrit. Sinna paigutatakse alused, millele on müüdud koht ellingusse. Seni on sadama personal lahendanud paigutamise probleemi paberi peal käsitsi. Esmalt kaardistatakse aluste parameetrid. Järgnevalt tuleb pakkimist alustades arvestada piirangutega, mille seavad ruumid ja aluste liigutamiseks kasutatavad

tõstemehhanismid. Pakkimisprotsessi käigus esineb ka olukordi, kus ellingus on vaja ka kohapeal alasid mõõta. (Lisa 1)

Aluste paigutamisel ellingutesse tuleb arvestada mitmete piirangutega. Eeskätt on need seotud ruumist tulenevate kitsendustega, kuid on ka aluseid paigutavate kraanade võimekusest ja mõõtudest tingitud. Ruumile on seatud nõue, et põrandapinnal peab seinte ääres olema vaba käigukoridor ühe meetri ulatuses. See vähendab kasutatava põrandapinna mahtu, kuid tegemist on määrusega, mida peab järgima. Ühtlasi tuleb pakkimisel arvestada ellingute uste kõrgusega. Kõrgemas ellingus on ukse kõrguseks 12,5m ja madalamas on see 8,5m. Viimaste mõõtudega peab arvestama raamkraana kasutamisel, kuna selle kõrgus seab takistused ellingutesse sisenemisel. (Lisa 1)

Aluste paigutamisel on jahisadamas kasutusel kolm tõstemehhanismi, kuid ellingute põrandapinna pakkimiseks kasutatakse neist kahte. Aluste puhul, mis on raskemad kui 25 tonni, kasutatakse Ascom raamkraanat. Kergemate aluste puhul kasutatakse platvormtreilerit Roodberg. Raamkraana kasutamisel tuleb arvestada kõrguse piiranguga ellingusse sisenemisel. Kuid mõlema tõstemehhanismi puhul tuleb arvestada tehniliste piirangutega. Sinna alla kuuluvad tõstukite kabariitidega arvestamine, tõstevõime ja pöörderaadiuste hindamine. Alusete pakkimise planeerimisel tuleb arvesse võtta, et tõstemehhanismidega pole võimalik igas ellingu alas liikuda. Jahisadamas kasutusel oleva suurima raamkraanaga pole võimalik madalamasse ellingusse siseneda, kuna selle lagi on liialt madal. Ühtlasi võtab kõige suurem raamtõstuk aluste paigutamisel aluse mõlemalt poolt umbes meetri ulatuses ruumi. (Lisa 1)

Rakenduse loomisel arvestatakse ellingu mõõtmetega. Lisaks on kaardistatud ruumile seatud nõuded ja piirangud, mis aluste pakkimist mõjutavad. Ühtlasi on arvestatud tõstemehhanismide eripäradega, mis seavad pakkimisprotsessile oma tingimused. Sellest lähtuvad on rakendus Haven Kakumäe jahisadama vajadusi ja võimalusi arvestades loodud ning vastab erijuhtudele, mis jahisadamas esineda võivad.

3.1.2 Hoiustatavate aluste kirjeldus

Haven Kakumäe jahisadamas tekib talvehooajal vajadus aluseid hoiustada. Selleks paigutatakse need kas staapelplatsile või ühte kahest ellingust. Kaikohti on sadamas 300 tükki [20]. Siiski ei kuulu alused samas mahus hoiustamisele. Sellest hoolimata on

eesmärgiks paigutada jahisadama alale võimalikult palju aluseid. Eriti oluline on optimeerida aluste paigutamine ellingute puhul, kuna selle pindala on piiratud. Kuna alusekohti müüakse ellingutesse, tuleneb siit ka toimiva lahenduse väärtus jahisadamale. Rohkemate aluste paigutamise võimalus tähendab ka jahisadamale suuremat tulu. (Lisa 1)

Probleemi muudab komplekssemaks asjaolu, et aluste mõõdud varieeruvad suuresti. Pakitavad paadid ja jahid on erineva suuruse ja parameetritega. Haven Kakumäe jahisadamas hoiustatavate aluste pikkused algavad 5 meetrist ja lõppevad 32 meetriga. (Lisa 1) Aluste kõrgust nende pakkimisel arvesse ei võeta. Seda põhjusel, et aluseid ei pakita koos püstises asendis mastidega. Lisaks on ellingute laed piisavalt kõrged, et mitte takistuseks osutuda.

Rakenduse lisaväärtus, täiendavalt aluste optimaalsele pakkimisele, on nende liigutamine vastavalt soovitud ajale. See kehtib nii aluste pakkimisel sobiva kuupäeva arvestamisega kui ka nende väljavõtmise planeerimisel. Taoline võimalus eeldab, et sadama personalil on teave aluse järgmise navigatsioonihooaja kohta. Pakkimisel valmistab probleeme olukord, kus aluste hooajad lõppevad erinevatel aegadel. Sellest tulenevalt ei pruugi olla alused jahisadamas samal ajal tõsteks valmis. Lisaks tuleb arvestada ka olukorraga, kus talveperioodil on alusel vaja teostada hooldus- või muid töid. See tähendab, et tööde teostamiseks peab tagama rohkem ruumi mõnes aluse piirkonnas. (Lisa 1)

Ka aluste ellingutest väljavõtmisel võib esineda juhte, kus tuleb arvestada ajalise teguriga. Valdavalt on veeskamine planeeritud aprilli teise poolde, kuid selles võib esineda anomaaliaid. Kui aluse omanik soovib hooaega varem alustada, on võimalik aluste pakkimisel sellega arvestada. Kliendile võimaldatakse aluse kättesaamine talle sobival ajal ning tagatakse sadama personalile võimalikult vähene täiendav lisakoormus. (Lisa 1)

Rakendus peab ühest küljest tagama, et alused sobituksid hoolimata oma mõõtmete erisustest. Teisalt on oluline arvestada ka ajalise faktoriga. Alused tuleb pakkida võimalikult kompaktselt, kuid arvestades olukorraga, kus mõnda neist on vaja erandkorras liigutada.

3.2 Lahenduse kirjeldus

3.2.1 Funktsionaalsed nõuded

Rakenduse loomiseks on vaja kaardistada nõudmised, mille täitmisele loodav rakendus peab vastama. Oluline on määrata kliendi vajadustest lähtuvalt tegevused, mida rakenduses peab olema võimalik teha. Lisaks peab funktsionaalsete nõuete kirjeldamisel lähtuma asjaolust, et rakendus oleks eesmärgipärane ning kataks selle saavutamiseks vajalikud punktid.

Funktsionaalsed nõuded kirjeldavad kriteeriume, mida süsteem peab tegema ja kuidas käituma [21, pp. 17 - 18]. Käesoleva rakenduse põhiline funktsionaalne nõue on aluste pakkimine. Seda toetab ellingute vaate kuvamise funktsionaalsus. Pakitud aluste plaan peab olema kasutajale nähtav. (Lisa 2) Samuti peab kasutajal olema võimalus näha tagasiulatuvalt varem moodustatud plaane.

Pakitud aluste plaan genereeritakse nimekirjas olevate aluste andmete põhjal. Käesoleva rakenduse puhul on vajalik aluste pakkimise jaoks need esmalt rakendusse lisada. Nimekirja kuvamine võimaldab kasutajal saada ülevaate rakenduses olevatest alustest. Lisaks on see indikaator ka seisust jahisadamas. Nimekirjas olevaid aluseid peab olema ka võimalik kustutada. Seda tehakse juhul kui alust enam talvel sadamas ei hoiustata. Seega on oluline, et kustutatud alust uute plaanide moodustamisel ei kaasataks.

Lisamise funktsionaalsus on alustele täiendavalt vajalik ka ellingute puhul. Vajadusel peab olema võimalik rakenduses lisada juurde pakitavaid ruume ja neid soovi korral kustutada. Ellingute hulka peab olema võimalik vastavalt võimaluste kujunemisele korrigeerida. Seejuures peab rakendus arvestama võimalusega, kus pakitavat pinda tekib juurde või jääb vastupidiselt vähemaks.

3.2.2 Mittefunktsionaalsed nõuded

Mittefunktsionaalsed nõuded kirjeldavad ära piirangud, kuidas funktsionaalseid nõudeid täidetakse [21, pp. 17 - 18]. Aluste nimekirja kuvamisel on oluline kajastada kindlat teavet. Sinna alla kuuluvad aluse nimi, pikkus, laius, kaal, omaniku nimi ja kontaktisiku andmed. Lisaks peab olema määratud aluse staatus, mis näitab aluse hoiustamislepingu

seisu. (Lisa 2) Viimane võimaldab sadama personalil hinnata ellingute täituvust ja planeerida sellele tuginedes oma tegevust.

