RĪGAS VALSTS TEHNIKUMS

DATORIKAS NODAĻA

Izglītības programma: Programmēšana

KVALIFIKĀCIJAS DARBS
“Automobiļu tirgus analīzes sistēma”

Paskaidrojošais raksts 110 lpp.

Audzēknis: Emīls Jurševics

Prakses vadītājs: Līga Paudere

Nodaļas vadītājs: Normunds Barbāns

Rīga 2025

ANOTĀCIJA

Kvalifikācijas darbs "Automobiļu tirgus analīzes sistēma" ir izstrādāts, lai efektīvi pārvaldītu automašīnu tirgus datus, veicot to analīzi un vizualizāciju. Sistēmas izstrādes laikā tika izmantotas tādas tehnoloģijas kā Python 3.10, Flask 2.0.1, React 18.2.0, SQLite 3.38.0, NumPy 1.22.3 un Matplotlib 3.5.1.

Kvalifikācijas darbs ietver ievadu, uzdevuma nostādni, prasību specifikāciju, uzdevuma risināšanas līdzekļu izvēles pamatojumu, programmatūras produkta modelēšanas un projektēšanas aprakstu, datu struktūru aprakstu un nobeigumu. Ievadā ir aprakstīta problēmas aktualitāte un programmsistēmas veidošanas vajadzība. Prasību specifikācijā sasīv no ieejas un izejas informācijas, kā arī no sistēmas funkcionālajām un nefunkcionālajām prasībām.

Kvalifikācijas darbs sastāv no 110 lappušu paskaidrojošā raksta, kurā ietilpst 18 attēli, 30 tabulas un 13 pielikumi. Pielikumi satur programmas pirmkoda fragmentus.

Annotation

The qualification work "Car Market Analysis System" has been developed to efficiently manage car market data by performing analysis and visualization. During the system development, technologies such as Python 3.10, Flask 2.0.1, React 18.2.0, SQLite 3.38.0, NumPy 1.22.3, and Matplotlib 3.5.1 were utilized.

The qualification work includes an introduction, task specification, requirements specification, justification for the choice of problem-solving tools, description of software product modeling and design, data structure description, and conclusion. The introduction describes the relevance of the problem and the need for creating the software system. The requirements specification consists of input and output information, as well as functional and non-functional system requirements.

The qualification work consists of a 110 page explanatory report, which includes 18 pictures, 30 tables, and 13 appendices. The appendices contain fragments of the program source code.

SATURS

IEVADS

Izvēloties kvalifikācijas darba tēmu, mani visvairāk saistīja iespēja apvienot programmēšanas prasmes ar reālas dzīves problēmas risināšanu. Kā daudziem Latvijas iedzīvotājiem, arī man ir bijusi pieredze ar automašīnu pirkšanu un pārdošanu, un šajā procesā vienmēr izjūtu nepieciešamību pēc objektīvas informācijas par tirgus cenām un tendencēm.

Kad pats meklēju automašīnu, sapratu - nav viegli orientēties tirgū. Katru dienu parādās simtiem jaunu sludinājumu, cenas lēkā, un grūti saprast, vai konkrētā cena ir normāla vai pārāk augsta. Mūsdienu digitālajā laikmetā ikdienā tiek publicēts liels daudzums automašīnu sludinājumu dažādās platformās, tomēr šīs informācijas efektīva analīze ir izaicinājums gan automašīnu tirgotājiem, gan potenciālajiem pircējiem.

Uzskatu, ka tieši Latvijas un Baltijas tirgū šī problēma ir īpaši aktuāla, kas apstiprinājās sarunās ar radiniekiem un paziņām, kuri ikdienā nodarbojas ar auto tirdzniecību. Šobrīd internetā pieejamās sludinājumu platformas kā SS.LV, AutoPlus.lt un citas, nodrošina vienīgi sludinājumu publicēšanu un vienkāršas meklēšanas iespējas. Veicot tirgus izpēti, konstatēju, ka lai gan pastāv dažādi automašīnu cenu katalogi, trūkst visaptverošs analītisks risinājums, kas apvienotu informāciju no dažādiem avotiem. Esošās platformas piedāvā tikai sludinājumu publicēšanu. Nekādas analīzes, nekādas statistikas. Rezultātā katram pašam jāizdomā, vai konkrētā mašīna maksā normāli vai ir pārāk dārga. Šādu instrumentu trūkums bieži noved pie neefektīvas tirgus darbības, kur ne pārdevēji, ne pircēji var pārliecinoši noteikt automašīnas reālo tirgus vērtību.

Pētot pieejamos datus un lietotāju atsauksmes, atklāju, ka gan privātie automašīnu pircēji, gan pārdevēji bieži paļaujas uz subjektīviem vērtējumiem vai manuāli veic salīdzinājumu starp līdzīgiem sludinājumiem, kas ir laikietilpīgs un dažreiz neprecīzs process. Profesionāli auto tirgotāji izmanto savu iekšējo pieredzi un zināšanas par tirgu, bet arī viņiem trūkst automatizēti rīki precīzai tirgus analīzei.

Automašīnu tirgus analīzes sistēma ir vērsta uz šo problēmu risināšanu, piedāvājot platformu, kas automatizēti vāc datus no sludinājumu vietnēm, veic to statistisko analīzi un nodrošina lietotājam intuitīvu saskarni, ar kuras palīdzību var ērti meklēt, filtrēt un salīdzināt automašīnu sludinājumus. Sistēma aprēķina tirgus vidējās cenas, nosaka cenu tendences dažādos reģionos un laika periodos, kā arī piedāvā pārskatāmas vizualizācijas.

Šāda sistēma var būt noderīga plašam lietotāju lokam:

- Privātpersonām, kas vēlas iegādāties vai pārdot automašīnu un noteikt tās faktisko tirgus vērtību
- Auto tirgotājiem, kuriem nepieciešama aktuāla informācija par tirgus tendencēm

- Finanšu institūcijām un apdrošināšanas kompānijām, kam jānosaka transportlīdzekļu vērtība
- Tirgus pētniekiem un analītiķiem, kas analizē automobiļu tirgus tendences

Kvalifikācijas darba ietvaros izstrādātā sistēma aprobežojas ar datu iegūšanu no populārās sludinājumu vietnes SS.LV un demonstrēšanā tiek izmantoti dati par četrām specifiskām automašīnu markām: Tesla, Infiniti, Smart un Suzuki. Sistēmas arhitektūra ir veidota tā, lai nākotnē būtu iespējams pievienot arī citus datu avotus un paplašināt analizējamo marku klāstu.

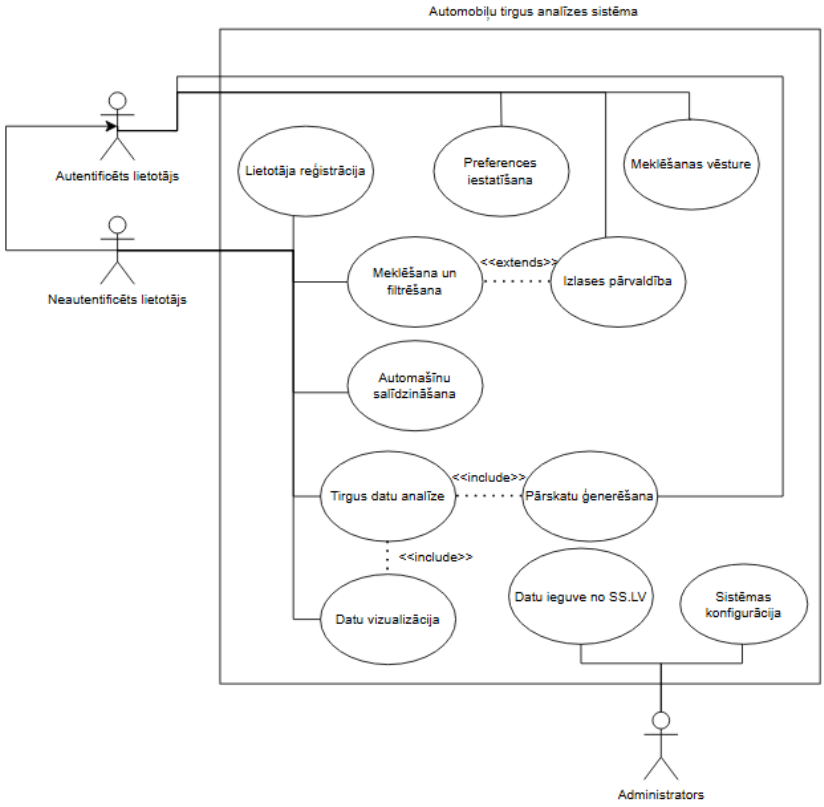
1. UZDEVUMA NOSTĀDNE

Kvalifikācijas darba uzdevums ir izveidot automašīnu tirgus analīzes sistēmu, kas iegūst, apstrādā un vizualizē automašīnu sludinājumu datus, lai palīdzētu lietotājiem pieņemt informētus lēmumus par automašīnu pirkšanu un pārdošanu. Sistēmā jānodrošina iespēja iegūt, analizēt un salīdzināt datus par automašīnu cenām, noteikt vidējo tirgus vērtību, kā arī vizualizēt tirgus tendences.

Šī problēma kļuva īpaši aktuāla COVID-19 pandēmijas laikā, kad automašīnu cenas piedzīvoja būtiskas svārstības, un tradicionālie cenu noteikšanas paņēmieni vairs nebija efektīvi. Analizējot esošās platformas, konstatēju, ka lielākā daļa no tām piedāvā tikai vienkāršu meklēšanu, bet nenodrošina dziļāku analīzi.

Lai risinātu šo problēmu, sistēmai jānodrošina vairākas būtiskas funkcionalitātes. Pirmkārt, tā automatizē datu iegūšanu no SS.LV vietnes, regulāri atjauninot informāciju par pieejamajiem sludinājumiem. Otrkārt, sistēma veic padziļinātu statistisko analīzi, aprēķinot ne tikai vidējās cenas, bet arī to sadalījumu pa reģioniem un laika periodiem. Treškārt, tā piedāvā intuitīvu vizualizāciju, kas palīdz lietotājiem ātri uztvert tirgus tendences.

Sistēmas lietotāji un to mijiedarbība ar sistēmu attēlota lietojumgadījumu diagrammā (1.1. att.), kas atspoguļo galvenās funkcionalitātes, kuras būs pieejamas dažādām lietotāju lomām.



1.1.att. Sistēmas lietošanas gadījumu diagramma.

Kā redzams lietojumgadījumu diagrammā, sistēma paredz divas galvenās lietotāju lomas: Neautentificēts lietotājs un Autentificēts lietotājs. Neautentificētam lietotājam būs pieejamas pamatfunkcionalitātes, piemēram, sludinājumu meklēšana, statistikas apskatīšana un automašīnu salīdzināšana. Savukārt autentificētam lietotājam būs pieejamas papildu funkcionalitātes, piemēram, izlases veidošana, analīzes rezultātu saglabāšana un visu pārskatu eksportēšana.

Sistēmai jābūt realizētai kā tīmekļa lietojumprogrammai ar intuitīvu lietotāja saskarni, kas nodrošina vieglu piekļuvi visām funkcionalitātēm. Tai jāspēj efektīvi darboties ar lielu datu apjomu, nodrošinot ātru meklēšanu un analīzi. Sistēmai jābūt pārnesamai un viegli uzstādāmai, lai to varētu izmantot dažādās vidēs.

Šī kvalifikācijas darba ietvaros sistēma koncentrējas uz datu analīzi četrām automašīnu markām: Tesla, Infiniti, Smart un Suzuki. Šis marku ierobežojums ir izvēlēts, lai nodrošinātu efektīvu un koncentrētu izstrādātās sistēmas demonstrēšanu un testēšanu. Sistēmas arhitektūra tomēr ir veidota tā, lai nākotnē varētu paplašināt analizējamo marku klāstu.

Sistēmai jābūt modulārai un paplašināmai, lai nākotnē būtu iespējams pievienot jaunus datu avotus, analīzes metodes un funkcionalitātes. Tāpat tai jābūt drošai, nodrošinot lietotāju datu aizsardzību un sistēmas stabilu darbību.

Automašīnu tirgus analīzes sistēma pašlaik nodrošina divas galvenās grafiskās analīzes funkcionalitātes: “Vidējās cenas pa reģioniem” un “Cenu sadalījums”. Nākotnē ir paredzēta trešās analīzes funkcionalitātes “Cenu tendences” ieviešana, kas būs atkarīga no automatizētās ikdienas datu iegūšanas implementācijas, lai uzkrātu pietiekamu vēsturisko datu apjomu. Būtisks sistēmas aspekts ir informācijas pieejamība un pārredzamība - lietotājiem jāspēj ātri iegūt skaidru priekšstatu par tirgus situāciju un cenu līmeni noteiktiem automašīnu modeļiem. Šim nolūkam sistēmai jānodrošina intuitīva un viegli lietojama saskarne ar skaidrām vizualizācijām un saprotamiem statistikas rādītājiem.

Lai sistēma būtu praktiski lietojama, tai jāspēj veikt šādus uzdevumus:

- Automatizēti iegūt sludinājumu datus no SS.LV vietnes, parādot aktuālo tirgus stāvokli
- Strukturēti saglabāt iegūtos datus, nodrošinot iespēju tos atjaunot un izmantot analīzei
- Veikt dažādus statistiskos aprēķinus, kas ļauj lietotājiem labāk izprast automobiļu tirgus situāciju
- Piedāvāt lietotājam dažādus rīkus automašīnu datu meklēšanai, salīdzināšanai un analīzei

Šobrīd sistēma tiek demonstrēta ar ierobežotu datu apjomu, koncentrējoties uz četrām konkrētām automašīnu markām, taču arhitektūras risinājumi ir izstrādāti tā, lai nodrošinātu sistēmas paplašināmību. Tas ietver gan tehnisko paplašināmību (jaunu datu avotu, analīzes metožu pievienošana), gan funkcionālo paplašināmību (jaunu funkcionalitāšu ieviešana).

Sistēmas mērķis nav tikai nodrošināt statistikas datus, bet arī palīdzēt lietotājiem pieņemt informētus lēmumus, izmantojot šos datus - vai nu automašīnas pārdošanai par adekvātu cenu, vai piemērotas automašīnas iegādei atbilstoši savām vajadzībām un budžetam. Šāda sistēma var būtiski uzlabot automobiļu tirgus efektivitāti, samazinot informācijas asimetriju starp pircējiem un pārdevējiem. Izstrādājot šo sistēmu, vispirms bija nepieciešams precīzi definēt tās prasības un funkcionalitātes. Nākamajā sadaļā detalizēti aprakstīšu sistēmas prasību specifikāciju.

2. PRASĪBU SPECIFIKĀCIJA

2.1. Ieejas informācijas apraksts

Sistēmā tiek izmantoti divi galvenie ieejas informācijas avoti: automašīnu sludinājumu dati, ko iegūst ar tīmekļa skrāpēšanas (web scraping) tehnoloģijām, un lietotāja ievadītie meklēšanas un filtrēšanas parametri.

2.1.1. Automašīnu sludinājumu dati

Automašīnu sludinājumu dati tiek iegūti automātiski no SS.LV vietnes un satur šādu informāciju:

1. Informācija par automašīnu marku:
    - Nosaukums - automašīnas ražotāja nosaukums, piemēram, “Tesla”, “Infiniti”, “Smart”, “Suzuki”. Burtu teksts ar maksimālo garumu 30 rakstzīmes.
2. Informācija par automašīnu modeli:
    - Nosaukums - automašīnas modelis, piemēram, “Model S”, “QX80”, “ForTwo”. Burtu teksts ar maksimālo garumu 30 rakstzīmes.
    - Marka - atsauce uz automašīnas marku, kurai pieder šis modelis.
3. Informācija par automašīnas reģionu:
    - Nosaukums - reģiona nosaukums, piemēram, “Rīga”, “Vidzeme”, “Kurzeme”. Burtu teksts ar maksimālo garumu 50 rakstzīmes.
    - Valsts - valsts nosaukums, piemēram, “Latvija”. Burtu teksts ar maksimālo garumu 30 rakstzīmes.
4. Informācija par automašīnu:
    - Modelis - atsauce uz automašīnas modeli.
    - Reģions - atsauce uz reģionu, kurā automašīna atrodas.
    - Gads - automašīnas izlaiduma gads. Vesels skaitlis robežās no 1900 līdz pašreizējam gadam.
    - Dzinēja tilpums - dzinēja tilpums litros. Decimālskaitlis ar precizitāti līdz 1 zīmei aiz komata, piemēram, 2.0, 1.6, 3.5. Var būt NULL elektroauto gadījumos.
    - Dzinēja tips - dzinēja veids, piemēram, “Benzīns”, “Dīzelis”, “Hibrīds”, “Elektriskais”. Burtu teksts ar maksimālo garumu 20 rakstzīmes.
    - Ātrumkārba - ātrumkārbas tips, piemēram, “Manuāla”, “Automātiska”. Burtu teksts ar maksimālo garumu 20 rakstzīmes.
    - Nobraukums - automašīnas nobraukums kilometros. Vesels skaitlis, piemēram, 150000.
    - Virsbūves tips - automašīnas virsbūves tips, piemēram, “Sedans”, “Universālis”, “Hečbeks”, “SUV”. Burtu teksts ar maksimālo garumu 20 rakstzīmes.
    - Krāsa - automašīnas krāsa, piemēram, “Melna”, “Balta”, “Sudraba”. Burtu teksts ar maksimālo garumu 20 rakstzīmes.
    - Tehniskā apskate - informācija par tehniskās apskates derīguma termiņu. Burtu teksts ar maksimālo garumu 20 rakstzīmes.
5. Informācija par datu avotu:
    - Nosaukums - datu avota nosaukums, piemēram, “SS.LV”. Burtu teksts ar maksimālo garumu 50 rakstzīmes.

- URL - datu avota tīmekļa vietnes URL, piemēram, “https://www.ss.lv/transport/cars/“. Burtu teksts ar maksimālo garumu 255 rakstzīmes.
- Valsts - valsts, kurā darbojas datu avots, piemēram, “Latvija”. Burtu teksts ar maksimālo garumu 30 rakstzīmes.
- Skrāpēšanas konfigurācija - JSON formāta teksts ar konfigurāciju skrāpēšanas procesam.
- Pēdējās skrāpēšanas datums - datums un laiks, kad pēdējo reizi veikta datu iegūšana. Datums un laiks formātā YYYY-MM-DD HH:MM:SS.

1. Informācija par sludinājumu:
   - Automašīna - atsauce uz automašīnu, par kuru ir sludinājums.
   - Avots - atsauce uz datu avotu, no kura iegūts sludinājums.
   - Ārējais ID - sludinājuma identifikators avota vietnē. Burtu teksts ar maksimālo garumu 50 rakstzīmes.
   - Cena - automašīnas cena eiro. Vesels skaitlis, piemēram, 15000, var būt NULL ja nav norādīts.
   - Sludinājuma datums - datums, kad sludinājums publicēts. Datums formātā YYYY-MM-DD.
   - Sludinājuma URL - saite uz sludinājumu avota vietnē. Burtu teksts ar maksimālo garumu 255 rakstzīmes.
   - Ir aktīvs - norāda, vai sludinājums joprojām ir aktīvs (nav izdzēsts vai pārdots). Būla vērtība (true/false).

2.1.2. Lietotāja ievadītie meklēšanas parametri

Lietotājs var ievadīt šādus meklēšanas un filtrēšanas parametrus:

1. Marka - automašīnas marka, ko lietotājs vēlas meklēt. Burtu teksts, izvēlēts no pieejamo marku saraksta.
2. Modelis - automašīnas modelis, ko lietotājs vēlas meklēt. Burtu teksts, izvēlēts no pieejamo modeļu saraksta (atkarībā no izvēlētās markas).
3. Gadu diapazons:
   - No - minimālais izlaiduma gads. Vesels skaitlis no 1990 līdz pašreizējam gadam.
   - Līdz - maksimālais izlaiduma gads. Vesels skaitlis no 1990 līdz pašreizējam gadam.
4. Cenu diapazons:
   - No - minimālā cena eiro. Vesels skaitlis no 0 līdz 1000000.
   - Līdz - maksimālā cena eiro. Vesels skaitlis no 0 līdz 1000000.
   - Dzinēja tips - izvēlēts dzinēja tips. Burtu teksts, izvēlēts no pieejamo tipu saraksta: “Benzīns”, “Dīzelis”, “Hibrīds”, “Elektriskais”, “Gāze”.
   - Ātrumkārba - izvēlēts ātrumkārbas tips. Burtu teksts, izvēlēts no pieejamo tipu saraksta: “Manuāla”, “Automātiska”, “Pusautomātiska”.

- Reģions - izvēlēts reģions. Burtu teksts, izvēlēts no pieejamo reģionu saraksta.

5. Kārtošana:
   - Kārtot pēc - kritērijs, pēc kura kārtot rezultātus. Burtu teksts, izvēlēts no pieejamo kritēriju saraksta: “cena”, “gads”, “nobraukums”.
   - Kārtošanas secība - secība, kādā kārtot rezultātus. Burtu teksts, izvēlēts no saraksta: “augošā” (ASC), “dilstošā” (DESC).

2.2. Izejas informācijas apraksts

Sistēma nodrošina dažāda veida izejas informāciju, kas ļauj lietotājam analizēt automašīnu tirgus datus un pieņemt informētus lēmumus.

2.2.1. Meklēšanas rezultāti

Meklēšanas rezultāti tiek attēloti tabulā ar šādām kolonnām:

- Marka un modelis - automašīnas marka un modelis, piemēram, “Tesla Model S”.
- Gads - automašīnas izlaiduma gads, piemēram, “2018”.
- Dzinējs - informācija par automašīnas dzinēju, piemēram, “2.0L Dīzelis”.
- Ātrumkārba - automašīnas ātrumkārbas tips, piemēram, “Automātiska”.
- Nobraukums - automašīnas nobraukums kilometros, piemēram, “150,000 km”.
- Cena - automašīnas cena eiro, piemēram, “€15,000”.
- Darbības - pogas dažādām darbībām ar sludinājumu (skatīt detaļas, pievienot izlasei, pievienot salīdzinājumam).

Papildus tabulai, meklēšanas rezultāti var tikt attēloti arī kartiņu (card) skatā, kas vizuāli pārskatāmā veidā parāda katru sludinājumu.

2.2.2. Cenu statistikas dati

Kad lietotājs meklē sludinājumus, sistēma automātiski aprēķina un attēlo šādus statistikas datus par atrasto sludinājumu cenām:

1. Vidējā cena - vidējā automašīnu cena eiro, aprēķināta no visiem atrastajiem sludinājumiem, piemēram, “€15,350”.
2. Mediānas cena - mediānas cena eiro, kas ir vidējā vērtība, ja visas cenas sakārto secībā, piemēram, “€14,800”.
3. Minimālā cena - zemākā cena starp atrastajiem sludinājumiem, piemēram, “€8,500”.
4. Maksimālā cena - augstākā cena starp atrastajiem sludinājumiem, piemēram, “€25,000”.
5. Sludinājumu skaits - kopējais atrasto sludinājumu skaits, piemēram, “42 sludinājumi”.

2.2.3. Cenu sadalījuma grafiks

Cenu sadalījuma grafiks (histogramma) rāda, cik bieži noteiktas cenu vērtības parādās sludinājumos:

1. X ass - automašīnu cenas, sadalītas intervālos, piemēram, “€5,000-€10,000”, “€10,000-€15,000”, utt.

2. Y ass - sludinājumu skaits katrā cenu intervālā.
3. Grafika nosaukums - attēlo meklēšanas kritērijus, piemēram, “Tesla Model S (2015-2020) cenu sadalījums”.

2.2.4. Cenu tendenču grafiks

Cenu tendenču grafiks (līniju grafiks) ir plānots nākotnē kā sistēmas paplašinājums un parādīs, kā mainījušās vidējās cenas laika gaitā:

1. X ass - laika periods, parasti mēneši, piemēram, “2025-01”, “2025-05”, utt.
2. Y ass - vidējā cena eiro katrā laika periodā.
3. Grafika nosaukums - attēlo meklēšanas kritērijus, piemēram, “Tesla Model S cenu tendences”.

Šī funkcionalitāte būs pieejama, kad tiks ieviesta automatizētā ikdienas datu iegūšana un būs uzkrāts pietiekams vēsturisko datu apjoms.

2.2.5. Reģionālā cenu salīdzinājuma tabula

Reģionālā cenu salīdzinājuma tabula rāda cenu statistiku dažādos reģionos:

1. Reģions - reģiona nosaukums, piemēram, “Rīga”, “Jelgava un raj.”, “Jūrmala”.
2. Vidējā cena - vidējā automašīnu cena reģionā eiro.
3. Salīdzinājums ar vidējo - procentuālā atšķirība no kopējās vidējās cenas valstī.
4. Minimālā cena - zemākā cena reģionā eiro.
5. Maksimālā cena - augstākā cena reģionā eiro.
6. Sludinājumu skaits - sludinājumu skaits reģionā.

2.2.6. Automašīnu salīdzinājuma tabula

Automašīnu salīdzinājuma tabula ļauj salīdzināt līdz 3 automašīnām vienlaikus:

1. Īpašība - salīdzināmā īpašība, piemēram, “Cena”, “Gads”, “Dzinējs”, “Ātrumkārba”, “Nobraukums”.
2. Automašīna 1 - pirmās automašīnas atbilstošās īpašības vērtība.
3. Automašīna 2 - otrās automašīnas atbilstošās īpašības vērtība.
4. Automašīna 3 - trešās automašīnas atbilstošās īpašības vērtība.

Tabulā tiek izceltas labākās vērtības (zemākā cena, jaunākais gads, mazākais nobraukums).

2.2.7. Eksporta formāti

Sistēma ļauj eksportēt datus dažādos formātos:

- CSV eksports - tabulas ar sludinājumu datiem vai salīdzinājumiem.
- PNG eksports - grafiku un diagrammu attēli.
- JSON eksports - strukturēti dati par sludinājumiem vai analīzes rezultātiem.

2.3. Funkcionālās prasības

Šajā sadaļā detalizēti aprakstītas sistēmas funkcionālās prasības, kas nosaka, kādas darbības un funkcionalitātes sistēmai jānodrošina. Definējot sistēmas funkcionālās prasības,

balstījos uz savu pieredzi kā potenciāla automašīnas pircēja, kā arī uz intervijām ar draugiem un paziņām, kas nesen ir meklējuši automašīnu. Izrādījās, ka visvairāk trūkst tieši analītisko rīku, kas palīdzētu izprast, vai konkrētā cena ir adekvāta.

FP-1: Automašīnu sludinājumu datu iegūšana

Sistēmai jāspēj automātiski iegūt datus par automašīnu sludinājumiem no SS.LVvietnes.

Detalizācija:

- Sistēma spēj iegūt jaunākos sludinājumus no SS.LV vietnes, izmantojot tīmekļa skrāpēšanas (web scraping) tehnoloģijas.
- Sistēma iegūst detalizētu informāciju par katru automašīnu, tostarp marku, modeli, gadu, dzinēja tipu, ātrumkārbu, nobraukumu, cenu un citus parametrus.
- Sistēma saglabā iegūtos datus datu bāzē, lai tos varētu izmantot analīzei un meklēšanai.
- Sistēma reģistrē datu iegūšanas datumu un laiku, lai varētu sekot līdzi sludinājumu aktualitātei.
- Sistēma atzīmē sludinājumus kā neaktīvus, ja tie vairs nav pieejami avota vietnē.
- Sistēma apstrādā un normalizē iegūtos datus, lai nodrošinātu to konsistenci un kvalitāti.
- Lietotājam ar administratora tiesībām ir iespēja manuāli iniciēt datu iegūšanas procesu.

FP-2: Automašīnu datu meklēšana un filtrēšana

Šī funkcionalitāte ir sistēmas sirds - bez efektīvas meklēšanas pārējās iespējas zaudē savu vērtību. Testējot ar draugiem, atklāju, ka lietotāji visbiežāk sāk ar plašu meklēšanu (piemēram, “Tesla, jebkurš modelis”) un pakāpeniski šaurina kritērijus.

Detalizācija:

- Sistēma nodrošina meklēšanu pēc automašīnas markas un modeļa, izmantojot atlases rīkus ar pilnīgu marku un modeļu sarakstu.
- Sistēma ļauj filtrēt rezultātus pēc izlaiduma gada diapazona, izmantojot slīdni vai ievades laukus.
- Sistēma ļauj filtrēt rezultātus pēc cenu diapazona, izmantojot slīdni vai ievades laukus.
- Sistēma ļauj filtrēt rezultātus pēc dzinēja tipa, ātrumkārbas tipa un reģiona, izmantojot atlases rīkus.
- Sistēma nodrošina rezultātu kārtošanu pēc dažādiem kritērijiem (cena, gads, nobraukums) un kārtošanas secības (augoša, dilstoša).
- Sistēma attēlo meklēšanas rezultātus tabulā vai kartiņu skatā, atkarībā no lietotāja izvēles.
- Sistēma nodrošina paginmāciju meklēšanas rezultātiem, ja to skaits pārsniedz noteiktu slieksni (10 rezultāti lapā).
- Meklēšanas rezultāti tiek atjaunināti reāllaika režīmā, kad lietotājs maina meklēšanas parametrus.
- Sistēma saglabā pēdējos meklēšanas parametrus lietotāja sesijas laikā.

