TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Sander Kangur 206835IABB
Kaspar Kinko 206216IABB

Telefonirakenduste arendamine elektritarbimise mustrite kujundamiseks läbi mänguliste ja informatiivsete elementide AS Enefiti näitel

Bakalaureusetöö

Juhendaja: Karl-Erik Karu
MSc

Tallinn 2024

Autorideklaratsioon

Kinnitame, et oleme 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.

Autorid: Sander Kangur, Kaspar Kinko

20.05.2024

Annotatsioon

Bakalaureusetöö eesmärgiks on elektritarbimise käitumismustrite kujundamiseks ja elektritarbimise optimeerimiseks telefonirakenduse väljatöötamine AS Enefit klientide jaoks ning antud rakenduse tehnilise lahenduse kirjeldamine.

Elektritarbimise optimeerimine on kasutajate jaoks keeruline, sest olemasolevad lahendused ei toeta otseselt energiasäästlike ja optimaalsete harjumuste kujunemist ning alternatiivsed lahendused pole intuitiivsed ega lihtsalt kasutatavad. Lisaks pole elektritarbimise optimaalsuse kaardistamiseks loodud koondlahendust, kus kasutaja saaks piisavalt ülevaatliku pildi enda elektritarbimise harjumustest.

Selle probleemi lahendamiseks pakuvad autorid välja koondlahenduse, mis hõlmab endas päevase elektrihinna ülevaadet graafikuna, järjestikuse elektritarbimise kõige optimaalsemaid ajavahemikke ning mängulist elementi, mille kasutamine on intuitiivne ja premeeriv ning mis motiveerib kasutajat enda elektritarbimise harjumusi kujundama.

Bakalaureusetöö arendati ettevõttes Enefit AS. Probleemi lahenduseks välja pakutud rakendus realiseeriti, koondades endasse elektrihinna graafiku ja tarbimisintervallide loogika olemasolevatest lahendustest ning lisades juurde elektritarbimise mängulise elemendi. Projekti tulemusena valmis puhta koodi arhitektuurile vastav kolmekihiline tervikrakendus, mis oli nii testitud kui ka dokumenteeritud.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 59 leheküljel, 5 peatükki, 33 joonist, 8 tabelit.

Abstract

DEVELOPMENT OF A PHONE APPLICATION FOR SHAPING THE PATTERNS OF ELECTRICITY CONSUMPTION THROUGH GAMIFICATION AND INFORMATIVE ELEMENTS ON THE EXAMPLE OF AS ENEFIT

The aim of this bachelor thesis is to develop a mobile application for AS Enefit clients to shape and optimize their electricity consumption patterns, and to describe the technical solution of this application.

Optimizing electricity consumption is challenging for users, as existing solutions do not directly support the development of energy-efficient, optimal habits, and alternative solutions are not intuitive and easily usable. Additionally, there is no comprehensive solution created for mapping the optimality of electricity consumption, where a user could get a sufficiently comprehensive overview of their electricity consumption habits.

To address this problem, the authors propose a comprehensive solution that includes a graphical overview of the daily electricity price, the most optimal time intervals for consecutive electricity consumption, and a gamification element that is intuitive and rewarding, motivating the user to shape their electricity consumption habits.

The bachelor's thesis was developed at Enefit AS. The proposed application to solve the problem was implemented by incorporating the logic of electricity price charts and consumption intervals from existing solutions and adding a gamification element to electricity consumption. As a result of the project, a three-layer application adhering to clean code architecture was made, tested, and documented.

The thesis is in Estonian and contains 59 pages of text, 5 chapters, 33 figures, 8 tables.

Lühendite ja mõistete sõnastik

Backend	Rakendustase/äriloogikakiht
Branch	*Git'iga* seotud koodi talletamise asukoht
CI/CD	Pidev juurutamine / pidev integratsioon (ingl. c*ontinuous integration/continuous development*)
Database	Andmebaas
Frontend	Kasutajaliides
IA	Arvutisüsteemide instituut
Issue	Tarkvaraarendamise ülesanne
Master branch	Toodangu koodi talletamise asukoht
Merge	Arendatud *branchi* integreerimine *master branchi*
Mõõtepunkt	Kinnisvaraline objekt, millele on sõlmitud Enefit AS elektrileping
Swagger	FastAPI isegenereeruv dokumentatsioon
System architecture	Süsteemi arhitektuur
Tehnoloogia *stack*	Kasutatavad tehnoloogiad (sh. programmeerimiskeeled, arenduskeskkonnad, andmebaas)
User Story	Kasutusjuht
UX/UI	Kasutaja kogemus / kasutaja liides (ingl. *user experience / user interface*)
Widget	Ükskõik milline *Flutteri* olem (sh. Nupp, tekstiväli, graafik), ühtlasi nimetatakse nii ka mobiiltelefoni avakuval väikest rakendust või elementi

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Energia jätkusuutlik kasutamine on tänapäeva ühiskonnas muutunud üha olulisemaks, kuna see mängib võtmerolli nii keskkonna kaitses kui ka majandusliku efektiivsuse edendamises. Oluline on, et tarbijad oleksid oma energiakasutuse mustritest teadlikud ja mõistaksid, kuidas optimeerida oma energiatarbimist. Selles kontekstis on autorid keskendunud mobiilirakenduse väljaarendamisele, mis on loodud Android ja iOS platvormidele. Rakenduse eesmärk on võimaldada AS Enefit klientidel jälgida oma jooksva kuu elektritarbimist ja sellega seotud kulusid, pakkudes samal ajal mängulisi ja informatiivseid elemente. Need elemendid on suunatud tarbijate energiatarbimise mustrite kujundamisele. Rakenduse kaudu saavad kasutajad mitte ainult jälgida oma energiakasutust, vaid ka osaleda erinevates mängulistel väljakutsetel, mis julgustavad säästlikku energiakasutust. Süsteemi toimimist kirjeldatakse Eesti Energias meeskonna projekti raames arendatud elektritarbimise telefonirakenduse näitel.

1.1 Probleem

Elektrienergia tarbimise mõistmine ja juhtimine on tänapäeva ühiskonnas keskne teema, eriti energia säästmise ja keskkonnasäästlikkuse kontekstis. AS Enefiti klientide seas on täheldatav suur vajadus paremate vahendite järele, mis aitaksid mõista ja optimeerida elektritarbimise mustreid. Praegused tööriistad ja andmete esitamise viisid ei ole piisavalt intuitiivsed ega kasutajasõbralikud, mistõttu on tarbijatel raskusi oma energiakasutuse efektiivsel juhtimisel. See piirang takistab teadlikku ja säästlikku energiakasutust, mis on hädavajalik ressursside säästmise ja keskkonnamõjude vähendamise vaatenurgast. Lisaks on oluline mõista, et säästliku energiakasutuse edendamine hõlmab ka tarbijate käitumisharjumuste kujundamist, et soodustada pikaajalisi ja jätkusuutlikke muutusi energiakasutuses. Selleks on vaja tööriistu, mis mitte ainult ei informeeri tarbijaid, vaid ka motiveerivad ja toetavad neid energiasäästlike harjumuste loomisel.

1.2 Eesmärk

Lõputöö eesmärgiks on välja töötada mobiilirakendus AS Enefiti klientide jaoks, mis võimaldab neil oma elektritarbimise käitumismustreid mõista ja optimeerida, aidates kaasa teadlikuma ja säästlikuma energiakasutuse suunas liikumisele. Eesmärgi saavutamiseks on välja töötatud järgmised alamülesanded:

- kasutajate elektritarbimise andmete kogumine ja visualiseerimine, pakkudes neile lihtsat ja intuitiivset viisi oma tarbimisharjumuste jälgimiseks ning analüüsimiseks;
- mängulistele elementidele loomine, et innustada kasutajaid oma energiakasutust teadlikumalt ja säästlikumalt korraldama. See hõlmab tagasiside pakkumist tarbimiskäitumise kohta ning eesmärkide seadmist ja nende saavutamise jälgimist;
- kolmekihilise arhitektuuri (*backend*, *frontend*, *database*) väljatöötamine, mis järgib puhta arhitektuuri põhimõtteid, tagades rakenduse paindlikkuse, laiendatavuse ja hooldatavuse;
- üldise kasutajakogemuse parandamine, pakkudes mugavat ja kasutajasõbralikku liidest, mis võimaldab kliendil hõlpsasti navigeerida ja rakenduse funktsioone kasutada;
- rakenduse kättesaadavaks tegemine nii Androidi kui ka iOS-i platvormidel, tagades laia kasutajaskonna ja kergesti kättesaadava funktsionaalsuse mobiilsetel seadmetel;
- tarbimisharjumuste muutumise soodustamine, juhtides tähelepanu säästlikumatele ja ökonoomsematele valikutele, läbi teadlikkuse tõstmise ja ergutades osalemist energiasäästlikes tegevustes;
- põhjaliku dokumentatsiooni ja tehniliste lahenduste testimise läbiviimine, et tagada rakenduse usaldusväärne ja tõrgeteta toimimine erinevatel seadmetel ja platvormidel;

Need alamülesanded on hoolikalt kavandatud, et tagada rakenduse terviklikkus ja kasutajatele sujuv kasutuskogemus. Iga funktsioon on loodud selleks, et toetada ja

parandada Enefiti klientide igapäevast energiakasutust, võimaldades neil teha teadlikke valikuid, mis aitavad kaasa energiasäästlikumale ja jätkusuutlikumale tulevikule.

Lisaks tuleb märkida, et kuigi mängulised elemendid on olulised tarbimisharjumuste muutmiseks, ei käsitleta *gamification* osa antud töös põhjalikumalt, kuna sama meeskonna teine töö nimega "Mänguelemendi mõju kasutajakogemusele Enefit AS mobiilirakenduse näitel" süveneb sellesse teemasse eraldi [1].

1.3 Töö struktuur

Töö on struktureeritud viieks põhiosaks. Esimene peatükk koosneb sissejuhatusest, kus tutvustatakse probleemi, eesmärki ning antakse lühiülevaade töö ülesehitusest. Teine peatükk keskendub teoreetilisele taustale, käsitledes mobiilirakenduste arendamise keskseid mõisteid ja teooriaid. Kolmandas peatükis, metoodika ja tööriistade jaotises tutvustatakse Pythoni ja FastAPI kasutamist *backend* arenduses, SQLAlchemy-d *database* halduseks ORM lahenduse kaudu ning Flutterit ja Darti kui *frontend* tehnoloogiaid, pakkudes detailset ülevaadet kasutatud arendusprotsessidest ja -vahenditest. Neljandas peatükis, rakenduse disaini ja arenduse osas, kirjeldatakse rakenduse arhitektuuri ning antakse põhjalik ülevaade iga komponendi kohta, sealhulgas *backend* loogika, *database* skeemid, *frontend* liidesed ja *backendi* ning *frontendi* ühiktestid ja integratsioonitestid. Viimases sisulises, analüüsi peatükis, analüüsitakse saadud tulemusi, tehakse järeldusi, tuuakse välja edasiarendusvõimalused ja olemasolevate lahendustega võrdlus.

2 Metoodika

Selles peatükis kirjeldatakse, milliseid tööriistu, raamistikke, programmeerimiskeeli ja tehnoloogiaid rakenduse arendamiseks kasutati ning milliseid teoreetilisi arhitektuurilisi põhimõtteid implementeeriti. Lisaks kirjeldatakse, kuidas toimis arendus agiilses Kanban keskkonnas.

2.1 Objekti kirjeldus

Käesolev töö on edasiarendus meeskonnaprojekti aine "ITB1706 Infosüsteemide arendamise meeskonnaprojekt: tellimus" raames Enefitis loodud projektile. Esialgse projekti käigus töötati välja mobiilirakendus, mis võimaldab Enefit AS klientidel turvaliselt autentida ja reaalajas jälgida NordPool börsi elektrihindu. Rakendus tuvastab tulevaste tundide kõige soodsamad hinnaperioodid ja arvutab välja nende keskmise hinna, aidates kasutajatel optimeerida oma elektritarbimist. Lisaks funktsionaalsuse arendamisele pöörati projektis tähelepanu ka arenduskeskkonna loomisele ja esialgsele koodiarhitektuuriline.

Lõputöö raames lisatakse rakendusele mitmeid uuenduslikke funktsioone. Integreeritud on elektrihindade *widget*, mis kuvab reaalajas elektrihinnad ja kõige soodsama hinnaperioodi otse mobiiltelefoni avakuvale. Rakendusse on samuti lisatud võimalus liituda käimasoleva nädala väljakutsega, mis soodustab regulaarset osalemist. Nädalapõhine väljakutsete süsteem kuvab kasutajatele nende eelneva päeva edusamme ja edetabel näitab kasutajate positsioone võrreldes teistega.

Bakalaureusetöö raames jagati neljaliikmeline meeskond kaheks kaheliikmeliseks rühmaks. Üks rühm pühendus rakenduse mänguliste elementide arendamisele, samal ajal kui teine keskendus rakenduse üldisele arhitektuurilisele ülesehitusele. Hoolimata rühmade eri fookustest, jätkati koostööd kui ühtne meeskond ühise projekti edendamisel. Antud uurimistöö keskendub rakenduse arhitektuurilistele aspektidele, uurides, kuidas arhitektuurilised valikud toetavad rakenduse üldist funktsionaalsust ja kasutajakogemust.

2.2 Arendustööriistad

Projekti arendustööriistade valik põhines Enefit AS E-Labi seenior-arendajate soovitatud tehnoloogilisel stackil. Valiku tegemisel arvestati kahe olulise asjaoluga: arendustöö toimub ettevõtte poolt väljastatud arvutitega, kasutades ettevõtte võrku ning kasutusel peavad olema ainult eelnevalt heakskiidu saanud tarkvaralised programmid. Seetõttu pidi valitud tarkvara-arhitektuurne lahendus olema kooskõlas ettevõtte auditeeritud programmidega.

Tarkvara arendamisel kasutati mitmeid tööriistu ja platvorme, et tagada efektiivne ja sujuv arendusprotsess. Arenduskeskkonnana kasutati Visual Studio Code (VS Code) rakendust, mis pakkus laia valikut laiendusi ja tuge erinevatele programmeerimiskeeltele, võimaldades tõhusat koodi kirjutamist ja silumist.

Tarkvara arendamisel ja andmehalduse teostamisel kasutati andmebaasile juurdepääsuks pgAdmin kasutajaliidest. pgAdmin on võimas ja intuitiivne tööriist PostgreSQL andmebaaside haldamiseks ja visualiseerimiseks. Selle graafiline kasutajaliides võimaldab hõlpsasti navigeerida andmebaasi struktuuris, täita SQL-päringuid, hallata andmebaasi objekte (nt tabeleid, vaateid, indekseid) ning jälgida andmebaasi jõudlust ja terviklikkust [2]. Andmebaasi enda loomisel ja haldamisel kasutati PostgreSQL relatsioonilist andmebaasihaldussüsteemi (RDBMS). PostgreSQL on tuntud oma usaldusväärsuse, andmete terviklikkuse ja rikkalike funktsioonide poolest. See toetab keerukaid päringuid, laiendusi ja andmetüüpide laia valikut, võimaldades arendajatel luua skaleeritavaid ja tõhusaid andmebaasirakendusi [3]. PostgreSQL-i kasutamine koos pgAdminiga tagab tõhusa andmehalduse ja võimaldab arendustiimil keskenduda rakenduse funktsionaalsuse arendamisele, pakkudes samal ajal tugevat ja paindlikku andmebaasi haldusvõimalust [4].

Versioonihalduse jaoks rakendati GIT-i ja GitHubi, mis võimaldasid koodimuudatuste jälgimist ja koostööd meeskonnaliikmete vahel. Lisaks kasutati GitHub Actions liidest Continuous Integration/Continuous Development (CI/CD) protsesside ja automaattestide rakendamiseks, mis tagas pideva integreerimise ja tarnimise, vähendades koodi veatõenäosust ja kiirendades arendusprotsessi [5].

Rakenduse konteinerdamiseks ja käivitamiseks kasutati Dockerit ja Docker Desktopi, mis võimaldasid ühtlustada arenduskeskkonda ja lihtsustada rakenduse juurutamist

erinevatesse keskkondadesse [6]. HTTP päringute tegemiseks ja API testimiseks kasutati Postmani ja Postman Desktopi, mis pakkusid intuitiivseid tööriistu API-de testimiseks ja silumiseks [7].

Mõõtepunktide andmete simuleerimiseks kasutati Mockapi.io lahendust, mis võimaldas luua ja hallata virtuaalseid API-sid ja andmekogusid, tagades testimiseks vajalikud andmed [8]. Lisaks kasutati andmete lugemiseks ka Exceli failidest, mis võimaldasid integreerida ja töödelda olemasolevaid andmekogusid rakenduse arendamisel [9].

Koodi testimisel kasutati rakendustaseme testimiseks *Python* mooduleid *Pytest* ja *unit test* ning esitlustaseme testimiseks Flutteri sisseehitatud moodulit *Flutter test*. Testimisprotsess hõlmas endas rakenduse manuaalset testimist telefonis ning koodisiseseid unit- ja integratsioonteste [10] [11].

2.2.1 Raamistikud

Backendi arendamiseks kasutati FastAPI raamistikku, mis on Pythonis kirjutatud veebirakenduste loomise raamistik. FastAPI on välja töötatud kiirust, lihtsust ja paindlikkust silmas pidades, keskendudes eriti APIde kõrgele jõudlusele. See võimaldab arendajatel kasutada Python 3.6+ tüübiviiteid API parameetrite ja tagastusväärtuste deklareerimiseks, toetades automaatset andmete valideerimist. Oluliseks eeliseks on automaatselt genereeritud interaktiivne API dokumentatsioon (Swagger UI), mis võimaldab API päringuid hõlpsalt testida [12]. FastAPI toetab ka asünkroonset programmeerimist, parandades veebirakenduste jõudlust. Lisaks on integreeritud tugi lihtsale andmete valideerimisele ja serialiseerimisele, kasutades Pydantic raamatukogu [13].

Frontend arendati kasutades Flutterit, mis on Google'i poolt loodud UI toolkit, võimaldades arendajatel luua visuaalselt atraktiivseid, sujuvalt toimivaid ja platvormiüleseid kasutajaliideseid. Flutterit toetab Dart programmeerimiskeel, pakkudes suurt komplekti eelvalmistatud *widget*-eid ja kiiret arendustsüklit, mis muudab kasutajaliidese loomise efektiivsemaks ja nauditavamaks [14].

Esialgne prototüüp loodi FlutterFlow programmiga, mis on visuaalne arenduskeskkond, mis võimaldab kasutajatel ehitada mobiil- ja veebirakenduste liideseid ilma sügava

kodeerimisoskuseta. Lisaks genereerib FlutterFlow automaatselt ehitatud liideste Flutter koodi, et seda reaalses arenduses kasutada [15].

2.2.2 Programmeerimiskeeled

Backendis kasutati Pythonit, mis on laialdaselt tunnustatud oma võimsuse ja paindlikkuse poolest serveripoolsetes rakendustes [16]. *Frontendis* eelistati Dart keelt, mida kasutatakse tihti mobiil- ja veebirakenduste arendamisel oma efektiivsuse ja kompaktsuse tõttu . Pideva integratsiooni ning juurutamise (CI/CD) protsessides kasutati YAML keelt. YAML on andmeserialiseerimisvorming, mis on eriti populaarne konfiguratsioonifailides tänu oma loetavusele ja lihtsusele [17]. JSON kasutati arenduskeskkonna konteinerite loomiseks, mis on veel üks levinud andmeserialiseerimisformaat, tuntud oma universaalsuse ja kergesti ligipääsetavuse tõttu eri programmeerimiskeeltes [18].

2.2.3 Tehnoloogiad

Lahenduse backendi jaoks kasutati sõnumivahetuseks HTTP protokolli ning REST liideseid. Liideseid dokumenteeriti ning kirjeldati SwaggerUI tööriistaga, mis aitab arendajatel mõista ja testida API funktsionaalsust interaktiivselt [19]. Frontendis kasutati kaasaegseid veebitehnoloogiaid, sealhulgas HTTP kliente andmete vahetamiseks REST API-dega ja JSON formaate andmete serialiseerimiseks ning deserialiseerimiseks. Olekuhalduseks rakendati tõhusat lahendust, nagu Riverpod, mis võimaldab paindlikult hallata rakenduse seisundit, tagades sellega parema sooritusvõime ja kasutajakogemuse [20].