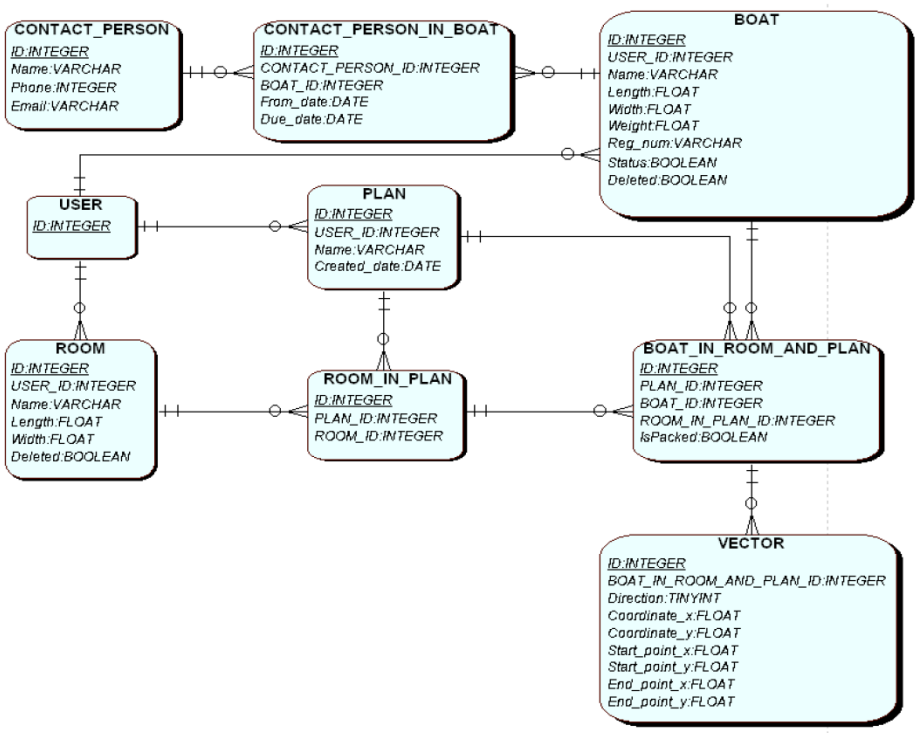
Aluste pakkimisel peab rakendus arvestama ohutusest tuleva nõudega, et seinte äärde peab jääma vaba ruumi üks meeter (Lisa 1). Sellele tuginedes peab ellingute mõõtudest lähtudes arvestama, et pakitavat ala hinnates oleks nõutavas mahus seinte ääred vabad. Lisaks peab rakendus arvestama piiranguga, et madalamasse ellingusse ei mahu sadama raamtõstukiga.

Rakenduse toel peab moodustuma võimalikult optimaalne pakkimisplaan. See tähendab, et kasutamata põrandapinda peab jääma võimalikult vähe. Plaani kuvamisel peavad alused olema plaani peal eristatavad ja nimetatud. See võimaldab reaalsel pakkimisel plaanist paremini lähtuda. Lisaks peab soovi korral olema võimalik näha aluse pikkust, laiust ja kaalu. (Lisa 2) Täiendavalt peab rakendus arvestama võimalusega, et mõnda alust on teistest varem vaja liigutada. Sellekohane märge on vaja eelnevalt lisada.

3.3 Rakenduse kirjeldus

3.3.1 Andmebaasi kirjeldus

Andmebaas on kirjeldatud olemi-suhte diagrammina. See koosneb kaheksast põhitabelist, millele lisaks kasutatakse süsteemisiseseid kasutaja identiteedi tabeleid. Igal tabeli kirjel on unikaalne ID-number, mis genereeritakse automaatselt. Joonisel 5 on visualiseeritud andmebaasi skeem.



Joonis 5. Rakenduse andmebaasi olemi-suhte diagramm.

Kõik alused, mis rakendusse sisestatakse, lisatakse BOAT tabelisse. Uue aluse lisamisel on kohustuslik sisestada aluse kontaktisiku nimi ja number. E-maili lisamine pole kohustuslik. Tarvilik on sisestada aluse nimi, pikkus, laius, kaal ja registreerimisnumber. Lisaks on võimalik alustele määrata staatus, et kas aluse kohta on kinnitatud allkirjastatud leping. Juhul kui alus kustutatakse, siis määratakse aluse kustutatud väärtus tõeseks. Kirjet ennast ei saa ära kustutada, kuna varasemalt moodustatud plaanidel on aluse andmed vajalikud. Sisestatud kontaktisiku andmete alusel luuakse uus kirje CONTACT_PERSON tabelisse ja seotakse selle kirje ID ja uue sisestatava aluse ID tabelisse CONTACT_PERSON_IN_BOAT, kuhu lisatakse ka käesolev kuupäev. Juhul kui aluse kontaktisik peaks mingil põhjusel muutuma, siis lisatakse sinna lõppkuupäev. Seda küll rakendus hetkel ei võimalda, aga skeem võimaldab.

Kõik ellingud, mis kasutaja lisab rakendusse, lisatakse tabelisse ROOM. Kohustuslik on sisestada ellingu nimetus, pikkus ja laius. Haven Kakumäe sadamas on hetkel küll kaks ellingut, aga andmebaasi skeem võimaldab vajadusel sinna ellinguid juurde lisada.

Kui kasutaja avaldab soovi uue aluse pakkimisskeemi loomiseks, luuakse PLAN tabelisse uus kirje, kuhu lisatakse jooksev kuupäev ja kasutaja lisab plaani nimetuse. Lisaks luuakse kasutaja iga ellingu kohta kirje tabelisse ROOM_IN_PLAN. Sinna

lisatakse loodud PLAN tabeli kirje ID ja ellingu ID. Järgnevalt lisatakse iga pakitava aluse kohta kirje BOAT_IN_ROOM_AND_PLAN tabelisse, kuhu lisatakse pakitava aluse ID number ja eelnevalt loodud ROOM_IN_PLAN tabeli kirje ID. Ühtlasi lisatakse tõene väärtus, kui alus sai pakitud. Kui alus mingil põhjusel ei mahtunud ellingusse, määratakse väärtuseks väär.

VECTOR tabelisse luuakse eelnevalt kirjeldatud tabelikirje parameetrite järgi kirjed, mille parameetrid täidab äriloogika.

3.3.2 Rakenduse serveripoolne kirjeldus

Rakendus on kirjutatud hajusa süsteemina kasutades klient-server-mudelit. Klient saadab oma soovi teatud ressursi kasutamiseks serverile. Viimane teostab nõutud tegevuse, kui see osutub vastuvõetavaks. Serveri pool on realiseeritud Microsoft'i .NET raamistikus, kasutades programmeerimiskeelt C#.

Serveri poolel on rakenduses loodud kindel arhitektuur. See on teostatud erinevaid disainimustreid kasutades. Koodis esinevad eristuvad kihid, mis omavahel suhtlevad. Taolise arhitektuuri abil saab rakenduses oleva andmetega seotud loogika hoida eraldi, sealhulgas ka rakenduse äriloogika. Lisaks saab kliendi poolega suhtlevad kontrollerid hoida äriloogikast puhtad. Kihilise struktuuri toel on võimalik muudatusi ellu viia vähesema vaevaga. Lisaks on koodi haldamine lihtsam ja selle taaskasutamine tulevastes projektides samuti hõlpsam.

Rakenduse serveripoolne struktuur on moodustatud mitmest kihist. Vaadeldes struktuuri andmebaasi poolt kliendi poole liikudes, on esimeseks kihiks DAL (*Data Access Layer*). Selles on kirjas kõik, mis on seotud andmebaasiga suhtlusel. Kihis on realiseeritud *Repository*, UOW (*Unit Of Work*) ja *Factory* disainimustrid. *Repository* ehk repositooriumite muster võimaldab viia andmebaasist sõltuvuse minimaalseks. See tähendab, et andmebaasi väljavahetamine on hõlpsam ja koodi ei ole rangelt sisse kirjutatud sõltuvus andmebaasist. Kogu andmebaasiga suhtlus toimub ühes kohas ja muust koodist eraldi. Iga andmebaasi tabeli kohta on loodud repositooriumi klass, kus on meetodid andmete salvestamiseks, muutmiseks, pärimiseks ja kustutamiseks antud tabelist. [22]

Kuna repositooriumeid on mitmeid, tekib probleem andmete muutmisel. See esineb juhul, kui on vajalik andmeid mitmest tabelist muuta. Iga repositoorium kasutab erinevat andmebaasi konteksti isendit, mistõttu üks muutus võib õnnestuda, aga teine mitte. Sellise olukorra vältimiseks on loodud UOW klass. UOW abil kasutavad kõik repositooriumid ühte ja sama andmebaasi isendit. [22] Selleks, et UOW poole pöördudes oskaks UOW küsitavat repositooriumit tagastada, on kasutatud *Factory* mustrit. Antud muster loob esimesel küsimisel uue repositooriumi isendi ja järgneval küsimusel tagastab juba sama isendi.

DAL-ile järgnev kiht on BLL (*Business Logic Layer*). Sinna on kirjutatud kogu rakenduse äriloogika. Kihis on kasutatud disainimustreid sarnaselt eelmisele kihile. Äriloogika on kirjutatud eraldi klassi ja klassimeetodeid kutsutakse välja *service*'ites ehk teenustes. Teenused suhtlevad andmebaasiga läbi UOW. Kuna teenuseid on samamoodi mitmeid nagu eelmises kihis oli repositooriumeid, tekib siinkohal sama probleem, et erinevad teenused kasutavad erinevat UOW isendit. Selle vältimiseks on loodud BLL klass, mis on oma olemuselt sama nagu UOW. Erinevate teenuste poole saab nüüd pöörduda läbi BLL'i, mis teenuse tagastamisel kasutab Factory mustrit. Sellega kindlustatakse, et iga teenus kasutab sama UOW isendit.

Kliendipoolseimas kihis asuvad kontrollerid. Kontrollerid on vahelüliks BLL kihi ja kliendi vahel. Klient pöördub serveri poole kontrolleri kaudu ning viimane omakorda BLL-i poole. Siiski saavad kontrolleri poole pöörduda ainult autoriseeritud kliendid. See tähendab, et iga pöördumisega tuleb päringule kaasa anda JWT (*JSON Web Token*). JWT on räsi, mis sisaldab kliendi infot ja salajast võtit. [23] Kliendile edastatakse JWT rakendusse sisse logimisel, mis tuleb iga päringuga kaasa anda, mille alusel saab server audentida.