Īpaša uzmanība pievērsta meklēšanas ātrumam - neviens negrib gaidīt vairāk par 3 sekundēm, lai redzētu rezultātus. Šis ātruma ierobežojums ietekmēja datubāzes indeksu dizainu un vaicājumu optimizāciju.

FP-3: Statistiskā analīze un datu vizualizācija

Sistēmai jāspēj veikt statistisko analīzi par automašīnu cenām un vizualizēt rezultātus, lai lietotājs varētu izprast tirgus situāciju.

Detalizācija:

- Sistēma aprēķina un attēlo galvenos statistikas rādītājus (vidējo cenu, mediānu, minimālo un maksimālo cenu) izvēlētajiem automašīnas modeļiem un filtriem.
- Sistēma vizualizē cenu sadalījumu, izmantojot histogrammas, kas parāda, cik bieži noteiktas cenu vērtības parādās sludinājumos. Šī vizualizācija palīdz identificēt cenu klasterus un izprast, vai konkrētā cena ir tipiska vai izņēmums.
- Sistēma salīdzina cenas dažādos reģionos un attēlo šo informāciju gan grafiskā veidā (stabiņu diagramma), gan tabulā, izceļot reģionus ar zemākām un augstākām cenām.
- Sistēma nodrošina interaktīvu statistikas paneli, kas dinamiski atjaunināsies, kad lietotājs maina meklēšanas parametrus, ļaujot reāllaika režīmā izpētīt dažādus tirgus segmentus.
- Sistēma ģenerē un eksportē grafikus PNG formātā, lai lietotājs varētu saglabāt analīzes rezultātus ārpus sistēmas.

FP-4: Datu vizualizācija

Sistemātiska datu analīze bieži vien noved pie sarežģītām skaitļu kopām, kuras grūti izprast bez piemērotas vizuālās reprezentācijas. Tāpēc esmu īpašu uzmanību pievērsis tam, lai lietotājiem būtu pieejami saprotami un informatīvi grafiki, kas palīdz ātrāk saskatīt tendences un veikt secinājumus.

Detalizācija:

- Cenu sadalījuma grafikā (histogrammā) lietotāji var redzēt, kāds ir automašīnu skaits dažādās cenu kategorijās. Šis ir īpaši noderīgi, lai noteiktu, vai konkrētā modeļa cenas tirgū veido izteiktus klasterus (piemēram, jaunāki modeļi augstākā cenu kategorijā un vecāki modeļi zemākā) vai ir vienmērīgi izkliedētas.
- Reģionālās atšķirības ir bieži ignorēts, bet ļoti svarīgs faktors auto tirgū. Izveidotie stabiņu grafiki ļauj lietotājam uzreiz pamanīt, ka, piemēram, Rīgā tās pašas automašīnas maksā vidēji par 10-15% dārgāk nekā mazpilsētās. Šāda informācija var palīdzēt pieņemt lēmumu par gatavību doties iegādāties auto citā reģionā.
- Interaktivitāte ir būtisks aspekts - lietotājs var novietot kursoru virs jebkura grafika elementa un iegūt detalizētāku informāciju (piemēram, precīzu automašīnu skaitu konkrētajā cenu intervālā vai vidējo nobraukumu attiecīgajam modelim). Tāpat

grafiki dinamiski atjauninās, kad lietotājs maina meklēšanas kritērijus, tādējādi nodrošinot tūlītēju atgriezenisko saiti.

- Lai nodrošinātu iespēju lietot šos datus arī ārpus platformas, esmu ieviesis funkcionalitāti, kas ļauj lietotājam lejupielādēt jebkuru grafiku augstas kvalitātes PNG formātā, piemēram, lai to iekļautu prezentācijā vai pārskatos.
- Vizualizācijas ir veidotas, izmantojot Matplotlib un Seaborn bibliotēkas, kas ļauj radīt profesionāla izskata grafikus. Īpaša uzmanība tika pievērsta krāsu paletei un marķējumiem, lai grafiki būtu viegli uztverami un estētiski pievilcīgi.

FP-5: Automašīnu salīdzināšana

Sistēmai jānodrošina iespēja salīdzināt līdz 3 automašīnas vienlaikus, lai atvieglotu lēmumu pieņemšanu pirms pirkšana.

Detalizācija:

- Sistēma ļauj lietotājam pievienot līdz 3 automašīnām salīdzinājumam no meklēšanas rezultātiem vai analīzes sadaļas.
- Sistēma attēlo automašīnu salīdzinājumu tabulā, kur katrā rindā ir viena īpašība (cena, gads, dzinējs, ātrumkārba, nobraukums, reģions) un katrā kolonnā ir viena automašīna.
- Sistēma vizuāli izceļ labākās vērtības (zemākā cena, jaunākais gads, mazākais nobraukums) salīdzinājuma tabulā, izmantojot krāsu kodēšanu (zaļa krāsa labākajām vērtībām).
- Sistēma ļauj lietotājam viegli pārvaldīt salīdzinājumu - pievienot jaunas automašīnas, noņemt esošās, vai notīrīt visu salīdzinājumu.
- Sistēma saglabā salīdzinājumu lietotāja sesijas laikā, pat ja lietotājs pāriet uz citām sistēmas sadaļām un atgriežas atpakaļ. Sistēma autentificētiem lietotājiem ļauj eksportēt salīdzinājuma rezultātus JSON formātā turpmākai analīzei.

FP-6: Lietotāju autentifikācija un profili

Sistēmai jānodrošina lietotāju autentifikācija un profilu veidošana, lai saglabātu lietotāju preferences un ļautu izmantot papildu funkcionalitātes.

Detalizācija:

- Sistēma ļauj lietotājam reģistrēties, ievadot e-pastu, lietotājvārdu un paroli.
- Sistēma ļauj lietotājam pieslēgties, izmantojot e-pastu un paroli.
- Sistēma nodrošina paroles atjaunošanas funkcionalitāti.
- Sistēma ļauj autentificētam lietotājam pievienot automašīnas izlasei.
- Sistēma ļauj autentificētam lietotājam apskatīt un pārvaldīt savu izlasi.
- Sistēma saglabā autentificēta lietotāja meklēšanas vēsturi.
- Sistēma ļauj lietotājam konfigurēt savas preferences (tumšais režīms, valoda, paziņojumu iestatījumi).
- Sistēma ļauj lietotājam eksportēt grafikus.

FP-7: Datu eksportēšana

Sistēmai jāspēj eksportēt datus dažādos formātos, lai lietotājs varētu tos izmantot ārpus sistēmas.

Detalizācija:

- Sistēma ļauj eksportēt meklēšanas rezultātus CSV formātā.
- Sistēma ļauj eksportēt grafikus un diagrammas PNG formātā.
- Sistēma ļauj eksportēt automašīnu salīdzinājumus JSON formātā.
- Eksportētie dati satur informāciju par eksporta datumu un izmantotajiem filtriem.

FP-8: Automašīnas detalizēta informācija

Sistēmai jānodrošina iespēja apskatīt detalizētu informāciju par konkrētu automašīnu.

Detalizācija:

- Sistēma ļauj lietotājam apskatīt detalizētu informāciju par automašīnu, klikšķinot uz sludinājuma meklēšanas rezultātos.
- Sistēma attēlo visu pieejamo informāciju par automašīnu, tostarp marku, modeli, gadu, dzinēja tipu, ātrumkārbu, nobraukumu, krāsu, virsbūves tipu, tehnisko apskati un cenu.
- Sistēma attēlo sludinājuma publicēšanas datumu un avotu.
- Sistēma nodrošina saiti uz oriģinālo sludinājumu avota vietnē (SS.LV).
- Sistēma ļauj lietotājam pievienot automašīnu izlasei vai salīdzinājumam tieši no detalizētās informācijas skata.
- Ja pieejama, sistēma attēlo automašīnas attēlu no sludinājuma.
- Sistēma attēlo sludinājuma aprakstu un aprīkojuma sarakstu, ja šī informācija ir pieejama.

2.4. Nefunkcionālās prasības

Šajā sadaļā aprakstītas sistēmas nefunkcionālās prasības, kas nosaka, kādiem kvalitātes kritērijiem sistēmai jāatbilst.

2.4.1. Veiktspēja un efektivitāte

Apstrādes apjoms:

- Sistēmai jāspēj apstrādāt līdz 200 sludinājumu vienā datu iegūšanas sesijā.
- Sistēmai jāspēj uzglabāt vismaz 1000 sludinājumu vēsturi.
- Kešošana ar localStorage priekš lietotāja preference.

Atbildes laiks:

- Meklēšanas pieprasījumiem jāizpildās ne vairāk kā 3 sekunžu laikā.
- Statistikas aprēķiniem jāizpildās ne vairāk kā 5 sekunžu laikā.
- Grafiku ģenerēšanai jānotiek ne vairāk kā 3 sekunžu laikā.
- Lapas ielādes laikam jābūt mazākam par 2 sekundēm.

Datu bāzes optimizācija:

- Datu bāzei jābūt optimizētai ātrai datu izgūšanai, izmantojot indeksus kritiskajām kolonnām.
- Kompleksām vaicājumu operācijām jāizmanto datu bāzes skati (views) vai saglabātās procedūras.

Kešošana:

- Sistēmai jāizmanto kešošanas mehānismi, lai samazinātu datu bāzes pieprasījumu skaitu bieži izmantotai informācijai.
- Grafiku un statistikas dati jākešo vismaz 5 minūtes, lai izvairītos no atkārtotas aprēķināšanas.

2.4.2. Drošība

Lietotāju autentifikācija:

- Lietotāju parolēm jābūt jāšifrētām, izmantojot drošus hašēšanas algoritmus.
- Pamata autentifikācija ar JWT tokeniem, 24 stundu sesijas.

Datu šifrēšana:

- Lietotāju sensitīvajiem datiem (e-pasti, paroles) jābūt šifrētiem datu bāzē.
- Datu pārraidei starp klientu un serveri jāizmanto HTTPS protokols.

Piekļuves kontrole:

- Sistēmai jānodrošina, ka lietotāji var piekļūt tikai tām funkcionalitātēm, kurām viņiem ir atbilstošas tiesības.
- Administratora funkcionalitātēm jābūt pieejamām tikai autorizētiem administratoriem.

Auditācija:

- Sistēmai jāreģistrē visi būtiskie notikumi, tostarp pieslēgšanās mēģinājumi, konfigurācijas izmaiņas un datu skrāpēšanas sesijas.
- Auditācijas žurnāliem jāsatur informācija par notikuma datumu, laiku, lietotāju un veikto darbību.

Datu aizsardzība:

- Sistēmai jānodrošina, ka lietotāju dati netiek izpausti nesankcionētām personām.
- Sistēmai jāatbilst datu aizsardzības normatīvo aktu prasībām.

2.4.3. Lietojamība

Lietotāja saskarne:

- Sistēmas saskarnei jābūt intuitīvai un viegli izmantojamai bez speciālas apmācības.
- Lietotāja saskarnei jābūt konsistentai visā sistēmā, izmantojot vienotu krāsu shēmu, fontus un kontrolelementu stilu.
- Sistēmai jānodrošina skaidra navigācija un struktūra, lai lietotājs vienmēr zinātu, kur atrodas.

Responsivitāte:

- Sistēmai jābūt pilnībā responsīvai, pielāgojoties dažādu izmēru ekrāniem (datoru, planšetdatoru, mobilo tālruņu).
- Mobilajām ierīcēm jānodrošina optimizēta saskarne ar lielākām pogām un kontrolelementiem.

Pieejamība:

- Sistēmai jāatbilst WCAG 2.1 AA līmeņa pieejamības vadlīnijām.
- Sistēmai jābūt pieejamai arī lietotājiem ar invaliditāti, nodrošinot alternatīvu saturu attēliem, krāsu kontrastu un tastatūras navigāciju.

Lokalizācija:

- Sistēmai jābūt pieejamai latviešu valodā.
- Sistēmai jānodrošina iespēja pārslēgties starp valodām, ja tiek ieviests atbalsts citām valodām.

Kļūdu apstrāde:

- Sistēmai jāparāda skaidri un saprotami kļūdu paziņojumi, ja notiek kļūda.
- Sistēmai jānodrošina iespēja atgriezties pie iepriekšējā stāvokļa pēc kļūdas.
- Ievades lauki jāvalidē reāllaika režīmā, lai novērstu kļūdainu datu ievadi.

Palīdzība:

- Sistēmai jānodrošina palīdzības funkcionalitāte, kas izskaidro, kā izmantot dažādas funkcijas.
- Sistēmai jānodrošina rīku padomus (tooltips) sarežģītākiem kontrolelementiem.

2.4.4. Pārnesamība

Tīmekļa pārlūkprogrammas:

- Sistēmai jādarbojas vismaz četrās populārākajās pārlūkprogrammās: Google Chrome, Mozilla Firefox, Safari, Microsoft Edge (jaunākajās versijās).

Operētājsistēmas:

- Sistēmas serverim jādarbojas uz vismaz trīs populārākajām operētājsistēmām: Windows, Linux, macOS.
- Klienta daļai jādarbojas uz jebkuras operētājsistēmas, kas atbalsta minētās tīmekļa pārlūkprogrammas.

Datu bāzes:

- Sistēmai jāspēj darboties ar dažādām datu bāzu pārvaldības sistēmām: SQLite, MySQL, PostgreSQL.
- Datu bāzes struktūrai jābūt dokumentētai, lai atvieglotu migrāciju starp dažādām datu bāzu sistēmām.

Instalācija:

- Sistēmai jābūt viegli instalējamai, izmantojot minimālu konfigurāciju.
- Sistēmai jānodrošina skaidras instalācijas instrukcijas dažādām platformām.

Atkarības:

- Sistēmai jāizmanto plaši izplatītas un labi uzturētas bibliotēkas un ietvari.
- Sistēmai jāspecificē visas ārējās atkarības un to versijas, lai nodrošinātu konsistentu darbību dažādās vidēs.

2.4.5. Uzturamība

Koda kvalitāte:

- Kodam jābūt labi strukturētam, sekojot programmēšanas labākajām praksēm.
- Kodam jābūt dokumentētam, izmantojot koda komentārus un nesarežģītiem nosaukumiem..
- Kodam jāatbilst pieņemtajiem kodēšanas standartiem (PEP 8 Python kodam, ESLint JavaScript kodam).

Modulārā arhitektūra:

- Sistēmai jābūt veidotai ar modulāru arhitektūru, kas ļauj viegli pievienot, modificēt vai noņemt atsevišķas komponentes.
- Sistēmai jāizmanto atkarību injekcijas principi, lai samazinātu komponentu ciešo saistību.

Testējamība:

- Sistēmai jābūt veidotai tā, lai to varētu viegli testēt.
- Jābūt iespējai izpildīt automatizētus testus.

Versiju kontrole:

- Sistēmas kodam jābūt glabātam versiju kontroles sistēmā (Git).
- Katrai būtiskai izmaiņai jābūt dokumentētai ar commit ziņojumu.

Dokumentācija:

- Sistēmai jānodrošina detalizēta tehniskā dokumentācija, kas apraksta arhitektūru, klašu struktūru un galvenās metodes.
- Sistēmai jānodrošina lietotāja dokumentācija, kas apraksta sistēmas funkcionalitātes.

2.4.6. Savietojamība

Tīmekļa standarti:

- Sistēmai jāatbilst HTML5, CSS3 un ECMAScript 6 (ES6) standartiem.
- Sistēmai jāpārbauda savietojamība ar dažādām pārlūkprogrammu versijām.

API:

- Sistēmas API jāimplementē, izmantojot RESTful principus.
- API jādokumentē, izmantojot OpenAPI vai līdzīgu specifikāciju.

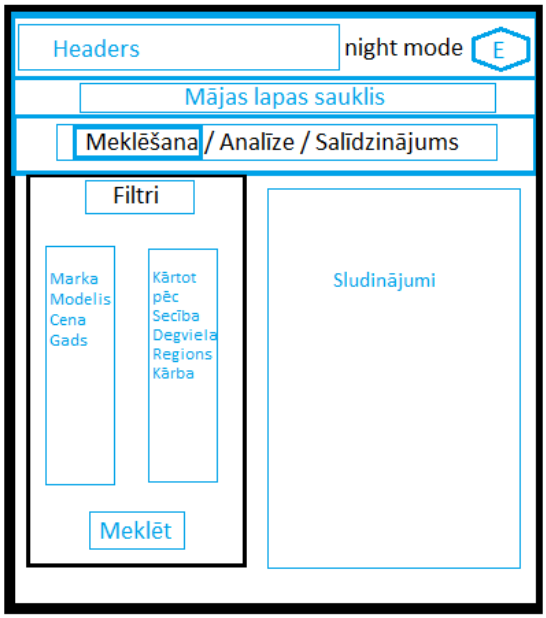
Datu formāti:

- Sistēmai jāatbalsta plaši izmantoti datu formāti (JSON, CSV) importam un eksportam.
- Sistēmai jānodrošina datu validācija un sanitizācija pirms importa.

2.4.7. Lietotāja interfeiss

Lietotāja interfeisa skices ir pieejamas 3.1.,3.2.,3.3.,3.4., attēlos. Tās demonstrē galvenos lietotāja saskarnes elementus, veidoju tos projekta sākumā ar parastiem zīmēšanas instrumentiem, lai iztēlotu nākotnes izskatu.

Sākuma ekrāna lietotāja interfeisa skice (3.1. att):

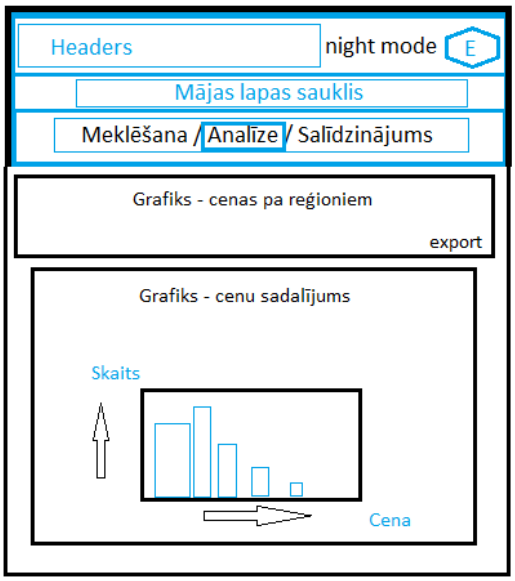


3.1. Att. Galvenais ekrāns skice

Sistēmas lietotāja saskarnei jāatbilst šādām prasībām:

- Jānodrošina meklēšanas forma ar visiem nepieciešamajiem filtrēšanas parametriem.
- Jāattēlo meklēšanas rezultāti tabulā vai kartiņu skatā.
- Jānodrošina pārslēgšanās starp dažādām skatīšanas režīmu cilnēm (meklēšana, analīze, salīdzināšana).

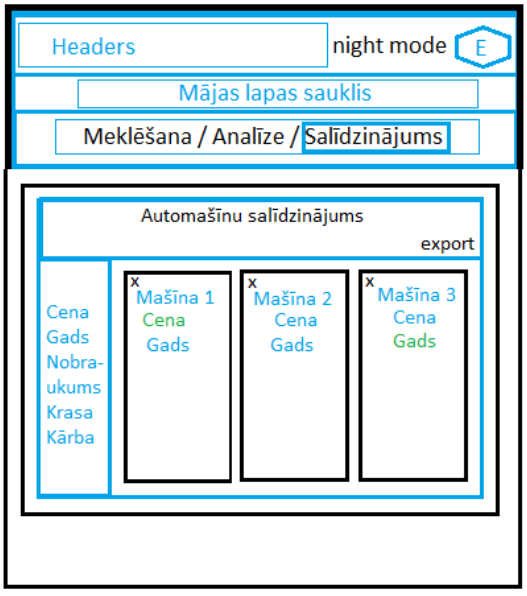
Analīzes sadaļas ekrāna skice (3.2. att.):



3.2. Att. Analīzes ekrāns skice

- Jāattēlo statistikas dati un grafiki par atrastajiem sludinājumiem.
- Jānodrošina interaktīvi elementi grafiku manipulēšanai.
- Jānodrošina eksporta funkcionalitāte dažādos formātos.

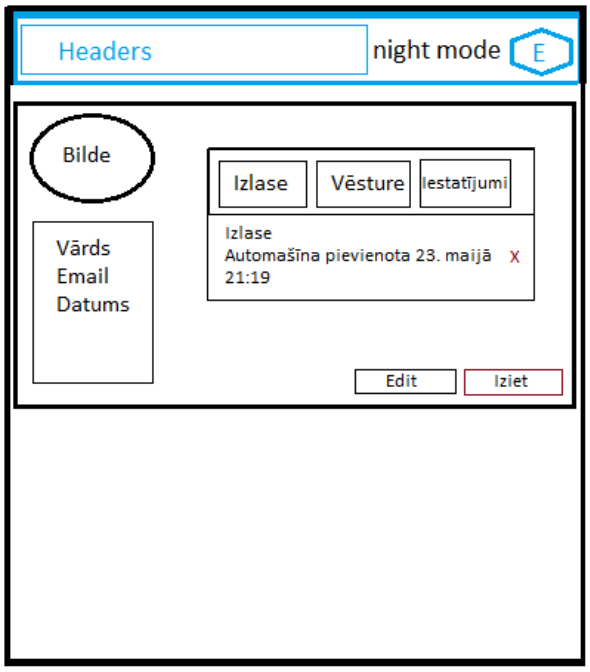
Salīdzināšanas sadaļas ekrāna skice (3.3. att.):



3.3. Att. Salīdzināšanas ekrāns skice

- Jāattēlo salīdzinājuma tabula ar izvēlētajām automašīnām.
- Jāizceļ labākās vērtības katrai īpašībai.
- Jānodrošina iespēja noņemt automašīnas no salīdzinājuma.
- Jānodrošina forma automašīnas parametru ievadei.

Lietotāja profila ekrāna skice (3.4. att.):



3.4.att. Lietotāja profils skice

- Jāattēlo lietotāja informācija un preferences.
- Jānodrošina iespēja mainīt lietotāja datus un preferences.
- Jāattēlo lietotāja izlases automašīnas un meklēšanas vēsture.

Pēc prasību definēšanas nākamais būtiskais lēmums bija tehnoloģiju izvēle. Šis process prasīja rūpīgi censties izprast katras tehnoloģijas priekšrocības un trūkumus konkrētā projekta kontekstā.

3. UZDEVUMA RISINĀŠANAS LĪDZEKĻU IZVĒLES PAMATOJUMS

Automobiļu tirgus analīzes sistēma ir kompleksa tīmekļa sistēma, kas apvieno datu iegūšanu, apstrādi, analīzi un vizualizāciju. Tās izstrādei izvēlēti rīki un tehnoloģijas, kas nodrošina sistēmas prasību efektīvu realizāciju. Šajā sadaļā tiek pamatota izvēlēto tehnoloģiju un rīku piemērotība uzdevuma risināšanai.

3.1. Serveris un backenda tehnoloģijas

3.1.1. Python (versija 3.10)

Izvēlējos Python galvenokārt tāpēc, ka man jau bija pieredze ar to. Bet izrādījās, ka tam ir arī citas priekšrocības šim projektam:

- Python ir ļoti labs darbam ar datiem. Tam ir daudz gatavu bibliotēku - NumPy skaitļiem, Matplotlib grafikiem, BeautifulSoup tīmekļa lapām. Tā vietā, lai visu rakstītu no nulles, varēju izmantot to, ko citi jau ir izdomājuši.
- Vēl viens pluss - Python kods ir viegli lasāms. Kad pēc nedēļas atgriezos pie sava koda, vēl sapratu, ko biju rakstījis. Ar C++ man tas ne vienmēr izdodas.
- Tīmekļa skrāpēšanai Python ir perfekts. BeautifulSoup un Requests bibliotēkas ļauj viegli paņemt datus no SS.LV. Sākumā mēģināju ar citiem rīkiem, bet Python bija vienkāršākais.
- Ja radās problēmas, internetā vienmēr atrada risinājumu. Python kopienā ir milzīga - praktiski jebkurai problēmai kāds jau ir saskāries un uzrakstījis risinājumu Stack Overflow mājaslapā.
- Visbeidzot, Python strādā uz visām operētājsistēmas. Mans draugs ar Mac operētājsistēmu varēja palaist to pašu kodu, ko es rakstīju uz Windows.
- Galvenais iemesls tomēr bija tas, ka ar Python var ātri izprotot idejas. Ja man radās doma, kā uzlabot analīzi, varēju to izprogrammēt pāris stundu laikā un uzreiz redzēt, vai tas strādā.

3.1.2. Flask (versija 2.0.1)

Priekš web servera izvēlējos Flask, jo:

- Manuprāt, Flask ir daudz vienkāršāks nekā Django. Nav jāmācās milzum daudz lietu uzreiz, kas nodrošina tikai pamatfunkcionalitāti, ļaujot izstrādātājam viegli izvēlēties nepieciešamās komponentes.
- RESTful API atbalsts: Flask ļauj viegli izveidot RESTful API, kas nepieciešams komunikācijai ar frontend daļu.
- Laba saderība ar SQLAlchemy: Flask labi integrējas ar SQLAlchemy ORM, kas atvieglo datu bāzes operācijas.
- Vienkārša konfigurācija: Flask ir viegli konfigurējams un palaižams dažādās vidēs.

- Plašs paplašinājumu klāsts: Flask-CORS, Flask-JWT-Extended un citi paplašinājumi atvieglo specifisko funkcionalitāšu implementāciju.

Flask izvēle bija apzināta - sākotnēji apsverēju arī Django izmantošanu, taču Flask vieglākā struktūra labāk atbilda projekta vajadzībām. Kvalifikācijas darba ietvaros man nebija nepieciešama Django sarežģītā arhitektūra, un Flask ļāva ātri izveidot funkcionējošu API bez liekas konfigurācijas.

3.1.3. SQLAlchemy (versija 1.4.35)

Datu bāzei izvēlējos SQLAlchemy. Sākumā domāju rakstīt SQL vaicājumus pats, bet izrādījās:

- Datu bāzu neatkarība: SQLAlchemy ļauj strādāt ar dažādām datu bāzēm (SQLite, MySQL, PostgreSQL), nemainot kodu.
- Objektu orientēta pieeja: SQLAlchemy piedāvā objektu orientētu pieeju datu bāzes operācijām, kas atvieglo datu manipulāciju.
- Efektīvas vaicājumu iespējas: SQLAlchemy nodrošina jaudīgu vaicājumu valodu kompleksiem datu izgūšanas uzdevumiem.
- Migrācijas atbalsts: SQLAlchemy integrējas ar Alembic, kas nodrošina datu bāzes shēmas migrācijas.

Sākumā domāju rakstīt tiešus SQL vaicājumus, bet SQLAlchemy ORM ietaupīja milzīgi daudz laika. Īpaši noderīgi bija relationship definīcijas, kas ļāva viegli pārvietoties starp saistītajiem datiem

3.1.4. BeautifulSoup (versija 4.9.3) un un aiohttp (versija 3.8.0)

SS.LV datu iegūšanai man vajadzēja divus rīkus:

- BeautifulSoup: Nodrošina vieglu un efektīvu HTML parsēšanu un navigāciju, kas ir ideāli piemērota statiskā satura izgūšanai.
- aiohttp: Asinhronā HTTP klienta bibliotēka, kas ļauj veikt daudzus pieprasījumus paralēli, būtiski uzlabojot scraping ātrumu. Nodrošina semaphore kontroli, lai nenoslogotu mērķa serveri.
- Papildināmība: Abas bibliotēkas labi darbojas kopā - aiohttp iegūst HTML saturu, BeautifulSoup to parsē.

Sākotnēji plānoju izmantot Selenium dinamiskā satura iegūšanai, taču izrādījās, ka SS.LV saturs ir lielākoties statisks, un aiohttp ātrums bija daudz svarīgāks par Selenium iespējām.

3.1.5. SQLite (versija 3.38.0)

SQLite ir izvēlēta kā datu bāzes pārvaldības sistēma, jo:

- Viegla un portatīva: SQLite neprasa atsevišķu servera uzstādīšanu, dati tiek glabāti vienā failā.

- Vienkārša uzturēšana: SQLite datubāze ir viegli dublējama, atjaunojama un pārvietojama, kas man patīk.
- Pārnesamība: SQLite darbojas visās populārākajās operētājsistēmās bez papildu konfigurācijas.
- Sistēma ir veidota tā, lai nepieciešamības gadījumā varētu viegli migrēt uz jaudīgākām datu bāzu sistēmām, piemēram, PostgreSQL.