Arenduskeskkond loodi kasutades Visual Studio Code DevContainers lahendust. DevContainers võimaldab defineerida ja käivitada konteineripõhist arenduskeskkonda, tagades, et kõik arendajad töötavad samadel tingimustel konfigureeritud keskkonnas, mis sisaldab kõiki vajalikke sõltuvusi ja tööriistu. See lahendus aitab vältida tüüpilisi arendamisega seotud probleeme nagu olukord, kus kood töötab vaid arvuti kontekstis. Lisaks toetab DevContainers kiiret ja hõlpsat arenduskeskkonna seadistamist, võimaldades arendajatel keskenduda rakenduse loomisele ilma pideva keskkonna seadistamiseta [21].

2.3 Arendusmetoodika

2.3.1 Projekti taust

Projekti arendustiim koosnes neljaliikmelisest tudengite meeskonnast. Igale tudengile oli määratud üks personaalne mentor Enefit E-Lab tarkvaraarendajate tiimist. Tudengid töötasid Enefit AS ettevõttes tasustatud praktikalepingu alusel, kaks päeva nädalas.

2.3.2 Arendamine ettevõttes

Arendusprotsessil järgiti agiilset Kanban metoodikat, mille teoreetiline taust pärineb Jaapanist ja oli algselt välja töötatud Toyota tootmissüsteemis kasutamiseks. Kanban, mis tõlkes tähendab "visuaalne kaart" või "tahvel", võimaldab meeskondadel jälgida tööprotsesse visuaalselt, kasutades tahvleid, kus ülesanded liiguvad erinevate arenguetappide vahel [22]. Erinevalt teistest agiilsetest meetoditest nagu Scrum, Kanban ei nõua kindlaid ajalisi tsükleid (sprinte), vaid keskendub pidevale töövoo optimeerimisele ja tõhususe suurendamisele, mis toetab paindlikku reageerimist muutustele ning suurendab produktiivsust ja töö kvaliteeti [23].

Lisas 4 on välja toodud projektitiimi Kanban tahvel.

Enefiti poolt oli tudengitele toeks mentorid. Tudengite meeskond hakkas arendama alates 2023. aasta oktoobrist, alustades kasutajalugude kirjutamisest ja *frontend* vaadete loomisest Flutter Flow programmiga. Projekti raames rakendati visuaalset tahvlit, kus jooksvad ülesanded ehk *issued* kuvati veergudesse, mis esindasid erinevaid tööetappe, aidates seeläbi optimeerida töövoogu ja tõsta efektiivsust. Tudengid võtsid enda peale vastutuse kõik vajalikud *issued* luua. Mentorite juhendamisel toimunud koosolekutel määratleti uued kasutajalood (*user stories*) ja koostati nende saavutamiseks vajalikud ülesanded. Ülesanded lisati GitHubi ja jaotati seejärel meeskonnaliikmete vahel.

Arendusprotsessi Kanban metoodika järgiti kõiki sellele omaseid etappe:

- igahommikused püstijalakoosolekud, mis olid umbes 10 minutised koosolekud, kus iga tudengite tiimi liige täpsustas, mida ta tegi oma arendustegevuses eile, mida teeb täna ja millised arendusega seotud probleemid tema tööd mõjutavad;

- rafineerimine, mis toimus peale demot, kus meeskond arutas täidetud ülesanded üle ning määras uued prioriteedid. Selles etapis uuendati Kanbani tahvlit, lisades uued ülesanded ja kohandades olemasolevaid vastavalt projekti vajadustele;
- pidev töövoog, kus *issued* realiseeriti arendustegevuses järjepidevalt, ilma ajapiiranguta;
- demokoosolek, kus toimusid mentoritele ja tootejuhile esitlused, analüüsiti saadud tagasisidet ja otsustati järgmiste ülesannete üle;
- tagasivaatus, kus vaadati tagasi lõpetatud tööle, arutati, mis läks hästi ja mis mitte, ning otsiti viise arendusprotsessi parendamiseks;

Töö toimus pidevas voos, kus iga etapi lõppedes oli võimalik kohe järgmisele etapile üle minna, mis on Kanban arenduse põhiprintsiip.

2.3.3 Rafineerimine

Rafineerimine hõlmas endas *issuede* loomist, antud toimingut viisid läbi tudengid ise. *Issuedele* määrati raskusaste numbrivahemikus *1* kuni 5, kus 1/2/3 tähendas, et arendamisele kulub 1/2/3 päev(a), 4 tähendas, et arendamisele kulub nädal ja 5 tähendas, et *issue* on liiga suur ja see peaks liigendama väiksemateks *issuedeks.* Rafineerimisel arutatud teemad realiseeriti teemablokkidena GitHubis *User Story*-dena. Iga tudeng valis endale meelepärase *issue,* määras selle enda GitHubi kasutajale ja hakkas seda koodis realiseerima. Jooksvate probleemide lahendamiseks abistasid tudengid üksteist ning suuremate probleemide korral pöörduti mentorite poole.

2.4 Protsess

2.4.1 Versioonihaldus

Arendusülesande valmimisel laeti see GitHubi üles, hoides valminud koodi eraldi harus ehk *branchis.* Enne valminud koodi integreerimist *master* harusse tõmbetaotluse (ingl. *Pull request*) näol, pidid kõik tudengid andma kirjutatud koodile heakskiidu, et tagada olukord, kus uus muudatus ei lõhuks ära vana funktsionaalsust ja et uus muudatus on toimiv koodi lisa. Lisaks tudengite enda ülevaatusele pidi kolm mentorit arendatud tüki heaks kiitma, alles siis realiseerus koodi integratsioon *master* harusse. Juhul kui mentorid või kaastudengid polnud muudatustega rahul, jäeti tõmbetaotluse külge GitHubis sõnum, kus kirjeldati probleemi või arendustöö puudujääke. Sel juhul pidi ülesande eest vastutav lahendama puudujäägi ning saatma oma muudatuse taas-ülevaatusele.

Lisaks manuaalsele ülevaatusprotsessile ja GitHubile iseloomulikule tõmbetaotluse kasutamisele, kasutati ka GitHub *Actions CI/CD* ja *backend* ning *frontend* automaattestide lahendust. *CI/CD* hõlmas endas *frontendile* ja *backendile* loodud automaattestide kasutamist. Antud testide põhiline eesmärk oli kontrollida, et arenduses poleks sisse jäänud koodi-süntaksilisi vigu. *Backend* ja *Frontend* automaattestide eesmärk oli jooksutada esitlus- ja rakendustaseme integratsioonitestide faile.

GitHub *Actions* skript seadistati GitHubis, installides see enda projekti külge, GitHubi moodulite loetelust. Skripti sisu realiseeriti koodis .github/workflows kaustas .yml faili sisuna.

Kogu arendamise *CI/CD* protsessi komponendid olid rakendatud selliselt, et pidev arendus toimus pideva tõmbetaotluste ülevaatamise ja *issue* lõpetamisel uue *issue* endale määramise teel. Pidev integratsioon toimus eelnevalt seadistatud GitHub *Actionsi* kasutamisel ning GitHubi konfigureerimisel selliselt, et kolme tõmbetaotluse heakskiidu saamisel sai arendus-*branch* automaatselt *maini* integreerida (ingl. *merge*).

2.4.2 Testimine

Rakenduse testimisel kasutati BDDd *(ingl. behaviour driven development)* ja ühiktestimise mustrit AAA *(*ingl. *arrange, act ja assert).* BDDd järgides luuakse stsenaariume, mis testivad kasutaja sisendit ja sellele vastavaid süsteemi reaktsioone. Stsenaariumi ülesehitus on kirjeldatav järgnevalt, “Kui mul on x andmed, siis ma saan kasutada funktsiooni y ning see annab mulle vaste z” [24]. AAA (Arrange, Act, Assert) on testide struktureerimise muster, mis aitab kirjutada loetavaid ja mõistetavaid ühikteste. *Arrange* etapp seadistab testi algtingimused, *Act* etapis toimub tegevus, mida testitakse ja *Assert* etapis kontrollitakse, kas tulemus vastab ootustele [25].

2.4.3 Puhas kood

"Puhas kood" on kontseptsioon, mida käsitleb Robert C. Martin raamatus "Clean Code: A Handbook of Agile Software Craftsmanship". See rõhutab parimaid praktikaid koodi kirjutamiseks, mis on loetav, hooldatav ja vigadeta [26]. Järgnevalt on välja toodud "Puhta koodi" põhiprintsiibid:

Puhas kood peab olema hästi struktureeritud ja loetav, et teised arendajad saaksid sellest kergesti aru ja seda hooldada. Selged ja tähendusrikkad nimed ning kommentaarid aitavad keerukaid osi selgitada [26].

Funktsioonid peaksid olema väikesed ja täitma ainult ühte ülesannet, kasutades minimaalseid parameetreid ning vältides kõrvalmõjusid, mis suurendavad korduvkasutatavust ja testitavust [26].

Kood peab olema hõlpsasti testitav, võimaldades automatiseeritud testide loomist ja käivitamist. Sõltuvuste süstimine ja lahtine sidumine komponentide vahel on testitavuse tagamiseks olulised [26].

Refaktoreerimine parandab pidevalt koodi struktuuri ilma välist käitumist muutmata, aidates säilitada koodi kvaliteeti ja vähendada tehnilist võlga. See eemaldab koodilõhnad, nagu duplikaatne kood ja liiga suured funktsioonid [26].

Puhas kood on kriitilise tähtsusega tarkvaraprojektide pikaajalise edu tagamiseks, aidates vähendada vigade arvu ning suurendades tarkvara kvaliteeti ja arendusprotsessi tõhusust [26].

2.5 Arhitektuurilised kontseptsioonid

2.5.1 Kolmekihiline arhitektuur

Kolmekihiline arhitektuur on laialdaselt kasutatav tarkvaraarenduslik mudel, mis jaotab rakenduse kolme selgelt eristuvasse kihti: esitluskiht, äriloogikakiht ja andmebaasikiht (Joonis 1). See jaotus võimaldab parandada koodi loetavust, haldavust ja skaalatavust, muutes rakenduse arendusprotsessi efektiivsemaks ja süsteemi hoolduse lihtsamaks [27].

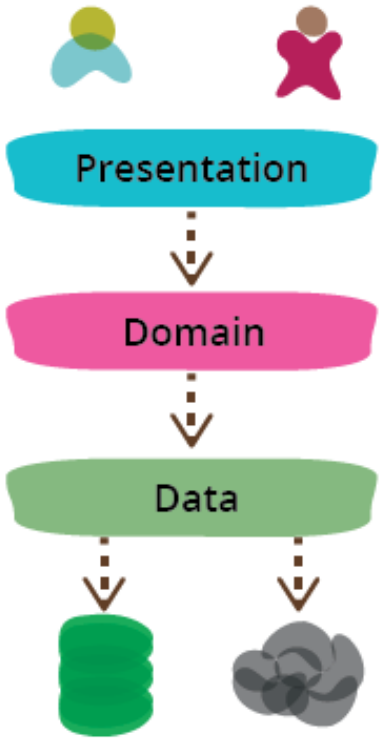
Järgnevalt on toodud kolmekihilise arhitektuurimudeli kolm peamist kihti:

- esitluskiht ehk *frontend* vastutab kasutajaliidese ja kasutaja interaktsioonide eest. See kiht käsitleb kõiki aspekte, mis on seotud andmete esitamisega kasutajale, ning on otseses suhtluses kasutajaga, võimaldades neil süsteemiga suhelda [27];

- äriloogikakiht ehk *backend* on süsteemi tuum, kus defineeritakse ja käideldakse rakenduse põhifunktsionaalsusi. See kiht sisaldab reegleid ja protseduure, mis

käsitlevad kasutaja poolt esitatud andmete töötlust ning tagavad äriprotsesside nõuetekohase täitmise [27];

- andmebaasikiht haldab andmete salvestamist ja talletamist. See kiht vastutab andmete säilitamise, taastamise ja turvalisuse eest, tagades, et andmeid saab usaldusväärselt säilitada ja vajadusel kätte saada [27].



Joonis 1. Kolmekihiline arhitektuur [28].

Nagu Martin Fowler oma raamatus "Patterns of Enterprise Application Architecture" märgib, on kolmekihilise arhitektuuri eesmärk tagada, et iga kiht keskenduks oma konkreetsetele ülesannetele, vähendades sõltuvusi ja suurendades süsteemi modulaarsust. See eraldamine aitab arendajatel paremini mõista süsteemi struktuuri ja võimaldab koodi lihtsamini loetavaks, testitavaks ja skaleeritavaks muuta. Samuti võimaldab see kihtide vahelist kommunikatsiooni standardiseerida, mis on eriti kasulik suurtes ja keerulistes süsteemides [27].

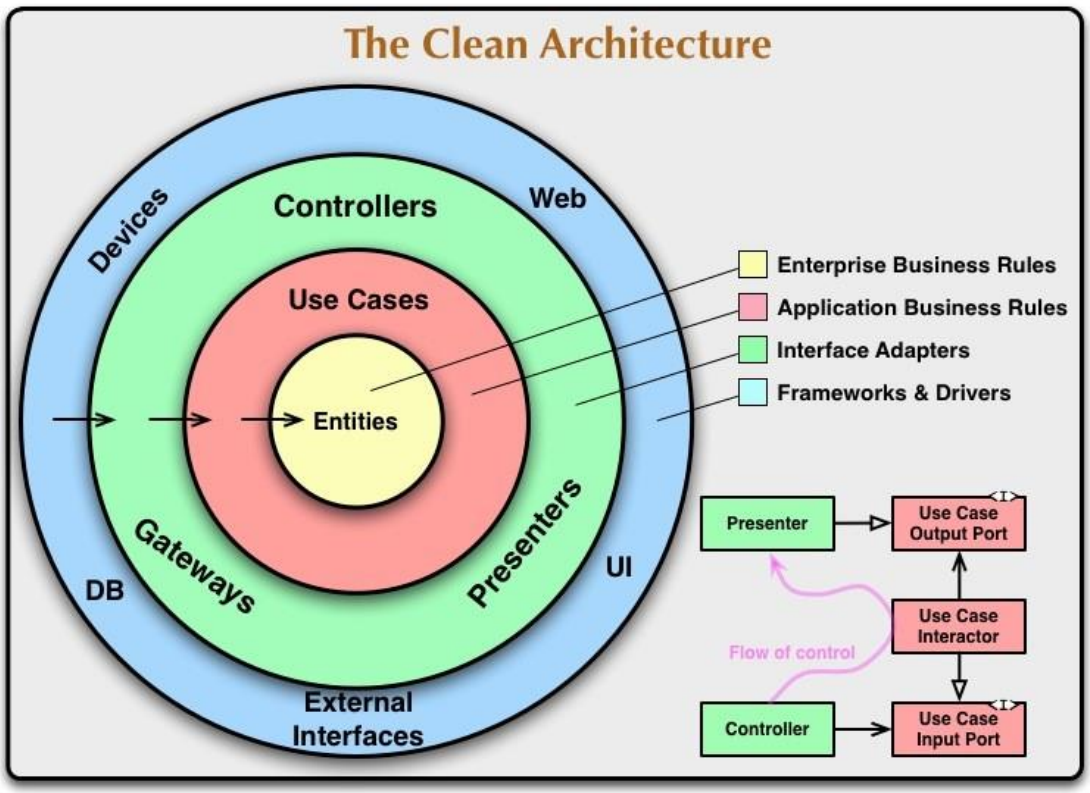
Kolmekihilise arhitektuuri rakendamine on muutunud üheks standardiks, mis võimaldab arendusmeeskondadel tõhusamalt hallata keerukaid rakendusi, vähendades samal ajal vigade riski ja suurendades süsteemi üldist kvaliteeti ja hooldatavust [27].

2.5.2 Puhas arhitektuur

Clean architecture ehk puhas arhitektuur on koodi struktureerimise viis, mis vähendab koodikomponentide sõltuvust üksteisest ja milles kirjutatud kood on kergesti loetav ning laiendatav. Antud arhitektuuriline mudel tugineb mitmete varasemate tarkvaraarhitektuuriliste konseptsioonide põhimõtetel, sealhulgas Hexagonal arhitektuur, Onion arhitektuur ja Layered arhitektuur [29].

Clean Architecture kontseptsiooni populariseeris Robert C. Martin (tuntud ka kui Uncle Bob) oma 2012. aastal avaldatud artiklis, kus ta esitas mudeli, mis soodustab süsteemi isoleeritust ja muudab selle komponentide vahelised sõltuvused minimaalseks. Selle arhitektuuri kese on loogika ja reeglid, mida rakendatakse süsteemi ülesehituses, tagades, et ärireeglid ei sõltu kasutajaliidesest, andmebaasist ega muudest välissüsteemidest [30].

Martin tõi välja, et puhta arhitektuuri peamine eesmärk on luua süsteeme, mis on vastupidavad muutustele, lihtsasti testitavad ja mõistetavad. Puhas arhitektuur soodustab ka mudeli ja esituskihtide eraldatust, mis aitab arendajatel säilitada süsteemi paindlikkust ja teostada komponentide vahelisi suhteid läbi selgelt defineeritud liideste. Selline disainiline lähenemine võimaldab arendajatel keskenduda süsteemi ärireeglitele, mis on pikemas perspektiivis kasulik süsteemi üldisele jätkusuutlikkusele [30].



Joonis 2. Puhta arhitektuuri kihiline mudel [30].

Puhast arhitektuuri kirjeldab joonis 2. Puhta arhitektuuri keskmes on *entities*, mis esindavad süsteemi ärireegleid. Ümbritsevad kihid, nagu *use cases* ja *controllers* pakuvad järjestikuseid abstraktsioonitasemeid, mis aitavad eraldada ärireegleid kasutajaliidesest ja välistest sõltuvustest nagu andmebaasid ja muud süsteemid. Iga kiht on oluline süsteemi terviklikuks toimimiseks ja aitab kaasa selle modulaarsusele ja paindlikkusele.

3 Nõuded

Selles peatükis kirjeldatakse ära rakenduse nõuded, analüüsitakse olemasolevaid lahendusi, uuritakse nende puudujääke ning tuuakse välja, mis moodi tudengite poolt arendatav rakendus olemasolevate rakenduste puudujääke lahendab. Nõuded kategoriseeritakse tarkvaraarhitektuurilisteks (ingl. *System architecture*), kasutajakogemuslikeks (ingl. UX/UI nõuded), mittefunktsionaalseteks ja funktsionaalseteks.

3.1 Nõuete kujunemine

Hetkel puudub turul selline telefonirakendus, mis pakuks kasutajatele koondatud ülevaadet elektrihinna kulgemisest päevas, isiku enda päevase elektritarbimise mahust ja rahalisest kulust. Kuigi elektrihinna päevase kõikumise graafikud ja isikliku elektrienergia kulu andmed on erinevates allikates saadaval, ei ole need andmed ühest kohast mugavalt kättesaadavad. Kuna tänapäeval on väga populaarseks kujunenud igapäevainformatsiooni koondumine nutiseadmetesse, tekkis nõue realiseerida eelmainitud lahendus just telefonirakenduse näol.

Rakenduse põhiline fookus on kasutaja elektritarbimise harjumuste kujundamine ja säästliku elektritarbimise propageerimine, mis võib osutuda motiveeriva ja lihtsasti kasutatava vahendi puudumisel väljakutsuvaks. Antud probleemi lahendamiseks läheneti asjale mängulises võtmes. Mängu loogika loodi selliselt, et kasutaja ei pea enda jaoks välja töötama mingit optimaalset tarbimisharjumust vaid peab järgima talle ette antud optimaalse elektritarbimise mustrit.

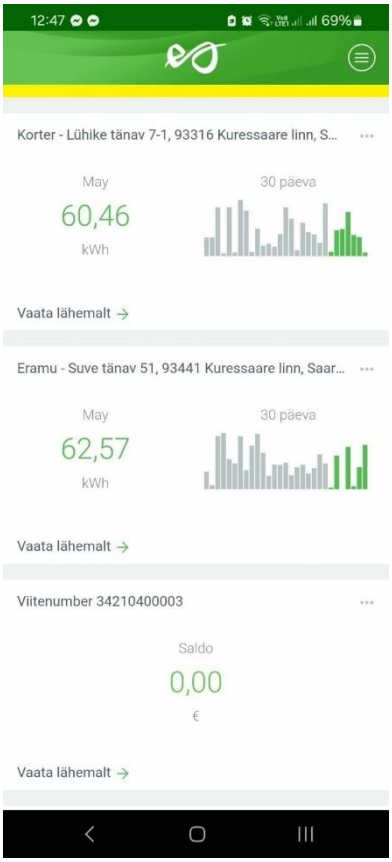
3.2 Olemasolevate lahenduste analüüs

Lahenduse asjakohasuses veendumiseks viisid tudengid läbi põgusa analüüsi olemasolevatest lahendustest, et veenduda, kas arendatav rakendus leiaks kasutust turul ning milliseid planeeritud rakenduse komponente on juba realiseeritud, veendumaks selles, et ei hakataks taaslooma mingeid lahendusi, mis on juba hästi realiseeritud.