Igas kihis on kasutusel oma DTO (*Data Transfer Object*) ehk andmeedastusobjekt, mis hoiab andmeid. Andmebaasi päringule vastavad andmed kopeeritakse ümber kohe pärast nende väljumist andmebaasist. DAL kihis hoitakse kopeeritud andmeid DAL-DTO-s. Nii pea kui andmed liiguvad BLL kihti, kopeeritakse need BLL-DTO-ks. Kontrollerites kopeeritakse andmed veel omakorda ümber andmeteks, mis saadetakse kliendi poole teele.

Andmete ümberkopeerimisega on võimalik vältida juhtu, kus andmete järgi saaks teha järeldusi andmebaasi skeemi kohta. Ühtlasi on tihti vaja andmete päring teostada mitme tabeli lõikes ja erinevates kihtides need omavahel siduda. DTO abil saab seotud andmed ühte objekti kokku panna ja nii on neid mugavam saata kliendi poolele. Lisaks saab sellega vältida mittevajatavate parameetrite saatmist ja sellega vähendada ka saadetavate andmete mahtu. Lisaks kaovad andmete kopeerimisel ära viited, kus üks objekt viitab teisele ja viimane taas esimesele. Selline viitamine tekitab koodis lõpmatut viitamist ühelt objektilt teisele. [24]

Pakkimise loogika on kirjutatud eraldi klassi, milles on staatilised meetodid. Põhilised meetodid on välja toodud Lisas 4. Meetodeid kutsutakse välja BLL kihis. Kliendi poole pöördumisel serverisse, sooviga moodustada uus pakkimise plaan, päritakse kõigepealt andmebaasist kasutaja kõik alused ja ellingud. Alused sorteeritakse pärimisel kaalu järgi kahanevas järjekorras ja siis pikkuse järgi kahanevas järjekorras.

Päritud andmetega pöördutakse pakkimise meetodi poole. Meetodis luuakse aluse pikkuse, laiuse ja 60 kraadise vöörinurga järgi vastupäeva suunatud vektorid, alustades aluse ahtri parempoolsest nurgast. Vektoritele määratakse koordinaadid. Joonisel 3 nähtav skeem kujutab vektorite toel mittelõikuvuse piirkonna moodustamist. Piirkonna loomisel võetakse pakitud aluse vektorid, pakitava aluse vastassuunalised vektorid ja kogutakse need ühte punkti kokku, suunaga sellest punktist eemale. Mittelõikuvuse piirkonna moodustamisel alustatakse pakitud aluse vööri vasakpoolsest vektorist, mille järel liigutakse vastupäeva ümber punkti, kuhu on vektorid kogutud. Seejärel paigutatakse nad üksteise otsa, andes neile algus- ja lõppkoordinaadid.

Ellingut on käsitletud pealtvaates koordinaatteljestikuna. Ellingu tagasein moodustab x-telje ja küljesein y-telje. Järgneva sammuna proovitakse pakitavat alust paigutada vastavalt ellingu parameetritele võimalikult madalale joondudes vasakule. Juhul kui valitud punkt ei asu üheski mittelõikuvuse piirkonnas, siis on aluse asukoht ellingus leitud. Punkti asumist või mitteasumist kontrollitakse vektori abil. Punktist luuakse vektor, mis on suunaga piki x-telge lõpmatuse poole. Joonisel 4 on visualiseeritud võtet, kuidas leida punkti asumine või mitte asumine mittelõikuvuse piirkonna sees.

Matemaatiliselt luuakse lõpmatuse poole suunatud punktist ühe punkti järgi sirge võrrand. Seejärel leitakse mittelõikuvuse piirkonna vektori alguspunktiga võrrandi

tulemus ja vektori lõpppunktiga võrrandi tulemus. Juhul kui nii alguspunkti tulemus, kui ka lõpppunkti tulemused on samamärgilised, on mittelõikumine tuvastatud. Kui tulemused on erimärgilised, siis need lõikuvad. Kui kummagi punkti tulemus on null, siis asub punkt sirge peal. Kuna moodustatud sirge on lõpmata pikk, võib punkt asuda sirgel sellises osas, mis pole oluline. Tuvastamaks, kas lõikumine leiab aset sirge olulises osas, luuakse vastupidi vektori punktidest sirge võrrand. Seejärel leitakse punkti alguspunktiga võrrandi tulemus ja järgmisena ka lõpppunktiga, mis on suurem kui vektori suurim x-väärtus. Kui tulemused on samamärgilised, siis järelikult nad ei lõiku. Kui tulemused on erimärgilised, siis vektorid lõikuvad. Siiski võib esineda eriolukord, kus lõikumine toimub kummagi vektori algus- või lõpppunktis.

Sellisel moel kontrollitakse kõiki mittelõikuvuse piirkonna vektoreid. Kõik lõikuvused loendatakse kokku. Kui lõikuvuste arv on paaris, on punkt mittelõikuvuse piirkondadest väljaspool ja kui arv on paaritu, asub see seespool. Vastavalt sellele saab teada, kas asukoht on valiidne või mitte.

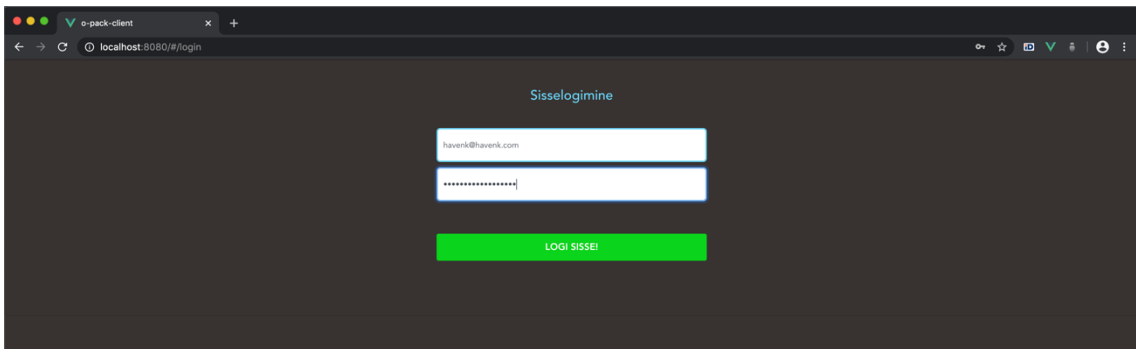
Seejärel määratakse eelnevalt loodud aluse vektoritele algus- ja lõppkoordinaadid. Kui kõik alused on pakitud, siis on pakkimise plaan valmis. Juhul kui alus ei peaks mingil põhjusel ellingusse mahtuma, jäetakse see alus meelde. Pakkimata alused kogutakse kokku ja edastatakse kliendile.

3.3.3 Rakenduse kliendipoolne kirjeldus

Rakenduse külge on loodud kasutajaliides. Selle toel on kliendil võimalik rakendust kasutada. Rakenduse kliendipoolne osa on realiseeritud Vue.js raamistikuga, millele on lisatud vajalikud teegid. See on tarvilik komponentide disainimise lihtsustamiseks ja aluste pakkimisskeemi ülesehitamiseks. Lisaks on koodi kirjutamisvigade vältimiseks kasutusel TypeScript.

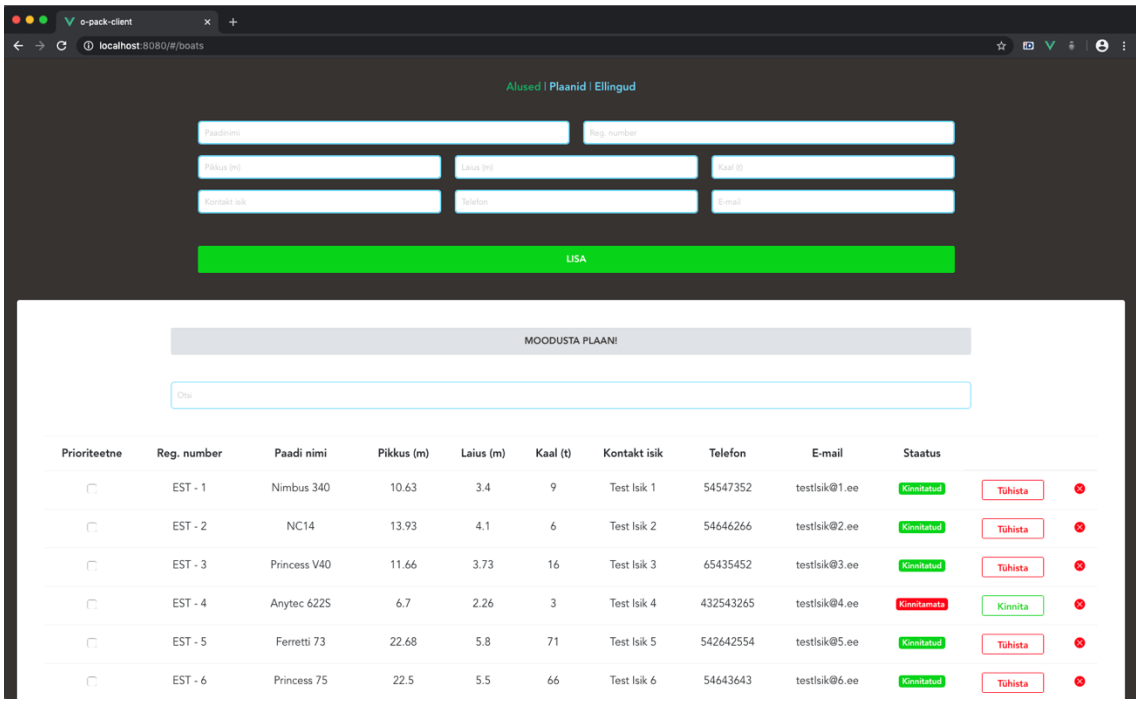
Koodi struktuur on oma ülesehituselt võrdlemisi lihtne ja loogiline. Teenustes on kirjeldatud serveriga suhtlemise loogika. Andmete pärimisel või saatmisel pöördutakse teenuse poole, mis omakorda lisab päringule kaasa sisselogimisel saadud JWT ja pöördub serveri kontrolleri poole. Selliste teenuste kasutamise abil hoitakse kogu serveri poole pöördumise loogika ühes kohas ja loogika ei vaju kogu projekti peale laiali.