Kvalifikācijas darba ietvaros SQLite bija ideāla izvēle - nav jātērē laiks uz datu bāzes servera uzstādīšanu un konfigurāciju. Sistēma ir veidota tā, lai nepieciešamības gadījumā varētu viegli migrēt uz jaudīgākām datu bāzu sistēmām, piemēram, PostgreSQL.

3.2. Frontend tehnoloģijas

3.2.1. JavaScript (ES6+)

Frontend daļai izvēlējos JavaScript, jo citas izvēles īsti man nav:

- Universālums: JavaScript ir standarta valoda tīmekļa pārlūkprogrammās, kas nodrošina pilnīgu saderību.
- Modernās iespējas: ES6+ piedāvā daudzas modernas programmēšanas iespējas, piemēram, klases, arrow funkcijas, destructuring, kas atvieglo izstrādi.
- Asinhronās operācijas: JavaScript nodrošina efektīvu asinhrono operāciju apstrādi ar Promise un async/await, kas ir būtiski API pieprasījumiem.

Lai gan sākotnēji apsvēru arī TypeScript izmantošanu, JavaScript izrādījās pietiekams projekta vajadzībām, un tā vienkāršība ļāva koncentrēties uz funkcionalitātes izstrādi.

3.2.2. React (versija 18.2.0)

React izvēlējos tāpēc, ka:

- Komponentu pieeja: React ļauj sadalīt saskarni atkārtoti izmantojamās komponentēs, kas atvieglo izstrādi un uzturēšanu.
- Virtuālais DOM: React virtuālais DOM(Document object model) nodrošina efektīvu lapas atjaunināšanu, kas uzlabo lietotāja pieredzi.
- Stāvokļa pārvaldība: React nodrošina efektīvu stāvokļa (state) pārvaldību, kas ir būtiska dinamiskām lietojumprogrammām.
- Lieliska ekosistēma: React ir plaša ekosistēma ar daudzām papildu bibliotēkām un rīkiem.
- Plašs kopienas atbalsts: React ir liela kopiena un plaša dokumentācija.

React man sākumā šķita sarežģīts ar saviem ‘hooks’ un ‘state’ pārvaldību, bet pēc pirmajām nedēļām kļuva skaidrs, kāpēc tas ir tik populārs - komponenti patiešām ir atkārtoti izmantojami un kods kļūst strukturētāks.

3.2.3. Material-UI (MUI, versija 5.0.0)

Material-UI ir izvēlēts kā React komponentu bibliotēka, jo:

- Gatavi komponenti: MUI piedāvā plašu gatavu komponentu klāstu, kas atbilst Material Design vadlīnijām.
- Pielāgojamība: MUI komponenti ir viegli pielāgojami, lai atbilstu specifiskām dizaina prasībām.
- Responsivitāte: MUI nodrošina iebūvētu atbalstu mobilajām ierīcēm un dažādiem ekrāna izmēriem.
- Tēmu atbalsts: MUI nodrošina vienkāršu tēmu pārvaldību, ļaujot viegli implementēt tumšo režīmu.

Izvēlējos MUI pēc CSS framework salīdzināšanas - tā dokumentācija bija skaidrāka nekā citu risinājumu, un komponenti izskatījās profesionāli bez papildu CSS rakstīšanas.

3.3. Datu analīzes un vizualizācijas bibliotēkas

3.3.1. NumPy (versija 1.22.3)

Statistikai un matemātikai izvēlējos NumPy:

- Matemātiskās operācijas: NumPy nodrošina efektīvas matemātiskās operācijas ar masīviem un matricām. - Statistiskās funkcijas: NumPy piedāvā plašu statistisko funkciju klāstu (np.mean(), np.median(), np.std()).
- Efektīva datu apstrāde: NumPy masīvi nodrošina ātru skaitlisko datu manipulāciju un aprēķinus.
- Atmiņas efektivitāte: NumPy C-based implementācija ir daudz ātrāka par standarta Python sarakstiem.

NumPy izrādījās pietiekams visiem nepieciešamajiem statistikas aprēķiniem. Tā efektīvie masīvi un matematik iskie algoritmi nodrošināja ātru datu apstrādi bez papildu sarežģītības.

3.3.2. Matplotlib (versija 3.5.1) un Seaborn (versija 0.11.2)

Šīs bibliotēkas ir izvēlētas datu vizualizācijai, jo:

- Plašas vizualizācijas iespējas: Matplotlib nodrošina zema līmeņa API detalizētai grafiku pielāgošanai.
- Statistiskie grafiki: Seaborn piedāvā augstāka līmeņa API, kas ir īpaši piemērots statistisko datu vizualizācijai.
- Estētika: Seaborn piedāvā modernākus un estētiskākus noklusējuma stilus.
- Laba integrācija: Abas bibliotēkas labi integrējas ar NumPy datiem.

Matplotlib sākumā šķita sarežģīts, bet Seaborn to padara daudz vienkāršāku. Kombinācijā var izveidot gan vienkāršus, gan profesionālus grafikus.

3.4. Citas tehnoloģijas un rīki

3.4.1. JWT (JSON Web Tokens, versija 2.3.0)

JWT ir izvēlēts autentifikācijai, jo:

- Bezstāvokļa autentifikācija: JWT ļauj implementēt bezstāvokļa autentifikāciju, kas ir skalējama un efektīva.
- Drošība: JWT nodrošina datu integritāti un autentiskumu.
- Kompaktums: JWT tokens ir kompakts un var tikt nodots HTTP galvenēs vai sīkfailos.

Izvēlējos JWT pēc tam, kad salīdzināju ar tradicionālajām sesijām - tā vienkāršība un neatkarība no servera sesiju glabāšanas bija ideāla šim projektam.

3.4.2. Git un GitHub

Git un GitHub ir izvēlēti kodu versiju kontrolei, jo:

- Izmaiņu izsekošana: Git nodrošina pilnīgu izmaiņu vēsturi, ļaujot izsekot koda evolūcijai.
- Sadarbība: GitHub atvieglo komandas sadarbību, nodrošinot koda pārskatīšanu un problēmu izsekošanu.
- Automatizācija: GitHub Action ļauj automatizēt izstrādes un izvietošanas procesus.

Pirms projekta veidošanas biju saskāries ar GitHub izmantošanu, taču līdz galām to nebiju apguvis. Šī projekta veidošanas procesā varu teikt, esmu iegūvis dziļākas zināšanas un pieņemā līmenī apguvis GitHub izmantošanu. Bez Git es būtu pazudis koda versijās - vairākkārt man noderēja iespēja atgriezties pie agrākām versijām, kad kaut kas saplīsa, un tas notika daudz.

3.4.3. ESLint un Prettier

Šie rīki ir izvēlēti koda kvalitātes nodrošināšanai, jo:

- Koda stila konsekvence: Prettier nodrošina konsekventu koda noformējumu.
- Kļūdu novēršana: ESLint palīdz atklāt potenciālās kļūdas un problēmas kodā.
- Konfigurējamība: Abi rīki ir konfigurējami, lai atbilstu projekta specifiskajām prasībām.

Šie rīki var šķist lieki sākumā, bet tiem pateicoties mans kods kļuva daudz tīrāks un konsistentāks. ESLint īpaši palīdzēja atrast kļūdas, ko es nemaz nebūtu pamanījis.

3.5. Izvēles pamatojuma kopsavilkums

Kopumā sanāca labs tehnoloģiju maisījums. Python backend ar React frontend strādā labi kopā. Servera puses Python ekosistēma kopā ar Flask un SQLAlchemy nodrošina efektīvu datu apstrādi un API. Klienta puses React ar Material UI nodrošina modernu un responsīvu lietotāja saskarni. Datu analīzes un vizualizācijas bibliotēkas (NumPy, Matplotlib, Seaborn) sniedz jaudīgus rīkus statistiskajai analīzei.

Tehnoloģijas izvēlētas, ņemot vērā to briedumu, kopienas atbalstu, dokumentāciju un saderību ar projekta prasībām. Sistēmas arhitektūra ir veidota modulāri, ļaujot viegli pievienot jaunas funkcionalitātes vai aizstāt atsevišķas komponentes nākotnē.

Kvalifikācijas darba ietvaros sistēma fokusējas uz četru automašīnu marku (Tesla, Infiniti, Smart, Suzuki) datu analīzi no SS.LV vietnes, taču izvēlētās tehnoloģijas nodrošina iespēju nākotnē paplašināt sistēmu, pievienojot jaunus datu avotus un paplašinot analizējamo marku klāstu.

Izmantojot šīs tehnoloģijas, ir izdevies izveidot stabilu, efektīvu un lietotājam draudzīgu automobiļu tirgus analīzes sistēmu, kas atbilst visām definētajām prasībām. Pēc tehnoloģiju izvēles pamatošanas ir būtiski apskatīt, kā šīs tehnoloģijas tiek izmantots sistēmas arhitektūrā un datu struktūru veidošanā.

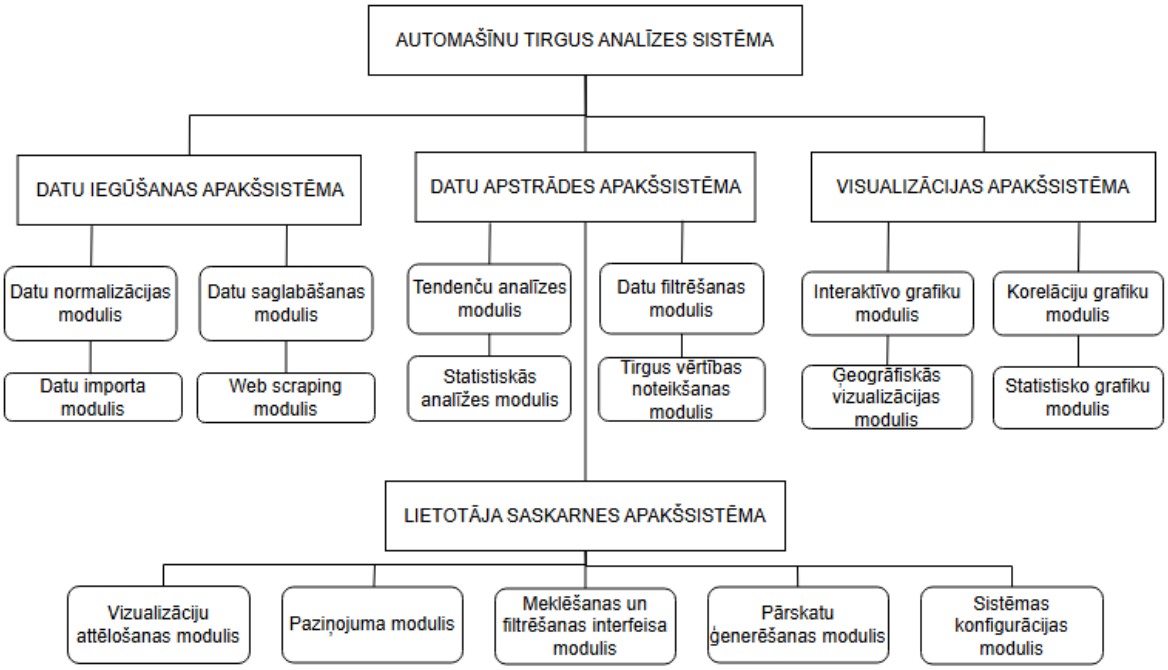
4. PROGRAMMATŪRAS PRODUKTA MODELĒŠANA UN PROJEKTĒŠANA

4.1. Sistēmas struktūras modelis

4.1.1. Sistēmas arhitektūra

Automobiļu tirgus analīzes sistēma ir izstrādāta, izmantojot mūsdienīgu, modulāru arhitektūru, kas balstās uz klienta-servera modeli ar REST API. Sistēmas arhitektūra ir sadalīta loģiskos slāņos, kas nodrošina skaidru atbildību nodalīšanu un atvieglo sistēmas uzturēšanu un paplašināšanu.

4.1. attēlā ir attēlota sistēmas funkcionālās dekompozīcijas diagramma. Šādas modulārās arhitektūras izvēle nebija nejauša. Kvalifikācijas darba sākumā mēģināju izveidot vienu lielu skriptu, kas veiktu visas darbības, taču ātri sapratu, ka šāda pieeja nav uzturama.



4.1.att. Sistēmas funkcionālās dekompozīcijas diagramma

Kā redzams 4.1. attēlā, sistēma sastāv no četrām galvenajām apakšsistēmām:

DATU IEGŪŠANAS APAKŠSISTĒMA

Šī apakšsistēma ir atbildīga par datu iegūšanu no ārējiem avotiem un to saglabāšanu datu bāzē. Tā sastāv no šādiem moduļiem:

- Web Scraping modulis - atbild par datu iegūšanu no SS.LV vietnes, izmantojot BeautifulSoup un aiohttp bibliotēkas. Šis modulis pēc pieprasījuma skenē vietni, lai iegūtu jaunākos automašīnu sludinājumus.. Modulis ir implementēts failā ss_scraper.py.
- Datu normalizācijas modulis - atbild par iegūto datu tīrīšanu, formatēšanu un normalizēšanu, lai nodrošinātu to konsistenci un kvalitāti. Šis modulis pārveido

dažādus datu formātus (piemēram, cenas ar vai bez valūtas simbola) vienotā formātā. Modulis ir implementēts kā daļa no ss_scraper.py faila.

- Datu saglabāšanas modulis - atbild par normalizēto datu saglabāšanu datu bāzē, izmantojot SQLAlchemy ORM. Šis modulis arī atjaunina esošos ierakstus un marķē vecus sludinājumus kā neaktīvus. Modulis ir implementēts kā daļa no ss_scraper.py un models.py failiem.

DATU APSTRĀDES APAKŠSISTĒMA

Šī apakšsistēma ir atbildīga par datu apstrādi, analīzi un statistisko aprēķinu veikšanu. Tā sastāv no šādiem moduļiem:

- Datu filtrēšanas modulis - atbild par datu filtrēšanu pēc lietotāja norādītajiem kritērijiem (marka, modelis, gads, cena, utt.). Šis modulis izmanto SQL vaicājumus un NumPy bibliotēku, lai efektīvi atlasītu datus. Modulis ir implementēts kā daļa no api.py faila.
- Statistiskās analīzes modulis - atbild par statistisko aprēķinu veikšanu (vidējā cena, mediāna, min/max vērtības, standartnovirze). Šis modulis izmanto NumPy bibliotēku, lai veiktu statistiskos aprēķinus. Modulis ir implementēts failā analysis.py.
- Tirgus vērtības noteikšanas modulis - atbild par automašīnas tirgus vērtības aprēķināšanu, balstoties uz tās parametriem un līdzīgām automašīnām tirgū. Šis modulis izmanto statistiskos algoritmus, lai noteiktu vērtības. Modulis ir implementēts kā daļa no analysis.py faila.
- Tendenču analīzes modulis - atbild par cenu tendenču analīzi laika gaitā un dažādos reģionos. Šis modulis izmanto laika rindas analīzes metodes, lai identificētu tendences. Modulis ir implementēts kā daļa no analysis.py faila.

VIZUALIZĀCIJAS APAKŠSISTĒMA

Šī apakšsistēma ir atbildīga par datu vizualizāciju, lai atvieglotu to interpretāciju. Tā sastāv no šādiem moduļiem:

- Statistisko grafiku modulis - atbild par histogrammu un līniju grafiku ģenerēšanu, kas atspoguļo cenu sadalījumu un tendences. Šis modulis izmanto Matplotlib un Seaborn bibliotēkas. Modulis ir implementēts kā daļa no analysis.py faila un ģenerē cenu sadalījuma histogrammas un cenu tendenču līniju grafikus.
- Interaktīvo grafiku modulis - frontendā implementēts modulis, kas nodrošina interaktīvas vizualizācijas un datu analīzi, izmantojot React un JavaScript. Šis modulis ietver specializētus komponentus dažādu analīžu veikšanai: - PriceAnalysisChart.jsx - galvenais grafiku komponents ar dažādiem skatījumu režīmiem - PriceYearTrendChart.jsx - cenu analīze pēc izlaiduma gadiem - PriceByFuelTypeChart.jsx - cenu salīdzinājums pēc degvielas tipiem - RegionalPriceComparison.jsx - reģionālo cenu atšķirību analīze - MarketInsights.jsx - tirgus tendences un statistika.

- Datu eksporta modulis - nodrošina iespēju saglabāt analīzes rezultātus dažādos formātos. Modulis ļauj eksportēt grafikus PNG formātā, salīdzinājuma datus JSON formātā, un meklēšanas rezultātus CSV formātā. Implementēts gan backend (analysis.py base64 kodēšana) gan frontend (exporthelpers.js) līmenī.

LIETOTĀJA SASKARNES APAKŠSISTĒMA

Šī apakšsistēma ir atbildīga par lietotāja mijiedarbību ar sistēmu. Tā sastāv no šādiem moduļiem:

- Meklēšanas un filtrēšanas interfeisa modulis - atbild par lietotāja saskarni, kas ļauj norādīt meklēšanas kritērijus un filtrēt rezultātus. Šis modulis ir implementēts kā React komponente SearchForm.js.
- Vizualizāciju attēlošanas modulis - atbild par grafiku un diagrammu attēlošanu interaktīvā veidā. Šis modulis ir implementēts kā React komponentes, piemēram, PriceChart.js un RegionalPriceComparison.js.
- Pārskatu ģenerēšanas modulis - atbild par pārskatu ģenerēšanu un eksportēšanu dažādos formātos (CSV, JSON). Šis modulis ir implementēts kā daļa no frontend komponentēm.
- Sistēmas konfigurācijas modulis - atbild par sistēmas iestatījumu pārvaldību, tostarp lietotāja preferences un datu avotu konfigurāciju. Šis modulis ir implementēts kā daļa no app.py un config.py failiem.
- Autentifikācijas modulis - atbild par lietotāju autentifikāciju un piekļuves kontroli. Šis modulis ir implementēts failā auth_models.py.

Sistēmas slāņu arhitektūra

Sistēmas arhitektūra ir balstīta uz šādiem slāņiem:

1. Datu piekļuves slānis - atbild par mijiedarbību ar datu bāzi, izmantojot SQLAlchemy ORM. Šis slānis ir implementēts failos models.py un daļēji analysis.py.
2. Biznesa loģikas slānis - atbild par datu apstrādi, analīzi un biznesa noteikumu piemērošanu. Šis slānis ir implementēts failos analysis.py un controller.py.
3. API slānis - atbild par REST API nodrošināšanu, kas ļauj frontendai mijiedarboties ar serverī esošajiem datiem un loģiku. Šis slānis ir implementēts failā api.py.
4. Frontend slānis - atbild par lietotāja saskarni un mijiedarbību ar lietotāju. Šis slānis ir implementēts React komponentēs, kas atrodas src/components/ direktorijā.

Komunikācijas plūsma

Sistēmas komponentes mijiedarbojas šādā veidā:

1. Lietotājs mijiedarbojas ar frontend saskarni, veicot darbības, piemēram, meklēšanu vai grafiku ģenerēšanu.
2. Frontend komponentes nosūta API pieprasījumus serverim, izmantojot HTTP protokolu.
3. API kontrolieri apstrādā pieprasījumus, izsaucot atbilstošās biznesa loģikas funkcijas.

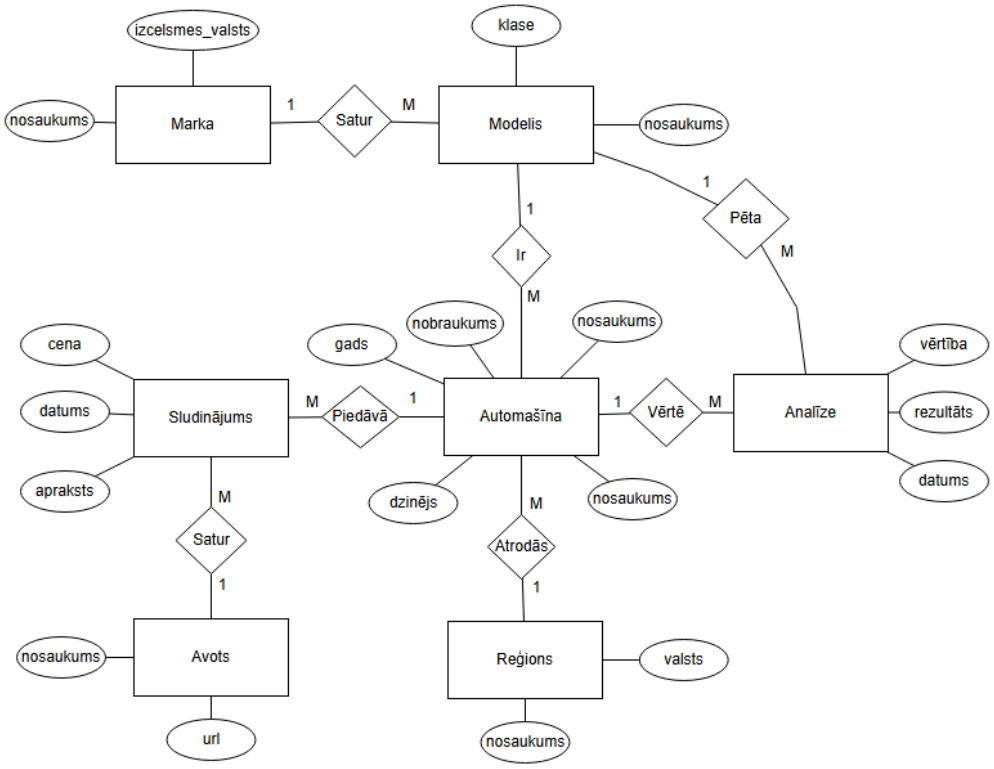
4. Biznesa loģikas funkcijas apstrādā datus, veic aprēķinus un mijiedarbojas ar datu piekļuves slāni.
5. Datu piekļuves slānis veic datu bāzes operācijas, izmantojot SQLAlchemy ORM.
6. Rezultāti tiek nodoti atpakaļ caur slāņiem līdz frontend saskarnei, kur tie tiek attēloti lietotājam.
7. Paralēli tam, Web Scraping modulis periodiski iegūst jaunus datus no SS.LV vietnes un saglabā tos datu bāzē.

   Šāda modulāra, slāņaina arhitektūra nodrošina vairākas priekšrocības:

   - Atbildību nodalīšana - katrs modulis un slānis ir atbildīgs par konkrētu funkcionalitāti, kas atvieglo sistēmas uzturēšanu un paplašināšanu.
   - Atkārtota izmantojamība - moduļus var izmantot atkārtoti dažādās sistēmas daļās.
   - Testējamība - moduļus var testēt neatkarīgi, kas atvieglo kļūdu atrašanu un novēršanu.
   - Skalējamība - sistēmu var viegli paplašināt, pievienojot jaunus moduļus vai modificējot esošos.
   - Paralēla izstrāde - dažādi izstrādātāji var strādāt pie dažādiem moduļiem vienlaicīgi.

4.1.2. Sistēmas ER modelis

Sistēmas datu struktūra ir modelēta, izmantojot entītiju-relāciju (ER) modeli, kas parāda galvenās datu entītijas un to savstarpējās attiecības. 7. attēlā ir attēlota sistēmas ER diagramma.



4.2.att. Sistēmas entītiju-relāciju diagramma

ER diagrammā ir attēlotas šādas galvenās entītijas:

Brand (Marka) - Šī entītija satur informāciju par automašīnu markām:

- brand_id: Unikāls markas identifikators (primārā atslēga)
- name: Markas nosaukums (piemēram, "Tesla", "Infiniti")
- country: Markas izcelsmes valsts
- logo_url: URL uz markas logotipu
- created_at, updated_at: Ieraksta izveidošanas un atjaunināšanas laika zīmogi

Model (Modelis) - Šī entītija satur informāciju par automašīnu modeļiem:

- model_id: Unikāls modeļa identifikators (primārā atslēga)
- brand_id: Atsauce uz marku (ārējā atslēga)
- name: Modeļa nosaukums (piemēram, "Model S", "QX80")
- class_type: Transportlīdzekļa klases tips
- production_start: Ražošanas sākuma gads
- production_end: Ražošanas beigu gads
- created_at, updated_at: Ieraksta izveidošanas un atjaunināšanas laika zīmogi

Region (Reģions) - Šī entītija satur informāciju par ģeogrāfiskajiem reģioniem:

- region_id: Unikāls reģiona identifikators (primārā atslēga)
- name: Reģiona nosaukums (piemēram, "Rīga", "Vidzeme")
- country: Valsts nosaukums
- lat, lng: Ģeogrāfiskās koordinātas (platuma un garuma grādi)
- created_at, updated_at: Ieraksta izveidošanas un atjaunināšanas laika zīmogi

Car (Automašīna) - Šī entītija satur informāciju par konkrētām automašīnām:

- car_id: Unikāls automašīnas identifikators (primārā atslēga)
- model_id: Atsauce uz modeli (ārējā atslēga)
- region_id: Atsauce uz reģionu (ārējā atslēga)
- year: Izlaiduma gads
- engine_volume: Dzinēja tilpums litros
- engine_type: Dzinēja tips (benzīns, dīzelis, u.c.)
- transmission: Ātrumkārbas tips
- tech_inspection: Tehniskās apskates statuss
- mileage: Nobraukums
- body_type: Virsbūves tips
- color: Automašīnas krāsa
- created_at, updated_at: Ieraksta izveidošanas un atjaunināšanas laika zīmogi

Source (Avots) - Šī entītija satur informāciju par datu avotiem:

- source_id: Unikāls avota identifikators (primārā atslēga)
- name: Avota nosaukums (piemēram, "SS.LV")
- url: Avota URL
- country: Valsts, kurā darbojas avots

- scraping_config: JSON konfigurācija datu iegūšanai
- last_scraped_at: Pēdējās datu iegūšanas laika zīmogs
- created_at, updated_at: Ieraksta izveidošanas un atjaunināšanas laika zīmogi

Listing (Sludinājums) - Šī entītija satur informāciju par automašīnu sludinājumiem:

- listing_id: Unikāls sludinājuma identifikators (primārā atslēga)
- car_id: Atsauce uz automašīnu (ārējā atslēga)
- source_id: Atsauce uz avotu (ārējā atslēga)
- external_id: Sludinājuma identifikators avota vietnē
- price: Cena eiro
- listing_date: Sludinājuma publicēšanas datums
- listing_url: URL uz oriģinālo sludinājumu
- is_active: Vai sludinājums joprojām ir aktīvs
- created_at, updated_at: Ieraksta izveidošanas un atjaunināšanas laika zīmogi

Analysis (Analīze) - Šī entītija satur informāciju par veiktajām analīzēm:

- analysis_id: Unikāls analīzes identifikators (primārā atslēga)
- car_id: Atsauce uz automašīnu (ārējā atslēga, neobligāta)
- model_id: Atsauce uz modeli (ārējā atslēga, neobligāta)
- title: Analīzes nosaukums
- description: Analīzes apraksts
- params: JSON parametri
- results: JSON rezultāti
- created_at, updated_at: Ieraksta izveidošanas un atjaunināšanas laika zīmogi

MarketValue (Tirgus vērtība) - Šī entītija satur informāciju par aprēķinātajām tirgus vērtībām:

- value_id: Unikāls vērtības identifikators (primārā atslēga)
- model_id: Atsauce uz modeli (ārējā atslēga)
- region_id: Atsauce uz reģionu (ārējā atslēga)
- year: Transportlīdzekļa gads
- engine_type: Dzinēja tips
- avg_price: Vidējā cena
- median_price: Mediānas cena
- min_price, max_price: Minimālā un maksimālā cena
- std_deviation: Standarta novirze
- sample_size: Paraugu skaits
- calculation_date: Aprēķina datums
- created_at, updated_at: Ieraksta izveidošanas un atjaunināšanas laika zīmogi

Entītiju savstarpējās saites ER diagrammā ir šādas:

1. Brand un Model - Viena marka var būt saistīta ar daudziem modeļiem (one to many). Saite realizēta caur Model.brand_id lauku, kas atsaucas uz Brand.brand_id.
2. Model un Car - Viens modelis var būt saistīts ar daudzām automašīnām (one to many). Saite realizēta caur Car.model_id lauku, kas atsaucas uz Model.model_id.
3. Region un Car - Viens reģions var būt saistīts ar daudzām automašīnām (one to many). Saite realizēta caur Car.region_id lauku, kas atsaucas uz Region.region_id.
4. Car un Listing - Viena automašīna var būt saistīta ar daudziem sludinājumiem (one to many). Saite realizēta caur Listing.car_id lauku, kas atsaucas uz Car.car_id.
5. Source un Listing - Viens avots var būt saistīts ar daudziem sludinājumiem (one to many). Saite realizēta caur Listing.source_id lauku, kas atsaucas uz Source.source_id.
6. Car un Analysis - Viena automašīna var būt saistīta ar daudzām analīzēm (one to many). Saite realizēta caur Analysis.car_id lauku, kas atsaucas uz Car.car_id.
7. Model un Analysis - Viens modelis var būt saistīts ar daudzām analīzēm (one to many). Saite realizēta caur Analysis.model_id lauku, kas atsaucas uz Model.model_id.
8. Model un MarketValue - Viens modelis var būt saistīts ar daudzām tirgus vērtībām (one to many). Saite realizēta caur MarketValue.model_id lauku, kas atsaucas uz Model.model_id.
9. Region un MarketValue - Viens reģions var būt saistīts ar daudzām tirgus vērtībām (one to many). Saite realizēta caur MarketValue.region_id lauku, kas atsaucas uz Region.region_id.