Lisaks komponentide lahenduste analüüsimisele uuriti ka seda, kas või kuidas olemasolevad lahendused üritavad kasutaja elektritarbimise harjumusi kujundada.

3.2.1 Enefit Eesti telefonirakendus

Enefit Eesti telefonirakenduse näol on tegemist Elering DataHubi elektritarbimise logi koondamisega lihtsasti kasutatavaks rakenduseks (Joonis 3, 4). Rakendus kuvab kasutajale tema Enefit AS elektrilepinguga sõlmitud kinnisvaraliste objektide elektritarbimise andmeid, elektri börsihinda ning sellist võrdlusvahendit, kus kasutaja saab enda erinevate päevade tarbimisandmeid võrrelda. Loodud lahenduses on olemas kasutaja päevase tarbimise võrdlemise funktsionaalsus, kuid projekti optimaalse tarbimise matkimist ei ole realiseeritud.



Joonis 3. Enefit Eesti telefonirakenduse avakuva vaade [31].

Joonis 4. Enefit Eesti telefonirakenduse korterivaade [31].

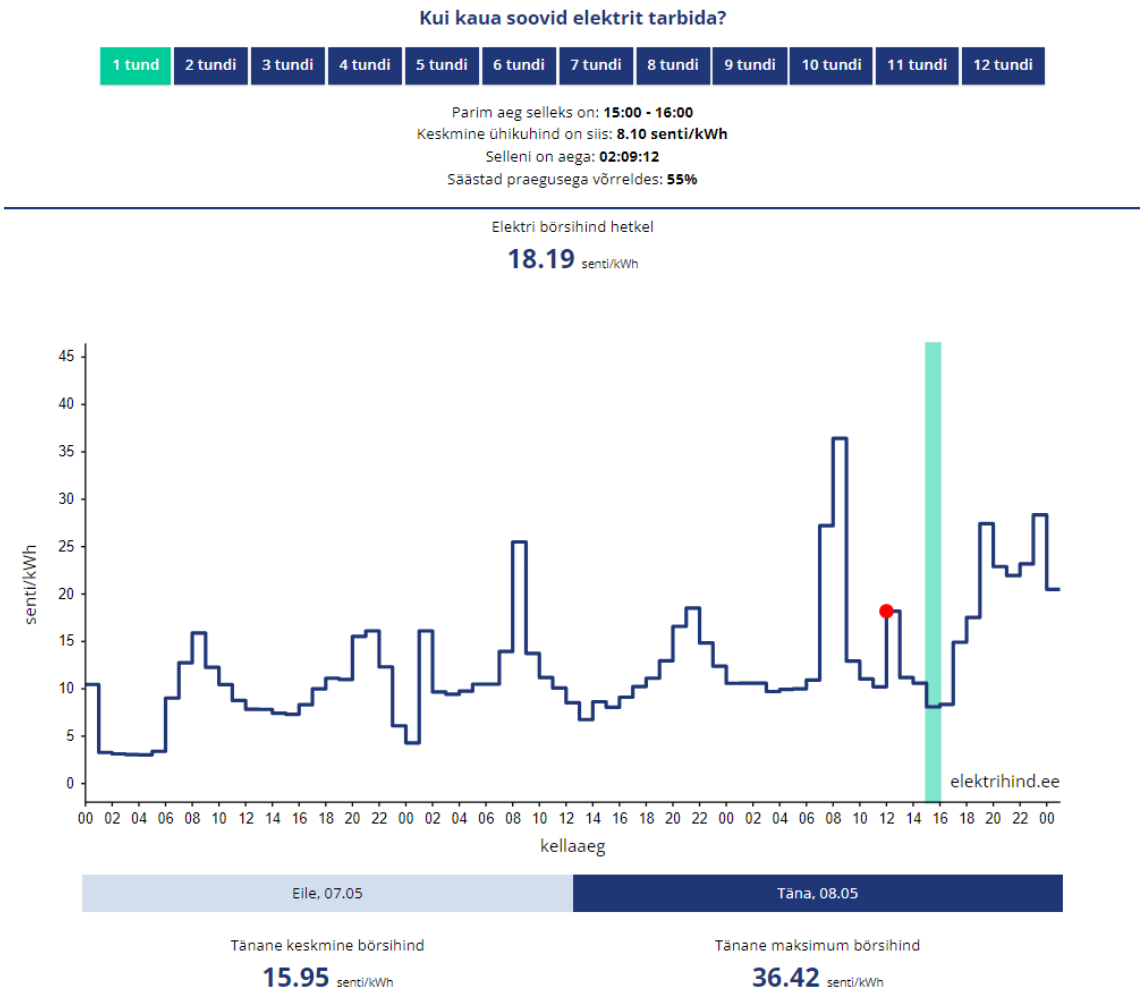
3.2.2 Elering DataHub

Elering DataHub pakub enda kodulehel kasutaja elektritarbimise koondvaate lahendust, eeldusel, et kasutajal on kehtiv Enefit AS elektrileping. Antud lahendus sisaldab endas logi kasutaja elektritarbimisest, logi põhjal loodud tarbimisgraafikut ning rahalist kulu jooksval kuul. Elektritarbimise harjumuste kujundamist otseselt ei puudutata [32].

3.2.3 Elektrihind.ee börsihinna tarbimissoovitaja

Elektrihind.ee kodulehel on realiseeritud jooksva päeva elektrihinna kõikumise graafik, mis sisaldab endas järjestikuste tundide odavaima elektritarbimise vahemiku funktsionaalsust (Joonis 5). Kasutaja valib tarbimise perioodi pikkuse vahemikus 1 kuni 12 tundi ja graafikul kuvatakse kasutaja jaoks kõige odavam vahemik. Lisaks on kuvatud päeva keskmine börsihind, maksimaalne börsihind ja odavaima elektritarbimise vahemiku keskmine hind. Antud funktsionaalsus tundus loogiline ja hästi mõistetav ning sarnaselt realiseeriti ka graafik tudengite projektis. Elektritarbimise harjumuste kujundamist antud lahendus ei puuduta ja optimaalne tarbimine on realiseeritud

tarbimisvahemiku funktsionaalsusega, mis pole otseselt optimaalse tarbimise terviklahendus [33].



Joonis 5. Elektrihind.ee börsihinna tarbimissoovitaja [33].

Tabel 1 analüüsib tudengite rakenduse komponentide esinemisi olemasolevates lahendustes.

Tabel 1. Olemasolevate lahenduste võrdlus tudengite rakendusega.

	Elektrihinna päevane hinna-graafik	**Kasutaja päevane elektri-tarbimine**	**Mänguline element**	**Elektrihinna graafiku widget**	**Järjestikuse elektri-tarbimise soovitaja**
Tudengite rakendus	ON	ON	ON	ON	ON
Enefit Eesti telefoni-rakendus	EI OLE	ON	EI OLE	EI OLE	EI OLE

	Elektrihinna päevane hinna-graafik	**Kasutaja päevane elektri-tarbimine**	**Mänguline element**	**Elektrihinna graafiku widget**	**Järjestikuse elektri-tarbimise soovitaja**
Elering DataHub	EI OLE	ON	EI OLE	EI OLE	EI OLE
Elektrihind.ee börsihinna tarbimis-soovitaja	ON	EI OLE	EI OLE	EI OLE	ON

3.3 Funktsionaalsed nõuded

Rakenduse funktsionaalsed nõuded kujunesid Enefit Eesti tooteomaniku visioonist ja tudengite meeskonna mentorite nõuetest. Tabel 2 esitab vaadeldava rakenduse funktsionaalsed nõuded.

Tabel 2. Rakenduse funktsionaalsed nõuded.

Tüüp	**Nõude kirjeldus**
Rakenduse haldus ja mängu loomine	Rakenduse haldur saab luua jooksva nädala mängu.
Autentimine ja kasutaja interaktsioon	Rakendus võimaldab sisselogimist Smart-IDga, pakkudes turvalist ja mugavat autentimismeetodit. Kasutajad saavad mänguga liituda mitme kinnisvaralise objektiga, Eleringi kontekstis Meter pointiga.
Kasutaja tegevused	Rakenduse edetabel uueneb automaatselt, hoides mängu konkurentsivõimelise ja kaasahaaravana. Kasutajal on võimalus mängust lahkuda ja oma tarbimisinfo kustutada, tagades kasutajate privaatsuse ja andmekaitse. Kasutaja saab rakenduse graafikut kuvada enda telefoni avakuval widgeti näol. Kasutajal on võimalik rakenduse graafikul leida kõige odavam järjestikuse tarbimise intervall.

3.4 Mittefunktsionaalsed nõuded

Rakenduse mittefunktsionaalsed nõuded kujunesid algselt Enefit Eesti tudengite meeskonna mentorite nõuetest ning hilisemalt arendusprotsessist tulenevalt, et tagada ühiselt mõistetav ja rakendatav arendus- ja tööprotsess. Tabel 3 esitab vaadeldava rakenduse mittefunktsionaalsed nõuded.

Tabel 3. Rakenduse mittefunktsionaalsed nõuded.

Tüüp	**Nõude kirjeldus**
Arendusprotsess	Rakendus järgib pideva juurutamise (Continuous Deployment, CD) ja integratsiooni (Continuous Integration, CI) põhimõtteid. See tagab tarkvaraarenduse protsessi efektiivsuse, kuna muudatused integreeritakse ja juurutatakse automaatselt, vähendades vigade riski ja kiirendades uute funktsioonide kasutuselevõttu.
Skaleeritavus ja hooldatavus	Rakendust on võimalik edasi arendada, tagades tarkvara pikaajalise elujõulisuse ja kohandamisvõime muutuvatele nõuetele.
Deployment ja konteineriseerimine	Kõik tarkvara on jooksutatav *Dockeris* lokaalselt, võimaldades ühtlustatud ja lihtsustatud arendus-, testimis- ja tootmiskeskkondi. Konteineriseerimine aitab vältida "töötab minu masinas" probleeme, muutes rakenduse käivitatavaks erinevates seadmetes ja arenduskeskkondades.
Koodi kvaliteet	Järgitakse puhta koodi ja koodiarhitektuuri parimaid tavasid. See hõlmab selge, loetava ja hooldatava koodi kirjutamist, mis järgib *Clean-Code* ja *Domain-Driven Development* standardeid ja parimaid praktikaid, tagades sujuva meeskonnatöö ja koodi loetavuse.
Dokumentatsioon	Tehniline lahendus on dokumenteeritud, tagades, et arendusmeeskond ja tulevased kasutajad mõistavad süsteemi ülesehitust ja toimimist. Hea dokumentatsioon aitab kaasa rakenduse hooldatavusele ja laiendatavusele, võimaldades uutel arendajatel lihtsamini projekti panustada.
Kvaliteedi tagamine	Tehniline lahendus on testitud, mis tähendab, et tarkvara läbib hoolikad testimisetapid (näiteks koodisisesed Unit testid ja GitHub Actions’iga seotud automaattestid), et tagada selle usaldusväärsus, jõudlus ja turvalisus. Testimine on oluline kvaliteedi tagamiseks ning võimalike vigade ja probleemide tuvastamiseks ja lahendamiseks enne tootmisse lükkamist.

Tüüp	Nõude kirjeldus
Arendusprotsess ja tehnoloogia	Rakenduse *frontend*, *backend* ja andmebaas jooksevad samas DevContaineris. See lihtsustab arendusprotsessi, tagades ühtse arenduskeskkonna, mis hõlbustab rakenduse komponentide integreerimist ja testide läbiviimist.
Tarkvaraarhitektuur	Rakendus on jaotatud eraldiseisvateks komponentideks: on olemas eraldiseisev ja Dockeris jooksev *backend*, *frontend* ning andmebaas. Kogu *frontendi* andmed pärinevad suhtlusest *backendi* APIga.
Kasutajakogemus	Rakendus kuvab avamisel elektrihindade graafikut ja võimaldab Smart-ID kaudu sisse logida, pakkudes seeläbi mugavat ja kiiret kasutajakogemust. Kasutajal on võimalik kuvada elektrihindade graafikut oma nutiseadme avakuval widgeti näol, mis suurendab rakenduse kasutusmugavust ja juurdepääsetavust.
Litsentsimine	Kasutatavad välisrakendused pole litsentseeritud, mis tähendab, et tarkvara kasutab avatud lähtekoodiga komponente, vältides sellega litsentsitasusid ja tagades suurema paindlikkuse ning kohandatavuse. See aitab vähendada projekti kogukulu ja kiirendab arendusprotsessi.

3.5 Kasutuslood

Kasutusjuht: Logi sisse

Tegutsejad: Kasutaja, Smart-ID sisselogimislahendus

Kirjeldus: Kasutaja sisestab sisselogimisekraanil enda isikukoodi ja vajutab “Log in” nupule. Smart-ID saadab kasutajale tagasi autentimiskoodi, mida kuvatakse sisselogimisekraanil. Kasutaja valib õige autentimiskoodi enda Smart-ID telefonirakenduses ja sisestab PIN1. Kui autentimine õnnestus logitakse kasutaja sisse ja kuvatakse sisselogimise vaate asemel profiili vaadet. Kui autentimine ebaõnnestub kuvatakse kasutajale ebaõnnestumise teksti sisselogimise vaates.

Kasutusjuht: Kasutaja tahab näha päevase elektrihinna graafikut

Tegutsejad: Kasutaja, Elering

Kirjeldus: Kasutaja avab rakenduse. Kasutajale kuvatakse kodulehel Eleringi publitseeritud päevase elektrituru tunnihindu graafilisel kujul.

Kasutusjuht: Kasutaja tahab liituda väljakutsega olles sisse loginud

Tegutsejad: Kasutaja, Elering

Kirjeldus: Kasutaja avab rakenduse ja valib navigeerimisriba pealt väljakutse ikooni. Kasutaja vajutab "Join a challenge" nupule. Talle kuvatakse rippmenüüna kõik tema isikukoodile vastavad kehtiva Enefit AS elektrilepinguga asukohad. Kasutaja valib ühe asukoha ning talle kuvatakse väljakutse ekraan.

Kasutusjuht: Kasutaja tahab liituda väljakutsega olles sisselogimata

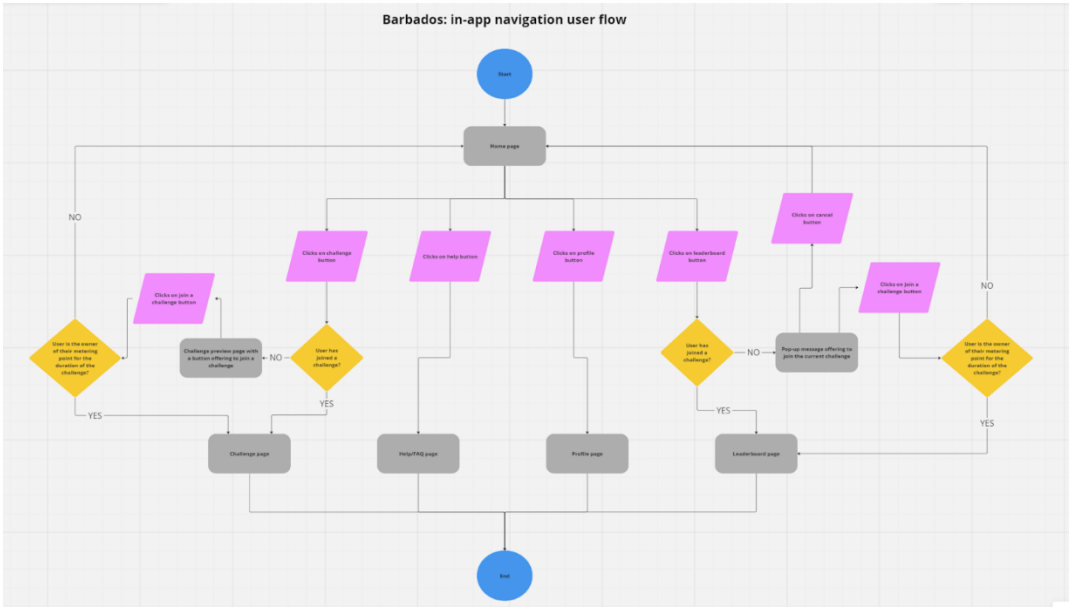
Tegutsejad: Kasutaja

Kirjeldus: Kasutaja avab rakenduse ja valib navigeerimisriba pealt väljakutse ikooni. Kasutaja vajutab "Join a challenge" nupule. Talle kuvatakse sõnum sisuga "To join a challenge you must be logged in" ning nupp "Log in". Nupule vajutades navigeeritakse kasutaja sisselogimise ekraanile.

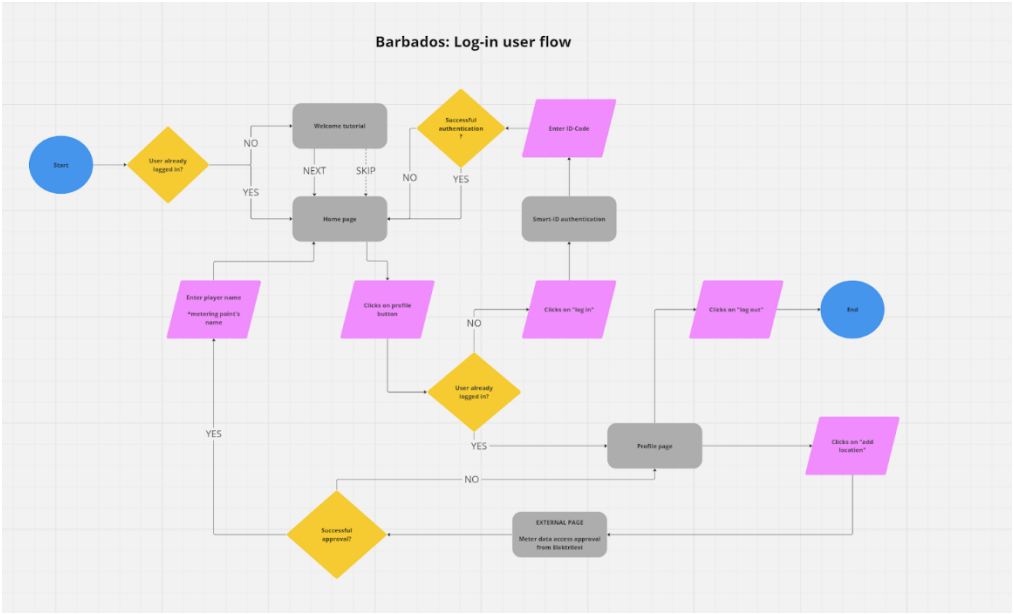
Kasutusjuht: Kasutaja tahab näha oma paiknemist edetabelis

Tegutsejad: Kasutaja

Kirjeldus: Kasutaja avab rakenduse ja valib navigeerimisriba pealt edetabeli ikooni. Kui kasutaja on väljakutsega liitunud kuvatakse kasutajale tema skoor ja paiknemine teiste väljakutses osalejatega. Kui kasutaja pole väljakutsega liitunud kuvatakse talle vastav sõnum.



Joonis 6. Kasutaja kasutuslugude diagramm.



Joonis 7. Kasutaja sisselogimise diagramm.

Rakenduse funktsionaalsus on kokkuvõtlikult järgnev: kasutajad saavad Smart-ID abil sisse logida, vaadata Eleringi poolt avaldatud päevase elektrihinna graafikut, liituda väljakutsetega sõltuvalt nende sisselogimise staatusest ning jälgida oma positsiooni väljakutse edetabelis (Joonis 6, 7).