Kasutajaliides koosneb mitmest vaatest, mis on kokku ehitatud erinevatest komponentidest. Komponentide ehitamisel on kasutatud BootstrapVue komponente ja CSS klasse. Erinevate vaadete vahel liikumiseks on üleval navigeerimisriba. Joonisel 6 on näha rakenduse esimene vaade, milleks on sisselogimise vaade. Kasutajal tuleb sisestada kasutajanimi ja salasõna, mis saadetakse serverile. Õnnestunud sisselogimise puhul tagastab server JWT, mis salvestatakse kliendi poolel ära, et järgnevatel pöördumistel serveri poole see kaasa anda. Kasutaja suunatakse edasi aluste vaatesse.



Joonis 6. Rakenduse sisselogimise vaade.

Joonisel 7 on kujutatud aluste sisestamisvorm, milles saab uusi aluseid rakendusse lisada. Aluse lisamisel tuleb sisestada aluse nimi, registreerimisnumber, pikkus, laius, kontaktisiku nimi ja telefoninumber. Kontaktisiku e-maili sisestamine on vabatahtlik. Kasutaja sisendid valideeritakse, et välistada olukorda, kus kasutaja sisestab telefoninumbri väljale mingi sõne, mis serverisse jõudes tekitab vigu.



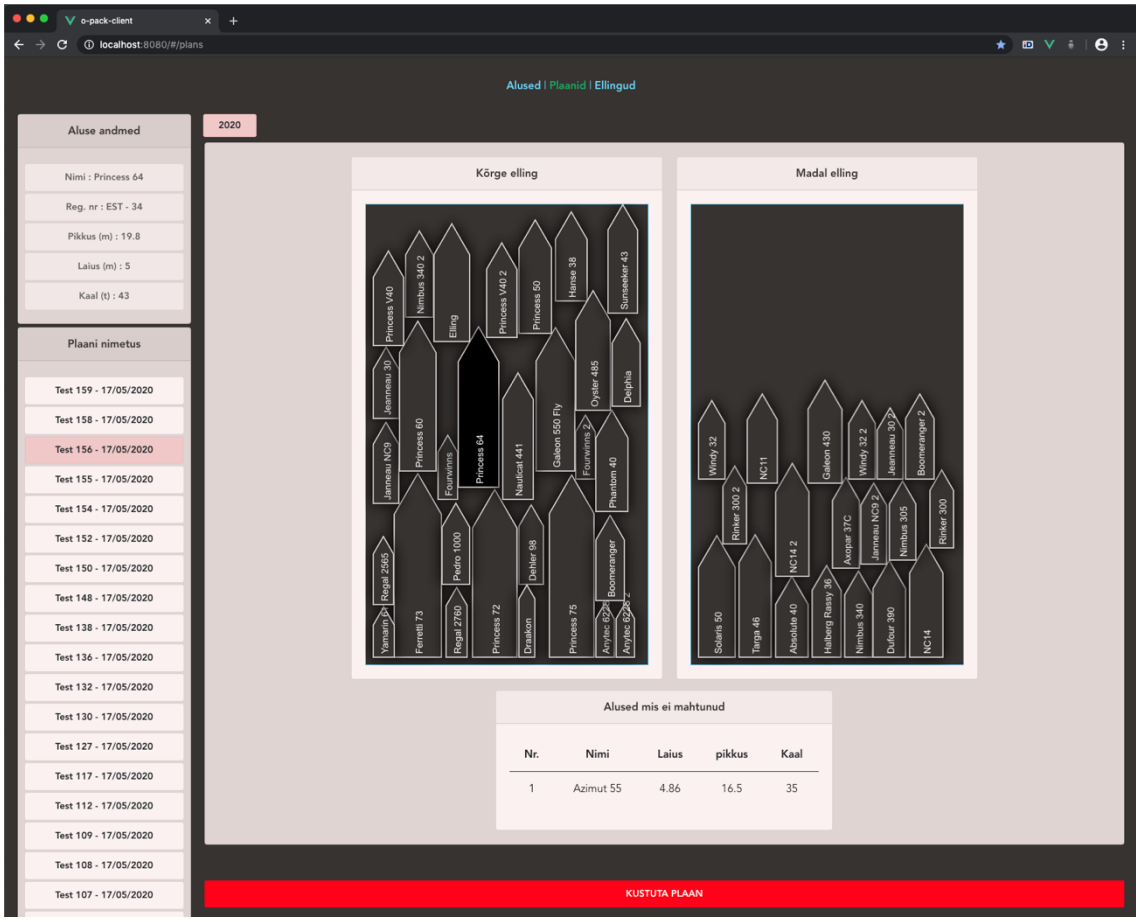
Joonis 7. Rakenduse aluste vaade.

Sisestamisvormile lisaks on aluste vaates kõikide aluste nimekiri, kus on kuvatud aluse ja kontaktisiku andmed. Aluste nimekirjast on võimalik teostada otsingut nime, registreerimisnumbri ja kontaktisiku andmete järgi. Vaates on võimalik muuta aluse staatust vastavalt sellele, kas aluse talvitumisleping on kinnitatud või mitte. Aluseid on võimalik ka nimekirjast kustutada, kui need enam sadamas ei talvitu. Varem moodustatud plaanidele jäävad need alused siiski alles. Arvestamaks pakkimisel juhuga, kus mõnda alust on vaja varem liigutada, tuleb need alused nimekirjas linnutada.

Uue pakkimisplaani genereerimiseks tuleb vajutada aluste vaates olevat vastavat nuppu. Nuppu vajutades kuvatakse kasutajale sisestamisvorm, kuhu tuleb lisada uue plaani nimetus. Joonisel 8 on plaanide vaade, kuhu kasutaja edasi suunatakse. Seal kuvatakse välja viimane moodustatud plaan. Kuvataval plaanil on näha kasutaja sisestatud ruumid ja nendesse pakitud aluseid. Iga aluse peale on kuvatud aluse nimi. Kui arvutihiirega liikuda aluse peale, muutub alus aktiivseks ja vasakule üles serve kuvatakse laeva parameetrid.

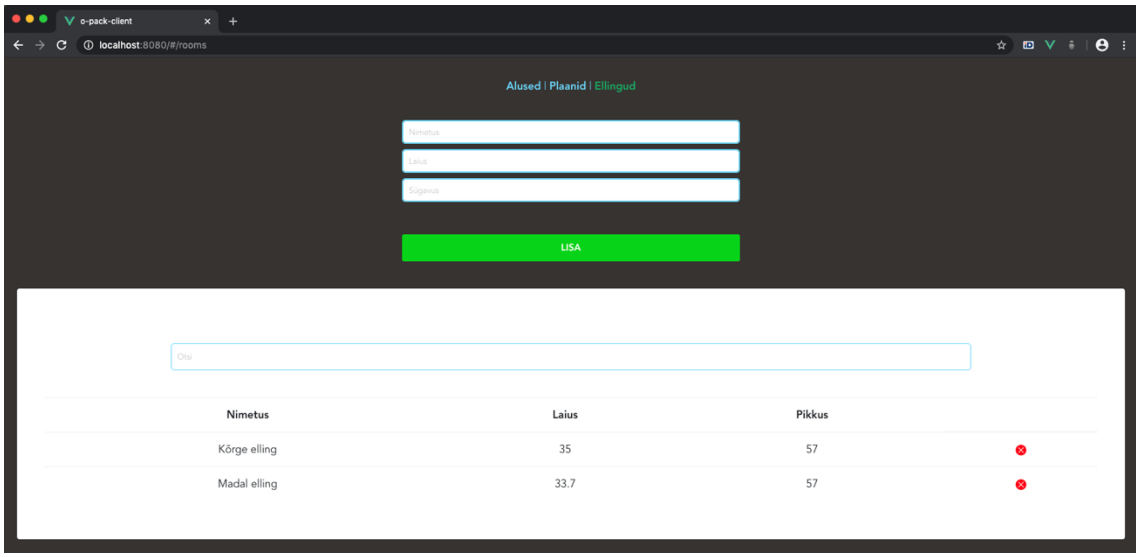
Plaanide vaates on kasutajal võimalik vaadata ka varem moodustatud plaane. Plaanide kohal on aastaarvude valik, mille alusel kuvatakse vaates vasakule serva valitud aasta jooksul moodustatud plaanid. Nende peale vajutades kuvatakse valitud plaan. Juhul kui plaani moodustamisel ei mahtunud kõik alused ellingutesse ära, kuvatakse need kasutajale plaani all. Soovi korral plaan kustutada, saab selleks kasutada

kustutamisnuppu vaate allosas. Sellele vajutades küsitakse eelnevalt ka kasutaja kinnitust.



Joonis 8. Rakenduse plaanide vaade.

Joonisel 9 olevas ellingute vaates on näha kasutaja sisestatud ellingute nimekiri. Vajadusel on võimalik luua uusi ellinguid, sisestades ellingu pikkuse, laiuse ning nimetuse. Nimekirjas on võimalik sooritada otsingut, kui pakitavate ruumide nimekiri peaks liialt pikaks kujunema. Ellingu rakendusest eemaldamiseks on olemas igal real ellingu järel kustutamisnupp.