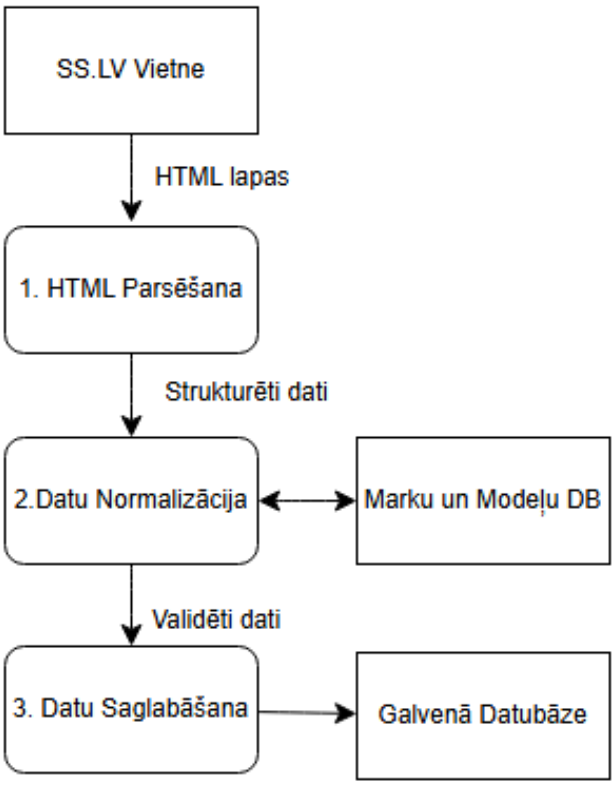
Šis ER modelis nodrošina efektīvu datu organizāciju, kas ļauj veikt dažādas analīzes un salīdzinājumus, kā arī nodrošina sistēmas paplašināmību nākotnē.

4.2. Datu plūsmu modelis

Šajā sadaļā tiks detalizēti aprakstīti sistēmas galvenie datu apstrādes procesi. Automobiļu tirgus analīzes sistēmā ir vairāki svarīgi procesi, kas ir būtiski datu apstrādē, analīzē un vizualizācijā. Turpmāk ir aprakstīti 6 nozīmīgākie procesi, kuru ietvaros notiek datu transformācija un apstrāde.

4.2.1. Web scraping modulis

Web scraping modulis ir atbildīgs par automašīnu sludinājumu datu iegūšanu no SS.LV vietnes. Šis process ir kritiski svarīgs, jo tas nodrošina sistēmai svaigus datus, kas nepieciešami turpmākai analīzei un vizualizācijai.



4.3.att. Web scraping procesa datu plūsmas diagramma

Datu iegūšanas process notiek šādi:

1. **HTML parsēšana:** SS.LV vietne tiek pieprasīta, izmantojot HTTP GET pieprasījumus ar rotējošiem User-Agent headeriem. Iegūtās HTML lapas tiek analizētas ar BeautifulSoup bibliotēku, identificējot automašīnu sludinājumu tabulas struktūru. Sistēma navigē caur marku un modeļu hierarhiju, iegūstot pamata informāciju: cenu, gadu, dzinēju, nobraukumu katram sludinājumam.
2. **Datu normalizācija:** Iegūtie raw HTML dati tiek pārveidoti strukturētos datos. Cenas tiek konvertētas no string formāta (“15 000 €”) uz integer (15000), dzinēja tipi tiek klasificēti kategorijās (benzīns, dīzelis, elektriskais), reģioni tiek standartizēti pēc Latvijas administratīvā iedalījuma. Sistēma mijiedarbojas ar Marku un Modeļu DB, lai pārbaudītu vai izveidotu nepieciešamos ierakstus.

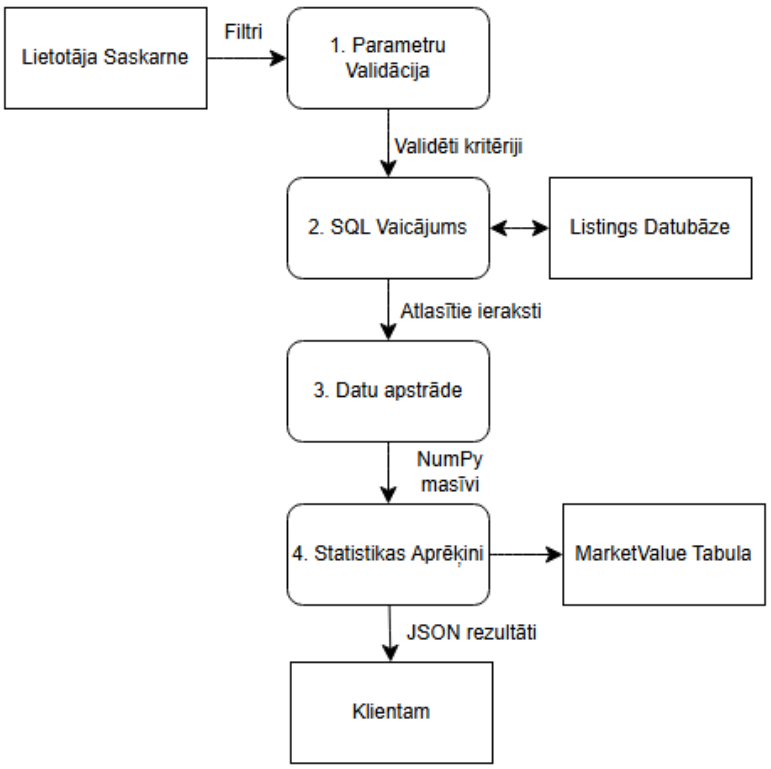
3. **Datu saglabāšana:** Validētie un normalizētie dati tiek saglabāti galvenajā datubāzē caur SQLAlchemy ORM. Sistēma pārbauda, vai automašīna jau eksistē (pēc parametru kombinācijas), un vai atjaunina esošu ierakstu, vai izveido jaunu. Sludinājumi tiek saglabāti listings tabulā ar atsauci uz car_id. Vecāki sludinājumi (>30 dienas) tiek marķēti kā neaktīvi.

Web scraping modulis nodrošina sistēmas datu avotu, automātiski iegūstot un normalizējot automašīnu sludinājumu informāciju no SS.LV vietnes. Asinhronā apstrāde ar aiohttp semaphore(3) ierobežojumu nodrošina efektīvu datu iegūšanu, nepārslogojot mērķa serveri

Šī procesa implementācija izrādījās sarežģītāka, nekā sākotnēji plānoju. SS.LV vietnes struktūra bieži mainās, kas prasīja izveidot elastīgu parsēšanas algoritmu. Īpaši izaicinājums bija Tesla modeļu apstrāde, jo to sludinājumi SS.LV ir strukturēti citādi nekā tradicionālo automašīnu sludinājumi.

4.2.2. Statistiskās analīzes modulis

Statistiskās analīzes modulis ir atbildīgs par sludinājumu datu statistisko apstrādi un analīzi, lai iegūtu noderīgu informāciju par automašīnu tirgus tendencēm.



4.4.att. Statistiskās analīzes procesa datu plūsmas diagramma

Statistiskās analīzes process notiek šādi:

1. **Parametru validācija:** Lietotājs ievada analīzes parametrus caur lietotāja saskarni: automašīnas marku, modeli, gadu diapazonu, reģionu, dzinēja tipu.

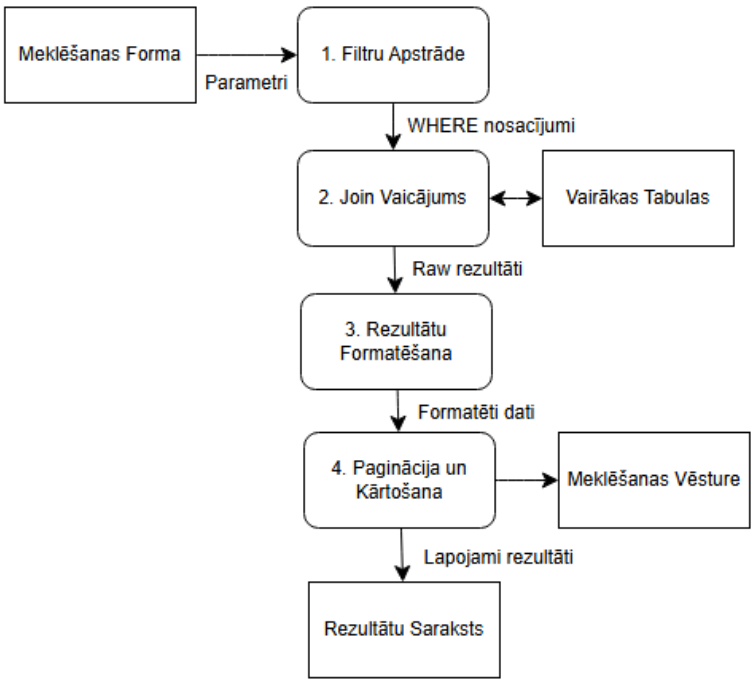
Sistēma validē ievadītos filtrus, pārbaudot to formātu un pieļaujamās vērtības. Validēti kritēriji tiek nodoti tālākai apstrādei.

2. **SQL vaicājuma izveide un izpilde:** Balstoties uz validētajiem kritērijiem, sistēma izveido SQL vaicājumu ar WHERE nosacījumiem. Vaicājums tiek izpildīts pret Listings datubāzi, izmantojot JOIN operācijas, lai savienotu nepieciešamās tabulas (cars, models, brands, listings). Tiek atlasīti tikai aktīvie sludinājumi, kas atbilst norādītajiem filtriem.
3. **Datu apstrāde:** Izmantojot NumPy funkcijas, sistēma aprēķina galvenos statistiskos rādītājus: vidējo cenu (np.mean()), mediānas cenu (np.median()), minimālo un maksimālo cenu, standarta novirzi (np.std()). Tiek veikta arī reģionālā analīze, izmantojot SQL GROUP BY operācijas un aprēķinot katram reģionam atsevišķas statistikas.
4. **Statistikas aprēķini:** Izmantojot NumPy, sistēma aprēķina galvenos statistiskos rādītājus: vidējo cenu (np.mean()), mediānas cenu (np.median()), minimālo un maksimālo cenu, standarta novirzi (np.std()). Tiek veikta arī reģionālā analīze, grupējot datus pēc reģioniem un aprēķinot katram reģionam atsevišķas statistikas. Rezultāti tiek saglabāti MarketValue tabulā kešošanai un JSON formātā tiek nosūtīti klientam.

Statistiskās analīzes modulis pārveido jēldatus par vērtīgu informāciju, kas palīdz lietotājiem pieņemt informētus lēmumus par automašīnu pirkšanu vai pārdošanu.

4.2.3. Meklēšanas un filtrēšanas modulis

Meklēšanas un filtrēšanas modulis nodrošina lietotājam iespēju atrast un filtrēt automašīnu sludinājumus pēc dažādiem kritērijiem.



4.5.att. Meklēšanas procesa datu plūsmas diagramma

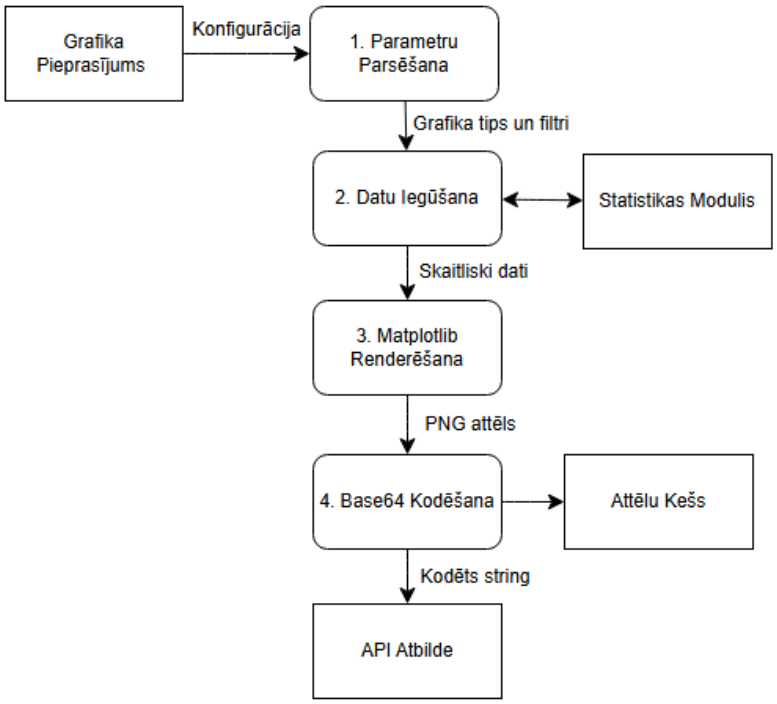
Meklēšanas un filtrēšanas process notiek šādi:

1. **Filtru apstrāde:** Lietotājs ievada meklēšanas kritērijus caur meklēšanas formu: marku, modeli, gadu un cenu diapazonu, dzinēja tipu, ātrumkārbu, reģionu. Sistēma validē un apstrādā šos parametrus, pārvēršot tos SQL WHERE nosacījumos. Tiek pārbaudīti datu tipi, diapazonu robežas un pieļaujamās vērtības.
2. **JOIN vaicājuma izveide:** Balstoties uz apstrādātajiem filtriem, sistēma izveido kompleksu SQL vaicājumu ar INNER JOIN operācijām starp cars, models, brands, listings un regions tabulām. Vaicājums tiek optimizēts, izmantojot datubāzes indeksus ātrākai izpildei. Tiek pievienoti ORDER BY un LIMIT/OFFSET nosacījumi kārtošanai un paginācijai.
3. **Rezultātu formatēšana:** No datubāzes iegūtie raw rezultāti tiek transformēti uz lietotājam draudzīgu formātu. Cenas tiek formatētas ar valūtas simboliem (€15,000), dzinēja tipa kodi tiek konvertēti uz lasāmu tekstu ("P" uz "Benzīns"), tiek aprēķināts sludinājuma vecums dienās un pievienoti papildu metadati.
4. **Paginācija un kārtošana:** Formatētie rezultāti tiek sakārtoti pēc lietotāja izvēlētā kritērija (cena, gads, nobraukums) augošā vai dilstošā secībā. LIMIT un OFFSET tiek piemēroti paginācijas nodrošināšanai (10 rezultāti lapā). Ja lietotājs ir autentificēts, meklēšanas parametri tiek saglabāti meklēšanas vēsturē.

Meklēšanas modulis nodrošina ātru un precīzu sludinājumu atlasi no liela datu apjoma, izmantojot optimizētus SQL vaicājumus un efektīvu rezultātu formatēšanu.

4.2.4. Vizualizāciju ģenerēšanas modulis

Vizualizāciju ģenerēšanas modulis ir atbildīgs par grafiku un diagrammu veidošanu, kas attēlo automašīnu cenu statistiskos datus un tendences.



4.6.att. Grafiku ģenerēšanas procesa datu plūsmas diagramma

Vizualizāciju ģenerēšanas process notiek šādi:

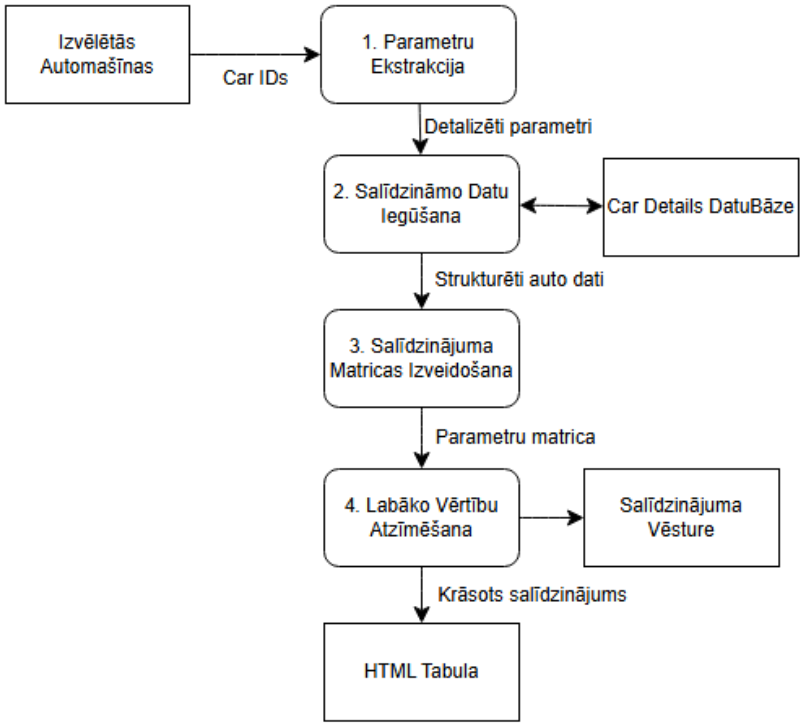
1. **Parametru parsēšana:** Lietotājs izvēlas grafika tipu (cenu sadalījums vai vidējās cenas pa reģioniem) un norāda analīzes parametrus caur grafika pieprasījumu. Sistēma parsē konfigurāciju, nosakot nepieciešamo datu tipu, grafika dimensijas un stilizācijas opcijas.
2. **Datu iegūšana:** Balstoties uz parsētajiem parametriem, sistēma izsauc statistikas moduli, lai iegūtu nepieciešamos skaitliskos datus. Cenu sadalījuma grafikam tiek iegūts cenu masīvs, bet reģionālajam grafikam - vidējās cenas pa reģioniem ar sludinājumu skaitu.
3. **Matplotlib renderēšana:** Izmantojot matplotlib.pyplot, sistēma izveido figure(10,6) ar atbilstošu grafika tipu. Histogrammām tiek izmantots plt.hist() ar 20 bins un kde=True blīvuma līknei, stabiņu diagrammām - plt.bar(). Tiek pievienoti plt.xlabel(), plt.ylabel(), plt.title() un seaborn stilizācija skaidrai vizualizācijai.
4. **Base64 kodēšana:** Ģenerētais PNG grafiks tiek saglabāts BytesIO buffer ar plt.savefig(format='png', dpi=100). Attēla saturs tiek kodēts base64.b64encode()

formātā web pārsūtīšanai. Kodētais string tiek nosūtīts klientam kopā ar grafika metadatiem API atbildē.

Vizualizāciju modulis pārveido sarežģītus statistiskos datus viegli uztveramās vizualizācijās, kas palīdz lietotājiem ātri identificēt tirgus tendences un cenu klasterus.

4.2.5. Automašīnu salīdzināšanas modulis

Automašīnu salīdzināšanas modulis ļauj lietotājiem salīdzināt vairākas automašīnas, lai atvieglotu lēmumu pieņemšanu.



4.7.att. Salīdzināšanas procesa datu plūsmas diagramma

Automašīnu salīdzināšanas process notiek šādi:

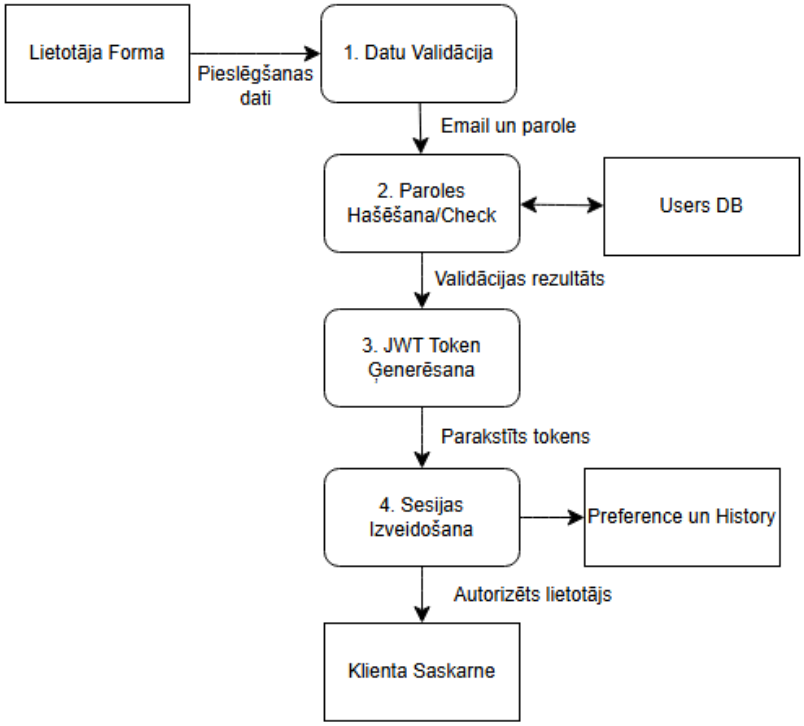
1. **Parametru ekstrakcija:** Lietotājs izvēlas līdz 3 automašīnas no meklēšanas rezultātiem, nosūtot to car_id sarakstu salīdzināšanas modulim. Sistēma validē, ka nepārsniedz maksimālo skaitu un ka visi ID eksistē datubāzē.
2. **Salīdzināmo datu iegūšana:** Katram car_id sistēma veic detalizētu SQL vaicājumu, iegūstot pilnu informāciju no car details datubāzes: cenu, gadu, dzinēja parametrus, ātrumkārbu, nobraukumu, virsbūves tipu, krāsu, tehnisko apskati un reģionu. Dati tiek strukturēti objektos ērtākai salīdzināšanai.
3. **Salīdzinājuma matricas izveidošana:** Iegūtie auto dati tiek organizēti salīdzinājuma matricā, kur rindas reprezentē parametrus (cena, gads, dzinējs, utt.), bet kolonnas - salīdzināmās automašīnas. Trūkstošie dati tiek aizstāti ar "Nav norādīts" vērtībām.
4. **Labāko vērtību atzīmēšana:** Algoritms iterē caur katru parametru, identificējot optimālās vērtības: min(cenas) zemākajā cenā, max(gadi) jaunākajā gadā,

min(nobraukums) mazākajā nobraukumā. Labākās vērtības tiek atzīmētas ar CSS klasi 'best-value' zaļai krāsai. Salīdzinājuma rezultāts tiek saglabāts lietotāja sesijā vai salīdzinājuma vēsturē.

Salīdzināšanas modulis vizuāli organizē automašīnu parametrus, ļaujot lietotājiem ātri identificēt katras automašīnas stiprās un vājās puses lēmumu pieņemšanas atvieglošanai.

4.2.6. Lietotāju autentifikācijas modulis

Lietotāju autentifikācijas modulis nodrošina lietotāju reģistrāciju, pieslēgšanos sistēmai un profilu pārvaldību.



4.8.att. Autentifikācijas procesa datu plūsmas diagramma

Lietotāju autentifikācijas process notiek šādi:

1. **Datu validācija:** Lietotājs ievada pieslēgšanās datus (e-pastu un paroli) caur lietotāja formu. Sistēma validē e-pasta formātu pēc RFC 5322 standarta, pārbauda paroles garumu (min. 8 simboli) un nepieciešamo rakstu zīmju klātbūtni. Ievades dati tiek sanitizēti SQL injekciju novēršanai.
2. **Paroles hašēšana/pārbaude:** Reģistrācijas gadījumā sistēma ģenerē unikālu salt vērtību un izmanto bcrypt.hashpw() paroles hašēšanai. Pieslēgšanās gadījumā tiek iegūts saglabātais hash no Users datubāzes un veikts bcrypt.checkpw() salīdzinājums ar ievadīto paroli. Neveiksmīgi mēģinājumi tiek skaitīti un ierobežoti (5 mēģinājumi uz 30 minūtēm).
3. **JWT token ģenerēšana:** Veiksmīgas autentifikācijas gadījumā sistēma ģenerē JWT token ar jwt.encode(), iekļaujot user_id, username un exp (derīguma termiņš

24 stundas). Token tiek parakstīts ar SECRET_KEY un nosūtīts klientam HTTP header vai response body.

4. **Sesijas izveidošana:** Autorizētam lietotājam sistēma ielādē viņa preferences no datubāzes (tumšais režīms, valoda, paziņojumu iestatījumi) un inicializē sesiju. Tiek atjaunināts last_login lauks Users tabulā un sagatavoti preference & history dati klienta saskarnei. JWT token tiek saglabāts localStorage pārlūkprogrammā turpmākajiem API pieprasījumiem.

Autentifikācijas modulis nodrošina drošu lietotāju identitātes pārbaudi un personalizētu sesiju izveidošanu, izmantojot mūsdienīgus kriptogrāfijas standartus.

5. DATU STRUKTŪRU APRAKSTS

Šajā sadaļā tiek aprakstīta sistēmā izmantojamo datu organizācija loģiskā līmenī. Automobiļu tirgus analīzes sistēma izmanto SQLite datubāzi, kas nodrošina efektīvu datu uzglabāšanu un piekļuvi.

5.1. Datu bāzes struktūras apraksts

Automobiļu tirgus analīzes sistēmas datubāze sastāv no divām galvenajām daļām:

1. Galvenā sistēmas datubāze - satur informāciju par automašīnām, sludinājumiem, analīzēm un citiem ar sistēmas pamatfunkcionalitāti saistītiem datiem.
2. Autentifikācijas datubāze - satur informāciju par lietotājiem, viņu preferencēm, izlasi un meklēšanas vēsturi.

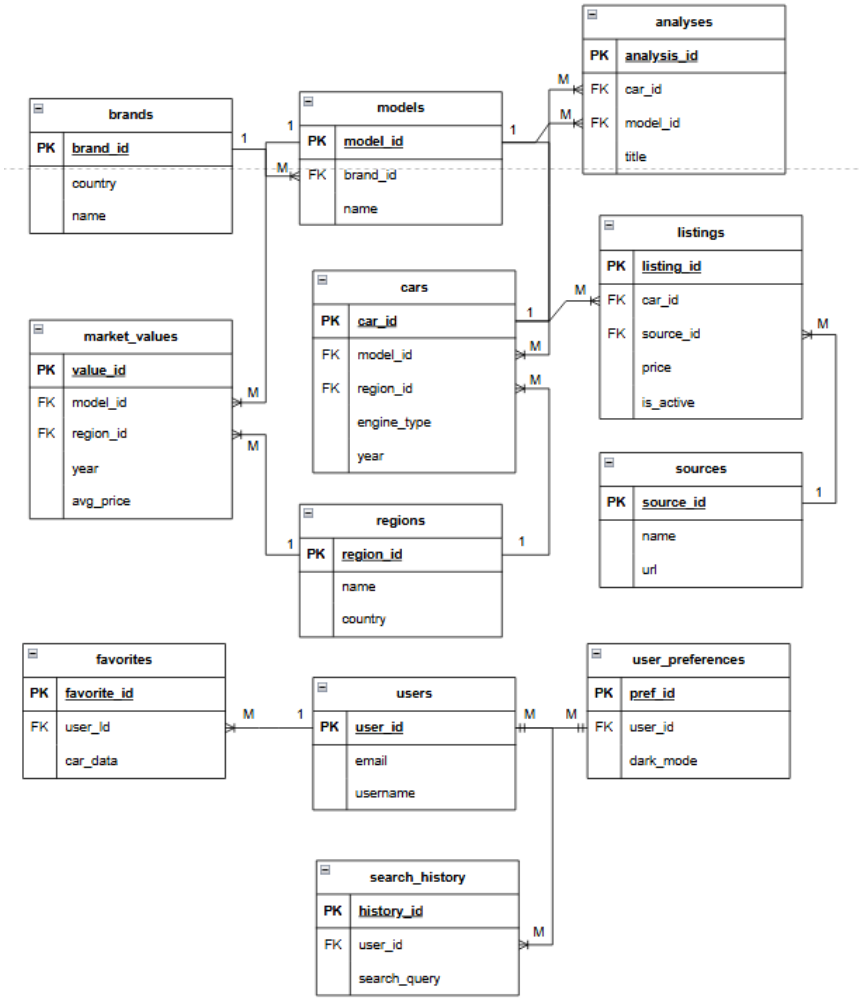
Galvenā sistēmas datubāze sastāv no 9 galvenajām tabulām:

1. brands - satur informāciju par automašīnu markām.
2. models - satur informāciju par automašīnu modeļiem.
3. regions - satur informāciju par ģeogrāfiskajiem reģioniem.
4. cars - satur informāciju par konkrētām automašīnām.
5. sources - satur informāciju par datu avotiem.
6. listings - satur informāciju par sludinājumiem.
7. analyses - satur informāciju par veiktajām analīzēm.
8. market_values - satur informāciju par aprēķinātajām tirgus vērtībām.
9. reports un report_analyses - satur informāciju par ģenerētajiem pārskatiem.