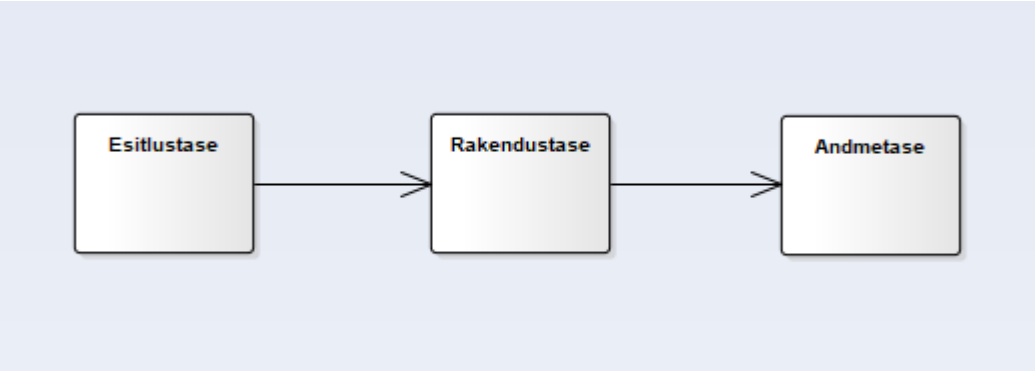
4 Tulemused

Järgnevas peatükis kirjeldatakse kogu rakenduse struktuuri, arhitektuuri ja kasutatavaid tehnoloogiaid ning rakenduse arhitektuuri kihte süvitsi. Iga kihi puhul analüüsitakse selle toimimist kogu arhitektuuri kontekstis, tuuakse välja kihisisesed komponendid ja nende tööpõhimõtted ning komponente kirjeldavad joonised.

4.1 Arhitektuur

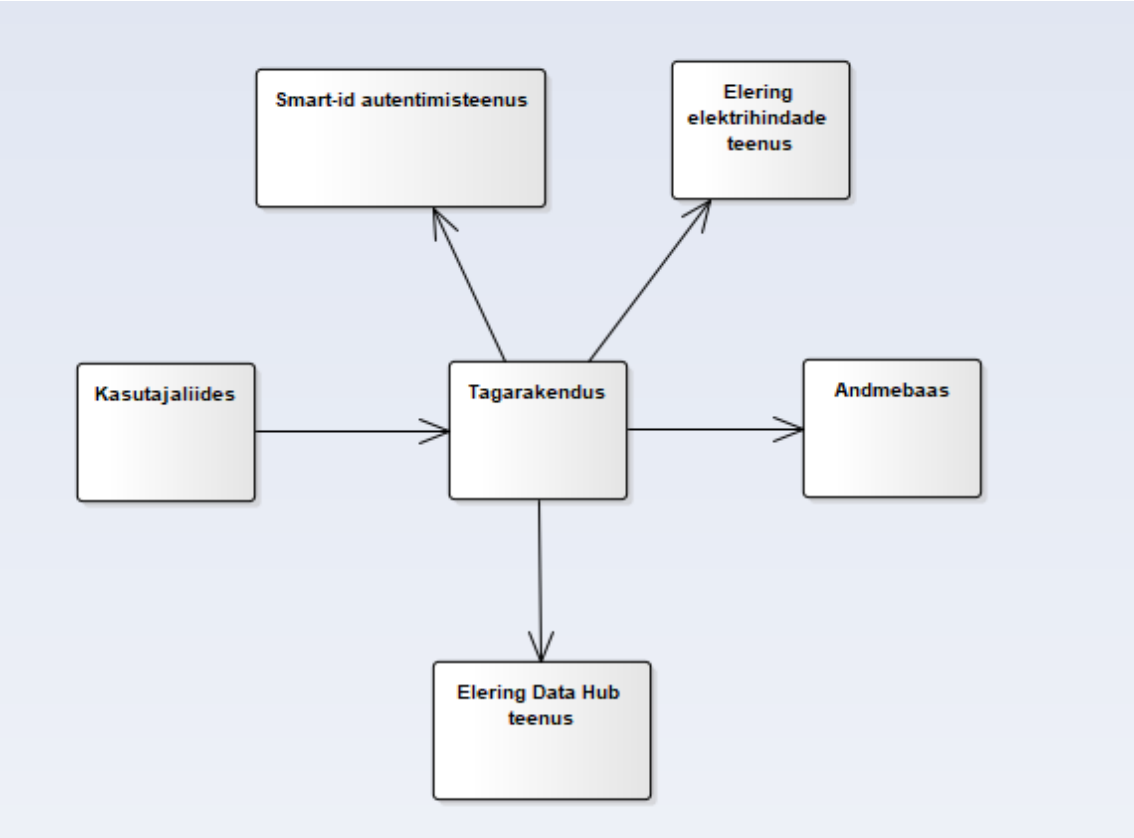
Tiimiprojekti käigus välja töötatud telefonirakendus põhineb kolmetasemelisel arhitektuuril. Sellise struktuuri valiku tegid mentorid, pidades silmas selle sobivust projekti eesmärkidega – see on lihtne ja tõhus, võimaldades meeskonnal keskenduda rohkem rakenduse põhifunktsionaalsuste arendamisele, hoides arhitektuuri piisavalt lihtsana, et kiiresti reageerida muutuvatele nõuetele. Kolmetasemeline arhitektuur koosneb esitlustasandist, rakendustasandist ja andmetasandist.

Joonis 8 allpool kujutab visuaalselt eelmainitud klassikalist kolmetasemelist süsteemi arhitektuuri, illustreerides iga kihi vastutust ja omavahelisi seoseid.



Joonis 8. Kolmetasemelise süsteemi arhitektuuri disain.

Joonisel 9 on esitatud konkreetsed tehnoloogilised komponendid, mis rakendavad kolmetasemelise arhitektuuri põhimõtteid praktilises projekti kontekstis.



Joonis 9. Lahenduse tehniline arhitektuur

Kasutajaliidese tasandil valitud Flutter toetab ristplatvormilist arendust, millele mentorid osutasid kui eelistatud valikule tänu selle kasvavale populaarsusele ja võimalustele. Teades juba projekti alguses, et Android Studio kasutamine pole võimalik, oli Flutteri valik strateegiline, mis võimaldas meil arendust tõhusalt läbi veebi edasi viia, hoides samal ajal fookust mobiilse kasutajakogemuse optimeerimisel.

Rakendustasandi arendati FastAPI-ga, sest see pakub kiiret arenduskogemust ja lihtsat integratsiooni erinevate tehnoloogiatega. Lisaks otsustas meeskond kasutada SQLAlchemy-d, ORM-raamistikku, mis lihtsustab andmete haldust ja suurendab süsteemi skaleeritavust, pakkudes eeliseid võrreldes lihtsamate mitte-ORM lahendustega. SQLAlchemy toetab ka andmete järjepidevuse tagamist ja tehingute juhtimist, mis võimaldab arendajatel efektiivsemalt hallata andmebaasi operatsioone ja vähendada andmetega seotud vigu.

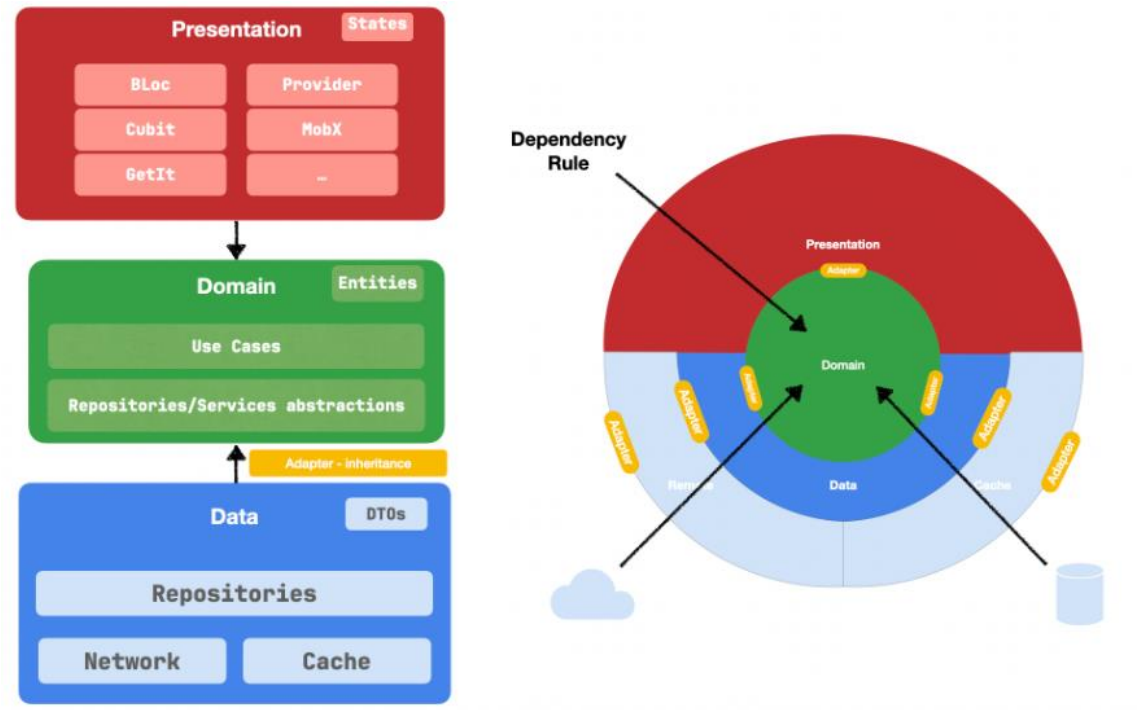
Andmetasandi lahendusena eelistati PostgreSQLi, mis on tuntud oma kõrge jõudluse poolest ja oli meeskonnale varasemast tuttav.

Välisteenustena integreeriti lahendused Elering elektrihindade teenus ja Data Hub-ile. Elering elektrihindade teenus valiti, kuna see pakub ligipääsu Nord Pooli Day-Ahead elektrihindadele, lisades nendele käibemaksu, mis on projektis oluline. Otse Nord Poolist päritud hinnad ei sisalda käibemaksu, seega Elering elektrihindade teenuse kasutamine vähendab vajadust täiendavate arvutuste järele, kuna see teeb käibemaksu lisamiseks vajalikud arvutused ise enne hindade edastamist. Data Hub teenust kasutatakse, kuna see toetab elektritarbimise andmete juurdepääsu ja haldust, võimaldades projektil koguda kasutaja tarbimisandmeid. See kombinatsioon annab võimaluse analüüsida kasutaja käitumist seoses elektrihindadega, pakkudes nii väärtuslikku tagasisidet energiatarbimise optimeerimiseks.

Projekti ajaliste piirangute tõttu otsustati simuleerida Datahub API lõpp-punkti. See strateegiline otsus võimaldab paindlikku üleminekut reaalse API kasutuselevõtuks tulevikus, kui on implementeeritud süsteem, kus on vaja saada kasutajatelt nõusolek nende andmete kasutamiseks. Simuleerimine aitab tagada süsteemi funktsionaalsuse testimise ja valmiduse, ilma et oleks vaja kohe täielikult reaalseid andmeid kasutada, võimaldades samal ajal jätkata arendustööd ilma katkestusteta.

4.1.1 Esitlustaseme arhitektuur

Esitlustase koosneb kolmest *clean architecture* printsiipidele toetuvast komponendist: esitluskiht (ingl. *presentation layer*), domeenikiht (ingl. *domain layer*) ja andmekiht (ingl. d*ata layer*) (Joonis 10).



Joonis 10. Esitlustaseme skeem [34].

Esitluskiht vastutab dünaamilise ja reageeriva kasutajaliidese eest. Kiht on jaotatud neljaks funktsionaalseks alamkihiks. Esimese ehk andmete halduse alamkihi (ingl. *provider*) funktsionaalsus baseerub *Flutter riverpodil*, mis käitub kasutajasisendist lähtudes, dünaamilise andmete salvestamise ja lugemise komponendina (Tabel 4. Andmete halduse komponendid rakenduses ja nende kirjeldus.), sidudes endas domeenikihi ja andmekihi funktsionaalsuse, muutes seeläbi andmed dünaamiliselt kasutatavaks. Teises alamkihis on defineeritud rakenduse vaated (ingl. *screens*). Kolmandas ehk teema alamkihis (ingl. *themes*) defineeritakse rakenduses kasutatavad värvikoodid, et tagada rakenduseülene visuaalne ühtlus. Neljandas vidinate alamkihis (ingl. *widgets*) defineeritakse rakenduses korduvkasutatavad kohandatud komponendid nagu navigeerimisriba, tavateated, veateated, graafik, edetabel, nupud, väljade ülesehitused.

Tabelis 4 kirjeldatakse ära Flutter riverpodiga loodud andmehaldurid ja nende komponendid.

Tabel 4. Andmete halduse komponendid rakenduses ja nende kirjeldus.

Andmehaldur (ingl. provider)	**Komponendid**	**Kirjeldus**
Badges provider	availableBadgesProvider earnedBadgesProvider	Tagastab kõik märgid, mida mängija saab teenida. Tagastab kõik märgid, mille mängija on juba teeninud.
Challenge chart provider	selectedDayIndexProvider getChallengeChartProvider	Tagastab selle päeva indeksi (vahemikus 1 kuni 7), millise challenge graafikut mängija kuvada soovib. Tagastab valitud indeksiga challenge graafiku.
Challenge provider	currentWeekChallengeFutureProvider personHasJoinedChallengeNotifier	Tagastab jooksva nädala challenge objekti. Tagastab oleku selle kohta, kas kasutaja on ühinenud jooksva nädala challenge'iga
Graph provider	getGraphDataProvider durationProvider selectedIndexProvider	Tagastab jooksva päeva elektrihinna graafiku andmed. Tagastab mängija valitud elektritarbimisperioodi pikkuse numbrina. Tagastab mängija valitud elektritarbimisperioodi indeksi.
Leaderboard provider	getProgressDataProvider getLeaderboardsByChallengeIdProvider	Tagastab mängija optimaalsusnäitaja. Tagastab viimase edetabeli mängijate seisu.
Login provider	loginProcessProvider	Vastutab logimisprotsessi olekute eest.
MeterPoint provider	meterPointNotifier meterPointsProvider	Tagastab mõõtepunkti, millega on kasutaja mänguga sidunud. Tagastab kõik mõõtepunktid, mis kasutajal üldse on.

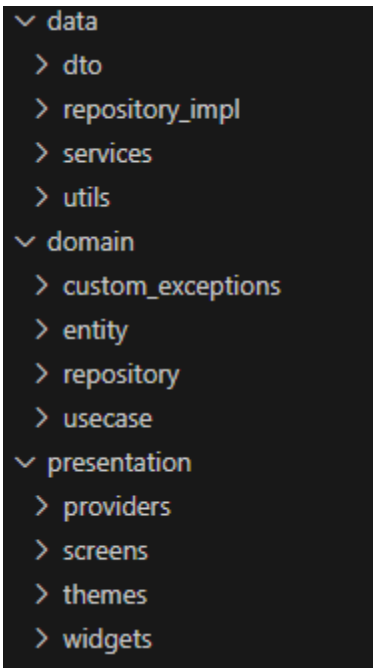
Andmehaldur (ingl. provider)	**Komponendid**	**Kirjeldus**
Person provider	authProvider addPersonIfAbsentProvider	Tagastab kasutaja oleku sisselogimise kohta ja tema andmed Lisab isiku andmebaasi, kui isikut andmebaasis salvestatud pole.
Player provider	playerNotifier joinChallengeProvider	Tagastab kasutaja mängija kindla kasutaja id ja challenge id alusel. Kasutaja lisatakse mängu
Time utils provider	timeUtilsProvider	Tagastab aja haldamise abivahendid.

Selle arhitektuurse ülesehituse oluliseks täienduseks on *local storage* süsteemi rakendamine läbi *provider'ite*. *Local storage* on vajalik komponent, mis aitab säilitada andmeid otse seadmes, pakkudes kasutajatele pidevat juurdepääsu teabele isegi siis, kui võrguühendus puudub. Antud lahendus ei paranda ainult rakenduse reageerimiskiirust, vaid vähendab ka serveripõhiste päringute vajadust, suurendades seeläbi rakenduse jõudlust ja parandades kasutajakogemust.

Domeenikiht toimib sillana esitluskihi ja andmekihi vahel. Antud kiht sisaldab endas ärireegleid ja kasutusjuhtumeid, mis defineerivad, kuidas kasutajaliidese tegevusi tõlgendatakse ja milliseid operatsioone need käivitavad. Siin toimub ka andmete valideerimine ja äriprotsesside koordineerimine, tagades, et kasutaja tegevus on kooskõlas rakenduse ärireeglitega. Domeenikiht koosneb neljas funktsionaalsest alamkihist. Esimeses alamkihis defineeritakse kohandatud veateated (ingl. c*ustom exceptions*) nagu konflikt (ingl. c*onflict exception*), viga laadimisel (ingl. *failed to load exception*), viga komponendi leidmisel (ingl. *not found exception*) ja vale sisend (ingl. *wrong input exception*). Teises alamkihis defineeritakse olemid (ingl. *entity*) *challenge chart, challenge, cheapest period, graph, leaderboard, meter point, person* ja *player*. Kolmanda alamkihi moodustavad repositooriumd iga olemi kohta . Neljandas alamkihis implementeeritakse olemite kasutuslood (ingl. *use case*), mis realiseerivad repositooriumimeetodeid.

Andmekiht esitlustasemel tegeleb andmete hankimisega rakendustaseme API-endpointidest. Antud kihi ülesandeks on suhelda rakendustaseme teenustega, hallates andmevahetust, mida on vaja kasutajaliidese funktsioonide toetamiseks. Selle kihi kaudu toimub andmevahetus REST API-de kaudu, mis tähendab, et andmeid päritakse ja edastatakse standardiseeritud veebiteenuste kaudu. Andmete kiht korraldab vajalike HTTP päringute tegemist, andmete deserialiseerimist ja edasist töötlemist domeeni kihis määratletud ärireeglite järgi, tagades andmete kooskõla ja sünkroonsuse kogu rakenduse erinevate osade vahel, minimeerides kasutajaliidese ja rakendusloogika vahelisi sõltuvusi. Andmekiht koosneb neljast funktsionaalsest alamkihist. Esimeses alamkihis defineeritakse andmekandmisobjektid (ingl. *data transfer objects*), tagamaks koodis kasutatavate objektide ühtlus. Teine alamkiht (ingl. *repository implementation*) haldab rakendustasemelt saadud päringuid, antud kihis defineeritakse repositooriumite ja teenuste alamkihi implementatsioon. Kolmandas ehk teenuste alamkihis (ingl. *services*) defineeritakse esitlustaseme suhtlus rakendustasemega, kus igale rakendustaseme API-le vastab talle omane teenus. Neljas alamkiht on administratiivkiht (ingl. *utils*), kus defineeritakse ja pannakse kokku rakendustaseme API-de URL aadressid.

Joonis 11 kirjeldab *Visual Studio Code's* esitlustaseme failipuu ülesehitust.



Joonis 11. Esitlustaseme failipuu ülesehitus.

4.1.2 Rakendustaseme arhitektuur

Projekti rakendustaseme arhitektuur on jaotatud, *Clean Architecture* põhimõtetest lähtuvalt, *Models, Repositories, Routers, Schemas ja Services* mooduliteks, kus iga moodul keskendub konkreetsele ülesandele.

Router, Service ja Repository töötavad koos, et võtta vastu päringuid esitlustasemelt, töödelda neid vastavalt ärireeglitele ja pakkuda vajalikku tagasisidet. Samuti suhtlevad nad väliselt määratletud API-de kaudu, nagu Smart-ID autentimise või elektritarbimise andmete saamiseks Elering DataHub'ist.