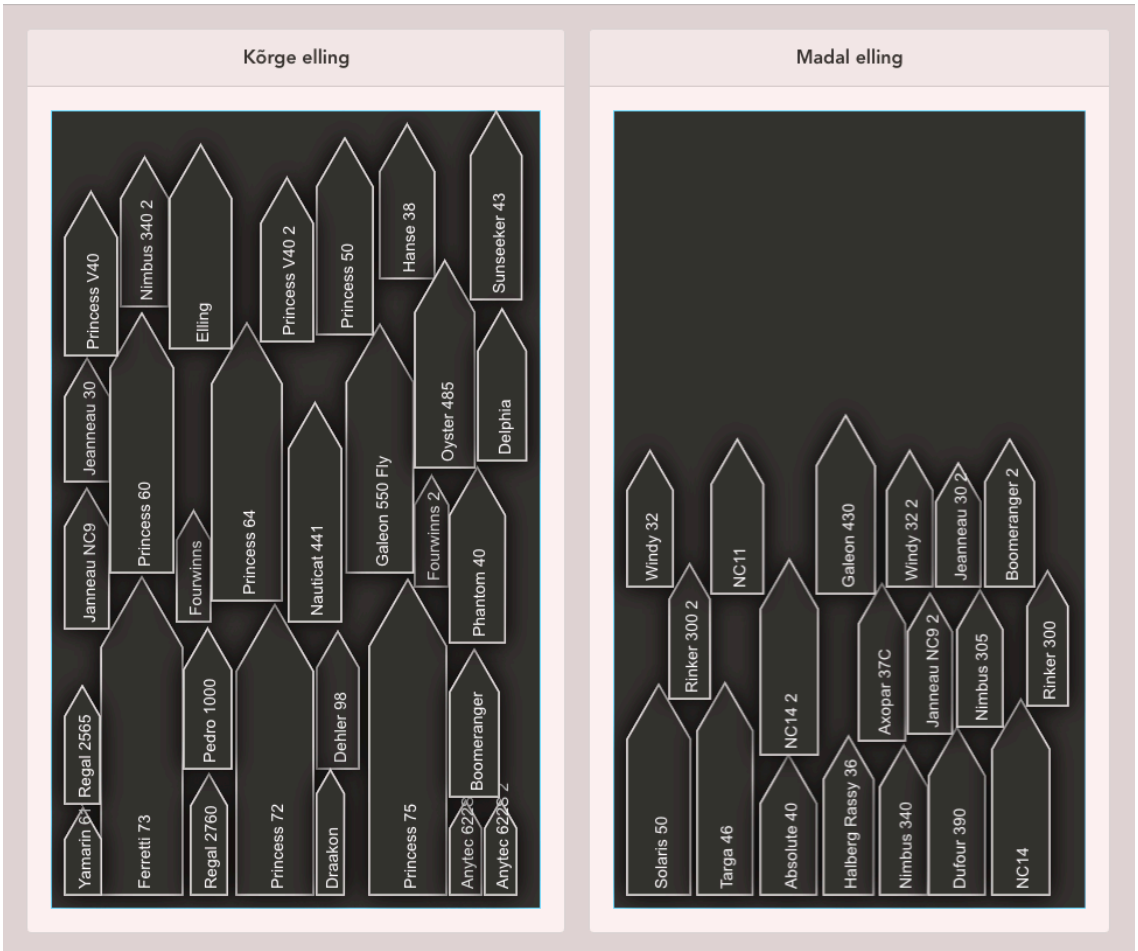


Joonis 9. Rakendus ellingute vaade.

3.3.4 Rakenduse testimine

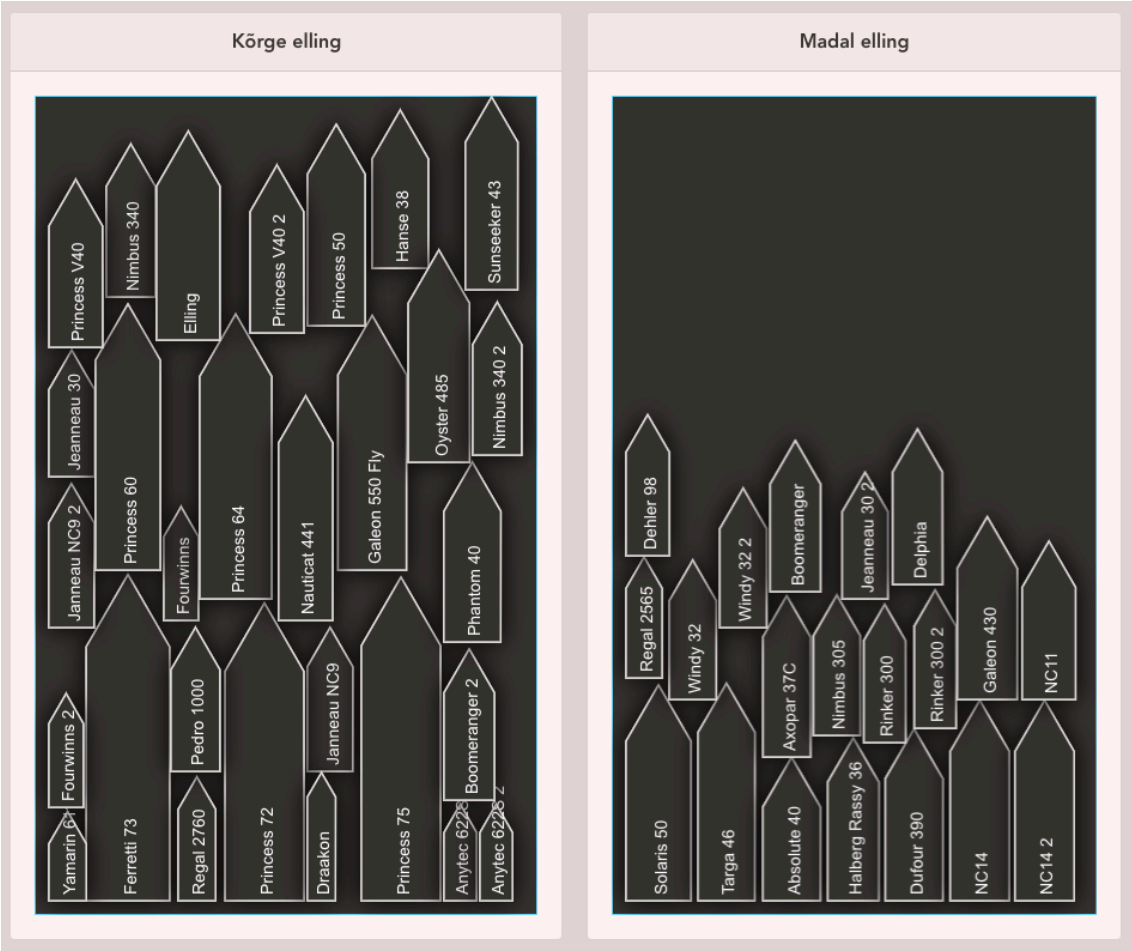
Rakenduse arendamisel on kasutatud sadama poolt edastatud andmeid. Nende abil on toimunud rakenduse järjepidev testimine. Andmed, millega rakendust testitakse, on toodud välja Lisas 3. Andmete mahu suurendamiseks on osade aluste andmeid duubeldatud. Rakenduse põhifunktsionaalsuste hindamiseks on viidud läbi kolm konkreetset testi. Nende abil on testitud etteantud aluste pakkimist, prioriteetsete alustega arvestamist ja suurte aluste paigutamist.

Rakenduse testimiseks on aluste nimekirjas 41 erinevate mõõtmetega alust. Sama nimekirja toel teostatakse kõik testid. Esimese testi eesmärk on pakkida kõik alused optimaalselt ellingutesse. Kõik nimekirja kantud alused peavad olema ellingutesse paigutatud võimalikult vähese ruumikaoga. Aluste pakkimine toimub suuruse alusel. Rakendus peab aga tagama, et suuremate aluste vahele jääv pind saaks samuti täidetud. Joonisel 10 on näha, et nimekirjas olnud 41. alusest on kõik pakitud. Lisaks on näha, et alused on paigutatud võimalikult tihedalt ning täidetud suuremad alad pikemate aluste vahel.