Autentifikācijas datubāze sastāv no 4 galvenajām tabulām:

1. users - satur informāciju par lietotājiem.
2. user_preferences - satur lietotāju preferences.
3. favorites - satur lietotāju izlases automašīnas.
4. search_history - satur lietotāju meklēšanas vēsturi.

Sistēmas datubāzes struktūra ir parādīta tabulu saišu shēmā (skat. 5.1. att.), kas atspoguļo galvenās entītijas un to savstarpējās relācijas.



5.1.att. Tabulu saišu shēma

Kā redzams tabulu saišu shēmā, starp tabulām ir izveidotas vairākas saites, kas nodrošina datu integritāti un efektīvu datu pārvaldību. Galvenās tabulu saites ir:

- brands un models (1:N) - viena marka var būt saistīta ar daudziem modeļiem.
- models un cars (1:N) - viens modelis var būt saistīts ar daudzām automašīnām.
- regions un cars (1:N) - viens reģions var būt saistīts ar daudzām automašīnām.
- cars un listings (1:N) - viena automašīna var būt saistīta ar daudziem sludinājumiem.
- sources un listings (1:N) - viens avots var būt saistīts ar daudziem sludinājumiem.
- cars un analyses (1:N) - viena automašīna var būt saistīta ar daudzām analīzēm.
- models un analyses (1:N) - viens modelis var būt saistīts ar daudzām analīzēm.
- models un market_values (1:N) - viens modelis var būt saistīts ar daudzām tirgus vērtībām.
- regions un market_values (1:N) - viens reģions var būt saistīts ar daudzām tirgus vērtībām.

5.2. Galveno tabulu apraksts

Šajā sadaļā tiek detalizēti aprakstītas galvenās datubāzes tabulas, to lauki un saites ar citām tabulām. Visas tabulas satur standarta created_at un updated_at laukus laika atzīmēšanai.

Tabula “brands”satur informāciju par automašīnu markām.

5.1.Tabula

Tabulas **“brands”** struktūra

Nr	Lauka nosaukums	Datu tips	Garums	Apraksts
1.	brand_id	INTEGER	-	Unikāls markas identifikators(primārā atslēga)
2.	name	STRING	30	Markas nosaukums(piemēram,”Tesla”,”Infiniti”)
3.	country	STRING	30	Markas izcelsmes valsts(piemēram,”Japāna”,”ASV”)
4.	logo_url	STRING	255	URL uz markas logotipu
5.	created_at	DATETIME	-	Ieraksta izveidošanas datums un laiks
6.	updated_at	DATETIME	-	Ieraksta pēdējās atjaunināšanas datums un laiks

Tabula “brands” ir saistīta ar tabulu “models” (1:N). Viena marka var būt saistīta ar daudziem modeļiem.

Tabula “models”satur informāciju par automašīnu modeļiem.

5.2. Tabula

Tabula **“models”** struktūra

Nr	Lauka nosaukums	Datu tips	Garums	Apraksts	Piezīmes
1.	model_id	INTEGER	-	Unikāls modeļa identifikators	Primārā atslēga
2.	brand_id	INTEGER	-	Saite uz marku	Ārējā atslēga uz tabulu “brands”
3.	name	STRING	30	Modeļa nosaukums	Piemēram, “Model S”, “QX80”
4.	class_type	STRING	20	Transportlīdzekļa klases tips	Piemēram, “Sedans”, “SUV”
5.	production_start	INTEGER	-	Ražošanas sākuma gads	-
6.	production_end	INTEGER	-	Ražošanas beigu gads	NULL, ja joprojām ražo
7.	created_at	DATETIME	-	Ieraksta izveidošanas datums un laiks	-
8.	updated_at	DATETIME	-	Ieraksta pēdējās atjaunināšanas datums un laiks	-

Tabula “models” ir saistīta ar:

- Tabulu “brands” (N:1) - daudzi modeļi var būt saistīti ar vienu marku.
- Tabulu “cars” (1:N) - viens modelis var būt saistīts ar daudzām automašīnām.
- Tabulu “market_values” (1:N) - viens modelis var būt saistīts ar daudzām tirgus vērtībām.
- Tabulu “analyses” (1:N) - viens modelis var būt saistīts ar daudzām analīzēm.

Tabula “regions”satur informāciju par ģeogrāfiskajiem reģioniem.

5.3. Tabula

Tabula **“regions”** struktūra

Nr	Lauka nosaukums	Datu tips	Garums	Apraksts
1.	region_id	INTEGER	-	Unikāls reģiona identifikators(Primārā atslēga)
2.	name	STRING	50	Reģiona nosaukums(Piemēram, “Rīga”, “Vidzeme”)
3.	country	STRING	30	Valsts nosaukums(Piemēram, “Latvija”)
4.	lat	FLOAT	-	Reģiona centra ģeogrāfiskais platums
5.	lng	FLOAT	-	Reģiona centra ģeogrāfiskais garums
6.	created_at	DATETIME	-	Ieraksta izveidošanas datums un laiks
7.	updated_at	DATETIME	-	Ieraksta pēdējās atjaunināšanas datums un laiks
8.				

Tabula “regions” ir saistīta ar:

- Tabulu “cars” (1:N) - viens reģions var būt saistīts ar daudzām automašīnām.
- Tabulu “market_values” (1:N) - viens reģions var būt saistīts ar daudzām tirgus vērtībām.

Tabula “cars” satur informāciju par konkrētām automašīnām.

5.4. Tabula

Tabula **“cars”** struktūra

Nr	Lauka nosaukums	Datu tips	Garums	Apraksts	Piezīmes
1.	car_id	INTEGER	-	Unikāls automašīnas identifikators	Primārā atslēga
2.	model_id	INTEGER	-	Saite uz modeli	Ārējā atslēga uz tabulu “models”
3.	region_id	INTEGER	-	Saite uz reģionu	Ārējā atslēga uz tabulu “regions”
4.	year	INTEGER	-	Automašīnas izlaiduma gads	Piemēram, 2020
5.	engine_volume	FLOAT	-	Dzinēja tilpums litros	Piemēram, 2.0
6.	engine_type	STRING	20	Dzinēja tips	Piemēram, “Benzīns”, “Dīzelis”
7.	transmission	STRING	20	Ātrumkārbas tips	Piemēram, “Automātiska”, “Manuāla”
8.	tech_inspection	STRING	20	Tehniskās apskates statuss	-
9.	mileage	INTEGER	-	Nobraukums kilometros	-
10.	body_type	STRING	20	Virsbūves tips	Piemēram, “Sedans”, “Universālis”
11.	color	STRING	20	Automašīnas krāsa	-

Tabula “cars” ir saistīta ar:

- Tabulu “models” (N:1) - daudzas automašīnas var būt saistītas ar vienu modeli.
- Tabulu “regions” (N:1) - daudzas automašīnas var būt saistītas ar vienu reģionu.
- Tabulu “listings” (1:N) - viena automašīna var būt saistīta ar daudziem sludinājumiem.
- Tabulu “analyses” (1:N) - viena automašīna var būt saistīta ar daudzām analīzēm.

Tabula **“**sources” satur informāciju par datu avotiem.

5.5. Tabula

Tabula **“sources”** struktūra

Nr	Lauka nosaukums	Datu tips	Garums	Apraksts	Piezīmes
1.	source_id	INTEGER	-	Unikāls avota identifikators	Primārā atslēga
2.	name	STRING	50	Avota nosaukums	Piemēram, “SS.LV”
3.	url	STRING	255	Avota URL	Piemēram, “https://www.ss.lv“
4.	country	STRING	30	Valsts, kurā darbojas avots	Piemēram, “Latvija”
5.	scraping_config	TEXT	-	JSON konfigurācija datu iegūšanai	-
6.	last_scraped_at	DATETIME	-	Pēdējās datu iegūšanas datums un laiks	-

Tabula “sources” ir saistīta ar tabulu “listings” (1:N). Viens avots var būt saistīts ar daudziem sludinājumiem.

Tabula “listings” satur informāciju par automašīnu sludinājumiem.

5.6. Tabula

Tabula **“listings”** struktūra

Nr	Lauka nosaukums	Datu tips	Garums	Apraksts	Piezīmes
1.	listing_id	INTEGER	-	Unikāls sludinājuma identifikators	Primārā atslēga
2.	car_id	INTEGER	-	Saite uz automašīnu	Ārējā atslēga uz tabulu “cars”
3.	source_id	INTEGER	-	Saite uz avotu	Ārējā atslēga uz tabulu “sources”
4.	external_id	STRING	50	Sludinājuma identifikators avota vietnē	-
5.	price	INTEGER	-	Cena eiro	-
6.	listing_date	DATE	-	Sludinājuma publicēšanas datums	-
7.	listing_url	STRING	255	URL uz oriģinālo sludinājumu	-
8.	is_active	BOOLEAN	-	Vai sludinājums joprojām ir aktīvs	True/False

Tabula “listings” ir saistīta ar:

- Tabulu “cars” (N:1) - daudzi sludinājumi var būt saistīti ar vienu automašīnu.
- Tabulu “sources” (N:1) - daudzi sludinājumi var būt saistīti ar vienu avotu.

Tabula satur “analyses” informāciju par veiktajām analīzēm.

5.7. Tabula

Tabula **“analyses”** struktūra

Nr	Lauka nosaukums	Datu tips	Garums	Apraksts	Piezīmes
1.	analysis_id	INTEGER	-	Unikāls analīzes identifikators	Primārā atslēga
2.	car_id	INTEGER	-	Saite uz automašīnu	Ārējā atslēga uz tabulu “cars”, var būt NULL
3.	model_id	INTEGER	-	Saite uz modeli	Ārējā atslēga uz tabulu “models”, var būt NULL
4.	title	STRING	100	Analīzes nosaukums	-
5.	description	TEXT	-	Analīzes apraksts	-
6.	params	TEXT	-	JSON parametri	-
7.	results	TEXT	-	JSON rezultāti	-

Tabula “analyses” ir saistīta ar:

- Tabulu “cars” (N:1) - daudzas analīzes var būt saistītas ar vienu automašīnu.
- Tabulu “models” (N:1) - daudzas analīzes var būt saistītas ar vienu modeli.
- Tabulu “report_analyses” (1:N) - viena analīze var būt saistīta ar daudzām pārskatu analīzēm.

Tabula satur “market_values” informāciju par aprēķinātajām tirgus vērtībām.

5.8. Tabula

Tabula **“market_values”** struktūra

Nr	Lauka nosaukums	Datu tips	Garums	Apraksts	Piezīmes
1.	value_id	INTEGER	-	Unikāls vērtības identifikators	Primārā atslēga
2.	model_id	INTEGER	-	Saite uz modeli	Ārējā atslēga uz tabulu “models”
3.	region_id	INTEGER	-	Saite uz reģionu	Ārējā atslēga uz tabulu “regions”
4.	year	INTEGER	-	Transportlīdzekļa gads	-
5.	engine_type	STRING	20	Dzinēja tips	-
6.	avg_price	INTEGER	-	Vidējā cena	-
7.	median_price	INTEGER	-	Mediānas cena	-
8.	min_price	INTEGER	-	Minimālā cena	-
9.	max_price	INTEGER	-	Maksimālā cena	-
10.	std_deviation	FLOAT	-	Standarta novirze	-
11.	sample_size	INTEGER	-	Paraugu skaits	-
12.	calculation_date	DATE	-	Aprēķina datums	-

Tabula “market_values” ir saistīta ar:

- Tabulu “models” (N:1) - daudzas tirgus vērtības var būt saistītas ar vienu modeli.
- Tabulu “regions” (N:1) - daudzas tirgus vērtības var būt saistītas ar vienu reģionu.

Autentifikācijas datubāzes tabulas

Autentifikācijas datubāze sastāv no šādām galvenajām tabulām:

5.9. Tabula

Tabula **“users”** struktūra

Nr	Lauka nosaukums	Datu tips	Garums	Apraksts	Piezīmes
1.	user_id	INTEGER	-	Unikāls lietotāja identifikators	Primārā atslēga
2.	email	TEXT	-	Lietotāja e-pasts	Unikāls(pārbaudīts aplikācijas līmenī)
3.	username	TEXT	-	Lietotājvārds	Unikāls
4.	password_hash	TEXT	-	Hašētā parole	-
5.	salt	TEXT	-	Paroles sāls	-
6.	created_at	TIMESTAMP	-	Konta izveidošanas laiks	-
7.	last_login	TIMESTAMP	-	Pēdējās pieslēgšanās laiks	-
8.	is_active	BOOLEAN	-	Konta aktīvais statuss	True/False

5.10. Tabula

Tabula **“user_preferences”** struktūra

Nr	Lauka nosaukums	Datu tips	Garums	Apraksts	Piezīmes
1.	pref_id	INTEGER	-	Unikāls preferenču identifikators	Primārā atslēga
2.	user_id	INTEGER	-	Saite uz lietotāju	Ārējā atslēga uz tabulu “users”
3.	dark_mode	BOOLEAN	-	Tumšā režīma preference	True/False
4.	notification_enabled	BOOLEAN	-	Paziņojumu preferences	True/False

5.11. Tabula

Tabula **“favorites”** struktūra

Nr	Lauka nosaukums	Datu tips	Garums	Apraksts	Piezīmes
1.	favorite_id	INTEGER	-	Unikāls izlases identifikators	Primārā atslēga
2.	user_id	INTEGER	-	Saite uz lietotāju	Ārējā atslēga uz tabulu “users”
3.	car_data	TEXT	-	JSON automašīnas dati	-
4.	added_at	TIMESTAMP	-	Pievienošanas laiks	-

5.12. Tabula

Tabula **“search_history”** struktūra

Nr	Lauka nosaukums	Datu tips	Garums	Apraksts	Piezīmes
1.	history_id	INTEGER	-	Unikāls vēstures identifikators	Primārā atslēga
2.	user_id	INTEGER	-	Saite uz lietotāju	Ārējā atslēga uz tabulu “users”
3.	search_query	TEXT	-	JSON meklēšanas parametri	-
4.	executed_at	TIMESTAMP	-	Meklēšanas izpildes laiks	-

5.2.1 Tabulu savstarpējās saites

Tabulu savstarpējās saites nodrošina datu integritāti un ļauj efektīvi piekļūt saistītajiem datiem. Šajā sadaļā tiek detalizēti aprakstītas galvenās saites starp tabulām.

Saites galvenajā sistēmas datubāzē

1. Saite starp brands un models:
   - Saites tips: One to Many (1:N)
   - Saites lauki: models.brand_id -> brands.brand_id
   - Apraksts: Viena marka var būt saistīta ar daudziem modeļiem.
2. Saite starp models un cars:
   - Saites tips: One to Many (1:N)
   - Saites lauki: cars.model_id -> models.model_id
   - Apraksts: Viens modelis var būt saistīts ar daudzām automašīnām.

3. Saite starp regions un cars:
    - Saites tips: One to Many (1:N)
    - Saites lauki: cars.region_id -> regions.region_id
    - Apraksts: Viens reģions var būt saistīts ar daudzām automašīnām.
4. Saite starp cars un listings:
    - Saites tips: One to Many (1:N)
    - Saites lauki: listings.car_id -> cars.car_id
    - Apraksts: Viena automašīna var būt saistīta ar daudziem sludinājumiem.
5. Saite starp sources un listings:
    - Saites tips: One to Many (1:N)
    - Saites lauki: listings.source_id -> sources.source_id
    - Apraksts: Viens avots var būt saistīts ar daudziem sludinājumiem.
6. Saite starp cars un analyses:
    - Saites tips: One to Many (1:N)
    - Saites lauki: analyses.car_id -> cars.car_id
    - Apraksts: Viena automašīna var būt saistīta ar daudzām analīzēm.
7. Saite starp models un analyses:
    - Saites tips: One to Many (1:N)
    - Saites lauki: analyses.model_id -> models.model_id
    - Apraksts: Viens modelis var būt saistīts ar daudzām analīzēm.
8. Saite starp models un market_values:
    - Saites tips: One to Many (1:N)
    - Saites lauki: market_values.model_id -> models.model_id
    - Apraksts: Viens modelis var būt saistīts ar daudzām tirgus vērtībām.
9. Saite starp regions un market_values:
    - Saites tips: One to Many (1:N)
    - Saites lauki: market_values.region_id -> regions.region_id
    - Apraksts: Viens reģions var būt saistīts ar daudzām tirgus vērtībām.

Saites autentifikācijas datubāzē

1. Saite starp users un user_preferences:
    - Saites tips: One to One (1:1)
    - Saites lauki: user_preferences.user_id -> users.user_id
    - Apraksts: Vienam lietotājam ir viena preference ieraksts.
2. Saite starp users un favorites:
    - Saites tips: One to Many (1:N)
    - Saites lauki: favorites.user_id -> users.user_id
    - Apraksts: Viens lietotājs var būt saistīts ar daudzām izlases automašīnām.
3. Saite starp users un search_history:
    - Saites tips: One to Many (1:N)

- Saites lauki: search_history.user_id -> users.user_id
- Apraksts: Viens lietotājs var būt saistīts ar daudziem meklēšanas vēstures ierakstiem.

Šī datu struktūra ir veidota tā, lai nodrošinātu efektīvu datu glabāšanu un piekļuvi, vienlaikus saglabājot datu integritāti. SQLite datubāze ir izvēlēta tās viegluma un pārnesamības dēļ, kas ir pietiekami šī projekta vajadzībām.

Sistēmas tehniskā realizācija ir tikai puse no darba - tikpat svarīgi ir nodrošināt, lai tā būtu viegli lietojama un uzstādāma. Nākamajā sadaļā aprakstīšu praktiskos aspektus, kas nepieciešami sistēmas izmantošanai.

6. LIETOTĀJA CEĻVEDIS

Šajā sadaļā sniegts apraksts par sistēmas uzstādīšanu, palaišanu un lietošanu. Lietotāja ceļvedis palīdzēs lietotājam ātri un efektīvi sākt izmantot Automobiļu tirgus analīzes sistēmu.

6.1. Sistēmas prasības aparatūrai un programmatūrai

Lai izmantotu Automobiļu tirgus analīzes sistēmu, ir nepieciešamas šādas minimālās aparatūras un programmatūras prasības:

6.1.1. Servera puses prasības

Aparatūras prasības:

- Procesors: Dual-core 2 GHz vai jaudīgāks
- Operatīvā atmiņa: Vismaz 4 GB RAM
- Cietais disks: Vismaz 1 GB brīvas vietas (atkarībā no datubāzes lieluma)
- Interneta pieslēgums: Stabilais interneta pieslēgums ar ātrumu vismaz 10 Mbps

Programmatūras prasības:

- Operētājsistēma: Windows 10/11, Linux (Ubuntu 20.04 vai jaunāka) vai macOS 10.15 vai jaunāka
- Python: Versija 3.10 vai jaunāka
- Datu bāze: SQLite 3.38.0 vai jaunāka (jau iekļauta Python instalācijā)

Nepieciešamās Python bibliotēkas:

- Flask (2.0.1+)
- SQLAlchemy (1.4.35+)
- BeautifulSoup4 (4.9.3+)
- NumPy (1.22.3+)
- Matplotlib (3.5.1+)
- Seaborn (0.11.2+)
- PyJWT (2.3+)
- Aiohtpps (versija 3.8.0+)

6.1.2. Klienta puses prasības

Aparatūras prasības:

- Jebkura ierīce ar pārlūkprogrammu un interneta pieslēgumu
- Ieteicams: HD izšķirtspējas ekrāns (1366x768 vai augstāka)

Programmatūras prasības:

- Pārlūkprogramma: Google Chrome (versija 90+), Mozilla Firefox (versija 88+), Safari (versija 14+), Microsoft Edge (versija 90+)

6.2. Sistēmas instalācija un palaišana

Šajā sadaļā ir aprakstīts, kā uzstādīt un palaist Automobiļu tirgus analīzes sistēmu. Instalācijas process ir sadalīts vairākos soļos, sākot ar programmatūras lejupielādi un beidzot ar tās palaišanu.

6.2.1. Programmatūras lejupielāde

1. Lejupielādējiet projekta kodu no GitHub repozitorijas un izveidot projektu:
   - git clone https://github.com/EmilsJur/car-price-analysis
   - cd car-price-analysis
2. Alternatīvi var lejupielādēt ZIP arhīvu no GitHub un atarhivējiet to vēlamajā direktorijā.

6.2.2. Atkarību instalēšana

Visas nepieciešamās darbības un atkarību instalēšana ir aprakstīta failā setup.txt:
Lūdzu, sekojiet šīm instrukcijām soli pa solim:

1. Pārliecinieties, ka Jums ir uzstādīts:
   - Python 3.8 vai jaunāks (ieteicams 3.9+),
   - Node.js 14+ un npm 6+,
   - Git.
2. Atveriet komandrindu un izpildiet darbības, kas norādītas setup.txt failā:
   - Projekta klonēšana no GitHub.
   - Python atkarību uzstādīšana ar pip.
   - React (Node.js) atkarību uzstādīšana ar npm.
   - Projekta inicializēšana un palaišana ar python app.py --mode init un npm start.
3. Alternatīvi var arī pip install requirements.txt

6.2.3. Datubāzes inicializācija

1. Palaist datubāzes inicializācijas skriptu:
   - python init_db.py
2. Šis skripts izveidos divas SQLite datubāzes:
   - car_data.db - galvenā sistēmas datubāze
   - auth.db - autentifikācijas datubāze

5.2.4. Sākotnējo datu iegūšana

1. Palaidiet web scraping skriptu, lai iegūtu sākotnējos datus par automašīnu sludinājumiem vai arī atjauninātu esošos:
   - python ss_scraper.py
2. Noklusējumā skripts iegūs datus par šādām automašīnu markām: Tesla, Infiniti, Smart, Suzuki.
3. Šis process var aizņemt līdz 2 minūtēm, atkarībā no jūsu interneta ātruma un pieejamo sludinājumu skaita(Apmēram 1 sludinājums/sekundē).

6.2.5. Sistēmas palaišana

1. Palaidiet terminālī API serveri:

- python app.py

2. Pēc noklusējuma serveris darbosies adresē http://localhost:5000.
3. Atveriet otru terminālu komandrindu un palaidiet frontend izstrādes serveri:
   - cd car-price-analysis-system
   - npm install
   - npm start
4. Frontend daļa būs pieejama adresē http://localhost:3000.
5. Atveriet pārlūkprogrammu un dodieties uz http://localhost:3000, lai sāktu izmantot sistēmu.

6.2.6. Demo versijas piekļuve

Sistēma ir publiski pieejama un gatava demonstrācijai kvalifikācijas eksāmena laikā:

1. Tiešsaistes pieeja:
   - URL: https://car-price-analysis.onrender.com
   - Sistēma ir pilnībā funkcionāla un pieejama bez iepriekšējas konfigurācijas
2. Autentifikācijas opcijas:
   - Reģistrēti lietotāji:
     1. Var izveidot jaunu kontu, izmantojot jebkuru e-pasta adresi
     2. Pilna funkcionalitāte: izlase, meklēšanas vēsture, eksportēšana
   - Viesi (bez autentifikācijas):
     1. Pilna piekļuve meklēšanas un analīzes funkcijām
     2. Ierobežota funkcionalitāte: nav izlases, nav eksportēšanas iespēju
3. Demonstrācijas dati:
   - Sistēmā jau ir ielādēti parauga dati no SS.LV
   - Pieejami populārākie auto zīmoli: Tesla, Infiniti, Smart, Suzuki
   - Vairāk nekā 500+ aktīvu sludinājumu analīzei

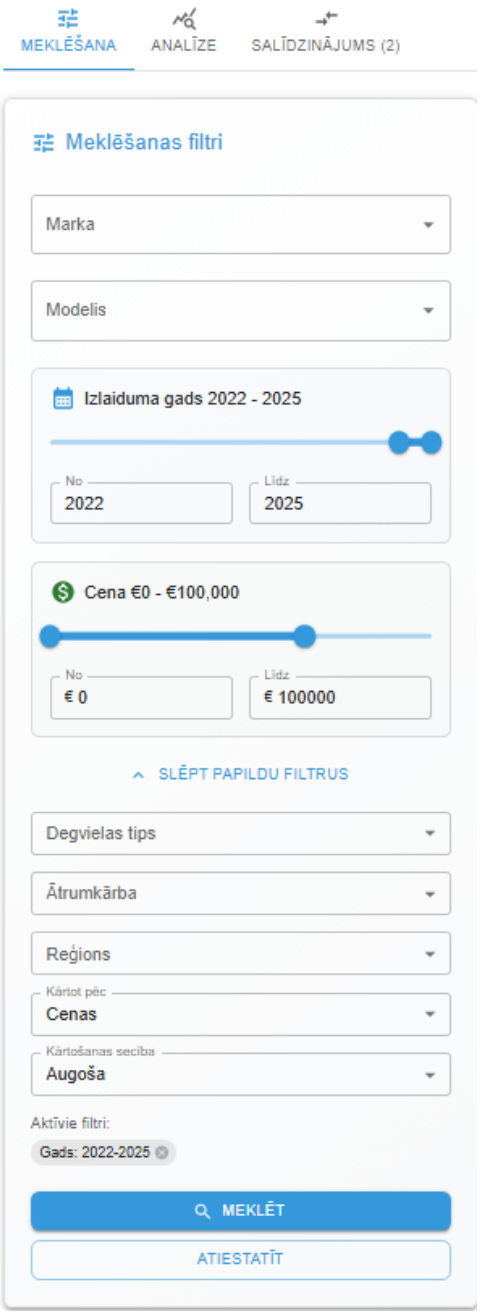
Sistēma šobrīd ir hostēta uz Render.com bezmaksas plāna, tāpēc pirmā pieprasījuma ielāde var aizņemt 30-60 sekundes, ja nav bijusi aktivitāte.

6.3. Programmas apraksts

Šajā sadaļā ir detalizēti aprakstīta Automobiļu tirgus analīzes sistēmas lietotāja saskarne un tās funkcionalitātes.

6.3.1. Meklēšanas sadaļa

Meklēšanas sadaļa ir sistēmas galvenā komponente, kas ļauj lietotājiem atrast automašīnu sludinājumus pēc dažādiem kritērijiem.



6.1. att. Meklēšanas saskarnes ekrāns

Meklēšanas forma

Meklēšanas forma atrodas lapas augšdaļā un satur šādus elementus:

1. Marka - nolaižamais saraksts ar pieejamajām automašīnu markām (Tesla, Infiniti, Smart, Suzuki).

2. Lai izvēlētos marku, klikšķiniet uz saraksta un izvēlieties vajadzīgo.
3. Pēc markas izvēles tiks automātiski atjaunināts modeļu saraksts.
4. Modelis - nolaižamais saraksts ar modeļiem, kas atbilst izvēlētajai markai.
5. Pieejamie modeļi mainās atkarībā no izvēlētās markas.
6. Varat atstāt "Visi modeļi", lai meklētu visus izvēlētās markas modeļus.
7. Gadu diapazons - divi ievades lauki vai slīdnis gadu diapazona norādīšanai:
   - No - minimālais izlaiduma gads (1990-2025).
   - Līdz - maksimālais izlaiduma gads (1990-2025).
8. Cenu diapazons - divi ievades lauki vai slīdnis cenu diapazona norādīšanai:
   - No - minimālā cena eiro (0-1000000).
   - Līdz - maksimālā cena eiro (0-1000000).
9. Papildu filtri (izvēršamā sadaļa):
   - Dzinēja tips - izvēles rūtiņas dažādu dzinēja tipu atlasīšanai (Benzīns, Dīzelis, Hibrīds, Elektriskais).
   - Ātrumkārba - izvēles rūtiņas dažādu ātrumkārbas tipu atlasīšanai (Manuāla, Automātiska).
   - Reģions - nolaižamais saraksts ar Latvijas reģioniem (Rīga, Vidzeme, Kurzeme, utt.).
10. Kārtošanas opcijas:
    - Kārtot pēc - nolaižamais saraksts ar kārtošanas kritērijiem (cena, gads, nobraukums).
    - Secība - pogas augošās vai dilstošās secības izvēlei.
11. Meklēt - poga meklēšanas uzsākšanai.
    Pēc pogas nospiešanas sistēma veiks meklēšanu atbilstoši norādītajiem kritērijiem.

Meklēšanas rezultāti

Meklēšanas rezultāti tiek attēloti tabulā, kas satur šādas kolonnas:

1. Marka un modelis - automašīnas marka un modelis, piemēram, "Tesla Model S".
2. Gads - automašīnas izlaiduma gads, piemēram, "2020".
3. Dzinējs - informācija par dzinēju, piemēram, "Elektriskais".
4. Ātrumkārba - ātrumkārbas tips, piemēram, "Automātiska".
5. Nobraukums - automašīnas nobraukums kilometros, piemēram, "25,000 km".
6. Cena - automašīnas cena eiro, piemēram, "€55,000".
7. Darbības - pogas papildu darbībām:
8. Detaļas - atvērt sludinājuma detalizētu skatu.
9. Salīdzināt - pievienot automašīnu salīdzinājumam.
10. Izlase - pievienot automašīnu izlasei (pieejams tikai autentificētiem lietotājiem).