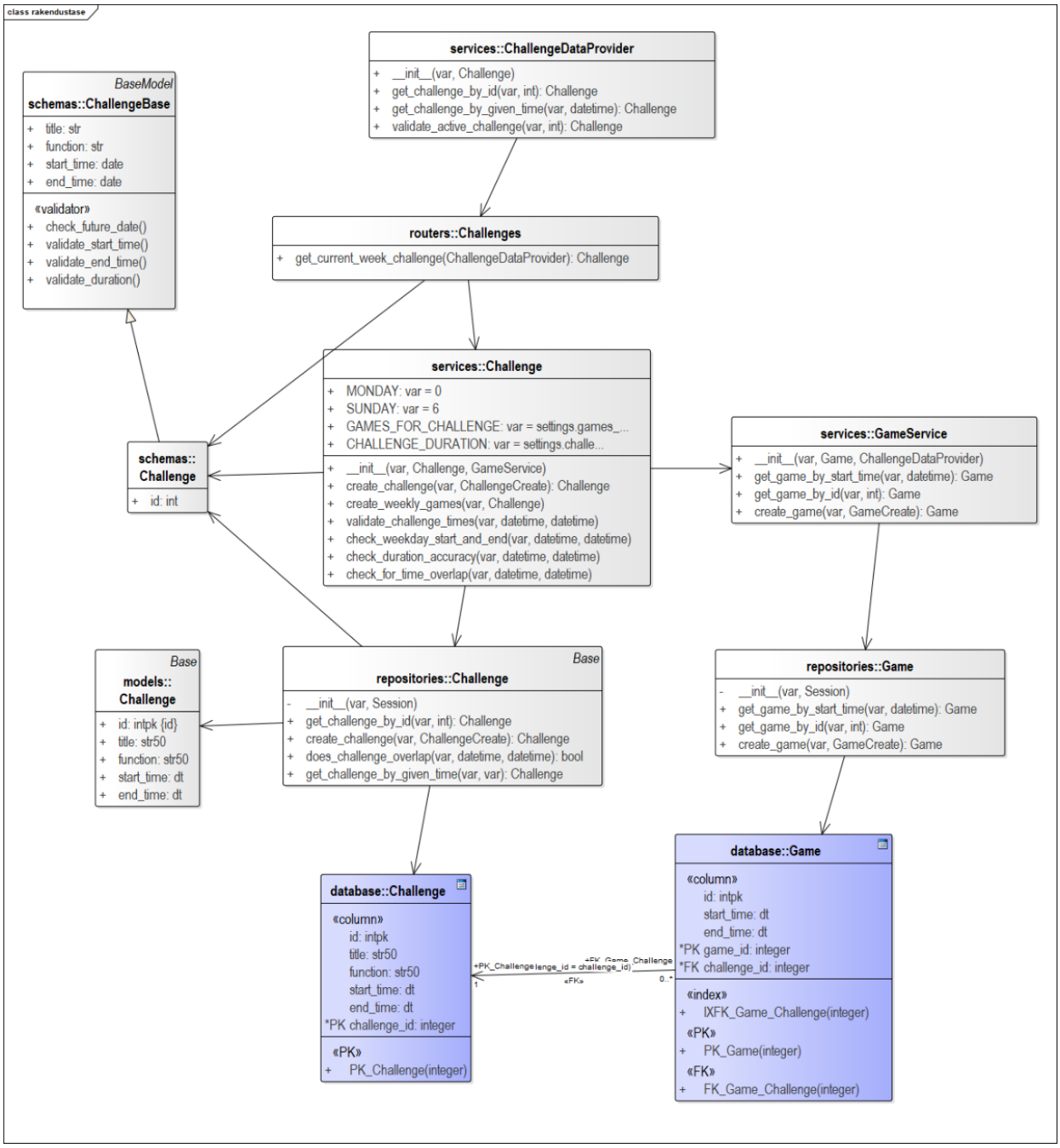
Schemade kihi eesmärk on andmete valideerimine ja normaliseerimine. Valideerimine hõlmab andmete kontrollimist eelnevalt defineeritud validaatorite alusel, veendumaks, et sisestatud või muudetud andmed on õiged ja sobivad ettekirjutatud nõuetega. Normaliseerimine tegeleb andmete teisendamisega ühtsesse ja standardiseeritud formaati, mis tagab andmete kindlas formaadis salvestamist ja töötlemist. Normaliseerimine aitab lisaks kaasa ka andmete integreeritavusele, muutes erinevatest allikatest pärit andmed omavahel kooskõlaliseks ja võrreldavaks.

Router kiht vastutab suhtluse eest esitlustasemega. See kiht võtab vastu päringuid esitlustasemelt ja suunab need edasi sobivatele kontrolleritele või teenustele rakendustasemes. Router kiht analüüsib ja interpreteerib HTTP päringuid, määrates kindlaks nende eesmärgi ja vajalikud tegevused, ning tagab, et päringud jõuavad õigesse sihtkohta rakenduse edasiseks töötlemiseks.

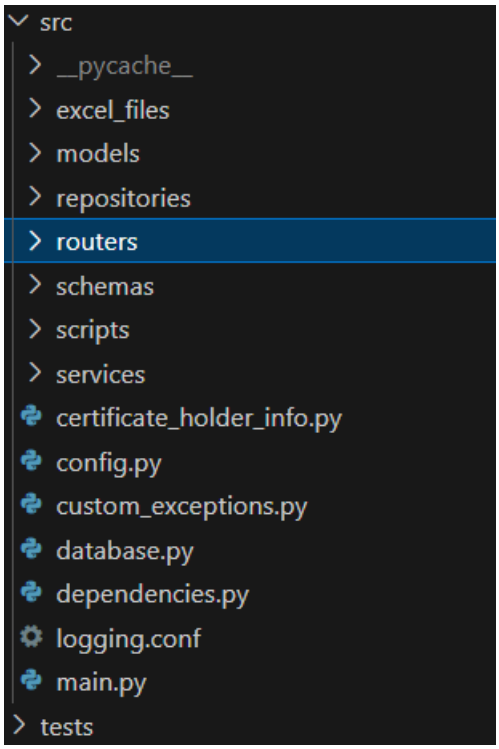
Service kiht on rakendustaseme keskne kiht, kus ärireeglid rakendatakse. See kiht haldab äriprotsesse, nagu kasutaja autentimine, andmete töötlemine ja teiste ärioperatsioonide täitmine. Teenused selles kihis hoolitsevad rakendusloogika eest, töötades tihedalt koos Repository kihiga, et hallata andmete voogu ja tagada ärireeglite järjepidev rakendamine.

Repository kiht on spetsialiseerunud andmebaasi suhtlusele. See kiht koosneb repositooriumidest, mis on vastutavad andmete hankimise, salvestamise ja hallatavuse eest. Selle kihi peamine ülesanne on abstraheerida andmebaasi juurdepääsu, pakkudes ühtset liidest andmete manipuleerimiseks. Antud kiht tagab, et *Service* kiht suudab andmeid tõhusalt pärida ja muuta, ilma et peaks muretsema konkreetsete andmebaasi-tehniliste operatsioonide pärast.

Joonis 12 kirjeldab *Challenge* arhitektuurilist toimimist. Joonis 13 kirjeldab *Visual Studio Code's* rakendustaseme failipuu ülesehitust



Joonis 12. Rakendustaseme toimimine diagrammina *Challenge* näitel.



Joonis 13. Rakendustaseme failipuu ülesehitus.

4.2 Andmetase

Projekti andmebaasi arhitektuur on üles ehitatud kasutades SQLAlchemy, mis on *Pythoni* objekt-relatsiooniline kaardistamise (ORM) raamistik. SQLAlchemy võimaldab *Pythoni* klassidel defineerida andmebaasi tabelite skeeme, kus iga klassi instants vastab tabelis ühele kirjele. Igale andmebaasi tabelile vastab üks mudeliklass, mis pärib SQLAlchemy baasklassilt enda esialgse ülesehituse ja andmebaasi tabeli kuju.

Andmebaasi skeem genereerub ja uueneb automaatselt SQLAlchemy create_all meetodi abil, mis kontrollib olemasolevate tabelite vastavust mudeliklasside definitsioonidele ja loob puuduvad tabelid. Andmebaasi konfiguratsioon, nagu ühenduse sätted ja muud parameetrid, on määratud konfiguratsioonifailis. Initsialiseerimisprotsess käivitub rakenduse käivitamisel.

Mudelid ehk andmebaasitabelid on defineeritud eraldi mudelite failis, kus igale mudelile vastab üks andmebaasitabel. Lisas 2 on välja toodud andmebaasiskeem.

4.3 Valdkonnamudelid

Mudelite alampeatükk keskendub rakenduses üldkasutatavate struktuuride kirjeldamisele, mis on äri- ja domeeniloogika aluseks. Mudelid on nii andmebaasi skeemide abstraktsioonid, kui ka rakenduses kasutatavate olemite definitsioonid, mis võimaldavad suhelda andmebaasiga *Pythoni* objektidena ja väldivad arenduse käigus objektide redefineerimist.

Tabel 5 esitab vaadeldava rakenduse mudelite definitsioonid ja mudelites defineeritud väljad.

Tabel 5. Mudelid.

Mudel	**Definitsioon**	**Väljad ja välja tüübid**
Challenge	Challenge mudel defineerib rakenduses erinevaid väljakutseid, luues võimaluse mängijate vaheliseks võistluseks	Id (*Integer*) Title (*String50*) Function (*String50*) Start_time (*DateTime*) End_time (*DateTime*)
Game	Game mudel esindab üksikut mängu, mis on seotud kindla väljakutsega (ingl. Challenge).	Id (*Integer*) Challenge_id (*Integer*) Start_time (*DateTime*) End_time (*DateTime*)
GameResult	GameResult mudel kujutab endast mängutulemuste registrit, salvestades iga mängu tulemused mängija (ingl. Player) kohta , sealhulgas saavutatud punktid ja mängu oleku.	Id (*Integer*) Player_id (*Integer*) Game_id (*Integer*) Old_rating (*Decimal*) New_rating (*Decimal*)
Leaderboard	Leaderboard mudel pakub dünaamilist ülevaadet mängijate järjestusest kehtiva väljakutse lõikes. Antud mudel kasutab GameResult mudelist pärinevat informatsiooni, et koostada edetabeleid, mis kajastavad mängijate skilli.	Id (*Integer*) Challenge_id (*Integer*) Player_id (*Integer*) Rating (*Decimal*) Position_change (*Integer*)

Mudel	**Definitsioon**	**Väljad ja välja tüübid**
MeterPoint	MeterPoint mudel ei eksisteeri andmebaasis tabelina, vaid seda kasutatakse koodiülese ühtsuse tagamiseks ning Eleringilt saadud EIC koodi (ingl. Meter Point) sidumiseks mängijaga (ingl. Player).	Id (*Integer*) Eic (*String16*)
Person	Person mudel on süsteemi isikute andmete keskne hoidla, mis kapseldab isikuga seotud põhiinformatsiooni, nagu nimi, kontaktandmed ja autentimisandmed.	Id (*Integer*) Code (*Integer*) Country_name (*String2*) First_name (*String50*) Last_name (*String50*)
Player	Player mudel esindab isikuid (ingl. Person), kes on liitunud mänguga (ingl. Game) EIC koodi (ingl. Meter Point) kaudu.	Id (*Integer*) Person_id (*Integer*) Meter_point_id (*Integer*) Challenge_id (*Integer*) Name (*String11*)
IdealConsumption	IdealConsumption mudelis defineeritakse iga mängu ideaalse elektritarbimise maht protsendina	Id (*Integer*) Game_id (*Integer*) Ideal_consumption (*Float*)

Lisaks mudelite definitsioonile defineeriti ka mudelitele välja-väljadepõhised validaatorid. Tabel 6 defineerib mudeli, tema validaatorid ja validaatorite definitsioonid.

Tabel 6. Mudelid ja nende validaatorid.

Mudel	**Validaator**	**Validaatori definitsioon**
Challenge	start_time_on_monday	Start_time välja kuupäev peab olema esmaspäev, ehk nädalapäeva indeks 1.
	end_time_on_sunday	End_time välja kuupäev peab olema pühapäev, ehk nädalapäeva indeks 7.
	duration_check	Start_time ja end_time vahe peab olema täpselt 6 päeva, 23 tundi, 59 minutit ja 59 sekundit.

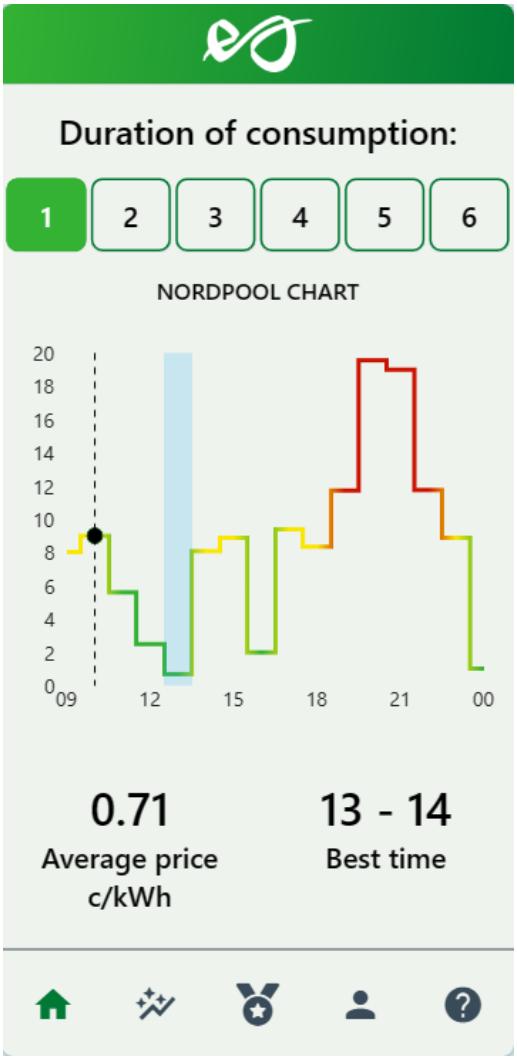
Mudel	**Validaator**	**Validaatori definitsioon**
	exclude_overlapping_challenges	Uus väljakutse luuakse vaid juhul kui antud start_time ja end_time vahemikus pole ühtegi eksisteerivat väljakutset.
Game	check_end_time_after_start_time	Mängu kuupäev end_time peab olema peale start_time kuupäeva.
	unique_start_time	Mängu algusaeg peab olema unikaalne.
GameResult	check_old_rating_not_negative	Old_rating peab olema >0.
	check_new_rating_not_negative	New_rating peab olema >=0.
	unique_player_id_game_id	Sama mängija ja mängu paari puhul ei saa uut GameResulti luua.
Leaderboard	unique_player_id_challenge_id	Sama mängija ja väljakutse paari puhul ei saa uut Leaderboardi luua.
MeterPoint	check_eic_length	Eic väli peab olema täpselt 16 tähemärki pikk.
Person	check_code_length	Code väli peab olema täpselt 11 tähemärki pikk.
	check_code_numeric	Code väli peab koosnema vaid numbritest.
	check_country_format	Country_name väli peab koosnema ainult suurtest tähemärkidest ning olema täpselt 2 tähemärki pikk.
	check_first_name_not_empty	First_name väli ei tohi olla tühi tekst.
	check_last_name_not_empty	Last_name väli ei tohi olla tühi tekst.
Player	unique_challenge_id_meter_point_id	Sama väljakutse ja elektritarbimispunkti paari puhul ei saa uut Playerit luua.

Tabelis defineeritud *check* võtmesõnaga algav validaator esitab välja või väljade kontrollpiirangut välja täitmisel, *unique* võtmesõnaga algav validaator esitab välja või väljade unikaalsuspiirangut ning *exclude* võtmesõnaga algav validaator esitab välja või väljade välistamispiirangut. Lisas 3 on välja toodud mudelite klassidiagramm.

4.4 Vaated

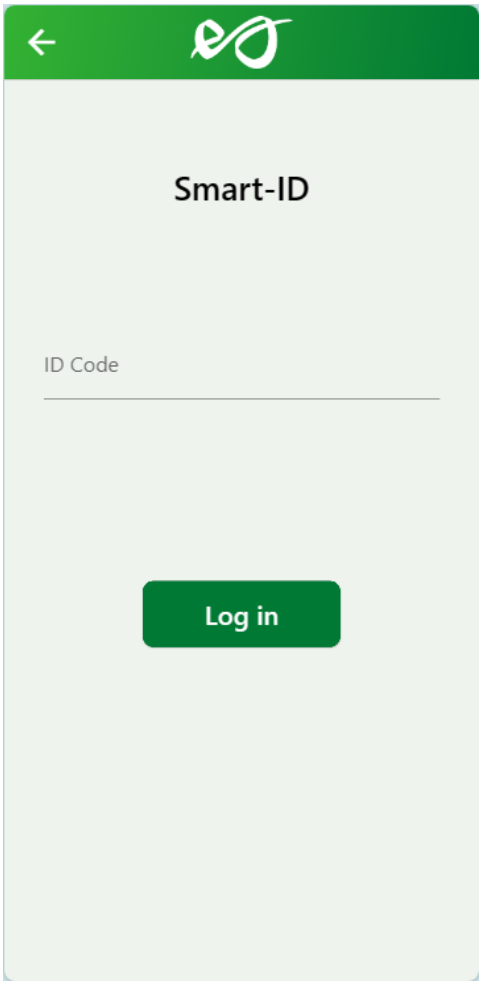
Joonisel 14 on kuvatud esimene vaade, mida kuvatakse kasutajale rakenduse käivitamisel. Joonisel 15 on koduleht, kuhu kasutaja navigeeritakse automaatselt rakenduse käivitamisel. Koduleht sisaldab endas jooksva päeva elektrihinna graafikut, elektritarbimise intervalli nuppe ning intervalli keskmist hinda ja ajalist vahemikku.

Joonis 14. Splash Screen

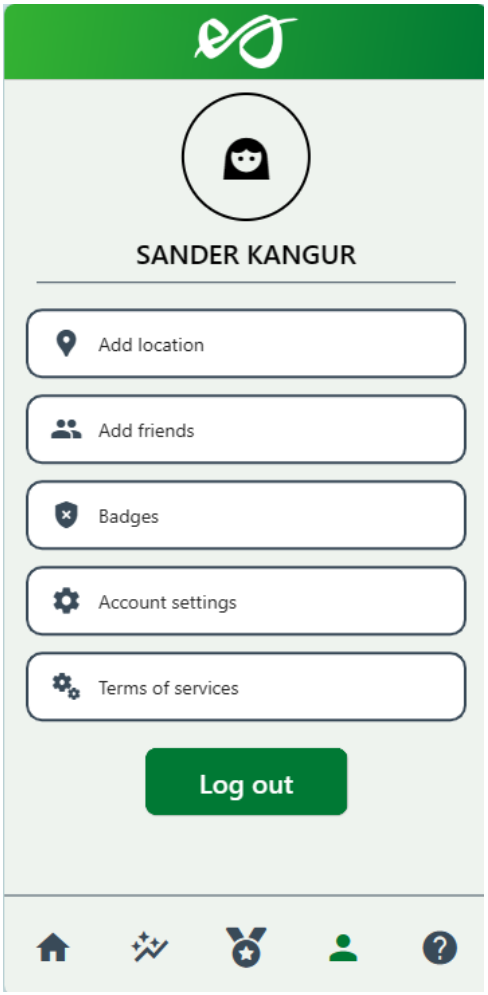


Joonis 15. Home Page Screen

Joonisel 16 on kuvatud sisselogimise vaade, kus kasutaja saab Smart-ID'ga sisse logida. Joonisel 17 on kuvatud sisselogitud kasutaja vaade sisselogimise ekraanil.