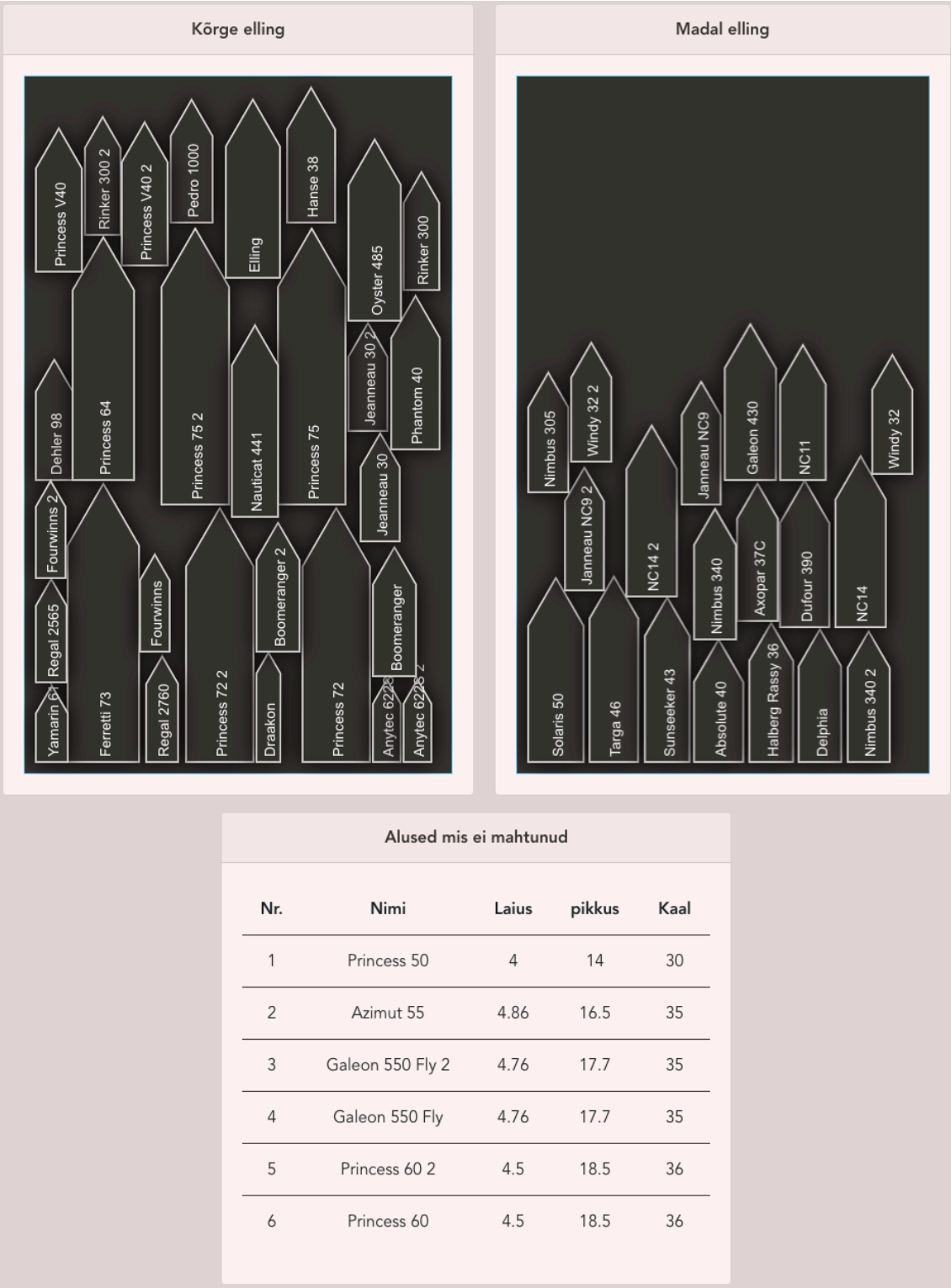
Joonis 10. Test 1, pakitud alused.

Teiseks on testitud rakendust prioriteetsete alustega arvestamise osas. Testitakse juhtu, kus neli alust on vaja ellingust varem välja tõsta. Selleks on enne pakkimisplaani moodustamist märgitud aluste nimekirjas linnukesed prioriteetsete aluste ette. Genereeritud plaanis peaksid valitud neli alust olema kõige ülemises kihis, mis võimaldaks neid varasemalt liigutada. Testi läbi viimiseks on valitud alused Boomeranger, Regal 2565, Delphia ja Dehler 98. Joonisel 11 on näha, et valitud alused on paigutatud kõige ülemisse kihti ja vajadusel on saab neid eelisjärjekorras liigutada.



Joonis 11. Test 2, pakitud alused, valitud alused ülemises kihis.

Kolmandana on testitud suurte aluste pakkimist. Tegemist on erijuhuga. Nimelt ei ole võimalik üle 25 tonni kaaluvaid aluseid paigutada madalamasse ellingusse. Raamtõstuk, millega aluseid liigutatakse, on sinna sisenemiseks liiga kõrge. Seega peab rakendus tagama, et üle 25 tonni kaaluvaid aluseid ei paigutataks madalamasse ellingusse, kui kõrgemas enam ruumi pole. Esialgses nimekirjas on suuri aluseid seitse tükki: Ferretti 73, Galeon 550 Fly, Princess 50, Princess 60, Princess 64, Princess 72, ja Princess 75. Neile on lisatud täiendavalt viis üle 25 tonni kaaluvat alust, et pakkimist testida. Juurde lisatud alused on: Azimut 55, Galeon 550 Fly 2, Princess 60 2, Princess 72 2 ja Princess 75 2. Tulemusena ei tohiks madalamas ellingus olla ühtegi suurt alust. Lisaks peab vaates olema kuvatud aluste nimekiri, mida pakkida ei olnud võimalik. Joonisel 12 on näha, et madalamas ellingus ei asu ühtegi suurt alust ning nimekiri pakkimata alustest asub ellingute all.



Alused mis ei mahtunud

Nr.	Nimi	Laius	pikkus	Kaal
1	Princess 50	4	14	30
2	Azimut 55	4.86	16.5	35
3	Galeon 550 Fly 2	4.76	17.7	35
4	Galeon 550 Fly	4.76	17.7	35
5	Princess 60 2	4.5	18.5	36
6	Princess 60	4.5	18.5	36

Joonis 12. Test 3, suured alused mis ei mahtunud.

Kolme testi käigus tõestati rakenduse põhifunktsionaalsuste vastavus püstitatud ülesannetele. Rakendus teostab aluste pakkimise nimekirja alusel optimaalselt. Alused paigutatakse ellingutesse võimalikult ruumisäästlikult. See tähendab, et esialgselt suuremate aluste vahele jääv pind täidetakse väiksemate alustega. Samuti on tagatud prioriteetse staatusega aluste paigutamine ellingu kõige ülemisse kihti, et need vajadusel

teistest varem ellingust välja liigutada. Ühtlasi arvestab rakendus erijuhuga, kus üle 25 tonni kaaluvaid aluseid pole võimalik madalamasse ellingusse paigutada. Kasutajale edastatakse ka nimekiri alustest, mida pole võimalik ellingutesse paigutada.

Rakendus täidab tööle püstitatud eesmärgi. Alused pakitakse nõuetekohaselt. Nimelt on pakkimine teostatud optimaalselt, mis tagab jahisadama ellingutes võimalikult väikese ruumikao. Lisaks on arvestatud konkreetse juhtumiga seotud eripäradega. Rakendus arvestab jahisamas pakkimisel esinevate piirangutega, mis tagab plaani genereerimisel realistliku ja pädeva lahenduse.

Kokkuvõte

Lõputöö probleemikäsitlus põhineb pakkimisprobleemide lahendamisel. Optimaalse pakkimise probleem esineb mitmes sektoris. Seetõttu on analoogset lähenemist võimalik kasutada valdkondade üleselt. Kui valdavalt leiab probleem käsitlust tootmisega seotud ettevõtetes, on käesolevas töös ülesandepüstitusse toodud jahisadam. Kõigil juhtudel on eesmärgiks minimeerida kulu. Jahisadama puhul on ülesandeks tagada aluste pakkimine ellingutesse võimalikult kompaktselt ja vähese ruumikaoga.

Probleemi lahendamiseks on autor andnud ülevaate võimalike kasutatavate pakkimisalgoritmide strateegiatest. Nende hulgast on autor valinud lähenemised, mis tagaksid seatud eesmärgi saavutamiseks parima võimaliku lahenduse. Ülesandepüstitusest lähtuvalt on kasutatud BL algoritmi põhimõtet. Ühtlasi on elementide kattuvuse välistamiseks kasutatud kontrollmehhanismina geomeetrilist tehnikat. Elementidena on seejuures käsitletud kumeraid hulknurki, mis kujult sarnanevad pakitavate alustega.

Vahendite valikul on tarvilik silmas pidada nii serveri kui ka kliendi poolt. Esimese loomiseks on kasutatud .NET raamistikku. Kliendipoolse osa kirjutamisel on otsus langetatud Vue.js'i kasuks. Viimasele on lisatud ka täiendavad teegid ja TypeScript.

Rakendus on loodud Haven Kakumäe jahisadama näitel. Selle tarbeks on kirjeldatud sadamale ja aluste hoiustamisele iseloomulikud parameetrid, millega rakenduse loomisel arvestada. Piiranguid seavad nii ellingute mõõtmed, raamkraana liikumine kui ka aluste parameetrid ja hulk. Rakenduse loomise eeldusena on kaardistatud ka funktsionaalsed ja mittefunktsionaalsed nõuded.

Rakendus on kirjutatud hajusa süsteemina kasutades klient-server-mudelit. Serveri poolel on loodud kihiline struktuur, mis võimaldab koodi lihtsamini hallata. Lisaks aitab selline lähenemine serveris asuva äriloogika muust koodist eraldi hoida. Kliendi poolele on ehitatud kasutajaliides rakenduse kasutamiseks. Selle kaudu on võimalik aluseid hallata ehk neid lisada, vaadata ja kustutada. Ühtlasi võimaldab rakendus pakkimisskeeme moodustada ja genereeritud pakkimisplaane näha.

Rakenduse töökindluse kontrollimiseks on läbi viidud kolm testi. Kuigi arendustegevuse vältel on rakendust järjepidevalt kontrollitud, peab autor oluliseks välja

tuua kolme põhifunktsionaalsust puudutavad testid. Esimene neist kinnitas, et nimekirjas olevad alused pakitakse vaba ruumi korral kõik ellingutesse ning tehakse seda optimaalselt. Teine juht tagas prioriteetsete aluste paigutamise kõige eesmisele positsioonile. See võimaldab need kõige varem ellingust välja liigutada. Kolmas test kinnitas, et rakendus arvestab suurte aluste pakkimisel neile kehtivate piirangutega. Lisaks teavitab rakendus kasutajat alustest, mida polnud võimalik pakkida.

Lõputöös on leitud vastused püstitatud ülesannetele. Nende täitmine on aluseks eesmärgipärasele lahendusele. Loodud rakendus tagab aluste optimaalse pakkimise Haven Kakumäe jahisadama ellingutesse. Toimiv rakendus vastab seatud tingimustele ja piirangutele, mis jahisadamas esinevad. Täiendavalt on arvestatud erijuhtudega, mis võimaldavad rakendust erandlikes situatsioonides kasutada.