Ja rezultātu ir daudz, tie tiek sadalīti lapās. Lapas apakšdaļā atrodas paginācijas kontroles elementi, kas ļauj pārvietoties starp rezultātu lapām vai arī palielināt radīto rindu skaitu vienā lapā.

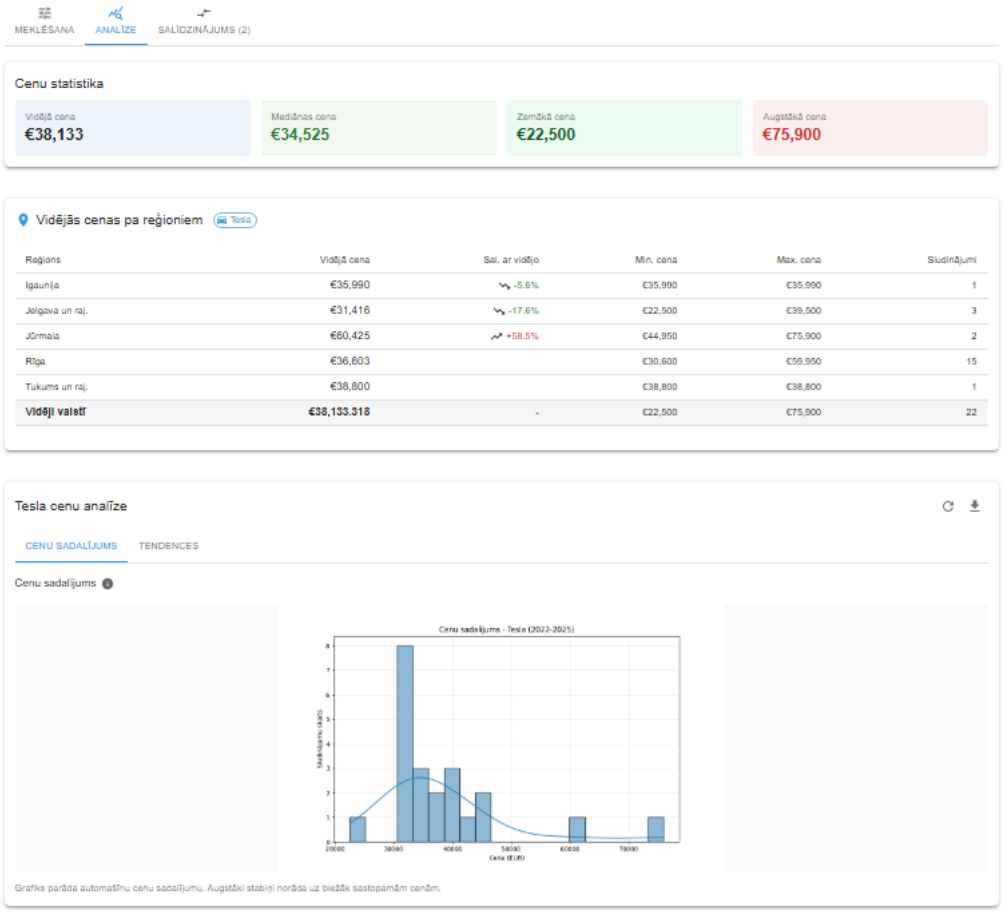
Sludinājuma detalizēts skats

Kad lietotājs noklikšķina uz “Detaļas” pogas, atveras modālais logs ar detalizētu informāciju par izvēlēto automašīnu:

1. Pamatinformācija:
    - Marka un modelis
    - Gads
    - Cena
    - Saite uz oriģinālo sludinājumu SS.LV
2. Tehniskā informācija:
    - Dzinēja tilpums un tips
    - Ātrumkārba
    - Nobraukums
    - Virsbūves tips
    - Krāsa
    - Tehniskā apskate
3. Papildu informācija:
    - Reģions
    - Sludinājuma datums
    - Sludinājuma statuss (aktīvs/neaktīvs)
4. Darbību pogas:
    - Pievienot salīdzinājumam - pievieno automašīnu salīdzinājuma sarakstam.
    - Pievienot izlasei - pievieno automašīnu izlasei (pieejams tikai autentificētiem lietotājiem).
    - Aizvērt - aizver detalizēto skatu.

6.3.2. Analīzes sadaļa

Analīzes sadaļa ļauj lietotājiem detalizētāk izpētīt automašīnu tirgus datus, izmantojot statistikas rādītājus un grafisko vizualizāciju. Kā redzams 14. attēlā, analīzes sadaļa ir sadalīta vairākās daļās.



6.2. att. Analīzes rezultātu ekrāns

Cenu statistikas pārskats

Analīzes sadaļas augšdaļā tiek attēlots statistikas pārskats ar četriem galvenajiem rādītājiem:

- Vidējā cena - aprēķinātā vidējā cena no visiem atrastajiem sludinājumiem (Piemēram, €14,295)
- Mediānas cena - mediānas vērtība, kas dala sludinājumus uz pusēm (Piemēram, €14,295)
- Zemākā cena - minimālā cena starp atrastajiem sludinājumiem (Piemēram, €11,340)
- Augstākā cena - maksimālā cena starp atrastajiem sludinājumiem (Piemēram, €17,250)

Šie rādītāji tiek automātiski aprēķināti, balstoties uz iepriekš veiktās meklēšanas rezultātiem.

Vidējās cenas pa reģioniem

Zem statistikas rādītājiem atrodas detalizēta tabula “Vidējās cenas pa reģioniem”, kas parāda:

- Reģions - Latvijas reģiona nosaukums
- Vidējā cena - aprēķinātā vidējā cena konkrētajā reģionā
- Salīdzinājums ar vidējo - procentuālā atšķirība no kopējās vidējās cenas (piemēram, +20.7% vai -20.7%)
- Min. cena un Maks. cena - cenu diapazons reģionā
- Sludinājumu skaits - pieejamo sludinājumu skaits katrā reģionā

Tabula ļauj ātri identificēt, kuros reģionos automašīnas ir dārgākas vai lētākas par valsts vidējo.

Tirgus cenu analīze

Analīzes sadaļas apakšdaļā atrodas grafiskā analīze ar divām cilnēm:

1. Cenu sadalījums:
   - Histogramma, kas parāda automašīnu cenu sadalījumu pa intervāliem
   - X ass attēlo cenu diapazonus (piemēram, €10,000-€12,000, €12,000-€14,000)
   - Y ass parāda sludinājumu skaitu katrā cenu intervālā
   - Ziļa blīvuma līkne uzrāda cenu sadalījuma tendenci
   - Grafika nosaukums norāda konkrētos meklēšanas parametrus
2. Tendences:
   - Cenu izmaiņu analīze laika gaitā (šī funkcionalitāte tiks ieviesta nākotnē)

Grafiku eksportēšana

Analīzes sadaļā ir pieejamas eksportēšanas iespējas:

- PNG eksports - saglabā grafiku kā attēlu
- CSV eksports - eksportē statistikas datus tabulas formātā turpmākai analīzei

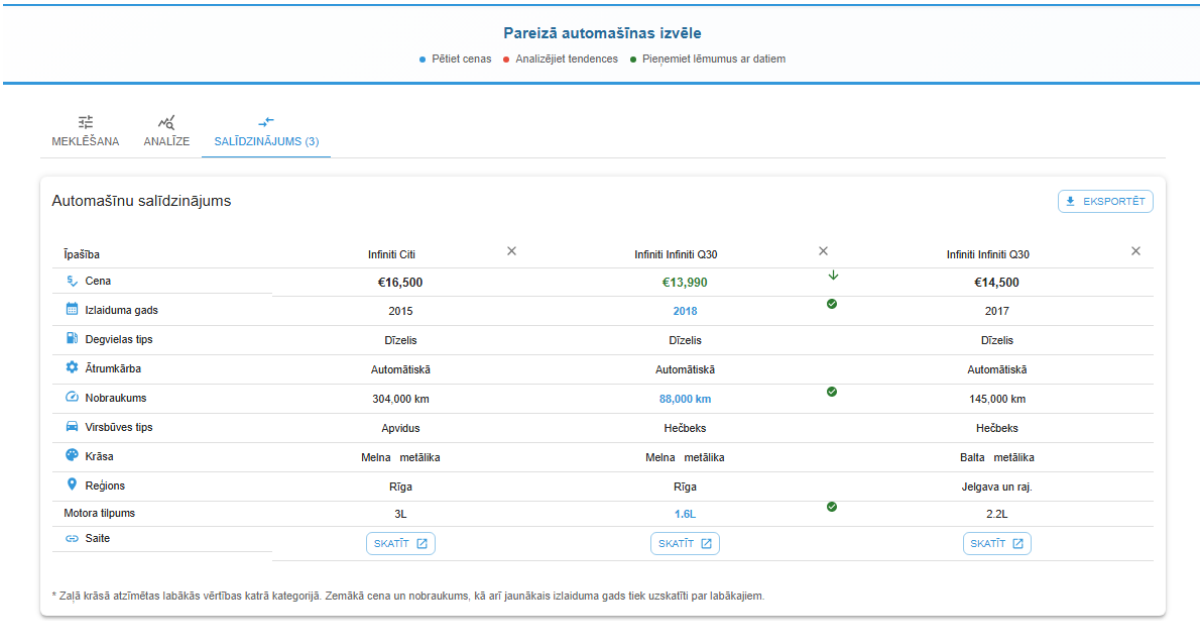
Lietotājs var lejupielādēt gan vizualizācijas, gan neapstrādātus datus citiem rīkiem.

Datu atjaunināšana

Analīzes dati tiek automātiski atjaunināti, kad lietotājs maina meklēšanas parametrus galvenajā meklēšanas sadaļā. Tas nodrošina, ka statistikas rādītāji un grafiki vienmēr atbilst pašreizējiem meklēšanas kritērijiem

6.3.3. Salīdzinājuma sadaļa

Salīdzinājuma sadaļa ļauj lietotājiem salīdzināt līdz 3 automašīnām vienlaikus, lai atvieglotu lēmumu pieņemšanu.



6.3. att. Salīdzināšanas ekrāns

Salīdzinājuma tabula

Galvenais elements salīdzinājuma sadaļā ir salīdzinājuma tabula, kurā rindās ir automašīnu parametri, bet kolonnās - salīdzināmās automašīnas:

1. Parametru kolonnas (rindas):
   - Cena - automašīnas cena eiro.
   - Marka un modelis - automašīnas marka un modelis.
   - Gads - automašīnas izlaiduma gads.
   - Degvielas tips – dīzelis, elektrisks, benzīns, utt.
   - Ātrumkārba - ātrumkārbas tips.
   - Nobraukums - automašīnas nobraukums kilometros.
   - Virsbūves tips - sedans, universālis, hečbeks, utt.
   - Krāsa - automašīnas krāsa.
   - Motora tilpums – 3L, 2.2L, utt.
   - Saite – ss.lv saite sludinājumam.
   - Reģions – kurā rajonā vai pilsētā automašīna atrodas.
2. Automašīnu kolonnas (1-3 kolonnas, atkarībā no izvēlēto automašīnu skaita):
   - Katrā kolonnā ir attēlota vienas automašīnas parametru vērtības.
3. Labāko vērtību izcelšana:
   - Labākās vērtības katram parametram tiek izceltas zaļā krāsā.
   - Piemēram, zemākā cena, jaunākais gads, mazākais nobraukums.

Automašīnu pārvaldība

Virs salīdzinājuma tabulas atrodas automašīnu pārvaldības elementi:

1. Pievienot automašīnu - poga, kas ļauj pievienot automašīnu salīdzinājumam no meklēšanas rezultātiem.
   - Šī opcija ir pieejama tikai tad, ja salīdzinājumā ir mazāk nekā 3 automašīnas.
2. Noņemt automašīnu - poga pie katras automašīnas kolonnas, kas ļauj noņemt to no salīdzinājuma.
3. Notīrīt salīdzinājumu - poga, kas noņem visas automašīnas no salīdzinājuma.

Salīdzinājuma eksportēšana

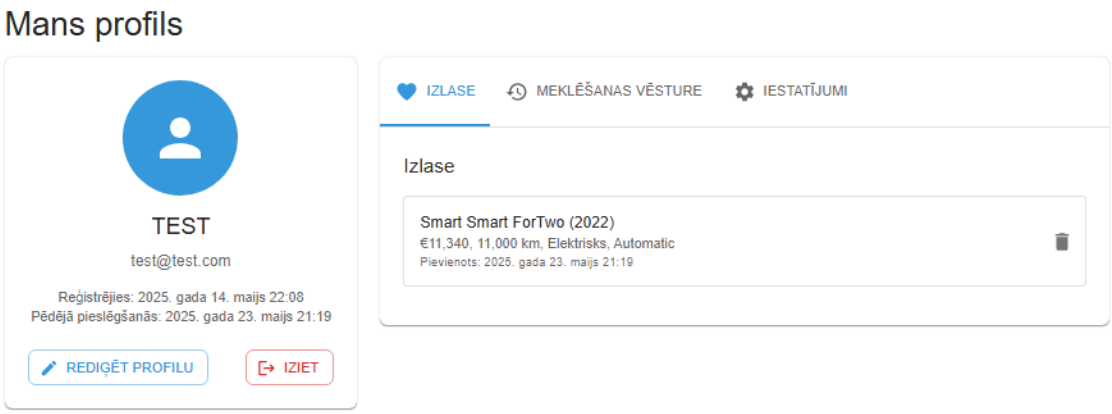
1. Zem salīdzinājuma tabulas atrodas eksporta poga:
   - EKSPORTĒT - eksportē salīdzinājuma datus JSON formātā.

Tukšs salīdzinājums

1. Ja salīdzinājumā nav pievienota neviena automašīna, tiek attēlots informatīvs paziņojums:
   - "Nav izvēlēta neviena automašīna salīdzināšanai. Lūdzu, pievienojiet automašīnas no meklēšanas rezultātiem, lai salīdzinātu."

6.3.4. Lietotāja profils un autentifikācija

Sistēma nodrošina lietotāju autentifikāciju un profilu pārvaldību, lai piedāvātu personalizētu pieredzi lietotājiem kas ir lēmuši piereģistrēties.



6.4. att. Lietotāja profila ekrāns

Reģistrācija un pieslēgšanās

1. Lai reģistrētos sistēmā, lietotājam jāaizpilda reģistrācijas forma:
   - E-pasts - lietotāja e-pasta adrese (tiks izmantota kā lietotājvārds).
   - Lietotājvārds - vēlamais lietotājvārds (parādīsies profilā).
   - Parole - droša parole (vismaz 8 simboli ar cipariem un burtiem).
   - Apstiprināt paroli - paroles atkārtota ievade.
   - Reģistrēties - poga reģistrācijas pabeigšanai.
2. Lai pieslēgtos sistēmai, lietotājam jāaizpilda pieslēgšanās forma:
   - E-pasts - reģistrētā e-pasta adrese.
   - Parole - izvēlētā parole.

- Pieslēgties - poga pieslēgšanās veikšanai.

Lietotāja profils

Pēc veiksmīgas autentifikācijas lietotājs var piekļūt savam profilam, kurā ir pieejamas šādas sadaļas:

1. Profila informācija:
   - Lietotājvārds
   - E-pasts
   - Reģistrācijas datums
   - Pēdējās pieslēgšanās datums
2. Preferences:
   - Tumšais režīms - ieslēgt/izslēgt tumšo režīmu.
   - Valoda - izvēlēties sistēmas valodu (pašlaik pieejama tikai latviešu valoda).
   - Paziņojumi - ieslēgt/izslēgt sistēmas paziņojumus.
3. Izlase:
   - Saraksts ar lietotāja izlasē pievienotajām automašīnām.
   - Katrai automašīnai tiek attēlota pamatinformācija (marka, modelis, gads, cena).
   - Lietotājs var noņemt automašīnas no izlases vai pievienot tās salīdzinājumam.
4. Meklēšanas vēsture:
   - Saraksts ar lietotāja veiktajām meklēšanām.
   - Katrai meklēšanai tiek attēloti izmantotie parametri un datums kad lietotājs veica meklēšanu.
5. Izrakstīšanās:
   - Poga, kas ļauj lietotājam izrakstīties no sistēmas.

6.4.5 Programmas izslēgšana

Lai izslēgtu Automobiļu tirgus analīzes sistēmu, veiciet šādas darbības:

1. Frontend servera apturēšana:
   - Komandas rindas logā, kurā darbojas frontend serveris, nospiediet pogas “Ctrl” kopā ar “C”.
   - Apstipriniet darbību, ja tiek prasīts.
2. API servera apturēšana:
   - Komandas rindas logā, kurā darbojas API serveris, nospiediet pogas “Ctrl” kopā ar “C”.
   - Apstipriniet darbību, ja tiek prasīts.
3. Virtuālās vides deaktivizēšana (ja tika izmantota):
   - Izmantojiet komandu: deactivate

Pēc šīm darbībām sistēmai vajadzētu būt pilnībā izslēgtai, un resursiem atbrīvotiem.

Lai pārliecinātos par sistēmas uzticamību un funkcionalitāti, veicu sistemātisku testēšanu, kas aptver gan pamatfunkcijas, gan robežgadījumus.

7. PROGRAMMATŪRAS LIETOJAMĪBAS TESTĒŠANA

Šajā sadaļā aprakstīti manis veiktie programmatūras testēšanas procesi un rezultāti, lai nodrošinātu sistēmas uzticamību un lietojamību.

7.1. Meklēšanas un filtrēšanas funkcionalitātes testēšana

Prasība FP-2: Automašīnu datu meklēšana un filtrēšana

Lietotājam jābūt iespējai meklēt un filtrēt automašīnu datus pēc dažādiem kritērijiem.

Tests #1: Meklēšana pēc markas un modeļa

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Marka: Tesla Modelis: Model S Citi parametri: nav norādīti	Lietotājs redz sarakstu ar Tesla Model S sludinājumiem. Ja ir pieejami sludinājumi, tie parādās tabulā ar cenu, nobraukumu un citiem parametriem, kā arī parādās paziņojums.	Izpildīts, atrasti 24 sludinājumi

Tests #2: Meklēšana ar cenu un gadu filtriem

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Marka: Tesla Modelis: Model S Gadu diapazons: 2015-2023 Cenu diapazons: €20,000-€40,000	Tiek attēloti tikai tie Tesla Model S sludinājumi, kas atbilst norādītajam gadu un cenu diapazonam. Rezultātu skaits ir lielāks par 0, bet mazāks nekā Testā #1. Visi rezultāti atbilst kritērijiem testa aprakstā.	Izpildīts, atrasti 10 sludinājumi

Tests #3: Meklēšana ar dzinēja tipa un ātrumkārbas filtriem

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Marka: Infiniti Modelis: Infiniti FX Izlaiduma gads: 2000-2025 Dzinēja tips: Benzīns Ātrumkārba: Automātiska	Tiek attēloti tikai tie Infiniti FX sludinājumi, kuriem ir benzīna dzinējs un automātiskā ātrumkārba. Rezultātu skaits ir lielāks par 0. Visi rezultāti atbilst kritērijiem testa aprakstā.	Izpildīts, atrasti 7 sludinājumi

Tests #4: Meklēšana ar neeksistējošu kombināciju

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Marka: Tesla Modelis: Model X Dzinēja tips: Dīzelis	Tiek attēlots paziņojums “Nav atrasts neviens sludinājums, kas atbilst jūsu kritērijiem.” Rezultātu skaits ir 0.	Izpildīts

7.2. Statistiskās analīzes funkcionalitātes testēšana

Prasība FP-3: Statistiskā analīze un cenu noteikšana

Sistēmai jāpalīdz lietotājiem saprast, vai konkrētās automašīnas cena ir adekvāta un kāda ir situācija tirgū kopumā.

Tests #1: Reģionālās analīzes veikšana

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Marka: Tesla Modelis: Model 3 Gadu diapazons: 2018-2023 Grafika tips: Vidējās cenas pa reģioniem	Tiek ģenerēts un attēlots stabiņu diagramma ar vidējām cenām dažādos reģionos. Grafiks satur vismaz 2 reģionus. Stabiņa grafiks satur detalizētu informāciju par mašīnu noteiktajā reģionā ja tāda ir pieejama.	Izpildīts

Tests #2: Reģionālās analīzes precizitāte

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Marka: Tesla Modelis: Visi modeļi Grafika tips: Vidējās cenas pa reģioniem	Tiek ģenerēts stabiņu grafiks ar vismaz 3 reģioniem. Rīgas vidējā cena ir augstāka par citiem reģioniem. Grafiks satur skaidrus marķējumus un krāsu kodēšanu.	Izpildīts, grafiks satur 10 reģionus

7.3. Automašīnu salīdzināšanas funkcionalitātes testēšana

Prasība FP-5: Automašīnu salīdzināšana

Sistēmai jānodrošina iespēja salīdzināt vairākas automašīnas, lai atvieglotu lēmumu pieņemšanu.

Tests #1: Divu automašīnu pievienošana salīdzinājumam

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Meklēt Tesla Model S Pievienot pirmo atrasto automašīnu salīdzinājumam Meklēt Tesla Model 3 Pievienot pirmo atrasto automašīnu salīdzinājumam	Salīdzinājuma tabulā tiek attēlotas abas izvēlētās automašīnas. Tabula satur rindas ar dažādiem parametriem (cena, gads, dzinējs, utt.). Labākās vērtības cenas, nobraukuma un gada parametros tiek izceltas zaļā krāsā.	Izpildīts, abas mašīnas rādās, parametri tiek izcelti

Tests #2: Automašīnas noņemšana no salīdzinājuma

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Atsaucamies uz Testu #1, kur ir divas automašīnas salīdzinājumā Noklikšķināt uz "Noņemt" pogas pirmās automašīnas kolonnā	Pirmā automašīna tiek noņemta no salīdzinājuma. Salīdzinājuma tabulā paliek tikai otrā automašīna. Tiek parādīts paziņojums "Automašīna noņemta no salīdzinājuma".	Izpildīts, paziņojums parādās

Tests #3: Maksimālā automašīnu skaita pārbaude

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Pievienot trīs dažādas automašīnas salīdzinājumam Mēģināt pievienot ceturto automašīnu	Salīdzinājuma tabulā tiek attēlotas trīs pievienotās automašīnas. Mēģinot pievienot ceturto automašīnu, parādās paziņojums "Var salīdzinat ne vairāk kā 3 automašīnas."	Izpildīts

Tests #4: Salīdzinājuma eksportēšana

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Autentificēties Pievienot divas dažādas automašīnas salīdzinājumam Noklikšķināt uz "EKSPORTĒT" pogas	Tiek ģenerēts un lejupielādēts JSON fails ar salīdzinājuma datiem. JSON fails satur abu automašīnu parametrus strukturētā formātā.	Izpildīts

7.4. Datu vizualizācijas funkcionalitātes testēšana

Prasība FP-4: Datu vizualizācija

Sistēmai jānodrošina datu vizualizācija, lai lietotājs varētu viegli interpretēt rezultātus.

Tests #1: Cenu sadalījuma histogrammas ģenerēšana

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Doties uz meklēšanas sadaļu Izvēlēties marku: Tesla Izvēlēties modeli: Model S Nospiest meklēt Pāriet uz sadaļu analīze Noklikšķināt uz "Atjaunināt"	Tiek ģenerēta un attēlota cenu sadalījuma histogramma. Histogramma satur vismaz 5 stabiņus, kas reprezentē dažādus cenu intervālus. Grafiks satur blīvuma līkni, kas parāda cenu sadalījuma tendenci. Grafiks ir skaidri marķēts ar asīm un nosaukumu.	Izpildīts, strādā kā paredzēts

Tests #2: Reģionālo cenu grafika ģenerēšana

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Doties uz meklēšanas sadaļu Izvēlēties marku: Tesla Izvēlēties modeli: Model S Nospiest meklēt Pāriet uz sadaļu analīze	Tiek ģenerēta un attēlota stabiņu diagramma ar vidējām cenām dažādos reģionos. Grafiks satur vismaz 2 reģionus. Reģioni ir sakārtoti pēc vidējās cenas. Grafiks ir skaidri marķēts ar asīm un nosaukumu. Zem grafika tiek attēlota tabula ar detalizētu informāciju par katru reģionu.	Izpildīts

Tests #3: Grafika eksportēšana PNG formātā

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: 1. Doties uz Analīzes sadaļu 2. Ģenerēt Cenu sadalījuma grafiku 3. Noklikšķināt uz “Lejupielādēt “ pogas	Tiek ģenerēts un lejupielādēts PNG attēls ar grafiku. Attēla kvalitāte ir laba un visi grafika elementi ir skaidri saskatāmi.	Izpildīts

Tests #4: Meklēšana ar tukšiem datiem

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Doties uz meklēšanas sadaļu Izvēlēties marku: Smart Izvēlēties modeli: ForTwo Izvēlēties gadu diapazonu: 2022-2025 (pieņemot, ka nav šādu datu) Izvēlēties grafika tipu: Cenu sadalījums Noklikšķināt uz “Atjaunināt”	Tiek attēlots paziņojums “Neidzevās ielādēt grafiku.” Grafika vietā tiek attēlots tukšs laukums vai paskaidrojošs attēls.	Izpildīts

7.5. Lietotāju autentifikācijas funkcionalitātes testēšana

Prasība FP-6: Lietotāju autentifikācija un profili

Sistēmai jānodrošina lietotāju autentifikācija un profilu veidošana, lai saglabātu lietotāju preferences un ļautu izmantot papildu funkcionalitātes.

Tests #1: Lietotāja reģistrācija

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Doties uz Reģistrācijas lapu Ievadīt e-pastu: test1@test.com Ievadīt lietotājvārdu: testuser Ievadīt paroli: Test12345. Apstiprināt paroli: Test12345. Noklikšķināt uz "Reģistrēties"	Lietotājs tiek veiksmīgi reģistrēts. Tiek attēlots paziņojums "Pieslēgšanās veiksmīga." Lietotājs tiek automātiski pieslēgts sistēmai un tiek novirzīts uz sākuma lapu.	Izpildīts

Tests #2: Lietotāja pieslēgšanās

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Doties uz Pieslēgšanās lapu Ievadīt e-pastu: test@test.com Ievadīt paroli: newpassword123. Noklikšķināt uz "Pieslēgties"	Lietotājs tiek veiksmīgi autentificēts. Tiek attēlots paziņojums "Pieslēgšanās veiksmīga!" Lietotājs tiek novirzīts uz galveno lapu. Navigācijas joslā tiek attēlots lietotāja vārda pirmais burts.	Izpildīts

Tests #3: Neveiksmīga pieslēgšanās ar nepareizu paroli

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Doties uz Pieslēgšanās lapu Ievadīt e-pastu: test@test.com Ievadīt paroli: NepareizaParole Noklikšķināt uz "Pieslēgties"	Lietotājs netiek autentificēts. Tiek attēlots kļūdas paziņojums "Nepareizs e-pasts vai parole." Lietotājs paliek pieslēgšanās lapā.	Izpildīts

Tests #4: Automašīnas pievienošana izlasei

Testa apraksts	Sagaidāmais rezultāts	Testēšanas statuss
Ievadāmā informācija: Pieslēgties kā test@test.com Meklēt automašīnas Izvēlēties pirmo atrasto automašīnu Noklikšķināt uz "Pievienot izlasei"	Automašīna tiek pievienota lietotāja izlasei. Tiek attēlots paziņojums "Automašīna pievienota izlasei." Izlases ikona pie automašīnas maina krāsu uz sarkanu, lai norādītu, ka tā ir izlasē.	Izpildīts

7.6. Testēšanas rezultātu kopsavilkums

Veicot testēšanu 5 galvenajām funkcionalitātēm ar kopumā 20 testiem, tika konstatēts, ka visas pārbaudītās funkcionalitātes darbojas atbilstoši prasībām. Visi testi tika izpildīti veiksmīgi, kas liecina par sistēmas stabilitāti un atbilstību definētajām prasībām.

Testēšanas laikā tika pārbaudītas šādas galvenās funkcionalitātes:

- Automašīnu datu meklēšana un filtrēšana (4 testi)
- Statistiskā analīze un cenu noteikšana (2 testi)
- Automašīnu salīdzināšana (4 testi)
- Datu vizualizācija (4 testi)
- Lietotāju autentifikācija un profili (4 testi)

Sistēma darbojas stabili un nodrošina visas plānotās funkcionalitātes. Lietotāja saskarne ir intuitīva un viegli lietojama, un sistēma reaģē ātri uz lietotāja darbībām. Datu vizualizācijas ir skaidras un informatīvas, kas palīdz lietotājiem viegli interpretēt rezultātus.