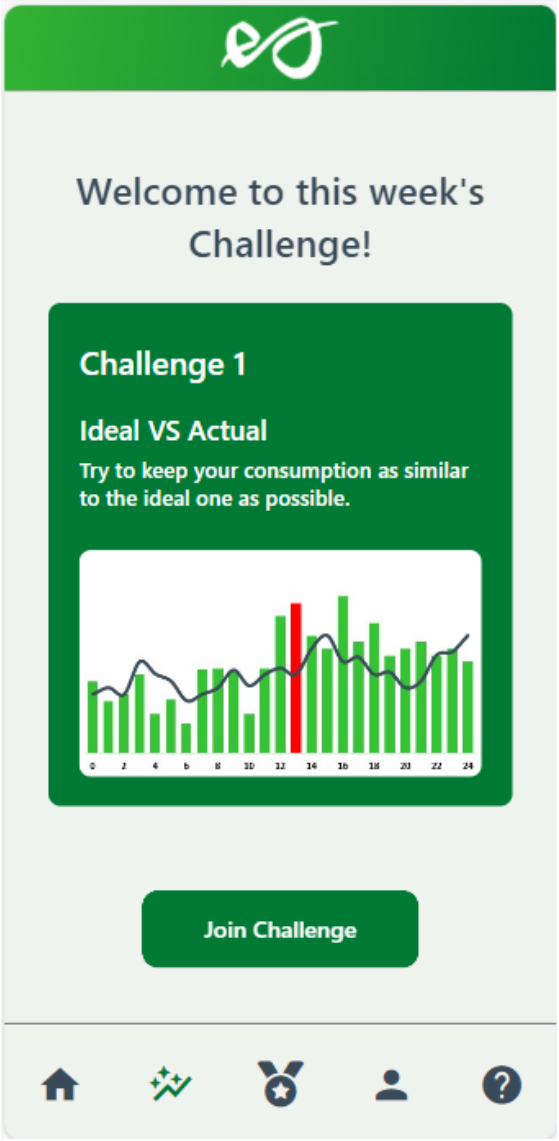
Joonis 16. Log In Screen



Joonis 17. Profile Screen

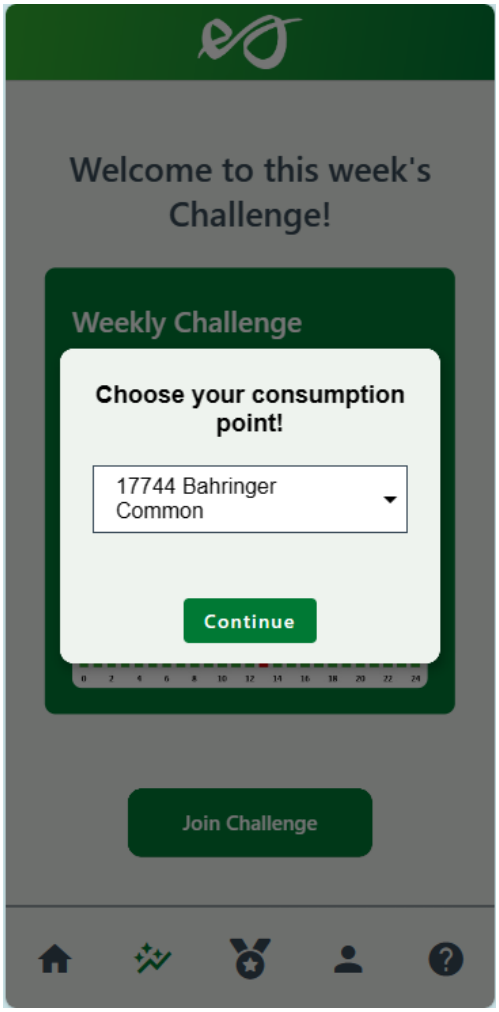
Joonisel 18 on kuvatud väljakutse vaade, kui ühtegi väljakutset ei toimu. Joonisel 19 on kuvatud Väljakutse vaade, kui kasutaja pole väljakutsega liitunud.

Joonis 18. Challenge Screen no active Challenges

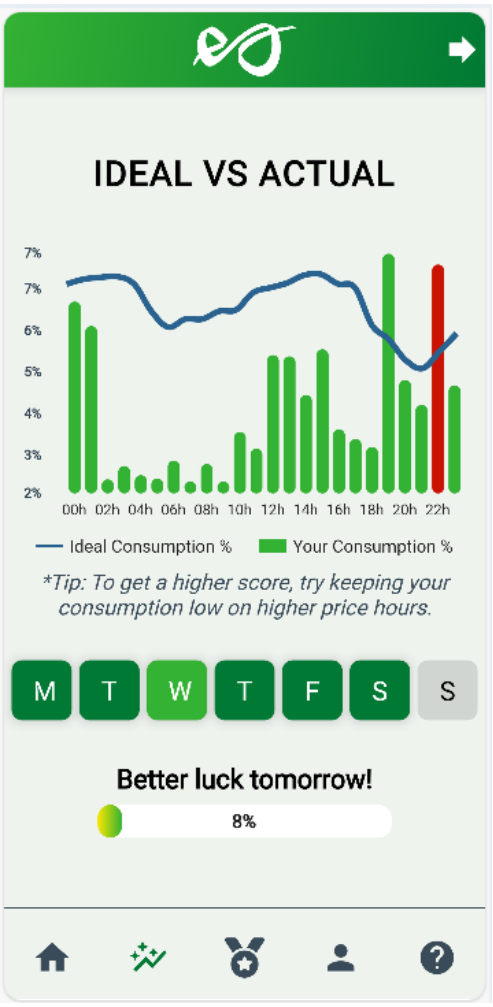


Joonis 19. Challenge Screen not Logged In

Joonisel 20 on kuvatud väljakutse vaade, kui sisselogitud kasutaja vajutab "*Join Challenge*" nupule. Joonisel 21 on väljakutsega liitunud kasutaja kolmapäevase elektritarbimise vaade.

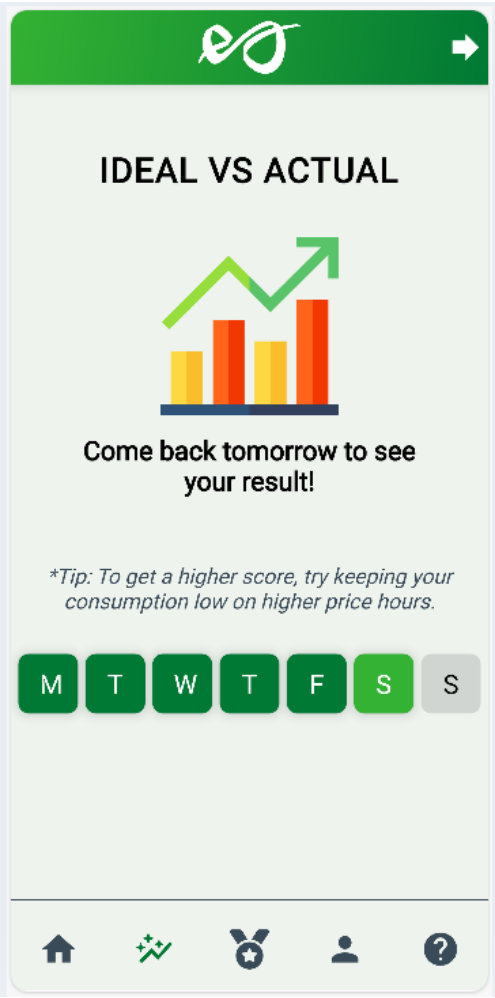


Joonis 20. Join Challenge Screen

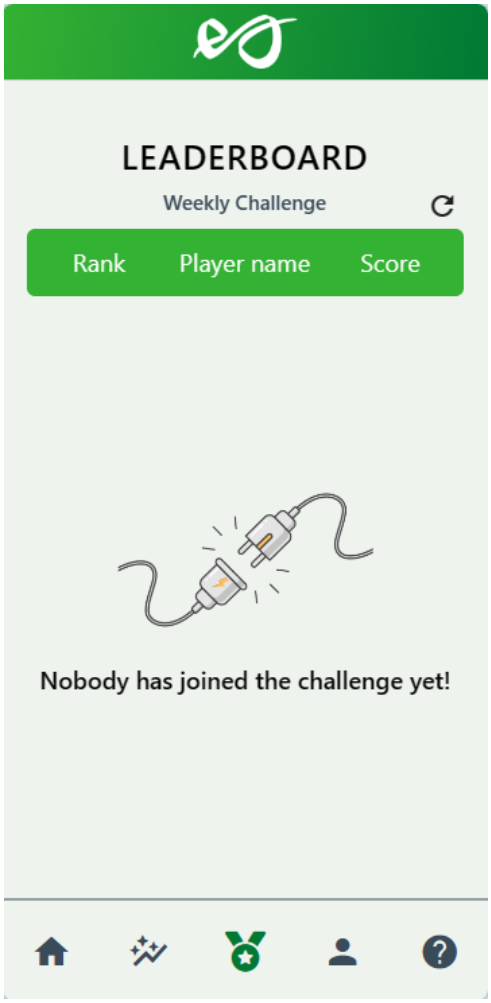


Joonis 21. Challenge Screen

Joonisel 22 on väljakutsega liitunud kasutaja, kui ta üritab jooksva päeva tulemust näha. Joonisel 23 on kasutaja edetabeli vaade, kui väljakutsega ei ole ühinenud ükski kasutaja.

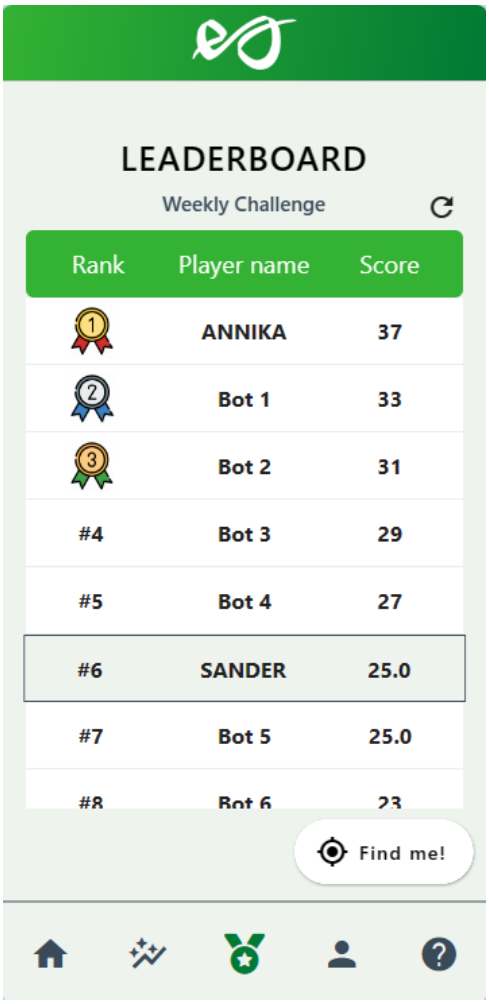


Joonis 22. Challenge Screen no data



Joonis 23. Leaderboard Screen no participants

Joonisel 24 on kuvatud kasutaja edetabeli vaade, kui ta on väljakutsega liitunud.



Joonis 24. Leaderboard Screen

4.5 Testimine

Rakendustaseme keerulisemaid funktsionaalsusi nagu *leaderboard*, *challenge* ja nende komponente testiti ühiktestide ja integratsioonitestide abil. *Challenge* puhul testiti *Challenge* olemit ühiktestidega, veendumaks olemile määratud validaatorite õigsuses. Lisaks testiti integratsioontesti näol *Challenge* põhifunktsionaalsust ehk väljakutsega liitumise funktsionaalsust. *Leaderboard* puhul testiti *Leaderboard* põhifunktsionaalsust ehk edetabeli uuendamise funktsionaalsust, mis koosnes mängija *skill* arvutusest, mängija tulemuslikkuse arvutamisest ja nende põhjal mängutulemuste loomisest. Antud funktsionaalsuse puhul testiti eelnevalt loetletud uuendamise alamfunktsionaalsusi eraldi.

Testimisraamistik loodi testskriptina eraldi jooksutamiseks ning implementeeriti *GitHub Actionsis* automaattestimise moodulina (Joonis 26).

Testimisulatuse analüüsist võib välja, et testid katavad 67% kogu rakenduse funktsionaalsusest (Joonis 25).

```
---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                                                                                                Stmts   Miss  Cover
-----------------------------------------------------------------------------------------------------------------------
/workspaces/barbados/backend/src/config.py                                                             16      0   100%
/workspaces/barbados/backend/src/custom_exceptions.py                                                   6      0   100%
/workspaces/barbados/backend/src/database.py                                                           17      7    59%
/workspaces/barbados/backend/src/dependencies.py                                                       85     44    48%
/workspaces/barbados/backend/src/models/__init__.py                                                     8      0   100%
/workspaces/barbados/backend/src/models/base.py                                                         5      0   100%
/workspaces/barbados/backend/src/models/challenge_model.py                                             17      0   100%
/workspaces/barbados/backend/src/models/game_model.py                                                  15      0   100%
/workspaces/barbados/backend/src/models/game_result_model.py                                           15      0   100%
/workspaces/barbados/backend/src/models/ideal_consumption_model.py                                     10      0   100%
/workspaces/barbados/backend/src/models/leaderboard_model.py                                           16      0   100%
/workspaces/barbados/backend/src/models/meter_point_model.py                                           11      0   100%
/workspaces/barbados/backend/src/models/person_model.py                                                14      0   100%
/workspaces/barbados/backend/src/models/player_model.py                                                18      0   100%
/workspaces/barbados/backend/src/models/types.py                                                       10      0   100%
/workspaces/barbados/backend/src/repositories/__init__.py                                               0      0   100%
/workspaces/barbados/backend/src/repositories/base_repository.py                                        9      5    44%
/workspaces/barbados/backend/src/repositories/challenge_repository.py                                  39     23    41%
/workspaces/barbados/backend/src/repositories/game_repository.py                                       20      9    55%
/workspaces/barbados/backend/src/repositories/game_result_repository.py                                19      9    53%
/workspaces/barbados/backend/src/repositories/ideal_consumption_repository.py                          14      7    50%
/workspaces/barbados/backend/src/repositories/leaderboard_repository.py                                34     19    44%
/workspaces/barbados/backend/src/repositories/meter_point_repository.py                                36     22    39%
/workspaces/barbados/backend/src/repositories/person_repository.py                                     26     15    42%
/workspaces/barbados/backend/src/repositories/player_repository.py                                     53     35    34%
/workspaces/barbados/backend/src/routers/__init__.py                                                    0      0   100%
/workspaces/barbados/backend/src/routers/elering.py                                                    35      7    80%
/workspaces/barbados/backend/src/routers/leaderboards.py                                               19      8    58%
/workspaces/barbados/backend/src/routers/prices.py                                                    124     95    23%
/workspaces/barbados/backend/src/schemas/__init__.py                                                    0      0   100%
/workspaces/barbados/backend/src/schemas/base_consumption_schema.py                                     7      0   100%
/workspaces/barbados/backend/src/schemas/challenge_schema.py                                           38      1    97%
/workspaces/barbados/backend/src/schemas/chart_data_schema.py                                          16      0   100%
/workspaces/barbados/backend/src/schemas/day_ahead_price.py                                            25      6    76%
/workspaces/barbados/backend/src/schemas/elering_schema.py                                             19      2    89%
/workspaces/barbados/backend/src/schemas/game_result_schema.py                                         12      0   100%
/workspaces/barbados/backend/src/schemas/game_schema.py                                                16      1    94%
/workspaces/barbados/backend/src/schemas/leaderboard_schema.py                                         14      0   100%
/workspaces/barbados/backend/src/schemas/meter_point_schema.py                                         12      0   100%
/workspaces/barbados/backend/src/schemas/person_schema.py                                              11      0   100%
/workspaces/barbados/backend/src/schemas/player_schema.py                                              14      0   100%
/workspaces/barbados/backend/src/services/__init__.py                                                   0      0   100%
/workspaces/barbados/backend/src/services/base_consumption_service.py                                  31      3    90%
/workspaces/barbados/backend/src/services/challenge_data_provider.py                                   29     15    48%
/workspaces/barbados/backend/src/services/challenge_service.py                                         45     23    49%
/workspaces/barbados/backend/src/services/game_result_service.py                                       27     17    37%
/workspaces/barbados/backend/src/services/game_service.py                                              40     22    45%
/workspaces/barbados/backend/src/services/ideal_consumption_service.py                                 49     28    43%
/workspaces/barbados/backend/src/services/join_challenge_provider.py                                   35     12    66%
/workspaces/barbados/backend/src/services/leaderboard_service.py                                      182     34    81%
/workspaces/barbados/backend/src/services/meter_point_service.py                                       20     11    45%
/workspaces/barbados/backend/src/services/person_service.py                                            27     16    41%
/workspaces/barbados/backend/src/services/player_service.py                                            53     37    30%
__init__.py                                                                                             0      0   100%
test_challenge.py                                                                                     101      8    92%
test_leaderboard.py                                                                                   124      0   100%
-----------------------------------------------------------------------------------------------------------------------
TOTAL                                                                                                1638    541    67%
```

Joonis 25. Rakendustaseme testimise ulatus.

Joonis 26. Rakendustaseme *Challenge* automaattesti mooduli väljavõte.

Esitlustaseme testprojektis implementeeriti väljakutse, kodulehe, väljakutsega liitumise, edetabeli, profiili ja sisselogimise lehtedele integratsioonitestid. Testimise põhifookuses oli tagada, et *widgetid* ehitavad (ingl. *render*) korrektselt. Sealhulgas testiti, et *widgetid*, mis sõltuvad teistest *widgetitest*, püsiksid funktsionaalsed ja stabiilsed. Lisaks mainitud visuaalsele ja struktuursele terviklikkusele testiti ka iga vaate *widgetite* funktsionaalsusi. See hõlmas endas nuppude funktsionaalsust ja muid kasutajapoolseid sisendeid. Testraamistik genereerus koos *Flutteri* projektiga ning oli käivitatav Visual Studio sisesest testimise *libraryst.* Lisaks implementeeriti esitlustaseme testid *GitHub Actionsis* automaattestimise moodulina (Joonis 28). Testimisulatuse graafiline vaade loodi *Flutter* lcov mooduliga.

Testimisulatuse analüüsist võib välja lugeda, et home_page ja profile_screen testimisulatus on madalaim (Joonis 27). Põhjuseks olid ajalised tegurid ja antud vaadete lihtsamoeline funktsionaalsus, mille mitte-toimimise tõenäosus on väga madal. Home_page kujutas endas rakenduse navigeerimisriba, mis lubas avada rakenduse käivitamisel ehitatud vaateid ning profile_screen keerulisem sisselogimise ja Smart-ID integratsioon, mis kujundas 50% vaate funktsionaalsust, sai testitud.

LCOV - code coverage report

Current view: top level - lib/presentation/screens
Test: lcov.info
Date: 2024-05-07 14:08:51

	Hit	Total	Coverage
Lines:	295	378	78.0 %
Functions:	0	0	-

Filename	Line Coverage		Functions	
challenge_screen.dart	77.0 %	67 / 87	-	0 / 0
home_page.dart	6.2 %	1 / 16	-	0 / 0
home_screen.dart	90.2 %	46 / 51	-	0 / 0
join_challenge_screen.dart	98.6 %	73 / 74	-	0 / 0
leaderboard_screen.dart	89.1 %	57 / 64	-	0 / 0
login_screen.dart	48.2 %	27 / 56	-	0 / 0
profile_screen.dart	80.0 %	24 / 30	-	0 / 0

Generated by: LCOV version 1.14

Joonis 27. Esitlustaseme testimise ulatus.

Test 22s

```
▶ Run flutter test -r expanded
00:00 +0: /home/runner/work/barbados/barbados/frontend/test/home_test.dart: Home page widget:  HomePageContent renders
00:00 +1: /home/runner/work/barbados/barbados/frontend/test/home_test.dart: Chart widget:  ChartContainer renders
00:00 +2: /home/runner/work/barbados/barbados/frontend/test/home_test.dart: Chart widget:  ChartContainer displays the data that is passed into it
00:02 +3: /home/runner/work/barbados/barbados/frontend/test/challenge_test.dart: Join Challenge Screen Tests:  Display Join Challenge widget when not logged in
00:02 +4: /home/runner/work/barbados/barbados/frontend/test/challenge_test.dart: Join Challenge Screen Tests:  Display pop up with the correct text when trying to join challenge when not logged in
00:03 +5: /home/runner/work/barbados/barbados/frontend/test/challenge_test.dart: Join Challenge Screen Tests:  Should transition to Challenge screen after joining if logged in
00:04 +6: /home/runner/work/barbados/barbados/frontend/test/leaderboard_test.dart: Leaderboard widget:  Leaderboard widget renders without errors
00:05 +7: /home/runner/work/barbados/barbados/frontend/test/leaderboard_test.dart: Leaderboard widget:  Leaderboard is populated by three players
00:05 +8: /home/runner/work/barbados/barbados/frontend/test/leaderboard_test.dart: Leaderboard widget functionality:  Find me button is pressed and leaderboard is scrolled
00:06 +9: /home/runner/work/barbados/barbados/frontend/test/profile_test.dart: Profile widget Profile widget renders without errors:
00:07 +10: /home/runner/work/barbados/barbados/frontend/test/profile_test.dart: Profile widget functionality:  If user is logged out then show the login screen
00:07 +11: /home/runner/work/barbados/barbados/frontend/test/profile_test.dart: Profile widget functionality:  If user is logged in then show the profile screen
00:07 +12: /home/runner/work/barbados/barbados/frontend/test/profile_test.dart: Profile widget functionality:  Press logout and see Login screen
00:07 +13: All tests passed!
```

Joonis 28. Esitlustaseme automaattestimise mooduli väljavõte.