Käesoleva töö rakendus lahendab ellingute põrandapinna pakkimise probleemi. Siiski kasutatakse Haven Kakumäe jahisadamas ka teisi pakkimislahendusi. Väiksemad alused asetatakse riiulitesse, mis asuvad madalamas ellingus. Täiendavalt on võimalik aluseid pakkida ka staapelplatsile. Siinkohal näeb autor võimalust töö edasiarenduseks, et tagada jahisadamale terviklik lahendus.

Viidatud allikad

[1] T. F. Gonzalez, Handbook of Approximation Algorithms and Metaheuristics, Santa Barbara: Chapman & Hall/CRC, 2018.

[2] S. S. Skiena, The Algorithm Design Manual, New York: Allan m. Wylde, 1998.

[3] R. E. Neapolitan, Foundation of algorithms - Fifth edition, Burlington: Cathy L. Esperti, 2015.

[4] A. Lodi, *Algorithms for Two-Dimensional Bin Packing and Assignment Problems,* Bologna, Firenze, Padova, Siena, 1996 - 1999.

[5] E. Burke, R. Hellier, G. Kendall and G. Whitwell, "Complete and Robust No-Fit Polygon Generation for the Irregular Stock Cutting Problem," *European Journal of Operational Research,* vol. 179, no. 1, pp. 27-49, 2007.

[6] E. López-Camacho, G. Ochoa, H. Terashima-Marín and E. K. Burke, "An effective heuristic for the two-dimensional irregular bin packing problem," *Annals of Operations Research,* vol. 206, pp. 241-264, 2013.

[7] T. Soome, *Hajussüsteemid,* Tartu, 2004.

[8] "Microsoft Docs," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/tour. [Accessed 27 March 2020].

[9] "Microsoft Docs," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework. [Accessed 27 March 2020].

[10] "GeeksforGeeks," [Online]. Available: https://www.geeksforgeeks.org/frontend-vs-backend/. [Accessed 30 March 2020].

[11] "CodeBurst.io," [Online]. Available: https://codeburst.io/8-javascript-alternatives-for-web-developers-to-consider-22f8d38bdfa9. [Accessed 2 April 2020].

[12] "Hackr.io," [Online]. Available: https://hackr.io/blog/best-programming-languages-to-learn-2020-jobs-future. [Accessed 2 April 2020].

[13] "freeCodeCamp," [Online]. Available: https://www.freecodecamp.org/news/do-we-still-need-javascript-frameworks-42576735949b/ . [Accessed 2 April 2020].

[14] "Existek," [Online]. Available: https://existek.com/blog/top-front-end-frameworks-2020/. [Accessed 2 April 2020].

[15] “Alligator.io,” [Online]. Available: https://alligator.io/typescript/typescript-benefits/. [Accessed 2 April 2020].

[16] “Typescript documentation,” [Online]. Available: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html. [Accessed 2 April 2020].

[17] “The Progressive JavaScript Framework,” [Online]. Available: https://vuejs.org/v2/guide/typescript.html?#ad. [Accessed 3 April 2020].

[18] “BootstrapVue,” [Online]. Available: https://bootstrap-vue.org. [Accessed 3 April 2020].

[19] “KonvaJS,” [Online]. Available: https://konvajs.org/docs/. [Accessed 3 April 2020].

[20] “Haven Kakumäe,” [Online]. Available: https://marina.havenk.com/mooring-and-storage/#port-mooring. [Accessed 7 April 2020].

[21] K. Viik, *Mittefunktsionaalsete nõuete määratlemine turvalise tarkvaraarenduse hankimiseks Eesti avalikus sektoris,* Tallinn, 2017.

[22] “Microsoft Docs,” [Online]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application. [Accessed 8 April 2020].

[23] “JWT.io,” [Online]. Available: https://jwt.io/introduction/. [Accessed 10 April 2020].

[24] “Microsoft Docs,” [Online]. Available: https://docs.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5 . [Accessed 10 April 2020].

Lisa 1 - Intervjuu jahisadama kapteniga

1. Kuidas toimub talvel teie sadamas paatide hoiustamine?

Aluste hoiustamine toimub kolmel põhimõttel:

a. Õuetingimustel staapelplatsil kiilplokkidel (kõik alused). Aluste paigaldamine (pakkimine) toimub Ascomi raamkraanaga (alused raskemad kui 25t) või platvormtreileriga Roodberg.
b. Sisetingimustes ellingus spetsiaalsetel raamidel riiulis (kuni 10m või kuni 7 tonni mootorpaadid). Aluste paigaldamine (pakkimine) toimub Ascom kahveltõstukiga
c. Sisetingimustel põrandapinnal. Aluste paigaldamine (pakkimine) toimub Ascomi raamkraanaga (alused raskemad kui 25t) või platvormtreileriga Roodberg (alused kuni 25t).

2. Kuidas planeerite aluste asetust ellingutesse?

Aluste paigutamine oleneb müüdud aluste parameetritest. Alustame suuremate laevade paigutamisest, mida saab liigutada ainult raamkraanat kasutades. Leiame paberil kõige optimaalsema lahenduse. Vajadusel mõõdame kitsamad kohad ka ruumis kohapeal üle. Juhul kui on teada infot aluse järgmise navigatsioonihooaja kohta (näiteks, et soovib varem alustada) või talve jooksul planeeritavatest töödest (tööde teostamiseks on mõnes laeva piirkonnas rohkem ruumi vaja), siis arvestame laevade paigutamisel ka sellega. Ruumi suurte laevade vahel paigutame täis platvormtreileriga.

3. Millised on ellingute mõõtmed?

a. Elling „Kõrge“ (EK) – võimalik on sisse sõita ka raamkraanaga. Põrandapind 35mx57m
b. Elling „Madal“ (EM) – põrandale pakkimine toimub ainult Roodbergiga. Põrandapind 33,7mx57m

4. Kas aluste ellingutesse pakkimisel tuleb arvestada ka kõrgusega?

Kõrguse puhul seavad piiranguid:

a. uste kõrgus (EK – 12,5m; E,M 8,5m).
b. Raamkraana *(travel hoist)* üldkõrgus koos mastikraanaga allalastud asendis (11,6m)
c. Raamkraana *(travel hoist)* eesmise tala alumine kõrgus (9,5m) – suurim laeva kõrgus millest saab raamkraanaga üle sõita

5. Kui palju aluseid on vaja ellingutesse mahutada?

Kuna tegemist on äriettevõttega on ellingutesse paigutamise eesmärgiks täita võimalikult palju ruutmeetreid.

6. Millised on paatide mõõdud ja kui suuresti need varieeruvad?

Paatide mõõdud algavad 5 meetrist ja lõppevad 32 meetriga.

7. Kirjelda aluste paigutamisel ellingutesse esinevaid piiranguid?

a. Aluse kättesaadavus – aluste hooajad lõppevad erinevatel aegadel ning aeg-ajalt ei ole sobivaid aluseid sadamas tõsteks valmis
b. Tõstemehhanismide tehnilised piirangud (kabariidid, tõstevõime, pöörderaadiused)
c. Tollipiirangud (ei ole kohaldatud) – tollilao osa peab olema eraldatud
d. Põrandapindade äärtesse peab jääma käigukoridor 1 m

8. Kas kõikide aluste ellingust välja võtmine toimub ühel ajal?

Valdavalt on veeskamise ajaks aprilli II pool.

Lisa 2 - Teine intervjuu jahisadama kapteniga

1. Kuidas paigutate alused juhul kui neid on vaja erandkorras liigutada?

Erandkorras (ette teatamata) liigutamisi on väga keeruline planeerida. Erandkorras (sügisel kokkuleppimata) aluse liigutamine eeldab tavaliselt mitmete aluste liigutamist.

2. Kuivõrd arvestate aluste pakkimisel vöörinurgaga?

Vööri kuju (nurk ülaltvaates) arvestamine ei ole senini ruumi planeerimisel olulist rolli määranud. Pigem on olulisem vööri kuju (nurk külgvaates). Suuremate laevade vööride alla saab teinekord paigutada väiksemaid paate ning suurte laevade vöörid saab paigutada järgmise laeva ahtri kohale (see võimalus ilmneb paraku alles reaalsel pakkimisel ning sellega ei saa enne laeva sadamasse jõumist arvestada kuna ka samade mudelite seas võib olla erinevusi radarite, targa, antennide, kraanade jm lisavarustuse paigutusel).

3. Kui oluline on, et aluste pakkimisel arvestaks rakendus konkreetse vöörinurgaga? Kas sellekohased andmed on saadavad? Juhul kui täpse nurga andmed pole olulised, milline on aluste keskmine vöörinurk?

Nurk selgub kirjandusest. Aga keskmiselt võiks kasutada näiteks 60 kraadist nurka.

4. Millised on peamised vajadused rakenduse puhul?

Vajadused: arvestab võimalikult ökonoomselt põranda kasutuse, kiire arvutuskäik, visualiseerib põrandaplaani.

5. Millised andmed peaksid aluste nimekirjas olema kirjeldatud?

Nimekiri. nimi, pikkus, laius, kaal, omaniku nimi, kontakt. Staatus - kinnitatud(allkirjastatud)/tõenäoline.

6. Kui suure tõenäosusega võivad alusega seotud andmed muutuda? Kui muutuvad, siis millised andmed?

Alustega seotud andmed suure tõenaosusega ei muutu. Võib juhtuda, et aluse omanik võib aluse andmeid müügiprotsessi käigus väiksemana näidata, aga valdavalt jäävad sedasorti ebakõlad koheselt töötajatele silma, lisaks kontrollime laevade mõõtmed üle.

7. Kui oluliseks peate kustutatud teabele ligipääsu, näiteks kustutatud alustele? Kas võib tekkida vajadus neid uuesti kasutada?