Tomēr testēšanas laikā tika identificēti daži aspekti, kurus nākotnē varētu uzlabot:

1. Datu apjoms - Pašlaik sistēma darbojas ar ierobežotu automašīnu marku skaitu (Tesla, Infiniti, Smart, Suzuki). Nākotnē būtu vēlams paplašināt datu apjomu, iekļaujot vairāk marku.
2. Automatizēta datu iegūšana - Pašlaik datu iegūšana tiek veikta manuāli. Nākotnē būtu lietderīgi ieviest automātisko datu iegūšanu pēc noteikta grafika (piemēram, reizi dienā).
3. Cenu tendenču analīze - Pašlaik sistēmā nav realizēta cenu tendenču analīze laika gaitā, jo tam nepieciešami vēsturiskie dati. Nākotnē, kad būs uzkrāts pietiekams datu apjoms, būtu vēlams implementēt šo funkcionalitāti.
4. Papildu datu avoti - Sistēma pašlaik iegūst datus tikai no SS.LV. Nākotnē būtu lietderīgi pievienot citus datu avotus, lai nodrošinātu plašāku tirgus pārklājumu.

Kopumā testēšanas rezultāti apliecina, ka Automobiļu tirgus analīzes sistēma ir funkcionējoša un lietojama, atbilstoši definētajām prasībām.

NOBEIGUMS

Šī kvalifikācijas darba izstrāde ir bijusi gan izaicinājums, gan vērtīga mācību pieredze. Sākotnēji biju plānojis izveidot vienkāršu cenu salīdzināšanas rīku, taču projekta gaitā sapratu, ka automobiļu tirgus analīze ir daudz sarežģītāka, nekā sākotnēji domāju. Katrs ieviestais risinājums atklāja jaunas problēmas un iespējas.

Darba procesā saskāros ar vairākiem izaicinājumiem, visvairāk nervus izmaksāja SS.LV parsēšana, ņemot vērā, ka man nebija pieredzes ar web scraping. Sākumā nevarēju saprast, kā pareizi atlasīt HTML elementus. Tad atklāju, ka daži sludinājumi maina struktūru, un mans kods saplīsa uz tiem, kuriem nebija paredzētās informācijas. Šādas epizodes bija vairākas. Vēl viens sarežģījums bija datu normalizācija - dažādi pārdevēji ievada informāciju atšķirīgos formātos, piemēram, viens norāda dzinēja tilpumu kā "2.0", bet otrs kā "2,0 L", kas prasīja rūpīgu datu tīrīšanas procesu un eksperimentēšanu.

Visvairāk gandarījuma sniedza statistiskās analīzes un vizualizācijas daļas izveide. Bija interesanti novērot, kā no “sausiem” datiem veidojas skaidra aina par tirgus situāciju - piemēram, atklājot, ka Rīgā Teslu markas automašīnas ir vidēji par 12% dārgākas nekā citos Latvijas reģionos, vai to, ka Infiniti modeļiem ar dīzeļdzinējiem nobraukums visbiežāk ir starp 150-200 tūkstošiem kilometru.

Šobrīd sistēma darbojas ar ierobežotu datu kopu (četras automašīnu markas), taču tās arhitektūra ļauj viegli paplašināt funkcionalitāti. Nākotnē plānoju pievienot vairāk datu avotu, piemēram, AutoPlus.lt, ieviest automatizētu ikdienas datu atjaunināšanu un izstrādāt prognostiskos modeļus, kas varētu paredzēt cenu izmaiņas, balstoties uz vēsturiskajiem datiem.

Esmu pārliecināts, ka izstrādātā sistēma varētu būt noderīga gan privātpersonām, kas meklē sev piemērotu automašīnu par saprātīgu cenu, gan auto tirgotājiem, kas vēlas balstīt savus lēmumus uz reāliem tirgus datiem. Īpaši vērtīga funkcionalitāte ir automašīnas aptuvenas tirgus vērtības aprēķins, kas ļauj izvairīties no pārmaksāšanas vai pārāk zemas pārdošanas cenas noteikšanas.

Izstrādājot šo projektu, esmu būtiski paplašinājis savas zināšanas datu analīzes jomā un pilnveidojis prasmes darbā ar moderniem tīmekļa izstrādes rīkiem. Uzskatu, ka apgūtās tehnoloģijas un pieredze būs vērtīga manas turpmākās profesionālās karjeras veidošanā.

INFORMĀCIJAS AVOTI

1. Auto24.lv Tirgus tendenču apskats. Pieejams: https://www.auto24.lv/ (Apskatīts: 18.05.2025).
2. BeautifulSoup Tīmekļa parsēšanas bibliotēkas dokumentācija. Pieejams: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (Apskatīts: 18.03.2025).
3. CSB Centrālās statistikas pārvaldes transporta statistika. Pieejams: https://stat.gov.lv/lv/statistikas-temas/noz/transports (Apskatīts: 12.05.2025).
4. Flask Tīmekļa ietvara dokumentācija. Pieejams: https://flask.palletsprojects.com/en/stable/ Apskatīts: 15.02.2025).
5. GitHub Repozitoriju veidošanas ceļvedis. Pieejams: https://docs.github.com/en/repositories/creating-and-managing-repositories (Apskatīts: 25.02.2025).
6. Material-UI React komponenšu bibliotēka. Pieejams: https://mui.com/material-ui/getting-started/ (Apskatīts: 05.03.2025).
7. Matplotlib Grafiku veidošanas dokumentācija. Pieejams: https://matplotlib.org/stable/tutorials/pyplot.html (Apskatīts: 02.04.2025).
8. MDN Web Docs JavaScript pamati. Pieejams: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting (Apskatīts: 10.03.2025).
9. NumPy Matemātisko operāciju bibliotēkas dokumentācija. Pieejams: https://numpy.org/doc/stable/user/quickstart.html (Apskatīts: 25.03.2025).
10. Python.org Aiohttp asinhrono HTTP dokumentācija. Pieejams: https://docs.aiohttp.org/en/stable/ (Apskatīts: 08.04.2025).
11. React Izstrādātāju dokumentācija. Pieejams: https://react.dev/learn (Apskatīts: 28.02.2025).
12. Reddit Programmēšanas kopienas diskusijas par React. Pieejams: https://www.reddit.com/r/reactjs/ (Apskatīts: 30.03.2025).
13. SQLAlchemy ORM dokumentācija. Pieejams: https://docs.sqlalchemy.org/en/20/tutorial/ (Apskatīts: 12.03.2025).
14. SS.LV Automašīnu sludinājumi. Pieejams: https://www.ss.lv/lv/transport/cars/ (Apskatīts: 20.05.2025).
15. Stack Overflow Python web scraping risinājumi. Pieejams: https://stackoverflow.com/questions/tagged/python+web-scraping (Apskatīts: 22.03.2025).
16. W3Schools HTML un CSS pamācības. Pieejams: https://www.w3schools.com/html/ (Apskatīts: 15.03.2025).
17. YouTube Python web scraping pamācības. Pieejams: https://www.youtube.com/results?search_query=python+web+scraping+tutorial (Apskatīts: 21.02.2025).

PIELIKUMI

1.pielikums. Web Scraping modulis sludinājumu detalizētās informācijas iegūšanai

Funkcija apmeklē konkrētu SS.LV auto sludinājuma lapu un izvelk no tās visu svarīgo informāciju - cenu, dzinēja datus, aprīkojumu, fotogrāfijas un aprakstu. Tā darbojas kā robots, kas lasa mājaslapas tāpat kā cilvēks, bet dara to ātri un samērā precīzi.

```
Api.py:
@app.route('/api/listing-details', methods=['GET'])
def listing_details():
    """Scrape full details from a SS.LV listing"""
    listing_url = request.args.get('url')
    if not listing_url:
        logger_listing_details.warning("No URL provided")
        return jsonify({"error": "URL parameter required"}), 400

    logger_listing_details.info(f"Getting details for: {listing_url}")

    try:
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
   Safari/537.36',
            'Accept-Language': 'lv-LV,lv;q=0.9,en-US;q=0.8,en;q=0.7',
            'Accept':
'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
            'Connection': 'keep-alive',
            'Referer': 'https://www.ss.lv/'
        }

        response = requests.get(listing_url, headers=headers,
   timeout=15)
        response.raise_for_status()

        soup = BeautifulSoup(response.content, 'html.parser')

        details = {}

        # Extract description before tables
        description_div = soup.select_one('div#msg_div_msg')
        if description_div:
            description_copy = description_div.__copy__()

            # Remove the options/price tables
            options_table =
   description_copy.select_one('table.options_list')
            if options_table:
                options_table.extract()
```

```
            # Remove price tables
            price_tables = description_copy.find_all('table')
            for table in price_tables:
                # Check if this looks like a price table
                price_indicators = [
                    table.find('td', class_='ads_opt_name_big'),
                    table.find('td', class_='ads_price'),
                    table.find('span', class_='ads_price'),
                    table.find('a', class_='a9a'),
                    table.find('img', src=lambda x: x and
'octa_logo.png' in x)
                ]

                if any(price_indicators):
                    table.extract()
                    continue

                # Also check text content
                table_text = table.get_text()
                if 'Cena:' in table_text or '€' in table_text or
'apdrošināšanu' in table_text:
                    table.extract()

            # Clean up description text
            description_text = description_copy.get_text(separator='\n',
strip=True)
            lines = [line.strip() for line in
description_text.split('\n') if line.strip()]
            description_text = '\n'.join(lines)

            details['description'] = description_text
        else:
            details['description'] = None

        # Get publication date
        date_cell = soup.select_one('td.msg_footer[align="right"]')
        if date_cell and 'Datums:' in date_cell.get_text():
            date_text = date_cell.get_text().strip()
            if 'Datums:' in date_text:
                publication_date = date_text.split('Datums:')[1].strip()
                details['publication_date'] = publication_date
        else:
            details['publication_date'] = None

        # Find main image
        image_elem = soup.select_one('img#msg_img_img')
        if image_elem and image_elem.has_attr('src'):
            details['image_url'] = image_elem['src']
        else:
```

```
        image_elem_fallback = soup.select_one('div#big_pic_div img')
        if image_elem_fallback and
image_elem_fallback.has_attr('src'):
            details['image_url'] = image_elem_fallback['src']
        else:
            details['image_url'] = None

    # Parse paramet table, map labels
    params_table = soup.select_one('div#msg_div_msg
table.options_list')
    if params_table:
        rows = params_table.select('tr')
        for row in rows:
            label_cell = row.select_one('td.ads_opt_name')
            value_cell = row.select_one('td.ads_opt')
            if label_cell and value_cell:
                label_text = label_cell.get_text(strip=True).lower()
                value_text = value_cell.get_text(strip=True)

                # Clean up value
                value_text = ' '.join(value_text.split())

                # Map labels to fields
                if 'marka' in label_text:
                    bold_elem = value_cell.select_one('b')
                    if bold_elem:
                        details['brand_model_detail'] =
bold_elem.get_text(strip=True)
                    else:
                        details['brand_model_detail'] = value_text
                elif 'izlaiduma gads' in label_text:
                    details['year_detail'] = value_text
                elif 'dzinēja tips' in label_text or 'motors' in
label_text:
                    details['engine_detail'] = value_text
                elif 'ātr.kārba' in label_text or 'ātrumkārba' in
label_text:
                    details['transmission_detail'] = value_text
                elif 'nobraukums' in label_text:
                    # Clean up mileage
                    mileage_digits = ''.join(filter(str.isdigit,
value_text))
                    details['mileage_detail'] = mileage_digits
                elif 'krāsa' in label_text:
                    color_text = value_text.split('\n')[0].strip()
                    details['color_detail'] = color_text
                elif 'virsbūves tips' in label_text:
                    details['body_type_detail'] = value_text
                elif 'tehniskā apskate' in label_text:
```

```
                    details['tech_inspection'] = value_text

        # Extract price
        price_elem = soup.select_one('span.ads_price#tdo_8')
        if price_elem:
            price_text = price_elem.get_text(strip=True)
            price_clean = price_text.replace('€', '').replace(' ',
'').replace(',', '')
            try:
                details['price_detail'] = int(price_clean)
            except ValueError:
                details['price_detail'] = price_text

        # Get equipment list
        equipment_list = []
        if description_div:
            checkboxes =
description_div.select('input[type="checkbox"]')
            for checkbox in checkboxes:
                label = checkbox.find_next_sibling(text=True)
                if label:
                    equipment_list.append(label.strip())

        # Also check description for equipment
        if details.get('description'):
            desc_text = details['description']
            lines = desc_text.split('\n')
            for line in lines:
                line = line.strip()
                if line.startswith('•') or line.startswith('-'):
                    equipment_list.append(line[1:].strip())

        details['equipment'] = equipment_list

        # Extract region from contacts table
        contacts_table = soup.select_one('table.contacts_table')
        if contacts_table:
            rows = contacts_table.select('tr')
            for row in rows:
                label_cell = row.select_one('td.ads_contacts_name')
                value_cell = row.select_one('td.ads_contacts')

                if label_cell and value_cell:
                    label_text = label_cell.get_text(strip=True)
                    if label_text == 'Vieta:':
                        details['region'] =
value_cell.get_text(strip=True)
                        break
```

```
        logger_listing_details.info(f"Details scraped successfully for
    {listing_url}")
        return jsonify(details)

     except requests.exceptions.Timeout:
        logger_listing_details.error(f"Timeout for {listing_url}")
        return jsonify({"error": "SS.LV timeout"}), 504
     except requests.exceptions.RequestException as e:
        logger_listing_details.error(f"Network error for {listing_url}:
    {str(e)}")
        return jsonify({"error": f"Network error: {type(e).__name__}"}),
    502
     except Exception as e:
        logger_listing_details.error(f"Scraping failed for
    {listing_url}: {str(e)}", exc_info=True)
        return jsonify({"error": "Failed to parse listing"}), 500
```

2.pielikums. Cenu sadalījuma grafika ģenerēšanas funkcija

Funkcija paņem visas automašīnu cenas no datubāzes un izveido histogrammu, kas parāda cik auto ir katrā cenu diapazonā. Tas palīdz lietotājiem ātri saprast, vai viņu iecerētā cena ir reālistiska vai pārāk optimistiska.

```
analysis.py:
def create_price_distribution_chart(self, brand=None, model=None,
   year_from=None, year_to=None):
    """Make a histogram showing price distribution"""
    from models import Brand, Model, Car, Listing

    try:
        query = self.session.query(
            Listing.price
        ).join(
            Car, Listing.car_id == Car.car_id
        ).join(
            Model, Car.model_id == Model.model_id
        ).join(
            Brand, Model.brand_id == Brand.brand_id
        )

        # Apply filters
        if brand:
            query = query.filter(func.lower(Brand.name) ==
    func.lower(brand))

        if model:
            query = query.filter(func.lower(Model.name) ==
    func.lower(model))
```

```
    if year_from:
        query = query.filter(Car.year >= year_from)

    if year_to:
        query = query.filter(Car.year <= year_to)

    prices = [item[0] for item in query.all()]

    if not prices or len(prices) < 5:
        logger.info(f"Not enough data for chart - only {len(prices)
if prices else 0} cars")
        return None

    # Create the chart
    plt.figure(figsize=(10, 6))
    sns.histplot(prices, bins=20, kde=True)

    # Chart title
    title = "Cenu sadalījums"
    if brand:
        title += f" - {brand}"
        if model:
            title += f" {model}"

    if year_from or year_to:
        years = ""
        if year_from:
            years += f"{year_from}"
        if year_to:
            years += f"-{year_to}"
        title += f" ({years})"

    plt.title(title)
    plt.xlabel("Cena (EUR)")
    plt.ylabel("Sludinājumu skaits")
    plt.grid(True, alpha=0.3)

    # Convert to base64 for API response
    buffer = BytesIO()
    plt.savefig(buffer, format='png', dpi=100)
    buffer.seek(0)

    image_base64 = base64.b64encode(buffer.getvalue()).decode('utf-
8')
    plt.close()

    logger.info(f"Created price distribution chart with
{len(prices)} cars")
    return image_base64
```

```
        except Exception as e:
            logger.error(f"Chart creation failed: {str(e)}")
            return None
```

3.pielikums. Datubāzes modeļu definīcijas

Šis kods definē, kā sistēma glabā informāciju par automašīnām - kuri lauki ir nepieciešami, kā tabulas saistās viena ar otru. Tas ir kā mājas pamati - bez tā nekas cits nestrādās.

```
models.py:
class Car(Base):
    """Model representing individual cars"""
    __tablename__ = 'cars'

    car_id = Column(Integer, primary_key=True)
    model_id = Column(Integer, ForeignKey('models.model_id'),
   nullable=False)
    region_id = Column(Integer, ForeignKey('regions.region_id'),
   nullable=False)
    year = Column(Integer, nullable=False)
    engine_volume = Column(Float)
    engine_type = Column(String(20))
    transmission = Column(String(20))
    tech_inspection = Column(String(20))
    mileage = Column(Integer)
    body_type = Column(String(20))
    color = Column(String(20))
    created_at = Column(DateTime, default=datetime.now)
    updated_at = Column(DateTime, default=datetime.now,
   onupdate=datetime.now)

    # Relationships
    model = relationship("Model", back_populates="cars")
    region = relationship("Region", back_populates="cars")
    listings = relationship("Listing", back_populates="car")
    analyses = relationship("Analysis", back_populates="car")

    def __repr__(self):
        return f"<Car(model_id={self.model_id}, year={self.year})>"

class Listing(Base):
    """Model representing car listings/advertisements"""
    __tablename__ = 'listings'

    listing_id = Column(Integer, primary_key=True)
    car_id = Column(Integer, ForeignKey('cars.car_id'), nullable=False)
    source_id = Column(Integer, ForeignKey('sources.source_id'),
   nullable=False)
```

```
    external_id = Column(String(50))
    price = Column(Integer, nullable=False)
    listing_date = Column(Date, nullable=False)
    listing_url = Column(String(255))
    is_active = Column(Boolean, default=True)
    created_at = Column(DateTime, default=datetime.now)
    updated_at = Column(DateTime, default=datetime.now,
onupdate=datetime.now)

    # Relationships
    car = relationship("Car", back_populates="listings")
    source = relationship("Source", back_populates="listings")

    def __repr__(self):
        return f"<Listing(car_id={self.car_id}, price={self.price})>"
```

4.pielikums. API galvenie endpoint un meklēšanas funkcionalitāte

Galvenā meklēšanas funkcija, kas apstrādā lietotāju pieprasījumus un atrod atbilstošas automašīnas datubāzē. Tā saprot dažādus filtrus (marka, gads, cena) un atgriež rezultātus, kas gatavi rādīšanai lietotājam.

```
api.py:
@app.route('/api/search', methods=['POST'])
def search_cars():
    """Handle car search with various filters"""
    try:
        data = request.json

        # Get search filters
        brand = data.get('brand')
        model = data.get('model')
        year_from = data.get('yearFrom')
        year_to = data.get('yearTo')
        fuel_type = data.get('fuelType')
        transmission = data.get('transmission')
        price_from = data.get('priceFrom')
        price_to = data.get('priceTo')
        region = data.get('region')

        logger.info(f"Search: brand={brand}, model={model},
fuel={fuel_type}")

        # Get price stats first - this is where the magic happens
        statistics = analyzer.get_price_statistics(
            brand=brand,
            model=model,
            year_from=year_from,
            year_to=year_to,
            fuel_type=fuel_type
```

```
    )

    # Search for actual cars - complex query building starts here
    listings = []
    if brand:  # Need at least a brand to search, jo citādi būtu
pārāk daudz rezultātu
        query = (
            analyzer.session.query(
                Brand.name.label('brand'),
                Model.name.label('model'),
                Car.year,
                Car.engine_volume,
                Car.engine_type,
                Car.transmission,
                Car.mileage,
                Car.body_type,
                Car.color,
                Car.tech_inspection,
                Listing.price,
                Listing.listing_date,
                Listing.listing_url,
                Listing.external_id,
                Region.name.label('region')
            )
            .join(Model, Brand.brand_id == Model.brand_id)
            .join(Car, Model.model_id == Car.model_id)
            .join(Listing, Car.car_id == Listing.car_id)
            .join(Region, Car.region_id == Region.region_id)
            .filter(func.lower(Brand.name) == func.lower(brand))
        )

        # Apply filters one by one - šeit notiek visas maģijas ar
filtrēšanu
        if model:
            query = query.filter(func.lower(Model.name) ==
func.lower(model))
        if year_from:
            query = query.filter(Car.year >= year_from)
        if year_to:
            query = query.filter(Car.year <= year_to)
        if fuel_type:
            logger.info(f"Filtering by fuel: {fuel_type}")

            # Handle different fuel type variations - šis ir īpaši
svarīgs, jo SS.LV ir dažādi formāti
            if fuel_type.lower() in ['petrol', 'benzīns',
'benzins']:
                query =
query.filter(func.lower(Car.engine_type).like('%benzīn%') |
```

```
func.lower(Car.engine_type).like('%petrol%'))
            elif fuel_type.lower() in ['diesel', 'dīzelis',
'dizelis']:
                query =
query.filter(func.lower(Car.engine_type).like('%dīzel%') |

func.lower(Car.engine_type).like('%diesel%'))
            elif fuel_type.lower() in ['hybrid', 'hibrīds',
'hibrids']:
                query =
query.filter(func.lower(Car.engine_type).like('%hibrīd%') |

func.lower(Car.engine_type).like('%hybrid%'))
            elif fuel_type.lower() in ['electric', 'elektriskais',
'elektrisks']:
                query =
query.filter(func.lower(Car.engine_type).like('%elektr%'))
            else:
                # Just try to match whatever they typed - fallback
option
                query =
query.filter(func.lower(Car.engine_type).like(f'%{fuel_type.lower()}%
'))

        # Handle sorting from frontend - users want control over
results
        sort_by = data.get('sortBy', 'price')
        sort_order = data.get('sortOrder', 'asc')

        if sort_by == 'price':
            if sort_order == 'desc':
                query = query.order_by(desc(Listing.price))
            else:
                query = query.order_by(asc(Listing.price))
        elif sort_by == 'year':
            if sort_order == 'desc':
                query = query.order_by(desc(Car.year))
            else:
                query = query.order_by(asc(Car.year))

        # Limit results for performance - nevajag nogalināt datubāzi
        query = query.limit(200)

        results = query.all()

        # Format results for frontend - šeit formatējam datus priekš
React komponentes
        for row in results:
```

```
                # Translate engine type to Latvian - svarīgi UX
                engine_type_latvian = "Nav norādīts"
                if row.engine_type:
                    engine_type_lower = row.engine_type.lower()
                    if 'petrol' in engine_type_lower:
                        engine_type_latvian = 'Benzīns'
                    elif 'diesel' in engine_type_lower:
                        engine_type_latvian = 'Dīzelis'
                    elif 'hybrid' in engine_type_lower:
                        engine_type_latvian = 'Hibrīds'
                    elif 'electric' in engine_type_lower:
                        engine_type_latvian = 'Elektriskais'

                listings.append({
                    'brand': row.brand,
                    'model': row.model,
                    'year': row.year,
                    'engine_type_latvian': engine_type_latvian,
                    'price': row.price,
                    'listing_url': row.listing_url,
                    'region': row.region or "Nav norādīts",
                    'id': row.external_id or f"listing-
{hash(str(row.listing_url))}"
                })

        return jsonify({
            "statistics": statistics if statistics else {},
            "listings": listings
        })

    except Exception as e:
        logger.error(f"Search failed: {str(e)}", exc_info=True)
        return jsonify({"error": "Meklēšana neizdevās"}), 500
```

5.pielikums. Lietotāju autentifikācijas sistēma

Sistēma, kas ļauj lietotājiem reģistrēties un pieslēgties, drošā veidā glabājot viņu paroles un preferences. Bez šīs daļas neviens nevarētu saglabāt savas mīļākās automašīnas vai meklēšanas vēsturi.

```
auth_models.py:
class AuthDB:
    def __init__(self, db_path=AUTH_DB_PATH):
        """Initialize the authentication database"""
        self.db_path = db_path
        self._ensure_db_exists()

    def create_user(self, email, username, password):
        """Create a new user with hashed password"""
        try:
```

```
        # Generate secure random salt - šis ir crucial priekš
security
        salt = secrets.token_hex(16)

        # Hash the password with the salt - never store plain
passwords!
        password_hash = self._hash_password(password, salt)

        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        # Insert the new user - email.lower() lai novērstu
duplicates ar case sensitivity
        cursor.execute(
            "INSERT INTO users (email, username, password_hash,
salt) VALUES (?, ?, ?, ?)",
            (email.lower(), username, password_hash, salt)
        )

        user_id = cursor.lastrowid

        # Create default preferences - every user gets basic
settings
        cursor.execute(
            "INSERT INTO user_preferences (user_id) VALUES (?)",
            (user_id,)
        )

        conn.commit()

        # Return user info (without sensitive data) - never return
passwords!
        user = self.get_user_by_id(user_id)
        conn.close()
        return user

    except sqlite3.IntegrityError:
        # Email jau eksistē - handle gracefully
        return {"error": "E-pasts jau tiek izmantots"}
    except Exception as e:
        return {"error": str(e)}

def authenticate_user(self, email, password):
    """Authenticate a user by email and password"""
    try:
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        # Get user by email - case insensitive search
```

```
            cursor.execute(
                "SELECT user_id, password_hash, salt FROM users WHERE
email = ? AND is_active = 1",
                (email.lower(),)
            )

            result = cursor.fetchone()
            if not result:
                return {"error": "Nepareizs e-pasts vai parole"}

            user_id, stored_hash, salt = result

            # Verify password - šeit notiek actual password checking
            if self._hash_password(password, salt) != stored_hash:
                return {"error": "Nepareizs e-pasts vai parole"}

            # Update last login time - good practice priekš analytics
            cursor.execute(
                "UPDATE users SET last_login = CURRENT_TIMESTAMP WHERE
user_id = ?",
                (user_id,)
            )
            conn.commit()

            # Get user with preferences
            user = self.get_user_by_id(user_id)

            # Generate JWT token - modern session management
            token = self._generate_token(user_id)
            user['token'] = token

            conn.close()
            return user

        except Exception as e:
            return {"error": str(e)}

    def _hash_password(self, password, salt):
        """Hash a password with the given salt"""
        # Kombinējam paroli ar salt un hash - security 101
        password_salt = password + salt
        hash_obj = hashlib.sha256(password_salt.encode())
        return hash_obj.hexdigest()

    def _generate_token(self, user_id):
        """Generate a JWT token for authentication"""
        # JWT ir better nekā sessions priekš API
        secret_key = "car_prices_app_secret_key"  # Reālā vidē būtu env
variable
```

```
        expiry = datetime.utcnow() + timedelta(days=1)  # 24h validity

        payload = {
            'user_id': user_id,
            'exp': expiry
        }

        token = jwt.encode(payload, secret_key, algorithm='HS256')
        return token
```

6.pielikums. React meklēšanas komponente ar filtriem

Lietotāju saskarne ar slideriem, dropdowniem un citiem kontroles elementiem, caur kuriem var norādīt meklēšanas kritērijus. Tas ir vizuālais slānis, ar ko lietotāji faktiski mijiedarbojas, lai atrastu sev piemērotu auto.