5 Analüüs ja järeldused

Selles peatükis antakse projektile hinnang, valideeritakse tulemusi, võrreldakse loodud rakendust olemasolevate rakendustega, analüüsitakse rakendust, tuuakse välja äriline kasum ja tagasiside, kirjeldatakse arenduskeskkonna edasiarendusvõimalusi ning realiseerimata nõudeid ja tuuakse välja tiimiliikmete logid ning tehtud töö tundides.

5.1 Hinnang

Projekti eesmärgiks oli arendada välja selline telefonirakendus, mis koondaks endas kasutajale olulised elektrihinna ja selle kõikumise andmeid graafikuna, mis parandaks kasutaja elektri tarbimisharjumusi ning mis pakuks võimalust elektrihinna graafiku kuvamiseks telefoni avakuval. Sellega saadi hakkama, realiseerides kõik eelmainitud komponendid. Rakendus implementeeriti tervikuna, kasutades parimaid tarkvaraarenduslikke tavasid ning järgides agiilse ja turvalise arendamise põhitõdesid. Kuigi projekti põhieesmärk sai täidetud, on rakenduses veel ruumi edasiarenduseks ning optimeerimiseks.

Elektritarbimisharjumuste kujundamisele võib läheneda erinevalt. Projektis realiseeritud mänguline element on vaid üks võimalikest variantidest selle eesmärgi täitmiseks. Loodud lahendus on hea baas antud mängu edasiarendamiseks või lisavõimaluste juurdeimplementeerimiseks, sest mängija osaleb vaid talle meelepärastes mängudes.

5.2 Tulemuste valideerimine

Meeskonnaprojektide aine raames esitati tudengite tiimile Enefit AS tootejuhi poolt järgmised nõuded:

- Luua arenduskeskkond;
- Luua rakenduses võimalus kasutajale kuvada jooksva päeva elektrihinna graafikut;

- Luua võimalus kuvada kasutaja telefoni avakuval jooksva päeva elektrihinna graafikut *widgeti* kujul;
- Luua elektritarbimisharjumuste kujundamiseks mänguline element;
- Premeerida kasutajat mingil moel rakenduse kasutamise eest;
- Luua võimalus rakenduse kasutamiseks veebibrauseris ja telefonis;

Lisaks eelnevalt mainitud nõuetele esitati tudengitele ka lisanõuded, millele keskenduda kui aega üle jääb:

- Luua statistiline element, mis iseloomustaks kasutaja elektritarbimise optimaalsust;
- Välja mõelda mingi lisa mänguline element, mis motiveeriks kasutajat rakendust kasutama;

Loodud rakendus vastab suures mahus meeskonnaprojekti aine raames esitatud nõuetele.

Tiimitööna valmis rakendusesisene elektrihinna graafik, mänguline element koos mängu edetabeliga, arenduskeskkond kasutades DevContainersit, võimalus kasutajal teenida *badge'sid* ehk progressimärke, *backend* ühiktestid, *frontend* integratsioontestid, koodisisene dokumentatsioon ja *swagger* dokumentatsioon.

Valmimata jäi rakenduse kättesaadavaks tegemine Androidi ja iOS platvormidel ning rakenduse elektrihinna graafiku kuvamine *widgeti* kujul.

Võrreldes olemasolevate lahendustega koondati Elering DataHubi ja Enefit Eesti telefonirakenduse kasutaja elektritarbimise kuvamise funktsionaalsus mängulise elemendina, lisaks realiseeriti, sarnaselt Elektrihind.ee börsihinna tarbimissoovitajale, elektrihinna kujunemise graafik rakenduse avakuvas. Eelmainitud funktsionaalsused on olulised teadlikule elektritarbijale ning nende koondamine muudab tema elu lihtsamaks ja mugavamaks.

5.3 Rakenduse analüüs

Hetkel on elektrihinna ja –tarbimisega seotud lahendused, mis võivad huvi pakkuda rahateadlikule, säästmismeelsele kasutajale, implementeeritud eraldi ja erinevates veebikeskkondades või äppides.

Meeskonna loodud rakendus koondab endas need erinevad komponendid, lisades ühe mängulise elemendi, mida pole varasemalt implementeeritud. Selline lähenemine pakub mugavat võimalust kasutajal enda elektritarbimist optimeerida.

Rakenduses implementeeritud mäng ei soodusta pidevat kasutaja kaasamist rakenduse kasutuses, sest mänguline element kuvab kasutajale huvipakkuvat informatsiooni korra päevas. Sellest tulenevalt ei soodusta mäng aktiivset kasutaja elektri tarbimisharjumuste kujundamist, väga olulisel kohal on kasutaja enda huvi.

5.4 Äriline kasum ja tagasiside

Valminud rakenduse näol on tegemist mängulises võtmes elektritarbimise optimeerimise ja elektrihinna teadlikkust toetavate komponentide koondumisega. Antud ülesehitusega rakendust pole varasemalt loodud seega reaalelulise elektrituru mängumaastikul ollakse esimesed. Kasumlikkus ettevõttele saavutatakse kahes osas: otseseid konkurente turul ei ole, seega alternatiivsed lahendused puuduvad ning mängu mängimine on võimalik vaid kehtiva Enefit AS elektrilepingu alusel.

5.5 Edasised võimalused

Tudengite arendatud rakendus on baasrakendus ressursirohkema arenduse jaoks. Implementeeritud rakenduse komponendid on skaleeruvad, stabiilsed, mille tagab piisav testitus ning loodud arenduskeskkond võimaldab parema ja moodsama tehnoloogia implementeerimist edasises arenduses.

5.5.1 Arenduskeskkond

Hetkel on rakendusesiseselt kasutatavad moodulid *Python* 3.10 versiooni moodulid. *Python* on pidevas arenemises ning edasises arenduses peaks optimeerima rakendust selliselt, et see käiks *Pythoni* versiooniga automaatselt kaasas.

Rakenduses kasutatav DevContainers lahendus on hea variant arenduskeskkonna ühtlustamiseks tiimiliikmete vahel, kuid kuna arendatakse telefonirakendust, peaks mõtlema pigem telefonirakenduse arendamiseks loodud lahenduste peale, näiteks Android Studio. DevContainers lahendus tuleks ümber kirjutada Android Studio liidese kasutamiseks ning kogu arendus peaks liikuma Visual Studio Code liidesest Android Studio liidesesse.

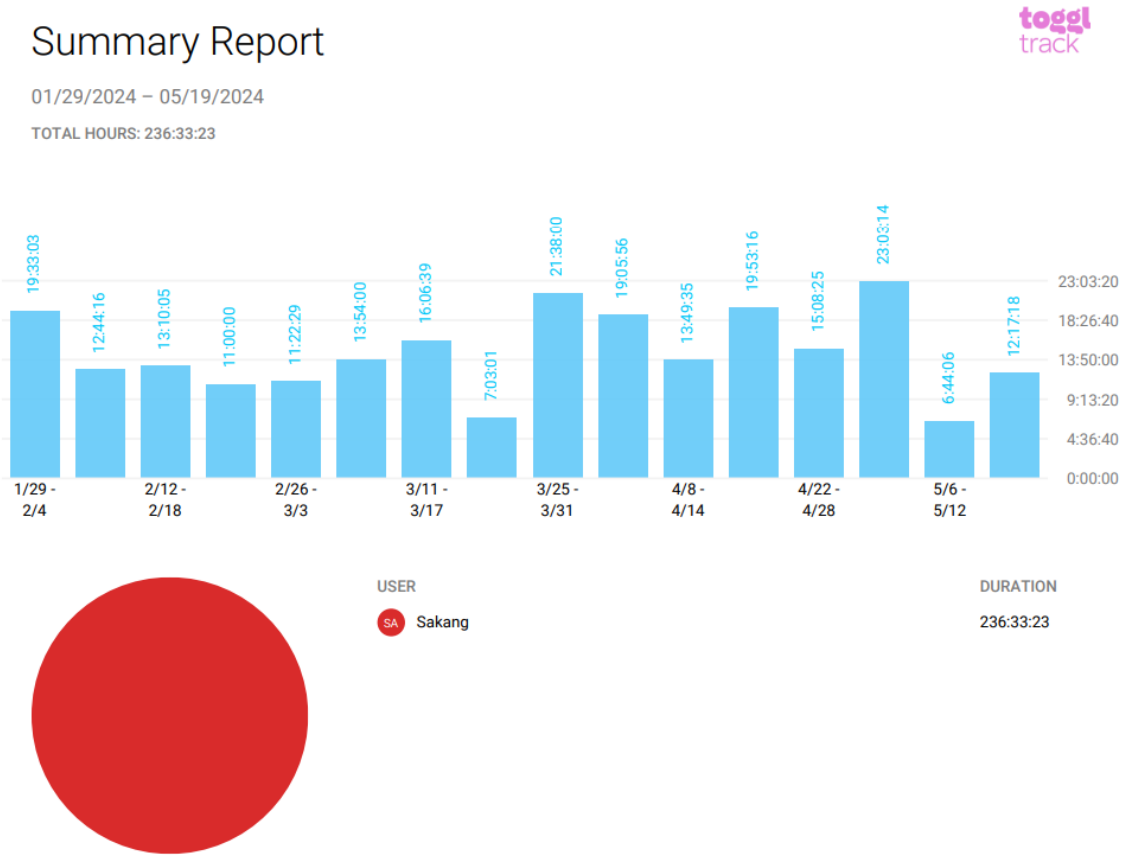
5.5.2 Realiseerimata lisanõuded

Tootejuhi lisanõuetest lähtudes saaks arendada mängu tulemuste põhjal statistilise informatsiooni arvutamist ja kasutajale kuvamist. Näiteks võib lisaks edetabeli liikumisele kuvada kasutaja optimaalsusnäitajaid, kuu- kvartali ja aastapõhiseid elektritarbimisharjumuste muutumisi arvuliselt.

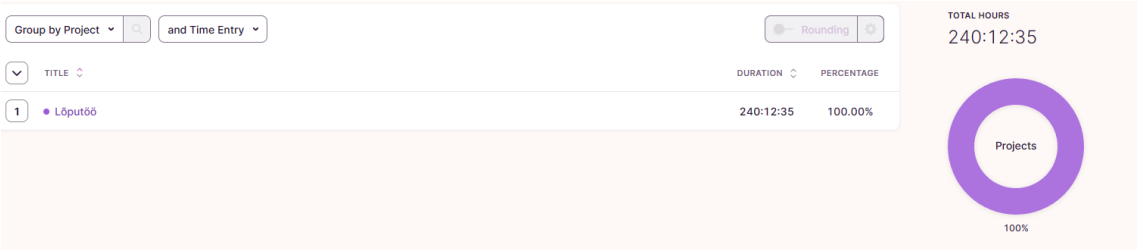
Lisaks tuleks arendada rakendusele juurde komponent või mitu komponenti, mis kaasaks kasutajat rohkem. Lisakomponendid peaks lähtuma sellest, et kuidas kasutajat premeerida ja soodustada kasutaja rakendusekasutust selliselt, et kasutaja huvi ei kaoks.

5.6 Logid ja meeskondlik hinnang

Meeskonna liikmed hindavad üksteise panust võrselt. Projektile pühendatud aeg logiti rakenduses Toggl, mille alusel logiti autorite peale kokku **236,5 + 240,2 = 476,7 tundi tööd** (Joonis 29, 30).



Joonis 29. Sander Kangur Toggl'e väljavõte



Joonis 30. Kaspar Kinko Toggl'e väljavõte.

5.6.1 Ajalogide sisuline kokkuvõte nädalate kaupa.

Tabelid 7 ja 8 kirjeldavad tiimiliikmete tööd nädalate kaupa, alates 29.01.2024 kuni 19.05.2024.

Tabel 7. Sander Kanguri logid.

Nädal	**Kirjeldus**
29.01 - 04.02	Backend parima elektritarbimise intervalli leidmise loogika arendamine Intervalli keskmise hinna loogika arendamine Intervallide loogika realiseerimine Frontendis
05.02 - 11.02	GraphData, CheapestPeriod olemite loomine ja implementatsioon frontendis dynamic muutujate asemel Vastavate olemite tagastamine backendis JSON objektide asemel
12.02 - 18.02	Elektritarbimispunkti (EIC) pop-up loomine frontendis EIC koodi struktuuri defineerimine backendis Vigaste graafiku andmete (GraphData) veahaldus (error-handling)
19.02 - 25.02	Riverpodi providerite implementeerimine, ehk andmete sisselaadimine splash-screeni ajal, ehk rakenduse optimeerimine, kiiremaks tegemine Implementeeritud providerid: getGraphDataProvider, durationProvider
26.02 - 03.03	Riverpodi providerite implementeerimine, ehk andmete sisselaadimine splash-screeni ajal, ehk rakenduse optimeerimine, kiiremaks tegemine Implementeeritud providerid: getCheapestPeriodResponse Custom_homepage_chart vaate ümbertegemine, providerite lisamine frontendis
04.03 - 10.03	Trueskill implementeerimine backendis, ehk edetabeli mängijate paiknemise loogika arendamine
11.03 - 17.03	Trueskill implementeerimine backendis, ehk edetabeli paiknemise loogika arendamine Implementeeritud funktsionaalsuse integratsioonitestide kirjutamine, et veenduda, kas loodud lahendus töötab (sest frontendis polnud vastavat vaadet veel implementeeritud)

Nädal	**Kirjeldus**
18.03 - 24.03	Kasutaja elektritarbimisinfo mockimine (nimega Elering Datahub Mock), sest puudus ligipääs reaalsetele Elering Datahub andmetele Mockitud lahenduse implementeerimine API endpointina backendis
25.03 - 31.03	Mock API endpointi modifitseerimine selliselt, et andmeid saaks lugeda excel failist (excel failid saime mentoritelt ja need sisaldasid endas nende 2023 aasta reaalelulisi elektritarbimisandmeid) Mainitud API exceli lugemisloogika modifitseerimine selliselt, et need oleks seotud reaalsete EIC punktidega (teine excel lugemise lahendus) Edetabeliloogika täiustamine kasutades lineaarregressiooni Python moodulit r2, mida kasutati mängija performance arvutamiseks (performance vajalik trueskill lahenduse arvutamiseks) Dokumentatsioon – EA mudelite, repositooriumite diagrammide loomine
01.03 - 07.04	Dokumentatsioon – EA andmebaasidiagrammide loomine Trueskill lahenduse bugide lahendamine (mis tulenesid r2 mooduli performance arvutamisest) Edetabeli ja BaseConsumption backend lahenduse testimine kasutades AAA testimismustrit Leaderboard.py refaktoorimine
08.04 - 14.04	Väljakutse teenuse testimine kasutades AAA testimismustrit Väljakutse teenuse testimine kasutades Behaviour Driven Test Python moodulit pytest-bdd
15.04 - 21.04	Frontend testimine kasutades Flutter-test moodulit Testitud vaated: koduleht, sisselogimine Bugide parandus Bug: frontend pääses backend API endpointidele ligi vaid rakenduse veebivaates

Nädal	**Kirjeldus**
22.04 - 28.04	Frontend testimine kasutades Flutter-test moodulit Testitavad vaated: edetabel
29.04 - 05.05	Frontend testimine kasutades Flutter-test moodulit Testitavad vaated: edetabel
06.05 - 12.05	Frontend testimine kasutades Flutter-test moodulit Testitavad vaated: väljakutse Frontend testide ülevaate genereerimine HTML lehena (ingl. Test-coverage report) Backend testide ümberkirjutamine, sest backend loogika muutus märkimisväärselt (väljakutse teenus + edetabeli teenus)
13.05 - 19.05	Frontend ja Backend testide implementeerimine GitHub Actionsis automaattestimise moodulitena Backend testide parandamine (koodisüntaksiline loogika muutus) Frontend testide parandamine (koodisüntaksiline loogika muutus)

Tabel 8. Kaspar Kinko logid.

Nädal	**Kirjeldus**
29.01 - 04.02	Andmebaasi tabelite loomine *Challenge*, *Player* ja *MeterPoint* jaoks. *Person* mudeli muutmine ORM-mudeliks. SQLAlchemy rakendamine *Challenge*, *Player* ja *MeterPoint* jaoks.
05.02 - 11.02	Andmebaasi tabelite loomine *Challenge*, *Player* ja *MeterPoint* jaoks. Migratsioonifaili koostamine. Migratsioonifaili automaatselt käivitumine rakenduse esimesel käivitumisel *Player* mudeli suhte muutmine *MeterPoint*'iga
12.02 - 18.02	SQLAlchemy *Backref* nimede mitteklappimise lahendamine. *Challenge*, *MeterPoint* ja *Player* skeemide loomine. Nimede konventsioonide ja trükivigade parandamine skeemidelt.
19.02 - 25.02	Rakenduseülese ajavööndi lisamine ja integreerimine *Challenge* skeemi faili. *Repository* kihi loomine *Challenge*, *MeterPoint* ja *Player* jaoks. *Naming mismatch* back_populates skeemide parandamine.
26.02 - 03.03	*Repository* kihi loomine *Challenge*, *MeterPoint* ja *Player* jaoks. Service kihi loomine *Challenge*, *MeterPoint* ja *Player* jaoks. *Get current week* backend lahenduse integreerimise alustamine.
04.03 - 10.03	Service kihi täiustamine *Challenge*, *MeterPoint* ja *Player* jaoks. *Get current week backend* lahenduse integreerimise jätkamine. *Get current week frontend* lahenduse integreerimise alustamine.
11.03 - 17.03	*Get current week* integreerimise lõpetamine. *Get current week* HTTP päringute vähendamine. Lõpuprojekti dokumentatsioon

Nädal	**Kirjeldus**
18.03 - 24.03	Lõpuprojekti dokumentatsioon *Backend* refaktoorimine. *Frontend widget* loogika ümberkirjutamine kasutades riverpodi omadusi.
25.03 - 31.03	Service kihi vigade parandamine *Challenge*, *MeterPoint* ja *Player* jaoks. Provideri loogika loomine*: Get current week, get challenge, get player, get meter point.*
01.03 - 07.04	Provideri loogika loomise alustamine, sh *get current week, get challenge, get player, get meter point.* *Challenge frontendi* põhja loomine. *Login page state management* ja *Profile* page dünaamika.
08.04 - 14.04	Provideri loogika loomine. *Challenge frontendi* põhja loomine
15.04 - 21.04	*Frontendi* lokaalne *storage*: *person, meter point, player, challenge*. *Join challenge* integreerimine
22.04 - 28.04	*Player* endpointi loomine ja selle integreerimine *frontendi*. *MeterPoint endpointi* loomine ja selle integreerimine *frontendi*. *Join challenge* integreerimine
29.04 - 05.05	*Frontendi* lokaalse *storage'i* lõpetamine: *Person, MeterPoint, Player, Challenge*. Kasutaja mänguga liitumise kontrolli *endpointi* loomine ja integreerimine. Lõpuprojekti dokumentatsiooni koostamine.
06.05 - 12.05	Lõpuprojekti dokumentatsioon
13.05 - 19.05	Lõpuprojekti dokumentatsioon

Kokkuvõte

Energia jätkusuutlik kasutamine on tänapäeva ühiskonnas muutunud üha olulisemaks. On oluline, et tarbijad oleksid teadlikud oma energiakasutuse mustritest ja mõistaksid, kuidas optimeerida oma energiatarbimist. Elektritarbimise optimeerimine on kasutajate jaoks keeruline, sest olemasolevad lahendused on hajutatud erinevate lahenduste vahel ega toeta otseselt energiasäästlike ja optimaalsete harjumuste kujunemist.

Bakalaureuse töö eesmärgiks oli luua selline telefonirakendus, mis on mugav ja kaasahaarav ning mis sisaldab endas koondlahendust olemasolevatest elektritarbimist optimeerivatest lahendustest.

Lõputöö lahendus loodi neljaliikmelises tudengite tiimis, milles järgiti Kanban agiilse arenduse metoodikat ning arendati *Clean Code Architecture* põhimõtteid järgides. Rakendus koostati kolmekihilise arhitektuurina, mis hõlmas *backend'i*, *frontend'i* ja andmebaasi, lisaks *Pytest* ja *Flutter-test* integratsioonitestide mooduleid. Loodud rakendus sisaldab endas elektritarbimise harjumusi kujundavat mängulist elementi, elektrihinna graafikut ja elektritarbimise intervallide loogikat.

Kasutatud kirjandus

[1] M. Piiriste ja K. Karjus, „Mänguelemendi mõju kasutajakogemusele Enefit AS mobiilirakenduse näitel,“ TalTech Press, Tallinn, 2024.