Kustutatud teabele ligipääs ei ole oluline, kuna teave paikneb eraldi andmebaasides. Rakenduse vajalikkus on ennekõike planeerimisprotsessis põrandaplaani koostamisel.

8. Kas tulevikus võib lisanduda alasid, kuhu aluseid pakkida?

Rankenduse abi võiks kasutada ka välisterritooriumi (staapelplatsi) ja riiulite töö planeerimisel.

9. Milline info peaks kajastuma pakitud aluste plaanil?

Plaanil kajastuv info (nimi, mark) - täiendav info pikkus, laius, kaal.

Lisa 3 - Aluste nimekiri

Aluse nimetus	Pikkus (m)	Laius (m)	Kaal (t)
Nimbus 340	10,63	3,4	9
NC14	13,93	4,1	6
Princess V40	11,66	3,73	16
Anytec 622S	6,7	2,26	3
Ferretti 73	22,68	5,8	71
Princess 75	22,5	5,5	66
Princess 50	14	4	30
Draakon	8,9	1,95	2,5
Fourwinns	7,9	2,4	4
Azimut 55	16,5	4,86	35
Princess 72	20,7	5,49	69
Janneau NC9	10	3,15	5
Axopar 37	11,2	2,95	4,5
Boomeranger	10,5	3,5	3,5
Jeanneau 30	8,79	3,18	4
Rinker 300	9,6	2,89	5
Windy 32	9,68	3,26	4
Solaris 50	15	4,5	15
Yamarin 6110	6,2	2,6	1,5
Regal 2760	8,6	2,6	2,5
Phantom 40,	12,5	3,95	20
Dehler 98	9,8	3	6
NC11	11	3,7	4
Regal 2565	8,4	2,5	3
Nimbus 305	9,74	3,25	5
Halberg Rassy 36	11,31	3,55	10
Elling	14,52	4,4	18
Axopar 37C	11,2	3,3	5
Nauticat 441	15,6	3,75	19
Sunseeker 43	13,4	3,63	12

Targa 46	15,11	3,95	13
Galeon 430	12,7	4,15	16
Hanse 38	10,99	3,9	12
Galeon 550 Fly	17,7	4,76	35
Princess 64	19,8	5	43
Dufour 390	11,94	3,99	8
Delphia	10,8	3,45	9
Absolute 40	9,97	3,99	11
Pedro 1000	10	3,4	13
Princess 60	18,5	4,5	36
Oyster 485	14,76	4,27	19

Lisa 4 – Rakenduse pakkimise meetodid

```
private static List<Vector> GetVectors(double width, double length)
{
    var vectors = new List<Vector>()
    {
        new Vector((0, length - width / 2 / Math.Tan(
            BoatBowDegree / 2 * Math.PI / 180)), Direction.N),
        new Vector((-1 * width / 2, width / 2 / Math.Tan(
            BoatBowDegree / 2 * Math.PI / 180)), Direction.NW),
        new Vector((-1 * width / 2, -1 * width / 2 / Math.Tan(
            BoatBowDegree / 2 * Math.PI / 180)), Direction.SW),
        new Vector((0, -1 * (length - width / 2 / Math.Tan(
            BoatBowDegree / 2 * Math.PI / 180))), Direction.S),
        new Vector((width, 0), Direction.E)
    };

    return vectors;
}

private static BoatInRoomAndPlan PlaceBoat(
    BoatInRoomAndPlan boatInRoom, (double x, double y) point)
{
    boatInRoom.Vectors[0].StartPointX = point.x;
    boatInRoom.Vectors[0].StartPointY = point.y;

    for (var i = 0; i < boatInRoom.Vectors.Count; i++)
    {
        if (i != 0)
        {
            boatInRoom.Vectors[i].StartPointX =
                boatInRoom.Vectors[i - 1].EndPointX;

            boatInRoom.Vectors[i].StartPointY =
                boatInRoom.Vectors[i - 1].EndPointY;
        }

        boatInRoom.Vectors[i].EndPointX =
            Math.Round(boatInRoom.Vectors[i].StartPointX
                       + boatInRoom.Vectors[i].CoordinateX, 2);

        boatInRoom.Vectors[i].EndPointY =
            Math.Round(boatInRoom.Vectors[i].StartPointY
                       + boatInRoom.Vectors[i].CoordinateY, 2);
    }

    return boatInRoom;
}
```

```
private static List<Vector> CreateNoFitPolygon(
    BoatInRoomAndPlan packedBoat, List<Vector> beingPackedBoatVectors)
{
    var noFitPolygonVectors = new List<Vector>();
    var vectors = new List<Vector>();

    vectors.AddRange(packedBoat.Vectors);
    vectors.AddRange(beingPackedBoatVectors.Select(vector =>
        new Vector((x: -1 * vector.CoordinateX,
            y: -1 * vector.CoordinateY),
            GetOppositeDirection(vector.Direction))));

    noFitPolygonVectors.AddRange(vectors.FindAll(vector =>
        vector.Direction == Direction.SW).Select(vector =>
        new Vector(vector)));
    noFitPolygonVectors.AddRange(vectors.FindAll(vector =>
        vector.Direction == Direction.S).Select(vector =>
        new Vector(vector)));
    noFitPolygonVectors.AddRange(vectors.FindAll(vector =>
        vector.Direction == Direction.SE).Select(vector =>
        new Vector(vector)));
    noFitPolygonVectors.AddRange(vectors.FindAll(vector =>
        vector.Direction == Direction.E).Select(vector =>
        new Vector(vector)));
    noFitPolygonVectors.AddRange(vectors.FindAll(vector =>
        vector.Direction == Direction.NE).Select(vector =>
        new Vector(vector)));
    noFitPolygonVectors.AddRange(vectors.FindAll(vector =>
        vector.Direction == Direction.N).Select(vector =>
        new Vector(vector)));
    noFitPolygonVectors.AddRange(vectors.FindAll(vector =>
        vector.Direction == Direction.NW).Select(vector =>
        new Vector(vector)));
    noFitPolygonVectors.AddRange(vectors.FindAll(vector =>
        vector.Direction == Direction.W).Select(vector =>
        new Vector(vector)));

    for (var i = 1; i < noFitPolygonVectors.Count; i++)
    {
        noFitPolygonVectors[i].StartPointX =
            noFitPolygonVectors[i - 1].EndPointX;
        noFitPolygonVectors[i].StartPointY =
            noFitPolygonVectors[i - 1].EndPointY;
        noFitPolygonVectors[i].EndPointX =
            noFitPolygonVectors[i].StartPointX
            + noFitPolygonVectors[i].CoordinateX;
        noFitPolygonVectors[i].EndPointY =
            noFitPolygonVectors[i].StartPointY
            + noFitPolygonVectors[i].CoordinateY;
    }

    return noFitPolygonVectors;
}
```

```
private static bool IsPointOutsidePolygon((double x, double y) point,
    List<Vector> polygon)
{
    var count = 0;
    var vertex = false;
    foreach (var vector in polygon)
    {
        var pointEquationVectorStartPointValue =
            vector.StartPointY - point.y;
        var pointEquationVectorEndPointValue =
            vector.EndPointY - point.y;

        var vectorEquationPointStartPointValue =
            (point.x - vector.StartPointX)
            / (vector.EndPointX - vector.StartPointX)
            - (point.y - vector.StartPointY)
            / (vector.EndPointY - vector.StartPointY);
        var vectorEquationPointEndPointValue =
            (Math.Max(vector.StartPointX, vector.EndPointX)
                + 1 - vector.StartPointX)
            / (vector.EndPointX - vector.StartPointX)
            - (point.y - vector.StartPointY)
            / (vector.EndPointY - vector.StartPointY);

        if (pointEquationVectorStartPointValue < 0
            && pointEquationVectorEndPointValue < 0
            || pointEquationVectorStartPointValue > 0
            && pointEquationVectorEndPointValue > 0
            || vectorEquationPointStartPointValue < 0
            && vectorEquationPointEndPointValue < 0
            || vectorEquationPointStartPointValue > 0
            && vectorEquationPointEndPointValue > 0
            || IsValuesAlmostEqual(vector.StartPointX, point.x)
            && IsValuesAlmostEqual(vector.StartPointY, point.y)
            || IsValuesAlmostEqual(vector.EndPointX, point.x)
            && IsValuesAlmostEqual(vector.EndPointY, point.y))
        {
            continue;
        }
        if (IsValuesAlmostEqual(pointEquationVectorStartPointValue, 0)
           || IsValuesAlmostEqual(pointEquationVectorEndPointValue,0))
        {
            if (vertex)
            {
                vertex = false;
                continue;
            }
            vertex = true;
        }
        count++;
    }
    return count % 2 == 0;
}
```

```
private static (double x, double y)? FindBoatValidPosition(
    Room room, List<List<Vector>> polygons,
    BoatInRoomAndPlan boat, double yCoordinate)
{
    var i = yCoordinate;
    var j = 1 + boat.Boat.Width;

    double hoistDimensions = 0;
    if (boat.Boat.Weight > WeightLimit)
    {
        hoistDimensions = HoistWidth / 2 + boat.Boat.Width / 2;
        j = hoistDimensions;
    }

    while (i + boat.Boat.Length < room.Length)
    {
        while (j < room.Width
            - (boat.Boat.Weight > WeightLimit ? HoistWidth / 2 : 1))
        {
            var coordinate = (x: j, y: i);
            var isValid = polygons.All(polygon
                => IsPointOutsidePolygon(coordinate, polygon));

            if (isValid)
            {
                return coordinate;
            }

            j += 0.2;
        }

        j = 1 + boat.Boat.Width;
        i += 0.2;

        if (boat.Boat.Weight > WeightLimit)
        {
            j = hoistDimensions;
        }
    }

    return null;
}
```