SearchForm.js:

```
const SearchForm = ({
  brands = [],
  models = [],
  params = {
    brand: '',
    model: '',
    yearFrom: new Date().getFullYear() - 10,
    yearTo: new Date().getFullYear(),
    priceFrom: 0,
    priceTo: 100000,
    fuelType: '',
    transmission: '',
    region: ''
  },
  onParamChange = () => {},
  onSearch = () => {},
  loading = false
}) => {
  const theme = useTheme();

  // Local state for form values with validation - real-time feedback ir
   svarīgs
  const [localParams, setLocalParams] = useState({...params});
  const [advanced, setAdvanced] = useState(false);
  const [errors, setErrors] = useState({});
  const [regions, setRegions] = useState([]);

  // Fetch regions on component mount - load data when needed
  useEffect(() => {
    const fetchRegions = async () => {
      try {
        const response = await getRegions();
        if (response && response.regions) {
```

```
        setRegions(response.regions);
      }
    } catch (err) {
      console.error('Error fetching regions:', err);
      // Nav critical error, tāpēc nebreak'ojam UI
    }
  };

  fetchRegions();
}, []);

// Validate price range - prevent nonsensical inputs
useEffect(() => {
  const newErrors = {};

  if (localParams.priceFrom > localParams.priceTo) {
    newErrors.price = 'Minimālā cena nevar pārsniegt maksimālo cenu';
  }

  if (localParams.yearFrom > localParams.yearTo) {
    newErrors.year = 'Minimālais gads nevar pārsniegt maksimālo gadu';
  }

  setErrors(newErrors);
}, [localParams.priceFrom, localParams.priceTo, localParams.yearFrom,
 localParams.yearTo]);

// Handle local changes before propagating to parent - debouncing būtu
 ideāli, bet šis ir OK
const handleLocalChange = (name, value) => {
  setLocalParams(prev => ({
    ...prev,
    [name]: value
  }));

  // Propagate to parent component immediately - for real-time updates
  onParamChange(name, value);
};

// Handle year range changes from slider - users love sliders
const handleYearChange = (event, newValue) => {
  handleLocalChange('yearFrom', newValue[0]);
  handleLocalChange('yearTo', newValue[1]);
};

// Handle price range changes - same pattern kā years
const handlePriceChange = (event, newValue) => {
  handleLocalChange('priceFrom', newValue[0]);
  handleLocalChange('priceTo', newValue[1]);
```

```
};

// Format currencies properly - user experience details matter
const formatCurrency = (value) => {
  if (value === undefined || value === null) {
    return '€0';
  }
  return `€${value.toLocaleString()}`;
};

// Fuel type options - match backend expectations exactly
const fuelTypes = [
  { value: 'Petrol', label: 'Benzīns' },
  { value: 'Diesel', label: 'Dīzelis' },
  { value: 'Hybrid', label: 'Hibrīds' },
  { value: 'Electric', label: 'Elektriskais' },
  { value: 'Gas', label: 'Gāze/LPG' }
];

return (
  <Box>
    {/* Main filters - most important ones first */}
    <FormControl fullWidth margin="normal">
      <Autocomplete
        id="brand-select"
        options={brands}
        getOptionLabel={(option) => option?.name || ''}
        value={brands.find(b => b.name === localParams.brand) || null}
        onChange={(event, newValue) => {
          handleLocalChange('brand', newValue ? newValue.name : '');
          handleLocalChange('model', ''); // Reset model when brand
 changes - UX logic
        }}
        renderInput={(params) => (
          <TextField
            {...params}
            label="Marka"
            variant="outlined"
            placeholder="Jebkura marka"
          />
        )}
        renderOption={(props, option) => (
          <li {...props}>
            {option.name} {option.count && <Typography
 component="span" variant="body2" color="text.secondary" sx={{ ml: 1
 }}>({option.count})</Typography>}
          </li>
        )}
        disabled={loading}
```

```
      />
    </FormControl>

    {/* Price range with fancy sliders - visual feedback ir crucial
*/}
    <Paper variant="outlined" sx={{
      p: 2,
      mt: 2,
      mb: 2,
      bgcolor: theme.palette.mode === 'dark'
        ? 'rgba(76, 175, 80, 0.12)'
        : 'rgba(76, 175, 80, 0.02)',
      border: `1px solid ${theme.palette.divider}`
    }}>
      <Box sx={{ display: 'flex', justifyContent: 'space-between',
alignItems: 'center', mb: 1 }}>
        <Typography id="price-range-slider" gutterBottom sx={{
display: 'flex', alignItems: 'center' }}>
          <MonetizationOnIcon sx={{ mr: 1, color: 'success.main' }} />
          Cena {formatCurrency(localParams.priceFrom)} -
{formatCurrency(localParams.priceTo)}
        </Typography>
        {errors.price && (
          <Typography variant="caption" color="error">
            {errors.price}
          </Typography>
        )}
      </Box>
      <Grid container spacing={2} alignItems="center">
        <Grid item xs={12}>
          <Slider
            value={[localParams.priceFrom, localParams.priceTo]}
            onChange={handlePriceChange}
            min={0}
            max={150000}
            step={1000}
            valueLabelDisplay="auto"
            aria-labelledby="price-range-slider"
            valueLabelFormat={formatCurrency}
            disabled={loading}
          />
        </Grid>
      </Grid>
    </Paper>

    {/* Search button - make it prominent */}
    <Box sx={{ mt: 3, display: 'flex', flexDirection: 'column', gap: 1
}}>
      <Button
```

```
            variant="contained"
            fullWidth
            startIcon={<SearchIcon />}
            onClick={onSearch}
            disabled={loading || Object.keys(errors).length > 0}
            color="primary"
          >
            {loading ? "Meklē..." : "Meklēt"}
          </Button>
        </Box>
      </Box>
    );
  };
```

7.pielikums. Asinhronā automašīnu sludinājumu apstrādes sistēma

Šī funkcija ļauj sistēmai apstrādāt simtiem sludinājumu vienlaikus, nevis pa vienam, kas paātrina scraping procesu no stundām uz minūtēm. Tā darbojas kā komanda strādnieku, kas katrs strādā pie sava uzdevuma paralēli.

```
ss_scraper.py:
async def process_listings_batch_async(self, listings):
    """Process a bunch of listings at the same time"""
    async with aiohttp.ClientSession() as session:
        tasks = []
        for listing in listings:
            tasks.append(self.process_listing_async(listing, session))

        results = await asyncio.gather(*tasks, return_exceptions=True)

        # Count what happened with each listing - statistics ir svarīgas
        result_counts = {
            "new": 0,
            "updated": 0,
            "unchanged": 0,
            "error": 0
        }

        for result in results:
            if isinstance(result, Exception):
                result_counts["error"] += 1
            else:
                result_counts[result] = result_counts.get(result, 0) + 1

        return result_counts

async def process_listing_async(self, listing_basic, session):
    try:
        # Get all the details from the listing page - šeit notiek actual
   scraping
```

```
            listing_details = await
        self.get_listing_details_async(listing_basic, session)

            # Skip listings marked for skipping - maiņas sludinājumi utt
            if listing_details.get('skip_listing'):
                logger.info(f"Skipping listing as marked:
        {listing_details.get('external_id', 'unknown')}")
                return "skipped"

            # Save to database - šeit nonāk clean data
            result = self.save_car_and_listing(listing_details)

            self.total_listings += 1

            # Add a small delay to be nice to the server - nevajag DOS
        attack
            await self._async_random_delay()

            return result
         except Exception as e:
            logger.error(f"Error processing listing: {str(e)}")
            self.error_count += 1
            return "error"
```

8.pielikums. Automašīnu salīdzināšanas tabulas ar labāko vērtību identificēšanu

Komponente, kas ļauj lietotājiem salīdzināt līdz 3 automašīnas blakus un automātiski iezīmē labākās vērtības katrā kategoriā. Tā kā gudrs pārdevējs, kas uzreiz parāda, kurš auto ir lētākais, jaunākais vai ar mazāko nobraukumu.

CarComparisonTable.jsx:

```
// Get comparison rows - separated by those with icons and those without
const getComparisonRows = () => {
 // Attributes with icons (prioritized) - svarīgākie parametri ar ikonām
  const attributesWithIcons = [
    {
      key: 'year',
      label: 'Izlaiduma gads',
      icon: <CalendarMonthIcon color="primary" fontSize="small" />
    },
    {
      key: 'engine_type',
      label: 'Degvielas tips',
      format: (value) => formatFuelType(value),
      icon: <LocalGasStationIcon color="primary" fontSize="small" />
    },
    {
      key: 'mileage',
      label: 'Nobraukums',
      format: (value) => value ? `${value.toLocaleString()} km` : 'Nav
    norādīts',
```

```
    icon: <SpeedIcon color="primary" fontSize="small" />
  }
];

return allAttributes.map(attr => {
  const bestValueIndex = getBestValue(attr.key); // Atrodam labāko
 vērtību

  return (
    <TableRow key={attr.key}>
      <TableCell component="th" scope="row" sx={{ display: 'flex',
 alignItems: 'center' }}>
        {attr.icon && <Box sx={{ mr: 1 }}>{attr.icon}</Box>}
        {attr.label}
      </TableCell>

      {cars.map((car, index) => {
        const value = car[attr.key];
        const formattedValue = attr.format ? attr.format(value) :
 (value || 'Nav norādīts');
        const isBest = index === bestValueIndex && cars.length > 1;

        return (
          <TableCell key={index} align="center">
            <Box sx={{ position: 'relative' }}>
              <Typography
                variant="body2"
                component="span"
                fontWeight={isBest ? 'bold' : 'normal'}
                color={isBest ? 'primary.main' : 'inherit'}
              >
                {formattedValue}
              </Typography>

              {/* Zaļa ķeksiņa ikona labākajai vērtībai */}
              {isBest && (
                <Tooltip title="Labākā vērtība">
                  <CheckCircleIcon
                    color="success"
                    fontSize="small"
                    sx={{
                      position: 'absolute',
                      top: -8,
                      right: -12,
                      width: 16,
                      height: 16
                    }}
                  />
                </Tooltip>
```

```
                )}
              </Box>
            </TableCell>
          );
        })}
      </TableRow>
    );
  });
};

// Find the best value in a row - šeit notiek "gudra" salīdzināšana
const getBestValue = (attribute) => {
  if (cars.length <= 1) return null;

  let bestIndex = 0;
  let bestValue = cars[0][attribute];
  let isNumeric = !isNaN(parseFloat(bestValue)) && isFinite(bestValue);

  // Determine if a higher or lower value is better - logic ir svarīgs
  const isHigherBetter = attribute === 'year'; // Jaunāks gads = labāks

  for (let i = 1; i < cars.length; i++) {
    const currentValue = cars[i][attribute];

    if (currentValue === undefined || currentValue === null) continue;

    if (isNumeric) {
      if (isHigherBetter && currentValue > bestValue) {
        bestValue = currentValue;
        bestIndex = i;
      } else if (!isHigherBetter && currentValue < bestValue) { //
   Zemāka cena = labāka
        bestValue = currentValue;
        bestIndex = i;
      }
    }
  }

  return bestIndex;
};
```

9.pielikums. Reģionālo cenu statistiku ģenerēšana ar kompleksiem JOIN

API funkcija, kas apkopo cenu datus pa Latvijas reģioniem un aprēķina vidējās, minimālās un maksimālās cenas katram. Tas ļauj lietotājiem redzēt, kurā pilsētā automašīnas ir vislētākās vai visdārgākās.

```
Api.py:
@app.route('/api/region-stats', methods=['GET'])
def region_statistics():
```

```
    """Get car price stats by region"""
    try:
        brand = request.args.get('brand')
        model = request.args.get('model')
        year_from = request.args.get('yearFrom', type=int)
        year_to = request.args.get('yearTo', type=int)

        logger.info(f"Region stats: Brand={brand}, Model={model}")

        # Simple query to avoid complex joins - performance matters
        query = session.query(
            Region.name.label('region_name'),
            func.avg(Listing.price).label('avg_price'),
            func.min(Listing.price).label('min_price'),
            func.max(Listing.price).label('max_price'),
            func.count(Listing.listing_id).label('count')
        )

        # Join tables step by step - building complex query carefully
        query = query.join(Car, Listing.car_id == Car.car_id)
        query = query.join(Region, Car.region_id == Region.region_id)

        # Add brand/model joins only if needed - conditional complexity
        if brand or model:
            query = query.join(Model, Car.model_id == Model.model_id)
            query = query.join(Brand, Model.brand_id == Brand.brand_id)

        # Active listings only - no point showing sold cars
        query = query.filter(Listing.is_active == True)

        # Apply filters based on user input
        if brand:
            query = query.filter(func.lower(Brand.name) ==
func.lower(brand))

        if model:
            query = query.filter(func.lower(Model.name) ==
func.lower(model))

        if year_from:
            query = query.filter(Car.year >= year_from)

        if year_to:
            query = query.filter(Car.year <= year_to)

        # Group by region - šeit notiek actual aggregation
        query = query.group_by(Region.name)

        results = query.all()
```

```
        # Format response for frontend consumption
        regions_data = []
        for row in results:
            if not row.region_name:
                continue

            # Convert to integers to avoid JSON issues
            avg_price = int(row.avg_price) if row.avg_price else 0
            min_price = int(row.min_price) if row.min_price else 0
            max_price = int(row.max_price) if row.max_price else 0

            regions_data.append({
                'name': row.region_name,
                'avgPrice': avg_price,
                'minPrice': min_price,
                'maxPrice': max_price,
                'count': row.count
            })

        logger.info(f"Found stats for {len(regions_data)} regions")
        return jsonify({"regions": regions_data})

    except Exception as e:
        logger.error(f"Region stats failed: {str(e)}", exc_info=True)
        return jsonify({"regions": [], "error": str(e)}), 500
```

10.pielikums. Rezultātu tabulas ar paplašināmām rindām un batch operācijām

React tabulas komponente, kas rāda auto sarakstu ar iespēju izvērst katru rindu papildu detaļām, kā arī veikt bulk operācijas. Var teikt ka tā ir kā Excel tabula, var kārtot, filtrēt, izvērst un izvēlēties vairākus ierakstus vienlaikus, un tas pat strāda.

ResultsSection.js:

```
const ResultsSection = ({
  cars = [],
  loading = false,
  compareMode = false,
  onSelectCar = () => {},
  onToggleFavorite = () => {},
  favorites = []
}) => {
  // State for complex table functionality
  const [page, setPage] = useState(0);
  const [rowsPerPage, setRowsPerPage] = useState(10);
  const [expandedRow, setExpandedRow] = useState(null);
  const [selectedRows, setSelectedRows] = useState([]);

  // Reset pagination when cars change - UX detail that matters
  useEffect(() => {
```

```
  setPage(0);
}, [cars]);

// Check if car is in favorites - performance optimized lookup
const isFavorite = (car) => {
  if (!car || !favorites) return false;
  return favorites.some(fav => fav && car && fav.id === car.id);
};

// Toggle row expansion for details - users love expanding rows
const toggleRowExpansion = (id) => {
  setExpandedRow(expandedRow === id ? null : id);
};

// Bulk select functionality - power users need this
const handleSelectAll = (checked) => {
  if (checked) {
    const currentPageCars = sortedCars.slice(page * rowsPerPage, page
 * rowsPerPage + rowsPerPage);
    setSelectedRows(currentPageCars.map(car => car.id));
  } else {
    setSelectedRows([]);
  }
};

return (
  <Box>
    <TableContainer component={Paper}>
      <Table size="small" aria-label="automašīnu saraksta tabula">
        <TableHead>
          <TableRow>
            {compareMode && (
              <TableCell padding="checkbox">
                <Checkbox
                  indeterminate={selectedRows.length > 0 &&
 selectedRows.length < cars.length}
                  checked={cars.length > 0 && selectedRows.length ===
 cars.length}
                  onChange={(e) => handleSelectAll(e.target.checked)}
                />
              </TableCell>
            )}
            <TableCell>Marka un Modelis</TableCell>
            <TableCell>Gads</TableCell>
            <TableCell>Cena</TableCell>
            <TableCell>Darbības</TableCell>
          </TableRow>
        </TableHead>
```

```
        <TableBody>
          {sortedCars
            .slice(page * rowsPerPage, page * rowsPerPage +
rowsPerPage)
            .map((car) => {
              const isItemSelected = selectedRows.includes(car.id);
              const isExpanded = expandedRow === car.id;

              return (
                <React.Fragment key={car.id}>
                  {/* Main row */}
                  <TableRow
                    hover
                    onClick={() => compareMode ? onSelectCar(car) :
toggleRowExpansion(car.id)}
                    selected={isItemSelected}
                    sx={{ cursor: 'pointer' }}
                  >
                    {compareMode && (
                      <TableCell padding="checkbox">
                        <Checkbox
                          checked={isItemSelected}
                          onChange={() => onSelectCar(car)}
                        />
                      </TableCell>
                    )}

                    <TableCell>
                      <Box sx={{ display: 'flex', alignItems: 'center'
}}>
                        <Typography variant="body2"
fontWeight="medium">
                          {car.brand} {car.model}
                        </Typography>
                        {isFavorite(car) && (
                          <FavoriteIcon color="error" fontSize="small"
sx={{ ml: 1 }} />
                        )}
                      </Box>
                    </TableCell>

                    <TableCell>{car.year}</TableCell>
                    <TableCell>
                      <Typography fontWeight="bold">
                        €{car.price?.toLocaleString()}
                      </Typography>
                    </TableCell>

                    <TableCell>
```

```
                    <IconButton onClick={() =>
onToggleFavorite(car)}>
                      {isFavorite(car) ? <FavoriteIcon /> :
<FavoriteBorderIcon />}
                    </IconButton>
                  </TableCell>
                </TableRow>

                {/* Expanded details row - šeit ir hidden magic */}
                <TableRow>
                  <TableCell colSpan={compareMode ? 5 : 4} sx={{ py:
0 }}>
                    <Collapse in={isExpanded} timeout="auto"
unmountOnExit>
                      <Box sx={{ py: 2, px: 1 }}>
                        <Grid container spacing={2}>
                          <Grid item xs={12} sm={6}>
                            <Typography variant="body2">
                              Dzinējs: {car.engine || 'Nav
norādīts'}
                            </Typography>
                            <Typography variant="body2">
                              Nobraukums: {car.mileage ?
`${car.mileage.toLocaleString()} km` : 'Nav norādīts'}
                            </Typography>
                          </Grid>
                          <Grid item xs={12} sm={6}>
                            <Typography variant="body2">
                              Reģions: {car.region || 'Nav
norādīts'}
                            </Typography>
                            <Typography variant="body2">
                              Krāsa: {car.color || 'Nav norādīts'}
                            </Typography>
                          </Grid>
                        </Grid>
                      </Box>
                    </Collapse>
                  </TableCell>
                </TableRow>
              </React.Fragment>
            );
          })}
        </TableBody>
      </Table>
    </TableContainer>

    {/* Pagination component - essential priekš large datasets */}
    <TablePagination
```

```
          component="div"
          count={sortedCars.length}
          rowsPerPage={rowsPerPage}
          page={page}
          onPageChange={handleChangePage}
          onRowsPerPageChange={handleChangeRowsPerPage}
          labelRowsPerPage="Rindas lapā:"
        />
      </Box>
    );
  };
```

11.pielikums. Automašīnu detaļu dialoga ar pilnu informāciju un attēlu

Modal dialogs ar detalizētu automašīnas informāciju un asinhronās datu ielādes funkcionalitāti par konkrētu auto, kas nav parādīts meklēšanas sarakstā.

```
CarDetailsDialog.jsx
const CarDetailsDialog = ({
  open,
  car,
  onClose,
  onAddToCompare,
  onToggleFavorite,
  isFavorite
}) => {
  // State for dialog content - managing complex data loading
  const [details, setDetails] = useState(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);
  const [imageErrors, setImageErrors] = useState({});
  const [expanded, setExpanded] = useState({
    basic: true,
    technical: false,
    equipment: false,
    description: false
  });

  // Load car details when dialog opens - fetch additional data
  useEffect(() => {
    const fetchDetails = async () => {
      setLoading(true);
      setError(null);

      try {
        console.log('Getting details for:', car.listing_url);
        const response = await getListingDetails(car.listing_url);

        if (response.error) {
          throw new Error(response.error);
        }
```

```
      // Merge with existing car data - combine local and remote data
      const fullDetails = {
        ...car,
        description: response.description,
        equipment: response.equipment || [],
        image_url: response.image_url,
        region: response.region || car.region,
        year: response.year_detail || car.year,
        engine: response.engine_detail || car.engine
      };

      console.log('Details loaded:', fullDetails);
      setDetails(fullDetails);

    } catch (err) {
      console.error('Failed to get details:', err);
      setError('Neizdevās ielādēt pilnu informāciju. Rāda pamata
 datus.');
      setDetails(car); // Fallback to basic car data
    } finally {
      setLoading(false);
    }
  };

  if (open && car) {
    setDetails(null);
    setLoading(false);
    setError(null);
    setImageErrors({});

    if (car.listing_url) {
      fetchDetails();
    } else {
      setDetails(car);
    }
  }
}, [open, car]);

// Handle image loading errors - graceful fallback
const handleImageError = (imageIndex) => {
  setImageErrors(prev => ({
    ...prev,
    [imageIndex]: true
  }));
};

// Toggle accordion sections - user controls what they see
const toggleSection = (section) => {
```

```
  setExpanded(prev => ({
    ...prev,
    [section]: !prev[section]
  }));
};

const currentDetails = details || car;

// Helper functions for formatting - clean display
const formatValue = (value, defaultText = 'Nav norādīts') => {
  if (value === null || value === undefined || value === '') {
    return defaultText;
  }
  return value;
};

const formatPrice = (value) => {
  return value ? `€${Number(value).toLocaleString()}` : 'Nav
 norādīts';
};

return (
  <Dialog
    open={open}
    onClose={onClose}
    maxWidth="md"
    fullWidth
    scroll="paper"
  >
    <DialogTitle sx={{ m: 0, p: 2, pr: 6 }}>
      <Box sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>
        <DirectionsCarIcon color="primary" />
        <Typography variant="h6">
          {car.brand} {car.model}
        </Typography>
        <Chip label={car.year} size="small" color="primary"
 variant="outlined" />
      </Box>

      <IconButton onClick={onClose} sx={{ position: 'absolute', right:
 8, top: 8 }}>
        <CloseIcon />
      </IconButton>
    </DialogTitle>

    <DialogContent dividers sx={{ p: 0 }}>
      {loading ? (
        <Box sx={{ display: 'flex', justifyContent: 'center', py: 4
 }}>
```

```
          <CircularProgress />
        </Box>
      ) : (
        <Box>
          {/* Car image section */}
          {currentDetails.image_url && (
            <Box sx={{ position: 'relative', mb: 2 }}>
              <Box
                component="img"
                src={currentDetails.image_url}
                alt={`${car.brand} ${car.model}`}
                onError={() => handleImageError(0)}
                sx={{
                  width: '100%',
                  maxHeight: 400,
                  objectFit: 'contain',
                  display: imageErrors[0] ? 'none' : 'block'
                }}
              />
              {imageErrors[0] && (
                <Box sx={{
                  width: '100%',
                  height: 300,
                  display: 'flex',
                  alignItems: 'center',
                  justifyContent: 'center',
                  bgcolor: 'grey.100',
                  border: '1px dashed',
                  borderColor: 'grey.400'
                }}>
                  <Typography color="text.secondary">
                    Attēlu nevar ielādēt
                  </Typography>
                </Box>
              )}
            </Box>
          )}

          {/* Price and action buttons */}
          <Paper elevation={1} sx={{ p: 2, m: 2, bgcolor:
'background.default' }}>
            <Grid container alignItems="center" justifyContent="space-
between">
              <Grid item>
                <Typography variant="h5" color="primary"
fontWeight="bold">
                  {formatPrice(currentDetails.price)}
                </Typography>
              </Grid>
```

```
                <Grid item>
                  <Box sx={{ display: 'flex', gap: 1 }}>
                    <IconButton onClick={() => onToggleFavorite(car)}
   color={isFavorite ? "error" : "default"}>
                      {isFavorite ? <FavoriteIcon /> :
   <FavoriteBorderIcon />}
                    </IconButton>
                    <IconButton onClick={() => onAddToCompare(car)}>
                      <CompareArrowsIcon />
                    </IconButton>
                  </Box>
                </Grid>
              </Grid>
            </Paper>
          </Box>
        )}
      </DialogContent>
    </Dialog>
  );
};
```

12.pielikums. Lietotāju izlases pārvaldības sistēma ar backend API integrāciju

Funkcionalitāte, kas ļauj lietotājiem saglabāt automašīnas savā izlasē un sinhronizēt tās ar serveri caur autentificētu API. Tas demonstrē kā savienot React state ar backend datubāzi un nodrošināt ‘seamless user experience’.

App.js:

```
const handleToggleFavorite = async (car) => {
  // Check if user is authenticated first - security check
  if (!isAuthenticated) {
    setNotification({
      open: true,
      message: 'Lūdzu, pieslēdzieties, lai izmantotu izlasi',
      severity: 'warning'
    });
    setCurrentPage('login');
    return; // Exit early if not authenticated
  }

  const isFavorite = favorites.some(fav => fav.id === car.id);

  // User is authenticated, use the backend services
  try {
    if (isFavorite) {
      await removeFavorite(car.id); // API call
      setFavorites(prev => prev.filter(fav => fav.id !== car.id));

      setNotification({
        open: true,
```

```
        message: `${car.brand} ${car.model} noņemts no izlases`,
        severity: 'info'
      });
    } else {
      await addFavorite(car); // API call with full car object
      setFavorites(prev => [...prev, car]);

      setNotification({
        open: true,
        message: `${car.brand} ${car.model} pievienots izlasei`,
        severity: 'success'
      });
    }
  } catch (error) {
    console.error('Error updating favorites:', error);

    setNotification({
      open: true,
      message: 'Neizdevās atjaunināt izlasi',
      severity: 'error'
    });

    // If unauthorized, prompt to login - handle expired tokens
    if (error.message && (error.message.includes('token') ||
   error.message.includes('authentication'))) {
      setNotification({
        open: true,
        message: 'Lūdzu, pieslēdzieties, lai izmantotu izlasi',
        severity: 'warning'
      });

      setCurrentPage('login');
    }
  }
};

// Backend API functions in authService.js
export const addFavorite = async (car) => {
  try {
    const token = getToken();

    if (!token) {
      throw new Error('No authentication token found');
    }

    const response = await fetch(`${API_BASE_URL}/user/favorites`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
```

```
          'Authorization': `Bearer ${token}`
        },
        body: JSON.stringify({ car }) // Send full car object
      });

      const data = await response.json();

      if (!response.ok) {
        throw new Error(data.error || 'Failed to add favorite');
      }

      return data;
    } catch (error) {
      console.error('Add favorite error:', error);
      throw error;
    }
  };

  export const removeFavorite = async (carId) => {
    try {
      const token = getToken();

      if (!token) {
        throw new Error('No authentication token');
      }

      const response = await
     fetch(`${API_BASE_URL}/user/favorites/${carId}`, {
        method: 'DELETE',
        headers: {
          'Authorization': `Bearer ${token}`,
        },
      });

      const data = await response.json();

      if (!response.ok) {
        throw new Error(data.error || 'Failed to remove favorite');
      }

      return data;
    } catch (error) {
      console.error('Remove favorite error:', error);
      throw error;
    }
  };
```

13.pielikums. Automašīnu salīdzināšanas loģika ar localStorage backup

State management sistēma, kas ļauj lietotājiem salīdzināt līdz 3 automašīnas, ar automātisku saglabāšanu localStorage un validation loģiku.

App.js:

```
const handleAddToCompare = (car) => {
  const isAlreadyAdded = carsToCompare.some(c => c.id === car.id);

  if (isAlreadyAdded) {
    // Remove if already added - toggle behavior
    setCarsToCompare(prev => prev.filter(c => c.id !== car.id));

    setNotification({
      open: true,
      message: `${car.brand} ${car.model} noņemts no salīdzinājuma`,
      severity: 'info'
    });
  } else {
    // Add if not yet added
    if (carsToCompare.length >= 3) {
      setNotification({
        open: true,
        message: 'Var salīdzināt ne vairāk kā 3 automašīnas',
        severity: 'warning'
      });
      return; // Business rule enforcement
    }

    // Ensure the car has all required fields for comparison - data
   validation
    const carForComparison = {
      ...car,
      id: car.id || car.external_id || `car-${Date.now()}`,
      brand: car.brand || 'Nav norādīts',
      model: car.model || 'Nav norādīts',
      year: car.year || 'Nav norādīts',
      price: car.price || 0,
      engine: car.engine || `${car.engine_volume || ''}L
   ${car.engine_type || ''}`,
      transmission: car.transmission || 'Nav norādīts',
      mileage: car.mileage || 0,
      region: car.region || 'Nav norādīts'
    };

    setCarsToCompare(prev => [...prev, carForComparison]);

    setNotification({
      open: true,
      message: `${car.brand} ${car.model} pievienots salīdzinājumam`,
      severity: 'success'
```

```
    });

    // Switch to comparison tab automatically - UX enhancement
    if (tabValue !== 2) {
      setTabValue(2);
    }
  }
};

// Save comparison cars to localStorage when they change - persistence
useEffect(() => {
  localStorage.setItem('carsToCompare', JSON.stringify(carsToCompare));
}, [carsToCompare]);

// Load saved comparison cars from localStorage on mount
useEffect(() => {
  const savedComparison = localStorage.getItem('carsToCompare');
  if (savedComparison) {
    try {
      setCarsToCompare(JSON.parse(savedComparison));
    } catch (err) {
      console.error('Failed to load comparison:', err);
      // Graceful fallback - just start with empty array
      setCarsToCompare([]);
    }
  }
}, []);

// Export comparison data as JSON - user requested feature
const handleExportComparison = () => {
  if (carsToCompare.length === 0) return;

  try {
    const exportData = {
      date: new Date().toISOString(),
      comparison: carsToCompare.map(car => ({
        brand: car.brand,
        model: car.model,
        year: car.year,
        price: car.price,
        engine_type: car.engine_type,
        engine_volume: car.engine_volume,
        transmission: car.transmission,
        mileage: car.mileage,
        body_type: car.body_type,
        color: car.color,
        region: car.region,
        listing_url: car.listing_url || car.url
      }))
```

```
    };

    // Create downloadable file - standard web pattern
    const jsonStr = JSON.stringify(exportData, null, 2);
    const blob = new Blob([jsonStr], { type: 'application/json' });
    const url = URL.createObjectURL(blob);

    const link = document.createElement('a');
    link.href = url;
    link.download = `car_comparison_${new
   Date().toISOString().split('T')[0]}.json`;
    document.body.appendChild(link);
    link.click();

    setTimeout(() => {
      document.body.removeChild(link);
      URL.revokeObjectURL(url); // Cleanup memory
    }, 100);

    setNotification({
      open: true,
      message: 'Salīdzinājums eksportēts',
      severity: 'success'
    });
  } catch (err) {
    console.error('Error exporting comparison:', err);
    setNotification({
      open: true,
      message: 'Neizdevās eksportēt salīdzinājumu',
      severity: 'error'
    });
  }
};
```