[2] „pgadmin.org,“ [Võrgumaterjal]. Available: https://www.pgadmin.org/docs/pgadmin4/latest/user_interface.html. [Kasutatud 13 mai 2024].

[3] GeeksforGeeks, „geeksforgeeks.org,“ 28 veebruar 2023. [Võrgumaterjal]. Available: https://www.geeksforgeeks.org/what-is-postgresql-introduction/. [Kasutatud 13 mai 2024].

[4] N. Lazarus, „dev.to,“ 23 juuli 2023. [Võrgumaterjal]. Available: https://dev.to/nilelazarus/unraveling-pgadmin-a-comprehensive-guide-to-postgresql-management-482o. [Kasutatud 13 mai 2024].

[5] „GitHub,“ 2024. [Võrgumaterjal]. Available: https://github.com/features/actions. [Kasutatud 13 mai 2024].

[6] „Docker,“ 2 mai 2024. [Võrgumaterjal]. Available: https://docs.docker.com/desktop/use-desktop/. [Kasutatud 13 mai 2024].

[7] „Postman REST Client,“ Postman, [Võrgumaterjal]. Available: https://www.postman.com/product/rest-client/. [Kasutatud 13 mai 2024].

[8] „MockAPI,“ [Võrgumaterjal]. Available: https://mockapi.io. [Kasutatud 13 mai 2024].

[9] „Pandas,“ [Võrgumaterjal]. Available: https://pandas.pydata.org/docs/reference/api/pandas.read_excel.html. [Kasutatud mai 13 2024].

[10] „Pytest,“ [Võrgumaterjal]. Available: https://docs.pytest.org/en/8.2.x/. [Kasutatud mai 13 2024].

[11] „Flutter Test,“ [Võrgumaterjal]. Available: https://docs.flutter.dev/cookbook/testing/unit/introduction. [Kasutatud mai 13 2024].

[12] „GeeksforGeeks,“ 29 september 2023. [Võrgumaterjal]. Available: https://www.geeksforgeeks.org/fastapi-introduction/. [Kasutatud 13 mai 2024].

[13] „Pydantic,“ [Võrgumaterjal]. Available: https://docs.pydantic.dev/latest/install/. [Kasutatud 13 mai 2024].

[14] „Flutter,“ [Võrgumaterjal]. Available: https://docs.pydantic.dev/latest/install/. [Kasutatud 13 mai 2024].

[15] „FlutterFlow,“ [Võrgumaterjal]. Available: https://docs.flutterflow.io/. [Kasutatud 13 mai 2024].

[16] R. Druzhinina, „Audax,“ 20 juuni 2023. [Võrgumaterjal]. Available: https://www.audax.global/blog/why-choose-python-as-a-backend-language. [Kasutatud 13 mai 2024].

[17] „YAML,“ [Võrgumaterjal]. Available: https://yaml.org/spec/1.2.2/. [Kasutatud 13 mai 2024].

[18] „JSON,“ [Võrgumaterjal]. Available: https://www.json.org/json-en.html. [Kasutatud 13 mai 2024].

[19] „Swagger,“ [Võrgumaterjal]. Available: https://swagger.io/docs/open-source-tools/swagger-editor/. [Kasutatud 13 mai 2024].

[20] „GeeksforGeeks,“ 7 september 2021. [Võrgumaterjal]. Available: https://www.geeksforgeeks.org/flutter-introduction-to-state-management-using-riverpod/. [Kasutatud 13 mai 2024].

[21] „Visual Studio Code,“ 3 november 2021. [Võrgumaterjal]. Available: https://code.visualstudio.com/docs/devcontainers/containers. [Kasutatud 13 mai 2024].

[22] „Atlassian,“ [Võrgumaterjal]. Available: https://www.atlassian.com/agile/kanban. [Kasutatud 13 mai 2024].

[23] M. Rehkopf, „Atlassian,“ [Võrgumaterjal]. Available: https://www.atlassian.com/agile/kanban/kanban-vs-scrum. [Kasutatud 13 mai 2024].

[24] L. Fitzgibbons, „techtarget.com,“ 7 oktoober 2021. [Võrgumaterjal]. Available: https://www.techtarget.com/searchsoftwarequality/definition/Behavior-driven-development-BDD. [Kasutatud 13 mai 2024].

[25] P. Gomes, „medium.com,“ 13 juuni 2018. [Võrgumaterjal]. Available: https://medium.com/@pjbgf/title-testing-code-ocd-and-the-aaa-pattern-df453975ab80. [Kasutatud 13 mai 2024].

[26] R. C. Martin, Clean Code: A Handbook of Agile Software Craftmanship, Prentice Hall, 2008.

[27] M. Fowler, Patterns of enterprise Application Architecture, Addison-Wesley Professional, 2002.

[28] M. Fowler, „martinfowler.com,“ 26 august 2015. [Võrgumaterjal]. Available: https://martinfowler.com/bliki/PresentationDomainDataLayering.html. [Kasutatud 13 mai 2024].

[29] R. C. Martin, Clean Architecture: A Craftman's Guide to Software Structure and Design, Pearson, 2017.

[30] R. C. Martin, „blog.cleancoder.com,“ 13 august 2012. [Võrgumaterjal]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html. [Kasutatud 13 mai 2024].

[31] E. AS, *Enefit Eesti telefonirakendus*, 2024.

[32] Elering, „Elering DataHub,“ [Võrgumaterjal]. Available: https://www.elering.ee/en/data-hub-information-systems. [Kasutatud 13 mai 2024].

[33] „Elektrihind,“ 1 aprill 2024. [Võrgumaterjal]. Available: https://elektrihind.ee/borsihind/. [Kasutatud 13 mai 2024].

[34] G. Silva, „GitHub,“ 7 mai 2024. [Võrgumaterjal]. Available: https://github.com/guilherme-v/flutter-clean-architecture-example. [Kasutatud 13 mai 2024].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

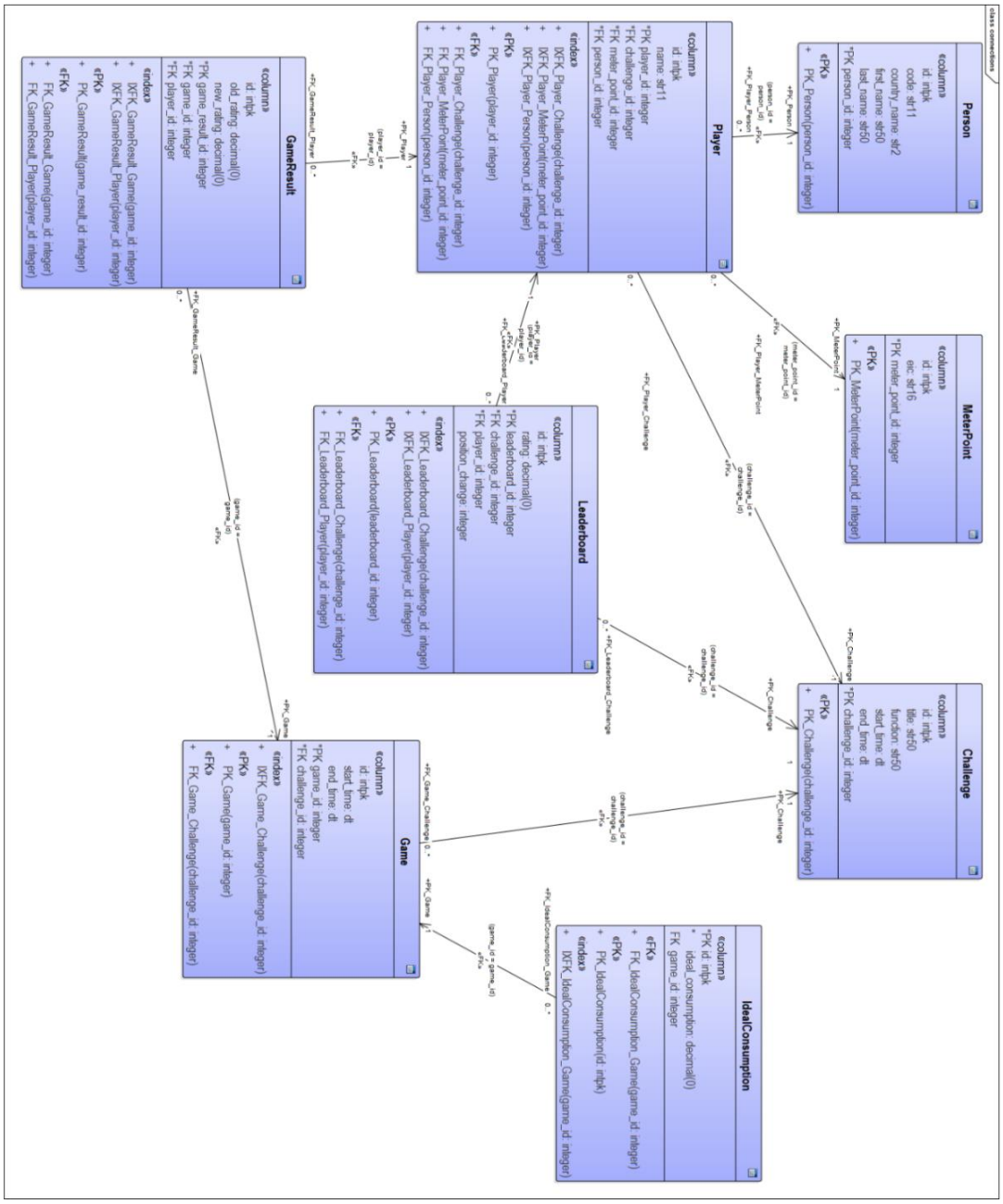
Meie, Sander Kangur ja Kaspar Kinko

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose " Telefonirakenduse arendamine elektritarbimise mustrite kujundamiseks läbi mänguliste ja informatiivsete elementide AS Enefiti näitel", mille juhendaja on Karl-Erik Karu.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Oleme teadlikud, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autoritele.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

20.05.2024

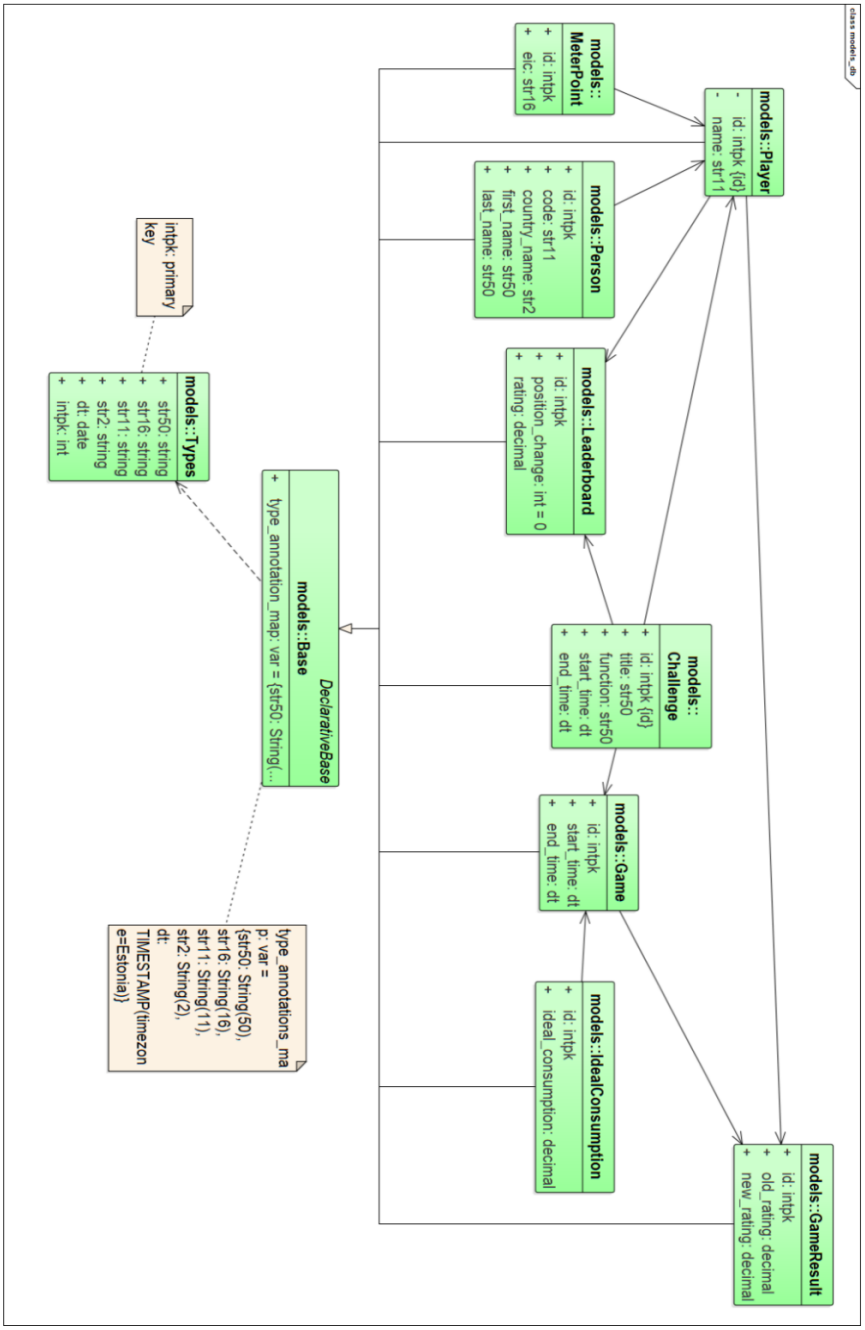
1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Andmebaasiskeem



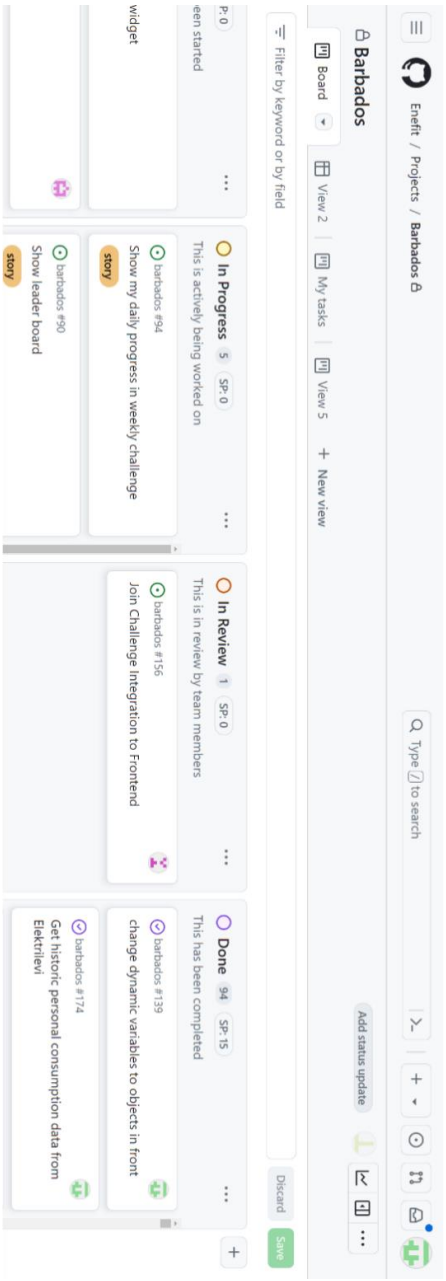
Joonis 31. Andmebaasi diagramm.

Lisa 3 – Mudelite skeem



Joonis 32. Mudelite diagramm.

Lisa 4 – Projektitiimi Kanban tahvel



Joonis 33. Projektitiimi Kanban tahvel.

Lisa 5 – Sander Kanguri eneseanalüüs

Minu panus lõputöö rakenduse arendamisesse:

- Eleringi elektrihinna kõikumise andmete ja nende põhjal optimaalsete tarbimisintervallide rakendustaseme loogika loomine ja esitlustaseme esialgne implementatsioon
- Elektritarbimispunkti *mock* loogika loomine
- Elektritarbimisandmete *mock* loogika loomine
- Väljakutse *skill* arvutuse loogika implementeerimine
- Edetabeli *performance* arvutuse loogika implementeerimine
- Testimiskeskkonna ülesseadimine
- Esitlustaseme ja rakendustaseme testimine
- Esitlustaseme ja rakendustaseme testide implementeerimine automaattestidena GitHubis

Kõige suuremaks väljakutseks oli minu jaoks väljakutse ja edetabeli *performance* ja *skill* arvutuse implementeerimine, sest lisaks õigele koodisüntaksile, tuli päriselt aru saada, kuidas antud *Pythoni* moodulid matemaatiliselt töötavad.

Lisaks äriloogika ja esitlustasemeloogika kirjutamisele sain kätt proovida ka integratsioontestimises, sain omal nahal tunda seda, kuidas testimisest lähtuvalt saab arendada selliselt koodi, et minu töö tulemuslikkus ei sõltu teiste tööst või pooleliarendustest. Varasemalt pole testimine olnud minu tugevaim külg, kuid tänu ühele Enefit AS seenior-arendajale oskan ma nüüd kirjutada struktureeritult ning loogiliselt teste ja seadistada testimiskeskkondi.

Arendamine Enefitis oli väga meeldiv kogemus, sest mul ja minu tiimikaaslastel oli esmakordselt võimalus arendada nullist üles professionaalsel tasemel rakendus. Lisaks arendustööle saime olla nii-öelda iseenda ülemused ning jagada tööd nii nagu ise

parimaks arvasime, olles seejuures osaks agiilse arenduse protsessist. Me saime teha päris asja kasutades päris tööriistu.

Loodud lahendus sisaldas endas minu jaoks võõraid tehnoloogiaid, ehk iga implementatsioon või tehnoloogia kasutuselevõtt oli minu jaoks õppemoment. Täiustasin enda teadmistepagasit kahe uue programmeerimiskeelega, uue arenduskeskkonnaga, õppisin toimima agiilses arenduskeskkonnas, õppisin implementeerima väliseid rakendusi ning oskan orienteeruda väliste rakenduste dokumentatsioonis. Lisaks tegin täpselt nii palju vigu, et erinevad veahetked või lahenduse mitte-toimimise hetked ei tekitanud enam teadmatust vaid tekitasid lahendusideid.

Lisa 6 – Kaspar Kinko eneseanalüüs

Enda põhilisteks panusteks lõputöö projektis tooksin välja:

- *Backend* arhitektuurilise struktuuri loomine
- Lokaalse vahemälu implementeerimine frontendis, mis parandas rakenduse jõudlust.
- Riverpodi *provider'ite* loomine ja nende integreerimine, mis võimaldas efektiivsemat olekute haldust ja andmevoogude juhtimist.
- *Join challenge* funktsionaalsuse integreerimine, mis lisas rakendusele interaktiivsust ja kasutajakogemust.

Projekti juures kõige huvitavam oli võimalus tegeleda *full-stack* arendusega. Eriti meeldis mulle võimalus ise arhitektuurselt üles ehitada terve rakendus algusest peale. See andis mulle tervikliku ülevaate ja kontrolli süsteemi funktsionaalsuse ning disaini üle.

Suurimad väljakutsed projekti teostamisel olid seotud *backend service layer'i* loomisega, kus tuli arvestada mitmete ärireeglitega, vältida *race condition'it* ning tegeleda veakäsitlusega. Lisaks oli väljakutseks Flutteri kasutamine *frontend'i* arendamisel, kuna see tehnoloogia oli mulle uus ja nõudis täiendavat pühendumist.

Projekti vältel õppisin kõige rohkem, kuidas luua terviklikku rakendust, valides just meie vajadustele sobivaid tehnilisi lahendusi. Samuti süvenesin *frontend'i* kujundamise ja kasutajaliideste arendamise nüanssidesse, mõistes, kui tähtis on tagada hea visuaal ja sujuv kasutajakogemus.

Lõpetuseks võib öelda, et see projekt andis mulle väärtusliku kogemuse kogu tarkvara arendusprotsessi vältel ning suurendas minu tehnilisi oskusi märkimisväärselt. Mul oli võimalus mitte ainult rakendada teoreetilisi teadmisi praktilistes situatsioonides, vaid ka omandada uusi oskusi ja lahendada keerukaid probleeme. Eelkõige hindan saadud kogemust töötades meeskonnas, kus sain vahetada ideid ja õppida meeskonnakaaslastelt

ja mentoritelt. See projekt on andnud mulle kindlust ja motivatsiooni edasiseks arenguks IT-valdkonnas.